

MPUM MINIPROJEKT I

MICHAŁ HOFFMANN

ABSTRACT. Wykorzystanie regresji liniowej do analizy danych z zadanego pliku. W projekcie wykorzystałem kilka metod regresji, jak i parę metod gradientowych. Użyłem też gotowych bibliotek jak i rozwiązania analitycznego jako benchmarków.

1. DANE I NORMALIZACJA

Zanim zaczniemy analizę danych, spójrzmy na nie ludzkim okiem. Dane to 7 kolumn i 1999 wierszy, w których pierwsze 6 kolumn to parametry obserwacji, a ostatnia jest wartością. Początkowo, parametry mają wartości na różnych przedziałach, więc analiza nie będzie miarodajna:

feature:	minimal value:	maximal value:
0	13	59
1	30	59
2	2	7
3	-10	9
4	-5	4
5	-10	9
y	-356.66	1260.63

Przed rozpoczęciem badania danych za pomocą regresji liniowej, musimy je przeskalować, aby wartości parametrów były na przedziale $[0, 1]$. W tym celu, zastosujemy normalizację *min-max*, która skaluje każdy parametr wzorem:

$$x' = \frac{x - x.\min()}{x.\max() - x.\min()}$$

gdzie $x.\min()$ i $x.\max()$ to funkcje zwracające odpowiednio największą i najmniejszą wartość danego parametru.

Być może, jest tu delikatna nieścisłość - najlepiej by było najpierw podzielić dane na zbiory: treningowy i testowy, a do skalowania zastosować wartości $\min()$ i $\max()$ ze zbioru testowego, aczkolwiek patrząc po częstotliwość występowania skrajnych wartości każdego parametru, będzie to bardzo mało istotne w praktyce, a utrudnia intuicyjność implementacji.

1.1. Początkowe dane a wartość:

Poniższe wykresy przedstawiają zależność wartości obserwacji od każdego parametru z osobna. Obie wersje, znormalizowana, jak i nie, przedstawiają podobny kształt - możemy być pewni, że skalowanie nie zmieniło rozkładu danych - nasza analiza będzie miarodajna.

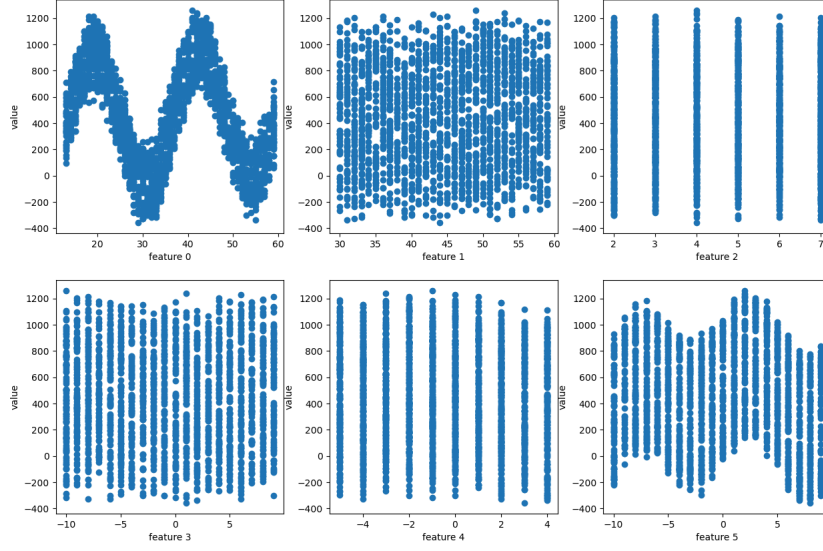


FIGURE 1. Funkcje zależności parametr:wartość bez skalowania.

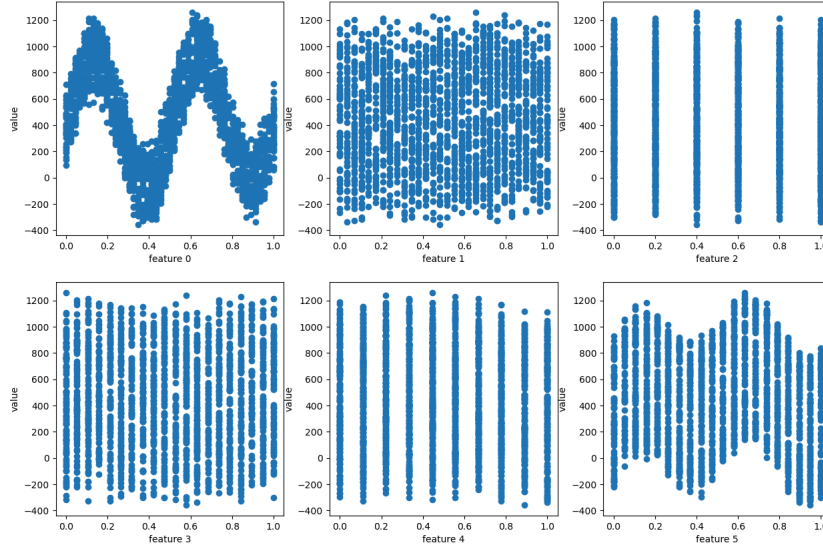


FIGURE 2. Funkcje zależności parametr:wartość po przeskalowaniu.

1.2. Obserwacje z wykresów.

Patrząc na cechy indywidualnie, szybko można zauważyć kilka istotnych własności. Przede wszystkim, w szczególności dla cech 0 i 5 występuje widocznie sinusoidalna zależność między nimi, a wynikiem. Dla cech 4 i 3 wprawne (lub przewrażliwione) oko, także może dostrzec zarys sinusoidy, ale o innym okresie i większym szumie. Ta obserwacja spowodowała, że do hiperparametrów dodałem funkcje $\sin(\pi x * j)$ i $\cos(\pi x * j)$ dla kilku różnych wartości j , a co tłumaczy, dlaczego używanie funkcji gaussowskich niewiele dało. Obserwacje zostaną poparte

obliczeniami w sekcji **Dodatki**, gdzie przeprowadziłem uczenie bez każdej z cech z osobna i zebrałem na jednym wykresie.

2. WYKORZYSTANE METODY:

Do rozwiązania problemu wykorzystałem czystą regresję liniową, jak i jej modyfikacje poznane na wykładzie, są to *Regularyzacja grzbietowa*, *Regularyzacja typu lasso*, jak i *Regularyzacja z siecią elastyczną*, które mają na celu zmniejszyć wartości wag przy parametrach θ i zapobiec przetrenowaniu. Ponadto, stosuję też kilka algorytmów spadku gradientu: wersję *minibatch* jak i *stochastyczną* - czyli minibatch dla batchy jednostkowych. Koniec końców, w podsumowaniu nie zawarłem połączenia regularyzacji i innych metod spadku gradientowego, gdyż paradoksalnie psuło to mój wynik. W trochę inny sposób przeprowadzam także algorytm spadku względem współrzędnych. Losuję jedynie ich kolejność, w każdej iteracji zmieniam gradient względem wszystkich, a nie tylko wybranej. Odniosłem wrażenie, że jest to bardziej stabilna metoda, a wyniki były bardzo podobne.

3. ROZWIĄZANIE ANALITYCZNE

Jak pokazaliśmy na wykładzie, analityczne rozwiązanie regresji liniowej jest postaci:

$$\theta = (X^T X)^{-1} X^T y$$

co dla bazowych parametrów (dla funkcji bazowych $\Phi(x) = x$) daje następujący średni koszt:

$$\text{MSE}(\theta) = 130302.18513264667$$

Dla regresji liniowej z użyciem regularyzacji grzbietowej:

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

Co dla kilku parametrów lambda i zwykłych funkcji bazowych daje następujące wyniki:

parametr:	$\lambda = 10^{-8}$	$\lambda = \frac{1}{2}$	$\lambda = 1$	$\lambda = 5$	$\lambda = 10$	$\lambda = 100$	$\lambda = 1000$
MSE:	133684	133565	133614	136052	139350	156909	237806

Regresja z regularyzacją typu lasso nie ma rozwiązania analitycznego. Jest to spowodowane użyciem normy L_1 , która nie jest różniczkowalna w zerze.

Wyniki dla podstawowych wartości parametrów nie są zbyt dokładne. Zmieni się to drastycznie, kiedy zastosujemy zbiór hiperparametrów znalezionych trenując regresję. Biorąc pod uwagę wszystkie **948** parametrów, których używam w regresji, otrzymuję następujące wyniki:

```

Test set loss for analytical solution: 72.72194907142406
Test set loss for ridge analytical solution: lambda: 1e-08 value: 73.07908486794156
Test set loss for ridge analytical solution: lambda: 0.5 value: 222.22538040902032
Test set loss for ridge analytical solution: lambda: 1 value: 328.37119244374566
Test set loss for ridge analytical solution: lambda: 5 value: 606.7593125697796
Test set loss for ridge analytical solution: lambda: 10 value: 746.1362022099825
Test set loss for ridge analytical solution: lambda: 100 value: 1955.6216021516645
Test set loss for ridge analytical solution: lambda: 1000 value: 19116.19126815643
Test set loss for ridge analytical solution: lambda: 10000 value: 99507.40828699258
Test set loss for ridge analytical solution: lambda: 100000 value: 170556.38039576568

```

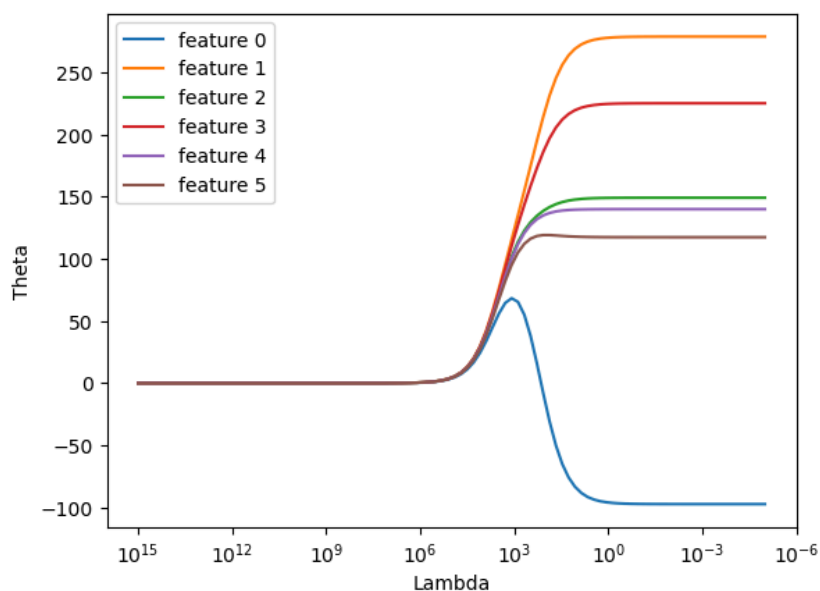
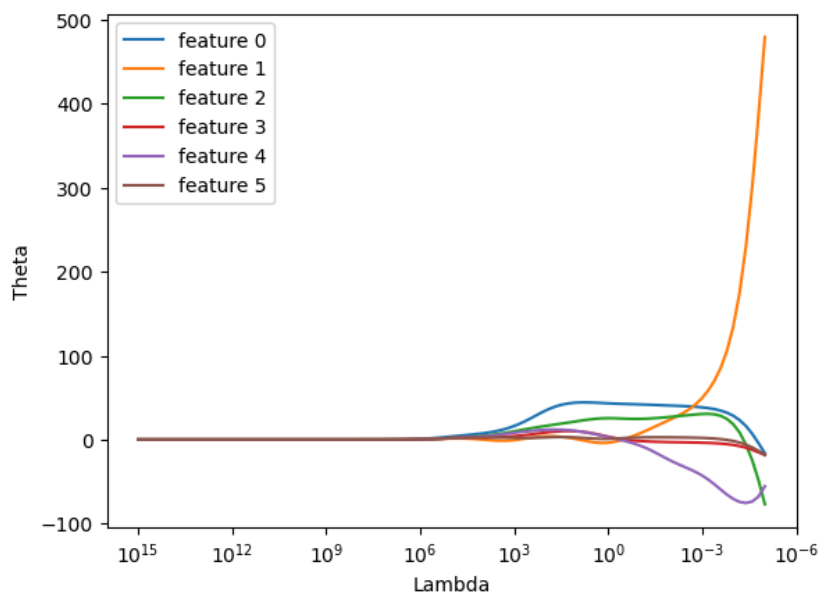
FIGURE 3. rozwiązanie analityczne dla pełnego zbioru hiperparametrów.

Jak widać, dodanie tych samych hiperparametrów, które polepszają wynik przy metodzie gradientowej, pozwala na uzyskanie dużo dokładniejszego wyniku za pomocą metod analitycznych. Błąd rozwiązania analitycznego jest dwukrotnie mniejszy od gradientowego, jednak wciąż dwa razy większy od benchmarku, co pozostawia pole do poprawy, na którą niestety zabrakło mi czasu.

Testując oba rozwiązania analityczne na zbiorach testowym i treningowym, zauważyłem, że funkcja obracająca macierze nie zawsze sobie z nimi radzi. Rozwiązaniem okazało się wykorzystanie implementacji metody, która notabene została wspomniana na wykładzie, czyli pseudo-inwersja Moore’a-Penrose’a, zaimplementowanej w bibliotece *numpy* jako *linalg.pinv*. Co ciekawe, po wstawieniu tej funkcji wszędzie gdzie to możliwe, także otrzymałem błędy, które zniknęły po wróceniu do zwykłego obracania macierzy (*linalg.inv*). Oznacza to, że jakiś szczegół implementacyjny różni się dla tych dwóch podejść, choć teoretycznie pseudo-inwersja jest równoważna inwersji dla macierzy nieosobliwych.

3.1. Istotność parametrów - dokładna analiza.

Zastosowanie regularyzacji grzbietowej dla wielu wartości λ pozwala zaobserwować, który parametr bazowy jest najbardziej istotny. Będzie to ten, który najpóźniej zbiega do zera dla coraz większych λ . Z takiej analizy wynika, że najbardziej znaczącym parametrem jest parametr 0, a najmniej - 4. Poniżej zamieszczam dwa wykresy - jeden dla bazowych 6 parametrów, a drugi, uwzględniający w macierzy \mathbf{X} także dodane przeze mnie hiperparametry, umieszczając na wykresie jedynie wyjściowe dane, otrzymuję dużo bardziej dokładny obraz tego, jak istotna jest każdy z parametrów. Pokrywa się to zarówno z obserwacjami z wykresu *parametr:wynik*, jak i analizą w **Dodatkach**. Wykres prezentuje się następująco:

FIGURE 4. Wartości θ dla różnych wartości λ , 6 parametrów.FIGURE 5. Wartości θ dla różnych wartości λ , 948 parametrów.

W ten sposób sprawdziłem jedynie parametry wynikające z danych, jednak z ciekawości spojrzałem jeszcze, jak wyglądałby wykres, gdyby w regularyzacji brać pod uwagę wyraz wolny. Okazuje się, że zmienia to całkiem sporo. Wartości λ , dla których parametry mają podobne wartości zostają podobne, jednak w nieskończoności, otrzymujemy zupełnie inne wyniki! Ku mojemu zdziwieniu, można zauważyć, że działa to tylko dla niewielu cech, gdyż w analizie na wszystkich paramet-

trach, wynikiem biasowanego wyniku jest bardzo podobna wartość, co w analizie bez tego czynnika.

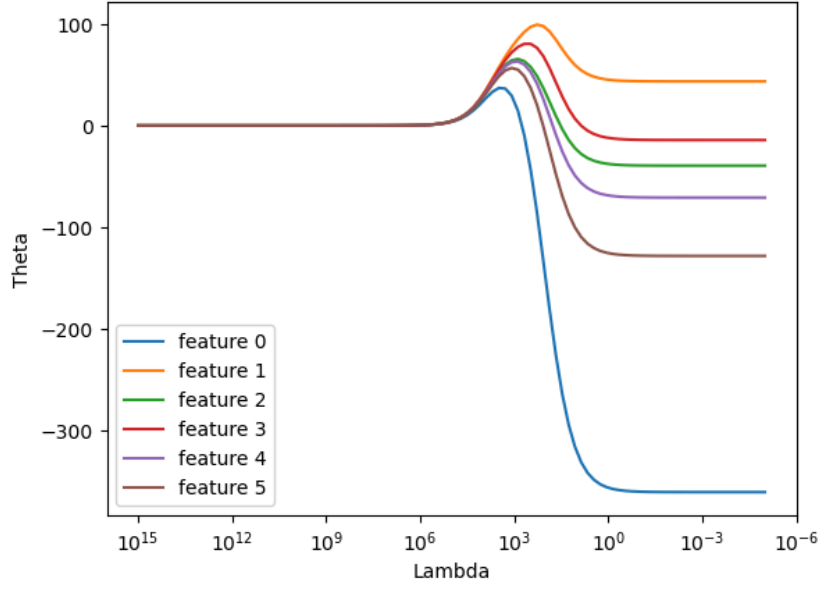


FIGURE 6. Wartości θ dla różnych wartości λ , 7 parametrów (dodany bias).

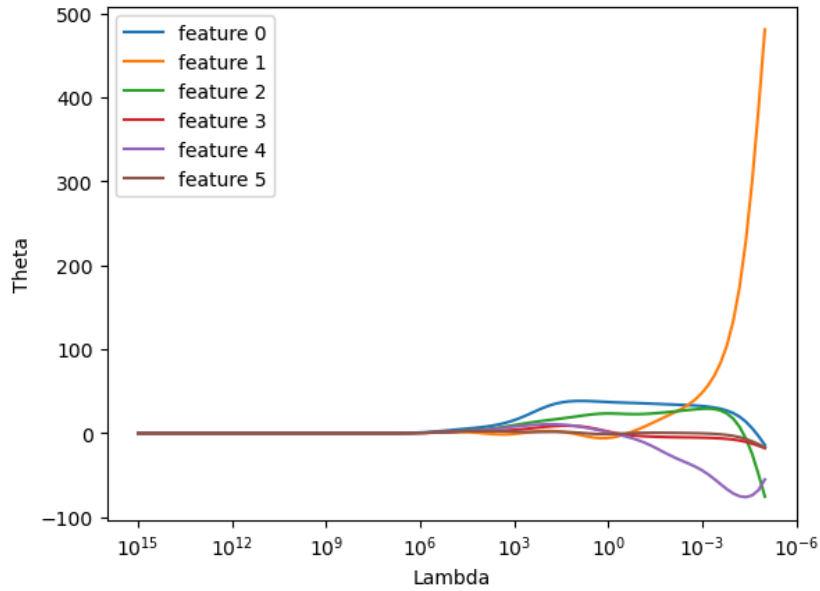


FIGURE 7. Wartości θ dla różnych wartości λ , 949 parametrów (dodany bias).

4. TESTOWANIE FUNKCJI BAZOWYCH I INNYCH HIPERPARAMETRÓW

Początkowe wyniki regresji nie były zbyt precyzyjne. Odpowiedzią na ten problem okazało się wprowadzenie większej liczby parametrów.

4.1. Parametry odrzucone.

Ze względu na kształt wykresów z sekcji 1.1, od początku nie byłem przekonany do funkcji gaussowskich, co potwierdziły eksperymenty. Być może dla innych danych funkcje te mogłyby polepszyć wyniki, ale w tym przypadku nie widzę skutecznego doboru takich parametrów, by te funkcje były pomocne.

Inne funkcje trygonometryczne i ich odwrotności poza tymi, które finalnie wykorzystałem, nie wydawały się wpływać pozytywnie na wyniki regresji.

4.2. Parametry wykorzystane.

W kolejności wykorzystywania (kolejne grupy funkcji bazowych wykorzystują zbiór początkowych parametrów razem z poprzednimi grupami)

1a) $\sin(j \cdot \pi \cdot x_i)$ i $\cos(j \cdot \pi \cdot x_i)$ dla $j \in [2, 4, 8]$. Motywacja jest tutaj prosta, wejściowe dane widocznie przedstawiają sinusoidalną strukturę zależności, więc użyłem dwóch znanych mi funkcji o tym kształcie. Długość okresu znalazłem eksperymentalnie, ale kilka wartości daje lepsze wyniki niż jedna, więc końcowo zawarłem te trzy.

1b) $x^k \mid k = 2^i, i \in [5, 6, 7, 8]$ - wysokie potęgi początkowych parametrów wprowadzane do modelu równoległe z punktem 1a)

2) $x_i \cdot x_j \quad \forall_{i < j}$ Pozwala na uwytłumienie parami mocnych cech i wyciszenie słabszych.

Końcowo, w celu tworzenia wykresów zrezygnowałem z umieszczania w modelu wysokich potęg, gdyż здавало się, że występowały przy takiej liczbie parametrów błędy precyzji, jednocześnie używając ich wydłużał się czas liczenia. Wynik nie odbiegał znacznie od tego, otrzymanego bez nich (średnio błąd oscylował w granicach 190 zamiast 210 dla regresji batchowych), więc zdecydowałem się je wykomentować.

4.3. Hiperparametry regularyzacji.

Choć koniec końców najlepsze wyniki nie pochodziły z regresji zregularyzacją, poświęciłem trochę czasu na dobranie odpowiednich parametrów λ dla każdego z typów regularyzacji.

Najczęściej, najskuteczniejszą w moim przypadku (choć o niewiele) okazywała się regularyzacja grzbietowa. Przełożyło się to także na parametryzację regularyzacji z siecią elastyczną, gdzie najlepiej sprawdza się parametr, który znacznie faworyzuje regresję grzbietową ponad lasso. W każdym z przypadków, najlepsze wyniki daje mały parametr regularyzacji około 0.001 (λ , tudzież α dla sieci elastycznej).

4.4. Kwestia funkcji straty.

Choć głównie skupiałem się na znalezieniu optymalnych parametrów dla kwadratowej funkcji straty, trochę czasu poświęciłem na szukanie lepszych wartości dla regresji liniowej ze stratą absolutną i kilkoma wartościami delty dla funkcji Hubera (gdzie najlepsze wyniki dostaje dla $\delta = 10$), ale wyniki były bardzo podobne (porównywałem MAE i MSE dla ostatecznych wyników). Zdecydowałem umieścić jedynie wyniki dla MSE, ale w kodzie można szybko podmienić, z której funkcji kosztu korzystamy i pracować w zależności od niej.

5. WYNIKI

Końcowy model sprawuje się zaskakująco dobrze. Najlepsze wyniki osiągam używając spadku gradientowego *minibatch*, gdzie najlepiej działającym rozmiarem paczek jest 10. Przeprowadziłem pewien skrót o zerowej szkodliwości i moja wersja algorytmu stochastycznego spadku, to po prostu minibatch dla batchy wielkości 1. Pozwala to efektywnie liczyć optymalny wynik, jednocześnie ucząc na wszystkich elementach jednocześnie (jest to równoważne spadkowi po współrzędnej dla wielu więcej iteracji).

Regresje z regularyzacją i takim algorytmem gradientowym mieściły się gdzieś pomiędzy minibatchem a samą regularyzacją, więc pomijam je w analizach. Istotne jest to, że samo dodanie regularyzacji znacznie pomaga osiągnąć lepszy wynik, jednak wciąż jest rząd wielkości gorszy od ulepszanego gradientu. Wyniki prezentują się następująco:

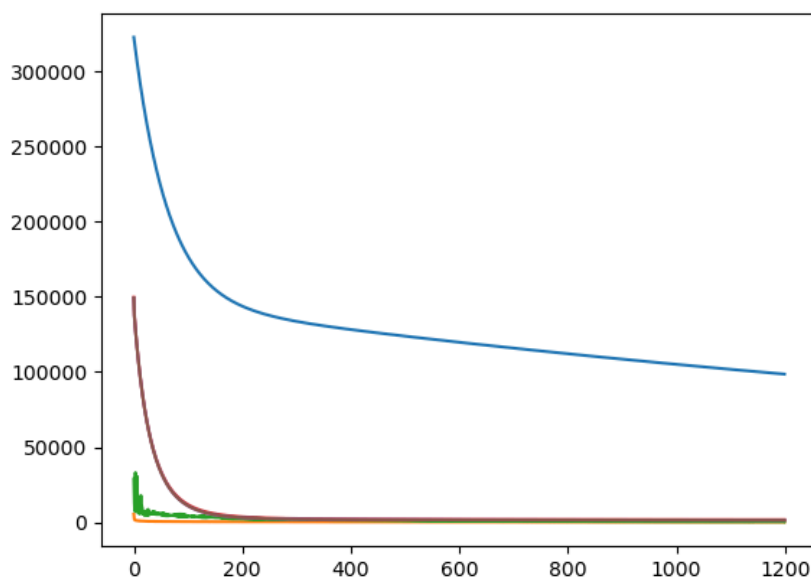


FIGURE 8. Krzywa uczenia dla różnych modeli regresji liniowej

Dokładne wartości osiągnane przez modele lepiej widać tutaj (wykresy zawierają wyłącznie końcowe 2/3 procesu uczenia):

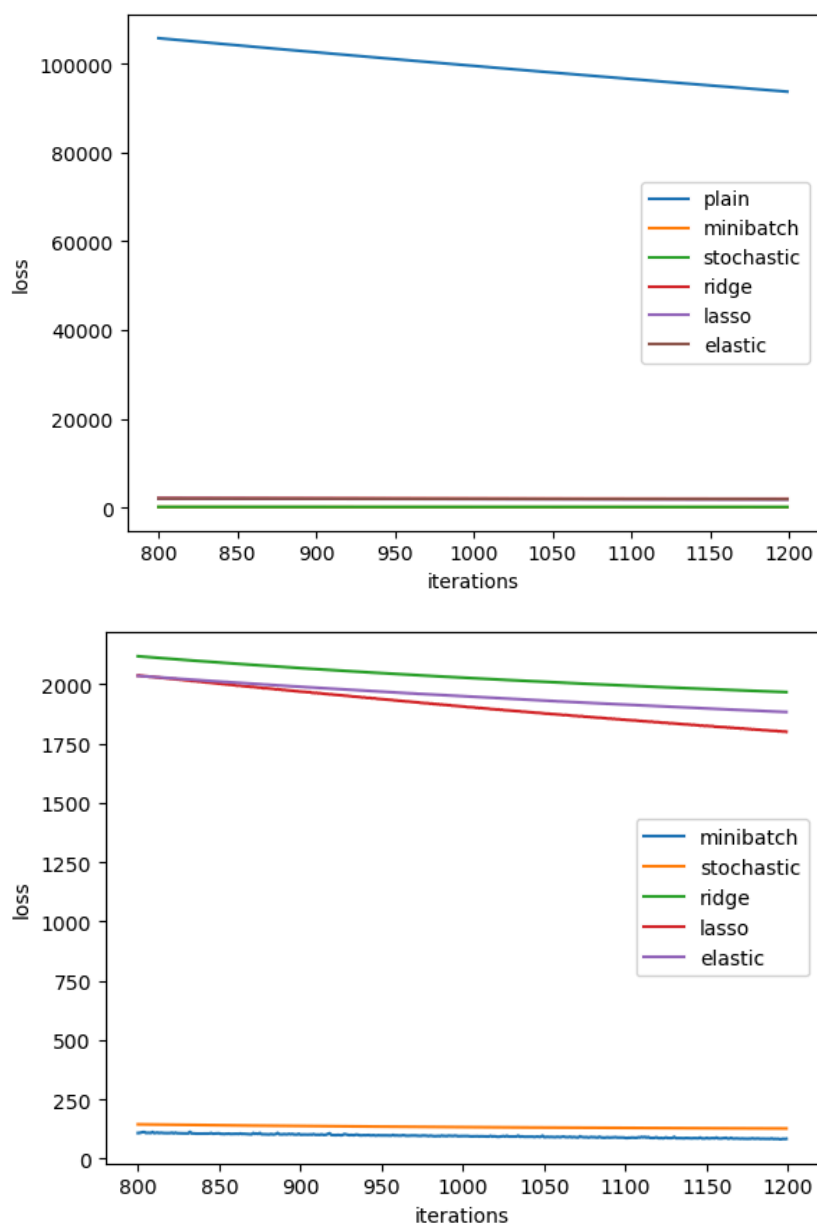


FIGURE 9. Porównanie wyników regresji bez jednego z parametrów (z przybliżeniem poniżej).

Żeby zaznaczyć tempo uczenia, dodałem kropki w [1%, 2%, 3%, 12.5%, 62.5%, 100%] procesu uczenia:

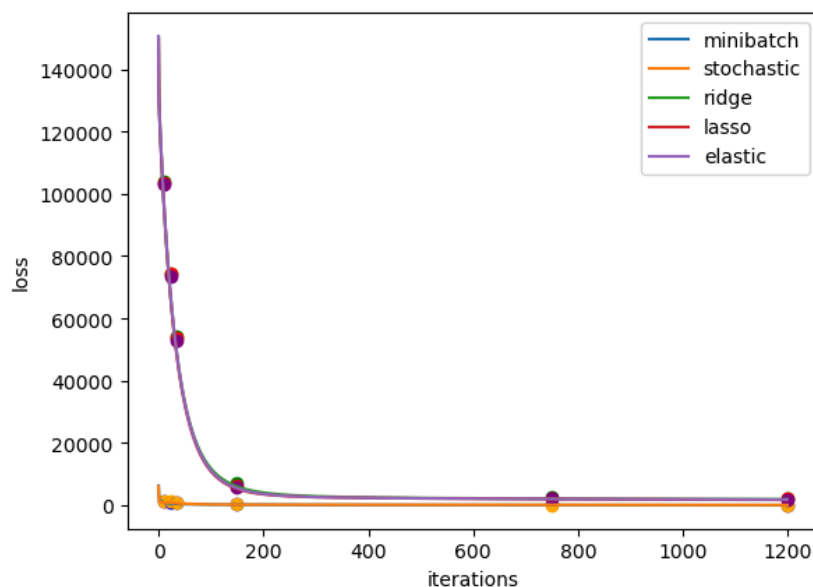


FIGURE 10. Uczenie z zaznaczonymi ułamkami liczby iteracji

6. DODATKI

6.1. Trenowanie na mniejszych fragmentach zbioru.

Dla regresji minibatch przeprowadziłem następujący eksperyment: Oddzieliłem od zbioru testowego mniejszy fragment i użyłem algorytmu gradientu wyłącznie dla tego fragmentu. Pozwoliło to porównać kilka rzeczy

1. Końcowy wynik dla mniejszego zbioru testowego, odpowiada to na pytanie “ile danych potrzeba, żeby model się nauczył”
2. Jakie jest tempo uczenia i czy zależy od ilości danych?

Odpowiedzi na te pytania są zgodne z intuicją, czyli mniejszy zbiór na ogół uczy się szybciej, ale gorzej (choć już od 12% całego zbioru wyniki były w tym samym rzędzie wielkości co dla pełnego, a dla połowy różnice były niemalże kosmetyczne).

Jednakże, wciąż wartościowym dla mnie było sprawdzenie tego własnoręcznie, czego wyniki prezentuje poniższy wykres:

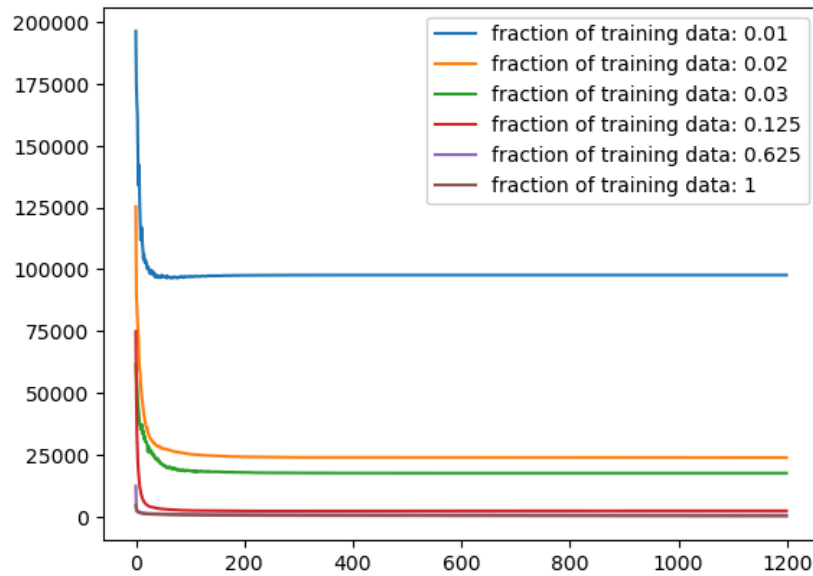


FIGURE 11. Uczenie używające jedynie części zbioru testowego.

6.2. Istotność parametru po raz trzeci.

Poza zadaną regresją liniową, zdecydowałem się sprawdzić za pomocą mojego modelu, w jakim stopniu każda z cech wpływa na wynik. Okazało się, że zarówno przewidywania rozwiązania analitycznego, jak i proste obserwacje zależności *wartość:parametr*, zgadzają się z wynikami, które tutaj przedstawiam.

Poniższe wyniki zostały otrzymane poprzez usunięcie z macierzy obserwacji całej kolumny, pozostawiając cały proces dodawania parametrów. Oznacza to, że każdy z mniejszych modeli ma **668** parametrów. Proces uczenia pozostaje bez zmian.

Na wykresie zawarłem jedynie wyniki ze zwykłego gradientu *minibatch*, była to stabilnie bardzo dokładna metoda, a porównywanie wielu, zaburzyłoby przejrzystość.

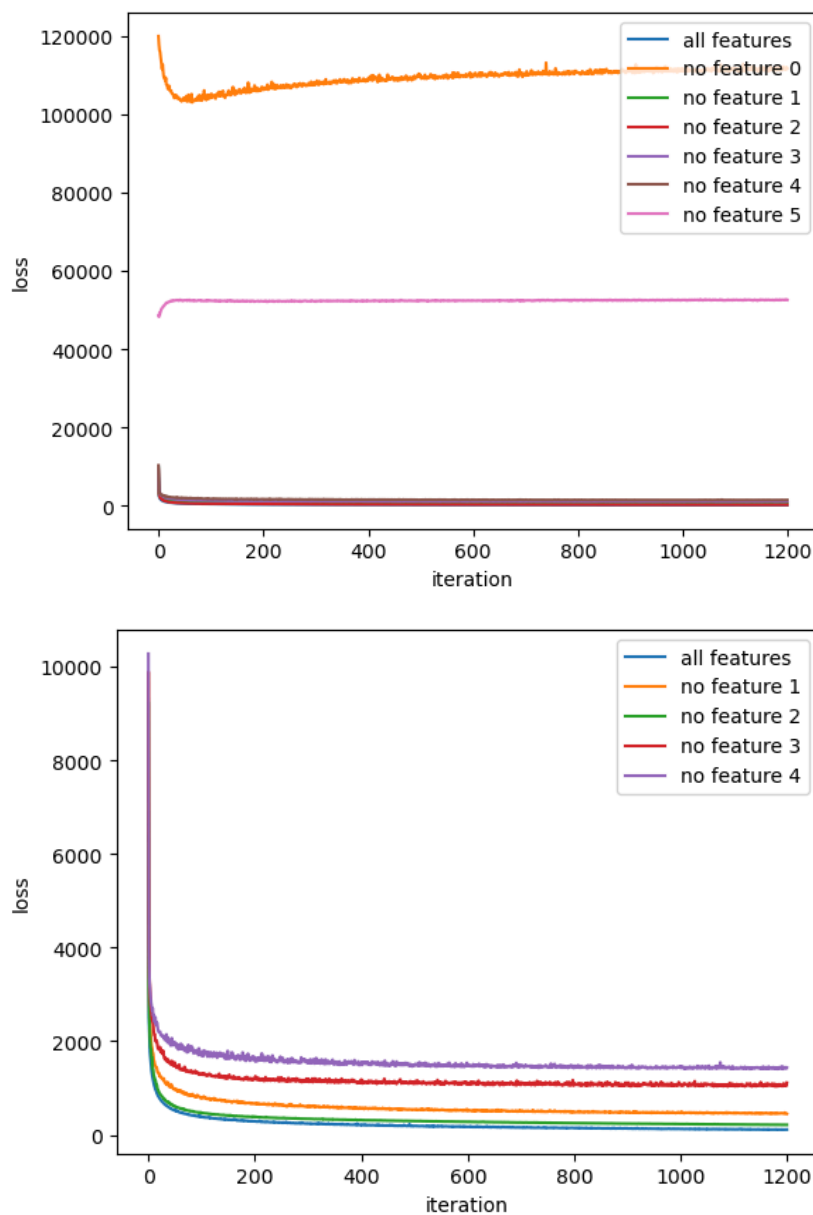
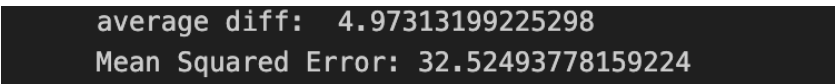


FIGURE 12. Porównanie wyników regresji bez jednego z parametrów (z przybliżeniem poniżej).

6.3. Gotowy benchmark.

Żeby ocenić swoją pracę, wykorzystałem gotową funkcjonalność biblioteki sklearn. Okazuje się, że gotowa funkcja w ułamku czasu, który zajmuje mi trenowanie modelu, otrzymuje wynik kilka razy lepszy. To słodko-gorzkie porównanie daje ogłód na to, jak wiele do odkrycia jest w prostej regresji liniowej i jak skuteczne są gotowe funkcje.



average diff: 4.97313199225298
Mean Squared Error: 32.52493778159224

FIGURE 13. Wyniki benchmarku.

REFERENCES

THEORETICAL COMPUTER SCIENCE, JAGIELLONIAN UNIVERSITY, KRAKÓW
Email address: `michal.hoffmann@student.uj.edu.pl`