

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Sprawozdanie

Wprowadzenie do sztucznej inteligencji

Ćwiczenie nr. 4

Mikołaj Bańkowski

Numer albumu 310408

prowadzący
Grzegorz Rypeś

Warszawa 2023

1. Rodzaj klasyfikatora

Indeks studenta = 310408

$$\sqrt{3,14 \cdot 310408} = 997,2594$$

x = 259

























259 mod(3) = 1

Rodzaj klasyfikatora: maszyna wektorów nośnych SVM

2. Inżynieria atrybutów

Generacja danych i przekształcanie stanów gry w cechy zostało zaimplementowane w funkcji `game_state_to_data_sample` w pliku `model.py`

Zagrano wiele razy w grę, aby zebrać dane z wielu scenariuszy gier

	2024-04-19_19-40-54	19.04.2024 19:40	Plik PICKLE	1 KB
	2024-04-19_19-41-07	19.04.2024 19:41	Plik PICKLE	3 KB
	2024-04-19_19-41-50	19.04.2024 19:41	Plik PICKLE	8 KB
	2024-04-19_19-49-16	19.04.2024 19:49	Plik PICKLE	9 KB
	2024-04-19_19-50-17	19.04.2024 19:50	Plik PICKLE	7 KB
	2024-04-19-20-18-14	19.04.2024 20:18	Plik PICKLE	33 KB
	2024-04-20_17-17-14	20.04.2024 17:17	Plik PICKLE	2 KB
	2024-04-20_17-23-27	20.04.2024 17:23	Plik PICKLE	5 KB
	2024-04-20_17-38-38	20.04.2024 17:38	Plik PICKLE	4 KB
	2024-04-20_18-24-30	20.04.2024 18:24	Plik PICKLE	1 KB
	2024-04-20_18-41-05	20.04.2024 18:41	Plik PICKLE	54 KB
	2024-04-20_18-45-46	20.04.2024 18:45	Plik PICKLE	40 KB
	2024-04-22-15-59-29	22.04.2024 15:59	Plik PICKLE	199 KB
	2024-04-22-16-07-40	22.04.2024 16:07	Plik PICKLE	271 KB
	2024-04-22-16-09-46	22.04.2024 16:09	Plik PICKLE	34 KB
	2024-04-22-16-18-57	22.04.2024 16:18	Plik PICKLE	261 KB
	2024-04-22-16-27-20	22.04.2024 16:27	Plik PICKLE	327 KB
	2024-04-22-16-38-12	22.04.2024 16:38	Plik PICKLE	489 KB
	2024-04-22-16-53-55	22.04.2024 16:53	Plik PICKLE	237 KB
	2024-04-22-17-21-32	22.04.2024 17:21	Plik PICKLE	675 KB
	2024-04-22-21-34-20	22.04.2024 21:34	Plik PICKLE	179 KB
	2024-04-22-21-40-58	22.04.2024 21:40	Plik PICKLE	78 KB
	2024-04-22-21-45-49	22.04.2024 21:45	Plik PICKLE	126 KB
	2024-04-22-21-58-14	22.04.2024 21:58	Plik PICKLE	437 KB

W inżynierii atrybutów wykorzystano zarówno informacje o otoczeniu węża, jak i lokalizacji jedzenia, aby stworzyć cechy opisujące stan gry.

Cechy dotyczące przeszkód - 4 atrybuty binarne

Wykorzystano sąsiadujące kwadraty wokół głowy węża, aby określić, czy w każdym kierunku znajduje się przeszkoda. Są to atrybuty binarne, gdzie 1 oznacza obecność przeszkody, a 0 jej brak.

```
# Cechy dotyczące przeszkód
obstacles = {
    'left': any((head_x - 1, head_y) == segment for segment in snake_body) or head_x == 0,
    'right': any((head_x + 1, head_y) == segment for segment in snake_body) or head_x == bounds[0] - 1,
    'up': any((head_x, head_y - 1) == segment for segment in snake_body) or head_y == 0,
    'down': any((head_x, head_y + 1) == segment for segment in snake_body) or head_y == bounds[1] - 1,
}
```

obstacle_left: Czy w lewym sąsiednim kwadracie jest przeszkoda.

obstacle_right: Czy w prawym sąsiednim kwadracie jest przeszkoda.

obstacle_up: Czy w górnym sąsiednim kwadracie jest przeszkoda.

obstacle_down: Czy w dolnym sąsiednim kwadracie jest przeszkoda.

Cechy dotyczące lokalizacji jedzenia - 4 atrybuty binarne

Sprawdzono, w którym kierunku znajduje się jedzenie względem głowy węża, aby ustalić, czy wężowi opłaca się w danym kierunku podążać. Są to atrybuty binarne, gdzie 1 oznacza obecność jedzenia, a 0 jego brak.

```
# Cechy dotyczące lokalizacji jedzenia
food_direction = {
    'food_left': food_position[0] < head_x,
    'food_right': food_position[0] > head_x,
    'food_up': food_position[1] < head_y,
    'food_down': food_position[1] > head_y,
}
```

food_left: Czy jedzenie znajduje się po lewej stronie głowy węża.

food_right: Czy jedzenie znajduje się po prawej stronie głowy węża.

food_up: Czy jedzenie znajduje się powyżej głowy węża.

food_down: Czy jedzenie znajduje się poniżej głowy węża.

Wykorzystanie tych cech pozwala modelowi na podejmowanie decyzji na podstawie lokalizacji przeszkód i jedzenia w otoczeniu węża, co ma istotny wpływ na jego zdolność do wykonywania skutecznych ruchów w grze.

3. Testowanie i ocena modelu

Do oceny jakości modelu na zbiorze danych użyto miary dokładności (ang. Accuracy)

Accuracy przyjmuje wartości z przedziału $[0,1]$ czyli 0% do 100%.

Proszę odpowiedzieć na następujące pytania:

Pytanie 1. Czy i dlaczego występuje przeuczenie lub niedouczenie (ang. overfitting, underfitting).

Niedouczenie występuje, gdy model jest zbyt prosty lub ma zbyt mało możliwości, aby dobrze uchwycić złożoność danych treningowych. Może to być spowodowane wyborem zbyt małej liczby cech (atrybutów), niedostatecznym dopasowaniem parametrów modelu lub niewystarczającą ilością danych treningowych.

Przeuczenie występuje wtedy, gdy parametr accuracy osiąga lepsze wartości dla zbioru danych treningowych niż dla zbioru danych testowych. Wynika z tego, że model za bardzo dopasowuje się do danych treningowych, przez co ma słabą zdolność do generalizacji na nowe, nie widziane wcześniej dane.

Pytanie 2. Jak można zminimalizować ten problem?

Aby zminimalizować problem niedouczenia, można podjąć kilka działań:

Zwiększenie liczby cech (atrybutów): Dodanie bardziej zróżnicowanych i bardziej szczegółowych cech do opisu danych treningowych może pomóc modelowi w lepszym uchwyceniu złożoności problemu. Możemy rozważyć dodanie dodatkowych cech dotyczących różnych aspektów gry, które mogą być istotne dla podejmowania decyzji przez model.

Użycie bardziej zaawansowanego modelu: Wybór bardziej złożonego modelu lub zwiększenie jego pojemności może umożliwić lepsze dopasowanie do danych treningowych. Możemy eksperymentować z bardziej zaawansowanymi modelami maszyny wektorów wspierających (SVM) lub innymi modelami, które są w stanie uchwycić bardziej skomplikowane zależności w danych.

Zwiększenie ilości danych treningowych: Poszerzenie zbioru danych treningowych poprzez zebranie większej ilości danych lub wygenerowanie sztucznych danych może pomóc w lepszym dopasowaniu modelu do danych treningowych. Większa różnorodność danych może umożliwić modelowi lepsze generalizowanie na nowe przypadki.

Optymalizacja hiperparametrów: Dobre dobranie hiperparametrów modelu, takich jak parametr regularyzacji C czy wybór jądra, może pomóc w uzyskaniu lepszej wydajności modelu. Możemy przeprowadzić procedurę doboru hiperparametrów (np. za pomocą walidacji krzyżowej) w celu znalezienia optymalnych ustawień.

Ulepszenie procesu uczenia: Zmiana procesu uczenia poprzez dostosowanie metody optymalizacji, zmniejszenie współczynnika uczenia lub zwiększenie liczby epok może pomóc w poprawie wydajności modelu. Należy eksperymentować z różnymi parametrami uczenia, aby znaleźć najlepszą konfigurację dla naszego problemu.

Zbiór danych testowych	Accuracy
1%	0.36538461538461536
10%	0.46397694524495675
100%	0.4928900845503459

Zbiór danych treningowych	Accuracy
1%	0.4519230769230769
10%	0.46673072303627194
100%	0.4846752498078401

4. Hiperparametr klasyfikatora

Dobrym hiperparametrem do analizy może być parametr regularyzacji C . Jest jednym z najważniejszych hiperparametrów w modelach SVM. Parametr ten kontroluje wagę kar, jakie otrzymują punkty danych za znajdowanie się po niewłaściwej stronie hiperpłaszczyzny separacji

Regularyzacja kontroluje, jak bardzo model dopasowuje się do danych treningowych kosztem złożoności modelu. Im większa wartość C , tym mniej regularyzacji jest stosowane, co może prowadzić do bardziej dopasowanego modelu do danych treningowych, ale może również zwiększyć ryzyko przeuczenia (overfitting). Z kolei mniejsze wartości C powodują większą regularyzację, prowadząc do prostszego modelu, ale mogącego być niedouczonym na bardziej złożonych danych.

Parametr C kontroluje kompromis między dopasowaniem do danych treningowych a złożonością modelu, a jego optymalna wartość zależy od konkretnego zbioru danych i problemu klasyfikacji. W praktyce, dobór optymalnej wartości C często wymaga eksperymentowania z różnymi wartościami i oceniania wyników na zbiorze walidacyjnym lub testowym.

Zbiór danych testowych	
Hiperparametr C	Accuracy
0.01	0.49116064565718676
0.1	0.4928900845503459
1	0.4928900845503459
10	0.4928900845503459
100	0.4928900845503459

Zbiór danych trenigowych	
Hiperparametr C	Accuracy
0.01	0.48395465026902384
0.1	0.4846752498078401
1	0.4846752498078401
10	0.4846752498078401
100	0.4846752498078401

5. BehavioralCloningAgent

Średni wynik modelu na 100 rozgrywek to 0,73

Pytanie. Czy model gra tak samo dobrze jak Państwo, dlaczego tak lub dlaczego nie?

Model nie gra tak samo dobrze, związane to jest z podejściem Behavioral Cloning, które polega na naśladowaniu zachowań ludzi na podstawie zbioru danych treningowych.

Po pierwsze, model może nie być w stanie nauczyć się wszystkich ludzkich zachowań, które są często nieprzewidywalne i losowe, które mogą być trudne do odzwierciedlenia w modelu.

Dane treningowe mogą być niepełne lub niezróżnicowane, co może prowadzić do niedoszacowania różnorodności zachowań ludzkich. Model może nauczyć się tylko ograniczonej liczby scenariuszy i nie będzie w stanie poradzić sobie w nowych sytuacjach.

Model oparty na Behavioral Cloning może mieć tendencję do kopiowania błędów lub niedoskonałości zachowań ludzkich ze zbioru danych treningowych. Jeśli ludzie popełniali błędy lub podejmowali nieoptymalne decyzje, model może je powielać.