Juliusz Wasieleski
Informatyka, III rok, grupa 6
grudzień 2023

Podstawy uczenia macierzowego – laboratorium nr 1 – sprawozdanie

1. Opis ćwiczenia

Naszym zadaniem było zrealizowanie trzech zadań:

1. Dla klasyfikacji binarnej
   1.1 Generujemy dataset to znaczy jakąś macierz na której znajdziemy krzywą dzielącą to wsztko
   1.2 Znajdujemy tę krzywą przy użyciu:
      a) regresji liniowej
      b) regresji logistycznej
2. Dla regresji
   2.1 Generujemy dataset to znaczy punkty ułożone na jakiejś krzywej z biasem
   2.2 Rozwiązujemy problem kilkoma modelami: krzywą liniową, pierwszego stopnia i drugiego stopnia
   2.3 Wybieramy najlepszy z tych modeli na podstawie validacji (albo k-cross, albo Train-Validation-Test) z metryką Minimal Square Error
3. Problem prędkości średniej
   Jako, że w nim wychodzi nam funkcja dwóch zmiennych, powinniśmy stworzyć model znajdujący tą krzywą

2. Środowisko, biblioteki, założenia oraz użyte narzędzia

Ćwiczenie wykonaliśmy w języku Python przy użyciu Jupyer Notebooka. Do obliczeń, przechowywania danych użyliśmy bibliotek *numpy, pandas, random, sklearn, math.*

Do rysowania wykresów użyliśmy biblioteki *matplotlib.*

Wszystkie obliczenia prowadziliśmy na komputerze Lenovo Y50-70 z systemem Windows 10 Pro w wersji 10.0.19045, procesor Intel Core i7-4720HQ 2.60GHz, 2601 MHz, rdzenie: 4, procesory logiczne: 8.

3. Rozwiązanie

3.1 Klasyfikacja binarna

3.1.1 Generacja danych

Dane do klasyfikacji binarnej generowałem zgodnie ze wzorem:

$$z(x,y) = \begin{cases} -1\ dla\ \sin\left(\frac{x}{5} - 1\right) > \frac{y}{5} - 1 \\ 1\ dla\ s\sin\left(\frac{x}{5} - 1\right) \leq \frac{y}{5} - 1 \end{cases} \quad (1)$$

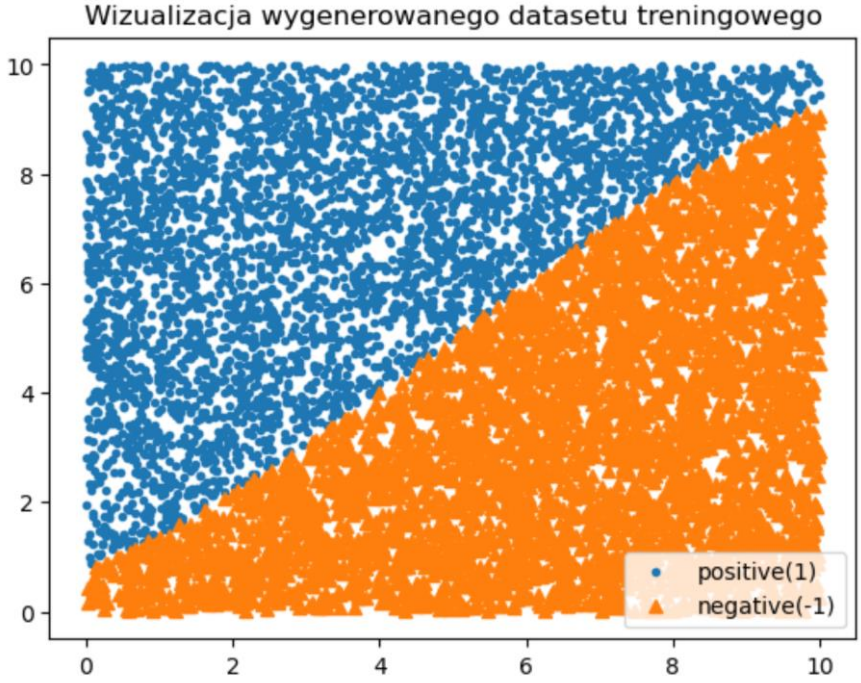
Przyjąłem, że zmienne x i y są losowane z przedziałów:

$$x \in \langle 0{,}10 \rangle$$
$$y \in \langle 0{,}10 \rangle \quad (2)$$

Generowałem dwa datasety:

- Treningowy o liczności 8000
- Testowy o liczności 2000

Wygenerowana funkcja wyglądała w poniższy sposób:



Rys. 1 Wizualizacja wygenerowanego zbioru treningowego

3.1.2 Trening modelu rozwiązującego problem klasyfikacji binarnej

W celu znalezienia modelu radzącego sobie najlepiej z problemem klasyfikacji binarnej sprawdziłem działanie trzech modeli z pakietu *sklearn.linear_model*:

- **Lasso** – model regesji liniowej z regularyzacją zgodnie ze wzorem:

$$L_{LASSO}(y,\hat{y}) = \frac{1}{n}(y - \hat{y})^2 + \alpha\|w\| \quad (3)$$

gdzie α to hiperparametr mówiący o sile regularyzacji

- **Ridge (grzbietowa)** – model regesji liniowej z regularyzacją zgodnie ze wzorem:

$$L_{ridge}(y, \hat{y}) = \frac{1}{n}(y - \hat{y})^2 + \lambda \|w\|^2 \tag{4}$$

gdzie λ to hiperparametr mówiący o sile regularyzacji

- **LogisticRegression** – model regresji logistycznej zgodnie ze wzorem:

$$\hat{y} = \frac{1}{1 + e^{-Xw}} \tag{5}$$

Po wytrenowaniu powyższych modeli na zbiorze treningowym dokonałem sprawdzenia ich poprawności na zbiorze testowym. Poprawność działania modeli mierzyłem funkcją *f1_score* z pakietu *sklearn.metrics* która implementuje wzór:

$$F1 = \frac{2 * precission * recall}{precission + recall} \tag{6}$$

gdzie:

$$precission = \frac{TP}{TP + FP}$$
$$recall = \frac{TP}{TP + FN} \tag{7}$$

gdzie:

- TP to True Positive (Prawdziwa wartość to True przewidziana to True)
- FP to False Positive (Prawdziwa wartość to False przewidziana to True)
- FN to False Negative (Prawdziwa wartość to True przewidziana to False)
-

Zgodnie z tą metryką otrzymaliśmy wyniki dla modeli:

Model	Wynik metryki F1
Lasso	0.918
Ridge	0.972
LogisticRegression	0.989

Tab1. Wyniki pomiarów poprawności dla modeli

Analizując tabelę nr 1 łatwo zauważyć, że największą dokładność miał model korzystający z regresji logistycznej. Podobny wynik miała regresja liniowa z regularyzacją grzbietową (ridge regularisation). Warto jednakże zauważyć, że ogólnie wyniki są i tak bardzo dobre, bo maksymalna wartość metryki wynosi 1.

3.2 Regresja

3.2.1 Generacja danych

Dane do problemu regresji generowałem zgodnie ze wzorem:

$$f(x) = 10\left(\left(\frac{x}{5} - 0.9\right)^5 - \left(\frac{x}{5} - 0.9\right)^3 + \left(\frac{x}{5} - 0.9\right)^2\right) \tag{8}$$

$$y(x) = Losowy\ wybór\ ze\ 100\ próbek\ rozkładu\ normalnego\ o\ wartości\ oczekiwanej\ równej\ f(x)\ i\ wariancji\ równej\ 0.5$$

Jako, że powyższy opis może być niejasny zamieszczam realizację w kodzie:

```
def foo_to_predict(x, sigma = 0.5):
    foo = lambda x: 10*(((0.2*x-0.9)**5) - ((0.2*x-0.9)**3)  + ((0.2*x - 0.9)**2))
    return random.choice(np.random.normal(foo(x), sigma, 100))
```

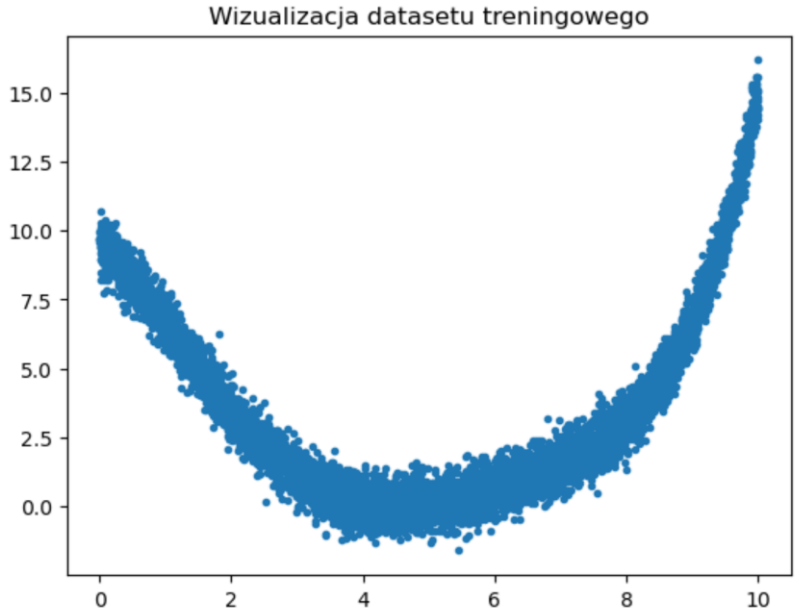
Przyjąłem, że zmienna x jest losowana z przedziału:

$$x \in \langle 0{,}10 \rangle$$

Generowałem dwa datasety:

- Treningowy o liczności 8000
- Testowy o liczności 2000

Wygenerowana funkcja wyglądała w poniższy sposób:

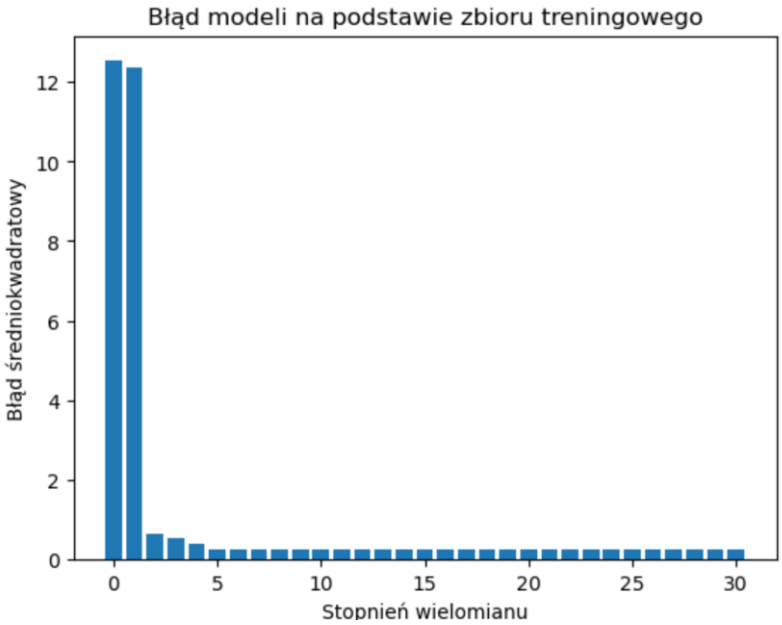


Rys. 2 Wizualizacja wygenerowanego zbioru treningowego

3.2.2 Poszukiwanie najlepszego modelu

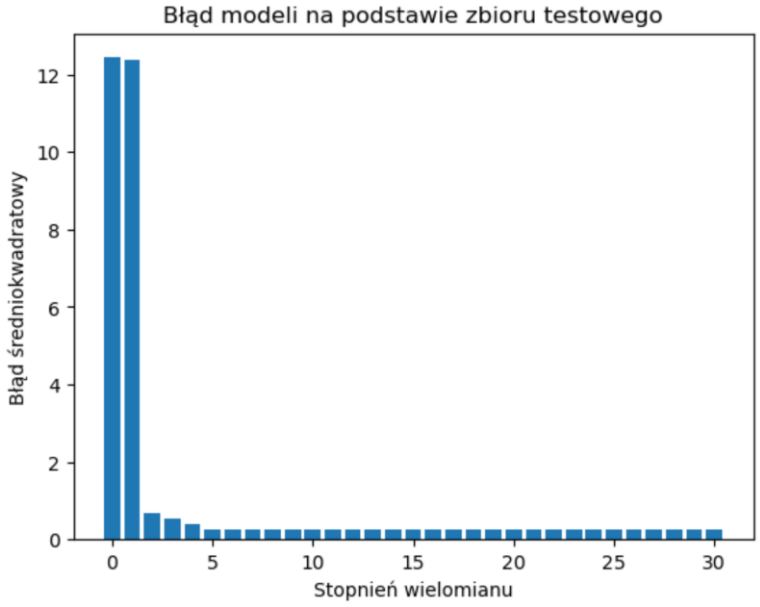
Najlepszego modelu szukałem wśród modeli regresji wielomianowych o stopniach z przedziału $\langle 1{,}30 \rangle$ korzystając z funkcji *polynomial* z pakietu *numpy.polynomial*. Funkcja ta nie zawiera wbudowanego mechanizmu walidacji ani regularyzacji, zatem w podpunkcie 3.2 jedyna walidacja będzie przy użyciu podziału na zbiór testowy i treningowy. Walidację każdego modelu dokonywałem na zbiorze treningowym oraz testowym mierząc błąd średniokwadratowy (MSE –mean square error) zgodnie ze wzorem:

$$MSE(y, \hat{y}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{9}$$



Rys.3 Błąd modeli na podstawie zbioru treningowego

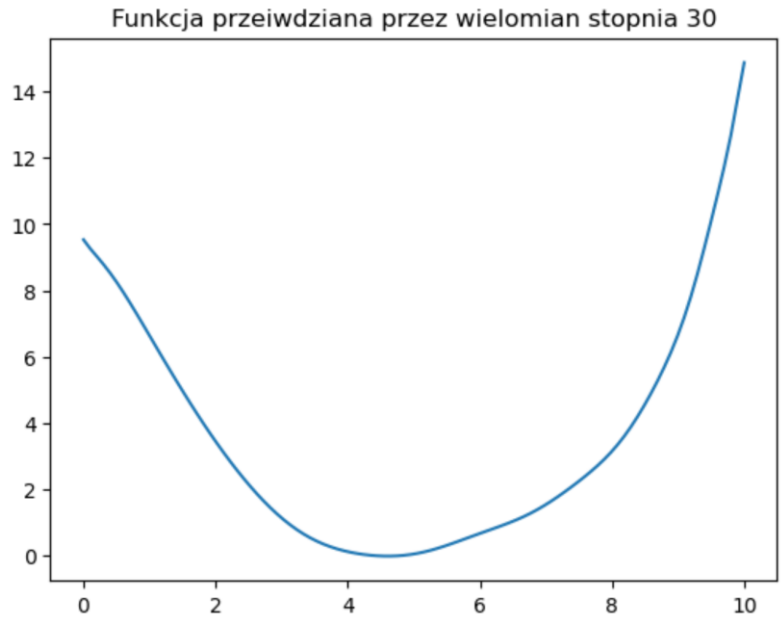
Powyższy efekt tzn. poprawienie modelu wraz z zwiększeniem stopnia wielomianu może znaczyć dwie rzeczy. Albo rzeczywiście jest coraz lepiej wraz ze wzrostem stopnia wielomianu, albo mamy do czynienia z overfittingiem czyli dopasowaniem do tych konkretnych danych.



Rys.4 Błąd modeli na podstawie zbioru testowego

Na szczęście analizując błąd zbioru testowego widzimy że wraz ze wzrostem stopnia wielomianu spada nam błąd. Zatem powinniśmy wybrać dość duży stopnień wielomianu i będzie dobrze. Jest tak ponieważ dataset treningowy jest na tyle duży, że można dobrze go wytrenować a przy tym możliwe że ta funkcja jest dość łatwa do przewidywania.

Jako ciekawostkę zamieszczam poniżej wykres funkcji przewidzianej przez model o stopniu równym 30:



Rys.5 Funkcja przewidziana przez wielomian stopnia 30

3.2.3 Znowu generujemy dane

Jako, że w podpunkcie 3.2.2 okazało się, że dane wygenrowane w podpunkcie 3.2.1 są mało podatne na overfitting i zbiory walidacyjne są zbędne, inspirując się notebookiem znalezionym na stronie https://www.kaggle.com/code/markow/validation-set-approach-and-polynomial-regression postanowiłem jeszcze raz wygenerować dane, tym razem zgodnie z inną funkcją, tak aby pokazać znaczenie zbiorów validacyjnych. Dodatkowo musiałem zmniejszyć rozmiar datasetu 100 krotnie.

Dane do klasyfikacji binarnej generowałem zgodnie ze wzorem:

$$f(x) = 0.000005x^5 + losowa\ liczba\ z\ przedziału\ \langle 0, liczba\ próbek \rangle \tag{10}$$

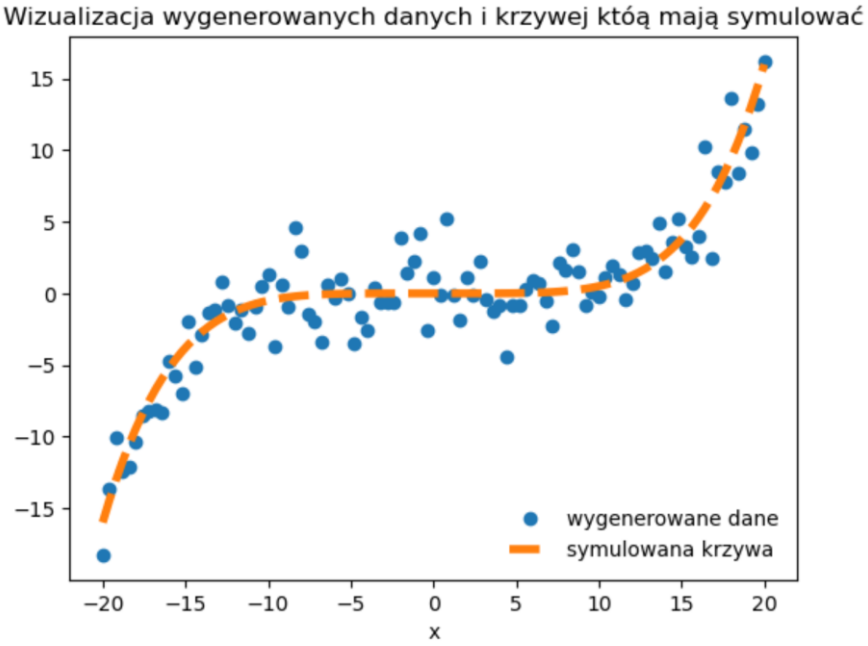
Przyjąłem, że zmienna x jest losowana z przedziału:

$$x \in \langle 0{,}10 \rangle$$

Generowałem dwa datasety:

- Treningowy o liczności 80
- Testowy o liczności 20

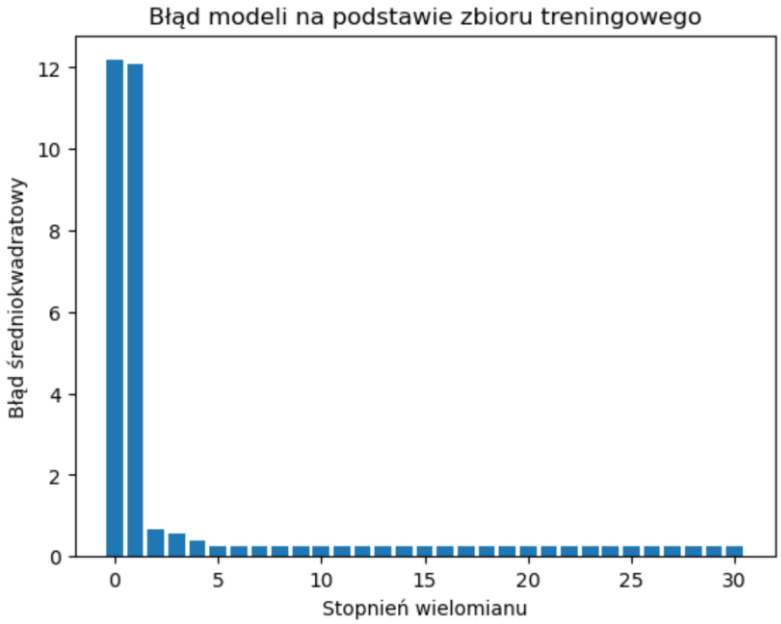
Wygenerowana funkcja wyglądała w poniższy sposób:



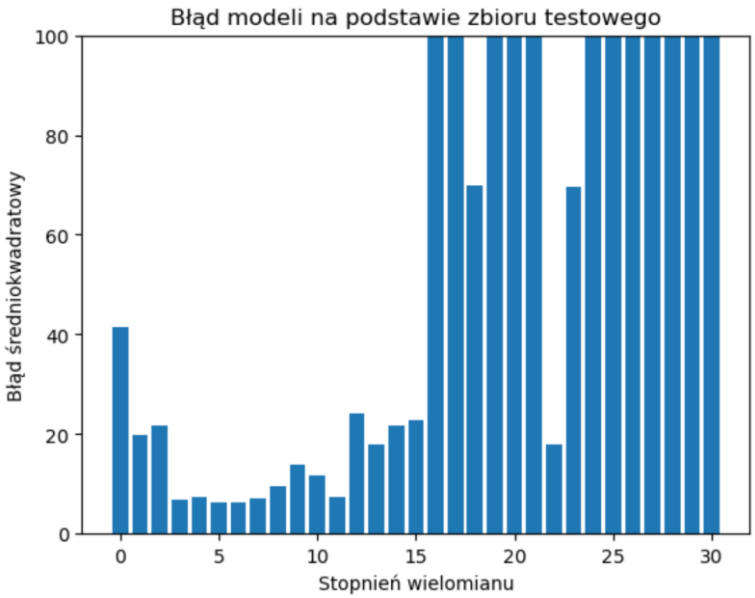
Rys. 6 Wizualizacja wygenerowanego zbioru treningowego

3.2.4 Poszukiwanie najlepszego modelu

Podejście było identyczne jak w podpunkcie 3.2.2, i pomiary wyszły następująco:



Rys. 7 Błąd modeli na podstawie zbioru treningowego



Rys. 8 Błąd modeli na podstawie zbioru testowego

W przeciwieństwie do podpunktu 3.2.2 tutaj widzimy, że od stopnia 7. wielomianu wzrasta błąd średniokwadratowy, jest to związane z overfittingiem czyli zbytnim dopasowaniem modelu do danych.

3.3 Zadanie z prędkościami i regresją wielomianową

3.3.1 Generacja danych

Dane do problemu przewidywania prędkości średniej (średnia harmoniczna dwóch liczb) generowałem zgodnie ze wzorem:

$$z(x,y) = \frac{2xy}{x+y}$$
$$y(x) = Losowy\ wybór\ ze\ 100\ próbek\ rozkładu\ normalnego\ o\ wartości\ oczekiwanej\ równej\ f(x)\ i\ wariancji\ równej\ 0.5$$

(11)

Jako, że powyższy opis może być niejasny zamieszczam realizację w kodzie:

```
def foo_to_predict2(x,y, sigma = 0.5):
    foo = lambda x, y: 2*x*y/(x+y)
    return random.choice(np.random.normal(foo(x, y), sigma, 100))
```

Przyjąłem, że zmienne x i y są losowane z przedziału:

$$x \in \langle 0,10 \rangle$$
$$y \in \langle 0,10 \rangle$$

(12)

Generowałem dwa datasety:

- Treningowy o liczności 8000
- Testowy o liczności 2000

Ponieważ wizualizacja musiałaby być na wykresie w 3d uznałem, że może nie być ona czytelna i nie wykonywałem jej

3.3.2 Poszukiwania najlepszego modelu

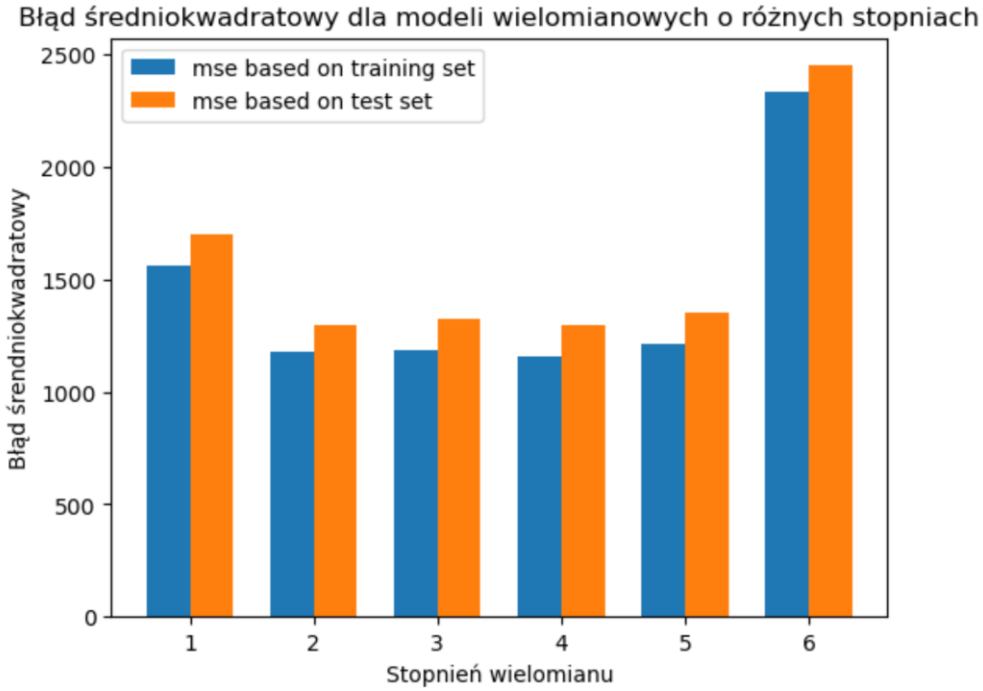
Z niewiadomych dla mnie powodów podejście które zaprezentuję poniżej nie działało w podpunktach 3.2.2 i 3.2.4, tzn. błąd średniokwadratowy się nie zmieniał wraz ze wzrostem stopnia wielomianu powyżej 1. Dlatego byłem zmuszony korzystać z polynomial fit z biblioteki numpy a nie z podejścia opisanego poniżej.

W celu znalezienia najlepszego modelu musiałem każdy z nich stworzyć a następnie porównać ich błędy w ocenianiu zbioru testowego. Aby zamienić wektor cech liniowych na wielomianowcyh użyłem klasy PolynomialFeatures z pakietu sklearn.preprocessing a następnie wykonywałem regresję liniową przy użyciu klasy RidgeCV z pakietu sklearn.linear_model. Warto zwrócić uwagę że RiggeCV zawiera wbudowaną regularyzację grzbietową oraz walidację skrośną. Badałem wielomiany o potęgach od 1 do 7. Poniżej zamieszczam pomiary błędu tych modeli:

Stopnień wielomianu	**Błąd zbioru treningowego**	**Błąd zbioru testowego**
1	1560.36	1700.19
2	1177.30	1299.92
3	1185.92	1326.74
4	1159.30	1298.14
5	1212.98	1355.53
6	2335.33	2450.95
7	$2.2 * 10^{57}$	$2.4 * 10^{57}$

Tab.2 Błędy dla sprawdzanych modeli dla zbiorów trenigowywch i testowych

Nie sprawdzałem wyższych stopni wielomianów ponieważ tam błędy były też już ogromne. Jako, że ten efekt występuje zarówno dla zbioru treningowego jak i testowego przypuszczam że wynika on błędów arytmetyki zmiennoprzecinkowej, która też ma swoje ograniczenia. Niemniej analiza tych siedmiu modeli może nam już coś powiedzieć.



Rys.9 Wykres wizualizujący tabelę nr 2 poza wielomianem stopnia 7, który jedynie zaciemniłby wyniki

Widzimy, że najlepiej poradził sobie model o stopniu równym dwa, a potem błąd zaczynał się powoli zwiększać. Więc ten model uznałbym jako najlepszy. Dodatkowo warto zauważyć, że błędy dla zbiorów testowych są większe niż dla treningowych (co nie powinno być zaskoczeniem, ale dopiero znaczące różnice wskazywałyby na overfitting i wtedy powinniśmy się nimi przejmować)

4 Wnioski

- Model Regresji Logistycznej lepiej radzi sobie z zagadnieniem klasyfikacji binarnej niż modele Regresji Liniowej (również te z regularyzacją)
- Regularyzacja grzbietowa (ridge) dała lepsze rezultaty niż regularyzacja Lasso w problemie klasyfikacji binarnej
- W problemie regresji okazało się, że dla funkcji (8) trudno jest uzyskać overfitting albo błędy spowodowane arytmetyką zmiennoprzecinkową zatem tam im wyższy stopień wielomianu tym lepsze dopasowanie
- W problemie regresji dla funkcji (10) okazało się już, że dla wielokrotnie mniejszego zbioru danych oraz innej funkcji możliwe jest uzyskanie overfittingu i walidacja ziorami tesotwymi pomaga znalezienie najlepszego modelu, overfitting był dla stopnia wielomianu równego 7 i więcej
- Dla problemu prędkości średniej okazało się, że najlepiej przewiduje prędkość średnią model regresji wielomianowej o stopniu równym 2