POLITECHNIKA GDAŃSKA | 120 LAT

Sztuczna inteligencja
MNIST i analiza metod klasyfikacji cyfr

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

gr. 2, sem. IV

Spis treści

1. Opis zadania

Niniejszy projekt zaliczeniowy z przedmiotu Sztuczna inteligencja przedstawia analizę różnych metod rozpoznawania cyfr pisanych odręcznie. Badane zagadnienie należy do dziedziny rozpoznawania obrazów (ang. computer vision), która skupia się na umożliwieniu komputerom rozumienia oraz interpretowania wizualnych danych ze świata zewnętrznego. Rozpoznawanie cyfr, w szczególności pisanych odręcznie, jest jednym z podstawowych problemów w dziedzinie rozpoznawania obrazów, dla którego opracowanie skutecznych metod znacząco zwiększa wydajność pracy i umożliwia efektywne przetwarzanie informacji utrwalonych na nieelektronicznych nośnikach danych. Jest to też problem z życia wzięty, ponieważ w praktyce spotykamy się z ręcznie pisanych cyframi na kartach pocztowych, formularzach, czy dokumentach urzędowych.
W niniejszej pracy zostaną omówione różnorodne podejścia do rozpoznawania cyfr, z naciskiem na techniki sztucznej inteligencji. Porównano sprawczość różnych metod rozpoznawania cyfr: k-najbliżsi sąsiedzi (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. convolutional neural network).
Projekt zrealizowano w języku Python, wykorzystując biblioteki: numpy, matplotlib, scikit-learn, tensorflow, keras.

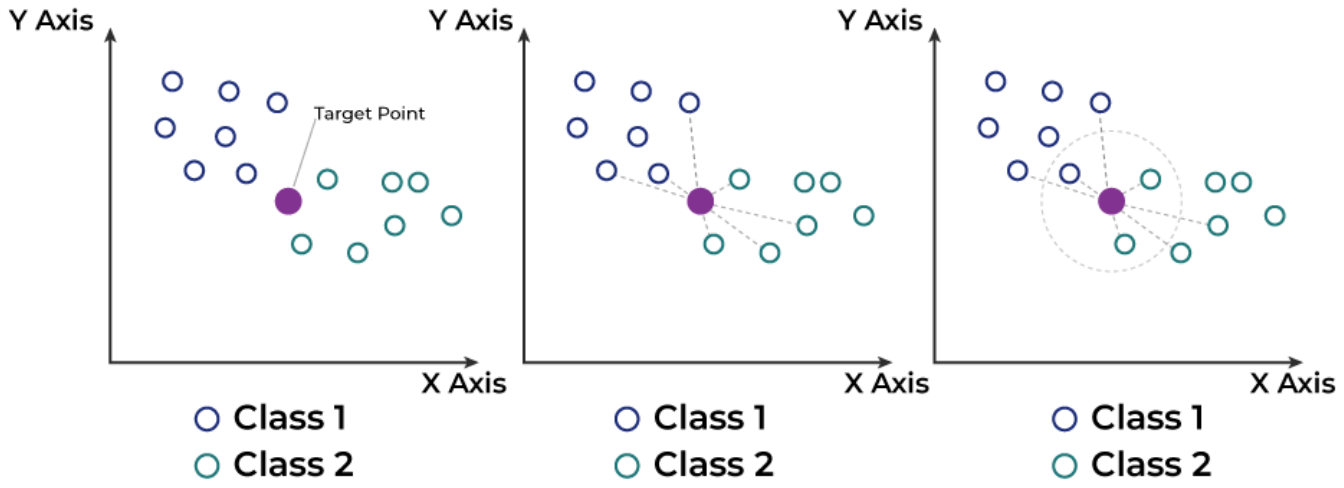
2. Opis zbioru danych

Zbiór danych, na których trenowano i testowano modele to MNIST opracowany przez National Institute of Standards and Technology (agencja rządowa USA odpowiedzialna za rozwój i promocję produktów przemysłu USA). 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].
Wykorzystano dataset MNIST dostarczony wraz z biblioteką `pytorch`. Dodatkowo pobraliśmy próbki od N = 6 osób, które rysowały przy użyciu myszki cyfry od 0 do 9. Poproszono każdą z N osób o narysowanie dokładnie 30 cyfr, po 3 z każdej. Cyfry do narysowania prezentowano w losowej kolejności. Łącznie pobrano 180 próbek. Celem tego działania była chęć sprawdzenia skuteczności modeli na danych, które nie pochodzą z datasetu MNIST a ze świata rzeczywistego. Obecnie w coraz większej ilości miejsc możemy spotkać się z pobieraniem tekstu od użytkownika przez rysowanie na ekranie myszką (lub rysikiem - nietestowane ze względu na brak urządzenia do pobrania takowej próbki), dlatego warto sprawdzić, jakie wyniki osiągną modele na takich danych.

3. Opis badanych metod

3.1. K-najbliżsi sąsiedzi

Metoda k-najbliższych sąsiadów (ang. k-nearest neighbors, KNN) jest jedną z najprostszych, a jednocześnie najbardziej intuicyjnych algorytmów klasyfikacji stosowanych w uczeniu maszynowym. Algorytm KNN działa na zasadzie “głosowania” wśród najbliższych sąsiadów danego punktu w przestrzeni cech. Proces klasyfikacji nowej próbki sprowadza się do znalezienia `K` najbliższych punktów treningowych (w metryce euklidesowej). Przypisana klasa dla badanej próbki jest taka, jak klasa najczęściej reprezentowana wśród sąsiadów.

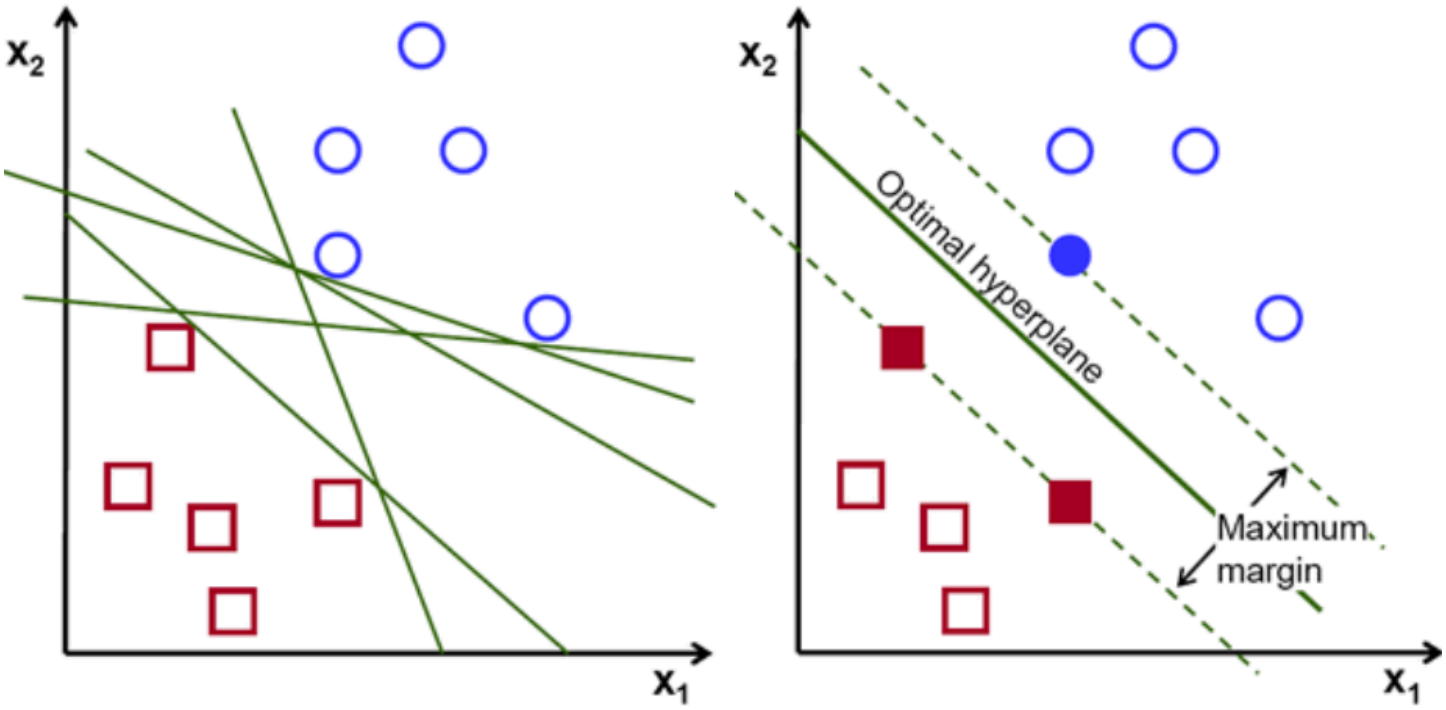


Schemat działania algorytmu K-najbliższych sąsiadów.

Przedmiotem badań jest metoda `KNeighborsClassifier()` z biblioteki `scikit-learn`, która implementuje opisaną wyżej metodę. Wybrano domyślne parametry dla trenowania modelu, w szczególności parametr `n_neighbors` ustawiono na 5. Parametr `n_neighbors` oznacza liczbę najbliższych sąsiadów, którzy mają wpływ na klasyfikację nowej próbki (utożsamiać z `k`). Zdecydowano się na takowe parametry, ponieważ zgodnie z literaturą i uprzednio przeprowadzonymi badaniami, dla takich wartości model osiąga najlepsze możliwe wyniki.

3.2. Liniowa maszyna wektorów nośnych

W ogólności SVM (Support Vector Machine, maszyna wektorów nośnych) to nadzorowana metoda uczenia, która służy do klasyfikacji danych przez znalezienie hiperpłaszczyzny, który maksymalizuje margines (tj. odległość między najbliższymi punktami z różnych klas a rzeczoną hiperpłaszczyzną) między klasami danych. W przypadku liniowej SVM hiperpłaszczyzna jest prostą. Algorytm stara się dobrać prostą tak, aby maksymalizować margines. Stosuje się też funkcję kosztu, dzięki której minimalizujemy błędy klasyfikacji przy jednoczesnym maksymalizacji marginesu.



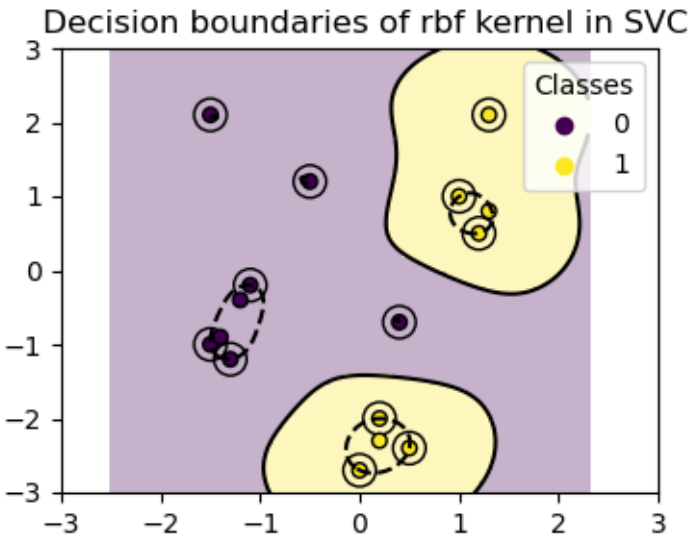
Schemat działania algorytmu liniowej maszyny wektorów.

W projekcie użyto metody `LinearSVC()` z biblioteki `scikit-learn` z parametrami domyślnymi, poza `dual='auto'`, który określa wartość tego parametru automatycznie, na podstawie dostarczonych danych i obranej strategii karania.

Nie zdecydowano się na edytowanie innych parametrów, ponieważ zgodnie z literaturą, domyślne wartości są optymalne dla tego typu problemów i gwarantują możliwie najlepsze dane.

3.3. Nieliniowa maszyna wektorów nośnych

Nieliniowa maszyna wektorów nośnych (Non-linear Support Vector Machine) to uogólnienie 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, tzn. istnieje hiperpłaszczyzna lub granica decyzyjna, która najlepiej oddzieli klasy danych.

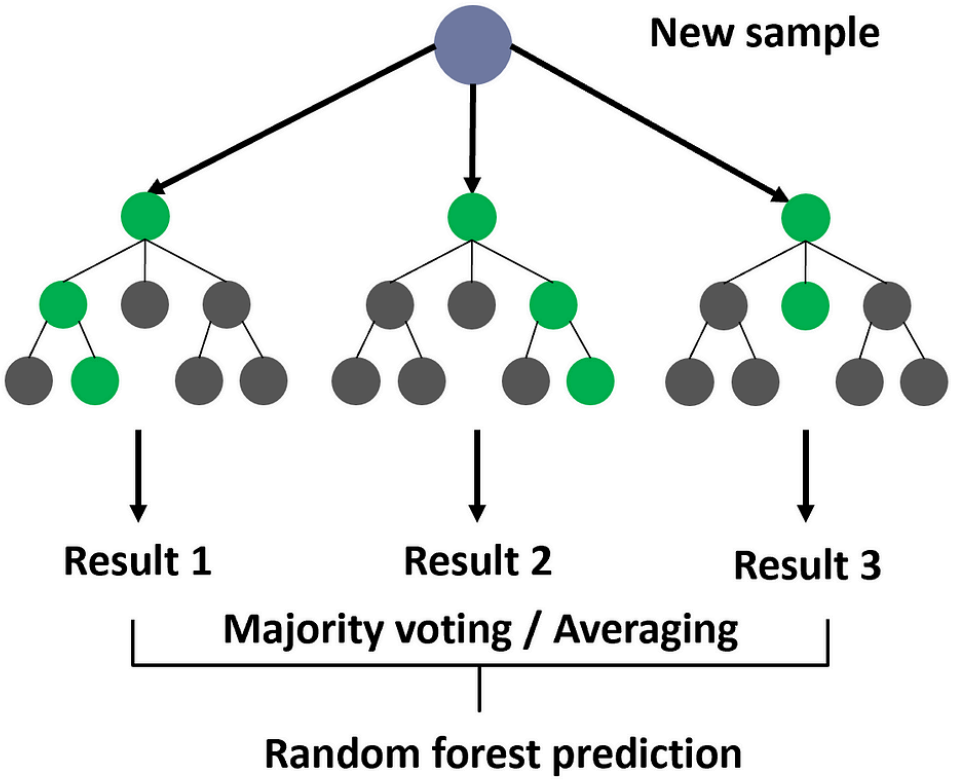


Przykład hiperpłaszczyzny podziału danych dla jądra Gaussa dla nieliniowej maszyny wektorów.

Zbadano nieliniową maszynę wektorów nośnych realizowaną przez metodą `SVC()` z biblioteki `scikit-learn`. Większość parametrów ustawiono na wartości domyślne, poza `kernel='rbf'`- przyjęto jądro Gaussa, `C=10` - skala regulacji dla funkcji minimalizującej stratę (loss), `gamma=0.001` - współczynnik jądra regulujący wpływ poszczególnej próbki na ustalenie granicy decyzji. Zdecydowano się na modyfikację większej ilości parametrów niż poprzednio celem maksymalnego dostosowania modelu do specyfiki danych. Wartości dobrano bazując na literaturze i wcześniejszych badaniach. Warto zwrócić uwagę na zapis w dokumentacji `The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples.`, co dla naszego zbioru danych sprawia, że trening jest czasochłonny.

3.4. Losowy las decyzyjny

Losowy las decyzyjny (Random Decision Forests) to metoda klasyfikacji oparta o budowanie lasu losowych drzew decyzyjnych. Zbieranie wyników opiera się na zasadzie agregacji rezultatów z poszczególnych drzew (zasada “mądrości tłumu”). Ma to również na celu poprawę dokładności klasyfikacji i zmniejszenia ryzyka przeuczenia (overfittingu). Każde drzewo decyzyjne jest zbudowane na podstawie losowego podzbioru danych treningowych i losowego podzbioru cech.



Wizualizacja działania algorytmu losowego lasu decyzyjnego.

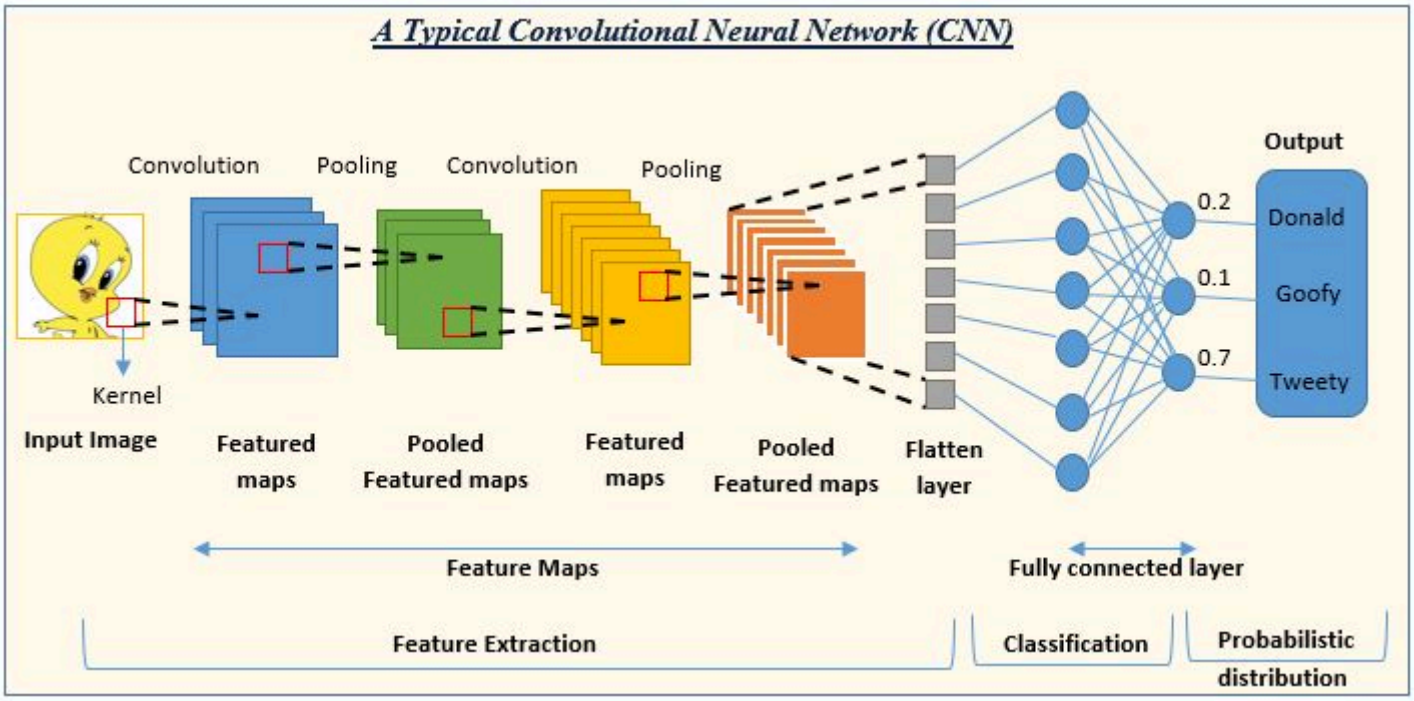
Zbadano metodę `RandomForestClassifier()` z biblioteki `scikit-learn` z domyślnymi parametrami. Ustawiono takie parametry, które dla tej metody okazały się być najlepsze dla zbioru danych MNIST, co potwierdzają wcześniejsze badania i literatura. Jedynym zmienionym parametrem jest `n_jobs=-1`, który zezwala na trenowanie modelu na wielu rdzeniach procesora. Jest to krok warty podjęcia ze względu na bardzo dużą ilość danych, które trzeba przetworzyć oraz ilość wielkość lasu. Potwierdza to też fakt, iż algorytm losowego lasu decyzyjnego cechuje się dużymi potrzebami obliczeniowymi.

3.5. Sieć neuronowa

Sieć neuronowa stanowi jeden z najbardziej zaawansowanych modeli uczenia maszynowego. Pomysłodawcy inspirowali się biologicznymi neuronami, pogrupowanymi w wiele warstw tworzących sieć. Konwolucyjna sieć neuronowa (Convolutional Neural Network) to zaawansowany architektura sieci neuronowej zaprojektowana do analizy danych w postaci siatki (macierzy) danych jak np. obrazy. Składa się ona z kilku warstw:

- konwolucyjna - służy do ekstrakcji cech z obrazu tworząc mapę cech,
- aktywacyjna - wprowadza nieliniowość do modelu,
- normalizacyjna - redukuje mapę cech agregując dane,
- spłaszczająca - przekształcenie mapy cech w wektor,
- w pełni połączona - klasyfikuje dane wejściowe,
- wyjściowa - generowanie ostatecznej predykcji.

Według literatury, sieci neuronowe dobrze sprawdzają się przy rozpoznawaniu obiektów i rzekomo gwarantują najlepszą trafność ze wszystkich badanych metod.



Wizualizacja konwolucyjnej sieci neuronowej.

Sieć neuronowa powstała przy użyciu metod z biblioteki `pytorch`. Posiada ona 2 warstwy konwolucyjne. Parametry pierwszej warstwy to: `input_channels=1` - ilość wejść, `output_channels=32` - ilość wyjść, `kernel_size=3x3` - rozmiar filtra ekstrakcji. Parametry drugiej warstwy to: `input_channels=32` - ilość wejść, `output_channels=64` - ilość wyjść, `kernel_size=3` - rozmiar filtra ekstrakcji. 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 parametrami `self.network.parameters()` - parametry dla każdej z warstw i `lr` - współczynnik skali przed wstawieniem do modułu. Budowę i parametry sieci neuronowej wzięliśmy z przykładowej implementacji przedstawionej w dokumentacji biblioteki / MNIST, a do dalszych badań wzięto model z najlepszymi rezultatami.

4. Trening modeli

Każdy z modeli został wytrenowany na tym samym zbiorze danych treningowych. Zastosowano ustawienia metod identyczne, jak opisane powyżej. Wykorzystano dane treningowe z MNIST. Dla wszystkich badanych metod zbiór danych treningowych to 60 000 obrazów. Trenowanie odbywa się przy pomocy metody `fit()`. Ocenę treningu przeprowadza się poprzez wywołanie metody `score()` na danych treningowych.
Sieć neuronowa została wytestowana na 14 epokach. Dane treningowe takie same jak dla pozostałych metod, generowane metodą `DataLoader()` z parametrem `batch_size=64` oraz włączonym tasowaniem zbioru. Jakość modelu oceniana jest na postawie wartości funkcji straty oraz dokładności klasyfikacji.

4.1. Specyfikacja środowiska trenowania

Do testów wykorzystano sprzęt o następujących parametrach:

- Arch Linux x86_64
- Procesor 13th Gen Intel(R) Core(TM) i5-13600K (20) @ 5.10 GHz
- NVIDIA GeForce RTX 3070 with CUDA
- 32 GB RAM

4.2. Czas trenowania modeli

Gdzie było to możliwe, korzystano z wielowątkowości. Czasy trenowania zapisano w tabeli.

Metoda	Czas treningu [min:s]
K-najbliżsi sąsiedzi	0:00
Liniowa maszyna wektorów nośnych	18:14
Nieliniowa maszyna wektorów nośnych	02:18
Losowy las decyzyjny	0:03
Sieć neuronowa	0:31

5. Testowanie modeli

Sprawdzono dokładność modeli na danych treningowych celem sprawdzenia, czy modele nie zostały przetrenowane. Następnie przetestowano modele na danych testowych znajdujących się w datasecie MNIST, wygenerowano macierz pomyłek. Na końcu przeprowadzono testy na danych własnych, które zostały pobrane od N = 6 osób, od każdej z nich próbko 30 cyfr (po 3 obrazki na cyfrę). Działanie miało na celu sprawdzenie skuteczności modeli na danych ze świata rzeczywistego. Wszystkie pomiary wykonano na takim samym sprzęcie, jak powyżej.

5.1. Testy z danych treningowych

Rezultaty z testów na danych treningowych przedstawia poniższa tabela.

Metoda	Czas testu [s]	Dokładność [%]
K-najbliżsi sąsiedzi	77.76	98.19
Liniowa maszyna wektorów nośnych	0.73	92.75
Nieliniowa maszyna wektorów nośnych	261.31	99.80
Losowy las decyzyjny	0.34	100.00
Sieć neuronowa	7.89	98.87

Dokładność każdego z modeli jest zadowalająca. Modele powinny dobrze radzić sobie z klasyfikacją danych testowych. Przy okazji obserwujemy, że najwolniejszym z modeli jest nieliniowa maszyna wektorów nośnych, co jest zgodne z przewidywaniami (złożność nie mniejsza niż kwadratowa). Drugim najwolniejszym modelem okazał się model k-najbliższych sąsiadów, co jest zaskakujące, ponieważ jest to jedna z najprostszych metod. Czas testu dla sieci neuronowej jest zadowalający, co może świadczyć o dobrej optymalizacji modelu (zapewne wiele pomogła wielowątkowość). Czas działania pozostałych modeli jest zgodny z oczekiwaniami.

5.2. Testy z danych MNIST

Rezultaty z testów na danych treningowych przedstawia poniższa tabela.

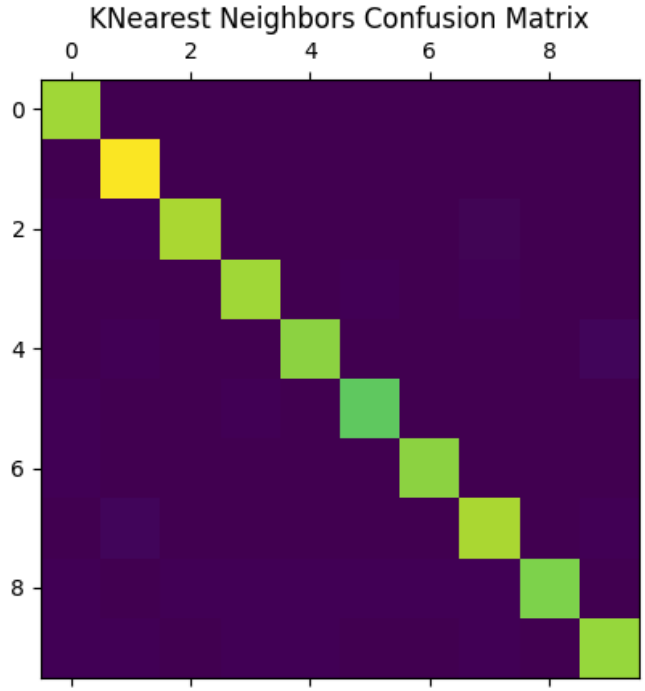
Metoda	Czas testu [s]	Dokładność [%]
K-najbliżsi sąsiedzi	13.01	96.88
Liniowa maszyna wektorów nośnych	0.73	91.55
Nieliniowa maszyna wektorów nośnych	42.54	97.24
Losowy las decyzyjny	0.05	97.05
Sieć neuronowa	1.30	98.34

Dla danych testowych z datasetu MNIST każdy z modeli osiągnął dobre wyniki, niższe niż dla danych treningowych. Najsłabiej wypadła liniowa maszyna wektorów nośnych, co jest zgodne z przewidywaniami. Najlepsze wyniki osiągnęła sieć neuronowa, co również jest zgodne z przewidywaniami. Warto zwrócić uwagę, że dokładność dla sieci neuronowych nieznacznie zmalała (ok. 0.5 punktu procentowego), co świadczy o dobrym treningu. Dla pozostałych modeli spadki są większe, choć nie przekraczają one 3 punktów procentowych. Można wnioskować, że parametry ustawione dla modeli są optymalne, a badania naukowe, które wykorzystano do ich ustawienia, nie kłamały.
Jeśli chodzi o czas wykrywania, to zaskakująco dobrze wypadł model losowego lasu decyzyjnego - jest najszybszy mimo wymaganych obliczeń na dużych drzewach. Najlepszą pod względem czasu i dokładności jest sieć neuronowa, co jest zgodne z przewidywaniami autorów projektu. Możemy też zaobserwować, że nieliniowa maszyna wektorów

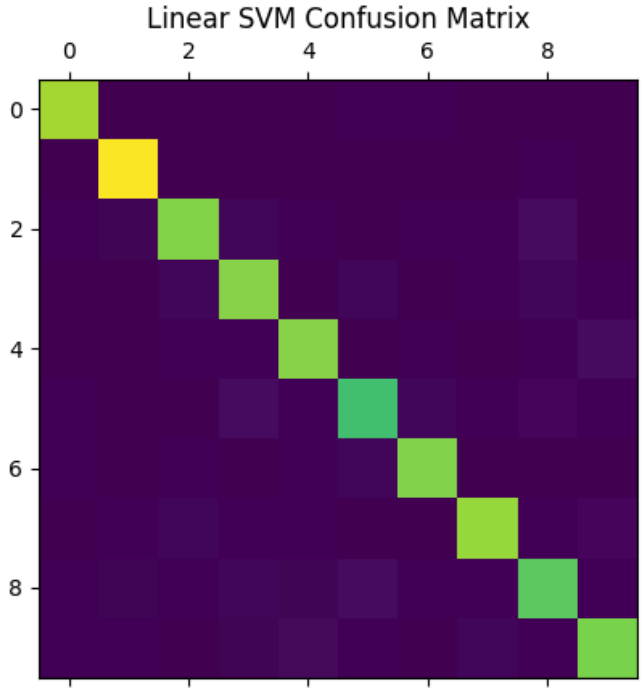
nośnych jest nieco lepsza od metody k-najbliższych sąsiadów, niemniej warto zastanowić się nad jej wdrożeniem. Różnica w dokładności jest na poziomie 0.36 punktu procentowego przy dużej różnicy w czasie wykonania (nieco mniej dokładna metoda jest 2,25 razy szybsza).

5.3. Macierze pomyłek

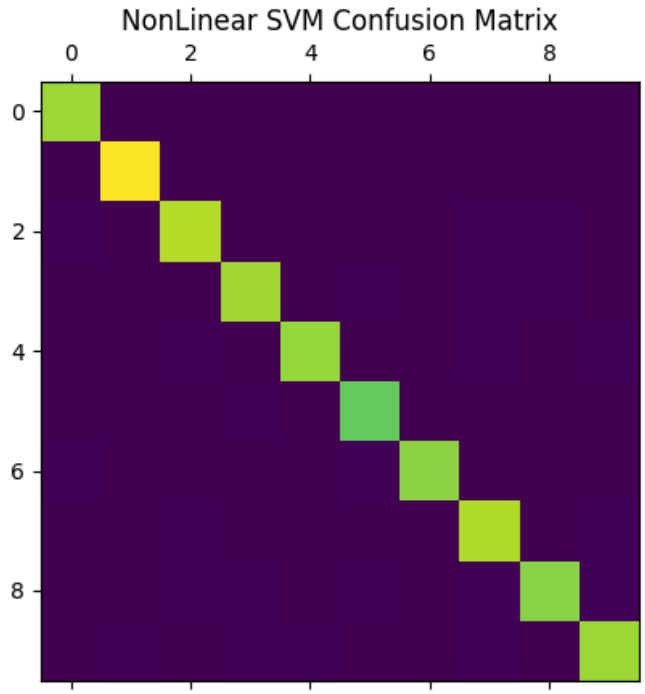
Macierze pomyłek dla każdej z metod zostały zapisane w plikach `KNearestNeighborsConfusionMatrix.png`, `LinearSVMConfusionMatrix.png`, `NonLinearSVMConfusionMatrix.png`, `RandomForestTreeConfusionMatrix.png`, `NeuralNetworkConfusionMatrix.png`. Wygenerowano je dla danych testowych z datasetu MNIST.



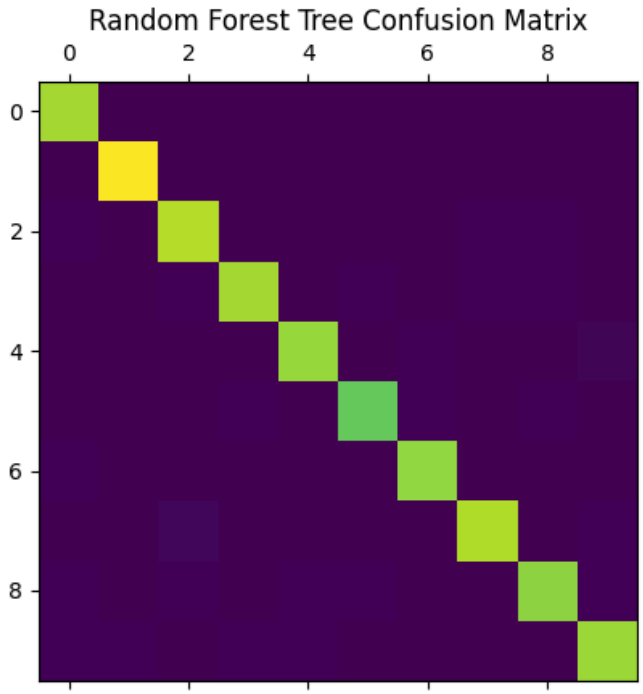
K-najbliżsi sąsiedzi



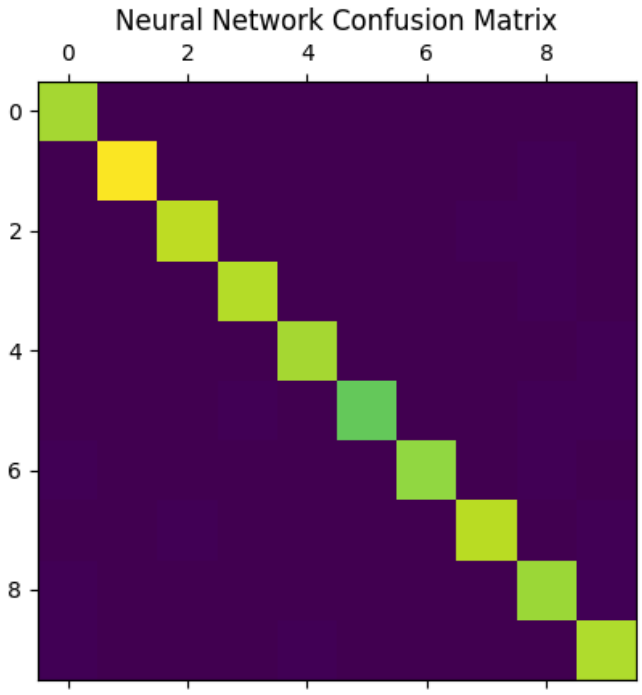
Liniowa maszyna wektorów nośnych



Nieliniowa maszyna wektorów nośnych



Losowy las decyzyjny



Sieć neuronowa

5.4. Ocena przetrenowania modeli

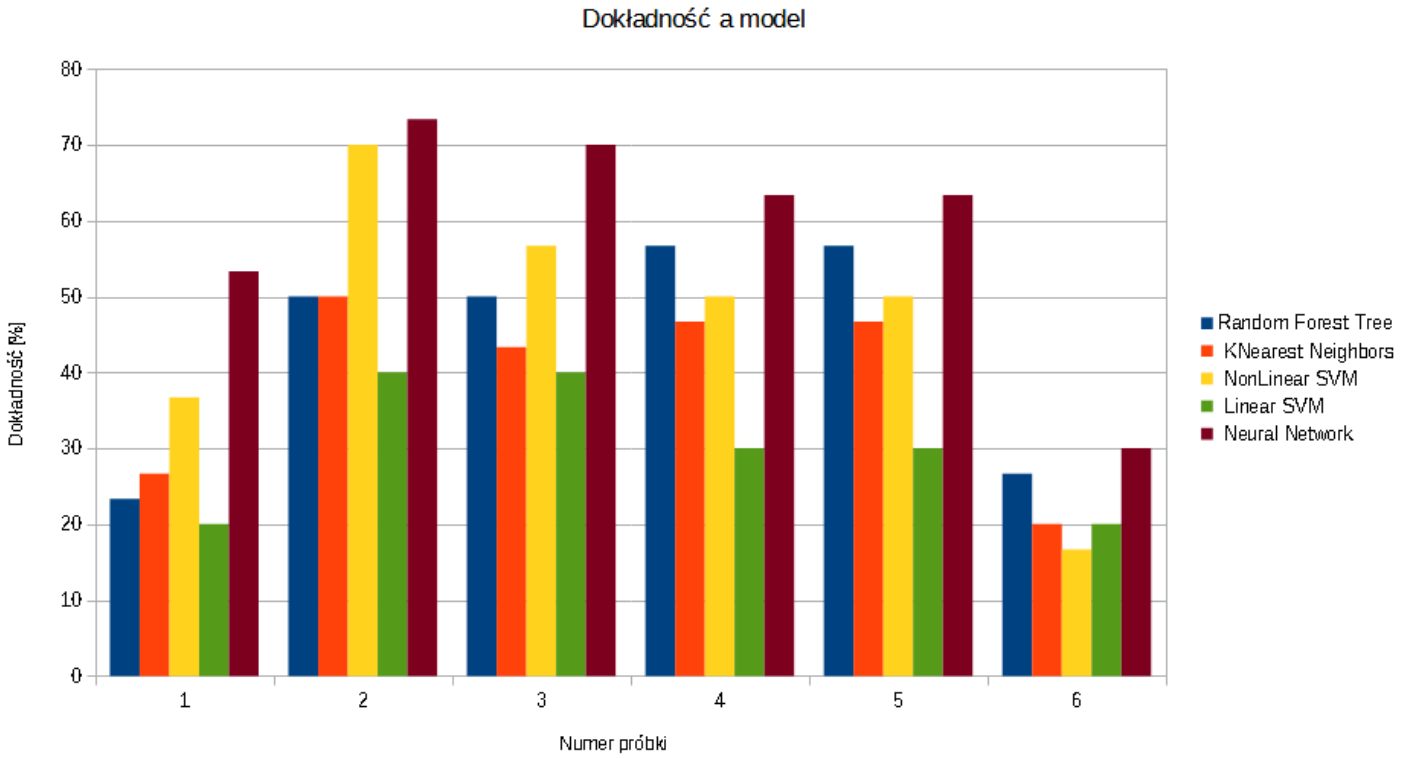
Poniższa tabela prezentuje dokładność modeli na danych treningowych i testowych oraz różnicę tychże wartości. Na jej podstawie można ocenić, czy modele nie zostały przetrenowane.

Metoda	Dokładność dla danych treningowych [%]	Dokładność dla danych testowych MNIST [%]	Różnica [p.p.]
Losowy las decyzyjny	100.00	97.05	2.95
Nieliniowa maszyna wektorów nośnych	99.80	97.24	2.56
K-najbliżsi sąsiedzi	98.19	96.88	1.31
Liniowa maszyna wektorów nośnych	92.75	91.55	1.2
Sieć neuronowa	98.87	98.34	0.53

Większość modeli nie została przetrenowana, w szczególności sieć neuronowa wykazuje bardzo małą różnicę wartości dokładności. Niepokojąco duże różnice zaobserwowano dla losowego lasu decyzyjnego oraz nieliniowej maszyny wektorów nośnych. Uznano, że prawdopodobnie modele zostały przetrenowane. W dalszych badaniach użyto nieedytowanych modeli celem sprawdzenia, jak potencjalnie przetrenowanie wpłynie na dokładność rozpoznawania cyfr.

5.5. Testy z danych własnych

Pobrano od N = 6 osób łącznie 180 próbek, po równo na każdą z cyfr. Z racji małej ilości danych do przetworzenia, nie zdecydowano się na pomiar czasu pracy każdego z modelu. Przez dokładność rozumie się stosunek dobrych rozpoznań do ilości próbek.
Rezultaty z testów na danych treningowych przedstawia poniższy wykres kolumnowy przedstawiający dokładność rozpoznania względem 30 cyfr narysowanych przez kolejne osoby.



Dokładność modeli dla zestawu cyfr narysowanych przez kolejne osoby.

Potwierdza się przypuszczenie, że sieć neuronowa najlepiej poradzi sobie z rozwiązaniem zadanego problemu klasyfikacji cyfr. Inną metodą, która również dobrze radzi sobie z klasyfikacją cyfr jest nieliniowa maszyna stanów nośnych, porównywalnie z metodą lasów losowych. Metoda k-najbliższych sąsiadów oraz liniowa maszyna wektorów nośnych radzą sobie zdecydowanie gorzej od innych. Najgorsze wyniki odnotowuje liniowa maszyna wektorów nośnych, prawdopodobnie przez problemy z dopasowaniem do danych.
Konkretne wartości dokładności w % dla każdej z osób przedstawia poniższa tabela.

Osoba	K-najbliżsi sąsiedzi [%]	Liniowa maszyna wektorów nośnych [%]	Nieliniowa maszyna wektorów nośnych [%]	Losowy las decyzyjny [%]	Sieć neuronowa [%]
1	2.33	2.67	3.67	20.00	5.33
2	50.00	50.00	70.00	40.00	7.33
3	50.00	4.33	5.67	40.00	70.00
4	5.66	4.67	50.00	30.00	6.33
5	5.66	4.67	50.00	30.00	6.33
6	2.66	20.00	1.67	20.00	30.00
średnia	43.89	38.89	46.67	30.00	58.89

6. 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.

6.1. 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.

6.2. 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. 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.
Zaobserwowano przetrenowanie tylko jednego modelu (losowy las decyzyjny), mimo iż nieliniowa maszyna wektorów nośnych również cechowała się dużą różnicą dokładności. Niemniej tylko losowy las decyzyjny okazał się osiągać słabe wyniki (możliwe, że jest to jeden z powodów).
Do rozpoznawania cyfr warto implementować model nieliniowej maszyny wektorów nośnych, ponieważ osiąga ona dużo lepsze wyniki niż jej liniowy odpowiednik.

8. Wnioski

Najlepsze efekty da się osiągnąć przy użyciu sieci neuronowej, co potwierdza hipotezę początkową. W naszym przypadku, sieć neuronowa osiągnęła najwyższą skuteczność w rozpoznawaniu cyfr. Warto zauważyć, że sieć neuronowa potrzebuje dużo czasu, choć nie najwięcej spośród testowanych metod. Można przyjąć, że szybkość jest średnia. Jeśli uwzględnić też skuteczność, to sieć neuronowa jest najlepszym wyborem. Inne metody sprawdzają się gorzej, ale mogą być znacznie szybsze. Zaskakująco dobrze działa też metoda nieliniowej maszyny wektorów nośnych, w teście dla narysowanych cyfr osiągnęła drugi najlepszy wynik.