Raport
Wprowadzenie do Sztucznej Inteligencji – Ćwiczenie 4.

Regresja i klasyfikacja

Autor: Aleksandra Jamróz, nr albumu: 310 708

Treść zadania

W ramach tego zadania należy zaimplementować drzewo decyzyjne przy pomocy algorytmu ID3 i przeprowadzić klasyfikację metodą k-krotnej walidacji krzyżowej dla zadanego zbioru danych dotyczącego klasyfikacji przyjęcia dzieci do przedszkola na podstawie informacji o strukturze i finansach rodziny. Zbiór powinien tworzyć 12960 obserwacji – 5 klas, z czego 3 z nich mają podobną liczność (ponad 4000 obserwacji) a 2 są rzadkie (2 i 328 obserwacje).

Implementacja klasyfikatora nie powinna zależeć od zadanego zbioru danych. Powinna ona być na tyle ogólna jak się da, aczkolwiek implementacja dodatkowej funkcjonalności nie jest wymagana (tzn. tworzenie wsparcia dla parametrów funkcji, które nie będą używane, nie jest konieczne).

Url do danych: https://archive.ics.uci.edu/ml/datasets/Nursery.

Rozwiązanie zadania – klasy

Elementy potrzebne do zaimplementowania ID3 postanowiłam skonstruować jako odrębne klasy. W związku z powyższym, powstały następujące:

1. **Dataset** – plik dataset.py

Klasa, od której zaczęłam implementację rozwiązania. Budowana jest na podstawie podanego zbioru danych, stworzonego uprzednio za pomocą funkcji *read_csv* biblioteki pandas z podanego pliku csv. Odzwierciedla zbiór danych w klasie. Jest to jej jedyne pole, poza nim klasa zawiera jedynie metody. Zaimplementowałam w niej funkcje potrzebne w kolejnych krokach do algorytmu ID3. Poniżej zamieszczam wzory, z których korzystałam w trakcie obliczeń.

- Entropia zbioru U:

$$I(U) = -\sum_i f_i \ln(f_i) \ ,$$

gdzie f_i - częstość i-tej klasy,

- Entropia zbioru podzielonego na podzbiory przez atrybut d:

$$Inf(d,U) = \sum_j \frac{|U_j|}{|U|} I(U_j)$$

- Zdobycz informacyjna:

$$InfGain(d,U) = I(U) - Inf(d,U)$$

Rysunek 1. Fragment wykładu dotyczącego drzew decyzyjnych. Wzory do liczenia entropii. Autor: Paweł Zawistowski

Funkcje:

- *self.frequency* – (float) – jako parametry przyjmuje nazwę kolumny tabeli oraz wartość, której będzie w danej kolumnie szukać. Liczy częstotliwość wystąpienia danej wartości w kolumnie;
- *self.complete_entropy* – (float) – na podstawie w.w. wzoru liczy entropię całego zbioru danych;
- *self.category_entropy* – (float) – służy do liczenia entropii pojedynczej kolumny. Jako parametr przyjmuje kategorię, według której chcemy liczyć. Dzieli cały zbiór danych na pomniejsze kawałki i dla każdego z nich liczy całkowite entropie, które sumują się do zwracanej wartości;
- *self.inf_gain* – (float) – korzystając z entropii całego zbioru i entropii kategorii podanej jako parametr, liczy dla niej zdobycz informacyjną;
- *self.best_inf_gains* – (tuple) – dla każdej kolumny w zbiorze danych liczy zdobycz informacyjną. Następnie sortuje je malejąco pod względem wartości i zwraca krotkę największej zdobyczy informacyjnej oraz nazwy kategorii;
- *self.split_by_category_plus* – (→ lista) – z tej funkcji korzystam później przy budowie właściwego drzewa. Dzieli ona zbiór danych na mniejsze podzbiory według wartości występujących w kolumnie podanej jako parametr. Dla każdej wartości tworzony jest mały zbiór, po czym kolumna zostaje z niego usunięta, ponieważ nie będzie więcej potrzebna. Zwraca listę wszystkich małych zbiorów;
- *self.get_category* – (int) – metoda potrzebna do porównywania wartości w zbiorze do wartości w liście argumentów, które chcemy zakwalifikować naszym modelem. Zwraca indeks kolumny podanej jako parametr, aby na jego podstawie wyłuskać z argumentów odpowiednią pozycję.

2. **AllNode** – plik tree.py

Zaimplementowanie uniwersalnego węzła, który można dodać do drzewa decyzyjnego okazało się bardzo ważnym zadaniem. W zależności od tego, czy węzeł jest liściem (nie posiada żadnych dzieci), czy nie, pełni inną rolę. Przechowuje wartości takie jak:

- *self.dataset* – zbiór danych, na którym operuje węzeł podczas wyznaczania predykcji;
- *self.category* – określa, jaką kategorię napotykamy na tym etapie schodzenia w głąb drzewa;
- *self.kids* – węzły potomne. Jeżeli węzeł jest liściem, nie posiada żadnych dzieci. Dzieci jest tyle, ile można utworzyć mniejszych zbiorów danych na podstawie kategorii rodzica;
- *self.parent_value* – wartość przechowywana przez starszy węzeł. Jeżeli węzeł jest korzeniem drzewa, nie posiada tej wartości, ponieważ nie posiada rodzica;
- *self.is_leaf* – True / False - określa czy węzeł jest liściem;
- *self.prediction* – wartość tego pola liczona jest tylko wtedy za pomocą metody count_prediction, gdy węzeł jest liściem. Wykorzystuje ona zbiór danych węzła i wnioskuje, jak zostanie zaklasyfikowany element, który dotrze do tego miejsca w drzewie.

3. **Tree** – plik tree.py

Algorithm 1: ID3

Input: Y: zbiór klas, D: zbiór atrybutów wejściowych, $U \neq \emptyset$: zbiór par uczących

1 **if** $\forall_{\{x_i,y_i\}\in U}\ y_i == y$ **then**
2 **return** *Liść zawierający klasę* y
3 **if** $|D| == 0$ **then**
4 **return** *Liść zawierający najczęstszą klasę w* U
5 $d = \arg\max_{d\in D} InfGain(d, U)$
6 $U_j = \{x_i, y_i\} \in U\ :\ x_i[d] = d_j,$ gdzie d_j - j-ta wartość atrybutu d
7 **return** *Drzewo z korzeniem* d *oraz krawędziami* $d_j, j = 1, 2, \ldots$ *prowadzącymi do drzew: ID3(*Y, $D - \{d\}$, U_1*), ID3(*Y, $D - \{d\}$, U_2*),...*

Rysunek 2. Fragment wykładu dotyczącego drzew decyzyjnych. Pseudokod algorytmu ID3. Autor: Paweł Zawistowski

Główną klasą stworzoną do właściwego wykorzystania drzewa, jest klasa Tree. Odpowiada ona za budowanie drzewa i przewidywanie przypisania dla podanego zbioru danych. Jego funkcja główna to *build_tree*, która na postawie zbioru danych generuje kolejne węzły oraz liście, łącząc je ze sobą w jedną strukturę.
Klasyfikacja podanego zbioru argumentów prowadzona jest według algorytmu ID3. Zaczyna się w korzeniu drzewa. Z listy pobierany jest odpowiedni argument, wyciągnięty na podstawie indeksu kolumny i porównany do wartości przechowywanej przez węzeł potomny od korzenia. Jeżeli dopasowanie jest prawdziwe, algorytm schodzi w głąb drzewa po wybranej gałęzi i rekurencyjnie powtarza proces klasyfikacji dla listy argumentów pomniejszonej o porównywany wcześniej element. Jeżeli ów węzeł jest liściem, klasyfikacja się kończy, a zwracana wartość to przewidywanie obliczone wewnątrz węzła. W sytuacji, gdy algorytm przejdzie po wszystkich dzieciach węzła, w którym obecnie się znajduje i nie znajdzie identycznej wartości, schodzi po gałęzi prowadzącej do kategorii o największej częstotliwości występowania. Taka sytuacja może zajść w przypadku, kiedy w zbiorze testowym wystąpi wartość, która nie wystąpiła w zbiorze trenującym.

Klasyfikacja za pomocą drzewa

Masowa kwalifikacja dużych zbiorów danych wykonywana jest za pomocą funkcji i3. Jako argumenty przyjmuje zbiór danych podzielony na zbiór danych treningowych oraz zbiór danych testowych. Nie przekazywałam jej całego zbioru danych jako parametru, ponieważ w różnych sytuacjach dzieliłam ten zbiór w inny sposób. Zwraca listę przewidywanych wartości.

Obliczanie metryk

W celu testowania poprawności i dokładności algorytmu napisałam funkcję zliczającą wartości True Positive, True Negative, False Positive oraz False Negative. Wartości False Positive i False Negative dzieliłam po obliczeniu na pół, ponieważ powtarzały się one dwukrotnie w wyniku sposobu implementacji funkcji zliczania. Na podstawie tych wartości wyznaczałam następujące współczynniki: Recall, Fall_out, Precision, Accuracy i F1, które były zwracane jako wynik działania metody.

Macierze pomyłek

W celu zwizualizowania macierzy pomyłek dla jednokrotnego wykonania funkcji i3 wykorzystałam bibliotekę pandas i z jej pomocą stworzyłam tabelę. Nazwami kolumn i indeksami wierszy zostały wartości, do których można zakwalifikować zbiór danych. Wszystkie pola początkowo ustawiane są na 0. W trakcie dopasowywania elementów porównywałam wartość prawdziwą do wartości przewidzianej i dodawałam 1 do odpowiedniego pola w macierzy. Tak powstałą macierz można narysować w konsoli.
Na przekątnej macierzy leżą wartości True Positive. Kolumny to wartości prawdziwe, rzędy oznaczają wartości przewidziane, tym samym w rzędach leżą wartości False Positive, kolumnach – False Negative. Pozostałe pola to True Negative.

	not_recom	spec_prior	priority	very_recom	recommend
not_recom	636	0	0	0	0
spec_prior	0	595	24	0	0
priority	0	27	604	19	0
very_recom	0	0	23	32	0
recommend	0	0	0	0	0

Rysunek 3. Przykład macierzy pomyłek dla klasyfikacji 11000:1950 elementów

	not_recom	spec_prior	priority	very_recom	recommend
not_recom	1114	0	0	0	0
spec_prior	0	946	60	0	0
priority	0	45	964	33	0
very_recom	0	0	22	56	0
recommend	0	0	0	0	0

Rysunek 4. Macierz pomyłek dla klasyfikacji dla pomieszanego zbioru w proporcji 0.75:0.25

Walidacja krzyżowa

Funkcja sprawdzająca efektywność algorytmu poprzez walidację krzyżową polega na kilkukrotnym stworzeniu drzewa i przetestowania zbioru dla różnych części zbioru danych. Krotność takiego podziału zbioru i wykonanych powtórzeń określa parametr k podawany funkcji wraz ze zbiorem danych jako parametr. W wyniku działania tej funkcji dla każdego z powtórzeń generowane były listy z metrykami opisanymi wyżej. Zwraca ona listę zawierającą uśrednione wartości każdej z metryk.

Testowanie algorytmu

Zdecydowałam się przeprowadzić 2 rodzaje testów. Pierwszy polegał na analizie różnic między wynikami klasyfikacji wykorzystującej walidację krzyżową o różnych wartościach parametru. Drugi natomiast na analizie różnic w wynikach klasyfikowania zbioru o różnych proporcjach zbioru treningowego i testowego. Oba rodzaje testów przeprowadzałam na dwóch plikach z danymi: w jednym dane były domyślnie posortowane, w drugim – specjalnie pomieszane.

I. Walidacja krzyżowa

Intuicyjnym byłoby stwierdzenie, że im więcej elementów zawiera zbiór treningowy, tym dokładniejsze będą predykcje klasyfikatora. Ta zależność w przypadku danych posortowanych ma miejsce. Przy podziale zbioru na 5 elementów wzwyż, dokładność klasyfikacji zwiększa się. Ze schematu wyłamuje się pierwszy przypadek, w którym dzielimy zbiór na 3 części – wyniki w tej sytuacji są prawie równie dobre, jak przy 10 częściach. Może to być spowodowane tym, że posortowane dane podzieliły się na znaczące części, co zwiększa liczbę elementów nieznajdujących swoich argumentów w drzewie.

Dane	k	Recall	Fall_out	Precision	Accuracy	F1_score
posortowane	3	0,95	0,95	0,89	0,81	0,92
posortowane	5	0,83	0,83	0,83	0,69	0,77
posortowane	7	0,84	0,84	0,87	0,76	0,82
posortowane	10	0,96	0,96	0,88	0,80	0,92
posortowane	20	1,00	1,00	0,97	0,93	0,98
Średnia wartość	9	0,91	0,91	0,89	0,80	0,88
Odchylenie standardowe	6,67	0,08	0,08	0,05	0,09	0,08

Rysunek 5. Tabela metryk klasyfikacji z walidacją krzyżową dla posortowanego zbioru danych

Z kolei w przypadku danych przemieszanych, nie ma zasadniczej różnicy, na ile elementów podzielimy zbiór, wyniki wychodzą zawsze podobnie. Są jednocześnie znacznie lepsze niż te dla danych posortowanych. Wynika to z większej różnorodności danych w każdym zestawie. Pozwala to na uzyskanie większej kombinacji wartości, tym samym bardziej rozbudowane drzewo, a co za tym idzie – dokładniejszy model.

Dane	k	Recall	Fall _out	Precision	Accuracy	F1_score
pomieszane	3	1,00	1,00	0,98	0,95	0,99
pomieszane	5	0,98	0,98	0,98	0,96	0,98
pomieszane	7	0,99	0,99	0,98	0,96	0,99
pomieszane	10	0,99	0,99	0,98	0,96	0,99
pomieszane	20	1,00	1,00	0,98	0,96	0,99
Średnia wartość	9	0,99	0,99	0,98	0,96	0,99
Odchylenie standardowe	6,67	0,01	0,01	0,00	0,00	0,00

Rysunek 6. Tabela metryk klasyfikacji z walidacją krzyżową dla pomieszanego zbioru danych

II. Różne proporcje zbioru testowego

Testując różne proporcje wielkości zbioru trenującego i testującego chciałam sprawdzić, jaka jest różnica w prawdomówności klasyfikatora. Przeprowadziłam próby dla 3 proporcji: 6:4, 75:25 oraz 9:1. Dla każdej z nich włączyłam algorytm dwukrotnie. Podobnie jak podczas prób z walidacją krzyżową, gorsze wyniki we wszystkich przypadkach uzyskał model działający na posortowanych danych. Różnica w wynikach przy proporcji 6:4 a 9:1 jest zauważalna. W przypadku danych pomieszanych różnica jest znikoma bądź nie ma jej wcale.

Dane	Test_size	Recall	Fall_out	Precision	Accuracy	F1_score
posortowane	0,1	1,00	1,00	0,98	0,96	0,99
posortowane	0,1	1,00	1,00	0,98	0,96	0,99
posortowane	0,25	0,93	0,93	0,98	0,96	0,96
posortowane	0,25	1,00	1,00	0,97	0,95	0,99
posortowane	0,4	0,98	0,98	0,98	0,95	0,98
posortowane	0,4	1,00	1,00	0,97	0,95	0,99
Odchylenie standardowe	0,134	0,027	0,027	0,003	0,004	0,013
Średnia	0,250	0,986	0,986	0,977	0,955	0,981

Rysunek 7. Tabela metryk dla klasyfikacji danych posortowanych dla różnych wartości proporcji

Dane	Test_size	Recall	Fall_out	Precision	Accuracy	F1_score
pomieszane	0,1	1,00	1,00	0,98	0,96	0,99
pomieszane	0,1	1,00	1,00	0,98	0,95	0,99
pomieszane	0,25	1,00	1,00	0,98	0,96	0,99
pomieszane	0,25	1,00	1,00	0,98	0,96	0,99
pomieszane	0,4	0,97	0,97	0,98	0,95	0,97
pomieszane	0,4	0,99	0,99	0,98	0,95	0,98
Odchylenie standardowe	0,134	0,011	0,011	0,001	0,003	0,006
Średnia	0,250	0,993	0,993	0,977	0,955	0,985

Rysunek 8. . Tabela metryk dla klasyfikacji danych pomieszanych dla różnych proporcji

Budowa drzewa

Drzewo decyzyjne budowane za pomocą algorytmu ID3 nie musi być symetryczne. W zależności od rozkładu danych na tej samej głębokości drzewa decyzja może zależeć od różnych kategorii. W przypadku zbioru danych o przedszkolakach, sekwencja kolejnych kategorii jest w większości przypadków stała. Główny nurt przebiega następująco:

Health → Has_nurs → Parents → Social → Housing → Finance → Form → Children

Uwagi

Zaimplementowany przeze mnie algorytm jest algorytmem uniwersalnym. To znaczy, że nie ma żadnych zdefiniowanych na stałe zmiennych ani nazw powiązanych z zadanym zbiorem danych. Można uruchomić go z dowolnym plikiem csv i będzie działał poprawnie.