Przewidywanie odporności na ściskanie betonu

KWD
Karolina Jeż
Dawid Górski

03.01.2023

Streszczenie

W projekcie zostały przedstawione wyniki badań nad trenowaniem modeli dla zbioru danych Concrete Compressive Strength Data Set. Należało przewidzieć odporność betonu na ściskanie. Zastosowaliśmy modele regresji liniowej, regresji wielomianowej z Polynomial Features oraz z zastosowaniem algorytmu Recursive Feature Elimination. Modele zostały ocenione za pomocą wartości błędu średniokwadratowego oraz współczynnika determinacji. Najlepsze wyniki uzyskano w przypadku modeli regresji wielomianowej dla danych skalowanych.

Spis treści

Rozdział 1

Wprowadzenie

Dane do projektu pochodzą ze zbioru danych *Concrete Compressive Strength Data Set*. Składają się one z dwóch plików: *Concrete_Data.xls* oraz *Concrete_Readme.txt*.

W pliku *Concrete_Data.xls* znajdują się dane dotyczące 1030 próbek betonu. Każda próbka została wykonana w laboratorium i poddana testowi na wytrzymałość na ściskanie. W pliku znajduje się 9 kolumn, które opisują cechy pobranych próbek betonu. W ostatniej z nich znajduje się wartość wytrzymałości na ściskanie. Pozostałe kolumny opisują cechy próbek betonu. Wszystkie dane w pliku są przechowywane jako liczby zmiennoprzecinkowe.

Cechy opisujące próbki betonu stanowią:

- **Cement** (component 1) – quantitative – kg in a m3 mixture – Input Variable
- **Blast Furnace Slag** (component 2) – quantitative – kg in a m3 mixture – Input Variable
- **Fly Ash** (component 3) – quantitative – kg in a m3 mixture – Input Variable
- **Water** (component 4) – quantitative – kg in a m3 mixture – Input Variable
- **Superplasticizer** (component 5) – quantitative – kg in a m3 mixture – Input Variable
- **Coarse Aggregate** (component 6) – quantitative – kg in a m3 mixture – Input Variable
- **Fine Aggregate** (component 7) – quantitative – kg in a m3 mixture – Input Variable

- **Age** – quantitative – Day (1 365) – Input Variable
- **Concrete compressive strength** – quantitative – MPa – Output Variable

W pliku *Concrete_Readme.txt* znajduje się krótki opis zbioru danych, tj. źródło, opis atrybutów oraz autorzy.

1.1 Opis problemu

Zadanie polega na wykonaniu modelu regresji liniowej, który będzie przewidywał wytrzymałość na ściskanie betonu na podstawie 9 cech numerycznych próbek betonu. W tym celu należy wykonać analizę eksploracyjną zbioru danych, i zdecydować, które cechy mogą zostać pominięte, oraz czy należy wykonać skalowanie danych.

W celu sprawdzenia czy model jest wystarczająco dokładny należy wykonać walidację krzyżową, a także zwrócić uwagę na wartość błędu średniokwadratowego oraz wariancji.

W celu wykonania zadania wykorzystaliśmy bibliotekę scikit-learn. Wszystkie metody zostały zaimplementowane w języku Python.

Rozdział 2

Opis metody

2.1 Wprowadzenie teoretyczne

W celu wykonania zadania wykorzystaliśmy kilka metod uczenia maszynowego, tj: **regresję liniową**, **regresję wielomianową**, algorytm **Recursive Feature Elimination** oraz metodę **Polynomial Features**. Na potrzeby walidacji modelu wykorzystaliśmy metodę **Cross Validation**.

Regresja liniowa jest jedną z najprostszych metod uczenia maszynowego. Polega ona na dopasowaniu funkcji liniowej do zbioru danych. Funkcja ta jest zdefiniowana wzorem:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \tag{2.1}$$

gdzie:

- $\hat{y}$ – wartość przewidywana
- n – liczba cech
- x_i – wartość cechy i
- θ_i – parametr modelu

Regresja wielomianowa jest rozszerzeniem regresji liniowej. Polega ona na dopasowaniu funkcji wielomianowej do zbioru danych. Funkcja ta jest zdefiniowana wzorem:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \ldots + \theta_n x_n^n \tag{2.2}$$

gdzie:

- $\hat{y}$ – wartość przewidywana
- n – liczba cech

- x_i – wartość cechy i
- θ_i – parametr modelu

Algorytm Recursive Feature Elimination jest algorytmem wybierającym cechy, które mają największy wpływ na przewidywanie wartości. Bazuje on na algorytmie regresji liniowej.

Kroki wykonywane przez algorytm RFE:

1. Zdefiniuj liczbę cech, które mają zostać wybrane
2. Wytrenuj model
3. Oblicz wartość błędu średniokwadratowego
4. Usuń cechę, która ma najmniejszy wpływ na przewidywanie wartości
5. Powtórz kroki 2-4, aż zostanie wybrana określona liczba cech
6. Zwróć wybrane cechy

Metoda Polynomial Features jest metodą, która pozwala na wygenerowanie wielu cech na podstawie istniejących. W tym celu należy podać stopień wielomianu, który ma zostać wygenerowany. Przykładowo: w przypadku stopnia 2 każda cecha zostanie podniesiona do kwadratu, a następnie zostaną wygenerowane wszystkie kombinacje dwóch cech. W przypadku stopnia 3, każda cecha zostanie podniesiona do sześcianu, a następnie zostaną wygenerowane wszystkie kombinacje trzech cech, itd.

Przykład:

- Stopień wielomianu: 2
- Cechy: x_1, x_2
- Wygenerowane cechy: x_1^2, x_2^2, x_1x_2

Cross Validation jest metodą walidacji modelu. Polega ona na podziale zbioru danych na k części. Następnie wykonywane jest k trenowań modelu, w których każde trenowanie wykorzystuje $k-1$ części zbioru danych. W ostatnim trenowaniu wykorzystywana jest pozostała część zbioru danych. W ten sposób wyznaczane są k wartości błędu średniokwadratowego. Następnie obliczana jest średnia z tych wartości.

2.2 Badania symulacyjne

Na początku programu w funkcji *importData()* wczytaliśmy zbiór danych *Concrete Compressive Strength Data Set*. Następnie wykonaliśmy wstępną analizę zbioru za pomocą funkcji *analizeData()*.

2.2.1 Analiza zbioru danych

Na początku wypisaliśmy pierwsze 5 wierszy ze zbioru danych. W tym celu skorzystaliśmy z funkcji *head()* z biblioteki *pandas*.

Wynik:

n	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Concrete compressive strength
1	540	0	0	162	2.5	1040	676	28	79.99
2	540	0	0	162	2.5	1055	676	28	61.89
3	332.5	142.5	0	228	0	932	594	270	40.27
4	332.5	142.5	0	228	0	932	594	365	41.05
5	198.6	132.4	0	192	0	978	825	360	44.3

Pierwsze 5 wierszy zbioru danych

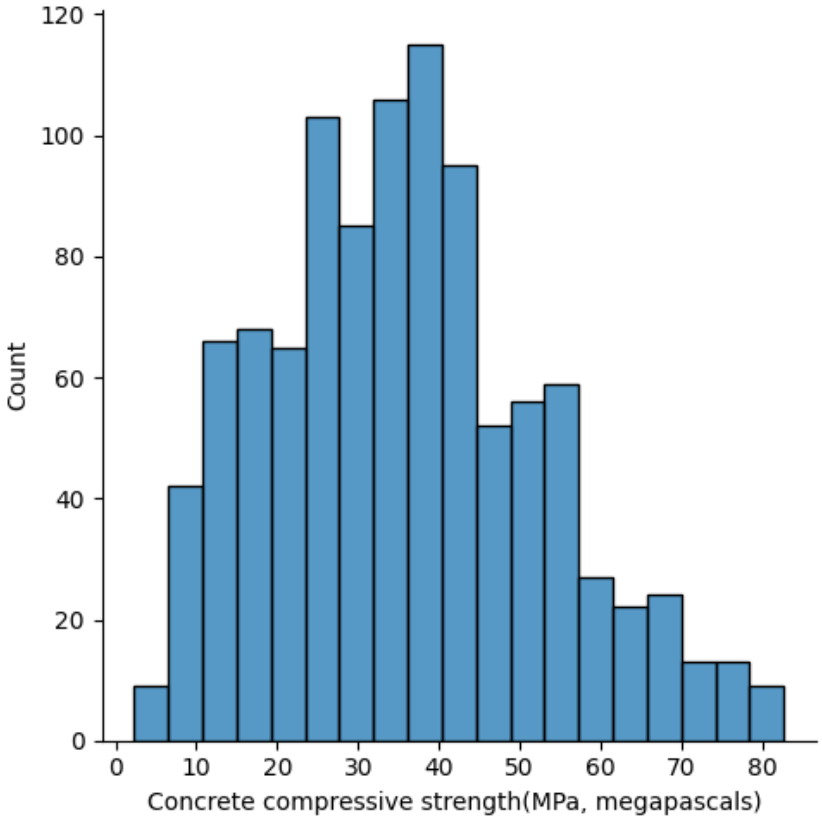
Następnie wyświetliliśmy podstawowe informacje o zbiorze danych. W tym celu skorzystaliśmy z funkcji *describe()*.
W wyniku otrzymaliśmy następujące informacje:

Nazwa kolumny	Min	Max	Odchylenie standardowe	Średnia
Cement	102	540	104.50	281.16
Blast Furnace Slag	0	359	86.27	73.89
Fly Ash	0	200	63.99	54.18
Water	121	247	21.35	181.56
Superplasticizer	0	32	5.97	6.20
Coarse Aggregate	801	1145	77.75	972.91
Fine Aggregate	594	992	77.66	773.58
Age	1	365	63.16	45.66
Concrete compressive strength	2.33	82.6	16.70	35.81

Podstawowe informacje o zbiorze danych uzyskane za pomocą funkcji describe()

Upewniliśmy się również, czy na pewno w zbiorze nie ma brakujących danych. Skorzystaliśmy do tego celu z funkcji *isnull()*, która potwierdziła, że w zbiorze znajdują się wszystkie dane.

Sprawdziliśmy również dystrybucję danych w zbiorze, aby określić czy są one zbalansowane. W tym celu utworzyliśmy wykres typu *distplot()* z biblioteki seaborn. Na wykresie widać, że większość danych znajduje się w przedziale od 0 do 70. Wartości wykraczające poza ten przedział są rzadkie.



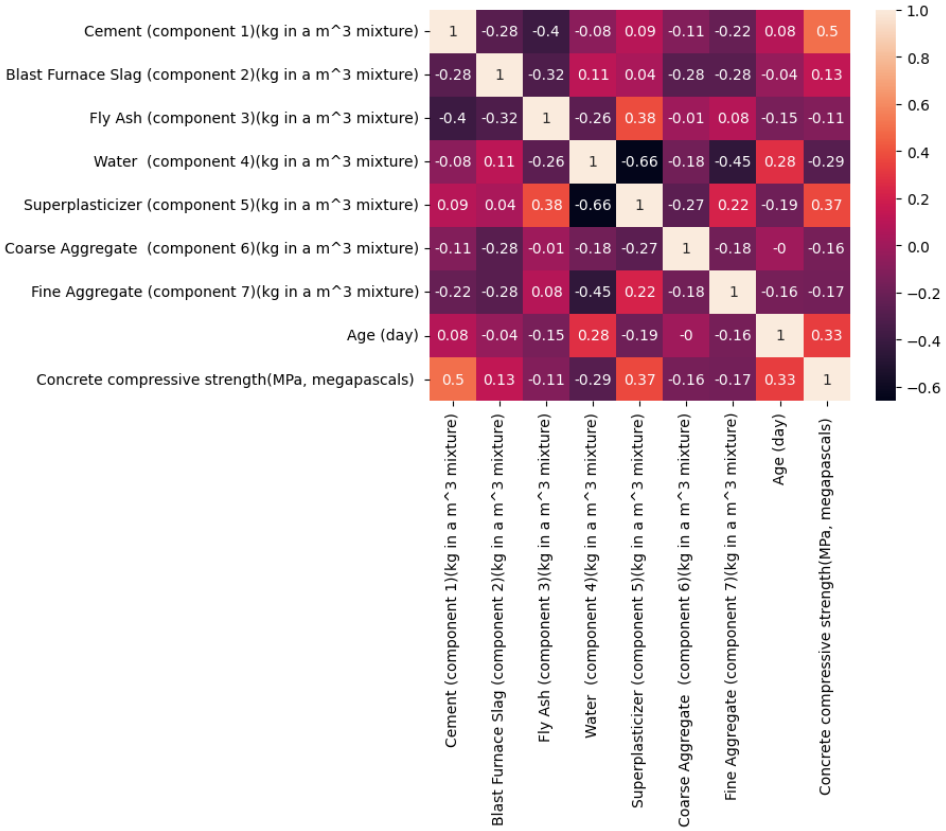
Dystrybucja danych w zbiorze

Za pomocą funkcji *corr()* obliczyliśmy korelację między cechami. W tym celu wykorzystaliśmy bibliotekę pandas. W wyniku otrzymaliśmy następującą macierz korelacji:

	C	B FS	FA	W	SP	CA	FA	A	CCS
Cement	1	-0.28	-0.40	-0.08	0.09	0.011	-0.22	0.08	0.50
Blast Furnace Slag	-0.28	1	-0.32	0.11	-0.04	-0.28	-0.28	0.04	0.13
Fly Ash	-0.40	-0.32	1	-0.26	0.37	0.08	0.15	-0.15	-0.32
Water	-0.08	0.11	-0.26	1	-0.65	-0.18	-0.45	0.27	-0.29
Superplasticizer	0.09	-0.04	0.37	-0.65	1	0.02	0.22	-0.19	-0.16
Coarse Aggregate	0.011	-0.28	0.08	-0.18	0.02	1	-0.18	-0.02	-0.16
Fine Aggregate	-0.22	-0.28	0.15	-0.45	0.22	-0.18	1	-0.16	-0.17
Age	0.08	0.04	-0.15	0.27	-0.19	-0.02	-0.16	1	0.33
Concrete compressive strength	0.50	0.13	-0.32	-0.29	-0.16	-0.16	-0.17	0.33	1

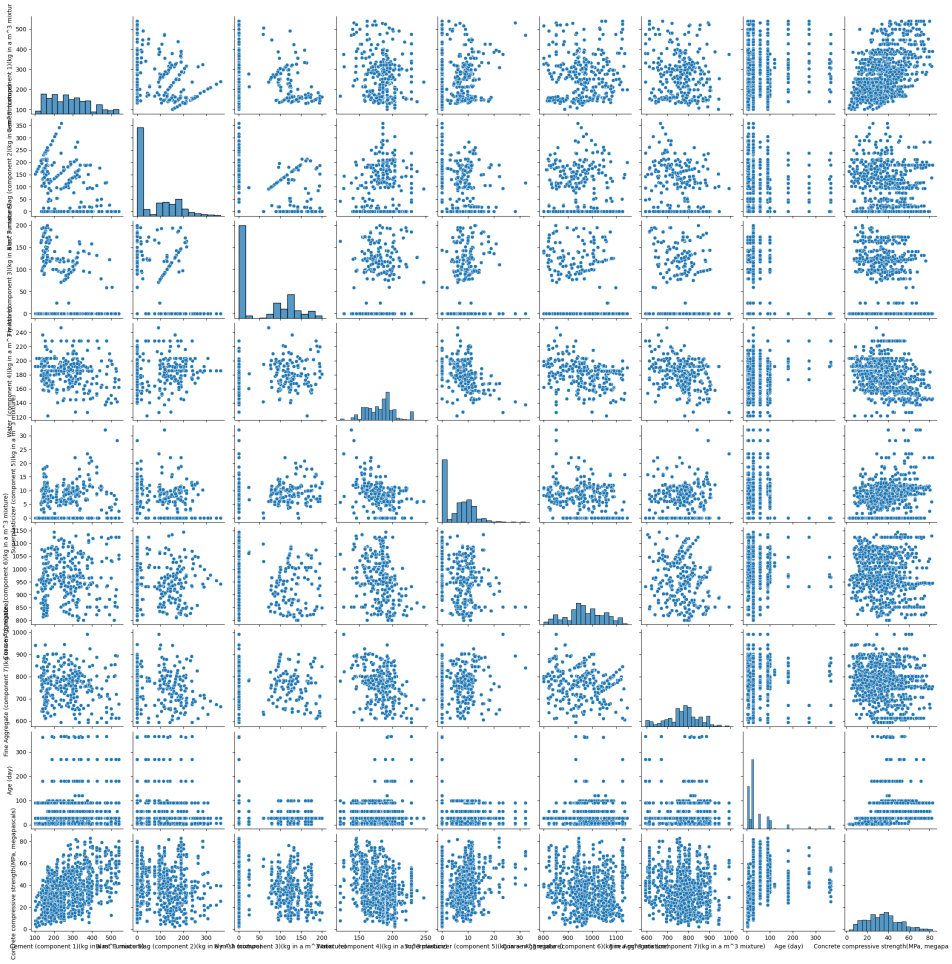
Macierz korelacji

Aby łatwiej odczytać macierz korelacji postanowiliśmy ją wyświetlić również w postaci heatmapy.



Heatmapa macierzy korelacji

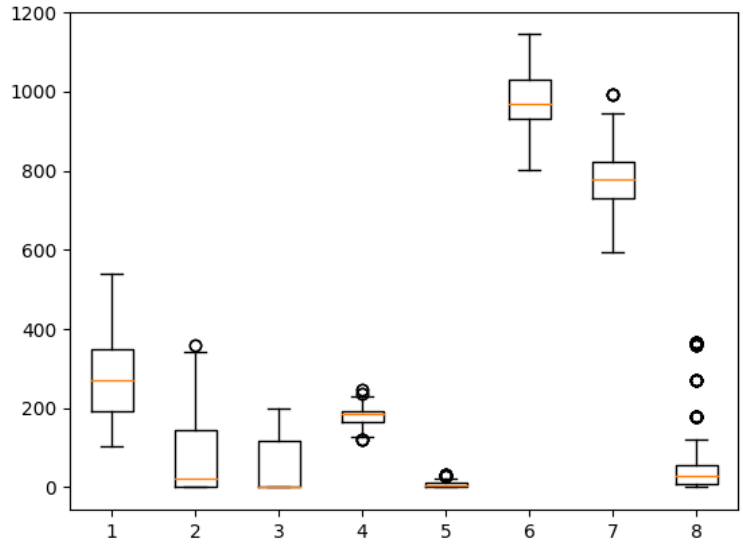
Wykorzystaliśmy również metodę *pairplot()*, aby sprawdzić zależności między cechami.



Wykresy zależności między cechami

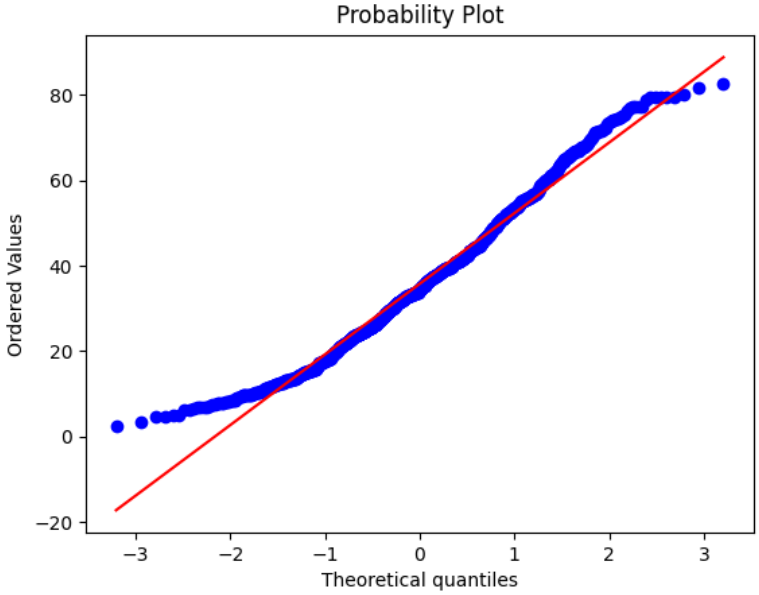
Na podstawie macierzy korelacji, heatmapy oraz wykresu zależności między cechami można stwierdzić, że cechy nie są mocno skorelowane. W naszym przypadku zmienna najbardziej skorelowana z Concrete compressive strength (target) to Cement. Wartość współczynnika korelacji wyniosła 0.5. Warto zwrócić uwagę na cechy Water oraz Superplasticizer, które są silnie negatywnie skorelowane (-0.66)

W celu sprawdzenia, czy w zbiorze występują wartości odstające, wykorzystaliśmy funkcję *boxplot()*. Na wykresie widać, że w zbiorze występują takie wartości w przypadku 7 kolumn.



Wykres typu boxplot

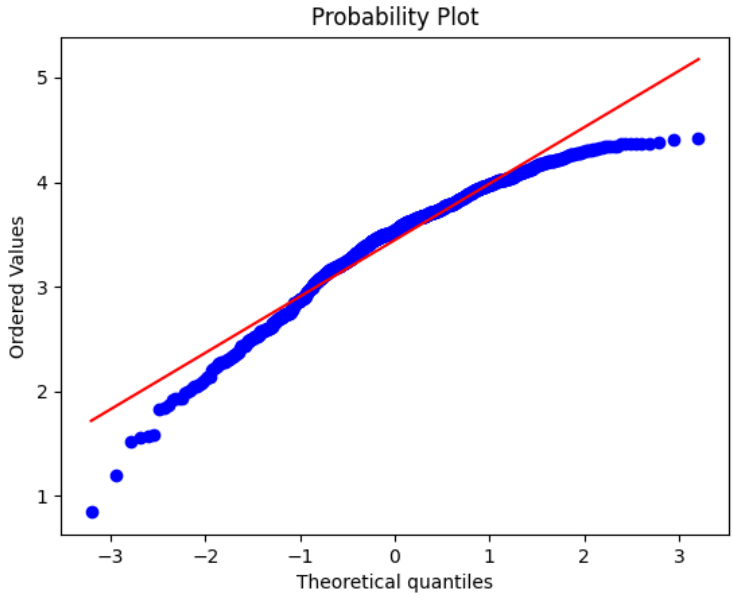
W celu sprawdzenia, czy dane są w rozkładzie normalnym, wykorzystaliśmy funkcję *probplot()*.



Wykres typu probplot dla rozkładu normalnego

Funkcję *probplot()* wykorzystaliśmy również do sprawdzenia, czy dane są w rozkładzie log-normalnym. Jednak ta skala okazała się być gorsza od nor-

malnej (widać to na wykresie poniżej). Można to wywnioskować z tego, że wykres jest bardziej rozproszony, gdzie w przypadku normalnego rozkładu wykres jest bardziej skupiony wokół prostej, co oznacza, że dane powinny być dobrym materiałem do trenowania modeli regresji liniowej.



Wykres typu probplot dla rozkładu log-normalnego

2.2.2 Przygotowanie danych

W celu przygotowania danych do modelowania postanowiliśmy:

- przeskalować dane
- podzielić zbiór na zbiór treningowy i testowy
- wybrać najlepsze cechy
- wygenerować dodatkowe cechy

W celu przeskalowania danych wykorzystaliśmy bibliotekę sklearn. Wykorzystaliśmy klasę *StandardScaler*, która przeskalowuje dane do rozkładu normalnego. W celu przeskalowania danych wykorzystaliśmy funkcję *fit_transform()*.

W celu podziału zbioru na zbiór treningowy i testowy wykorzystaliśmy funkcję *train_test_split()* z biblioteki sklearn. Funkcja ta dzieli zbiór na zbiór treningowy i testowy w podanych proporcjach.

Wybór cech wykonywany jest za pomocą klasy *RFE* z biblioteki sklearn (funkcja *fit()*).

Generowanie dodatkowych cech wykonywane jest za pomocą klasy *PolynomialFeatures* z biblioteki sklearn (funkcja *fit_transform()*).

2.2.3 Schemat symulacji

W celu eksperymentu postanowiliśmy wytrenować modele według następujących reguł:

- trenowanie modeli wykonujemy zarówno na zbiorze oryginalnym jak i na zbiorze po przeskalowaniu
- w każdym przypadku trenujemy 3 rodzaje modeli: za pomocą regresji liniowej, regresji wielomianowej z zastosowaniem Polynomial Features oraz algorytmu Recursive Feature Elimination (RFE)
- każdy rodzaj modelu trenujemy dla różnych rozmiarów zbioru treningowego (20, 30, 40, 50%)
- w przypadku zastosowania Polynomial Features trenowane są modele o stopniu wielomianu 2, 3 oraz 4
- w przypadku zastosowania RFE trenowane są modele dla różnej liczby cech (od 5 do 14), dla stopnia wielomianu 2 (z naszych obserwacji wynika, że stopień wielomianu 2 jest najlepszy)
- dla każdego modelu obliczane są następujące metryki: średni błąd kwadratowy, współczynnik determinacji (R2), waliacja krzyżowa (Cross Validation) dla cv od 2 do 4
- każdy model ma generowaną unikalną nazwę
- dla modeli trenowanych na zbiorze oryginalnym i przeskalowanym wybierany jest po jednym najlepszym modelu (na podstawie wartości średniego błędu kwadratowego, współczynnika determinacji oraz walidacji krzyżowej)
- wartości metryk dla każdego modelu zapisywane są do pliku csv (osobno dla zbioru oryginalnego i przeskalowanego)
- dla każdego modelu generowany jest wykres porównujący wartości rzeczywiste z wartościami przewidywanymi

- wykresy zapisywane są do plików png o nazwie odpowiadającej nazwie modelu

Schemat nazewnictwa modeli:

[skrót_metody]_[parametry_modelu]_[s/ns]

Gdzie:

- skrót_metody - akronim metody użytej do trenowania modelu, są to odpowiednio: LR (dla regresji liniowej), PR (dla regresji wielomianowej), RFE (dla Recursive Feature Elimination)
- parametry_modelu - parametry modelu:
  - s - rozmiar zbioru treningowego
  - d - stopień wielomianu
  - nf - liczba cech
- s/ns - oznaczenie danych (ns - dane oryginalne, s - dane przeskalowane); znajduje się na końcu nazwy modelu

Przykładowe nazwy modeli:

- LR_s_0.2_s - regresja liniowa, 20% zbioru treningowego, zbiór skalowany
- RFE_d_2_nf_6_s_0.5_ns - Recursive Feature Elimination, stopień wielomianu 2, 6 cech, 50% zbioru treningowego, zbiór nieskalowany
- PR_d_3_s_0.3_s - regresja wielomianowa, stopień wielomianu 3, 30% zbioru treningowego, zbiór skalowany

Schemat zapisu metryk do pliku csv:

- nazwa plików to *non-scaled_results.csv* i *scaled_results.csv* odpowiednio dla zbioru oryginalnego i przeskalowanego
- dane modeli zapisywane są w kolejnych wierszach
- nazwy kolumn to:
  - name - nazwa modelu
  - mean - błąd średniokwadratowy
  - variance - wariancja
  - cv2 - walidacja krzyżowa dla cv=2
  - cv3 - walidacja krzyżowa dla cv=3
  - cv4 - walidacja krzyżowa dla cv=4

2.2.4 Wyniki symulacji dla danych bez skalowania

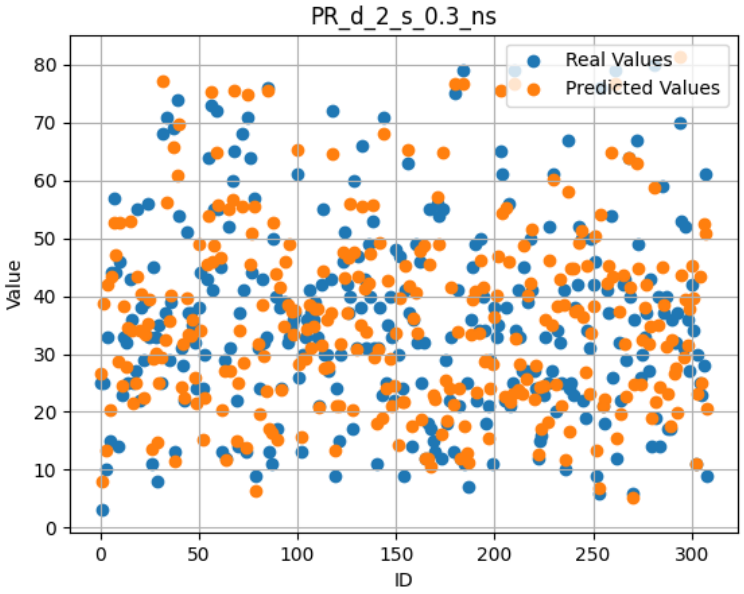
Na podstawie wygenerowanych modeli wybraliśmy trzy najlepsze z nich dla zbioru oryginalnego. Są to:

1. Model regresji wielomianowej stopnia 2 dla 30% zbioru treningowego
2. Model regresji wielomianowej stopnia 2 dla 50% zbioru treningowego
3. Model z wykorzystaniem RFE stopnia 2 o 8 cechach dla 30% zbioru treningowego

Zostały one wybrane na podstawie wartości błędu średniokwadratowego, współczynnika determinacji oraz walidacji krzyżowej. Wszystkie trzy modele osiągnęły bardzo dobre wyniki, wartości błędu średniokwadratowego dla tych modeli były najmniejsze.

Model regresji wielomianowej stopnia 2 dla 30% zbioru treningowego:

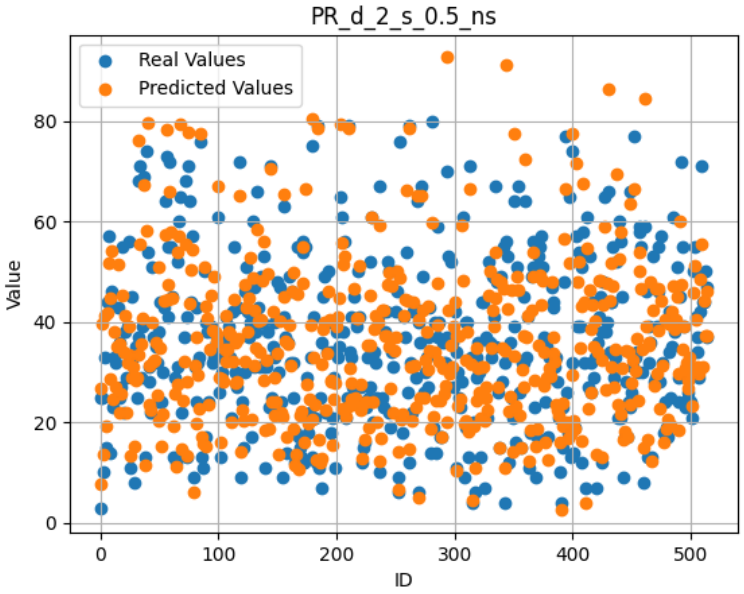
- nazwa modelu: PR_d_2_s_0.3_ns
- błąd średniokwadratowy: 58.945251381737094
- współczynnik determinacji: 0.7887209792098617
- walidacja krzyżowa dla cv=2: 0.5587452765381222
- walidacja krzyżowa dla cv=3: 0.6612126286969477
- walidacja krzyżowa dla cv=4: 0.5905217307658854
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu PR_d_2_s_0.3_ns

Model regresji wielomianowej stopnia 2 dla 50% zbioru treningowego:

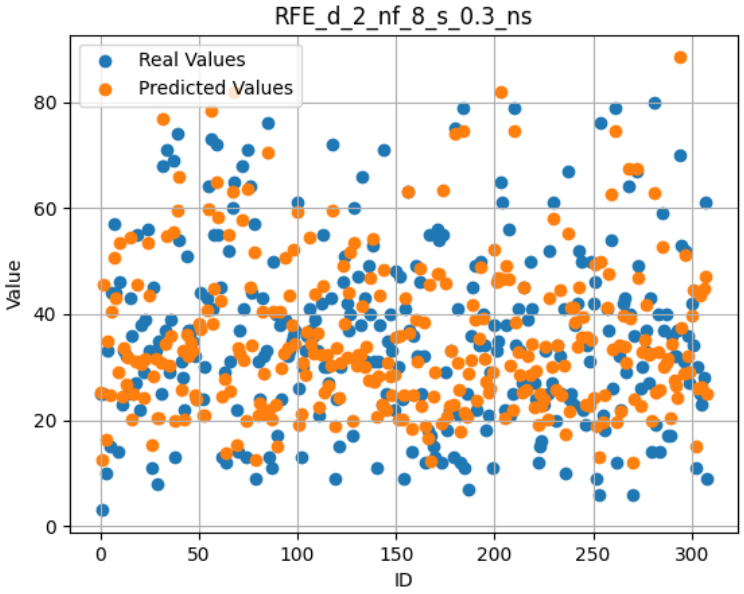
- nazwa modelu: PR_d_2_s_0.5_ns
- błąd średniokwadratowy: 66.36971163392116
- współczynnik determinacji: 0.7599499615396053
- walidacja krzyżowa dla cv=2: 0.6565261166365575
- walidacja krzyżowa dla cv=3: 0.754689342213766
- walidacja krzyżowa dla cv=4: 0.762473153465002
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu PR_d_2_s_0.5_ns

Model z wykorzystaniem RFE stopnia 2 o 8 cechach dla 30% zbioru treningowego:

- nazwa modelu: RFE_d_2_nf_8_s_0.3_ns
- błąd średniokwadratowy: 79.12807052235767
- współczynnik determinacji: 0.7163791677007509
- walidacja krzyżowa dla cv=2: 0.5587452765381222
- walidacja krzyżowa dla cv=3: 0.6612126286969477
- walidacja krzyżowa dla cv=4: 0.5905217307658854
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu RFE_d_2_nf_8_s_0.3_ns

Wszystkie trzy modele mają współczynnik determinacji między 0.71 a 0.78, co oznacza, że modele są dość dobre. Wartości walidacji krzyżowej dla cv o wartości 2, 3 i 4 są zbliżone dla wybranych modeli, co oznacza, że są one stabilne.

2.2.5 Wyniki symulacji dla danych ze skalowaniem

Na podstawie wygenerowanych modeli wybraliśmy trzy najlepsze z nich dla zbioru ze skalowaniem. Są to:

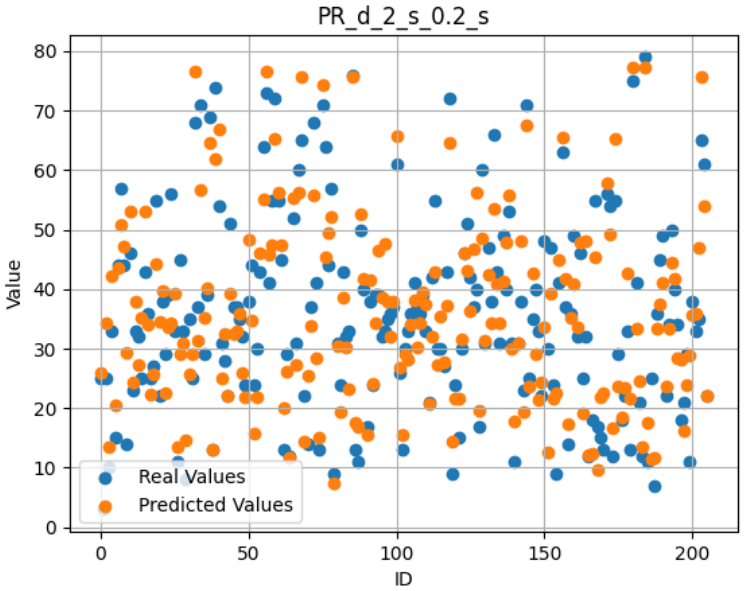
1. Model regresji wielomianowej stopnia 2 dla 20% zbioru treningowego
2. Model regresji wielomianowej stopnia 2 dla 30% zbioru treningowego
3. Model regresji wielomianowej stopnia 2 dla 50% zbioru treningowego

Zostały one wybrane na podstawie tych samych wyznaczników jak w przypadku modeli wygenerowanych bez skalowania zbioru danych.

Model regresji wielomianowej stopnia 2 dla 20% zbioru treningowego:

- nazwa modelu: PR_d_2_s_0.2_s
- błąd średniokwadratowy: 51.0119559547858
- współczynnik determinacji: 0.8174597501290759

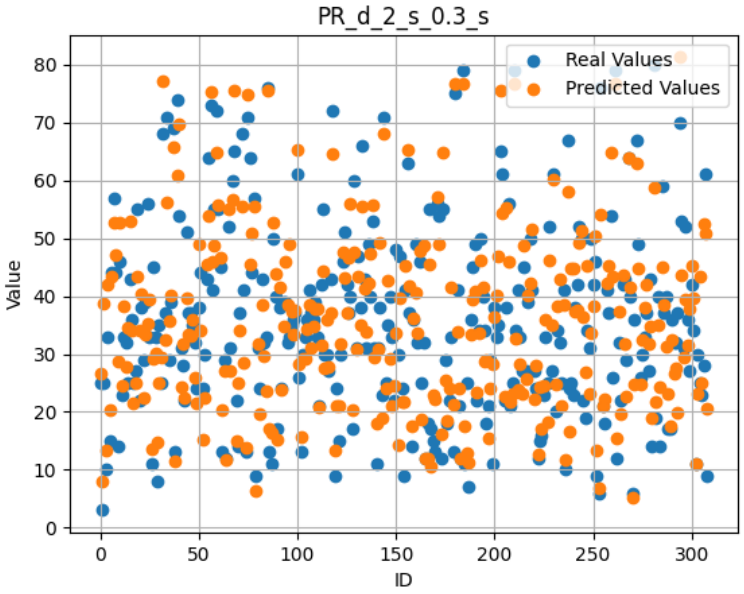
- walidacja krzyżowa dla cv=2: 0.3020929003546536
- walidacja krzyżowa dla cv=3: 0.6135095033450021
- walidacja krzyżowa dla cv=4: 0.7070370517697293
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu PR_d_2_s_0.2_s

Model regresji wielomianowej stopnia 2 dla 30% zbioru treningowego:

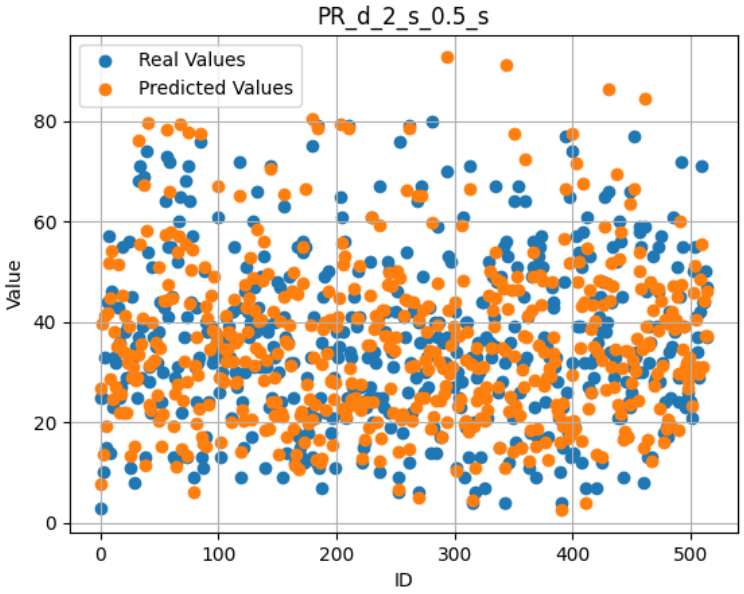
- nazwa modelu: PR_d_2_s_0.3_s
- błąd średniokwadratowy: 58.945251382045186
- współczynnik determinacji: 0.7887209792087575
- walidacja krzyżowa dla cv=2: 0.5837776709109881
- walidacja krzyżowa dla cv=3: 0.6956812085237528
- walidacja krzyżowa dla cv=4: 0.5081744180788107
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu PR_d_2_s_0.3_s

Model regresji wielomianowej stopnia 2 dla 50% zbioru treningowego:

- nazwa modelu: PR_d_2_s_0.5_s
- błąd średniokwadratowy: 66.36971163441056
- współczynnik determinacji: 0.7599499615378352
- walidacja krzyżowa dla cv=2: 0.6987226149686151
- walidacja krzyżowa dla cv=3: 0.7546893422127049
- walidacja krzyżowa dla cv=4: 0.7624731534636663
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu PR_d_2_s_0.5_s

Wszystkie trzy modele mają współczynnik determinacji między 0.75 a 0.82, co oznacza, że modele są dość dobre. Warto zwrócić uwagę na fakt, że w przypadku skalowanych danych wszystkie 3 najlepsze modele są modelami regresji wielomianowej stopnia 2. Różnią się one jedynie wielkością zbioru treningowego. W tym przypadku uzyskano też niższe wartości błędu średniokwadratowego niż w przypadku modeli bez skalowania.

2.2.6 Wyniki symulacji dla modeli regresji liniowej

Ponieważ żaden model regresji liniowej nie był na tyle dobry, aby znaleźć się wśród najlepszych modeli, a tego dotyczyło główne polecenie zadania, to postanowiliśmy wygenerować ranking tylko dla modeli regresji liniowej.

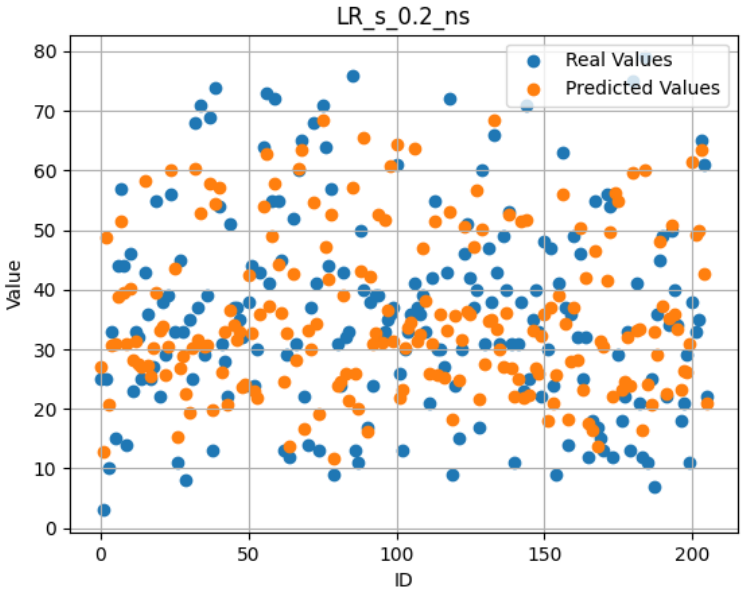
Trzy najlepsze modele dla zbioru bez skalowania danych:

1. Model regresji liniowej dla 20% zbioru treningowego
2. Model regresji liniowej dla 30% zbioru treningowego
3. Model regresji liniowej dla 40% zbioru treningowego

Model regresji liniowej dla 20% zbioru treningowego:

- nazwa modelu: LR_s_0.2_ns
- błąd średniokwadratowy: 100.67526042702907

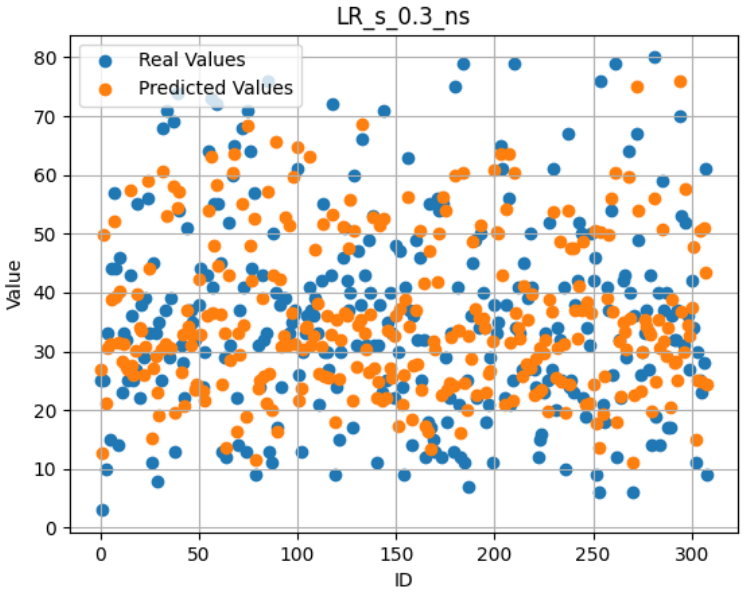
- współczynnik determinacji: 0.6397454900482769
- walidacja krzyżowa dla cv=2: 0.6055998055170613
- walidacja krzyżowa dla cv=3: 0.6017776001163333
- walidacja krzyżowa dla cv=4: 0.5991471569663839
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu LR_s_0.2_ns

Model regresji liniowej dla 30% zbioru treningowego:

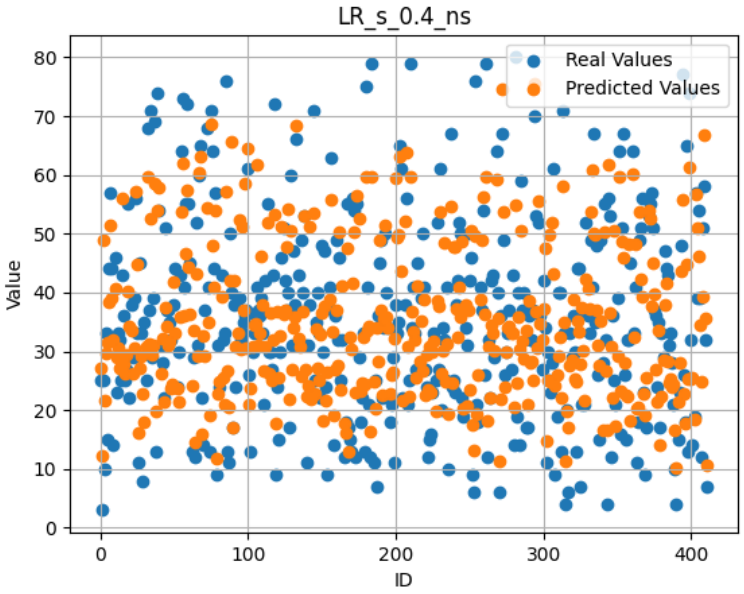
- nazwa modelu: LR_s_0.3_ns
- błąd średniokwadratowy: 105.69685723034375
- współczynnik determinacji: 0.6211479640386942
- walidacja krzyżowa dla cv=2: 0.6033683093006181
- walidacja krzyżowa dla cv=3: 0.6033559121495217
- walidacja krzyżowa dla cv=4: 0.5857841384745779
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu LR_s_0.3_ns

Model regresji liniowej dla 40% zbioru treningowego:

- nazwa modelu: LR_s_0.4_ns
- błąd średniokwadratowy: 106.25022155383516
- współczynnik determinacji: 0.6293974100833089
- walidacja krzyżowa dla cv=2: 0.612296554063505
- walidacja krzyżowa dla cv=3: 0.6192414216103493
- walidacja krzyżowa dla cv=4: 0.6224977912969356
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



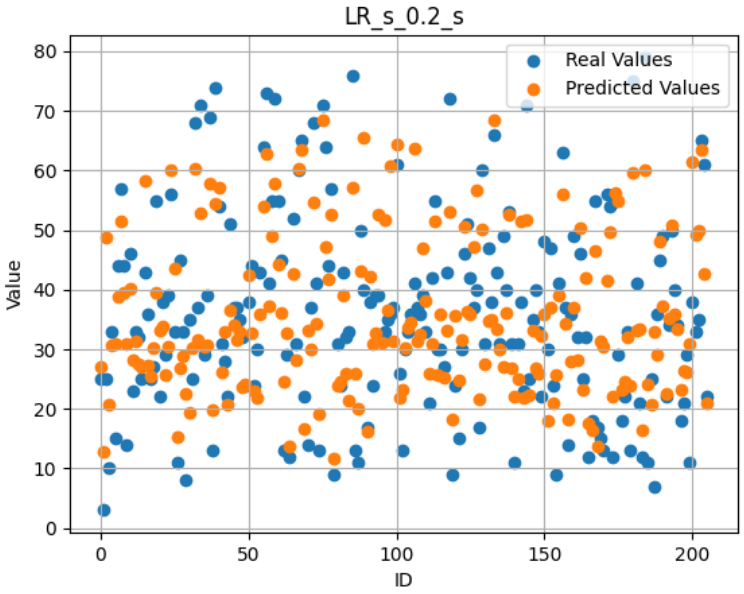
Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu LR_s_0.4_ns

Trzy najlepsze modele regresji liniowej dla zbioru ze skalowaniem danych:

1. Model regresji liniowej dla 20% zbioru treningowego
2. Model regresji liniowej dla 30% zbioru treningowego
3. Model regresji liniowej dla 40% zbioru treningowego

Model regresji liniowej dla 20% zbioru treningowego:

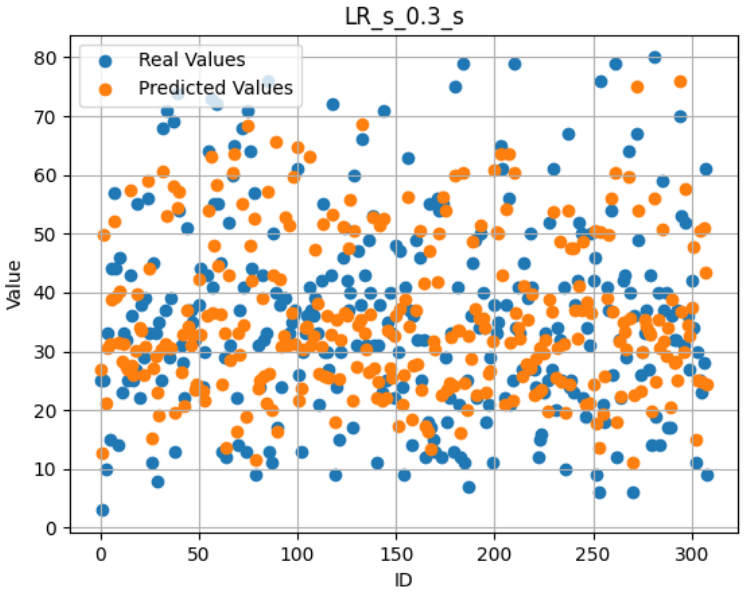
- nazwa modelu: LR_s_0.2_s
- błąd średniokwadratowy: 100.67526042702907
- współczynnik determinacji: 0.6397454900482769
- walidacja krzyżowa dla cv=2: 0.6055998055170614
- walidacja krzyżowa dla cv=3: 0.6017776001163332
- walidacja krzyżowa dla cv=4: 0.599147156966384
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu LR_s_0.2_s

Model regresji liniowej dla 30% zbioru treningowego:

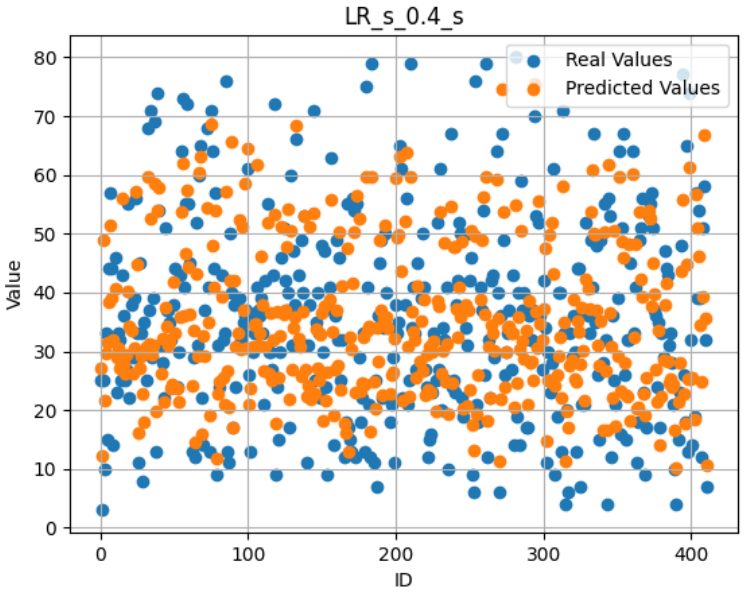
- nazwa modelu: LR_s_0.3_s
- błąd średniokwadratowy: 105.69685723034382
- współczynnik determinacji: 0.621147964038694
- walidacja krzyżowa dla cv=2: 0.6033683093006181
- walidacja krzyżowa dla cv=3: 0.6033559121495217
- walidacja krzyżowa dla cv=4: 0.5857841384745779
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu LR_s_0.3_s

Model regresji liniowej dla 40% zbioru treningowego:

- nazwa modelu: LR_s_0.4_s
- błąd średniokwadratowy: 106.25022155383522
- współczynnik determinacji: 0.6293974100833086
- walidacja krzyżowa dla cv=2: 0.6122965540635052
- walidacja krzyżowa dla cv=3: 0.6192414216103493
- walidacja krzyżowa dla cv=4: 0.6224977912969356
- wykres porównujący wartości rzeczywiste z wartościami przewidywanymi:



Wykres porównujący wartości rzeczywiste z przewidywanymi dla modelu LR_s_0.4_s

Powyższe modele uzyskały podobne wartości błędu średniokwadratowego, który w każdym przypadku wyniósł wartość większą niż 100. Wartość współczynnika determinacji również nie jest zadowalająca, ponieważ w każdym przypadku jest mniejsza niż 0.7.
Warto zwrócić uwagę na fakt, że w przypadku tych modeli zwiększenie rozmiaru zbioru treningowego nie wpłynęło na poprawę jakości modeli.

2.2.7 Porównanie wyników uzyskanych przez modele

Po porównaniu metryk uzyskanych przez najlepsze modele dla zbioru bez skalowania i ze skalowaniem można stwierdzić, że modele dla zbioru ze skalowaniem są lepsze. Uzyskano znacznie niższe wartości błędu średniokwadratowego, gdzie jednocześnie wartości współczynnika determinacji są minimalnie wyższe. Warto zwrócić uwagę na fakt, że 5 na 6 najlepszych modeli stanowią modele regresji wielomianowej stopnia 2. Niestety nie udało się uzyskać modelu regresji liniowej, który dorównałby tym modelom.

W przypadku najlepszych modeli regresji liniowej wyznaczone metryki, tj. błąd średniokwadratowy i współczynnik determinacji, są bardzo zbliżone. Można więc stwierdzić, że w przypadku regresji liniowej nie ma znaczącej różnicy między skalowaniem i nie skalowaniem danych. W porównaniu do regresji wielomianowej stopnia 2, regresja liniowa jest znacznie gorsza (dla badanego zbioru danych). Widać to zwłaszcza po uzyskanych wartościach współczynnika determinacji.

2.2.8 Uwagi

W wielu przypadkach po wyznaczeniu walidacji krzyżowej dla cv od 2 do 4 otrzymywaliśmy ujemne wartości. Możliwe, że jest to związane z tym, iż w danym przypadku model był zbyt złożony lub zbyt prosty dla wybranych danych. Skutkuje to tym, że model nie jest w stanie dobrze przewidywać wartości dla zbioru testowego.

Rozdział 3

Podsumowanie

W pracy zostały przedstawione wyniki badań nad trenowaniem modeli regresji dla zbioru danych dotyczącego odporności betonu na ściskanie. W sumie wygenerowano i porównano 112 modeli, wykorzystując metody regresji liniowej, wielomianowej z zastosowaniem Polynomial Features oraz algorytmu Recursive Feature Elimination.
Modele zostały porównane na podstawie wartości błędu średniokwadratowego oraz współczynnika determinacji. Najlepsze wyniki uzyskano dla modeli regresji wielomianowej stopnia 2. Żaden model regresji liniowej nie był na tyle dobry, aby dorównać modelom regresji wielomianowej.
Najlepszy model regresji liniowej uzyskał wartość współczynnika determinacji równą zaledwie 0.63, gdzie najlepszy model regresji wielomianowej stopnia 2 uzyskał wartość współczynnika determinacji równą 0.81. Jest to znaczna różnica, dlatego w przypadku tego zbioru danych lepiej rozważyć inną metodę uczenia maszynowego niż algorytm regresji liniowej.
Modele trenowane na podstawie skalowanych danych uzyskały lepsze wyniki niż modele trenowane na podstawie nie skalowanych danych, co było spodziewanym rezultatem. Różnice te nie były aż tak widoczne w przypadku modeli regresji liniowej, jednak mimo wszystko warto było przeprowadzić skalowanie danych.
Ciekawym zjawiskiem było to, że część modeli uzyskała ujemne wartości po przeprowadzeniu walidacji krzyżowej. Skutkowało to tym, że model nie był w stanie dobrze przewidywać wartości dla zbioru testowego.

Dodatek A

Dane modeli

A.1 Pliki CSV z metrykami modeli

Pliki CSV z metrykami modeli znajdują się w folerze `załączniki`. Nazwy plików to non_scaled_results.csv i scaled_results.csv.

A.2 Wygenerowane wykresy porównujące wartości rzeczywiste z przewidywanymi

Wszystkie wykresy wygenerowane dla modeli znajdują się w folderze `graphics/plots`.

Dodatek B

Kod programu

Link do repozytorium: KWD-Concrete

```
import csv
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
import seaborn as sb
from scipy import stats
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFE


def importData():
    df = pd.read_excel(r'Concrete_Data.xls')
    pd.set_option('display.max_columns', None)
    print("Dataframe description:")
    print(df.describe())
    print("Dataframe head:")
    print(df.head())
    data = df.iloc[1:, 0:-1]
    target = df.iloc[1:, -1]

    return data, target, df
```

```
def analizeData(target, df, data_np):
    print("Value counts:")
    print(target.value_counts())
    sb.displot(target)
    plt.savefig("plots/target_distribution.png")
    plt.show()
    stats.probplot(target,
                   plot=plt) # Note: The target variable is not normally distributed
    plt.savefig("plots/probplot.png")
    plt.show()
    targetLog = np.log(target)
    stats.probplot(targetLog, plot=plt)  # Note: log transformation seems to be wors
    plt.savefig("plots/log_probplot.png")
    plt.show()
    sb.pairplot(df)
    plt.savefig("plots/pairplot.png")
    plt.show()
    print("HeatMap")
    createHeatMap(df)
    print("Number of NULL values in each column:")
    print(df.isnull().sum())
    plt.boxplot(data_np)
    plt.savefig("plots/boxplot.png")
    plt.show()
    print('---Mean---')
    print(data_np.mean(axis=0))
    print('--std---')
    print(data_np.std(axis=0))


def createHeatMap(df):
    correlation_matrix = df.corr().round(2)
    print("Correlation matrix:")
    print(correlation_matrix)
    sb.heatmap(data=correlation_matrix,
               annot=True)  # Note: The strongest correlation is between cement and
    # Note: the heatmap confirms the results from the pairplot
    plt.savefig("plots/heatmap.png", bbox_inches='tight')
    plt.show()


def convertToNpArray(data, target):
```

```
    data_np = np.array(data, dtype=np.int16)
    target_np = np.array(target, dtype=np.int16)

    print(data_np.shape)
    print(target_np.shape)
    return data_np, target_np


def scaleData(data):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)
    print(scaled_data[1, :])
    print('---Mean---')
    print(scaled_data.mean(axis=0))
    print('--std---')
    print(scaled_data.std(axis=0))
    return scaled_data


def splitData(data, target, size):
    train_data, test_data, \
    train_target, test_target = \
        train_test_split(data, target, test_size=size, random_state=10)

    print("Training dataset:")
    print("train_data:", train_data.shape)
    print("train_target:", train_target.shape)
    print("Testing dataset:")
    print("test_data:", test_data.shape)
    print("test_target:", test_target.shape)
    return train_data, test_data, train_target, test_target


def trainLinearRegressionModel(train_data, train_target):
    linear_regression = LinearRegression()
    linear_regression.fit(train_data, train_target)
    return linear_regression


def getPredictedValues(linear_regression, test_data, test_target):
    realValues = []
    predictedValues = []
    ids = []
    for x in range(0, len(test_data)):
```

```
        prediction = linear_regression.predict(test_data[x, :].reshape(1, -1))
        realValues.append(test_target[x])
        predictedValues.append(prediction)
        ids.append(x)

    return realValues, predictedValues, ids


def createPlot(arr1, arr2, title):
    plt.scatter(arr1['ids'], arr1['arr'], label=arr1['label'])
    plt.scatter(arr2['ids'], arr2['arr'], label=arr2['label'])
    plt.title(title)
    plt.xlabel('ID')
    plt.ylabel('Value')
    plt.legend()
    plt.grid(True)
    plt.savefig("plots/" + title + ".png")
    plt.show()


def prepDict(arr, ids, label):
    return {
        'arr': arr,
        'ids': ids,
        'label': label
    }


def crossValidate(data, target, cv):
    scores = cross_val_score(LinearRegression(), data, target, cv=cv)
    print(scores)
    print(scores.mean())
    return scores.mean()


def polynomialFeatures(trainData, testData, trainTarget, degree):
    poly = PolynomialFeatures(degree)
    trainDataPoly = poly.fit_transform(trainData)
    testDataPoly = poly.fit_transform(testData)
    lr_poly = LinearRegression()
    lr_poly.fit(trainDataPoly, trainTarget)
    return lr_poly, trainDataPoly, testDataPoly
```

```
def reduceFeatures(trainData, testData, trainTarget, degree,
                   numOfFeatures):  # Note: degree = 1 -> without polynomial feature
    lr_poly, trainDataPoly, testDataPoly = polynomialFeatures(trainData, testData, t
    rfe = RFE(lr_poly, n_features_to_select=numOfFeatures)
    rfe = rfe.fit(trainDataPoly, trainTarget)
    print(rfe.support_)
    print(rfe.ranking_)
    return rfe, trainDataPoly, testDataPoly


def validateModel(data, target, model, name):
    realValues, predictedValues, ids = getPredictedValues(model, data, target)
    print("Mean squared error of a learned model: ", mean_squared_error(realValues,
    print('Variance score: ', r2_score(realValues, predictedValues))
    cv = []
    for x in range(2, 5):
        cv.append({
            'cv': x,
            'score': crossValidate(data, target, x)
        })
    realValsDict = prepDict(realValues, ids, "Real Values")
    predictedValsDict = prepDict(predictedValues, ids, "Predicted Values")
    createPlot(realValsDict, predictedValsDict, name)
    return {
        'mean': mean_squared_error(realValues, predictedValues),
        'variance': r2_score(realValues, predictedValues),
        'cv': cv
    }


def trainModels(data, target, postfix):
    dataSizes = [0.2, 0.3, 0.4, 0.5]

    results = []
    for size in dataSizes:
        models = []
        train_data, test_data, train_target, test_target = splitData(data, target, s

        lrModel = trainLinearRegressionModel(train_data, train_target)
        models.append({
            'model': lrModel,
            'testData': test_data,
            'dataSize': size,
            'name': 'LR_s_' + str(size) + "_" + postfix,
```

```
            'type': 'LR'
        })

        for degree in range(2, 5):
            lrPolyModel, trainDataPoly, testDataPoly = polynomialFeatures(train_data
            models.append({
                'model': lrPolyModel,
                'testData': testDataPoly,
                'degree': degree,
                'dataSize': size,
                'name': 'PR_d_' + str(degree) + "_s_" + str(size) + "_" + postfix,
                'type': 'PR'
            })

        # Note: degree = 2 -> the best model for this dataset
        for numOfFeatures in range(5, 15):
            rfe, trainDataPoly, testDataPoly = reduceFeatures(train_data, test_data
            models.append({
                'model': rfe,
                'testData': testDataPoly,
                'numOfFeatures': numOfFeatures,
                'dataSize': size,
                'name': 'RFE_d_2_nf_' + str(numOfFeatures) + "_s_" + str(size) + "_
                'type': 'RFE'
            })

        for model in models:
            print(model['name'])
            results.append({
                'model': model,
                'validation': validateModel(model['testData'], test_target, model['
            })

    return results


def printModelsResults(results):
    for result in results:
        print(result['model']['name'])
        print("Mean squared error: ", result['validation']['mean'])
        print("Variance score: ", result['validation']['variance'])
        print("Cross Validation: ")
        for cv in result['validation']['cv']:
            print("CV: ", cv['cv'], " Score: ", cv['score'])
```

```
        print("")


def saveModelsResultsToCSV(results, filename):
    with open(filename, 'w') as csvfile:
        fieldnames = ['name', 'mean', 'variance', 'cv2', 'cv3', 'cv4']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for result in results:
            writer.writerow({
                'name': result['model']['name'],
                'mean': result['validation']['mean'],
                'variance': result['validation']['variance'],
                'cv2': result['validation']['cv'][0]['score'],
                'cv3': result['validation']['cv'][1]['score'],
                'cv4': result['validation']['cv'][2]['score']
            })


def chooseBestModel(results):
    bestModel = results[0]
    for result in results:
        if result['validation']['mean'] >= bestModel['validation']['mean']:
            continue
        if result['validation']['variance'] <= bestModel['validation']['variance']:
            continue
        if result['validation']['cv'][0]['score'] <= 0:
            continue
        if result['validation']['cv'][1]['score'] <= 0:
            continue
        if result['validation']['cv'][2]['score'] <= 0:
            continue
        bestModel = result
    return bestModel


def getNBestModels(results, n):
    bestModels = []
    for i in range(0, n):
        bestModel = chooseBestModel(results)
        bestModels.append(bestModel)
        results.remove(bestModel)
    return bestModels
```

```
def getNBestLRModels(results, n):
    lrModels = []
    for result in results:
        if result['model']['type'] == 'LR':
            lrModels.append(result)
    bestModels = []
    for i in range(0, n):
        bestModel = chooseBestModel(lrModels)
        if bestModel['model']['type'] != 'LR':
            continue
        bestModels.append(bestModel)
        lrModels.remove(bestModel)
    return bestModels


def init():
    data, target, df = importData()
    data_np, target_np = convertToNpArray(data, target)
    analizeData(target, df, data_np)

    results = trainModels(data_np, target_np, 'ns')
    scaledResults = trainModels(scaleData(data_np), target_np, 's')

    print("Results:")
    print(results)
    print("Scaled results:")
    print(scaledResults)
    saveModelsResultsToCSV(results, 'non-scaled_results.csv')
    saveModelsResultsToCSV(scaledResults, 'scaled_results.csv')
    printModelsResults(results)
    printModelsResults(scaledResults)

    bestModel = chooseBestModel(results)
    print("Best model for non-scaled data: ", bestModel['model']['name'])
    printModelsResults([bestModel])

    bestModel = chooseBestModel(scaledResults)
    print("Best model for scaled data: ", bestModel['model']['name'])
    printModelsResults([bestModel])

    bestModelsScaled = getNBestModels(scaledResults, 3)
    print("Best models for scaled data: ")
    printModelsResults(bestModelsScaled)
```

```
    bestModelsNonScaled = getNBestModels(results, 3)
    print("Best models for non-scaled data: ")
    printModelsResults(bestModelsNonScaled)

    bestLRModelsScaled = getNBestLRModels(scaledResults, 3)
    print("Best LR models for scaled data: ")
    printModelsResults(bestLRModelsScaled)
    bestLRModelsNonScaled = getNBestLRModels(results, 3)
    print("Best LR models for non-scaled data: ")
    printModelsResults(bestLRModelsNonScaled)

init()
```