Dokumentacja projektu symulacji środowiska oraz jego agenta: podsumowanie projektu

Zespół projektowy nr. 2:
Członkowie:
Marcin Drzewiczak, Michał Kubiak, Julia Maria May, Przemysław Owczarczyk
Kierownik zespołu:
Adam Lewicki

1 Podsumowanie projektu

Poniżej przedstawione zostały raporty końcowe, sporządzone przez członków projektu oraz usystematyzowane przez kierownika zespołu. Opisane zostały trzy metody rozwiązań symulacji za pomocą uczenia maszynowego oraz testy projektu.

2 Vowpal Wabbit

2.1 Implementacja

Do zaimplemenowania Vowpal Wabbit został wykorzystany pakiet vw zainstalowany na wybranej maszynie. Obsługa pakietu jest prowadzona przez emulację konsoli, co pozwala na uruchomienie algorytmu na dowolnej maszynie zawierającej vw.

2.2 Model

Do trenowania systemu uczącego Vowpal Wabbit użyto 4572 przykładów uczących. Każdy przykład składa się z etykiety (liczby z zakresu od 1 do 4) określającej właściwy dla danego stanu otoczenia agenta kierunek ruchu: w górę, w dół, w prawo lub w lewo, uzyskany z wykorzystaniem algorytmu Depth First Search, uruchomionego dla danego fragmentu planszy (sąsiedztwa agenta o rozmiarze 5x5 pól). Cechy danego przykładu uczącego to wymienione po kolei wszystkie pola danego fragmentu planszy z przypisanymi im liczbami całkowitymi, oznaczającymi typ danego pola, czyli znajdujący się na nim obiekt taki jak stół, kuchnia, ściana lub jego brak, czyli puste pole, po którym agent może się poruszać.

Ostateczna wersja modelu systemu klasyfikującego Vowpal Wabbit została wytrenowana na 25 przebiegach przez zestaw danych uczących, z wykorzystaniem pliku cache do przyspieszenia procesu nauki.

2.3 Próby aktywacji

Wytrenowany model nie radzi sobie ze znajdowaniem odpowiedniego ruchu w danej sytuacji oraz wskazuje, że właściwym wyborem byłoby wejście na pole ze ścianą lub poza obszar planszy. Pomimo zastosowania różnorakich sposobów, mogących potencjalnie poprawić jego działanie, takich jak: zwiększenie liczby przebiegów przez zestaw danych uczących (nawet do 10000 przebiegów), zwiększenie oraz zmniejszenie learning rate (minimalne 0.0001, maksymalne 100), czy zastosowanie opcji holdout off, aby uzyskać lepsze dopasowanie do danych uczących kosztem przetrenowania modelu.

Mimo wszystkich zabiegów, average loss dla trenowania modelu na danych uczących nigdy nie uzyskało wartości mniejszej od 1, co w tym przypadku uzyskało przełożenie na niepoprawne przewidywania ruchu, nawet dla danych uczących. Świadczy to o dużym obciążeniu modelu, który nie wykrył żadnych zależności w danych, przez co nie był w stanie generalizować przewidywań na nieznane wcześniej stany planszy.

3 Algorytm SVM

3.1 Implementacja

Do zaimplemenowania SVM został wykorzystany moduł "Scikit", który oferuje pełną implementację tej metody.

3.2 Model

Ilość wykorzystanych próbek wynosi dokładnie 4572. Tyle próbek zostało mi dostarczone przez ooby zajmujące się utowrzeniem modelu danych. Odpowiednio dla ruchów:

- W -1221
- S -1061
- A -1088
- D -1202

Po sprawdzeniu modelu autorzy doszli do wniosku, że ilość danych jest wystarczająca, oraz że dostarczone dane były zbalansowane co do typu ruchu i nie nastąpił problem nadmiernego dopasowania.

3.3 Próby aktywacji

Pierwsza próba uruchomienia algorytmu zakończyła się niepowodzeniem. SVM cały czas wybierał drogę w górę (zwracał warość 'W'). Problem ten jednak został rozwiązany.

Druga próba uruchomienia zmieniła wartości gamma i scale w konstruktorze obiektu dostarczonego przez Scikit learn gdzie według dokumentacji:

> C: penalty parameter C of the error term.
>
> gamma: kernel coefficient for 'rbf', 'poly' and 'sigmoid'
>
> scale: if gamma = 'scale' is passed then it uses $(nfeatures * X.var())^{-1}$ as value of gamma

Zostały wprowadzone następujące modyfikacje parametrów:

> C = 100
>
> gamma = 'scale'

Dzięki wskazanym modyfikacjom nastąpiła znacząca poprawa w działaniu algorytmu, który dobiera teraz ścieżkę wystarczająco prawidłowo.

Zauważone błędy:

1. Kelner po aktywowaniu stolika nadal próbuje go aktywować, mimo wyraźnego wskazania w modelu różnicy między stolikiem aktywnym a nieaktywnym (model wskazuje na przyciąganie stolika aktywnego i odpychanie nieaktywnego). Interwencja kontrolera odsuwająca kelnera od stolika najczęściej naprawia ten błąd;
2. Petla - gdy w otoczeniu kelnera nie znajdują się żadne obiekty, SVM zwraca pary przeciwnych ruchów (góra-dół lub prawo-lewo), ze względu na jednolitość otoczenia. Ręczna korekta (wyprowadzenie kelnera z korytarza) naprawia ten błąd. Został również wprowadzony "bezpiecznik" opisany w dalszej części dokumentu.

4 Algorytm drzewa decyzyjnego

4.1 Implementacja

Źródło: https://scikit-learn.org/stable/modules/tree.html

4.2 Model

Algorytm drzewa decyzyjnego został wyuczony na 4528 rekordach danych, tzn. różnych kombinacjach ruchów i położeń obiektów w sąsiedztwie kelnera. Jedna partia danych uczących to pole 5x5, zgodnie z przyjętym wewnątrz projektu standardem.

4.2.1 Próby aktywacji

Algorytm drzewa decyzyjnego radzi sobie wystarczająco, aby uznać go za prawidłowo zrealizowany. Nie uzyskano informacji o dodatkowych parametrach zwiększających precyzję rozwiązań.

Algorytm może jednak trafić w pętlę przeciwnych ruchów, jeśli zostanie umieszczony w korytarzu niezawierającym punktów odniesienia - stołów lub kuchni. Jest to zaskakujacy rezultat, biorąc pod uwagę reprezentację korytarzy w modelu. Problem ten został rozwiązany przez wprowadzenie "bezpiecznika", wspólnego dla wszystkich algorytmów uczenia - po napotkaniu zapętlonej sekwencji ruchów, następny ruch jest wybierany losowo. Ponadto projekt wspiera użycie strzałek do emulowania ruchu kelnera, co pozwala na wyprowadzenie go z pętli przez operatora symulacji.

Z przeprowadzonych doświadczeń z algorytmem drzewa decyzyjnego wynika, że algorytm nie daje prawidłowych wyników dla stosunkowo prostych ścieżek, gdy są one podobne do innych ścieżek zawartych w modelu, lecz z przeciwnym rozwiązaniem.

5 Algorytm regresji logistycznej

5.1 Implementacja

Algorytm Logistic Regression z parametrami solver='lbfgs', multi_class='multinomial', c=100.0

5.2 Model

Projekt zawiera 4526 rekordów danych, każdy z nich złożony z położenia obiektów oraz ruchu jaki jest wykonywany przez kelnera. Jeden rekord to macierz wielkości 5x5.

5.2.1 Próby aktywacji

Algorytm regresji logistycznej nie działa wystarczająco poprawnie. W większości doświadczeń wykonywał ruchy zakazane (wejścia na ścianę lub poza obszar symulacji) lub pętle prawo-lewo lub góra-dół.

Na podstawie dostępnych źródeł wyszło na jaw, że algorytm regresji logistycznej bardzo podatny na "overfitting" - przy dużym zbiorze danych i zmiennych niezależnych model staje się przeładowany, co zmniejsza jego możliwości uogólnienia danych do których model jest dopasowany. Wielokrotne próby odkrycia właściwej kombinacji współczynników, algorytm nadal nie uzyskuje pożądanych rezultatów.

6 Testy projektu

Testy projektu zostały przeprowadzone przez Przemysława Owczarczyka.

Testowane procedury:

1. move
2. calculate_dfs_path
3. get_dfs_path
4. caluculate_bfs_path
5. get_bfs_path
6. calculate_bestfs_path
7. get_bestfs_path

Testy modułu move:

1. Tester wykonał serię wyświetleń podczas działania algorytmów i sprawdzał, czy wyniki były zgodne z przewidywaniami;
2. Sprawdzenie wykonania otrzymanego ruchu i aktualizacji pozycji kelnera w prawidłowy sposób;
3. Sprawdzenie, czy ruch został usunięty ze ścieżki ruchów.

Testy pozostałych modułów odbywały się w tym samym schemacie wyświetlenia wyniku, sprawdzenia prognozowanego wyjścia, wykonania procedury i aktualizacji pozycji oraz wpływu procedury na stan zmiennych wewnętrznych kelnera.

Tester wykonał również osobiście zaprojektowane testy kontrolne agenta. Opis testu:

1. stworzenie listy testowej oraz dodanie do funkcji ruchu zapisu ruchów w konwencji WASD do listy testowej podczas wykonania algorytmu. Spowodowało to co krok kelnera (wywołanym emulacją naciśnięcia klawisza SPACE) dodanie ruchu do listy testowej.
2. zapis listy testowej do pliku;
3. dla każdego algorytmu tester ręcznie uzupełnił przewidywaną listę kroków w pliku ./documentation/test.txt.
4. napisanie skryptu ./scripts/test.py porównującego 3 listy ruchów z 3 różnych algorytmów z listami ruchów napisanymi samodzielnie.

Test osiągnął oczekiwaną wartość 100% zgodności otrzymanych ruchów w stosunku do przewidywań testera.

Inne moduły nie zostały poddane testom, jako że ich skuteczność została wykazana pośrednio podczas rozwiązywania testów ruchu.

Zauważone błędy projektu:

- zaimplementowane algorytmy nie działają, jeżeli symulacja zawiera pętle. Zaproponowano w związku z tym rozwiązanie w postaci atrybutu "odwiedzony" lub "nieodwiedzony" w wierzchołkach grafu.

  Takie rozwiązanie pozwala na przeliczenie trasy, jednak nie jest ona najkrótsza. Została podjęta decyzja o zaprzestaniu inwestycji środków w to zagadnienie, gdyż nie powoduje ono obniżenia wartości rzeczywistej projektu, którego celem było pokazanie możliwości rozwiązania symulacji w niekoniecznie optymalny sposób.