

Laboratorium: Klasyfikacja

March 10, 2022

1 Cel/Zakres

- Podział zbiorów uczących i testujących.
- Użycie klasyfikatorów.
- Określenie jakości klasyfikatora, metryki.

2 Przygotowanie danych

Pobierz [zbiór danych MNIST](#) przy pomocy wbudowanej funkcji *scikit-learn*:

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
```

Sprawdź w [dokumentacji użytej funkcji](#) jakie własności otrzymanego obiektu przechowują cechy (*features*; zwyczajowo nazywany *X*) oraz etykiet (*labels*; zwyczajowo nazywany *y*).

Obejrzyj zbiory cech oraz etykiet w pobranym zbiorze danych. Jakiego są typu? Czy zbiór jest uporządkowany według tego, jaka cyfra znajduje się na danym obrazku?

Możesz też obejrzeć przykładową „bitmapę”, na przykład w ten sposób:

```
print((np.array(mnist.data.loc[42]).reshape(28, 28) > 0).astype(int))
```

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

3 Zbiór uczący i testowy

W procesie uczenia maszynowego ważne jest, aby zbiór testowy był *reprezentatywny* względem zbioru uczącego. Zasymlujemy sytuację, w której tak nie jest, aby zobaczyć jakie mogą być tego konsekwencje.

Posortuj oba zbiory tak, aby rekordy były uporządkowane według cyfry którą reprezentują. Można to zrobić na przykład tak:

1. Dla wygody utwórz osobne DataFrame'y z cechami i etykietami (nazwij je na przykład **X** i **y**).
2. Posortuj zbiór **y** rosnąco. Zauważ, że zmieniła się kolejność elementów w serii zbiorze, ale każdy element „powędrował” wraz ze swoim indeksem (obejrzyj **y.index**).
3. Posortuj identycznie zbiór **X** wykorzystując funkcję **reindex**.

Podziel ręcznie posortowane dane na zbiory uczący i testujący w proporcjach 80-20 (pierwsze 80% – następne 20%). Możesz skorzystać z kodu poniżej:

```
X_train, X_test = X[:56000], X[56000:]
y_train, y_test = y[:56000], y[56000:]
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(56000, 784) (56000,)
(14000, 784) (14000,)
```

Sprawdź jakie klasy znajdują się w zbiorach **y_train** oraz **y_test**. Spodziewane wyniki:

```
y_train: ['0', '1', '2', '3', '4', '5', '6', '7']
y_test: ['7', '8', '9']
```

Czy ww. zbiory są poprawne? Jakie problemy mogą wynikać z takiej ich postaci?

W praktyce, zazwyczaj do takiego podziału używamy gotowych funkcji, jak np. **train_test_split** z pakietu *scikit-learn*. Sprawdź jej dokumentację i odpowiedz, czy podział identyczny jak dokonany ręcznie da się przeprowadzić przy jej pomocy.

Teraz przeprowadź przy pomocy tej samej funkcji dzielenie zbiorów tak, aby wybór próbek w obu zbiorach był losowy (cały czas zachowując proporcje 80/20). Sprawdź klasy w obu zbiorach. Oczywiście tam razem spodziewany wynik to:

```
y_train: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
y_test: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

4 Uczenie, jedna klasa

Naucz klasyfikator *Stochastic Gradient Descent* wykrywać cyfrę 0.

Policz dokładność ww. klasyfikatora na zbiorze uczącym oraz na zbiorze testującym.

Umieść wyniki jako listę i zapisz w pliku Pickle o nazwie `sgd_acc.pkl`.

1 pkt utworzenie pliku, 2 pkt prawidłowa zawartość

Policz 3-punktową walidację krzyżową dokładności (*accuracy*) modelu dla zbioru uczącego. Zapisz wynik (tablica) w pliku Pickle o nazwie `sgd_cva.pkl`.

1 pkt utworzenie pliku, 3 pkt prawidłowa zawartość

5 Uczenie, wiele klas

Użyj klasyfikatora *Stochastic Gradient Descent* do klasyfikacji wszystkich cyfr.

Utwórz macierz błędów i zapisz ją w pliku Pickle o nazwie `sgd_cmx.pkl`.

1 pkt utworzenie pliku, 5 pkt prawidłowa zawartość

6 Prześlij raport

Prześlij plik o nazwie `lab2.py` realizujący ww. ćwiczenia.

Sprawdzone będzie, czy skrypt Pythona tworzy wszystkie wymagane pliki oraz czy ich zawartość jest poprawna.