Wydział Matematyki i Informatyki UAM
Przedmiot: Sztuczna Inteligencja
2018/2019
Raport

Czaplicka Anna

Winiarski Bartłomiej

1. Metoda uczenia. Instalacja programu.

Do uczenia maszynowego zastosowano system Vowpal Wabbit. Po nieudanej próbie zbudowania programu ze źródeł na systemie Windows (raz za pomocą CMake i vcpkg, a drugi raz za pomocą Visual Studio 2017) program zainstalowany został z repozytorium pakietów na systemie Ubuntu 18.04. Komenda użyta do instalacji: apt-get install vowpal-wabbit.

2. Przygotowanie danych testowych.

Jako podstawę uczenia wykorzystano algorytm DFS, którego rozwiązania posłużyły do wygenerowania danych testowych. Algorytm DFS zastosowano na dwudziestu ośmiu różnych planszach o różnym stopniu trudności (tj. o różnej liczbie stolików, zamówień, pól, po których może poruszać się agent).

Ideą zastosowania tej metody uczenia jest zapisywanie najbliższego otoczenia agenta (*environment*) – kelnera, wraz z podjętą decyzją o wykonaniu ruchu dla danego stanu planszy. W tym celu zaimplementowano metodę `get_waiter_environment` oraz `object_at`, która dla parametrów *width* i *height* równych 3 zapisywała obszar 49 pól (7 na 7 pól), z umieszczonym pośrodku kelnerem w zmiennej `state`. Poniżej zamieszczono kod źródłowy obu metod:

```
def get_waiter_environment(self, width, height):
    env = ""
    for i in range(2 * height + 1):
        for j in range(2 * width + 1):
            obj_pos_x = self.waiter.x - height + i
            obj_pos_y = self.waiter.y - width + j
            obj = self.object_at(obj_pos_x, obj_pos_y)
            env += f"{(-1)*(height - j)}_{width - i}:{obj.num() if obj is
not None else -1} "
```

```
    return env

def object_at(self, x, y):
    if x >= self.board_size or x < 0 or y >= self.board_size or y < 0:
        return None
```

Rysunek 1. prezentuje obszar zapisywanego środowiska, dla planszy o pliku `board1.txt`:

```
10
7 3
1
0 rosol
F F F F F F F F F F
F F F F F F F F F F
F F F F F F F F F F
F T C C C C C C K F
F F F F F F F F F F
F F F F F F F F F F
F F F F F F F F F F
F F F F F F F F F F
F F F F F F F F F F
F F F F F F F F F F
```

Rysunek 1. Reprezentacja środowiska kelnera.

-3_ 3 **F_S**	-2_3 **F_S**	-1_3 **F_S**	0_3 **F_S**	1_3 **F_S**	2_3 **F_S**	3_3 **None**
-3_2 **F_S**	-2_2 **F_S**	-1_2 **F_S**	0_2 **F_S**	1_2 **F_S**	2_2 **F_S**	3 _2 **None**
-3_1 **F_S**	-2_1 **F_S**	-1_1 **F_S**	0_1 **F_S**	1_1 **F_S**	2_1 **F_S**	3_1 **None**
-3_0 **CARPET**	-2_0 **CARPET**	-1_0 **CARPET**	0_0 **CARPET**	1_0 **KITCHEN**	2_0 **F_S**	3_0 **None**
-3_-1 **F_S**	-2_ -1 **F_S**	-1_-1 **F_S**	0_-1 **F_S**	1_-1 **F_S**	2_ -1 **F_S**	3_-1 **None**
-3_-2 **F_S**	-2_-2 **F_S**	-1_-2 **F_S**	0_-2 **F_S**	1_-2 **F_S**	2_ -2 **F_S**	3_-2 **None**
-3_-3 **F_S**	-2_-3 **F_S**	-1_-3 **F_S**	0_-3 **F_S**	1_-3 **F_S**	2_-3 **F_S**	3_-3 **None**

Kelner ustawiony jest zawsze na polu 0_0. Informacja ta przekazywana jest w celu nauczania

reguły o poruszaniu się agenta wyłącznie po polu Carpet. F_S na rysunku oznacza wolne pole (Free Space), a None pole, które znajduje się poza planszą.

Możliwym polom dla planszy odpowiadają cyfry zaprezentowane w tabeli 1.

Tabela 1. Pola i ich wartości liczbowe.

Pole	Wartość
Brak pola (None)	-1
Carpet	0
Free Space	1
Kitchen	2
Table	3

Dla powyższego przykładu, w stanie początkowym (rysunek 1) środowisko przybiera postać:

```
|Environment -3_3:1 -2_3:1 -1_3:1 0_3:1 1_3:1 2_3:1 3_3:-1 -3_2:1 -2_2:1 -
1_2:1 0_2:1 1_2:1 2_2:1 3_2:-1 -3_1:1 -2_1:1 -1_1:1 0_1:1 1_1:1 2_1:1 3_1:-
1 -3_0:0 -2_0:0 -1_0:0 0_0:0 1_0:2 2_0:1 3_0:-1 -3_-1:1 -2_-1:1 -1_-1:1 0_-
1:1 1_-1:1 2_-1:1 3_-1:-1 -3_-2:1 -2_-2:1 -1_-2:1 0_-2:1 1_-2:1 2_-2:1 3_-
2:-1 -3_-3:1 -2_-3:1 -1_-3:1 0_-3:1 1_-3:1 2_-3:1 3_-3:-1
```

Dodatkowo oprócz środowiska dla stanu planszy zapisywano także w zmiennej `state` możliwe do wykonania na danym polu ruchy oraz liczbę zamówień trzymanych przez kelnera. Poniżej znajdują się kod prezentujący sposób zapisywania stanu:

```
def __repr__(self):
    s = "|Environment "
    s += self.get_waiter_environment(3, 3)

    return s

def __repr__(self):
    return f"|Waiter ordersCount:{len(self.heldOrders)}"

state += repr(board)

for i, movee in enumerate(board.get_possible_waiter_moves(previous_move)):
    if str(movee.type.value) not in state:
        state += f"{movee.type.value} "

new_solution = list(current_solution)
new_solution.append([previous_move, state])

state += repr(board.waiter)
```

Ruchom także nadano wartości liczbowe, które odpowiadają multiklasom w Vowpallu. Ruchy i ich wartości prezentuje tabela 2.

Tabela 2. Możliwe ruchy i ich wartości.

Ruch	Wartość
1	UP
2	DOWN
3	RIGHT
4	LEFT
5	TAKE ORER
6	SERVE ORDER

W zwracanym przez algorytm DFS rozwiązaniu otrzymywano stan planszy, wszystkie możliwe do wykonania wówczas ruchy oraz ilość dań kelnera oraz podjętą decyzję. Na tej podstawie dla każdego wykonanego w rozwiązaniu ruchu, dokonywano zapisu rekordu do pliku `testing_data.txt`. Dla powyższego przykładu, pojedynczym rekordem dla pierwszego stanu planszy jest:

```
5 'board1.txt |Possible_moves 4 5 |Waiter ordersCount:0 |Environment -3_3:1
-2_3:1 -1_3:1 0_3:1 1_3:1 2_3:1 3_3:-1 -3_2:1 -2_2:1 -1_2:1 0_2:1 1_2:1
2_2:1 3_2:-1 -3_1:1 -2_1:1 -1_1:1 0_1:1 1_1:1 2_1:1 3_1:-1 -3_0:0 -2_0:0 -
1_0:0 0_0:0 1_0:2 2_0:1 3_0:-1 -3_-1:1 -2_-1:1 -1_-1:1 0_-1:1 1_-1:1 2_-1:1
3_-1:-1 -3_-2:1 -2_-2:1 -1_-2:1 0_-2:1 1_-2:1 2_-2:1 3_-2:-1 -3_-3:1 -2_-
3:1 -1_-3:1 0_-3:1 1_-3:1 2_-3:1 3_-3:-1
```

Gdzie z możliwych ruchów: LEFT (4), TAKE ORDER (5), została podjęta decyzja o wykonaniu ruchu 5. Dodatkową informacją uczącą jest ilość posiadanych przez kelnera dań. Dodano także trzy etykiety: Possible_moves, Waiter ordersCount oraz Environment.

Podsumowując, na podstawie rozwiązania z każdej z planszy uczących, na której zastosowano algorytm DFS, zapisano odpowiednie rekordy do pliku `testing_data.txt`. Dla 28 plansz uczących uzyskano 1397 rekordów. Rekordy zapisywane były zgodne z dokumentacją Vowpalla.

3. Generowanie i uczenie modelu.

Do wygenerowania modelu uczącego i przewidującego ruchy (finalnego regresora) użyliśmy metody Multiclass – One Against All (-ooa) żeby odpowiednio sklasyfikować możliwe w danym stanie ruchy. Niestety niemożliwe by to było w standardowej klasyfikacji vowpala (binarnej). Możliwych ruchów w każdym stanie było maksymalnie 6, więc jako argument opisujący ilość klas podaliśmy liczbę 6. Do uczenia powstającego modelu użyliśmy 20 przejść. Model z każdym przejściem używał pliku cache z poprzednich przejść (argument

–c). Pozwoliło to nam osiągnąć prawie idealną konwergencje (0.015503). W drodze obserwacji, najlepszym parametrem opisującym moc zmniejszania prędkości nauczania (power on the learning rate decay) został wybrany współczynnik 0.2 (z domyślnego 0.5). Wyłączone zostało zmniejszanie ilości dostępnych próbek po każdym przejściu za pomocą opcji –holdout|_off. Wygenerowany model zapisano do pliku waiter.model

Podsumowując, komenda użyta do wygenerowania modelu wyglądała następująco:

vw --oaa 6 testing_data.txt -c --power_t 0.2 --passes 20 --holdout_off -f waiter.model

Output z vowpala:
final_regressor = waiter.model

Num weight bits = 18

learning rate = 0.5

initial_t = 0

power_t = 0.2

decay_learning_rate = 1

creating cache_file = testing_data.txt.cache

Reading datafile = testing_data.txt

num sources = 1

average loss	since last	example counter	example weight	current label	current predict	current features
1.000000	1.000000	1	1.0	5	1	16
1.000000	1.000000	2	2.0	4	5	15
0.500000	0.000000	4	4.0	4	4	15
0.500000	0.500000	8	8.0	4	6	16
0.500000	0.500000	16	16.0	5	6	15
0.437500	0.375000	32	32.0	6	1	15
0.390625	0.343750	64	64.0	6	2	15
0.296875	0.203125	128	128.0	3	3	15
0.167969	0.039062	256	256.0	3	3	14

0.091797 0.015625 512 512.0 2 2 13

0.076172 0.060547 1024 1024.0 4 4 14

0.059570 0.042969 2048 2048.0 2 2 12

0.049561 0.039551 4096 4096.0 2 2 13

0.035889 0.022217 8192 8192.0 1 1 14

0.025696 0.015503 16384 16384.0 1 1 12

finished run

number of examples per pass = 1411

passes used = 20

weighted example sum = 28220.000000

weighted label sum = 0.000000

average loss = 0.020695

total feature number = 347780

4. Komunikacja z modelem i odczytywanie predykcji

Aby na bieżąco otrzymywać predykcje od vowpala za pomocą naszego wygenerowanego modelu, został on uruchomiony w trybie daemon. Pozwoliło to na wysyłanie i odbieranie predykcji za pomocą socketu tcp. Vowpal nasłuchiwał na porcie 26542 Użyta komenda:
vw -i waiter.model --daemon --quiet --port 26542

Następnym krokiem było połączenie się z serverem w naszej aplikacji restauracji.

```
PORT = 26542
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("localhost", PORT))
```

I dopóki plansza nie została wykonana, wysyłać obecny stan kelnera, środowiska i możliwych ruchów do vowpala i otrzymywać predykcje. Vowpal zwracał liczbę z przedziału 1-6, opisującą jeden z ruchów do wykonania. Potem konwertowaliśmy tą liczbę na odpowiedni ruch i go wykonywaliśmy na planszy. Warto zaznaczyć że dane w pliku uczącym opisujące możliwe ruchy były przesunięte o + 1 gdyż vowpal nie przyjmuje wartości ujemnych w klasach. Zatem przy odczytywaniu wartości ruchu, trzeba było odjąć jeden.

```
previous_move = Move(MoveType.EMPTY_MOVE)
while not board.all_orders_served():
    possible_moves = board.get_possible_waiter_moves(previous_move)
    state = ""
    for i, movee in enumerate(board.get_possible_waiter_moves(previous_move)):
        if str(movee.type.value + 1) not in state:
            state += f"{movee.type.value + 1} "
    state + repr(board.waiter)
    state += repr(board)
    state += "\n"
    s.send(bytes(f"{state}", 'utf-8'))
    data = s.recv(1024).strip()

    data = int(data) - 1
    move = [move for move in possible_moves if move.type.value == data][0]
    sprites = board.to_sprite_group(WINDOW_WIDTH, WINDOW_HEIGHT)
    pygame.display.set_caption(f"Restaurant - vowpal doing {move_counter} move: {move}")

    board.do(move)
    previous_move = move
    time.sleep(STEP_TIME)
    DISPLAYSURF.blit(background_image, (0, 0))
    sprites.draw(DISPLAYSURF)
    pygame.display.flip()
    fpsClock.tick(FPS)
```

5. Konkluzje i wynik.

Niestety, pomimo przygotowania takiej ilości danych do testów, vowpal pomimo wysyłania do niego tylko możliwych ruchów, zwracał ruchy niemożliwe. Szczególnie nie odstawiał dań do stolika, lub nie brał dań z kuchni. Nauczył się natomiast chodzić tylko po dywanie i do stolików.