Rozpoznawanie cyfr

Krzysztof Pożoga
Zuzanna Tyszka
Kacper Walasek

28.05.2021

1 Wstęp

Niniejszy projekt to aplikacja służąca do rozpoznawania cyfr na bitmapach w skali szarości. Celem projektu jest zapoznanie się z podstawami technik uczenia maszynowego, a w szczególności algorytmami *k najbliższych sąsiadów* oraz *maszyna wektorów nośnych*.

Program trenuje wybrane algorytmy, a także umożliwia użytkownikowi wprowadzenie własnego przykładu, który zostanie poddany próbie rozpoznania cyfry.

2 Opis problemu

2.1 Dane

Źródłem danych dla naszego zagadnienia jest strona kaggle.com oraz zamieszczony na niej konkurs pt. "Digit Recognizer"(https://www.kaggle.com/c/digit-recognizer/overview). Korzystamy ze znajdujących się na stronie danych w pliku train.csv. Plik ten zawiera 785 kolumn, z czego w pierwszej kolumnie przechowywana jest informacja o etykiecie danej próbki, a w pozostałych wartości liczbowe kolorów kolejnych pikseli. Badane obrazy (czyli poszczególne wiersze dokumentu) zawierają odręcznie napisane cyfry (z przedziału 0-9) w odcieniach szarości.

2.2 Problem

Naszym zadaniem jest, przy pomocy wyżej opisanego zbioru, zapoznanie się z podstawowymi technikami uczenia maszynowego i wytrenowanie wybranych algorytmów w celu uzyskania jak najwyższej skuteczności rozpoznawania obrazu. Postanowiliśmy do tego celu wybrać algorytmy k najbliższych sąsiadów oraz maszyna wektorów nośnych (SVM), jako jedne z najpowszechniej stosowanych i najbardziej uniwersalnych algorytmów.

2.3 K najbliższych sąsiadów

Jest to algorytm stosowany w zagadnieniach klasyfikacji. Metoda ta polega na wyznaczeniu k sąsiadów, względem których badany element jest położony najbliżej (zgodnie z wybraną metryką, np. metryką Euklidesową), a następnie wyznaczeniu wyniku w oparciu o głos większości. Oznacza to, że badany obiekt klasyfikujemy do tej klasy, do której należy najwięcej jego sąsiadów.

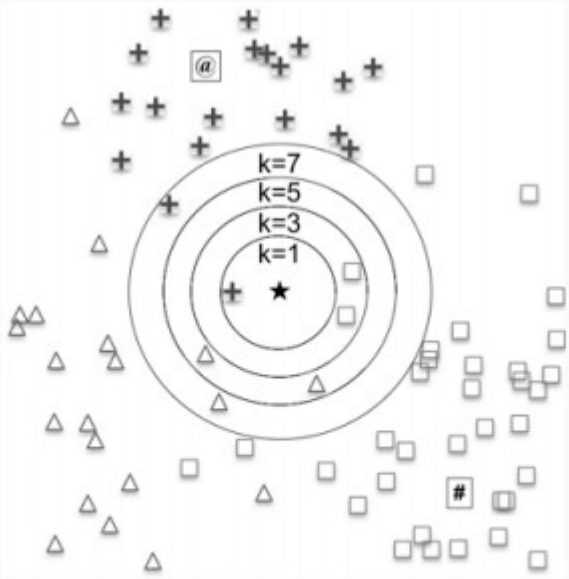
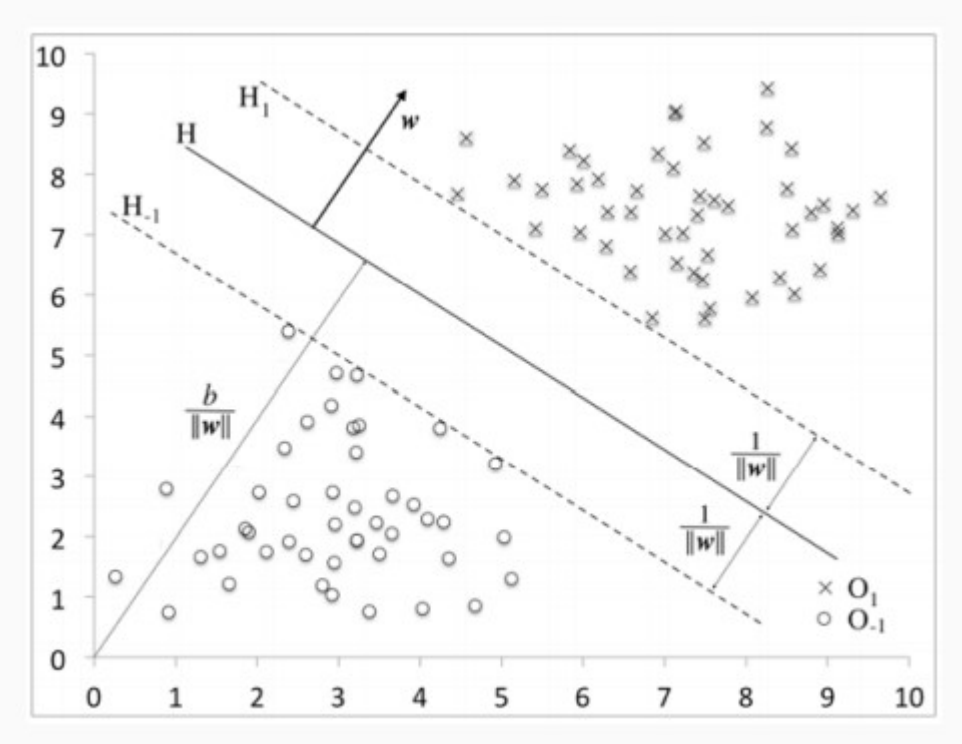


Figure 1: Illustration of the k-NN classifier. The pattern marked with a black star is being classified.

Rysunek 1: Ilustracja klasyfikatora k najbliższych sąsiadów. (źródło: slajdy z wykładu WTUM)

2.4 Maszyna wektorów nośnych (SVM)

Metoda ta polega na znalezieniu hiperpłaszczyzny możliwie najlepiej rozdzielającej dwie badane klasy (w rzeczywistości niezwykle rzadkie są przypadki klas liniowo separowalnych, dlatego dążymy tu do osiągnięcia najlepszego kompromisu pomiędzy dokładnością a szybkością obliczeń), a także odpowiednio szerokiego marginesu pomiędzy tymi klasami (im szerszy margines, tym lepsze własności generalizacji i mniejsze ryzyko przeuczenia algorytmu). Metoda ta została zaprojektowana dla klasyfikacji dwóch klas, dlatego w przypadku problemu wieloklasowego należy algorytm zastosować n-krotnie, gdzie n to ilość klas badanego problemu, w schemacie i-ta klasa vs. reszta danych.



Rysunek 2: Ilustracja klasyfikatora maszyna wektorów nośnych. (źródło: slajdy z wykładu WTUM)

3 Opis algorytmów

3.1 Przygotowanie danych

Zanim przystąpimy do trenowania algorytmów musimy odpowiednio przygotować nasze dane. Po pobraniu danych z pliku .csv przy użyciu metody **read_data.py** dokonujemy binaryzacji obrazu (**binarization.py**). Metody **prepare_sets.py** i **sort_entries.py** dzielą wczytane dane na zbiór treningowy i testowy losowo, jednak z zachowaniem proporcji pomiędzy klasami problemu.

Następnie dokonujemy rzutowania oraz tranzycji horyzontalnych i wertykalnych (**transition_horizontal.py**, **transition_vertical.py**, **projection_ horizontal.py**, **projection_vertical.py**). Z tak uzyskanych histogramów ekstrahujemy następnie informacje o minimalnych i maksymalnych wartościach, medianie i średniej (**find_average.py**, **find_median.py**, **find_min_position.py**, **find_max_position.py**).

Finalny etap przygotowania danych stanowi metoda **data_extraction.py**. Korzysta ona z wszystkich wcześniej wymienionych metod oraz dodatkowych metod dostarczonych przez bibliotekę *Mahotas*, aby dla każdego badanego obrazu wyekstrahować zestaw kluczowych cech: pozycję i wartość minimalnej i maksymalnej wartości oraz medianę i średnią w każdym z rzutowań i tranzycji, a także ekscentryczność, informację o osiach elipsy i stopień zaokrąglenia. Taki zestaw cech będzie bezpośrednio stosowany do trenowania algorytmów, co, w porównaniu z surowymi danymi, znacząco przyspieszy ten proces.

3.2 Główne algorytmy

Algorytmy implementują wspólny interfejs **alghoritm.py**, zawierający dwie metody - train i predict.

3.2.1 K najbliższych sąsiadów

Opisany we wcześniejszym rozdziale algorytm K najbliższych sąsiadów implementujemy przy pomocy klasy *KNeighboursClassifier* z biblioteki *Sklearn*.

3.2.2 Maszyna wektorów nośnych

W przypadku maszyny wektorów nośnych korzystamy z klasy *svm* pochodzącej z biblioteki *Sklearn* a konkretniej z SVC (czyli Support Vector Classifica-

tion). Na tym etapie mamy możliwość wyboru kernela oraz jego parametrów, co ma znaczący wpływ na skuteczność działania programu (więcej w dalszym rozdziale).

3.3 Zastosowanie algorytmów i interfejs użytkownika

Metody **train_algorithm.py** oraz **validate.py** służą do trenowania i testowania zadanych algorytmów. Metoda **validate.py** sprawdza dla każdego w wyników działania wytrenowanego algorytmu jego zgodność z etykietą próbki danych.

Z punktu widzenia użytkownika najważniejszy jednak jest interfejs, zaimplementowany w **show_interface.py**. Metoda ta tworzy prosty interfejs graficzny, umożliwiający w łatwy sposób wytrenowanie danych algorytmem k najbliższych sąsiadów bądź maszyna wektorów nośnych (poprzez wciśnięcie odpowiedniego przycisku), a także przetestowanie własnego przykładu: użytkownik może narysować dowolny przykład, a po wciśnięciu przycisku wyświetli się rezultat dla obu algorytmów.

4 Scenariusze testowe

Przy użyciu stworzonego rozwiązania przeprowadzono testy mające na celu zbadanie następujących zagadnień:

- Porównanie skuteczności różnych algorytmów klasyfikacji na podstawie danych pobranych ze strony konkursu,
- Porównanie skuteczności rozpoznawania dla różnych cyfr,
- Porównanie skuteczności rozpoznawania cyfr wprowadzonych na obrazkach wprowadzonych przez użytkownika do skuteczności na obrazkach w takim samym formacie jak obrazki w zbiorze treningowym.

4.1 Podział danych

Aby przeprowadzić testy w wymienionych obszarach, należało w odpowiedni sposób podzielić wykorzystywane dane. Ustalony podział zakładał, że 90% danych zostanie przeznaczony na trening algorytmów (zbiór treningowy),

natomiast 10% zostanie użyty do testów uzyskanych rozwiązań (zbiór walidacyjny). Wydzielenie zbioru testowego nie było konieczne ze względu na użycie metod cross validacyjnych.

Przy podziale danych dopilnowano, aby stosunek ilości obrazków przedstawiających poszczególne cyfry w obu zbiorach był taki sam. Upewniono się również, że ilość obrazków przedstawiających poszczególne liczby jest podobna.

4.2 Porównanie skuteczności algorytmów klasyfikacji

W tym scenariuszu porównano skuteczności rozpoznawania cyfr z obrazów dla algorytmów k najbliższych sąsiadów oraz SVM z kernelem liniowym i radialnym.

Na początku dla każdego z algorytmów poszukano parametrów, które zapewniały najlepsze właściwości. Dobieranie parametrów algorytmów odbywało się przy pomocy clasy $GridSearchCV$ zawartej w bibliotece $scikit-learn$. Jest to narzędzie poszukujące najskuteczniejszego zestawu parametrów dla algorytmu spośród podanych możliwości.

W przypadku algorytmu SVM z kernelem radialnym dobierano w ten sposób wartość regularyzacji C oraz parametru gamma, a dla kernela liniowego była to tylko regularyzacja C. Tym sposobem szukano również najlepszej wartości k dla algorytmu k najbliższych sąsiadów. Przeprowadzono testy dla wartości przedstawionych w tablicy 1.

Kernel	Parametr	Wartość						
Radialny	C	40	50	60	70	80	90	100
	gamma	0.0001	0.001	0.005	0.01	0.02	0.03	0.1
Liniowy	C	1	2	5	10			

Tablica 1: Testowane wartości

Wyniki tego eksperymentu pokazały najlepsze parametry użytych metod. Na koniec tego scenariusza przetestowano skuteczność rozpoznawania liczb na obrazkach przy użyciu trzech wymienionych algorytmów z najlepszymi parametrami. Najlepsze parametry algorytmów i ich skuteczności przedstawiono w tabeli 2.

Spośród użytych algorytmów najskuteczniejszy okazał się być SVM z kernelem radialnym. Natomiast najgorzej zaprezentował się algorytm k najbliższych sąsiadów.

Metoda	Parametry	Skuteczność
SVM - kernel radialny	C=60, gamma=0.1	84.53%
SVM - kernel liniowy	C=1	79.98%
k najbliższych sąsiadów	k=7	76.55%

Tablica 2: Najlepsze parametry metod i ich skuteczność

4.3 Porównanie skuteczności rozpoznawania dla różnych cyfr

Dla trzech algorytmów uzyskanych w poprzednim rozdziale przeanalizowano rezultaty rozpoznawania cyfr w zbiorze testowym. Sprawdzono skuteczność rozpoznawania każdej z cyfr. Rezultaty przedstawiono w tabeli 3.

Cyfra na obrazku	SVM - radialny	SVM - liniowy	k najbliższych sąsiadów
0	94%	90%	89%
1	97%	97%	96%
2	74%	65%	70%
3	65%	59%	53%
4	89%	85%	83%
5	63%	63%	50%
6	87%	79%	78%
7	92%	85%	84%
8	91%	89%	79%
9	87%	78%	76%

Tablica 3: Skuteczność dla każdej z cyfr

4.4 Wyznaczanie skuteczności rozpoznawania cyfr wpisanych ręcznie

Na koniec przeprowadzono test sprawdzający skuteczność rozpoznawania cyfr wpisywanych ręcznie przez użytkownika. Stworzono zbiór 50 wprowadzonych przez użytkownika obrazków (po 5 na każdą cyfrę) i sprawdzono skuteczność z jaką algorytmy rozpoznawały cyfry na obrazkach. Dla algorytmu SVM z kernelem radialnym wartość to wynosiła 58%, natomiast dla algorytmu k najbliższych sąsiadów - 54%.

Wyniki tego testu pokazują, że skuteczność rozpoznawania cyfr wpisywanych przez użytkownika jest dużo mniejsza niż w przypadku zblioru testowego.

5 Bibliografia

- slajdy z przedmiotu Warsztaty Technik uczenia maszynowego, dr Agnieszka Jastrzębska
- dokumentacja biblioteki scikit-learn, scikit-learn.org