Sztuczna inteligencja
MNIST i analiza metod klasyfikacji cyfr

Jakub Bronowski, 193208
Bartłomiej Krawisz, 193319
Stanisław Nieradko, 193044

gr. 2, sem. IV

1. Zastrzeżenia

Kanarek, w tej chwili accuracy liczone jest na 20% danych treningowych, a trenujesz na 80%. Zmień.
Bartek, odpal u siebie nowego maina z tworzeniem raportu z treningu. Zebrane dane umieścimy w sprawozdaniu.
Wszyscy, dopisać obserwacje i wnioski końcowe.

Spis treści

1. Zastrzeżenia 2
2. Opis zadania 4
3. Opis zbioru danych 4
4. Opis badanych metod 4
5. Trening modeli 5
6. Testowanie modeli 5
7. Implementacja aplikacji do testowania własnoręcznie narysowanych cyfr 7
8. Obserwacje 9
9. Wnioski 9

2. Opis zadania

Jako projekt zaliczeniowy z przedmiotu Sztuczna inteligencja zdecydowaliśmy się na analizę różnych metod rozpoznawania cyfr pisanych odręcznie. Celem projektu jest ich porównanie. W projekcie użyto 5 metod: k-najbliższych sąsiadów (ang. k-nearest neighbors), liniowa maszyna wektorów nośnych (ang. linear SVM recogniser), nieliniowa maszyna wektorów nośnych (ang. non-linear SVM recogniser), losowy las decyzyjny (ang. random decision forests) oraz sieć neuronowa (ang. neural network). Projekt zrealizowano w języku Python, wykorzystując biblioteki: numpy, matplotlib, scikit-learn, tensorflow, keras.

3. Opis zbioru danych

Zbiór danych to MNIST opracowany przez National Institute of Standards and Technology (agencja rządowa USA odpowiedzialna za rozwój i promocję produktów przemysłu USA). Wykorzystano wersję dostępną w bibliotece `pytorch`. Zbiór ten zawiera łącznie 70 000 obrazów przedstawiających cyfry o rozmiarze 28x28 pikseli pisanych odręcznie, białym tuszem na czarnym tle. 60 000 z nich to dane treningowe (pobierane z pliku `train-images-idx3-ubyte`) a pozostałe 10 000 to dane testowe (pobierane z pliku `t10k-images-idx3-ubyte`). Dane z obrazu są interpretowane jako macierz 26 x 26 z wartościami pikseli w skali szarości o kolorze z zakresu [0, 255], gdzie 0 oznacza kolor czarny, a 255 kolor biały. Każda cyfra jest przypisana do jednej z 10 klas oznaczających jej wartość [0, 9].

4. Opis badanych metod

4.1. K-najbliżsi sąsiedzi

Metoda k-najbliższych sąsiadów (ang. k-nearest neighbors, KNN) jest jedną z najprostszych metod klasyfikacji. Podjęcie decyzji o przynależności do klasy oparte jest na ocenie przynależności do klas k najbliższych punktów ze zbioru referencyjnego w przestrzeni cech. Użyto metody `KNeighborsClassifier()` z biblioteki `scikit-learn`. Wybrano domyślne parametry dla trenowania modelu, w szczególności parametr `n_neighbors` ustawiono na 5. Według literatury, metoda dobrze sprawdza się w przypadku problemów klasyfikacji, gdzie granica decyzyjna jest złożona i nieregularna.

4.2. Liniowa maszyna wektorów nośnych

Liniowa maszyna wektorów nośnych (Linear Support Vector Machine) to popularny algorytm klasyfikacji, który stara się znaleźć najlepszą linię, która maksymalnie oddziela klasy danych. Algorytm stara się znaleźć taką linię, która rozdziela dane na różne klasy maksymalizując marginesy, czyli odległości między linią a najbliższymi punktami danych z każdej klasy. Użyto metody `SVC()` z biblioteki `scikit-learn` z parametrem `kernel='linear'`. Uczenie polega na maksymalizacji marginesu przy jednoczesnym minimalizowaniu wartości kar dla źle rozdzielonych klas.

4.3. Nieliniowa maszyna wektorów nośnych

Nieliniowa maszyna wektorów nośnych (Non-linear Support Vector Machine) to rozszerzenie liniowej maszyny wektorów nośnych, które pozwala na rozdzielenie danych nieliniowych. W tym celu wykorzystuje się funkcję jądra (ang. kernel), która mapuje dane do przestrzeni o wyższej wymiarowości, licząc na to, że dane są w niej liniowo separowalne. Użyto metody `SVC()` z biblioteki `scikit-learn` z parametrem `kernel='rbf'` - przyjęto jądro Gaussa, `C=10` - skala regularyzacji, `gamma=0.001` - współczynnik jądra. Uczenie polega na znalezieniu hiperpłaszczyzny, która najlepiej separuje dane w przestrzeni cech. Według literatury, metoda ta dobrze sprawdza się w przypadku danych nieliniowo separowalnych o skomplikowanej strukturze.

4.4. Losowy las decyzyjny

Losowy las decyzyjny (Random Decision Forests) to metoda klasyfikacji, która polega na zbudowaniu wielu drzew decyzyjnych i wybraniu klasy, która jest najczęściej wybierana przez poszczególne drzewa (zasada “mądrości tłumu”, każda próbka do oceny jest analizowana przez każde z drzew). Drzewa są trenowane na podzbiorze losowo wybranych cech przy jednoczesnej redukcji overfittingu. Użyto metody `RandomForestClassifier()` z biblioteki `scikit-learn` z domyślnymi parametrami. Według literatury, metoda ta dobrze sprawdza się w przypadku dużych zbiorów danych, gdzie granica decyzyjna jest złożona i nieregularna.

4.5. Sieć neuronowa

Sieć neuronowa (Neural Network) to model inspirowany biologicznymi neuronami, pogrupowanymi w wiele warstw. Sieć jest trenowana na danych uczących, a proces ten polega na dostosowaniu wag między neuronami w taki sposób, aby zminimalizować błąd predykcji. Według literatury, sieci neuronowe dobrze sprawdzają się przy rozpoznawaniu

obiektów i rzekomo gwarantują najlepszą trafność. Sieć neuronowa powstała przy użyciu metod z biblioteki `pytorch`. Posiada ona 2 warstwy konwolucyjne. Pamatery pierwszej warstwy to: `input_channels=1`, `output_channels=32`, `kernel_size=3x3`. Parametry drugiej warstwy to: `input_channels=32`, `output_channels=64`, `kernel_size=3`. Następne 2 wartswy odrzucają odpowiednio 25% i 50% danych. Następne 2 warstwy są liniowe (liniowe transformacje danych z wykorzystaniem uprzednio wyznaczonych wag i biasów), z odpowiednio 9212 i 128 wejściami oraz 128 i 10 wyjściami. Wybraną funkcją aktywacji jest `Relu`. Wykorzystano także optymalizator `Adadelta` z domyślnymi parametrami.

5. Trening modeli

Kazdy z modeli został wytrenowany na tym samym zbiorze danych treningowych. Zastosowano ustawienia metod identyczne, jak opsiane powyżej. Wykorzystano dane treningowe z MNIST. Dla wszystkich badanych metod zbiór danych teningowych to 60 000 obrazów. Trenowanie odbywa się przy pomocy metody `fit()`. Ocenę treningu przeprowadza się poprzez wywołanie metody `score()` na danych treningowych z zestawy MNIST, który wypisuje się w konsoli. Tworzy się także raport z trenowania `training_report.txt`.
Sieć neuronowa została wytestowana na 14 epokach. Dane treningowe takie same jak dla pozostałych metod, generowane metodą `DataLoader()` z pamaretrem `batch_size=64` oraz włączonym tasowaniem zbioru. Jakość modelu oceniana jest na postawie wartości funkcji straty oraz dokładności klasyfikacji. Wartości wyznaczane są co epokę i zapisywane są w pliku `training_report.txt`.

5.1. Raport z treningu

Zrobić tabelę z wynikami accuracy dla każdej metody, dodać czas trenowania.

Metoda	Accuracy (%)
K-najbliżsi sąsiedzi	xx.xx
Liniowa maszyna wektorów nośnych	xx.xx
Nieliniowa maszyna wektorów nośnych	xx.xx
Losowy las decyzyjny	xx.xx

Tabela 1. Wyniki trenowania modeli niebędących siecią neuronową.

Epoka	Loss
1	xx.xx
2	yy.yy
3	and so on

Tabela 2. Wartość `loss` w kolejnych epokach.

6. Testowanie modeli

Dla każdej z metod po uprzednim wytrenowaniu wyznacza się macierz pomyłek, reprezentowaną graficznie. Im bardzej fioletowe punkty poza główną przekątną, tym większa liczba błędów. Na przecięciu wiersza i kolumny widać liczbę obrazów, które zostały zaklasyfikowane jako cyfra z wiersza, a były w rzeczywistości cyfrą z kolumny. Macierz pomyłek powstałą na podstawie dancyh testowych z MNIST (10 000 obrazów). Istnieje też możliowość wytestowania modelu na własnych danych, w folderze `test_data` należy umieścić obrazy do sprawdzenia. Wymaga się, aby cyfry były namalowane czarnym kolorem na białym tle, a obrazek był w proporcjach 1:1 (kwadrat). Wyniki testowania zapisywane są w pliku `testing_report.txt`. Modele można też testować przy użyciu aplikacji `drawing.py`.

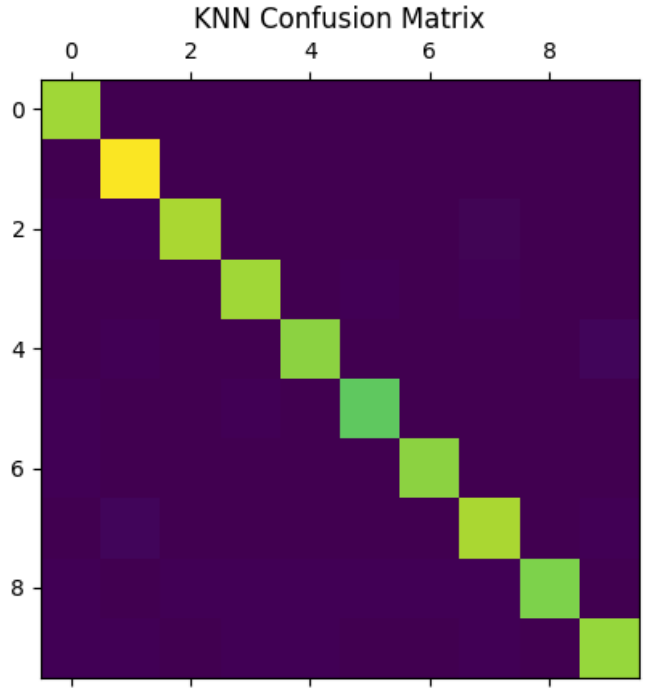
6.1. Raport z testowania ręcznego

W tabeli poniżej przedstawiono wyniki testowania modeli na danych opracowanych ręcznie.

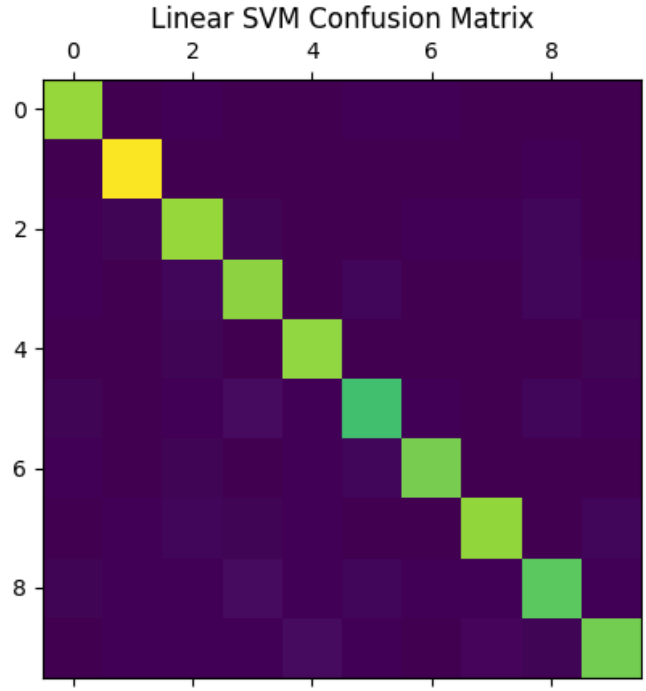
Wartość do rozpoznania	k-najbliżsi sąsiedzi	Liniowa maszyna wektorów nośnych	Nieliniowa maszyna wektorów nośnych	Losowy las decyzyjny	Sieć neuronowa
1	✓	✗	✓	✓	✓
4 (prosta)	✓	✗	✗	✗	✗
4 (trudna)	✓	✓	✓	✓	✓
9	✗	✓	✗	✗	✓

Tabela 3. Poprawność rozpoznania danych (testy ręczne).

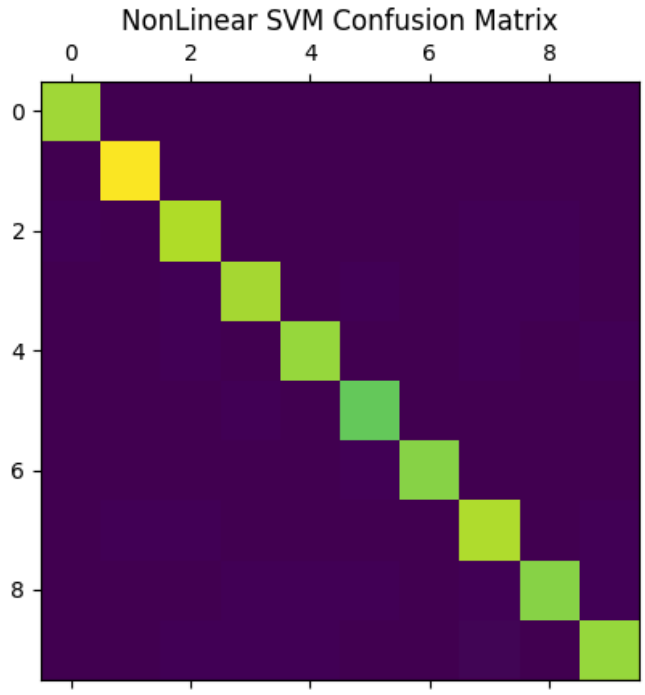
6.2. Macierze pomyłek



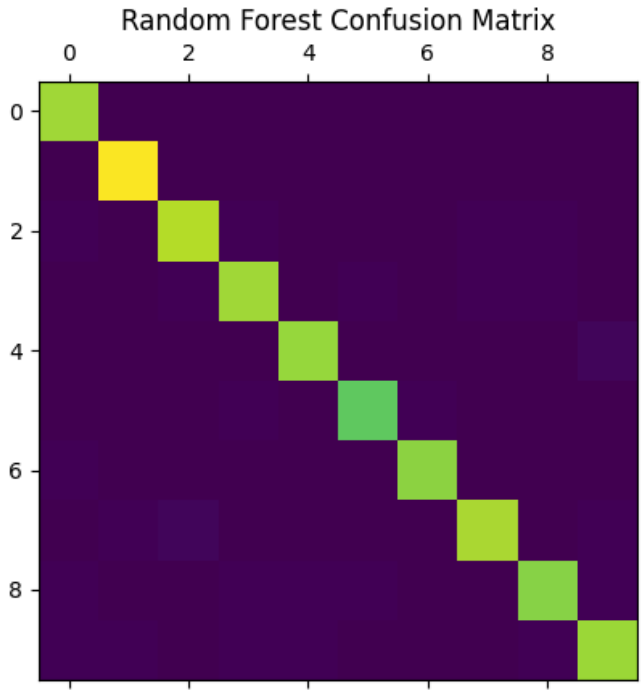
K-najbliżsi sąsiedzi



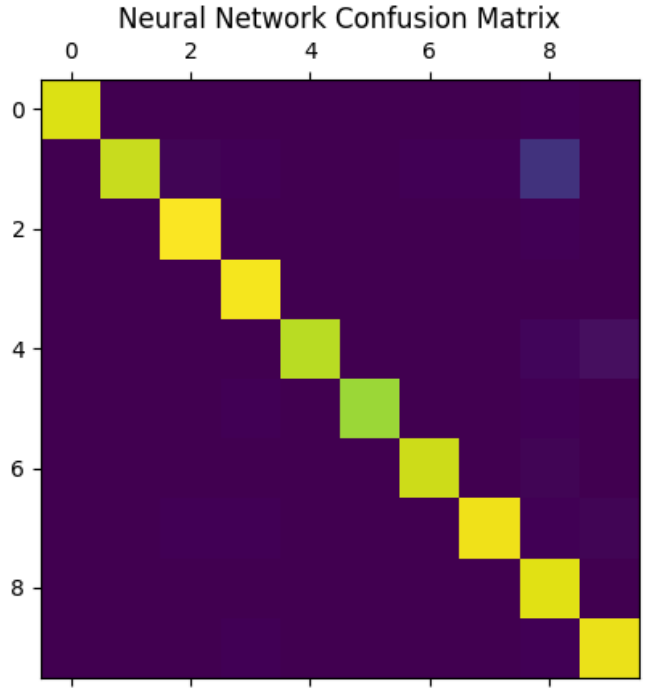
Liniowa maszyna wektorów nośnych



Nieliniowa maszyna wektorów nośnych



Losowy las decyzyjny



Sieć neuronowa

7. Implementacja aplikacji do testowania własnoręcznie narysowanych cyfr

Aby ułatwić testowanie własnoręcznie narysowanych cyfr, stworzono aplikację `drawing.py`, która pozwala na narysowanie myszą cyfry na ekranie i przetestowanie pobranych danych na wszystkich modelach. Zdecydowano się na taki krok z racji wygody użytkowania i możliwości szybkiego sprawdzenia działania modeli.

7.1. Opis aplikacji

GUI aplikacji stworzono przy użyciu biblioteki `pygame`. W prawym górnym rogu aplikacji umieszczono podgląd obrazu, który zostaje przekazany do modeli. Jest to obrazek 26x26 pikseli, na czarnym tle (zgodnie ze specyfikacją datasetu). W lewym górnym rogu pokazano wynik rozpoznania przez poszczególne modele, który jest również pisany na konsoli. W lewym dolnym rogu, gdy rozpoznawanie cyfry trwa, wyświetla się stosowna informacja. Klawiszami `0` - `9` można

wyświetlić w tle cyfrę z datasetu treningowego. Klawiszem `r` włączamy / wyłączamy możliwość generowania raportu (plik `drawing_report.txt`). Klawisz `s` zapisuje narysowaną cyfrę. Klawisz `c` czyści ekran. Klawisz `q` zamyka aplikację.

7.2. Przetwarzanie i normalizacja zebranych danych

Po narysowaniu cyfry, obrazek jest przekazywany do modeli. Aby to zrobić, obrazek jest przetwarzany w taki sposób, aby był zgodny ze specyfikacją danych treningowych. W trakcie rozwoju aplikacji zauważono polepszenie wyników, gdy dane zostały przekształcone w taki sposób, aby były bardziej zbliżone do danych treningowych. W tym celu zastosowano:

1. zapis obrazu w skali szarości,
2. przeskalowanie do rozmiaru 28x28 pikseli,
3. odwrócenie kolorów (z białego tła na czarne),
4. dla każdego piksela usunięcie szumów (wartości > 30 zamieniane na 255, w przeciwnym przypadku ustawiane na 0),
5. wycentrowanie cyfry

Operacje na obrazie to krok w dobrą stronę, ponieważ wyraźnie zwiększyła się poprawność rozpoznawania cyfr.

7.3. Moduł generowania raportu

Po wciśnięciu klawisza ‘r’ aplikacja zaczyna generować raport. Przed narysowaniem cyfry należy wcisnąć klawisz z cyfrą, którą zamierzamy narysować. Niewciśnięcie klawisza przed rysowaniem uniemożliwia dopisanie do raportu. Po narysowaniu jej zbiera się wynik rozpoznania. Dla każdego modelu i dla każdej cyfry zlicza się, ile razy model poprawnie rozpoznał daną cyfrę a ile razy pomylił się. Wyniki procentowe zapisywane są w pliku `drawing_report.txt`.

7.4. Przykładowe wyniki raportu z aplikacji

Dane zebrane podczas losowego wybierania cyfr do narysowania.

```
Total number of tests: 45

RandomForestTreeRecognizer:
0: 6.67% correct, 93.33% incorrect
1: 24.44% correct, 75.55% incorrect
2: 2.22% correct, 97.7% incorrect
3: 11.11% correct, 88.88% incorrect
4: 31.11% correct, 68.88% incorrect
5: 2.22% correct, 97.77% incorrect
6: 0.00% correct, 100.00% incorrect
7: 2.22% correct, 97.77% incorrect
8: 20.00% correct, 80.00% incorrect
9: 8.88% correct, 91.11% incorrect


KNearestNeighborsRecognizer:
0: 8.89% correct, 91.11% incorrect
1: 20.00% correct, 80.00% incorrect
2: 6.67% correct, 93.33% incorrect
3: 13.33% correct, 86.67% incorrect
4: 22.22% correct, 77.78% incorrect
5: 15.56% correct, 84.44% incorrect
6: 2.22% correct, 97.78% incorrect
7: 4.44% correct, 95.56% incorrect
8: 6.67% correct, 93.33% incorrect
9: 8.89% correct, 91.11% incorrect


NonLinearSVMRecognizer: 0: 4.44% correct, 95.56% incorrect
1: 13.33% correct, 86.67% incorrect
2: 6.67% correct, 93.33% incorrect
```

```
3: 11.11% correct, 88.89% incorrect
4: 33.33% correct, 66.67% incorrect
5: 8.89% correct, 91.11% incorrect
6: 6.67% correct, 93.33% incorrect
7: 6.67% correct, 93.33% incorrect
8: 8.89% correct, 91.11% incorrect
9: 8.89% correct, 91.11% incorrect


LinearSVMRecognizer: 0: 6.67% correct, 93.33% incorrect
1: 6.67% correct, 93.33% incorrect
2: 11.11% correct, 88.89% incorrect
3: 20.00% correct, 80.00% incorrect
4: 6.67% correct, 93.33% incorrect
5: 37.78% correct, 62.22% incorrect
6: 2.22% correct, 97.78% incorrect
7: 4.44% correct, 95.56% incorrect
8: 6.67% correct, 93.33% incorrect
9: 6.67% correct, 93.33% incorrect


NeuralNetworkRecognizer:
0: 6.66% correct, 93.33% incorrect
1: 33.33% correct, 66.66% incorrect
2: 8.88% correct, 91.11% incorrect 3: 6.66% correct, 93.33% incorrect
4: 8.88% correct, 91.11% incorrect
5: 20.00% correct, 80.00% incorrect
6: 0.00% correct, 100.00% incorrect
7: 4.44% correct, 95.55% incorrect
8: 6.66% correct, 93.33% incorrect
9: 13.33% correct, 86.66% incorrect
```

8. Obserwacje

W trakcie prowadzenia badań zaobserwowano, że im większy rysunek w aplikacji `drawing.py`, tym większa szansa na poprawne rozpoznanie. Zapewne ma to związek z późniejszym zmniejszaniem obrazka. Dla szeroko napisanych cyfr, na zmniejszonym obrazie wyraźniej widać przerwy np. ramiona cyfry 4, co zwiększa szansę na poprawne rozpoznanie. Sztuczna sieć neuronowa potrzebuje najwięcej czasu na rozpoznanie cyfry ze wszystkich badanych metod.

9. Wnioski

Najlepsze efekty da się osiągnąć przy użyciu sieci neuronowej. W naszym przypadku, sieć neuronowa osiągnęła najwyższą skuteczność w rozpoznawaniu cyfr. Warto zauważyć, że sieć neuronowa potrzebuje najwięcej czasu na rozpoznanie cyfry. Inne medoty sprawdzają się gorzej, ale są znacznie szybsze. Zaskakująco dobrze działa też metoda liniowej maszyny wektorów nośnych, w teście rysowania cyfr osiągnęła dla poszczególnych cyfr lepsze wyniki niż sieć neuronowa.