WSI: Laboratorium

Ćwiczenie: Regresja i klasyfikacja

Paweł Skierś, 310895

Semestr zimowy 2021/22

ZADANIE

Celem ćwiczenia jest implementacja drzew decyzyjnych tworzonych algorytmem ID3 z ograniczeniem maksymalnej głębokości drzewa. Następnie należy wykorzystać stworzony algorytm do stworzenia i zbadania jakości klasyfikatorów dla zbioru danych iris.

DOKUMENTACJA ROZWIĄZANIA

Program napisany na potrzebę tego ćwiczenia został podzielony na 4 pliki: data_loader.py, decision_tree.py, main.py i node.py. W dalszej części zostanie omówione za co odpowiadają poszczególne pliki oraz zastosowane w nich rozwiązania.

data_loader.py

Plik zawiera funkcję pozwalającą na załadowanie zbioru danych iris i wyciągnięcie z niego liczących się danych. Funkcja nazwana jest get_formated_data. Funkcja ta przyjmuje za argument liczbę klas dyskretnych dla argumentu. Ładuje ona z pomocą biblioteki sklearn zbiór iris, wyciąga z niego ilość klas i atrybutów, dyskretyzuje dane, a następnie dzieli je na 3 zbiory: treningowy, walidacyjny i testowy. Na koniec funkcja zwraca wszystkie wymienione powyżej rzeczy.

decision_tree.py

Plik ten zawiera klasę reprezentującą drzewo decyzyjne. Jej atrybutami są głębokość maksymalna drzewa, korzeń drzewa, klasy zbioru, atrybutu zbioru, zbiór danych treningowych, zbiór danych walidacyjnych, zbiór poprawnych klas dla zbioru treningowego, zbiór poprawnych klas dla zbioru walidacyjnego oraz listę liści drzewa. Klasa udostępnia również metody:

- train() – metoda wołająca dla każdego liścia metodę tego liścia odpowiedzialną za implementacje algorytmu ID3, a następnie uaktualniająca listę liści drzewa
- fit() – metoda pobierająca potrzebne do treningu informacje i opcjonalnie zbiór walidacyjny. Następnie metoda trenuje na podanych danych klasyfikator wołając odpowiednią liczbę razy metodę train() oraz po każdym takim zawołaniu wypisuje dokładności modelu na zbiorze treningowym i ewentualnie walidacyjnym.
- evaluate() – metoda ta dla podanych jej zbiorze i poprawnych klasach oblicza dokładność drzewa.
- predict() – metoda zwracająca predykcje modelu dla podanych tej funkcji danych.

node.py

Plik ten zawiera klasę reprezentującą wierzchołek drzewa decyzyjnego. Jej atrybutami są dostępne w wierzchołku atrybuty, dostępne klasy, atrybut i jego wartość zapamiętana w tym wierzchołku, dzieci wierzchołka, najlepsza klasa dla wierzchołka, zbiór treningowy w wierzchołku oraz atrybut mówiący czy wierzchołek jest liściem. Klasa udostępnia również metody:

- predict() – metoda ta dla danej pojedynczej danej zwraca najlepszą klasę wierzchołka jeśli wierzchołek jest liściem, a jeśli nie to zwraca wartość tej metody wywołanej dla dziecka tego wierzchołka, którego atrybut i jego wartość jest taka jak ta w zdanej funkcji danej
- induce() – metoda wykonująca algorytm ID3, a następnie dopisująca stworzone nowe wierzchołki do listy dzieci wierzchołka
- get_best_attribute() – metoda zwraca klasę najczęściej pojawiającą się w danych w wierzchołku
- infGain() – metoda jest implementacją funkcji wzrostu informacji. Zwraca ona wzrost informacji dla podanego atrybutu

main.py

Jest to plik główny programu. Znajduje się w nim funkcja main() pozwalająca na wczytanie maksymalnej głębokości drzewa i ilości klas dyskretnych dla atrybutu. Następnie pobierane są zbiory walidacyjne, treningowe i testowe, tworzone jest drzewo decyzyjne, które następnie jest trenowane na pobranym zbiorze treningowym. Na koniec następuje ewaluacja modelu na zbiorze testowym i wypisanie jej na standardowe wyjście.

BADANIE JAKOŚCI KLASYFIKATORÓW DLA ZBIORU DANYCH IRIS

W celu zbadania jakości tworzonych klasyfikatorów postanowiono zbadać wpływ maksymalnej głębokości tworzonego drzewa oraz wpływ ilości tworzonych dyskretnych klas dla pojedynczego atrybutu. Do oceny wpływów posłużyła dokładność wytworzonych drzew.

Wpływ ilości tworzonych dyskretnych klas

Stworzono drzewa decyzyjne dla kilku wybranych ilości klas dyskretnych. Poniżej widoczne są uzyskane dokładności stworzonych klasyfikatorów.

```
Enter max tree depth: 4
Enter discretizer bins number: 2
Max depth:  0 ;    Train accuracy:  0.34615384615384615 ;    Validation accuracy:  0.21739130434782608
Max depth:  1 ;    Train accuracy:  0.6730769230769231 ;    Validation accuracy:  0.6956521739130435
Max depth:  2 ;    Train accuracy:  0.7115384615384616 ;    Validation accuracy:  0.782608695652174
Max depth:  3 ;    Train accuracy:  0.7403846153846154 ;    Validation accuracy:  0.782608695652174
Max depth:  4 ;    Train accuracy:  0.7403846153846154 ;    Validation accuracy:  0.782608695652174

Test accuracy:  0.782608695652174
```

```
Enter max tree depth: 4
Enter discretizer bins number: 3
Max depth:  0 ;    Train accuracy:  0.3557692307692308 ;    Validation accuracy:  0.2608695652173913
Max depth:  1 ;    Train accuracy:  0.9519230769230769 ;    Validation accuracy:  0.9130434782608695
Max depth:  2 ;    Train accuracy:  0.9807692307692307 ;    Validation accuracy:  0.9565217391304348
Max depth:  3 ;    Train accuracy:  0.9807692307692307 ;    Validation accuracy:  0.9565217391304348
Max depth:  4 ;    Train accuracy:  0.9807692307692307 ;    Validation accuracy:  0.9565217391304348

Test accuracy:  0.9565217391304348
```

```
Enter max tree depth: 4
Enter discretizer bins number: 4
Max depth:  0 ;    Train accuracy:  0.34615384615384615 ;    Validation accuracy:  0.21739130434782608
Max depth:  1 ;    Train accuracy:  0.8942307692307693 ;    Validation accuracy:  0.8695652173913043
Max depth:  2 ;    Train accuracy:  0.9326923076923077 ;    Validation accuracy:  0.9565217391304348
Max depth:  3 ;    Train accuracy:  0.9423076923076923 ;    Validation accuracy:  0.9130434782608695
Max depth:  4 ;    Train accuracy:  0.9519230769230769 ;    Validation accuracy:  0.9565217391304348

Test accuracy:  0.9130434782608695
```

```
Enter max tree depth: 4
Enter discretizer bins number: 6
Max depth:  0 ;    Train accuracy:  0.34615384615384615 ;    Validation accuracy:  0.34782608695652173
Max depth:  1 ;    Train accuracy:  0.9711538461538461 ;    Validation accuracy:  0.9130434782608695
Max depth:  2 ;    Train accuracy:  0.9807692307692307 ;    Validation accuracy:  0.9565217391304348
Max depth:  3 ;    Train accuracy:  1.0 ;    Validation accuracy:  0.9130434782608695
Max depth:  4 ;    Train accuracy:  1.0 ;    Validation accuracy:  0.9130434782608695

Test accuracy:  0.8695652173913043
```

```
Enter max tree depth: 4
Enter discretizer bins number: 28
Max depth:  0 ;    Train accuracy:  0.33653846153846156 ;    Validation accuracy:  0.34782608695652173
Max depth:  1 ;    Train accuracy:  0.9423076923076923 ;    Validation accuracy:  0.9130434782608695
Max depth:  2 ;    Train accuracy:  1.0 ;    Validation accuracy:  0.782608695652174
Max depth:  3 ;    Train accuracy:  1.0 ;    Validation accuracy:  0.782608695652174
Max depth:  4 ;    Train accuracy:  1.0 ;    Validation accuracy:  0.782608695652174

Test accuracy:  0.8695652173913043
```

Analizując otrzymane wyniki zauważyć można kilka rzeczy. Po pierwsze zwiększanie ilości klas dyskretnych zwiększa dokładność modelu na zbiorze treningowym. Natomiast jeśli chodzi o dokładność na zbiorach walidacyjnych i testowych (czyli tych na których dokładności nam zależy (głównie na dokładności testowego)) zwiększanie liczby klas dyskretnych zwiększa dokładność modelu na tych danych do pewnego momentu (w tym przypadku do 3). Następnie wraz ze wzrostem liczby klas dokładności na zbiorach walidacyjnych i testowych spada. Mamy tu do czynienia ze zjawiskiem przetrenowania. Przy dużej ilości klas model zamiast uczyć się ogólnych zasad, zapamiętuje podany mu zbiór treningowy na pamięć. Nie jest to zjawisko pożądane. Należy więc dobierać ilość klas w rozsądny sposób.

Wpływ maksymalnej głębokości drzewa

Ilość klas dyskretnych = 2

```
Enter max tree depth: 1
Enter discretizer bins number: 2
Max depth:  0 ;    Train accuracy:  0.36538461538461536 ;    Validation accuracy:  0.26086956521739131
Max depth:  1 ;    Train accuracy:  0.6730769230769231 ;    Validation accuracy:  0.6956521739130435

Test accuracy:  0.6086956521739131
```

```
Enter max tree depth: 2
Enter discretizer bins number: 2
Max depth:  0 ;    Train accuracy:  0.375 ;    Validation accuracy:  0.21739130434782608
Max depth:  1 ;    Train accuracy:  0.5576923076923077 ;    Validation accuracy:  0.6086956521739131
Max depth:  2 ;    Train accuracy:  0.6826923076923077 ;    Validation accuracy:  0.782608695652174

Test accuracy:  0.782608695652174
```

```
Enter max tree depth: 3
Enter discretizer bins number: 2
Max depth:  0 ;    Train accuracy:  0.34615384615384615 ;    Validation accuracy:  0.21739130434782608
Max depth:  1 ;    Train accuracy:  0.6538461538461539 ;    Validation accuracy:  0.782608695652174
Max depth:  2 ;    Train accuracy:  0.7596153846153846 ;    Validation accuracy:  0.6956521739130435
Max depth:  3 ;    Train accuracy:  0.7788461538461539 ;    Validation accuracy:  0.6521739130434783

Test accuracy:  0.7391304347826086
```

```
Enter max tree depth: 4
Enter discretizer bins number: 2
Max depth:  0 ;    Train accuracy:  0.34615384615384615 ;    Validation accuracy:  0.30434782608695654
Max depth:  1 ;    Train accuracy:  0.6634615384615384 ;    Validation accuracy:  0.6956521739130435
Max depth:  2 ;    Train accuracy:  0.7211538461538461 ;    Validation accuracy:  0.6521739130434783
Max depth:  3 ;    Train accuracy:  0.7307692307692307 ;    Validation accuracy:  0.6956521739130435
Max depth:  4 ;    Train accuracy:  0.7403846153846154 ;    Validation accuracy:  0.6521739130434783

Test accuracy:  0.6521739130434783
```

Ilość klas dyskretnych = 3

```
Enter max tree depth: 1
Enter discretizer bins number: 3
Max depth:  0 ;    Train accuracy:  0.38461538461538464 ;    Validation accuracy:  0.17391304347826086
Max depth:  1 ;    Train accuracy:  0.9519230769230769 ;    Validation accuracy:  1.0

Test accuracy:  0.9565217391304348
```

```
Enter max tree depth: 2
Enter discretizer bins number: 3
Max depth:  0 ;    Train accuracy:  0.3557692307692308 ;    Validation accuracy:  0.391304347826087
Max depth:  1 ;    Train accuracy:  0.9519230769230769 ;    Validation accuracy:  0.9565217391304348
Max depth:  2 ;    Train accuracy:  0.9711538461538461 ;    Validation accuracy:  1.0

Test accuracy:  1.0
```

```
Enter max tree depth: 3
Enter discretizer bins number: 3
Max depth:  0 ;    Train accuracy:  0.3557692307692308 ;    Validation accuracy:  0.391304347826087
Max depth:  1 ;    Train accuracy:  0.9519230769230769 ;    Validation accuracy:  0.9565217391304348
Max depth:  2 ;    Train accuracy:  0.9807692307692307 ;    Validation accuracy:  0.9565217391304348
Max depth:  3 ;    Train accuracy:  0.9807692307692307 ;    Validation accuracy:  0.9130434782608695

Test accuracy:  0.9565217391304348
```

```
Enter max tree depth: 4
Enter discretizer bins number: 3
Max depth:  0 ;    Train accuracy:  0.38461538461538464 ;    Validation accuracy:  0.21739130434782608
Max depth:  1 ;    Train accuracy:  0.9615384615384616 ;    Validation accuracy:  1.0
Max depth:  2 ;    Train accuracy:  0.9903846153846154 ;    Validation accuracy:  1.0
Max depth:  3 ;    Train accuracy:  0.9903846153846154 ;    Validation accuracy:  1.0
Max depth:  4 ;    Train accuracy:  0.9903846153846154 ;    Validation accuracy:  1.0

Test accuracy:  0.9130434782608695
```

Z otrzymanych wyników możemy wyciągnąć kilka wniosków. Tak jak przy zwiększaniu ilości klas, zwiększenie maksymalnej głębokości, zwiększa dokładność modelu na danych treningowych. Natomiast dokładność klasyfikatora na zbiorach walidacyjnym i testowym zwiększa się do pewnego momentu, a następnie zmniejsza się. Po raz kolejny możemy zaobserwować powyżej pewnej głębokości, że model zaczyna się przetrenowywać. Nie warto więc zawsze tworzyć drzew o głębokości pozwalającej na użycie wszystkich atrybutów.

Dodatkowe spostrzeżenia

Zbiór iris, na którym trenowano klasyfikatory jest dość specyficzny. Przede wszystkim jest on bardzo mały. Zawiera on niewiele elementów, a do tego elementy tego zbioru mają tylko 4 atrybuty. Ilość atrybutów ogranicza więc dość ostro głębokości maksymalne wytworzonych z tego zbioru drzew. Co więcej niewielka liczność tego zbioru sprawia, że to jakie wybierzemy zbiory walidacyjne, testowe i treningowe ma duży wpływ na otrzymywane wyniki i dokładności. Zaobserwować można np. czasem, że dokładność modelu na zbiorze walidacyjnym i na zbiorze testowym są od siebie znacząco różne. Czasem zdarza nawet sytuacja, w której dokładność na zbiorze treningowym jest mniejsza niż na zbiorze testowym lub walidacyjnym. Zastosowanie walidacji krzyżowej mogłoby zapewne dawać dużo bardziej rozsądne wyniki, nie wyeliminowałaby jednak ona przypadkowości związanej z wyborem zbioru testowego.