

写真判定に
挑戦してみよう

LEARNING

人間ならば、簡単に判別できるものが、コンピューターには判別が難しいというものたくさんあります。例えば、レモンとイチゴを判定しようとしても、コンピューターだと簡単にはいきません。しかし、機械学習を使えば判定できるので挑戦してみましょう。

▶ 利用するデータセットについて

今回は、筆者が公開しているGitHubのデータを利用して、データセットを作成しましょう。

- [URL] https://github.com/kujirahand/fruits_db/

このGitHubのデータは、オンラインの写真共有サービス「Flickr（フリッカー）」から「Creative Commons」の「営利利用可能」ライセンスになっている写真を利用しています。「lemon」（レモン）と「strawberry」（いちご）の画像をダウンロードして分類を行い、データセットにしています。

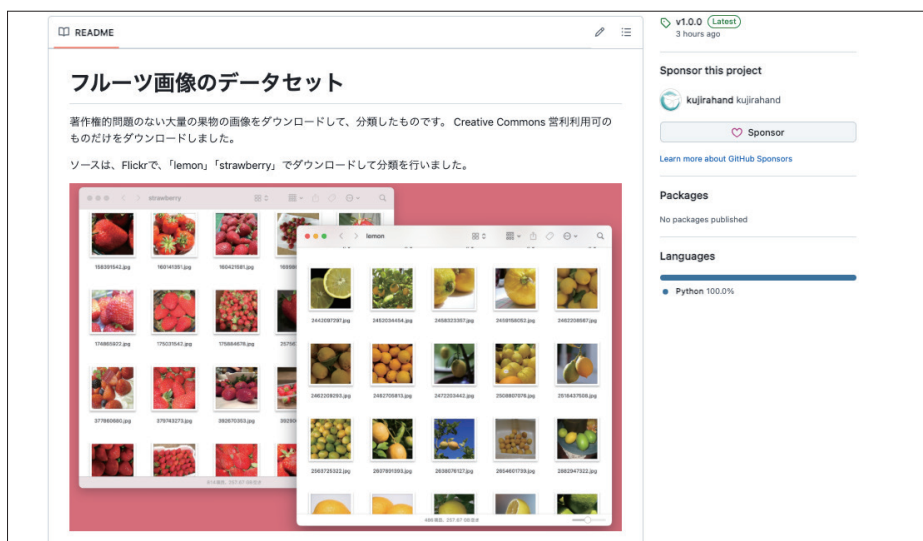


図7-5-1：今回利用するGitHubのデータセット



プログラムを実行して写真をダウンロードしよう

では、前述のGitHubから写真をダウンロードするプログラムを実行してみましょう。

Jupyter Notebookで以下のプログラムを実行してみてください。

画像ファイルがダウンロードされる様子が表示されます。

Jupyter Notebookのセルに入力

```
# 写真データをダウンロードしよう
import os
import urllib.request
import zipfile

# ダウンロード先の URL と保存ファイル名を指定
url = "https://github.com/kujirahand/fruits_db/archive/refs/tags/1.0.0.zip"
zip_path = "fruits_db.zip"
extract_dir = "./"

# ZIP ファイルをダウンロード
print("Downloading...", url)
urllib.request.urlretrieve(url, zip_path)
print("Downloaded:", zip_path)

# 解凍先ディレクトリを作成
os.makedirs(extract_dir, exist_ok=True)

# ZIP ファイルを解凍
with zipfile.ZipFile(zip_path, "r") as zf:
    zf.extractall(extract_dir)

print("Extracted to:", extract_dir)

# 分かりやすいフォルダ名に変更
os.rename("./fruits_db-1.0.0", "./fruits_db")
```

実行すると、ZIPファイルをダウンロードして、1枚ずつ順に画像が展開されます。

Zipの解凍までプログラムで実行されるよ!



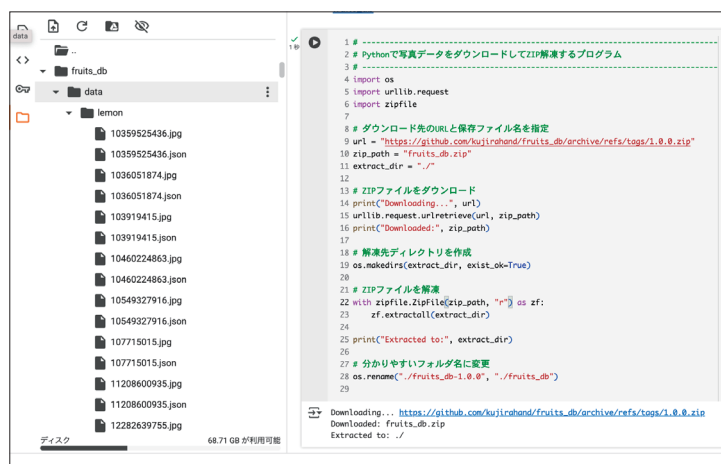


図7-5-2：プログラムを実行するとZIPファイルを取得し画像ファイルが展開されます

プログラムによって、レモンの写真が約240枚、イチゴの写真が約407枚、ダウンロードされます。

今回はJupyter Notebookで実行しているので、画面上部の「View > File Browser」をクリックすると、「fruits_db」→「data」フォルダの下に、「lemon」と「strawberry」のフォルダが作成されており、その中にデータが保存されているのが確認できます。

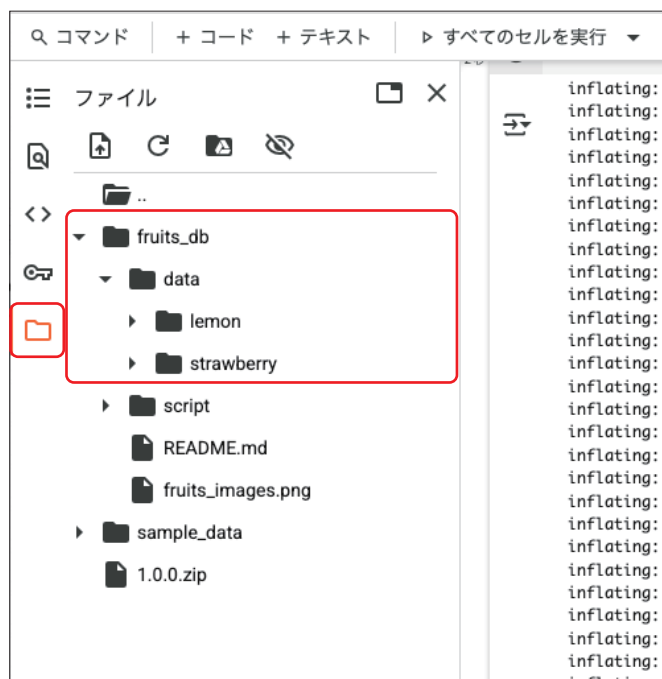


図7-5-3：ダウンロードしたデータ

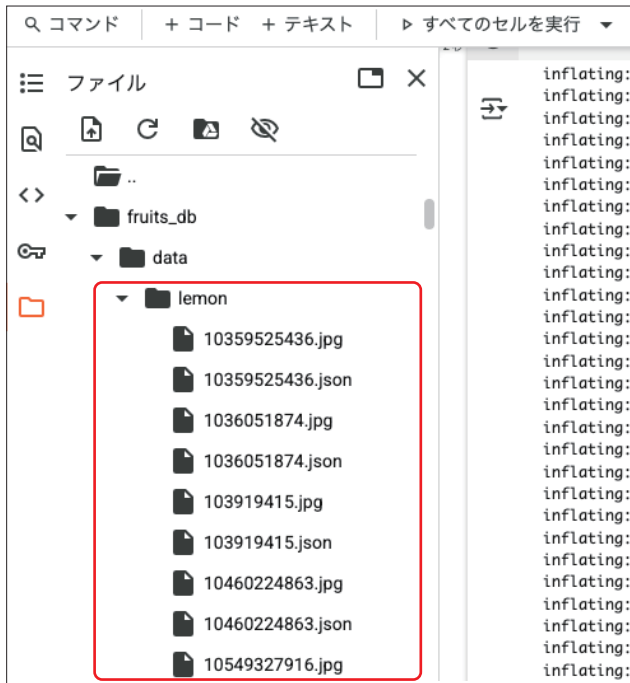


図7-5-4 : 「lemon」フォルダの中身

▶ 目視して不要な写真を削除しよう

ところで、大量に画像をダウンロードすると、中には、レモンやイチゴと関係のない写真が混ざっていることが多くあります。自分の目で確認して、明らかに間違っている画像や余分なオブジェが写っている写真を削除しましょう。この作業がとても重要で、間違っている画像が残っていると、機械学習の精度が落ちてしまいます。しっかりと、画像を確認しましょう。

今回は、著者が用意したデータですので、レモンやイチゴと関係のない写真は混ざっていません。「.json」形式のファイルが入っていますが、これは写真の著作権情報についてのファイルです。このあと機械学習で学習する場合には、「.jpg」ファイルだけを学習対象にするので、「.json」ファイルの削除は不要です。

▶ データを学習させよう

それでは、いよいよデータを学習させましょう。

まずは、ダウンロードした「fruits_db」のデータの内容を表示させてみましょう。

Jupyter Notebookのセルに入力

```

01 DIR_DATA = "./fruits_db/data"
02
03 # レモンとイチゴを分類する分類器を作ります
04 LABEL_TYPES = ["lemon", "strawberry"]
05
06 # fruits_db の中身を確認してみよう!!
07 import matplotlib.pyplot as plt
08 import matplotlib.image as mpimg
09 import os
10 import glob
11
12 for ramen_type in LABEL_TYPES:
13     files = glob.glob(f"{DIR_DATA}/{ramen_type}/*.jpg")
14     num_images_to_display = min(5, len(files))
15     plt.figure(figsize=(15, 5))
16     for i in range(num_images_to_display):
17         img = mpimg.imread(files[i])
18         plt.subplot(1, num_images_to_display, i + 1)
19         plt.imshow(img)
20         plt.axis('off')
21
22 plt.show()

```

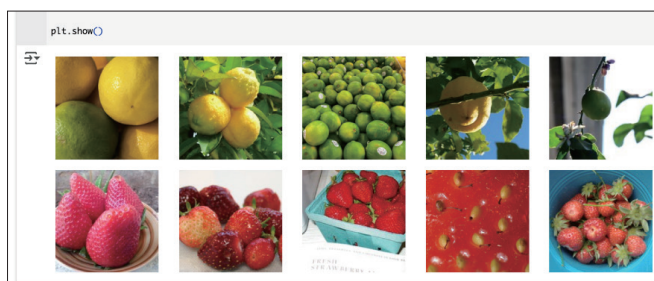


図7-5-5：プログラムを実行したところ

▶ 画像について

前述の手順でダウンロードした画像は、150×150ピクセルとなっています。ずいぶん小さいですが、これでも十分です。

というのも、大量の画像を判定させるため、大きい画像を判定しようと思うと、メモリ不足になってしまうからです。思い起こせば、**STEP 04**で手書き数字データセットを判定したときも、8×8ピクセルと小さな画像で十分な精度が出ていました。

そこで、今回の画像データでも、最初に16×16ピクセルで試してみましよう。ただし、STEP 04では、手書き数字はグレースケールなので1色の情報で良かったのですが、今回はカラー写真なので、光の三原色を表す赤・緑・青と3色のデータを扱う必要があります。そのため、16×16ピクセルでも3色分で、 $16 \times 16 \times 3 = 768$ 個の情報を学習させることになります。カラーにするだけで一気に情報は3倍になるのです。

以下のプログラムは、指定ディレクトリに以下にある画像を読み込み、一次元の配列に変換し、変数dataとラベル情報を表す変数targetに追加するものです。Jupyter Notebookで実行してみてください。

Jupyter Notebookのセルに入力

```
01 from PIL import Image
02 import numpy as np
03 import os
04 import glob
05
06 # 画像サイズの指定
07 IMAGE_SIZE = 16
08
09 # 画像を読み込んでデータとラベルに追加する
10 data = [] # 画像を入れるリストを空で作成
11 target = [] # ラベル情報を入れるリストを空で作成
12
13 def glob_images(dir, label, size=IMAGE_SIZE):
14     data_dir = os.path.join(DIR_DATA, dir)
15     files = glob.glob(data_dir + "/*.jpg")
16     for f in files:
17         img = Image.open(f) # 画像を開く
18         img = img.convert("RGB") # 念のため RGB 画質に変換
19         img = img.resize((size, size), Image.LANCZOS) # サイズを変換方法を指定してリサイズ
20         ary = np.asarray(img, dtype=np.float32) / 255.0 # 正規化
21         ary = ary.reshape(-1,) # 1次元の配列にする
22         data.append(ary) # データに追加
23         target.append(label) # ラベルに追加
24
25 # 画像ディレクトリとラベル、画像サイズを指定してデータを追加
26 for label, dir in enumerate(LABEL_TYPES):
27     glob_images(dir, label)
```


それでは、このデータセットを学習させて、評価してみましょう。新規セルを作成し、以下のプログラムを記述して、実行してみましょう。

Jupyter Notebookのセルに入力

```
01 # 学習用とテスト用に分割 --- (*1)
02 from sklearn.model_selection import train_test_split
03
04 # リストを numpy 配列に変換
05 X = np.array(data)
06 y = np.array(target)
07
08 # 学習用とテスト用に分割 (25% をテスト用に)
09 X_train, X_test, y_train, y_test = train_test_split(
10     X, y, test_size=0.25, random_state=123
11 )
12
13 print(" 学習用の画像 =", len(y_train), "/ テスト用の画像 =", len(y_test))
14
15 # データを学習 --- (*2)
16 from sklearn import svm
17 clf = svm.SVC()
18 clf.fit(X_train, y_train)
19
20 # モデルを評価 --- (*3)
21 pred = clf.predict(X_test)
22 result = list(pred == y_test).count(True) / len(y_test)
23 print(" 正解率 =" + str(result))
```

そうすると、以下のような値が表示されます。最初からなかなかの精度が出ました。

```
学習用の画像 = 487 / テスト用の画像 = 163
正解率=0.950920245398773
```

もちろん、これは筆者の環境で実行した場合であって、実際には、これに近い値が表示されることでしょう。しかし、これよりも、ずっと低い値が表示された場合、モデルの学習に使った画像に問題があります。

もし、正解率が低い場合は、もう一度、画像フォルダの中を確認し、関係のない写真が入っていないかどうかを確認してみてください。

今回使ったデータでは、関係のない写真は入っていないけど、違うデータを使う場合には画像データの精査を行ってみてね!



それではプログラムを見てみましょう。このプログラムは、前節で紹介したものとはほとんど同じです。(*1)でデータを学習用とテスト用に分割し、(*2)でSVMライブラリのサポートベクターマシン (SVM) アルゴリズムでデータを学習します。

そして、(*3)でテストデータを使って分類を予測し、それがどの程度合っているのか正解率を求めるという処理になっています。

▶ チューニングしてみよう

▶ LinearSVC アルゴリズムを使おう

先ほどのプログラムでも、まずまずの成果が出たことでしょう。しかし、機械学習のアルゴリズムを変更すると、さらに精度を高めることができるかもしれません。

そこで、LinearSVC というアルゴリズムを使ってみましょう。

Jupyter Notebookのセルに入力 - サンプルプログラム : src/ch7/get_photos6 .txt

```
01 # データを学習
02 from sklearn import svm
03 clf = svm.LinearSVC(max_iter=10000)
04 clf.fit(X_train, y_train)
05
06 # モデルを評価
07 pred = clf.predict(X_test)
08 result = list(pred == y_test).count(True) / len(y_test)
09 print(" 正解率 =" + str(result))
```

実行結果は以下のようになりました。少し正解率が上がりましたね。

正解率 =0.9570552147239264

ここで利用したLinearSVCも、サポートベクターマシン (SVM) の一種です。先ほど利用したSVCは非線形の境界データに強いものでしたが、LinearSVCは、線形最適化を反復法 (座標降下法など) で近似的に解きます。大規模データや高次元データに強く、高速に学習できるアルゴリズムです。

▶ Random Forestアルゴリズムを使おう

続いて、Random Forestというアルゴリズムを試してみましょう。

Jupyter Notebookのセルに入力 - サンプルプログラム : src/ch7/get_photos7 .txt

```
01 from sklearn.ensemble import RandomForestClassifier
02 clf_rf = RandomForestClassifier(n_estimators=500, random_state=12345, n_
03 jobs=-1)
04 clf_rf.fit(X_train, y_train)
05
06 # モデルを評価
07 pred = clf_rf.predict(X_test)
08 result = list(pred == y_test).count(True) / len(y_test)
09 print(" 正解率=" + str(result))
```

先ほどよりも、精度が上がりました。

正解率=0.9815950920245399

Random Forestは、アンサンブル学習に属するアルゴリズムの一つです。**複数の決定木**をランダム性を持たせて構築し、その結果を統合することで分類や回帰を行います。分類では多数決、回帰では平均を取ること、単一の決定木よりも安定して高精度な予測が可能になります。

このように、学習アルゴリズムを変更するなど、いろいろ試すことで、より精度を高めることができます。

もし、どうしても精度が上がらない場合、データの選び直しから、やり直してみましょう。その際、色に注目すると、ぐっと精度が上がります。

各種アルゴリズムは、ここでは「そういうものがあるんだな」ぐらいの理解で大丈夫だよ！興味があったら調べてみてね



機械学習のまとめ

コラム

以上、ここまでの部分で、いろいろな題材を取り上げて、機械学習を実践してみました。Chapter 7の最初 (P.221) で、機械学習を実践する手順を紹介しました。最終的には、データを集めて、学習モデルを構築し、モデルを評価するという手順に沿って進めてみました。

一通りの手順を踏んでみて分かったと思いますが、実際に機械学習のプログラムを作る点で、一番大変なのは、データを収集し、機械学習のモデルに合うようにデータを整形する部分であるということが分かった

のではないのでしょうか。

本書で扱ったscikit-learnは機械学習の初心者にも使いやすく設計されていますが、ここまで見てきたように、データの分類や文字認識、画像認識など、さまざまな分野で活用できます。また、さまざまな本格的なアルゴリズムがサポートされており、実践でも十分役立つものです。身近に、機械学習を適用できそうなデータがあれば、それを活用するプログラムを作ってみると良いでしょう。



まとめ

今回は、あらかじめ用意されたデータセットではなく、GitHubから画像セットをダウンロードして、そこから画像判定を行ってみました。イチゴとレモンの写真をPythonを使って判別することができました。

【ここで学んだこと】

- 写真の判定処理
- レモンとイチゴを判別しよう