UMA

Projekt – Sprawozdanie Końcowe

Jakub Cholewiński 313329, Miłosz Kowalewski 318381

1. Założenia wstępne

W ramach dokumentacji wstępnej nakreśliliśmy podstawowe założenia dotyczące projektu, który polega na implementacji zmodyfikowanej wersji algorytmu generowania lasu losowego regresji, w której do generowania kolejnych drzew losowane są częściej elementy ze zbioru uczącego, na których dotychczasowy model popełniał większe błędy.

Z tego względu naszym celem było zaimplementowanie metody takiego losowania rekordów do nauki, aby model częściej wybierał potencjalnie trudniejsze przykłady. Idea takiego losowania była dość prosta – obliczamy błąd absolutny ze stworzonego już modelu tj. jednego, dwóch, dziesięciu itd. drzew, które stanowią część lasu i na jego podstawie wyznaczana jest waga do kolejnych losowań rekordów do wyznaczania następnych drzew.

Zgodnie z takim założeniem opracowaliśmy na początku tego projektu pseudokod, który przedstawiamy ponownie:

```
2
3  data_set = load_dataset()
4
5  data_set = data_set.init_weights() # all weights equal 1
6
7  trees = []
8
9  for(i:trees_num):
10     generated_samples = samples_with_weights(data_set, samples_number)
11     tree = generate_tree(params)
12
13     tree.train(generated_samples)
14
15     data_set = tree.validate_samples(data_set, test_samples_number)
16
17     trees.append(tree)
18
19
```

Zbiorem danych, na którym walidowaliśmy nasz model dotyczył cen sprzedanych mieszkań z wieloma atrybutami dostępny na stronie Kaggle – jego dokładny opis zamieszczony jest w kolejnych punktach.

2. Opis funkcjonalny

W projekcie, wykorzystaliśmy gotowe drzewa regresyjne zaimplementowane w bibliotece **sklearn**. Pozwoliło nam to na szybką implementację rozwiązania, oraz pozwoliło nam to wykluczyć ewentualny błąd związany z implementacją samego drzewa. W pierwszej kolejności, zaczynamy od ustalenia głównych parametrów, takich jak głębokość drzewa, maksymalna liczba cech, liczba drzew oraz maksymalna waga, pojedynczej próbki, o której będzie mowa w dalszej części dokumentacji.

Następnie dla pojedynczego drzewa, na podstawie wag, wybieramy próbki ze zwracaniem, które zostają wykorzystane do uczenia. Ilość próbek wybieranych do tworzenia drzewa jest równa 63,2% wszystkich danych wybranych do uczenia. Po wybraniu próbek i utworzeniu drzewa, dodajemy go do listy już istniejący drzew i na ich podstawie dokonujemy predykcji dla całego zbioru danych. Predykcja jest średnią z wartości wyznaczonych w pojedynczych drzewach. W dalszej części projektu, wyznaczone predykcje są wykorzystywane w celu wyznaczenia wag dla poszczególnych próbek, które następnie są używane w kolejnych iteracjach algorytmu.

3. Opis algorytmu i zbioru danych

A) Algorytm

Głównym punktem naszego projektu było ustalenie odpowiedniego algorytmu, przypisującego większe wagi próbkom generującym większy błąd. Przypisując wagi, należy zachować balans, pomiędzy gratyfikowaniem błędnych próbek, a pomijaniem dobrych rozwiązań. Dlatego odrzuciliśmy pomysły bezpośredniego przypisywania błędu jako wagi danej próbki i wykorzystaliśmy go jako przeskalowaną część wagi. Bazowo każda próbka ma wagę równą 1 i następnie, dodajemy do niej część związaną z wielkością generowanego błędu. Błąd dzielimy przez maksymalny błąd występujący w zabiorze danych, co gwarantuje nam wartości z przedziału 0-1, które następnie przemnażamy przez stałą równą maksymalnej wadze minus jeden i dodajemy do bazowej wartości. Takie rozwiązanie daje nam możliwość parametryzacji gratyfikacji błędnych rozwiązań oraz w miarę zrównoważone wykorzystywanie dobrych danych.

B) Zbiór danych

Zbiór danych na jakim bazowaliśmy dotyczył cen sprzedanych mieszkań dostępny poprzez stronę Kaggle. Zawiera on 79 atrybutów liczbowych jak również i kategorycznych (słowa) oraz 1460 rekordów. Wśród atrybutów można wskazać takie jak: typ strefy gdzie znajduje się budynek (rolnicza, komercyjna, przemysłowa,

zamieszkania etc.), typ drogi obok budynku, przedziały wysokości piwnicy, czy również atrybuty ciągłe jak np. metraż (a w zasadzie ilość stóp kwadratowych), ilość pokoi, czy łazienek a także rok, czy miesiąc sprzedaży.

W ramach preprocessingu danych usunęliśmy kilka bardzo wybrakowanych kolumn np. dla parametrów “Fence”, czy “MiscFeature” jedynie około połowa rekordów miała dane, zaś “rekordzistą” okazał się atrybut “Alley” dla którego jedynie 91 z 1460 rekordów miała dane. Dodatkowo do trenowania modelu wybraliśmy 90% pozostałych rekordów zaś do testowania pozostałe 10%.

Dla atrybutów kategorycznych (słowa) zastosowaliśmy rozwiązanie typowe w modela uczenia maszynowego czyli tzw. 1 z n. Polega ono na rozbiciu jednej kolumny atrybuty kategorycznego na n innych, gdzie n to ilość elementów ze zbioru wartości pierwotnego atrybutu kategorycznego. W tak stworzonych kolumnach wartościami są jeden lub zero i jedynka występuje jedynie w kolumnie odpowiadającej pierwotnej wartości atrybutu kategorycznego.

Pełny opis zbioru danych można przeczytać na stronie:

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

4. Eksperymenty

W ramach eksperymentów chcieliśmy się skupić na porównaniu naszej implementacji losu losowego z implementacją dostępną w ramach biblioteki **sklearn**. Przede wszystkim testowaliśmy wpływ kilku parametrów takich jak: maksymalna głębokość drzew, ilość drzew w lesie, maksymalna liczba uwzględnianych cech a także maksymalna waga przy losowaniu rekordów do nauki, o której była mowa w sekcji o algorytmie.

Zdecydowaliśmy się wykorzystać po kilka wartości tych parametrów:

Maksymalna głębokość: 4, 7, **10**, 15, 20, 30

Ilość drzew: 10, 30, 50, **100**, 150, 200

Maksymalna waga: 2, 5, **10**, 20, 50, 100

Maksymalna liczba uwzględnianych cech: 5, 10, **15**, 50, 100, 200

Jak widać powyżej niektóre liczby są pogrubione - są to domyślne wartości parametrów. Ze względu na bardzo długie czasy wykonania nie testowaliśmy wszystkich możliwych kombinacji, których liczba wynosi 1296, lecz zdecydowaliśmy się na testowania z osobna tych parametrów. Jesteśmy świadomi, że nie jest to rozwiązanie idealne, lecz staraliśmy możliwie dobrze dobrać wartości domyślne. Zgodnie z informacjami

dostępnymi w literaturze oraz na wykładach maksymalna liczba uwzględnianych cech typowo powinna wynosić pierwiastek z liczby atrybutów zaokrąglony w dół. W naszym przypadku ilość atrybutów po zastosowaniu kodowania 1 z n dla atrybutów kategorycznych wyniosła 255, stąd pierwiastek z tej liczby zaokrąglony w dół do liczby całkowitej to 15. Analogicznie potraktowaliśmy maksymalną głębokość drzew oraz ich ilość szukając o nich informacji w typowych implementacjach lasów losowych. Jedynym elementem, który w zasadzie przyjęliśmy względnie arbitralnie jest maksymalna waga. Zdecydowaliśmy się na wartość 10 chcąc jakościowo dokonać pewnego kompromisu między częstszym wybieraniem tych przykładów, na których generowane są większe błędy a jednoczesnym niezdominowaniu losowania ze zwracaniem przez te przykłady.

Do samej oceny między naszą implementacją a tą z biblioteki **sklearn** zdecydowaliśmy się na wyznaczanie kilku wskaźników jakości - błąd średniokwadratowy, błąd absolutny oraz wynik z funkcji score, który jest domyślny dla tej biblioteki. Jego opis można przeczytać poniżej a na jego użycie wpłynęła chęć możliwie zbliżonej oceny do tej domyślnej z biblioteki **sklearn**:

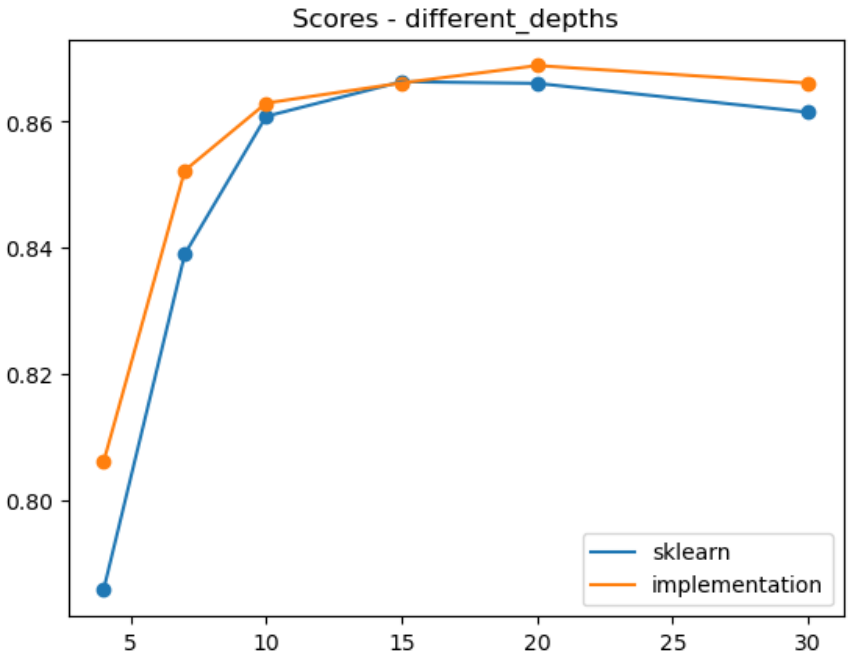
```
score(X, y, sample_weight=None)                                    [source]
```

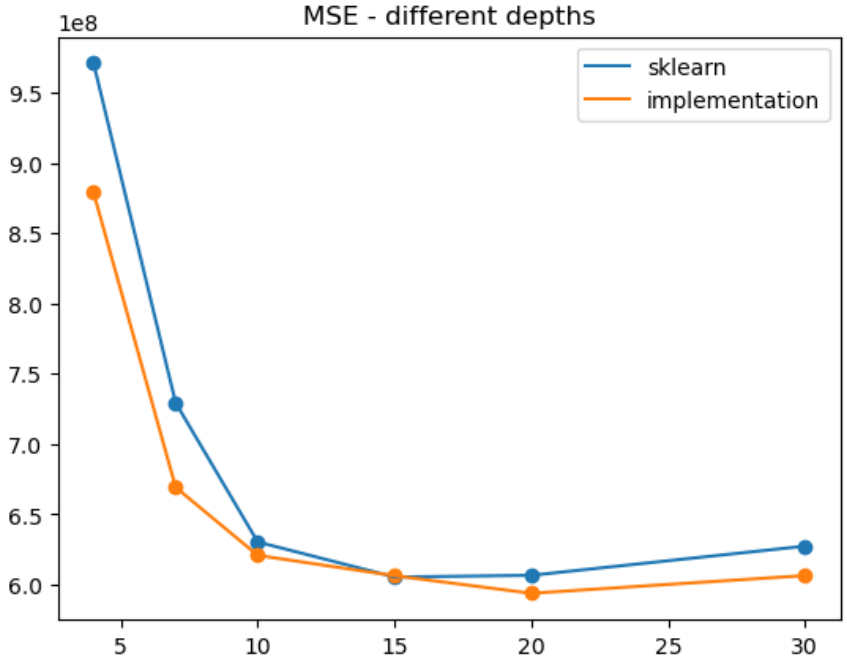
Return the coefficient of determination of the prediction.

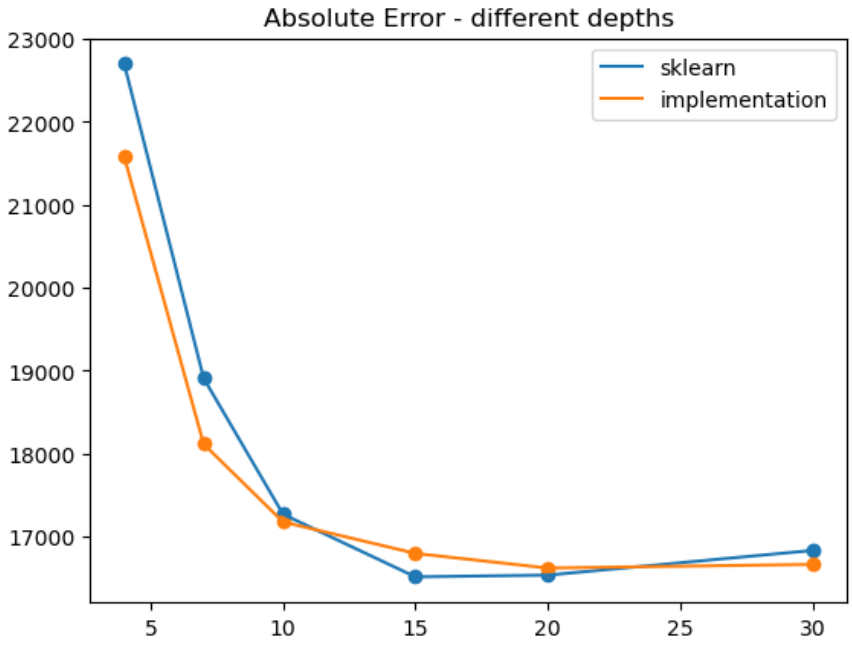
The coefficient of determination R^2 is defined as $(1 - \frac{u}{v})$, where u is the residual sum of squares `((y_true - y_pred)** 2).sum()` and v is the total sum of squares `((y_true - y_true.mean()) ** 2).sum()`. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of `y`, disregarding the input features, would get a R^2 score of 0.0.

Dodatkowo warto zaznaczyć, że las losowy, jak sama nazwa wskazuje ma w sobie elementy losowe, stąd wyznaczając każdy ze wskaźników dokonaliśmy wielu odpaleń i uśrednialiśmy ich wartości tak, aby otrzymywać możliwie klarowny obraz wpływu parametrów na działanie modelu.

A) **Maksymalna głębokość**

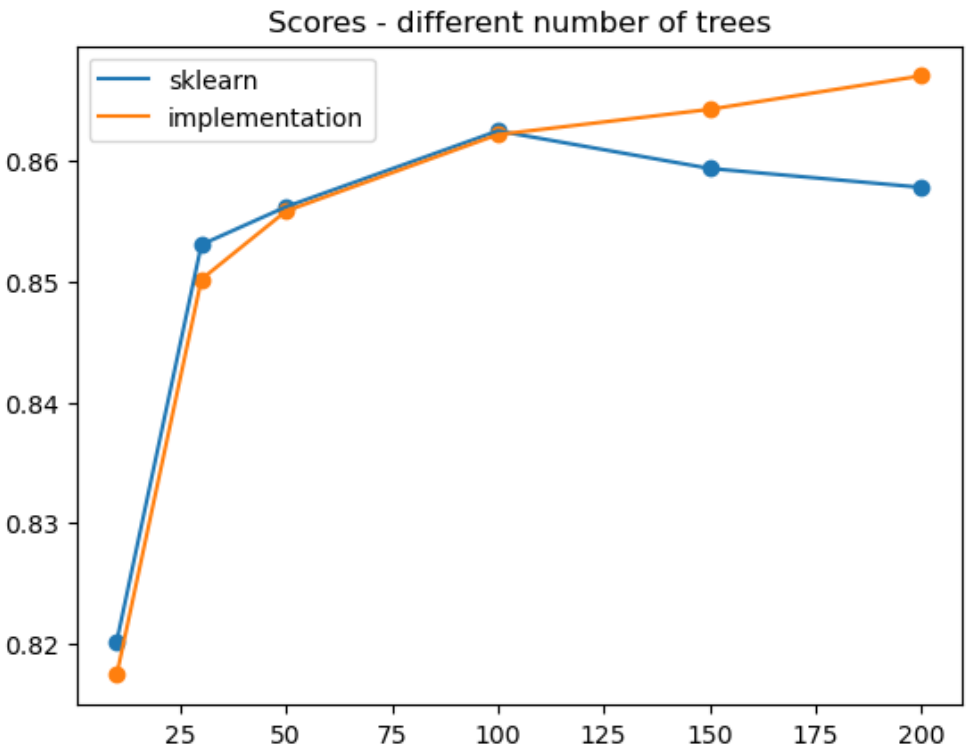

Scores - different_depths
0.86
0.84
0.82
0.80
5
10
15
20
25
30
sklearn
implementation

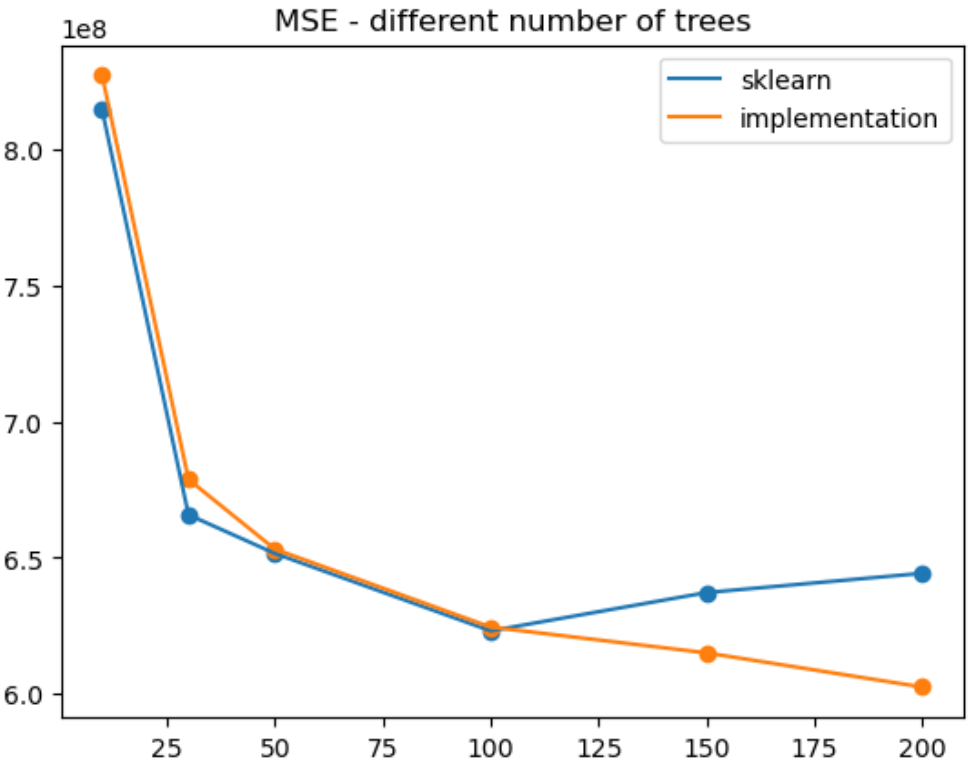

MSE - different depths
1e8
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
5
10
15
20
25
30
sklearn
implementation

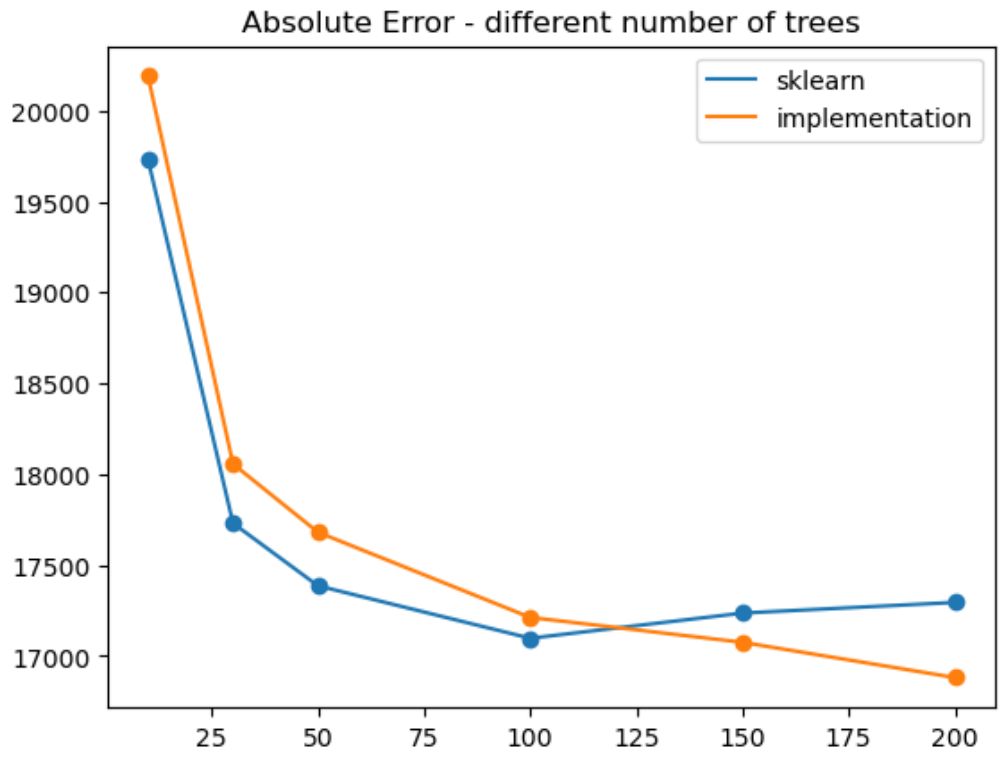


Jak możemy zauważyć z powyższych zależności, z pewnością możemy stwierdzić, że nasza implementacja generuje lepsze rezultaty dla małych głębokości drzew. W przypadku większych głębokości rezultaty nie są tak oczywiste, ale możemy stwierdzić, że nasze rozwiązanie jest bardziej stabilne i nie powoduje zbytniego przeuczenia zbioru dla danych uczących. Dodatkowo możemy stwierdzić, że w naszej implementacji głębokość drzew nie ma zbytniego wpływu na czas wykonywanych obliczeń, natomiast w przypadku sklearn ten czas się wydłuża wraz z większą głębokością.

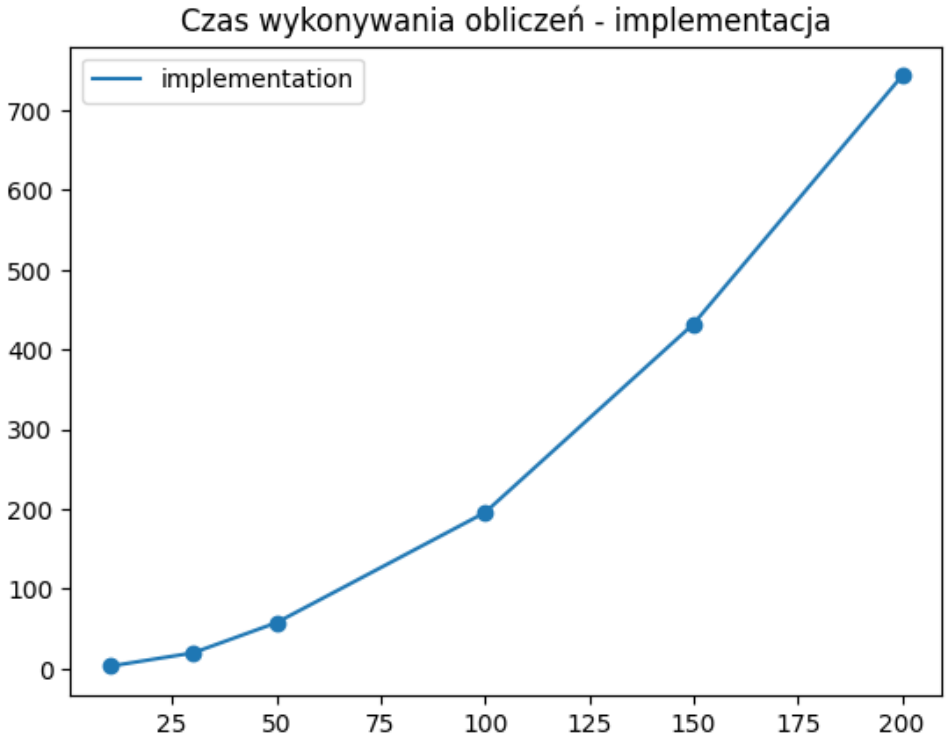
B) Ilość drzew w lesie


Scores - different number of trees
sklearn
implementation
0.86
0.85
0.84
0.83
0.82
25
50
75
100
125
150
175
200


MSE - different number of trees
1e8
sklearn
implementation
8.0
7.5
7.0
6.5
6.0
25
50
75
100
125
150
175
200

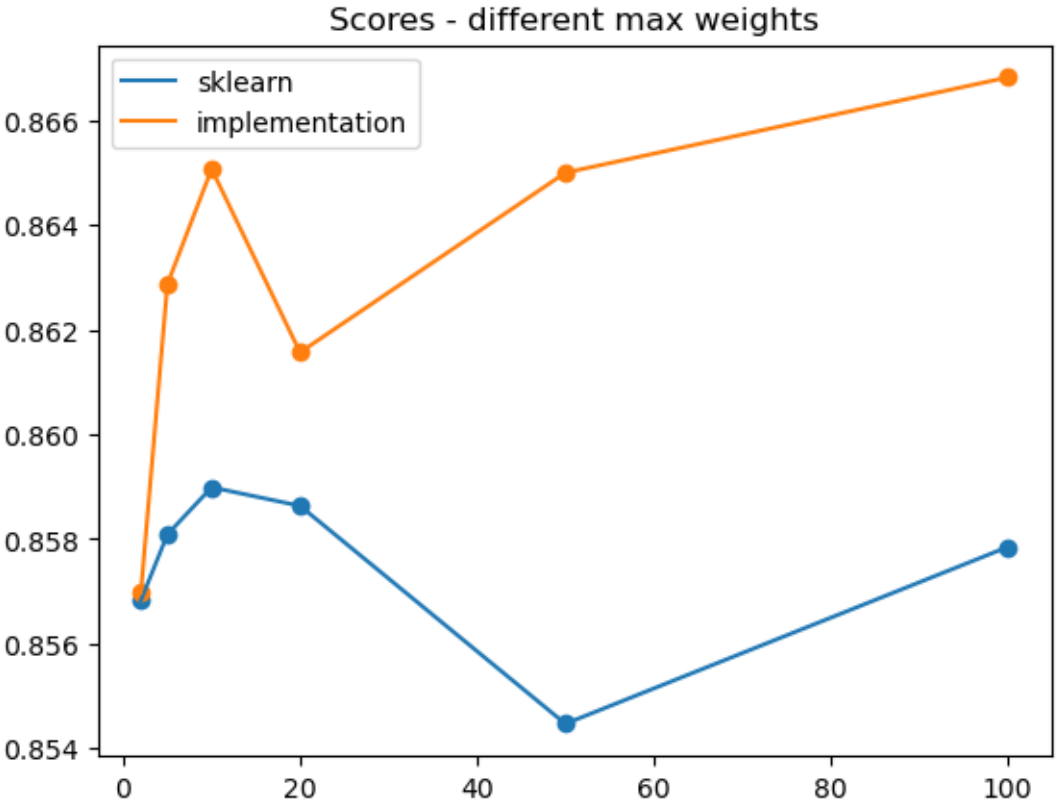


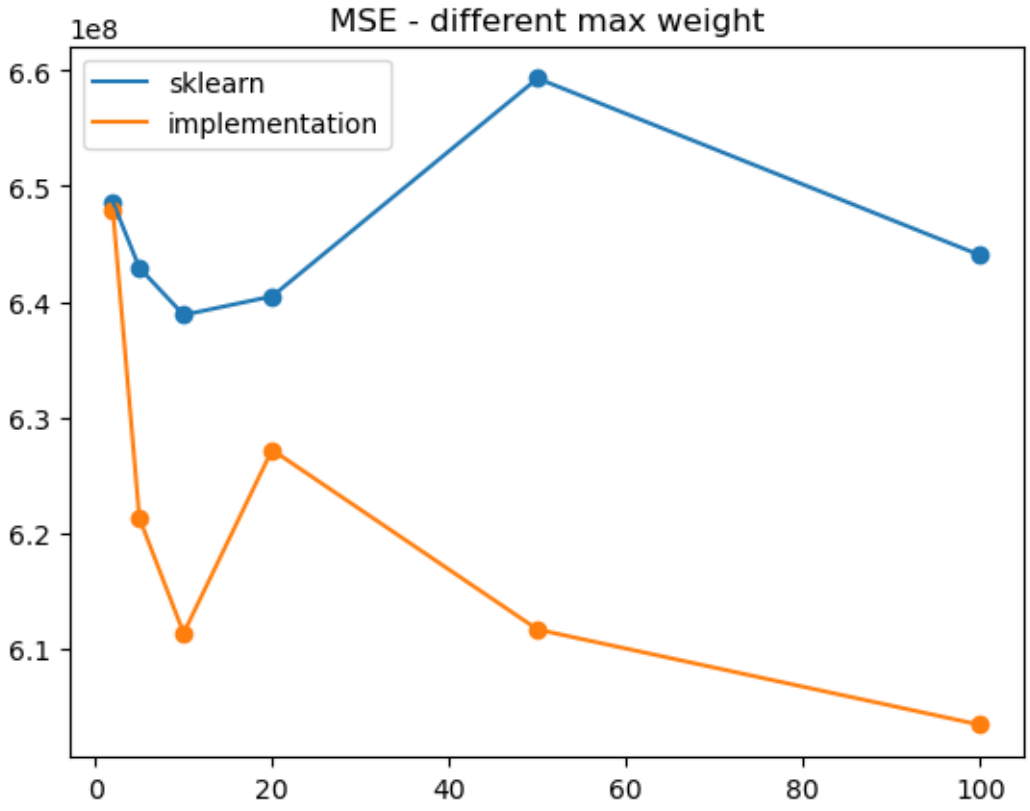
Z powyższego eksperymentu, dotyczącego liczby drzew decyzyjnych w lasie losowym, możemy dojść do wniosku, że nasza implementacja lepiej prezentuje się dla większej ilości drzew, co jest dość oczywiste, z powodu częstszego wybierania próbek generujących duży błąd. Dla mniejszych głębokości wyniki są dość porównywalne.

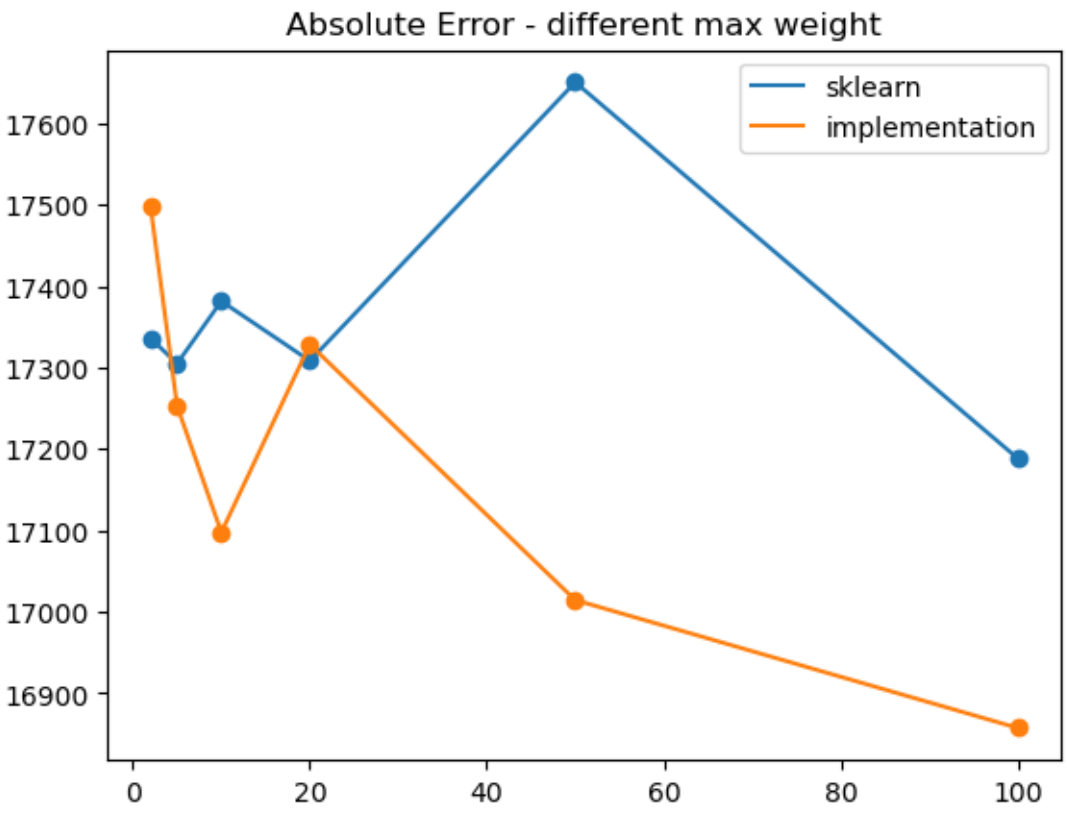


Ciekawą rzecz możemy zauważyć przy czasie wykonywanych obliczeń. W przypadku implementacji sklearn czas rośnie liniowo do ilości drzew, natomiast dla naszej implementacji czas rośnie zdecydowanie szybciej. Jest to spowodowane dodatkowa predykcją dla wszystkich danych w celu aktualizacji wag.

C) Maksymalna waga



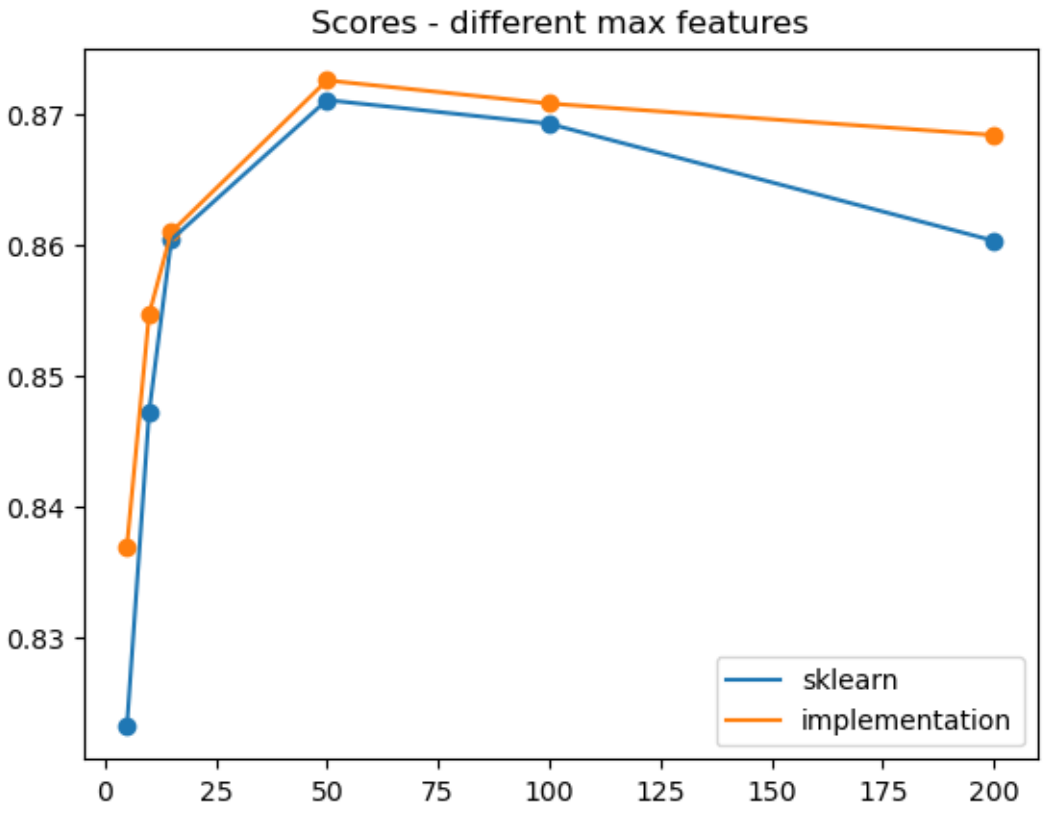


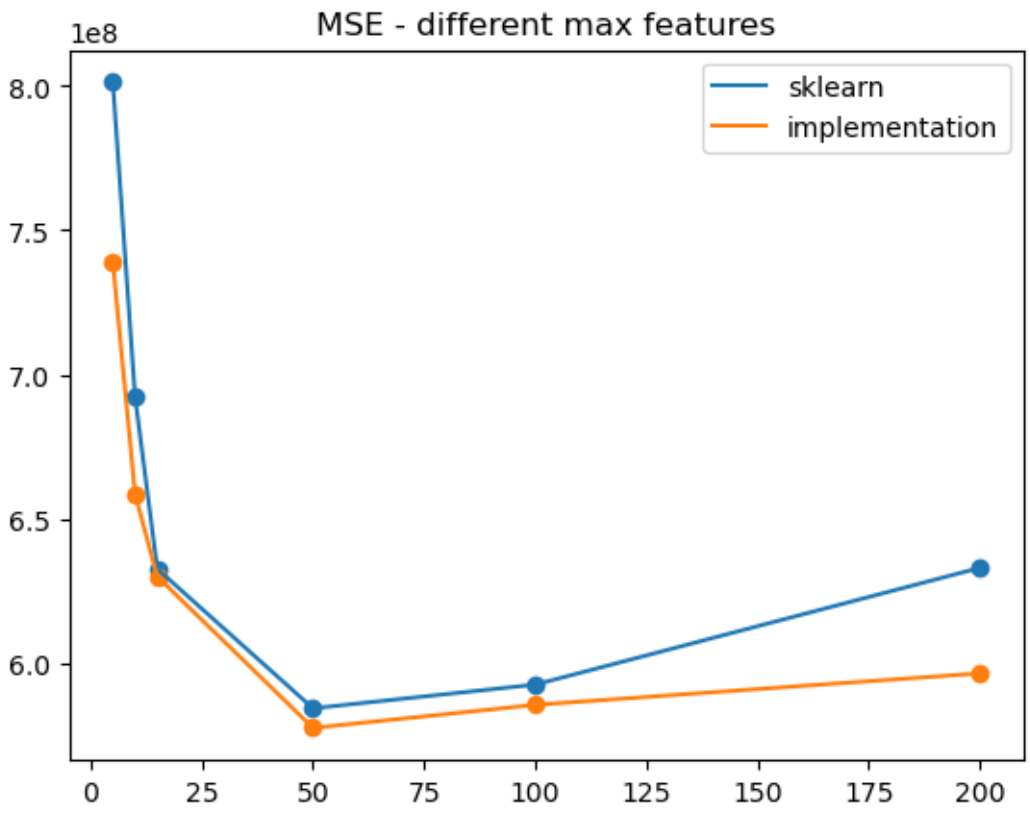


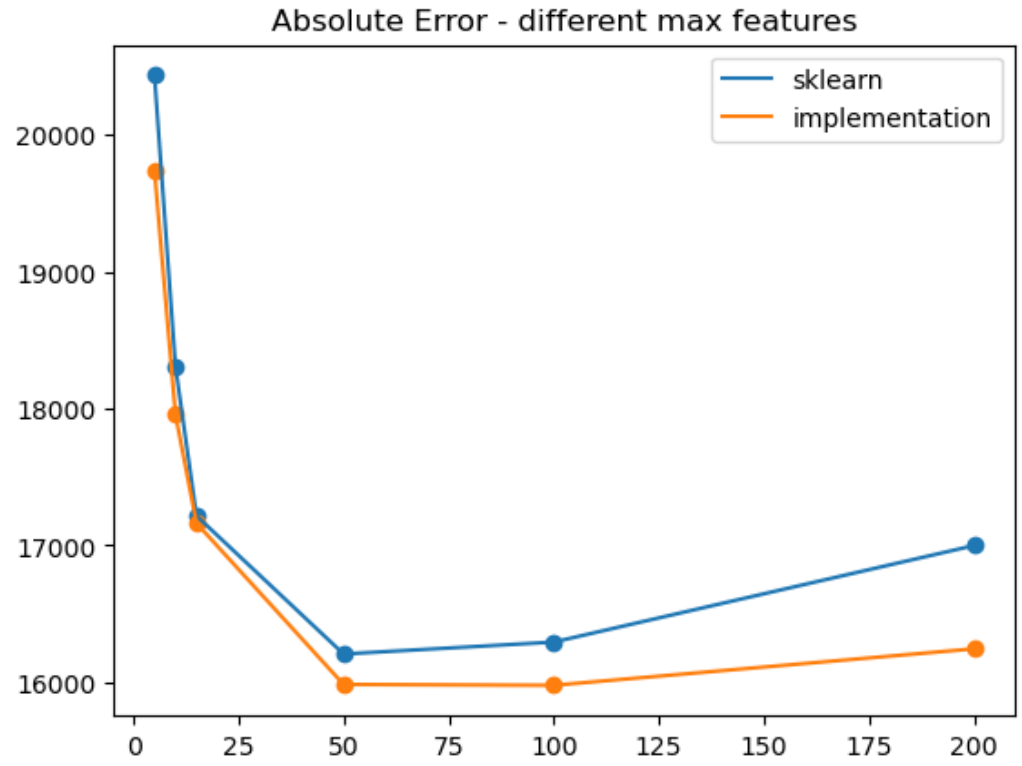
W przypadku gotowej implementacji maksymalna waga oczywiście w żaden sposób nie wpływa - nie jest nawet uwzględniana, lecz co ciekawe nawet w naszej implementacji trudno mówić o jednoznacznej tendencji zmian wartości wskaźników. Ponownie widać pewne różnice między wynikami dwóch implementacji, lecz trudno wskazywać na ogromny wpływ naszego sposobu losowania. Zakres zmian wartości wskaźników jest podobny dla obu implementacji, lecz warto zwrócić uwagę na maksymalną wagę równą dwa – wtedy modele są bardzo zbliżone do siebie, ponieważ rekordy nie są szczególnie mocno faworyzowane i odzwierciedla się to w bardziej zbliżonych wartościach parametrów.

Zgodnie z oczekiwaniami zmiana tego parametru nie ma wpływu na czasy wykonania, które wciąż wynoszą około 3 minuty i 20 sekund dla naszej implementacji (10 odpaleń dla danej wartości parametru), zaś dla wersji z **sklearn** konsekwentnie wynosi między 2 a 3 sekundy.

D) Maksymalna liczba uwzględnianych atrybutów

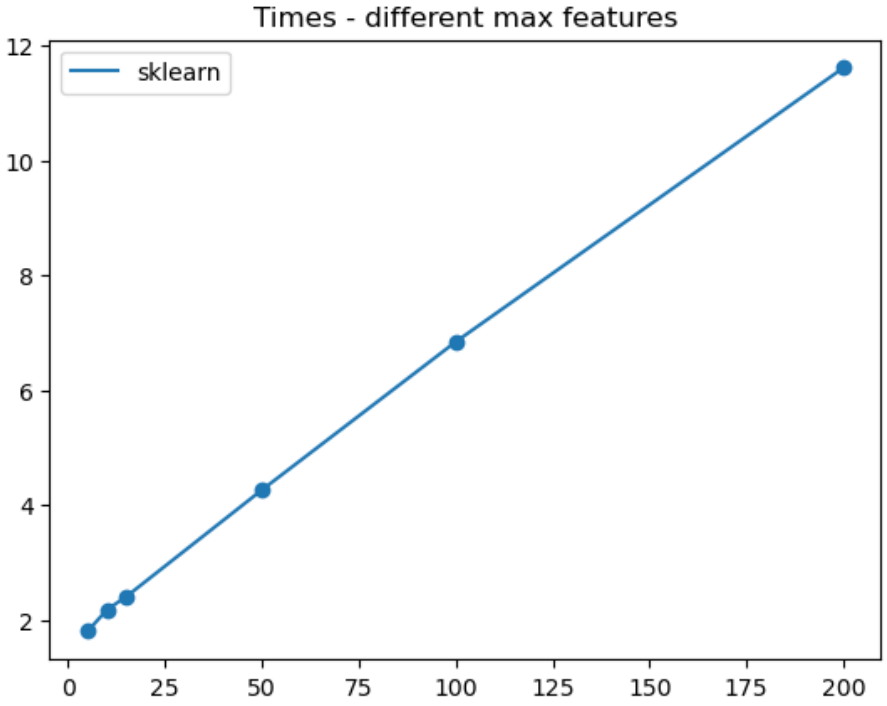






W przypadku maksymalnej liczby atrybutów widać odbicie w wartości 50 i można wskazać, że jest to kluczowy punkt, dla którego każdy wskaźnik przyjmuje najlepszą

wartość. Jest to o tyle ciekawe, że zauważalnie odbiega to od typowej wartości podawanej w literaturze. Niemniej ma to odbicie na czasie wykonania dla implementacji z **sklearn**:



Widać, że rośnie on liniowo w zależności od maksymalnej ilości uwzględnianych cech. W przypadku nie ma to istotnego wpływa, ponieważ czasy wykonania dalej wynoszą około 3 minut, co oczywiście wynika ze żmudnego procesu aktualizacja wag do losowania.

5. Podsumowanie eksperymentów

Trudno jednoznacznie wypowiedzieć się o jakości modyfikacji algorytmu losowego. Z jednej strony konsekwentnie dla różnych wartości parametrów wskaźniki jakości były lepsze dla naszej implementacji, niemniej różnice w tych wartościach nie były zbyt duże. Z tego powodu zdecydowaliśmy, że warto wykonać ostatnie porównanie - wybraliśmy takie wartości parametrów, dla których uzyskaliśmy najlepsze wyniki i dla nich jeszcze raz wykonaliśmy wiele uruchomień (tym razem 30). Jeszcze raz warto tu zaznaczyć, że w naszych testach badaliśmy wpływ zmiany jednego parametru ze stałymi wartościami pozostałymi, więc kombinacja użyta w ostatnim teście wcale nie musi być optymalna, niemniej czasy wykonania były na tyle duże, że nie widzieliśmy większego sensu sprawdzania każdej kombinacji.

Zatem najlepsze parametry dla naszej implementacji to:

Maksymalna głębokość: 20

Ilość drzew: 200

Maksymalna waga: 100

Maksymalna liczba uwzględnianych cech: 50

Zaś dla implementacji z **sklearn**:

Maksymalna głębokość: 20

Ilość drzew: 100

Maksymalna liczba uwzględnianych cech: 50

Dla tak dobranych parametrów uśrednione wartości wskaźników jakości prezentują się następująco:

Nasza implementacja:

Score: 0.8781, MSE: 552141323.7651, Błąd absolutny: 15650.3921

Implementacja **sklearn**:

Score: 0.8744, MSE: 568912334.7797, Błąd absolutny: 15903.4977

Na podstawie powyższego testu dla wybranych parametrów minimalnie lepsza okazała się nasza implementacja. Różnice są marginalne, stąd nie można mówić o jakiejś dominacji danej metody. Warto jednak wskazać, że przy badaniu wpływu parametrów rzeczywiście nasza implementacja konsekwentnie generowała wyniki lepsze bądź równe względem metody z **sklearn**. Wskazuje to na potencjał takiego losowania rekordów, co można byłoby osiągnąć poprzez dalszą eksplorację wpływu parametrów. Szczególnie obiecujący wydaje się wpływ ilości drzew w lesie – niemniej z tym parametrem związana jest również największa wada naszego rozwiązania - czas wykonania. Rośnie on nieliniowo (możliwe, że jest to wzrost kwadratowy) wraz ze wzrostem parametru, co znacząco wpływa na ewentualne możliwości zastosowania oraz dalszej eksploracji wpływu parametrów - sam czas wykonania 30 uruchomień wyniósł 39 minut, gdzie gotowe rozwiązanie z **sklearn** wygenerowało wyniki dla tylu odpaleń w 30 sekund. Oczywiście czas wykonania jest ściśle powiązany z maszyną, na której zachodzą uruchomienia, lecz różnica w tym przypadku jest kolosalna przez co rozważana metoda losowania w ramach tego projektu mogłaby się okazać nieaplikowalna w przypadku dużych zbiorów danych.

6. Opis wykorzystanych narzędzi

RandomForesstRegressor – gotowa implementacja regresyjnego lasu losowego z biblioteki sklearn. Pozwala nam na porównanie naszej wersji algorytmu

DecisionTreeRegressor – gotowa implementacja regresyjnego drzewa z biblioteki sklearn. Umożliwia nam własną implementację lasu losowego

Pandas – biblioteka wykorzystywana do ekstrakcji i przechowywania danych

Numpy – biblioteka umożliwiająca łatwe operacje matematyczne na macierzach