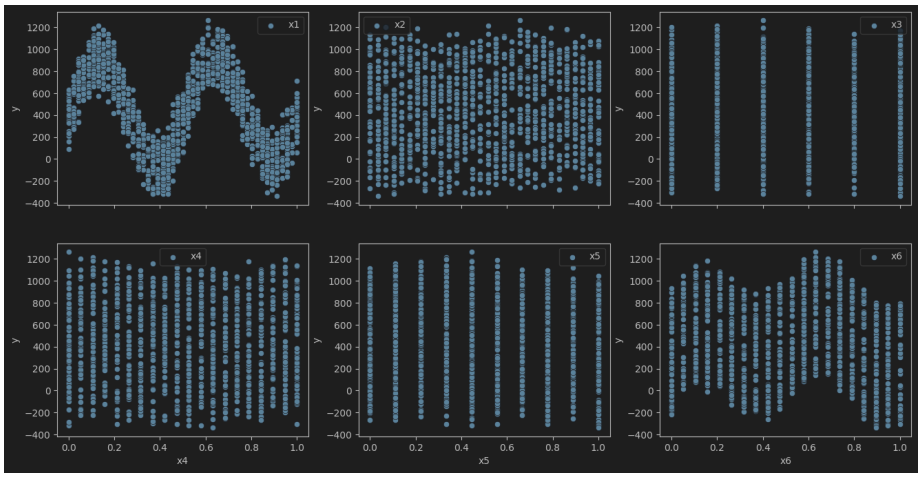
MPUM - Projekt 1

Marcin Wykpis

1. Analiza danych

Przed rozpoczęciem pracy nad odpowiednim modelem problemu regresji liniowej, należałoby zovaczyć jaki rozkład mają dane wejściowe względem oczekiwanego wyniku.

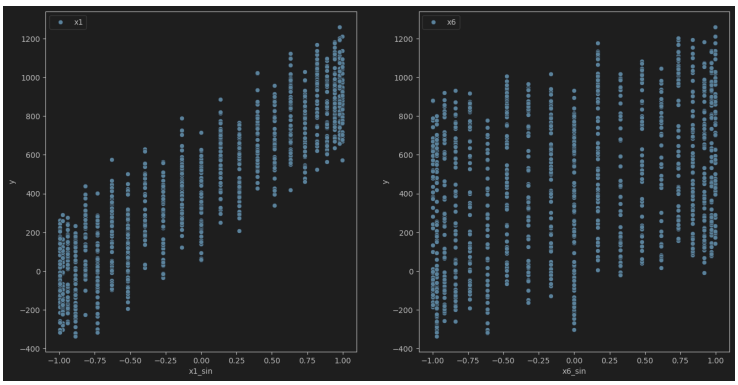
Poniżej sporządziłem wykresy cech $x_1 ... x_6$ względem wyniku y.



Analizując otrzymane wykresy wysuwają się następujące wnioski:

- Cechy x_1, x_6 mają rozkład przypominający funkcję sinus.
- W obu przypadkach wykresy te odpowiada fragmentowi sinusoidy $[0, 4\pi]$.
- Pozostałe cechy nie mają jednostajnego rozkładu, jednak ciężko przypisać im odpowiadające funkcje.

Żeby poprawić wydajność modelu, zdecydowałem się użyć dodatkowych cech - funckji bazowych $\sin(4\pi \cdot x_1)$, $\sin(4\pi \cdot x_6)$ oraz funkcji jednomianów x^k niskiego stopnia oraz iloczyny $x_i \cdot x_j$.



2. Podział danych

Dane zostały przeze mnie podzielone w pseudo-losowy sposób w proporcjach:
60/20/20 (dane treningowe / walidacyjne / testowe)

```
train:  (1199, 72)
validate:  (400, 72)
test:  (400, 72)
```

```
def partition(df, frac, seed=1234):
    assert 0 < frac < 1
    df_train = df.sample(frac=frac, random_state=seed)
    df_test = df.drop(df_train.index)
    return df_train, df_test

df_train, df_test = partition(df, 6/10)
df_valid, df_test = partition(df_test, 1/2)
```

3. Preprocessing

Dane wejściowe $x_1 ... x_6$ zostały ustandaryzowane według metody min-max:

$$x' = \frac{x - \min x}{\max x - \min x}$$

Odpowiednie stałe (min i max dla danej cechy) zostały wyznaczone jedynie przy użyciu danych treningowych, tak by nie przemycić informacji o danych testowych podczas treningu.

```
class MinMaxScaler:
    def __init__(self, A):
        self.shift = A.min(axis=0)
        self.factor = A.max(axis=0)-A.min(axis=0)

    def scale(self, A):
        return (A-self.shift)/self.factor

def scale(df, scaler):
    df_scaled = scaler.scale(df.drop('y', axis=1))
    df_scaled['y'] = df['y']
    return df_scaled

scaler = MinMaxScaler(df_train.drop('y', axis=1))
df_train = scale(df_train, scaler)
df_valid = scale(df_valid, scaler)
df_test = scale(df_test, scaler)
```

4. Metoda najmniejszych kwadratów

Problem ten będzie analizować tworząc modele dla czterech wariantów:
1. Bez zastosowania żadnej regularyzacji
2. Z regularyzacją l_2 (regresja grzbietowa)
3. Z regularyzacją l_1 (regresja lasso)
4. Zarówno z regularyzacją l_1 jak i l_2 (regresja z siecią elastyczną).

Dla każdej z tych metod rozpatrzymy funkcję straty dla rozwiazania analitycznego oraz startę uzyskaną przez metodę spadku gradientu. W każdym z tych modeli użyłem metody spadku gradientu w wersji `minibatch`, z wielkością jednego batcha wynoszącą 100.

4.1. Bez regularyzacji

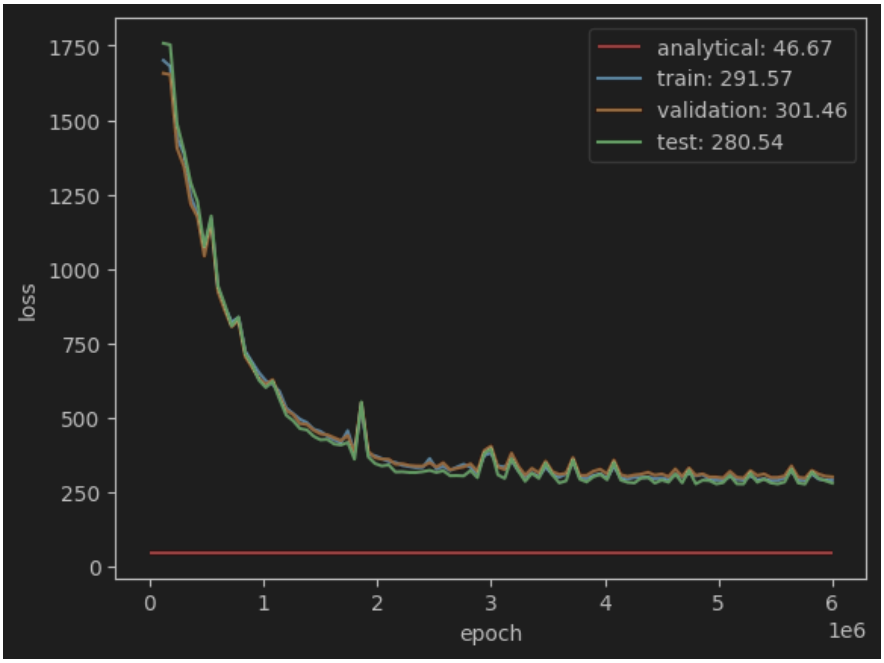
Model ten osiąga bardzo dobry wynik ~47 straty w rozwiązaniu analitycznym.

```
loss: 46.67268742635361
CPU times: user 20.5 ms, sys: 4.75 ms, total: 25.3 ms
Wall time: 2.74 ms
```

Metodę spatku gradientu wykonałem dla 500.000 iteracji, otrzymując stratę ~320.
Czas wykonywania programu wynosi ok. 2min

```
loss: 321.97082984481693
CPU times: user 1min 52s, sys: 412 ms, total: 1min 52s
Wall time: 1min 52s
```

Tak prezentuję się wykres funkcji straty dla kolejnych epok:



Podczas rozwiązywania problemu regresji metodą analityczną napotkałem pewien problem. Skutkował on ogromnym wynikiem (wartością funkcji straty), znacznie większym niż rozwiązane uzyskane metodą spadku gradientu. Problem ten miał miejsce, gdy próbowałem dodać większa licbę dodatkowych cech. Po upewnieniu się, że mój kod nie posiada oczywistego błędu, zacząłem zastanawiać się co możę powodowac zaistniała sytuację.
Oczywistym „winowajcą" wydawała się operacja odwracania macierzy. Domyślałem się że w przypadku, gdy wyznacznik macierzy XX^T był bardzo bliski zeru, wyznaczana macierz odwrotna (prawdopodobnie przez dokładność liczb zmiennoprzecinkowych) była daleka ideału. Rzeczywiście, moje podejrzenia potwierdziły się, gdy zamieniłem `np.linalg.inv` na `np.linalg.pinv`, czyli odwrotność *Moore'a–Penrose'a*, która została wspomniana na wykładzie.

4.2. Regularyzacja grzbietowa (l_2)

W rozwiązaniu analitycznym posłużyłem się wzorem udowodnionym na ćwiczeniach:

$$\widehat{\theta^r} = (X^T X + \lambda I)^{-1} X^T y$$

Po porównaniu kilku parametrów lambda, względem otrzymanych strat dla zbioru walidacyjnego, wybrałem $\lambda = 10^{-9}$.

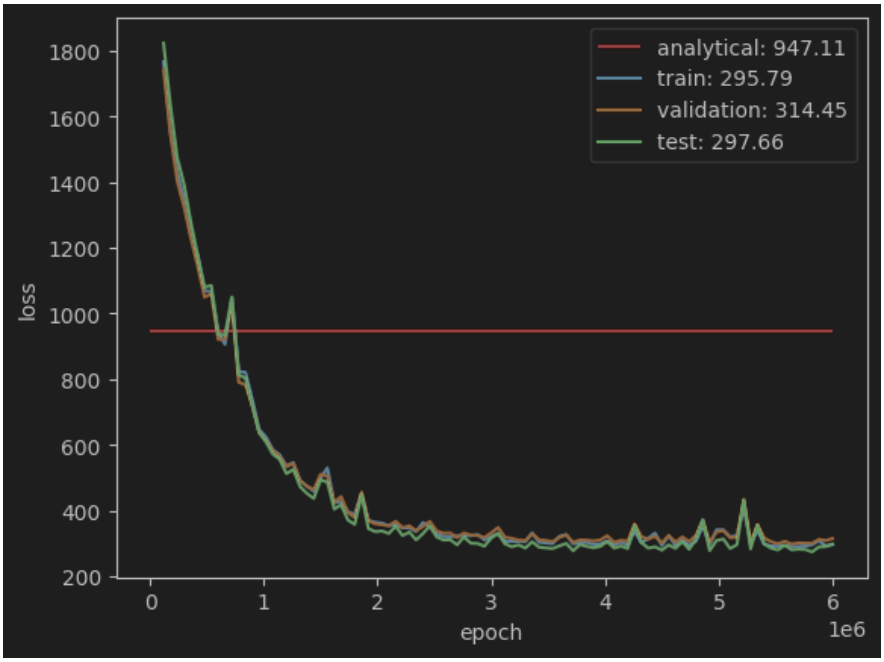
Strata uzyskana za pomocą rozwiązania analitycznego*: ~947

```
loss: 947.1079183515436
CPU times: user 9.69 ms, sys: 48.1 ms, total: 57.8 ms
Wall time: 10.6 ms
```

Metodę spatku gradientu wykonałem dla 500.000 iteracji, otrzymując stratę ~262.
Czas wykonywania programu wynosi ok. 2min

```
loss: 262.45325292601655
CPU times: user 2min 20s, sys: 471 ms, total: 2min 20s
Wall time: 2min 20s
```

Metodę spatku gradientu wykonałem dla 500.000 iteracji, otrzymując stratę ~322.
Czas wykonywania programu wynosi ok. 2min



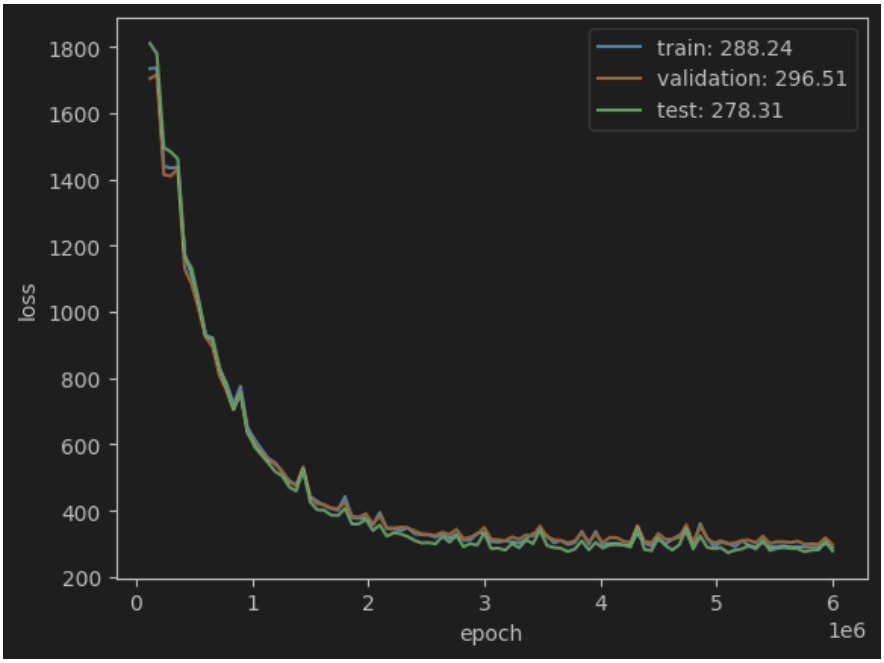
∗ W tym przypadku roziwązanie analityczne okazało się mieć większą wartość straty niż rozwiązanie osiągnięty przy pomocy gradientu. Poraz kolejny podejrzewam że problem wynika z odwracania macierzy, jednak tym razem odwrotność *Moore'a–Penrose'a* dalej zwraca za duży wynik.

4.3. Regularyzacja lasso (l_1)

Poniższe wyniki osiągnięte zostały przy $\lambda = 10^{-6}$

Metodę spatku gradientu wykonałem dla 500.000 iteracji, otrzymując stratę ~256. Czas wykonywania programu wynosi ok. 2min

```
loss: 256.322372513923
CPU times: user 2min 26s, sys: 1.23 s, total: 2min 28s
Wall time: 2min 29s
```

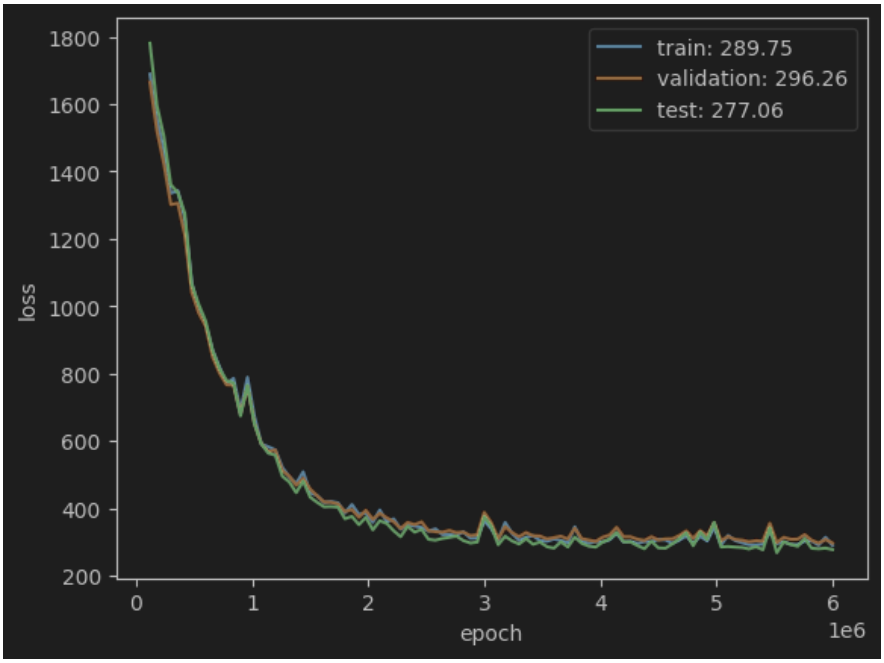


4.4. Regularyzacja siecią elastyczną $(l_1 + l_2)$

Poniższe wyniki osiągnięte zostały przy $\lambda_1 = 10^{-6}, \lambda_2 = 10^{-9}$

Metodę spatku gradientu wykonałem dla 500.000 iteracji, otrzymując stratę ~256. Czas wykonywania programu wynosi ok. 2min

```
loss: 278.3478993890484
CPU times: user 3min 1s, sys: 1.49 s, total: 3min 2s
Wall time: 3min 4s
```



5. Implementacja

Implementację powyższych modeli oraz użyte w raporcie wykresy można znaleźć w repozytorium na githubie: https://github.com/Marwyk2003/mpum_miniproject1