Podstawy sztucznej inteligencji – uczenie maszynowe

Treść zadania:

Zaimplementować algorytm Gradient Boosting do predykcji bazujący na sekwencji drzew decyzyjnych. Zbiór danych do użycia: Boston Housing. Uzyskane rezultaty porównać z wynikami dla wybranej implementacji algorytmu ML z dostępnych bibliotek.

Założenia:

Dane zostały zaimportowane za pomocą modułów dostępnych w bibliotece sklearn. Dane zostały podzielone na treningowe oraz testowe za pomocą k-krotnej walidacji (w tym przypadku k=5). Ze względu na rodzaj danych implementowany algorytm oraz drzewo decyzyjne będzie bazować na regresji (a nie klasyfikacji).

Podział odpowiedzialności:

- **Kacper Kostecki:** odpowiednie przygotowanie danych do treningu/testowania, implementacja algorytmu, dokumentacja
- **Katarzyna Kulpa:** implementacja drzewa decyzyjnego, wykresy, poprawa błędów

Opis algorytmu:

Użyta funkcja straty: $L = \frac{1}{2}\left(y - F(x)\right)^2$ - dzięki tak dobranej funkcji straty jesteśmy w stanie obliczyć wartości minimalizujące funkcje przewidujące bez użycia metod iteracyjnych (wartość, która minimalizuje wszelkie funkcje występujące w algorytmie jest równa średniej wartości drugiej zmiennej).
Parametry możliwe do dostosowania w celu znalezienia najlepszego dopasowania:

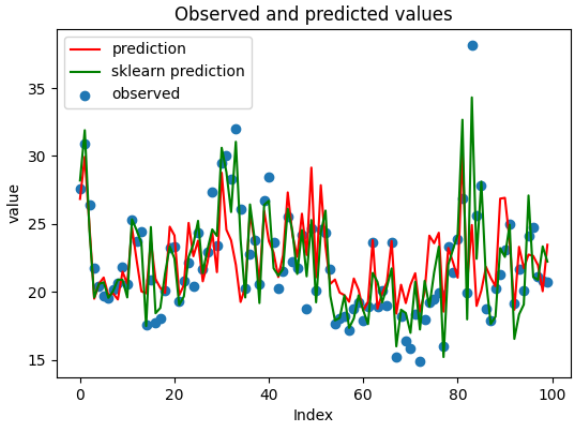
- Głębokość drzewa
- Ilość stworzonych drzew (iteracje)
- Stosunek podziału danych na treningowe oraz testowe (k-fold)
- Learning rate.

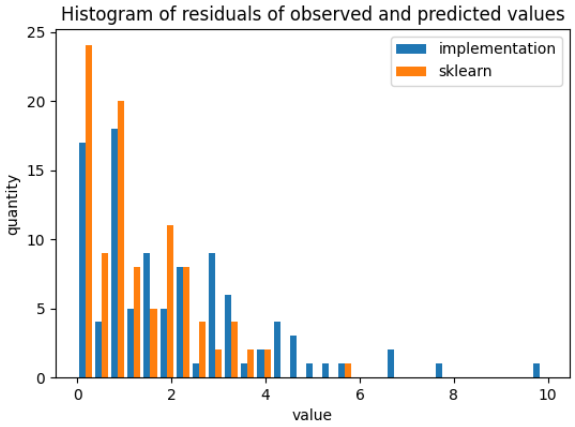
Poszukiwanie najlepszej wartości rozdzielającej dane w drzewie na dwie gałęzie odbywa się poprzez porównywanie sumy kwadratów różnicy między średnią wartością residuum oraz wartością danego punktu. W przypadku kilku wartości pseudo residuum w jednym liściu wyjściowa wartość równa jest średniej arytmetycznej wszystkich wartości w danym liściu – wynika to z wyboru odpowiedniej funkcji straty. Drzewo decyzyjne tworzone jest w sposób rekurencyjny na podstawie danych wejściowych oraz obliczonych residuów. Algorytm buduje z góry określoną liczbę drzew w celu dopasowania wyników. Dane są dzielone na treningowe oraz testowe w z wybranej proporcji – za każdym razem algorytm wykonuję k-krotną liczbę iteracje, gdzie k jest równe ilości części na które zostały podzielone dane. Ostateczny wynik jest obliczany na podstawie średniej z k iteracji.

Przeprowadzone eksperymenty:

Porównane zostaną parametry (średnia kwadratowa błędów, minimalna i maksymalna wartość, mediana, średnia arytmetyczna) obliczone na podstawie predykcji danych z naszej implementacji oraz implementacji z biblioteki sklearn, jak również danych zaobserwowanych. Wykresy przedstawiają poglądowo jak wygląda przybliżona funkcja do danych zaobserwowanych oraz histogram różnic pomiędzy wartością zaobserwowaną oraz przewidzianą.
Widocznym w każdym eksperymencie rażącą różnicą jest czas wykonania algorytmów, który w przypadku naszej implementacji jest znacząco większy w porównaniu do implementacji sklearn – bardziej szczegółowy opis przyczyn tego problemu został umieszczony we wnioskach końcowych.

1.Learning rate (parametry: k-fold=5, iteracje=200, tree depth=4):





	Dane testowe	projekt	sklearn	projekt	sklearn	projekt	sklearn	projekt	sklearn
Learning rate	-	**0.1**		**0.2**		**0.5**		**0.8**	
RMSE	-	3,14	1,443	3,125	1,459	3,067	1,735	3,256	2,039
Minimum	14.880	18,546	15,044	18,633	14,761	18,404	15,211	18,199	15,064
Maksimum	38.180	30,303	34,09	30,149	34,955	30,221	34,316	31,654	32,853
Mediana	21.230	21,555	21,271	21,589	21,339	21,619	21,368	22,153	21,638
Średnia	21.970	22,383	22,054	22,388	22,166	22,273	22,085	22,542	22,395
Czas [s]	-	23,301	1,146	19,281	1,138	17,196	1,225	17,255	1,264

W przypadku zwiększania learning rate można zauważyć powiększanie się rmse, przy jednoczesnym powiększaniu się skrajnych wartości. Niewielkie zwiększenie wskaźnika do poziomu 0,2 może poprawić przewidywanie skrajnych, odbiegających wartości, przy małym wzroście średniego odchylenia. Dalsze powiększanie wskaźnika prowadzi do zbyt dużego odchylenia.

2.Głębokość drzewa (parametry: learning rate=0.2, iteracje=200, k-fold=5):

	Dane testowe	projekt	sklearn	projekt	sklearn	projekt	sklearn	projekt	sklearn
Tree depth	-	**2**		**3**		**4**		**5**	
RMSE	-	3,041	1,635	3,027	1,463	2,706	1,492	3,125	1,515
Minimum	14.880	18,809	15,124	18,509	14,781	14,508	12,481	18,633	14,985
Maksimum	38.180	30,162	33,04	31,119	34,018	28,75	27,258	30,149	35,035
Mediana	21.230	21,816	21,185	21,53	21,322	17,739	17,937	21,589	21,412
Średnia	21.970	22,418	21,873	22,466	22	18,34	17,929	22,388	22,166
Czas [s]	-	17,249	0,68	17,988	0,934	17,487	0,886	19,683	1,163

Dla obu algorytmów można zauważyć wzrost czasu wykonywania wraz ze zwiększaniem drzewa z powodu większej ilości kroków potrzebnych do jego zbudowania. W przypadku innych parametrów najlepsze wyniki można zaobserwować dla depth=4 – w tym przypadku drzewo jest ograniczone do maksymalnie 16 liści, co teoretycznie powinno być najlepszym przedziałem dla algorytmu gradient boosting.

3. k-fold (parametry: learning rate=0.2, iteracje=200, tree depth=4):

	Dane testowe	projekt	sklearn	projekt	sklearn	projekt	sklearn
k-fold	-	**3**		**4**		**5**	
RMSE	-	2,479	1,744	2,106	1,53	1,925	1,43
Minimum	14.880	10,909	8,492	14,508	12,584	18,633	14,855
Maksimum	38.180	21,274	22,118	28,75	27,134	33,149	35,014
Mediana	21.230	13,617	13,252	17,739	17,989	21,589	21,344
Średnia	21.970	14,014	13,44	18,34	17,989	22,388	22,126
Czas [s]	-	9,668	0,617	14,881	0,902	20,38	1,189

Ze względu na większą ilości iteracji w przypadku zwiększania krotności, znacząco zwiększa się czas wykonania algorytmów. Zwiększanie krotności znacząco wpływa na rmse – lepsze dopasowanie do danych, dzięki różnym grupom testowym oraz treningowym.

4. Iteracje (parametry: learning rate=0.2, k-fold=4, tree depth=4):

	Dane testowe	projekt	sklearn	projekt	sklearn	projekt	sklearn	projekt	sklearn
iteracje	-	**50**		**100**		**200**		**400**	
RMSE	-	2,723	1,522	2,643	1,535	2,512	1,529	2,106	1,506
Minimum	14.880	14,508	12,561	13,508	12,625	13,348	12,517	13,534	12,603
Maksimum	38.180	28,75	26,974	29,397	27,49	29,342	27,16	28,965	27,179
Mediana	21.230	17,739	17,824	17,745	17,923	17,976	17,969	17,774	17,986
Średnia	21.970	18,342	17,944	18,213	17,992	18,321	17,996	18,543	17,963
Czas [s]	-	5,873	0,233	8,339	0,455	15,798	0,934	27,148	1,727

Wraz ze zwiększaniem ilości iteracji można zaobserwować mały spadek rmse oraz ogólne lepsze przybliżenie reszty parametrów. Jednakże zwiększanie ilości iteracji wiążę się ze znaczącym wzrostem czasu pracy algorytmu, co w tym przypadku nie jest aż tak opłacalne ze względu na stosunkowo mały wzrost dopasowania.

Wnioski:

Najsłabszym elementem naszej implementacji algorytmu jest tworzenie drzewa decyzyjnego, a w szczególności sposób poszukiwania odpowiedniej wartości rozdzielającej dane. Przez dużą złożoność obliczeniową zastosowanego rozwiązania, czas wykonywania algorytmu jest bardzo wysoki, co nie jest satysfakcjonującym wynikiem.
Ze względu na zróżnicowanie danych wejściowych, algorytm często ma problem z dopasowaniem się do skrajnie dużych wartości, które odbiegają od średniej, ale mimo to występuje ich sporo w całej bazie danych. Z tego powodu można zaobserwować stosunkowo dużą wartość rmse.

Instrukcja:

Program został napisany przy pomocy języka programowania python w wersji 3.9.1. Głównym plikiem projektu jest: main.py, który powinien zostać uruchomiony za pomocą komendy:

```
python3 main.py
```

Wszelkie parametry są możliwe do zmienienia w kodzie źródłowym. Biblioteki potrzebne do uruchomienia programu: pandas, matplotlib, sklearn, numpy