Autor:
Kacper Górka

Prowadzący:
mgr inż. Rypeść Grzegorz

RAPORT DO LABORATORIUM NR 4

UCZENIE MASZYNOWE „SNAKE"

WPROWADZENIE DO SZTUCZNEJ INTELIGENCJI

1 Treść ćwiczenia

1.1 Zadania do wykonania

Polecenie:

Należy wygenerować zbiór danych, zaimplementować klasyfikator, wytrenować model na danych i przetestować w grze. Model ma na podstawie stanu gry zwracać akcję, jaką ma wykonać wąż.

Proszę zagrać w załączoną implementację gry *Snake* (*main.py*). Jest ona tak napisana, że po wyjściu z gry zostanie zrzucony na dysk zapis z rozgrywki. Taki zapis składa się z przykładów danych będącymi krotkami (ang. tuple). Krotka zawiera stan gry i podjętą przez Państwa dla niego akcję. Musicie zagrać kilka razy w grę, wygenerować kilka zapisów, a następnie dokonać inżynierii atrybutów, aby przekształcić stany gry w atrybuty, na których wytrenujecie modele. Zadanie jest problemem klasyfikacji - wąż może poruszać się w górę, prawo, dół i lewo. Istnieją zatem 4 klasy. Chciałbym, aby atrybutów skonstruowali Państwo co najmniej 8. Polecam zrobić następujące: 4 binarne (czy w sąsiadującym kwadracie jest przeszkoda) i 4 binarne (czy w danym kierunku jest jedzenie). Złe atrybuty spowodują, że model nie będzie grał dobrze. Proszę krótko opisać stworzone atrybuty w raporcie.
Połączone zapisy rozgrywek przekształcone inżynierią atrybutów stanowią zbiór danych. Gotowy, powien być macierzą typu *numpy.ndarray* lub *torch.tensor*. Wiersze to przykłady danych, a kolumny to atrybuty. Przykładów danych powinno być co najmniej 2000.
Zbiór danych należy podzielić na zbiór treningowy i testowy w proporcji 4:1. Do oceny jakości modelu na zbiorze danych proszę użyć miary dokładności (funkcję można wziąć z biblioteki *scikit-learn*, albo *torchmetrics*). Proszę odpowiedzieć na następujące pytania: czy i dlaczego występuje przeuczenie lub niedouczenie (ang. overfitting, underfitting). Jak można zminimalizować ten problem? Proszę to sprawdzić dla zbioru danych wynoszącego 1%, 10% i 100% całego zbioru danych.

Proszę wybrać jeden hiperparametr klasyfikatora i dla 5 różnych jego wartości przebadać i opisać wpływ na wyniki uzyskiwane na zbiorze treningowym i testowym.

Dysponując wytrenowanym modelem należy dokończyć implementację klasy *BehavioralCloningAgent* reprezentującą agenta. Proszę dla 100 rozgrywek węża sprawdzić jakie wyniki osiąga model. Czy model gra tak samo dobrze jak Państwo? Dlaczego tak lub dlaczego nie?

2 Budowa modelu

2.1 Przygotowanie danych

Zebrane dane (w katalogu data/) po grze poddano obróbce. Usunięto dane, które mogły prowadzić do niepoprawnego wytrenowania modelu. Są to:

- Wiersz tuż przed wyjściem za plansze
- Wiersz tuż przed uderzeniem we własny ogon

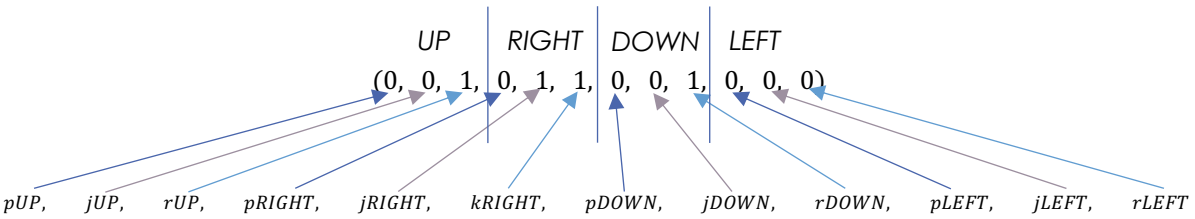
Po zastosowaniu tych poprawek wąż znacznie rzadziej wykonywał niepoprawne ruchy. Ilość wierszy wynosi 2359, co jest większe od zakładanych minimum 2000.

Obróbka danych przeprowadzona została w pliku `dane.py`

2.2 Wybór atrybutów

W modelu stworzono 12 atrybutów. Pierwsze 8 z nich są atrybutami zalecanymi, to znaczy 4 binarne (czy w sąsiadującym kwadracie jest przeszkoda) oraz 4 binarne (czy w danym kierunku północ/wschód/południe/zachód jest jedzenie). Ostatnie dodatkowe 4 binarne określają zwrot, w którym kierunku porusza się aktualnie wąż.

Aby zapewnić dobrą jakość danych jako człowiek grano tak, aby poruszać się w kierunku jedzenia (tzn. suma iloczynów logicznych par kierunku jedzenia i aktualnego poruszania się był równy 1 [*Równanie 1*]).



p – czy w tym kierunku jest przeszkoda?

j – czy w tym kierunku jest jedzenie?

r – czy jest to aktualny kierunek?

$$f(j,r) = j1 \cdot r1 + j2 \cdot r2 + j3 \cdot r3 + j4 \cdot r4$$

Równanie 1

W trakcie gry jako człowiek dążono, aby $f(j,r)$ była równa 1, co w praktyce oznacza, że wąż porusza się w kierunku jedzenia

Gdy wąż był już duży, to było to oczywiście niemożliwe. Wówczas starano się poruszać najkrótszą drogą do jedzenia wzdłuż lub przy samym ciele węża. Pozwoli to na nauczenie modelu, że powodem braku poruszania się w kierunku jedzenia jest fakt, że blokuje nas przeszkoda (aby była ona wykrywalna, to należy poruszać się przy samym ciele, gdyż tylko wtedy jest to wykrywalne przez atrybuty).

2.3 Klasyfikator

W projekcie zastosowano klasyfikator regresji logistycznej. Jest to połączenie problemu regresji (dopasowanie funkcji ciągłej do danych) oraz problemu klasyfikacji, czyli nadaniu dyskretnych wartości (klas) zbiorowi danych z dziedziny. W efekcie na podstawie wyników regresji liniowej wyznacza się prawdopodobieństwo przynależności do danej klasy.

W zadaniu konieczne było zastosowanie rozszerzonego klasyfikatora, który pozwoli na rozróżnienie wielu klas. Typowe klasyfikatory wyznaczają prawdopodobieństwo przynależności do pierwszej klasy, które jest porównywane z progiem 0,5. Jeżeli prawdopodobieństwo nie przekracza tego progu to przykład jest przepisywany do drugiej klasy. W projekcie rozwiązano to inaczej. Wyznaczano bowiem osobno prawdopodobieństwo dla każdej z klas, a następnie wybierano klasę z największym prawdopodobieństwem. W ten sposób klasyfikator obsługuje 4 klasy (UP, RIGHT, DOWN, LEFT).

Klasyfikator znajduje się w pliku classifier.py. Jako porównanie wykorzystano klasyfikator regresji logistycznej z biblioteki *scikit-learn*.

3 Wyniki

3.1 Rezultaty w zależności od zbioru danych

Przeprowadzono uczenia dla 1%, 10% i 100% zbioru danych, a następnie oceniono model.

Tabela 1. Rezultat dla własnego klasyfikatora classifier.py

Wykorzystanie zbioru danych	1%	10%	100%
Zbiór treningowy	0,705	0,704	0,695
Zbiór walidacyjny	0,800	0,702	0,672

Tabela 2. Rezultat dla klasyfikatora *scikit-learn*

Wykorzystanie zbioru danych	1%	10%	100%
Zbiór treningowy	0,889	0,828	0,794
Zbiór walidacyjny	0,800	0,766	0,790

Czy i dlaczego występuje przeuczenie lub niedouczenie (ang. overfitting, underfitting). Jak można zminimalizować ten problem? Proszę to sprawdzić dla zbioru danych wynoszącego 1%, 10% i 100% całego zbioru danych.

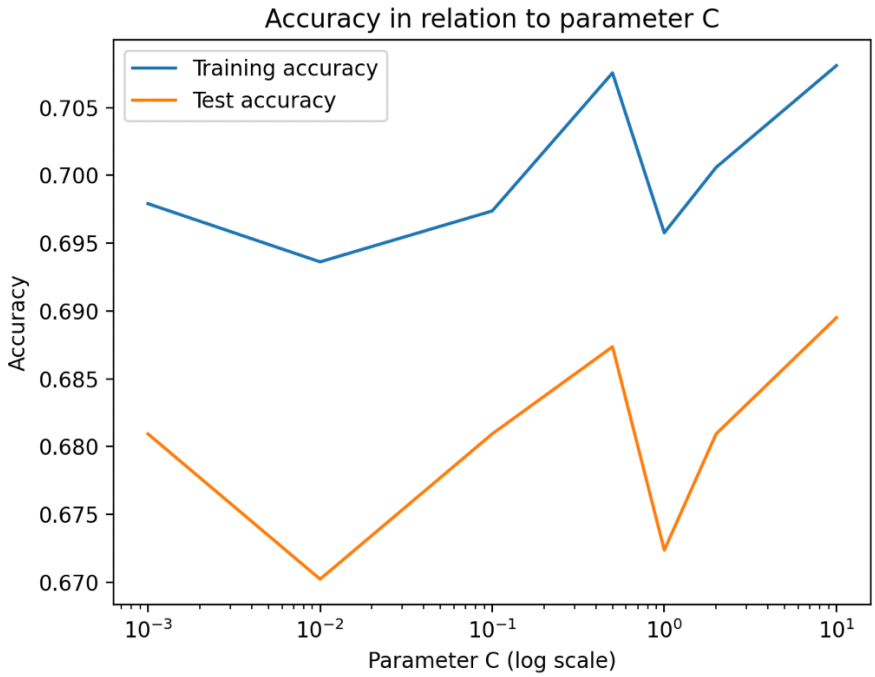
Dla pełnego zbioru danych, dokładność na zbiorze treningowym jest podobna, co znaczy, że nie nastąpiło przeuczenie ani niedouczenie. Model bardzo dobrze dopasowuje się do nowego zbioru danych jakim są dane testowe, co oznacza, że dobrze uogólnia.

Przeuczenie występuje dla 1% i 10% zbioru danych dla klasyfikatora z biblioteki *scikit-learn*, ponieważ zbiór dobrze dopasował się do zbioru treningowego, ale jego skuteczność dla nowego zbioru (czyli danych testowych) jest niższa.

3.2 Wpływ hiperparametru C klasyfikatora na dokładność modelu

3.2.1 KLASYFIKATOR REGRESJI LOGISTYCZNEJ Z PLIKU CLASSIFIER.PY

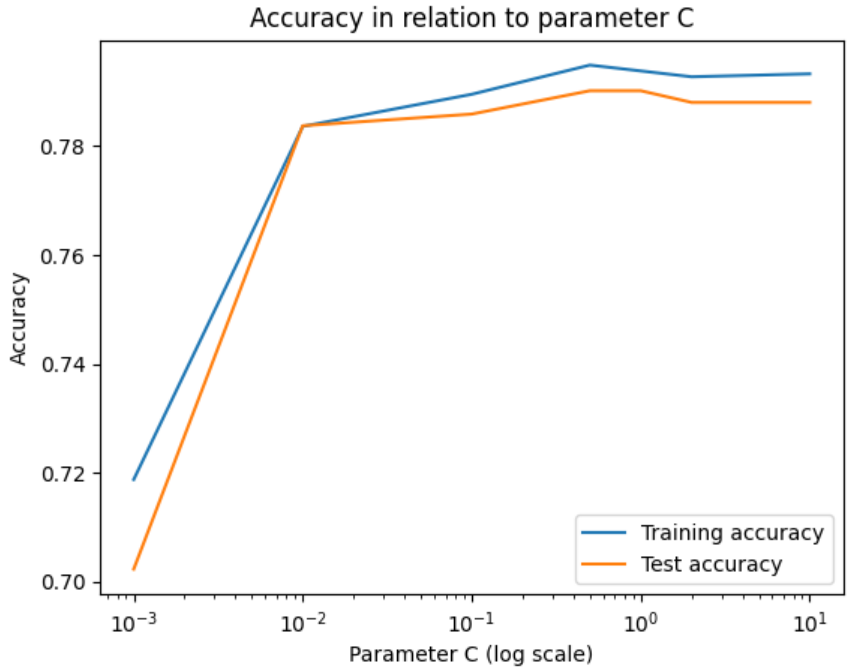
Hiperparametr C określa w jakim stopniu traktujemy zbyt duże wartości wag dla każdej z cech. Im większa wartość C, tym mniejsza kara za zbyt duże wagi. W efekcie model może być podatny na zbyt duże dopasowanie się do danych treningowych. Nie oznacza to że większe wartości C są błędne, gdyż mniejsze mimo że prowadzą do bardziej ogólnego modelu, to mogą również prowadzić do utraty ważnych informacji.



Najlepszy wynik osiągnięto dla hiperparametru C równego 10.

Parametr C	0,001	0,01	0,1	0,5	1	2	10
Zbiór treningowy	0,6979	0,6936	0,6973	0,7075	0,6957	0,7005	0,7080
Zbiór walidacyjny	0,6809	0,6702	0,6809	0,6873	0,6723	0,6809	0,6895

3.2.2 KLASYFIKATOR REGRESJI LOGISTYCZNEJ Z PAKIETU *SCIKIT-LEARN*



Najlepszy wynik uzyskano dla hiperparametru C równego 1, gdyż wówczas różnica między dokładnością na zbiorze treningowym, a walidacyjnym jest najmniejsza.

Parametr C	000,1	0,01	0,1	0,5	1	2	10
Zbiór treningowy	0,7188	0,7836	0,7895	0,7949	0,7938	0,7927	0,7933
Zbiór walidacyjny	0,7023	0,7837	0,7859	0,7901	0,7901	0,7880	0,7880

3.3 Rezultaty osiągane przez model

Wyniki dla modelu w 100 rozgrywkach:

Scores: [0, 9, 9, 8, 11, 11, 8, 11, 14, 10, 10, 11, 7, 6, 5, 11, 14, 8, 3, 9, 13, 12, 5, 6, 17, 16, 3, 7, 15, 9, 12, 9, 7, 5, 17, 4, 6, 2, 13, 13, 13, 12, 6, 6, 8, 6, 14, 8, 5, 10, 12, 6, 9, 4, 7, 8, 7, 16, 5, 10, 12, 5, 6, 0, 12, 14, 8, 7, 8, 6, 9, 11, 7, 2, 8, 6, 12, 1, 4, 13, 8, 9, 4, 7, 10, 3, 18, 2, 7, 15, 2, 12, 0, 8, 6, 10, 15, 11, 4, 1]

Średnia: 8.41

Wyniki dla modelu z pakietu ***scikit-learn*** w 100 rozgrywkach:

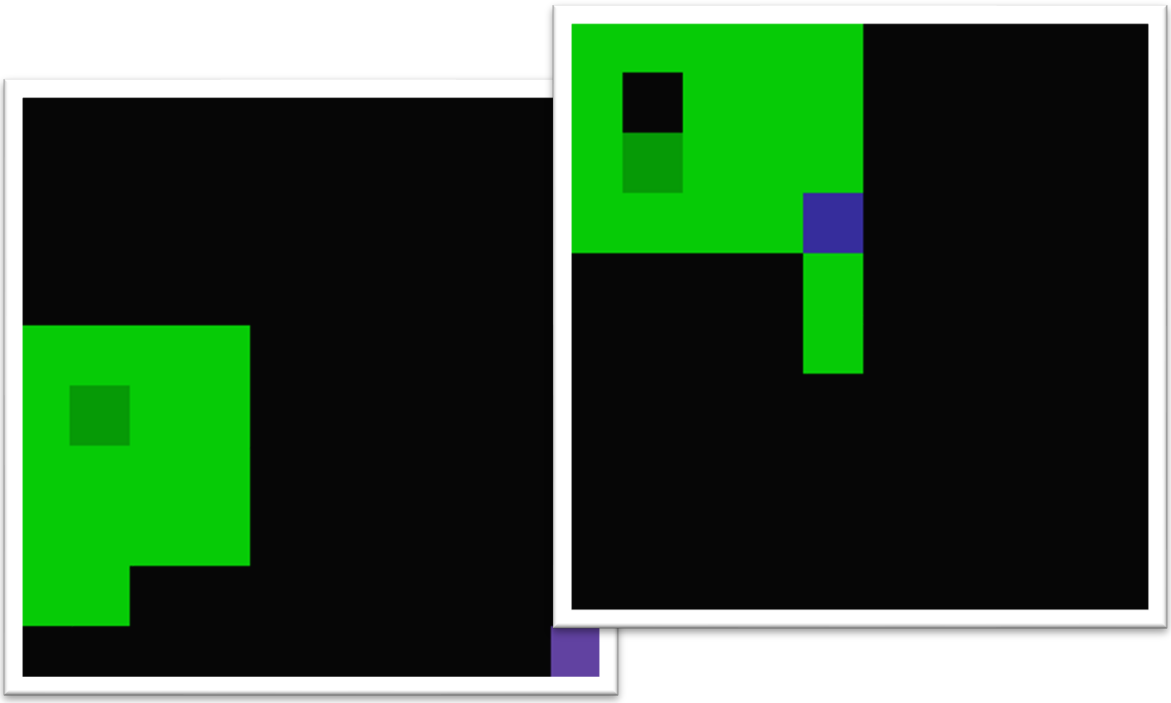
Scores: [16, 13, 7, 16, 15, 23, 12, 28, 7, 5, 18, 21, 11, 10, 12, 17, 11, 13, 17, 13, 16, 6, 20, 18, 24, 18, 13, 15, 16, 16, 19, 17, 5, 15, 13, 8, 10, 13, 13, 9, 20, 12, 21, 18, 14, 15, 14, 21, 14, 17, 9, 31, 6, 14, 17, 19, 14, 13, 7, 22, 19, 5, 20, 8, 8, 23, 15, 14, 9, 16, 16, 13, 10, 10, 9, 11, 9, 8, 11, 16, 11, 12, 29, 10, 19, 13, 22, 7, 11, 10, 19, 25, 8, 15, 13, 19, 7, 21, 10, 16]

Średnia: 14.34

Model z pakietu ***scikit-learn*** gra o 70% lepiej od modelu zaimplementowanego w pliku *classifier.py*

Czy model gra tak samo dobrze jak Państwo? Dlaczego tak lub dlaczego nie?

Model gra zadowalająco dobrze, jednak znacznie częściej od człowieka popada w zakleszczenia.



Powodem zakleszczeń jest to, że wąż nie „widzi" dalej niż jego najbliższe otoczenie. Jednym z rozwiązań byłoby dodanie większej ilości atrybutów dotyczących przeszkód w dalszych polach.

Jego zaletą jest wykluczenie sytuacji w których człowiek nie zdąży zareagować lub popełni „głupi" błąd.