Klasyfikacja

Igor Wojnicki

March 7, 2022

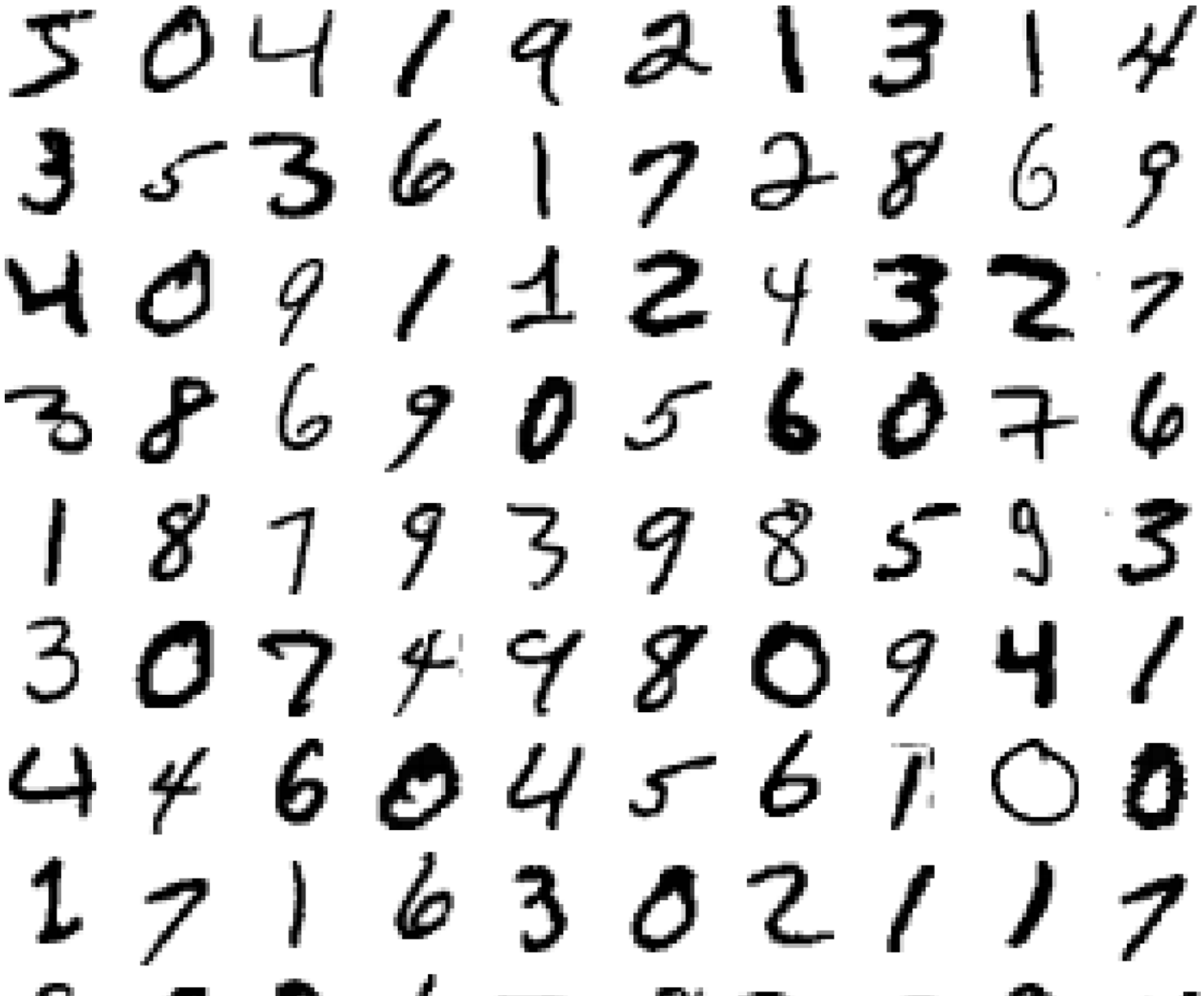
Plan prezentacji

Początek

Czyli jak dużo można zrobić nie mając pojęcia o co chodzi w uczeniu maszynowym.

Klasyfikacja liter: MNIST

Modified National Institute of Standards and Technology database



Zbiór danych

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

dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'DESCR', 'details', 'categories', 'url'])

- "data" – 1-wymiarowa tablica tablic pikseli (odcienie szarości), obrazek oryginalny: 28x28,
- "target" – reprezentacja tekstowa, tablica jednoelementowych łańcuchów znaków.

Znak

```
1  digit = mnist["data"][2].reshape(28,28)
2  print(mnist["target"][2])
3  digit = digit > 0
4  print(digit.astype(int))
5  import numpy as np
6  t = mnist["target"]
7  print(len(np.where(t == '4')[0])) # ile razy występuje 4?
```

Znak

```
4
[[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 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 1 1 1 0 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0]
 [0 0 0 0 1 1 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 1 1 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 1 1 1 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 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 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 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 1 1 1 1 1 1 1 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 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 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 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 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 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]]
6824
```

Zbiór uczący i testujący

- Pierwsze 60000 zbiór uczący, pozostałe 10000 zbiór testujący.
- Zbiory są równomierne (przemieszane).
- "target" ma etykiety tekstowe np. "5", lepiej będzie z liczbowymi.

```
1 X, y = mnist["data"], mnist["target"].astype(np.uint8)
2 print(X.shape, y.shape)
3 X_train, X_test = X[:60000], X[60000:]
4 y_train, y_test = y[:60000], y[60000:]
```

```
(70000, 784) (70000,)
```

Plan prezentacji

Dane

Klasyfikacja binarna

Klasyfikacja wieloklasowa

Klasyfikator binarny

- Stochastic Gradient Descent
- Czy wejście należy, czy nie należy do klasy?
  - Czy wejście jest cyfrą "5"?

```
1 y_train_5 = (y_train == 5)
2 y_test_5 = (y_test == 5)
3 print(y_train_5)
4 print(np.unique(y_train_5))
5 print(len(y_train_5))
```

```
[ True False False ...  True False False]
[False  True]
60000
```

Klasyfikator binarny, uczenie i klasyfikacja

- uczenie

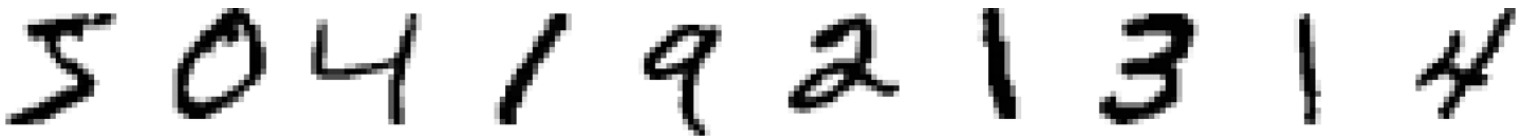
```
from sklearn.linear_model import SGDClassifier
import time
start = time.time()
sgd_clf = SGDClassifier(random_state=42) # 42?
sgd_clf.fit(X_train, y_train_5)
print(time.time() - start)
```

```
16.994534969329834
```

- klasyfikacja

```
start = time.time()
print(sgd_clf.predict([mnist["data"][0],
                       mnist["data"][1]]))
print(time.time() - start)
```

```
[ True False]
0.0004053115844726562
```



Jak dobrze działa klasyfikator?

- k-fold cross-validation (k-krotny sprawdzian krzyżowy / walidacja krzyżowa)
  - podział próby na k podzbiory,
  - przeprowadzenie oceny dla każdego ze zbiorów osobno,
    - użyj 1-szego podzbioru jako dane testowe, pozostałych jako uczące,
    - powtórz dla pozostałych
  - porównanie k wyników.

Jak dobrze działa klasyfikator? Przykład

```
1  from sklearn.model_selection import cross_val_score
2  start = time.time()
3  score = cross_val_score(sgd_clf, X_train, y_train_5,
4                          cv=3, scoring="accuracy",
5                          n_jobs=-1)
6  print(time.time() - start)
7  print(score)
```

```
9.850900888442993
[0.95035 0.96035 0.9604 ]
```

- 3 podzbiory,
- policz trafność ("accuracy"): =prawidłowe/wszystkie,
- użyj wszystkich dostępnych CPU (działa również dla procesu uczenia, zależy od modelu/algorytmu).

Uwaga: jak trafne będzie znalezienie innego znaku niż "5", gdy "5" stanowi 10% zbioru?

- trafność nie jest najlepszą miarą do oceny klasyfikatora...

Lepsza ocena: tablica pomyłek/macierz błędów, confusion matrix

```
1  from sklearn.model_selection import cross_val_predict
2  from sklearn.metrics import confusion_matrix
3  y_train_pred = cross_val_predict(sgd_clf, X_train,
4                                   y_train_5, cv=3,
5                                   n_jobs=-1)
6  print(y_train_pred)
7  print(len(y_train_pred))
8  print(confusion_matrix(y_train_5, y_train_pred))
```

```
[ True False False ...  True False False]
60000
[[53892   687]
 [ 1891  3530]]
```

nie-5 jako nie-5 *nie-5 jako 5*

5 jako nie-5 *5 jako 5*

true negative *false positive*

false negative *true positive*

Idealny klasyfikator, macierz błędów

- Idealny klasyfikator będzie miał tylko: *true negative* i *true positive*.

```
1 print(confusion_matrix(y_train_5, y_train_5))
```

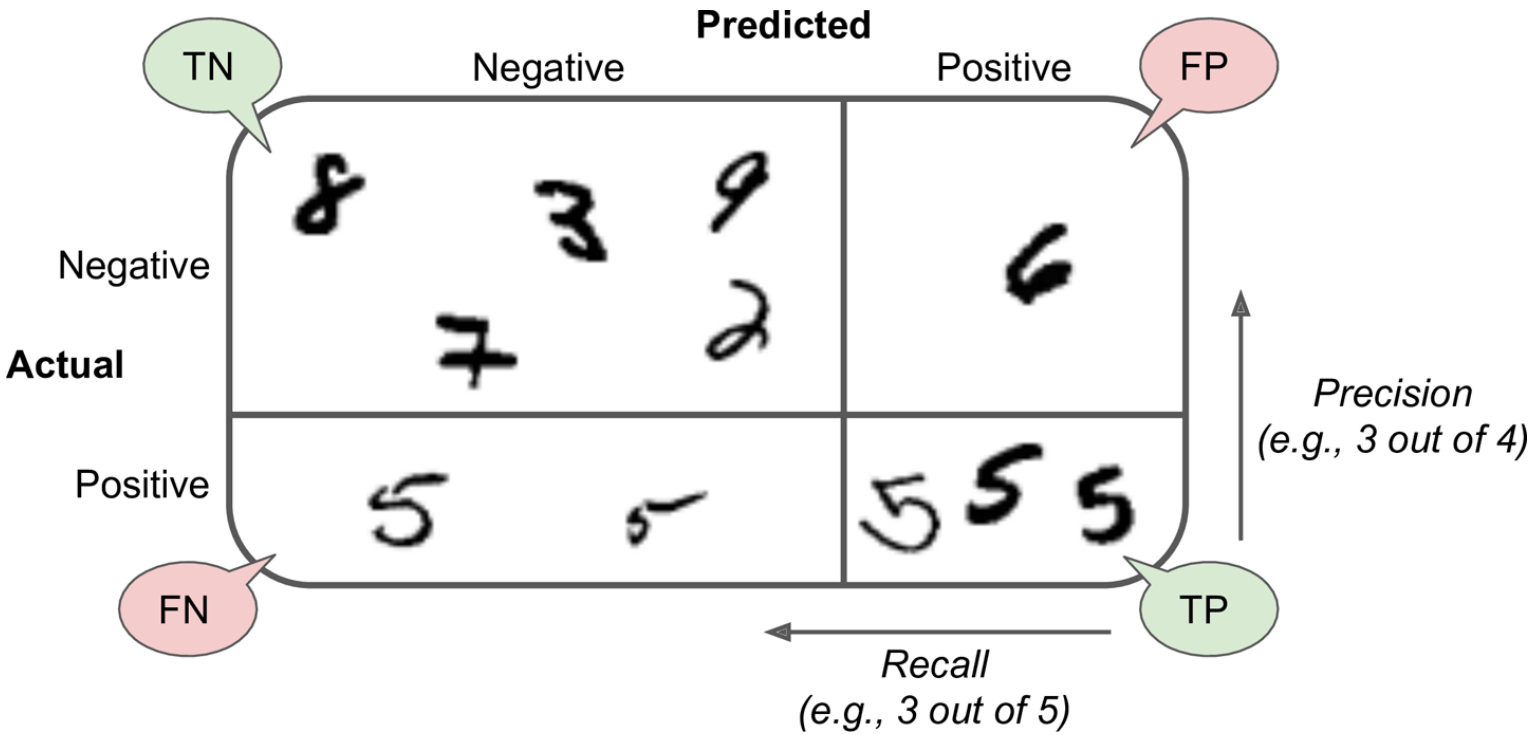
```
[[54579     0]
 [    0  5421]]
```

Precyzja (precision) i czułość (recall)

- precision: positive predictive value
- recall (sensitivity): true positive rate

$precision = \frac{true\ positive}{true\ positive + false\ postive}$ – ile zidentyfikowanych jako dodatnie jest dodatnie

$recall = \frac{true\ positive}{true\ positive + false\ negative}$ – jaka część dodatnich została wykryta



Raz jeszcze...

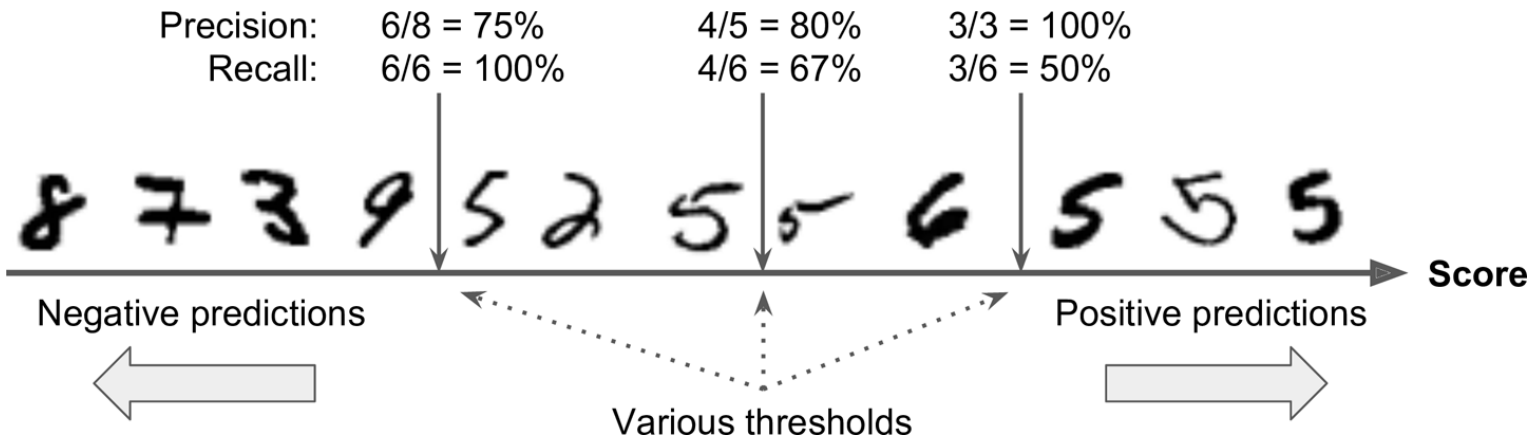
```
1 from sklearn.metrics import precision_score, recall_score
2 print(precision_score(y_train_5, y_train_pred),
3       recall_score(y_train_5, y_train_pred))
```

```
0.8370879772350012 0.651171370595831 1
```

- 5 zostało pozytywnie zidentyfikowane w 83%,
- zostało zidentyfikowanych 65%

precision vs. *recall*

- Zwiększając *precision* zmniejsza się *recall*.



F_1

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

- Jedna wartość.
- Faworyzuje klasyfikatory o podobnym *precision* i *recall*.
- Wszystko zależy od potrzeb: wysoki *precision* niski *recall*?

```
1 from sklearn.metrics import f1_score
2 print(f1_score(y_train_5, y_train_pred))
```

```
0.7325171197343846
```

Plan prezentacji

Wiele klas, strategie

- SGD, Random Forest, naive Bayes.
- Logistic Regression, SVM – binarne, ale można wykorzystać: wiele binarnych = jeden wieloklasowy.

Klasyfikacja cyfr, strategie:

- binarny klasyfikator dla każdej cyfry, ocenić, wybrać najtrafniejszy wynik: *one-versus-the-rest* (OvR), *one-versus-all*,
- binarny klasyfikator dla każdej pary cyfr: *one-versus-one* (OvO),
  - duuuuużo klasyfikatorów,
  - ale można je trenować na podzbiorze zawierającym tylko cyfry z pary.

Przykład

sklearn automatycznie wybierze odpowiednią strategię w zależności od algorytmu np. OvO a nie OvR dla Support Vector Machine.

```
1 from sklearn.svm import SVC
2 svm_clf = SVC()
3 svm_clf.fit(X_train, y_train) # wszystkie klasy z y_train
4 print(svm_clf.predict([mnist["data"][0],
5                        mnist["data"][1]]))
6 print(svm_clf.decision_function([mnist["data"][0],
7                                  mnist["data"][1]]))
8 print(svm_clf.classes_)
```

```
[5 0]
[[ 1.72501977  2.72809088  7.2510018   8.3076379  -0.31087254  9.313248
   1.70975103  2.76765202  6.23049537  4.84771048]
 [ 9.31776763  0.69966542  8.26937495  3.82063539 -0.30671293  7.271416
   3.80978873  1.72165536  6.0316466   3.83885601]]
[0 1 2 3 4 5 6 7 8 9]
```

Przykład wieloklasowy

```
1  sgd_m_clf = SGDClassifier(random_state=42,n_jobs=-1)
2  sgd_m_clf.fit(X_train, y_train)
3  print(sgd_m_clf.predict([mnist["data"][0],
4                           mnist["data"][1]]))
```

```
[3 0]
```

Analiza błędów, macierz błędów

```
print(cross_val_score(sgd_m_clf, X_train, y_train,
                      cv=3, scoring="accuracy", n_jobs=-1))
y_train_pred = cross_val_predict(sgd_m_clf, X_train,
                                 y_train, cv=3, n_jobs=-1)
```

```
[0.87365 0.85835 0.8689 ]
```

```
conf_mx = confusion_matrix(y_train, y_train_pred)
print(conf_mx)
```

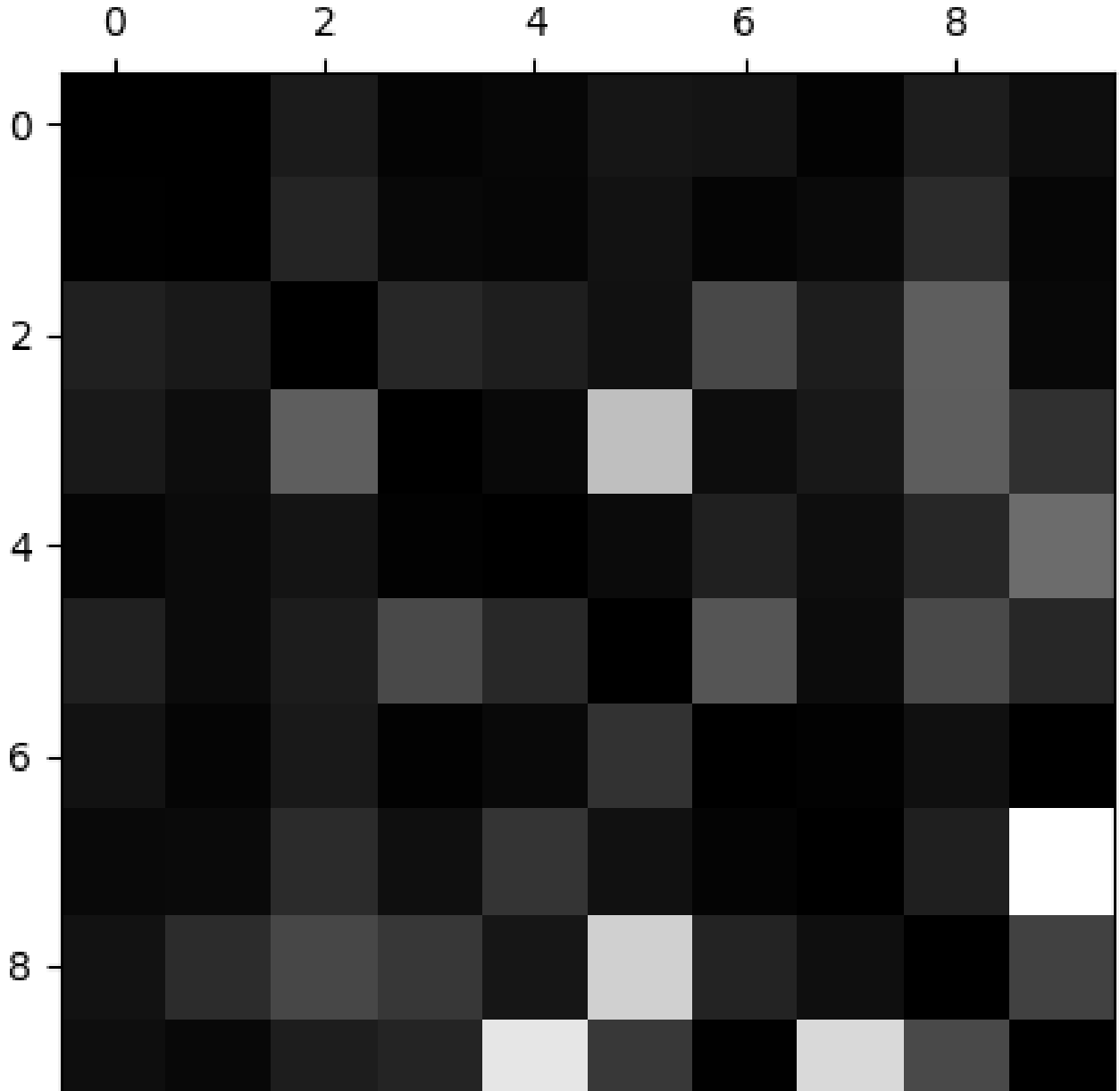
```
[[5635    0   61   10   16   50   46    7   66   32]
 [   3 6393   95   21   16   47   15   27  109   16]
 [  72   56 5174   89   69   39  163   66  212   18]
 [  58   32  217 4941   23  441   32   56  216  115]
 [  11   26   46    6 5298   26   73   32   87  237]
 [  68   23   58  150   83 4606  174   26  152   81]
 [  40   13   56    6   22  113 5625    5   36    2]
 [  23   24  103   36  124   40   10 5228   75  602]
 [  40  101  158  122   49  457   77   35 4666  146]
 [  33   18   66   83  515  127    4  485  166 4452]]
```

Gdzie są błędy?

- normalizacja: podziel każdy element macierzy przez ilość próbek danej klasy

```
1  import matplotlib.pyplot as plt
2  row_sums = conf_mx.sum(axis=1, keepdims=True)
3  norm_conf_mx = conf_mx / row_sums
4  np.fill_diagonal(norm_conf_mx, 0) # żeby nie przeszkadzało
5  plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
6  f = "norm_conf_mx.png"
7  plt.savefig(f)
8  print(f)
```

Macierz błędów



Multi-label classification

- Przyporządkowanie do wielu klas jednocześnie.
- np. detekcja wielu twarzy na pojedynczym zdjęciu (każda twarz to osobna klasa).
- algorytm: K-neighbors

```
1 from sklearn.neighbors import KNeighborsClassifier
2 y_train_large = (y_train >=7) # duże cyfry
3 y_train_odd = (y_train % 2 == 1) # nieparzyste cyfry
4 y_multilabel = np.c_[y_train_large, y_train_odd]
5 print(y_multilabel)
```

```
[[False  True]
 [False False]
 [False False]
 ...
 [False  True]
 [False False]
 [ True False]]
```

Uczenie i sprawdzanie

```
1  knn_clf = KNeighborsClassifier()
2  knn_clf.fit(X_train, y_multilabel)
3  print(knn_clf.predict([mnist["data"][0]]))
```

```
[[False  True]]
```

- jest dużą cyfrą: False
- jest nieparzyste: True