Wydział
Informatyki

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE

WYDZIAŁ INFORMATYKI

mgr inż. Jacek Klimaszewski

Uczenie modeli liniowych z regularyzacją dla małych zbiorów danych

Rozprawa doktorska

Promotor: dr hab. inż. Marcin Korzeń, prof. ZUT

Szczecin 2024

Streszczenie

W niniejszej rozprawie poruszono zagadnienie uczenia modeli liniowych z regularyzacją, skupiając się na małych zbiorach danych (liczba atrybutów znacznie przewyższa liczbę obserwacji). Przeanalizowano i zbadano wydajność istniejących procedur uczenia na tych zbiorach oraz różnych form regularyzacji modeli. Sprawdzono aspekty numeryczne algorytmów, m.in. wpływ typu zmiennoprzecinkowego pojedynczej i podwójnej precyzji oraz rodzaju minimalizacji kierunkowej na szybkość uczenia oraz sposoby reprezentacji danych uczących. Wykazano teoretycznie i empirycznie, że w pewnym zakresie parametów wykorzystanie formuły Woodbury'ego bądź rozkładu QR znacznie skraca czas uczenia modeli z regularyzacją ℓ^2 — sprawdzono analogiczne rozwiązanie dla modeli wyposażonych w rzadkie funkcje kary. Eksperymenty wykonano w środowisku `Python` wraz z fragmentami zaimplementowanymi w języku `C++`, biorąc za punkt odniesienia istniejące biblioteki uczenia maszynowego, jak `scikit-learn` czy `liblinear`. Przetestowano działanie proponowanych rozwiązań w zadaniach klasyfikacji na rzeczywistych i syntetycznych zbiorach danych oraz wskazano potencjalne zastosowania uczenia regularyzowanych modeli liniowych w problemach nieliniowych.

słowa kluczowe: uczenie maszynowe, modele liniowe, regularyzacja, optymalizacja, metoda Newtona,

Abstract

This PhD thesis deals with the issue of training regularized linear models on small datasets (i.e. datasets containing much less observations than features). Existing optimization algorithms were analysed and their behaviour on those datasets was examined. Some numerical issues were checked, such as influence of single precision or double precision floating-point type or line-search procedure on training speed. It was proved theoretically and empirically, that in some configurations use of Woodbury's formula or QR factorization leads to shorter time of training ℓ^2-regularized models — the same approach was checked for sparsity-inducing penalizers. Experiments were conducted mainly in `Python` environment, with some parts written in `C++`, taking as a reference point existing machine learning libraries, i.e. `scikit-learn` or `liblinear`. Performance of proposed solutions were tested in the classification task for some real and sythetic datasets. Moreover, some potential use of training penalized linear models in non-linear problems was highlighted.

keywords: machine learning, linear models, regularization, optimization, Newton's method

Spis treści

Lista symboli

$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ oznaczają macierze.

$\boldsymbol{a}_{:,j}$ oznaczają j-tą kolumnę macierzy $\boldsymbol{A}$

$\boldsymbol{b}_{i,:}$ oznaczają i-ty wiersz macierzy $\boldsymbol{B}$

$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ oznaczają wektory.

$\langle \cdot, \cdot \rangle$ oznacza iloczyn skalarny

$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ dane uczące, gdzie $\boldsymbol{x}_i \in \mathbb{R}^p$ oraz $y_i \in \{-1, +1\}$ w przypadku klasyfikacji binarnej lub $y_i \in \mathbb{R}$ w przypadku regresji.

a, b, c oznaczają skalary.

n liczba próbek

p liczba atrybutów

Wprowadzenie

Zagadnienie uczenia modelu na podstawie danych można podzielić na trzy grupy: zadanie uczenia, predykcji oraz testowania modelu. Zasadnicza część pracy dotyczy zadania uczenia modeli, głównie w sposób nadzorowany [55]. Celem zadania uczenia jest wybór modelu, który optymalizuje pewną miarę jakości predykcji na danych testowych. Taką miarą w przypadku zadania klasyfikacji może być błąd klasyfikacji, jednak zadanie uczenia zwykle nie jest rozwiązywane poprzez bezpośrednią optymalizację tej miary jakości na danych uczących — zwykle wykorzystuje się tu pewne miary, które są funkcjami ciągłymi lub gładkimi parametrów modelu, natomiast zadanie uczenia sprowadza się do rozwiązania pewnych zadań optymalizacyjnych. Jako funkcję celu wybiera się w najprostszym przypadku funkcję wiarygodności (zwaną czasami entropią krzyżową) lub błąd średniokwadratowy.

Zakres modeli analizowanych w pracy jest ograniczony do modeli liniowych. W przypadku dwóch podstawowych zadań (klasyfikacji i regresji) mamy dwie stosunkowo liczne grupy modeli, tj. klasyfikatory liniowe oraz liniowe modele regresyjne. Zasadnicza cześć metod przedstawionych i analizowanych w pracy odnosi się do obu grup modeli, z zastrzeżeniem, że niektóre zadania w przypadku modeli regresyjnych są nieco prostsze do rozwiązania. W pracy większy nacisk położono na zadania klasyfikacji. Praca ogranicza się do technik strojenia modeli linowych, jednak to ograniczenie nie jest tak duże, jak mogłoby się wydawać. Zagadnienia optymalizacyjne, które pojawiają się przy strojeniu modeli liniowych, mogą być wykorzystywane w różnych modelach analizy danych, jak np. filtracja sygnałów. Przykłady pokazane w pracach [42, 62] wskazują, że omówione techniki strojenia modeli liniowych mogą być również efektywnie wykorzystane do uczenia modeli nieliniowych.

Modele liniowe mają stosunkowo prostą strukturę, tzn. wyjście modelu jest kombinacją liniową zmiennych i współczynników modelu z ewentualnym dodaniem pewnej skalarnej funkcji nieliniowej. Jednak ze względu na różny sposób uczenia (nawet dla tych samych danych) mogą prowadzić do różnych modeli, znacznie różniących się własnościami.

Mimo że współcześnie powszechne jest stosowanie znacznie bardziej złożonych modeli, to należy podkreślić, że istnieją zadania oraz zbiory danych, dla których stosowanie takich złożonych modeli napotyka przeszkody. Jednym z ograniczeń jest złożoność próbkowa[1] modelu [4] — potocznie mówiąc, jest to minimalny rozmiar próby, przy której model może być poprawnie uczony, a której zwiększanie nie prowadzi do wzrostu jakości predykcji na danych testowych. Złożone modele mają zwykle dużą złożoność próbkową, czyli do poprawnego uczenia wymagają dużych prób uczących. Istnieją jednak problemy analizy danych, w których zwiększanie rozmiaru próby jest kłopotliwe lub kosztochłonne (np. dane mikromacierzowe).

Prezentowane w pracy podejścia są zorientowane na przypadki, gdy próby są małe, co należy rozumieć jako małe w relacji do liczby atrybutów oraz gdy próba ucząca nie może być zwiększana w sposób dowolny. Należy podkreślić, że w przypadku stosunkowo prostych modeli liniowych ograniczenia typu Vapnika czy Rademachera [6, 80] gwarantują, że przy wzroście próby do nieskończoności wszystkie modele stają się takie same.

Zachowanie modelu może być kształtowane poprzez dobór odpowiedniej funkcji kryterialnej Q, która w ogólności może się składać z dwóch składników:

$$Q(\boldsymbol{model}; \mathbf{dane}) = L(\boldsymbol{model}; \mathbf{dane}) + \lambda \cdot R(\boldsymbol{model}), \tag{1}$$

tj. składnika odpowiadającego za dopasowanie modelu do danych L oraz składnika regularyzującego model (lub funkcji kary) R. Taką postać funkcji kryterialnej mają główne procedury uczenia modeli liniowych. Przykładowo w modelu regresji liniowej jako funkcję L wykorzystuje się błąd średniokwadratowy, natomiast w zależności od składnika R wyróżniamy model standardowy (bez regularyzacji), regresję grzbietową[2] [33], gdy R jest normą ℓ^2 wektora wag, oraz modele LASSO lub LARS [18, 75, 83], gdy R jest normą ℓ^1 wektora wag. Podobnie w przypadku modelu SVM [13] funkcja L to margines separacji lub suma marginesów próbek, natomiast R oryginalnie jest normą ℓ^2 wektora wag. Historycznie, w przypadku strojenia sieci neuronowych, składnik kary w postaci normy ℓ^2 nazywa się butwieniem wag[3] i jest stosowany od lat dziewięćdziesiątych XX wieku.

Wykorzystanie normy ℓ^2 rozwiązania jako składnika regularyzacyjnego historycznie pochodzi od A. N. Tchionva [78] i była ona użyta do rozwiązywania źle uwarunkowanych lub osobliwych problemów numerycznych. Podobne znaczenie ma dodanie innych składników regularyzacyjnych. Piewsze użycie regularyzacji ℓ^2 w analizie danych pojawiło się

[1] ang. sample complexity
[2] ang. ridge regression
[3] ang. weight decay

z cytowanej pracy dot. regresji grzbietowej [33]. Na znaczenie i numeryczne trudności przy optymalizacji modeli z karą ℓ^1 wskazano po raz pierwszy w pracy [83], gdzie podkreślono między innymi, że w punkcie optimum część współczynników modelu zostaje wyzerowana. Oprócz aspektów numerycznych w modelu liniowym użycie regularyzacji typu ℓ^2 odpowiada założeniu, że parametry modelu mają rozkład gaussowski, natomiast analogicznie zastosowanie regularyzacji ℓ^1 odpowiada założeniu, że parametry modelu mają rozkład Laplace'a [75, 83, 89]. W pracach [75, 89] wskazano dodatkowo na: 1) grupujący efekt normy ℓ^2, tj. współczynniki przy zmiennych silnie skorelowanych mają tendencję do zrównywania się (co w praktyce daje efekt uśredniania atrybutów) oraz 2) ponownie podkreślono fakt, że przy regularyzacji ℓ^1 wraz ze wzrostem parametru regularyzacyjnego coraz więcej współczynników jest zerowanych. Prace te zapoczątkowały intensywne badania różnych form regularyzacji. Zwrócono również uwagę na możliwość stosowania pseudonorm niewypukłych. Niezależnie i nieco wcześniej na geometryczny sens regularyzacji typu ℓ^2 i związek z maksymalizacją marginesu zwrócił uwagę V. Vapnik wraz ze współautorami [13]. Regularyzacja ℓ^1 ze względu na efekt zerowania się współczynników ma związek z problemem selekcji zmiennych. Nawet współcześnie problem wyboru zmiennych w modelach liniowych uważany jest za jeden z istotniejszych w obrębie statystyki (por. [32]). Główne problemy obliczeniowe związane z uczeniem modeli liniowych z regularyzacją dotyczą uwarunkowania macierzy, zwłaszcza w przypadkach, gdy liczba atrybutów jest znaczna oraz gdy w danych jest dużo korelacji. Drugi problem dotyczy zadań uczenia z wykorzystaniem funkcji niegładkich, np. zawierajacej normę ℓ^1 (np. modele lasso, fused lasso, elastic net), natomiast stosowanie pseudonorm $\ell^q, 0 \leqslant q < 1$ dodatkowo prowadzi do niewypukłych problemów optymalizacyjnych.

Należy podkreślić, że regularyzacja nie jest jedyną formą poprawy procesu uczenia oraz własności algorytmów uczących. Obecnie w dobie intensywnego rozwoju sieci głębokich powszechnie stosowane jest wzbogacanie danych[4] [64, 71, 74] czy technika drop out [72].

Cel i teza pracy

Celem badań jest poszukiwanie efektywnych metod uczenia regularyzowanych modeli liniowych przy uwzględnieniu: rozmiaru danych (np. mała próba w stosunku do liczby atrybutów), zależności występujących w danych oraz własności regularyzatora (funkcji kary).

[4] ang. augmentation

Biorąc pod uwagę parametry próby uczącej (takie jak rozmiar danych, rozkład czy korelacje), możliwe jest opracowanie algorytmów uczących dla wybranych problemów uczenia z regularyzacją działających efektywniej w stosunku do istniejących rozwiązań. Pod pojęciem efektywności rozumie się uzyskanie krótszych czasów uczenia przy zachowaniu własności i parametrów modelu i referencyjnego algorytmu uczącego lub uzyskanie wyższej dokładności modeli przez zastosowanie odpowiednich technik regularyzacji.

Główne wyniki uzyskane w pracy

W pracy poruszono zagadnienie uczenia modeli liniowych z regularyzacją, skupiając się na „małych" zbiorach danych, to znaczy takich, w których liczba atrybutów znacznie przewyższa liczbę obserwacji. W takich przypadkach regularyzacja jest jednym z głównych narzędzi poprawy własności numerycznych algorytmów, natomiast różne formy regularyzacji umożliwiają kształtowanie własności maszyn uczących pożądanych z punktu widzenia analizy danych. Najmocniejszy wynik prezentowany w pracy dotyczy usprawnienia algorytmu uczenia modelu regresji z regularyzacją ℓ^2, poprzez zastosowanie rozkładu QR, a w sposób szczególny formuły Woodbury'ego (sekcja 2.4). Pokazano formalnie oraz w eksperymentach numerycznych, że tablicując macierz $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ (gdzie $\boldsymbol{X}$ jest macierzą danych) można — zachowując równoważność algorytmów — istotnie przewyższyć standardową procedurę Newtona obliczania kierunku minimalizacji za pomocą metody gradientu sprzężonego. Ponadto ideę wykorzystania rozkładu QR oraz zastosowania formuły Woodbury'ego zaadaptowano do uczenia modeli z rzadkimi funkcjami kary (typu lasso, elastic net), jednak (co pokazały eksperymenty numeryczne) zysk w tym przypadku nie jest tak spektakularny, jak w przypadku regularyzacji ℓ^2.

Całość eksperymentów wykonano w języku `Python`, biorąc za punkt odniesienia dostępne biblioteki uczenia maszynowego `scikit-learn` i `liblinear` oraz inne algorytmy opisane w literaturze. Większość modeli została samodzielnie zaimplementowana w języku C++ w ramach jednolitego środowiska testowego. W części numerycznej posiłkowano się bibliotekami numerycznej algebry liniowej, takimi jak: `LAPACK`, `Armadillo`, `Intel® Math Kernel Library`. Przeanalizowano szereg ustawień i własności numerycznych algorytmów, jak np. wpływ reprezentacji zmiennopozycyjnej, metody iteracyjnego rozwiązywania układów równań liniowych, sposoby realizacji minimalizacji kierunkowej, wpływ sposobu reprezentacji macierzy danych (gęste, rzadkie). W pracy przedstawiono również zastosowanie technik uczenia modeli liniowych do strojenia modeli nieliniowych.

W ramach badań przygotowano kilka prac dotyczących wybranych aspektów ucze-

nia modeli regularyzowanych oraz część prezentowanych wyników została opublikowana. W pracy [45] analizowano algorytmy polegające zastąpieniu niegładkiej funkcji wartości bezwzględnej jej gładką aproksymacją. W pracach [43, 44] przedstawiono i przeanalizowano zastosowanie niegładkich i niewypukłych „norm" ℓ^q dla $0 \leqslant q < 1$ w zagadnieniach uczenia modeli linowych i filtracji obrazów. W pracy [42] przeanalizowano zastosowanie rozkładu QR dla modeli z regularyzacją ℓ^2 oraz zaproponowano analogiczny algorytm dla innych form regularyzacji, w tym ograniczeń niegładkich. W pracy [62] przedstawiono zastosowanie technik regularyzacyjnych przy uczeniu sieci neuronowych oraz wskazano zastosowanie struktury sieci z regularyzacją typu ℓ^1 do strojenie rzadkiego autoenkodera. Proponowane rozwiązanie porównano z algorytmem uczenia słownikowego zaimplementowanego w pakiecie `scikit-learn`.

1. Modele liniowe w uczeniu maszynowym

1.1 Sformułowanie problemu

Model liniowy opiera się na liniowej kombinacji atrybutów obserwacji $\boldsymbol{x}$ oraz parametrów modelu $\boldsymbol{w}$, co można symbolicznie zapisać:

$$y = w_0 + \sum_{j=1}^{p} x_j \cdot w_j \equiv \boldsymbol{w}^{\mathrm{T}} \cdot \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}. \tag{1.1}$$

Taką postać modelu wykorzystuje się w zadaniach klasyfikacji i regresji (w przypadku klasyfikacji patrzy się zwykle na znak wyrażenia y).

Problem uczenia modelu najczęściej sprowadza się do zadania optymalizacji pewnej funkcji straty (przykładowo może nią być błąd), która to określa stopień dopasowania modelu do danych uczących. Pożądaną własnością funkcji straty jest jej różniczkowalność, co daje możliwość zastosowania gradientowych metod optymalizacji.

Bezpośrednia optymalizacja funkcji straty może nie przynieść oczekiwanych rezultatów z kilku powodów:

- dane zwykle zawierają szum,
- wystarczająco złożony model (o zbyt wielu stopniach swobody) może się dopasować idealnie do próbek (tzw. przeuczenie),
- czasami liczba atrybutów przewyższa liczbę obserwacji (np. dane mikromacierzowe), co w wyniku utrudnia użycie pochodnych drugiego rzędu (niepełny rząd macierzy Hessego).

Jeden ze sposobów rozwiązujący powyższe problemy polega na dodaniu do funkcji straty pewnej funkcji kary — odpowiada to optymalizacji z ograniczeniami. Jej zadanie polega na preferowaniu takich modeli, które spełniają wskazane ograniczenia (np. mniejsza norma kwadratowa parametrów modelu). Czasami natura danych sugeruje rodzaj ograniczenia, które warto zastosować, aby zwiększyć jakość modelu — przykładowo w szeregu czasowym sąsiednie wartości są ze sobą silnie skorelowane, zatem nałożenie ograniczeń na różnice między nimi powinno skutkować nie tylko prostszym, ale i lepszym modelem.

Ostatecznie funkcja kosztu podlegająca optymalizacji ma postać:

$$Q(\boldsymbol{w}; \mathbf{dane}) = L(\boldsymbol{w}; \mathbf{dane}) + \lambda \cdot R(\boldsymbol{w}), \tag{1.2}$$

gdzie $L(\boldsymbol{w}; \mathbf{dane})$ oznacza funkcję straty, $R(\boldsymbol{w})$ oznacza funkcję kary, zaś λ steruje siłą kary. Powyższy problem można także sformułować jako zadanie z ograniczeniami:

$$\begin{cases} L(\boldsymbol{w}; \mathbf{dane}) \to \min \\ \quad R(\boldsymbol{w}) \leqslant \gamma \\ \quad \boldsymbol{w} \in \mathbb{R}^{p+1} \end{cases}. \tag{1.3}$$

Między parametrami λ i γ istnieje jednoznaczne odwzorowanie [29]. Wyraz wolny w_0 nie powinien podlegać karaniu.

1.2 Przejście z modelu liniowego na nieliniowy

Modele nieliniowe potrafią dla pewnych danych osiągnąć dużo lepsze wyniki niż modele liniowe, jednakże są sposoby, które umożliwiają modelom liniowym uzyskiwanie porównywalnych wyników. Literatura notuje co najmniej dwa takie sposoby.

Pierwszy z nich, stosowany powszechnie w kontekście Support Vector Machine, polega za zastosowaniu sztuczki z przekształceniem jądrowym [29]. Opiera się ona na obserwacji, że parametry modelu można wyrazić jako kombinację próbek:

$$\boldsymbol{w} = \sum_{i=1}^{n} \boldsymbol{x}_i \alpha_i = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{\alpha}, \tag{1.4}$$

a także na niejawnym podnoszeniu wymiarowości próbek. Efektem ubocznym tego podejścia jest redukcja wymiaru przestrzeni do n, co w przypadku $p \gg n$ powinno także skutkować skróceniem czasu uczenia.

Drugi z nich, znany jako Extreme Learning Machine [73], bazuje na dwuwarstwowej sieci neuronowej, w której strojeniu podlegają jedynie współczynniki warstwy wyjściowej, natomiast pierwsza warstwa ma ustalone wagi, a liczba neuronów w warstwie ukrytej jest dużo większa od liczby atrybutów. Dzięki temu zabiegowi dane, po przejściu przez pierwszą warstwę, jawnie trafiają do wyższej przestrzeni, w której separacja powinna być łatwiejsza.

2. Uczenie modeli liniowych

Niech dany będzie zbiór par uczących $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, gdzie $\boldsymbol{x}_i \in \mathbb{R}^p$ i $y_i \in \{-1, +1\}$ (w przypadku klasyfikacji binarnej) lub $y_i \in \mathbb{R}$ (w przypadku regresji) — przy tych oznaczeniach p to liczba atrybutów, a n to liczba próbek. Niech $p(\boldsymbol{x}) \equiv \Pr(y = 1)$ oznacza prawdopodobieństwo, że obserwacja $\boldsymbol{x}$ należy do klasy +1.

2.1 Model regresji liniowej

W celu wyznaczenia prostej regresji dokonuje się minimalizacji sumy kwadratów różnic prawdziwej i szacowanej wartości funkcji w punktach — funkcję straty można więc zapisać następująco:

$$L(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)^2 = \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2. \tag{2.1}$$

Istnieje rozwiązanie analityczne powyższego problemu dane wzorem:

$$\boldsymbol{w} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}. \tag{2.2}$$

W praktyce bezpośrednie użycie wzoru (2.2) (nieuwzględniającego regularyzacji) może albo dać słaby wynik (np. punkty odstające silnie zaburzają kierunek prostej), albo się nie powieść (z uwagi na niewystarczającą liczbę punktów) — w tym drugim przypadku nakładając dodatkowe ograniczenie można znaleźć za pomocą rozkładu QR takie rozwiązanie, które ma możliwie najmniejszą normę kwadratową [25].

2.2 Model regresji logistycznej

Do znalezienia postaci prawdopodobieństwa $p(\boldsymbol{x})$, aproksymowanego za pomocą kombinacji liniowej, wykorzystuje się logarytm szansy (logit):

$$\log\left(\frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})}\right) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle. \tag{2.3}$$

Odpowiednie przekształcenie prowadzi do funkcji sigmoidalnej:

$$p(\boldsymbol{x}) = \frac{e^{\langle \boldsymbol{w},\boldsymbol{x}\rangle}}{1+e^{\langle \boldsymbol{w},\boldsymbol{x}\rangle}} = \frac{1}{1+e^{-\langle \boldsymbol{w},\boldsymbol{x}\rangle}}. \tag{2.4}$$

Korzystając z faktu, że zmienna decyzyjna przyjmuje wartość -1 lub $+1$, wyrażenie na prawdopodobieństwo przynależności próbki $\boldsymbol{x}_i$ do klasy y_i można zapisać w jednolitej formie:

$$P(y_i|\boldsymbol{x}_i) = \frac{1}{1+e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}}. \tag{2.5}$$

Współczynniki modelu regresji logistycznej znajduje się za pomocą metody największej wiarygodności. Funkcja wiarygodności modelu regresji logistycznej ma postać:

$$l(\boldsymbol{w}) = \prod_{i=1}^{n} \frac{1}{1+e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}}. \tag{2.6}$$

Jej maksymalizacja nie jest prosta, toteż w praktyce minimalizacji podlega ujemny logarytm funkcji wiarygodności:

$$L(\boldsymbol{w}) = -\sum_{i=1}^{n} \log\left(\frac{1}{1+e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}}\right) = \sum_{i=1}^{n} \log\left(1+e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}\right). \tag{2.7}$$

Jest to funkcja wypukła (kombinacja funkcji wypukłych), więc istnieje globalne minimum. Gradient i macierz Hessego powyższej funkcji względem $\boldsymbol{w}$ mają postać:

$$\frac{\partial L}{\partial \boldsymbol{w}} \equiv \boldsymbol{g} = -\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v}, \tag{2.8}$$

$$\frac{\partial^2 L}{\partial \boldsymbol{w}\partial \boldsymbol{w}^{\mathrm{T}}} \equiv \boldsymbol{H} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X}, \tag{2.9}$$

gdzie $\boldsymbol{v}$ to wektor n-elementowy i:

$$v_i = y_i \cdot \frac{e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}}{1+e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}} \equiv y_i \cdot \frac{1}{1+e^{y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}}, \tag{2.10}$$

a macierz $\boldsymbol{D}$ oznacza macierz diagonalną i $d_{i,i} = \frac{1}{1+e^{-\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}} \cdot \left(1 - \frac{1}{1+e^{-\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}}\right)$. Do rozwiązania układu równań $\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{0}$ (warunek konieczny istnienia ekstremum) trzeba użyć metod numerycznych, ponieważ nie istnieje analityczne rozwiązanie.

W toku obliczeń trzeba uważać na przepełnienie w potęgowaniu, gdy wyrażenie $-y_i \cdot \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle$ osiąga dużą dodatnią wartość (dzieje się to w przypadku obserwacji niepoprawnie klasyfikowanych w danej chwili) — wtedy należy skorzystać z tożsamości:

$$\log\left(1+e^{-y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}\right) = -y_i \cdot \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + \log\left(1+e^{y_i\cdot\langle \boldsymbol{w},\boldsymbol{x}_i\rangle}\right). \tag{2.11}$$

Analogicznie w przypadku obliczania v_i w zależności od znaku tego wyrażenia należy wybrać pierwszą formułę (gdy $y_i \cdot \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle > 0$) lub drugą (w przeciwnym wypadku).

2.3 Idea regularyzacji

W trakcie uczenia modelu zazwyczaj pojawiają się pewne problemy, z którymi trzeba sobie poradzić — należą do nich między innymi:

- szum danych — w przypadku pomiarów wyniki zwykle obarczone są pewnym błędem wynikającym z niedoskonałości urządzeń mierniczych bądź niedokładnego odczytania pomiaru;
- przeuczenie — jeśli model idealnie dopasuje się do danych (co objawia się bezbłędną klasyfikacją próbek uczących), może on utracić zdolność generalizowania (zwracania poprawnego wyniku dla próbek, które nie brały udziału w trakcie uczenia);
- brak jednoznacznego rozwiązania — gdy dane są separowalne lub gdy nie ma wystarczającej liczby obserwacji, granicę decyzyjną można poprowadzić na wiele sposobów.

Zastosowanie regularyzacji, polegającej na dodaniu do funkcji straty pewnej funkcji kary, która ogranicza zbiór wartości, jakie mogą przyjąć parametry modelu, w naturalny sposób niweluje wyżej wymienione problemy. Otrzymana funkcja nosi nazwę funkcji kosztu i można ją zapisać następująco:

$$Q(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \cdot R(\boldsymbol{w}), \tag{2.12}$$

gdzie $L(\boldsymbol{w})$ to funkcja straty (zależna od danych i parametrów modelu), $R(\boldsymbol{w})$ to funkcja kary (zależna od parametrów modelu), zaś λ to parametr decydujący o sile regularyzacji ($\lambda > 0$). Funkcja kary może mieć rozmaitą postać, co przedstawiono w tabeli 2.1.

Ponadto dzięki regularyzacji można nałożyć pewną wiedzę a priori na parametry modelu, na przykład karząc różnice między sąsiednimi wartościami atrybutów, gdy spodziewamy się, że w jednowymiarowym sygnale występują „obszary płaskie”.

2.4 Ograniczenia wypukłe i regularyzacja ℓ^2

Po raz pierwszy o regularyzacji ℓ^2 wspomina się w kontekście rozwiązywania źle uwarunkowanych problemów [78]. W tym przypadku funkcja kary stanowi kwadrat normy euklidesowej (z pominięciem wyrazu wolnego w_0). Optymalizowana funkcja kosztu ma postać:

$$Q(\boldsymbol{w}) = L(\boldsymbol{w}) + \frac{\lambda}{2} \sum_{j=1}^{p} w_j^2. \tag{2.13}$$

Tabela 2.1: Przykładowe funkcje kary.

nazwa kary	postać funkcji $R(\boldsymbol{w})$
ridge (ℓ^2) [29, 78]	$\|\boldsymbol{w}\|_2^2$
LASSO (ℓ^1) [75]	$\|\boldsymbol{w}\|_1$
elastic net [89]	$\alpha\|\boldsymbol{w}\|_1 + \frac{1}{2}(1-\alpha)\|\boldsymbol{w}\|_2^2$
Total Variation (TV) [65]	$\sum_{j=2}^{p} \lvert w_j - w_{j-1}\rvert$
uogólniony TV [10]	$\|\boldsymbol{F}\boldsymbol{w}\|_1$
fused LASSO [76]	$\alpha\|\boldsymbol{w}\|_1 + (1-\alpha)\sum_{j=2}^{p} \lvert w_j - w_{j-1}\rvert$
SCAD [20]	$\sum f(\lvert w_j\rvert),\ f(x) = \begin{cases} x, & x \leqslant \lambda \\ \frac{x^2 - 2a\lambda x + \lambda^2}{2(1-a)\lambda}, & \lambda < x \leqslant a\lambda \\ \frac{\lambda(1+a)}{2}, & a\lambda < x \end{cases}$

W uczeniu maszynowym ten sposób regularyzacji, w kontekście regresji liniowej, funkcjonuje pod anglojęzyczną nazwą ridge regression [29], a rozwiązanie dane jest wzorem:

$$\boldsymbol{w} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}, \tag{2.14}$$

gdzie $\boldsymbol{I}$ to macierz jednostkowa.

Do znalezienia minimum funkcji (2.13) w przypadku regresji logistycznej można posłużyć się dowolną gradientową metodą optymalizacji, zarówno pierwszego jak i drugiego rzędu [54]. W metodzie Newtona w postaci macierzowej wzór na kierunek poszukiwania minimum opisuje poniższe równanie:

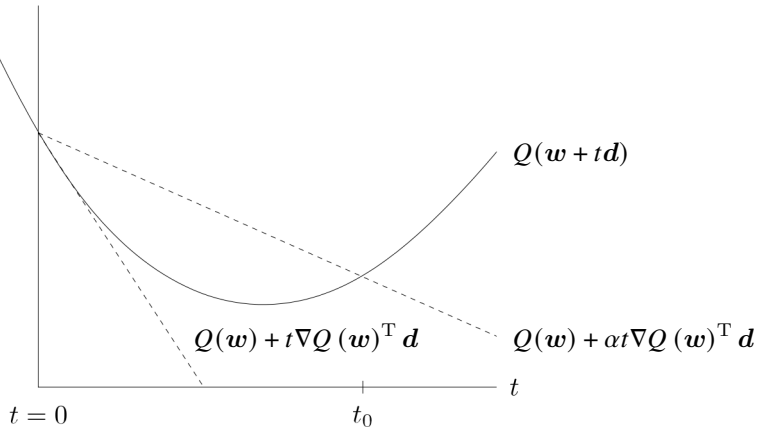
$$\boldsymbol{d} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{w}\right), \tag{2.15}$$

a nowy punkt to $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + t\boldsymbol{d}$, gdzie t stanowi krok znaleziony w wyniku minimalizacji kierunkowej. Samą metodę Newtona można przerwać, gdy $\sqrt{-\nabla Q\left(\boldsymbol{w}^{(k)}\right)^{\mathrm{T}}\boldsymbol{d}} \leqslant \epsilon$, gdzie $\nabla Q\left(\boldsymbol{w}^{(k)}\right)^{\mathrm{T}}\boldsymbol{d}$ to pochodna kierunkowa w k-tej iteracji, a ϵ to dokładność. Jeżeli pochodna kierunkowa jest dodatnia, to kierunek $\boldsymbol{d}$ nie wskazuje kierunku minimalizacji. W części eksperymentalnej powyższy model nazwano `l2_logreg_ordinary`.

Krok t można znajdywać w taki sposób, by wyrażenie $Q(\boldsymbol{w} + t\boldsymbol{d})$ osiągało dokładne minimum (tzw. dokładna minimalizacja kierunkowa[1]):

$$t = \underset{s \geqslant 0}{\arg\min}\, Q(\boldsymbol{w} + s\boldsymbol{d}). \tag{2.16}$$

[1] ang. exact linesearch



Rysunek 2.1: Wrotna minimalizacja kierunkowa. Krzywa przedstawia funkcję kosztu Q ograniczoną do obszaru poszukiwań. Dolna przerywana linia reprezentuje liniową ekstrapolację funkcji, natomiast górna ma współczynnik nachylenia pomniejszony wprost proporcjonalnie do wartości α. Krok z przedziału $[0, t_0]$ spełnia warunek stopu wrotnej minimalizacji kierunkowej [9].

Podejście to można stosować szczególnie wtedy, gdy znalezienie minimum tejże funkcji jednej zmiennej ma niski koszt w porównaniu z kosztem wyznaczenia kierunku poszukiwań [9].

Gdy koszt obliczenia $Q\,(\boldsymbol{w}+t\boldsymbol{d})$ jest duży, w praktyce wystarczy jedynie przybliżona minimalizacja[2]. Jedną z takich metod jest wrotna minimalizacja kierunkowa[3], która zależy od dwóch stałych: $\alpha \in (0;0{,}5)$ i $\beta \in (0,1)$. Nazwa wywodzi się z faktu, że zaczynając od kroku jednostkowego zmniejszamy go proporcjonalnie do wartości czynnika β, aż będzie spełniony warunek stopu $Q(\boldsymbol{w}+t\boldsymbol{d}) < Q(\boldsymbol{w})+\alpha t\nabla Q\,(\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{d}$. Z faktu, że $\boldsymbol{d}$ jest kierunkiem spadku, mamy $\nabla Q\,(\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{d} < 0$, więc dla dostatecznie małego t wynika

$$Q(\boldsymbol{w}+t\boldsymbol{d}) \approx Q(\boldsymbol{w})+t\nabla Q\,(\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{d} < Q(\boldsymbol{w})+\alpha t\nabla Q\,(\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{d},$$

co pokazuje, że wrotna minimalizacja kierunkowa w końcu się zatrzyma [9]. Podejście to podsumowuje algorytm 1. Wyliczając wcześniej wartość iloczynu $\boldsymbol{X}\boldsymbol{d}$ koszt obliczenia $Q(\boldsymbol{w}+t\boldsymbol{d})$ i $\nabla Q\,(\boldsymbol{w}+t\boldsymbol{d})^{\mathrm{T}}\,\boldsymbol{d}$ dla dowolnego t ma złożoność rzędu $O\,(n)$ [22]. W części eksperymentalnej modele, podczas uczenia których wykorzystano dokładną minimalizację kierunkową, opatrzono w nazwie symbolem `e+`, natomiast `e-` oznacza, że użyto przybliżoną minimalizację kierunkową w toku optymalizacji funkcji celu.

Do znalezienia kierunku $\boldsymbol{d}$ nie trzeba w sposób jawny odwracać macierz Hessego

[2] ang. inexact linesearch
[3] ang. backtracking line search

Algorytm 1 Wrotna minimalizacja kierunkowa.

Wejście: kierunek spadku $\boldsymbol{d}$, $\alpha \in (0; 0{,}5)$, $\beta \in (0, 1)$.
$t \leftarrow 1$
while $Q(\boldsymbol{w} + t\boldsymbol{d}) > Q(\boldsymbol{w}) + \alpha t \nabla Q\,(\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{d}$ **do**
  $t \leftarrow \beta t$
end while

— zamiast tego stosuje się iteracyjną metodę gradientu sprzężonego w celu rozwiązania układu równań:

$$\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{d} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{w}, \tag{2.17}$$

ponieważ macierz układu równań jest dodatnio określona [5, 31]. Tempo zbieżności metody zależy od widma macierzy, zatem w celu zwiększenia tempa należy przekształcić układ równań do równoważnej postaci, w której poprawieniu uległo widmo — uzyskuje się to za pomocą pewnej macierzy[4] $\boldsymbol{M}$, która w jakimś stopniu aproksymuje macierz $\boldsymbol{H}$, zaś rozwiązać należy układ równań

$$\boldsymbol{M}^{-1}\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{M}^{-1}\boldsymbol{g}, \tag{2.18}$$

mający takie samo rozwiązanie, co $\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}$, jednak w tym przypadku macierz $\boldsymbol{M}^{-1}\boldsymbol{H}$ posiada korzystniejsze spektrum. Im lepiej macierz $\boldsymbol{M}$ aproksymuje macierz $\boldsymbol{H}$, tym większe tempo zbieżności, ale koszt obliczenia $\boldsymbol{M}^{-1}$ wzrasta, dlatego znaleźć trzeba kompromis między dokładnością a złożonością. W najprostszym przypadku można za $\boldsymbol{M}$ przyjąć macierz diagonalną, która zawiera elementy z przekątnej macierzy $\boldsymbol{H}$ (tzw. preconditioner Jacobiego). Bardziej wyrafinowane postaci macierzy $\boldsymbol{M}$ opierają się na niekompletnej faktoryzacji [5, 66]. W części eksperymentalnej modele używające tego udoskonalenia mają w nazwie dopisek `p+`, a w przypadku $\boldsymbol{M} = \boldsymbol{I}$ do nazwy dopisano `p-`.

Szkic metody gradientu sprzężonego prezentuje algorytm 2. Metoda ta wymaga tylko procedury obliczającej iloczyn macierzy układu równań przez dowolny wektor $\boldsymbol{s}$ — opierając się na prawie łączności iloczynu można go rozpisać jako trzy iloczyny macierzowo-wektorowe w przypadku regresji logistycznej z regularyzacją ℓ^2:

$$\boldsymbol{H}\boldsymbol{s} = \boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{D}\left(\boldsymbol{X}\boldsymbol{s}\right)\right) + \lambda\boldsymbol{s}. \tag{2.19}$$

Ponadto w początkowych iteracjach wystarczy jedynie zgrubne wyznaczanie kierunku — takie podejście prowadzi do „truncated Newton” [52].

[4] W języku angielskim o takiej macierzy mówi się preconditioner [25].

Algorytm 2 Metoda gradientu sprzężonego [5].

Wejście: $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{x}^{(0)}$ (wybrany dowolnie, np. $\mathbf{0}$), ϵ, $i_{\max}$
$\boldsymbol{r}^{(0)} \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$.
Rozwiąż układ równań $\boldsymbol{M}\boldsymbol{z}^{(0)} = \boldsymbol{r}^{(0)}$.
$\rho^{(0)} \leftarrow \langle \boldsymbol{r}^{(0)}, \boldsymbol{z}^{(0)} \rangle$.
$\boldsymbol{p}^{(1)} \leftarrow \boldsymbol{z}^{(0)}$.
for $i = 1, 2, \ldots$ **do**
  $\alpha^{(i)} \leftarrow \frac{\rho^{(i-1)}}{\langle \boldsymbol{p}^{(i)}, \boldsymbol{A}\boldsymbol{p}^{(i)} \rangle}$.
  $\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i-1)} + \alpha^{(i)}\boldsymbol{p}^{(i)}$.
  $\boldsymbol{r}^{(i)} \leftarrow \boldsymbol{r}^{(i-1)} - \alpha^{(i)}\boldsymbol{A}\boldsymbol{p}^{(i)}$.
  if $\|\boldsymbol{r}^{(i)}\|_2 \leqslant \epsilon\|\boldsymbol{b}\|_2 \vee i \geqslant i_{\max}$ **then**
    Zakończ.
  end if
  Rozwiąż układ równań $\boldsymbol{M}\boldsymbol{z}^{(i)} = \boldsymbol{r}^{(i)}$.
  $\rho^{(i)} \leftarrow \langle \boldsymbol{r}^{(i)}, \boldsymbol{z}^{(i)} \rangle$.
  $\beta^{(i)} \leftarrow \frac{\rho^{(i)}}{\rho^{(i-1)}}$.
  $\boldsymbol{p}^{(i+1)} \leftarrow \boldsymbol{z}^{(i)} + \beta^{(i)}\boldsymbol{p}^{(i)}$.
end for
Wyjście: rozwiązanie układu równań $\boldsymbol{x}$.

Problem można także wyrazić w formie ważonych najmniejszych kwadratów, a następnie iteracyjnie rozwiązywać w celu uzyskania zbieżności [27]. Sprowadzenie do problemu najmniejszych kwadratów umożliwia też zastosowanie sztuczki z przekształceniem jądrowym [63].

Ponadto można z powodzeniem stosować metody gradientowe pierwszego rzędu do optymalizacji funkcji (2.13) [16, 68].

Włączając wyraz wolny w_0 równanie określające kierunek poszukiwań przybieże następującą postać:

$$\begin{bmatrix} d_0 \\ \boldsymbol{d} \end{bmatrix} = \begin{bmatrix} \mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1} & \mathbf{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\mathbf{1} & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \mathbf{1}^{\mathrm{T}}\boldsymbol{v} \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{w} \end{bmatrix}. \tag{2.20}$$

Ponownie iloczyn powyższej macierzy i dowolnego wektora można rozpisać jako trzy iloczyny macierzowo-wektorowe (opierając się na prawie łączności iloczynu), tym samym unikając jawnego formowania macierzy. W części eksperymentalnej modele zawierające wyraz wolny mają w nazwie `b+`, natomiast te bez wyrazu wolnego mają `b-`.

2.4.1 Zastosowanie rozkładu QR

W wyniku rozkładu QR macierzy rzeczywistej $\boldsymbol{X}_{n\times p}$ $(n \geqslant p)$ o pełnym rzędzie powstaje macierz ortogonalna $\boldsymbol{Q}_{n\times n}$ i macierz górnotrójkątna $\boldsymbol{R}_{n\times p}$:

$$\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R} = \begin{bmatrix}\boldsymbol{Q}_1 & \boldsymbol{Q}_2\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{R}_1 \\ \boldsymbol{0}\end{bmatrix} = \boldsymbol{Q}_1\boldsymbol{R}_1, \tag{2.21}$$

a do wyznaczenia tego rozkładu można posłużyć się np. odbiciami Householdera. Rozkład $\boldsymbol{Q}_1\boldsymbol{R}_1$ bywa nazywany cienkim rozkładem z uwagi na mniejsze wymiary wynikowych macierzy. W przypadku $n < p$ należy skorzystać z analogicznego rozkładu LQ bądź dokonać rozkład QR macierzy transponowanej i uwzględnić transpozycję w dalszych obliczeniach. Liczba operacji zmiennoprzecinkowych wymaganych do policzenia rozkładu QR za pomocą odbić Householdera wynosi $2p^2\left(n - \frac{p}{3}\right)$ [25].

W przypadku regresji liniowej bez regularyzacji możemy znaleźć formułę na współczynniki za pomocą rozkładu QR [55]:

$$\boldsymbol{w} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \left(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{R}\right)^{-1}\boldsymbol{R}^{\mathrm{T}}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{R}_1^{-1}\boldsymbol{Q}_1^{\mathrm{T}}\boldsymbol{y}. \tag{2.22}$$

Jeśli $n < p$, można za pomocą rozkładu LQ znaleźć rozwiązanie, które będzie miało możliwie najmniejszą normę [25]:

$$\boldsymbol{w} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \left(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{Q}\right)^{-1}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{Q}_1^{\mathrm{T}}\boldsymbol{L}_1^{-1}\boldsymbol{y}. \tag{2.23}$$

Dla regresji liniowej z regularyzacją, po podstawieniu $\boldsymbol{L}\boldsymbol{Q}$ w miejsce $\boldsymbol{X}$ w (2.14) uzyskamy:

$$\begin{aligned}\boldsymbol{w} &= \left(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{Q} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{Q}^{\mathrm{T}}\left(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{y} = \\ &= \begin{bmatrix}\boldsymbol{Q}_1^{\mathrm{T}} & \boldsymbol{Q}_2^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{L}_1 + \lambda\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \lambda\boldsymbol{I}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{y} \\ \boldsymbol{0}\end{bmatrix} = \boldsymbol{Q}_1^{\mathrm{T}}\left(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{L}_1 + \lambda\boldsymbol{I}\right)\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{y}.\end{aligned}$$

Bazując na fakcie, że regresja logistyczna z karą ℓ^2 jest obrotowo niezmiennicza[5] [57], można w przypadku $p \gg n$ macierz danych $\boldsymbol{X}$ poddać rozkładowi LQ (bądź równoważnie macierz $\boldsymbol{X}^{\mathrm{T}}$ rozkładowi QR) i tym samym zredukować wymiar problemu do n — powrót do pierwotnej przestrzeni uzyskuje się mnożąc wynik przez macierz $\boldsymbol{Q}^{\mathrm{T}}$ (bądź $\boldsymbol{Q}$ w przypadku rozkładu QR) [28].

[5] Algorytm nazywamy obrotowo niezmienniczym, gdy jego wynik działania (np. wynik klasyfikacji czy prawdopodobieństwa posteriori) są identyczne, zarówno dla danych oryginalnych, jak i dla danych obróconych, tj. przemnożonych przez macierz ortogonalną.

Algorytm 3 Uczenie modelu regresji logistycznej z regularyzacją ℓ^2 za pomocą metody Newtona wykorzystującej rozkład LQ.

Dane wejściowe: $\boldsymbol{X} = \mathbb{R}^{n\times p}$, $\boldsymbol{y}_{n\times 1} : y_i \in \{-1, +1\}$, $n < p$
Inicjalizacja: $[\boldsymbol{L}_1, \boldsymbol{Q}_1] = \mathrm{lq}\,(\boldsymbol{X})$, $\hat{\boldsymbol{w}}^{(0)}_{n\times 1} = \boldsymbol{0}$
for $k = 0, 1, \ldots, k_{\max}$ **do**
  Oblicz wartości $\boldsymbol{v}$ (zgodnie z (2.10)) i $\boldsymbol{D}$ dla bieżącego $\hat{\boldsymbol{w}}^{(k)}$.
  Rozwiąż układ równań $\left(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}_1 + \lambda\boldsymbol{I}\right)\cdot\hat{\boldsymbol{d}} = \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{v} - \lambda\hat{\boldsymbol{w}}$.
  if $\sqrt{\langle \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{v} - \lambda\hat{\boldsymbol{w}}^{(k)}, \hat{\boldsymbol{d}}\rangle} < \epsilon$ **then**
    Zakończ pętlę i zwróć wynik.
  end if
  $\hat{\boldsymbol{w}}^{(k+1)} \leftarrow \hat{\boldsymbol{w}}^{(k)} + \hat{\boldsymbol{d}}\cdot\arg\min_t Q\left(\hat{\boldsymbol{w}}^{(k)} + t\hat{\boldsymbol{d}}\right)$.
end for
Wyjście: $\boldsymbol{w} = \boldsymbol{Q}_1^{\mathrm{T}}\hat{\boldsymbol{w}}$

Podstawiając $\boldsymbol{L}_1$ za $\boldsymbol{X}$ w równaniu (2.15) uzyskujemy na mocy twierdzenia pierwszego zawartego w [28] poniższą tożsamość:

$$\boldsymbol{d} = \boldsymbol{Q}_1^{\mathrm{T}}\left(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}_1 + \lambda\boldsymbol{I}\right)^{-1}\left(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{Q}_1\boldsymbol{w}\right). \tag{2.24}$$

Przemnożenie przez $\boldsymbol{Q}_1$ rzutuje $\boldsymbol{w}$ do mniejszej przestrzeni, w której następuje rozwiązanie układu równań, po czym mnożenie przez $\boldsymbol{Q}_1^{\mathrm{T}}$ przywraca wynik do pierwotnej przestrzeni. Nie trzeba za każdym razem wracać do oryginalnej przestrzeni — wszystkie obliczenia można dokonać w mniejszej przestrzeni, a dopiero końcowy rezultat cofnąć do pierwotnej przestrzeni. Powyższe podejście przedstawiono w algorytmie 3.

W celu włączenia wyrazu wolnego w_0 należy postąpić analogicznie, jak miało to miejsce w równaniu (2.20), z tą różnicą, że macierz $\boldsymbol{X}$ zastępujemy macierzą $\boldsymbol{L}_1$:

$$\begin{bmatrix} d_0 \\ \hat{\boldsymbol{d}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{1} & \boldsymbol{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}_1 \\ \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{D}\boldsymbol{1} & \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{D}\boldsymbol{L}_1 + \lambda\boldsymbol{I} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \boldsymbol{1}^{\mathrm{T}}\boldsymbol{v} \\ \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{v} - \lambda\hat{\boldsymbol{w}} \end{bmatrix}. \tag{2.25}$$

Wykorzystanie rozkładu QR nie jest czymś zupełnie nowym [28, 57], jednakże zastosowanie tej sztuczki wydaje się być mało powszechne w bibliotekach uczenia maszynowego — przykładowo technika ta nie jest wykorzystywana w pakiecie `scikit-learn` [61]. Możemy jednak w prosty sposób przerobić model regresji logistycznej z regularyzacją ℓ^2, rozszerzając oryginalną klasę i przeciążając metodę `fit`. Ilustruje to poniższy fragment kodu:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
class LR_with_lq(LogisticRegression):
    def __init__(self, **kwargs):
        assert kwargs['penalty'] == 'l2'
        super().__init__(**kwargs)
    def fit(self, X, y):
        if X.shape[0] < X.shape[1]:
            Q, R = np.linalg.qr(X.T, mode = 'reduced')
            super().fit(R.T, y)
            self.coef_ = self.coef_ @ Q.T
        else:
            super().fit(X, y)
        return self
```

Takie usprawnienie powinno dawać zysk przy liczbie próbek mniejszej niż liczba atrybutów. Co więcej, ponieważ transformacja $\boldsymbol{R}$ do $\boldsymbol{X}$ za pomocą macierzy $\boldsymbol{Q}$ ($\boldsymbol{X} = \boldsymbol{QR}$), jest przekształceniem ortogonalnym, to może być ona wykorzystywana we wszystkich algorytmach, które są obrotowo niezmiennicze. Krótki eksperyment ze zbiorem danych `colon` o rozmiarze 62×2000 pokazuje zysk ok. dwuipółkrotny w czasie uczenia, uwzględniając koszt rozkładu QR.

W części eksperymentalnej (rodz. 4) model ten widnieje pod nazwą `l2_logreg_qr` (gdy użyto rozkładu QR) bądź `l2_logreg_lq` (gdy użyto rozkładu LQ).

2.4.2 Zastosowanie formuły Woodbury'ego

Dzięki rozkładowi QR rozmiar rozwiązywanego problemu redukuje się do liczby próbek, a także zyskujemy stabilność numeryczną procedury uczenia, jednak w przypadku macierzy rzadkich wyznaczenie rozkładu, który zachowa pierwotną rzadkość macierzy, wymaga większego nakładu pracy. Traktowanie macierzy rzadkiej jako gęstej może również nie przynieść zysku obliczeniowego. Mimo to efekt redukcji wielkości problemu można uzyskać po zastosowaniu formuły Woodbury'ego, pozwalającej obliczyć odwrotność sumy dwóch macierzy [30, 84]:

$$(\boldsymbol{A} + \boldsymbol{UBV})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}\left(\boldsymbol{B}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}. \tag{2.26}$$

Podstawiając $\boldsymbol{A} = \lambda\boldsymbol{I}$, $\boldsymbol{U} = \boldsymbol{X}^{\mathrm{T}}$, $\boldsymbol{B} = \boldsymbol{D}$, $\boldsymbol{V} = \boldsymbol{X}$ otrzymamy:

$$\left(\lambda\boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X}\right)^{-1} = \frac{1}{\lambda}\boldsymbol{I} - \frac{1}{\lambda^2}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{D}^{-1} + \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)^{-1}\boldsymbol{X}. \tag{2.27}$$

Macierz $\frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ nie zależy w ogóle od parametrów modelu $\boldsymbol{w}$, więc przed rozpoczęciem optymalizacji należy ją policzyć, by skrócić czas rozwiązywania układu równań — z uwagi na jej symetryczność wystarczy przechowywać górny lub dolny trójkąt macierzy, co pozwoli dwukrotnie zmniejszyć zajętość w pamięci. Koszt wyznaczenia tej macierzy jest rzędu $O\left(n^2 p\right)$, natomiast w przypadku macierzy rzadkiej koszt ten będzie mniejszy proporcjonalnie do stopnia rzadkości. Trzeba też uważać na wartości bliskie zeru na przekątnej macierzy $\boldsymbol{D}$, gdy prawdopodobieństwo przynależności i-tej próbki do klasy pozytywnej lub negatywnej dąży do zera lub jedności — można tego uniknąć prostym zabezpieczeniem: $\min\left(\max\left(d_{i,i}, \epsilon\right), 1-\epsilon\right)$, przyjmując np. $\epsilon = 10^{-5}$ [21].

Wstawiając (2.27) do (2.15) i porządkując wszystko otrzymamy następujące równanie określające kierunek poszukiwań:

$$\boldsymbol{d} = \frac{1}{\lambda}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{v} - \boldsymbol{z}\right) - \boldsymbol{w}, \tag{2.28}$$

gdzie $\boldsymbol{z}$ stanowi rozwiązanie poniższego układu równań:

$$\left(\boldsymbol{D}^{-1} + \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)\boldsymbol{z} = \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{X}\boldsymbol{w}. \tag{2.29}$$

Pomimo stosunkowo niedużego rozmiaru macierzy, metoda dokładna rozwiązywania układu równań może okazać się bardziej kosztowna od metody iteracyjnej — koszt policzenia rozkładu Cholesky'ego wymaga $\frac{n^3}{3}$ operacji zmiennoprzecinkowych, natomiast metoda iteracyjna wymaga w przybliżeniu kn^2 operacji, gdzie k oznacza liczbę iteracji. Uzyskanie satysfakcjonującego rozwiązania wymaga zazwyczaj mniejszej liczby iteracji niż n. Pewien problem ze zbieżnością metody iteracyjnej może tutaj powodować złe uwarunkowanie macierzy, więc nieodzowne może okazać się zastosowanie preconditioningu.

Aby znaleźć wzór na kierunek poszukiwań uwzględniający wyraz wolny, wprowadźmy $\boldsymbol{b} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}$, $c = \mathbf{1}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}$ i zapiszmy (2.20) w postaci układu równań:

$$\begin{cases} cd_0 + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{d} = \mathbf{1}^{\mathrm{T}}\boldsymbol{v} \\ \boldsymbol{b}d_0 + \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{d} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{w} \end{cases}. \tag{2.30}$$

Wyznaczając z pierwszego równania d_0 i wstawiając do drugiego równania otrzymamy:

$$\begin{cases} d_0 = \frac{\mathbf{1}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{d}}{c} \\ \boldsymbol{d} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I} - \frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}\right)^{-1}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{w} - \frac{\mathbf{1}^{\mathrm{T}}\boldsymbol{v}}{c}\boldsymbol{b}\right) \end{cases}. \tag{2.31}$$

Podobnie jak w (2.27), podstawiając $\boldsymbol{A} = \lambda\boldsymbol{I} - \frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$ i korzystając z formuły Shermana-Morrisona do policzenia $\boldsymbol{A}^{-1}$ [70], a następnie z formuły Woodbury'ego do policzenia

$\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I} - \frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}\right)^{-1}$ i wstawiając wynik do drugiego równania w (2.31) ostatecznie otrzymamy (dla czytelności wprowadzono zmienne pomocnicze):

$$\begin{aligned}
\boldsymbol{s} &= \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\mathbf{1}, \\
\gamma &= \frac{\langle \boldsymbol{D}\mathbf{1}, \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{X}\boldsymbol{w}\rangle - \mathbf{1}^{\mathrm{T}}\boldsymbol{v}}{c - \langle \boldsymbol{D}\mathbf{1}, \boldsymbol{s}\rangle}, \\
\boldsymbol{z} &= \left(\boldsymbol{D}^{-1} + \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} + \frac{\boldsymbol{s}\boldsymbol{s}^{\mathrm{T}}}{c - \langle \boldsymbol{D}\mathbf{1}, \boldsymbol{s}\rangle}\right)^{-1}\left(\frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{X}\boldsymbol{w} + \gamma\boldsymbol{s}\right), \\
\boldsymbol{d} &= \frac{1}{\lambda}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{v} - \boldsymbol{z} + \left(\gamma - \frac{\langle \boldsymbol{z}, \boldsymbol{s}\rangle}{c - \langle \boldsymbol{D}\mathbf{1}, \boldsymbol{s}\rangle}\right)\boldsymbol{D}\mathbf{1}\right) - \boldsymbol{w}.
\end{aligned} \tag{2.32}$$

Mając $\boldsymbol{d}$ można z łatwością policzyć d_0 zgodnie z (2.31). Należy pamiętać, że do iloczynu $\boldsymbol{X}\boldsymbol{w}$ nie dodano w_0. Włączenie wyrazu wolnego wiąże się z dodatkowym kosztem policzenia wektora $\boldsymbol{s}$ i kilkoma iloczynami skalarnymi. Dodatkowo macierz $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I} - \frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$ może być nieokreślona, dlatego zamiast metody gradientu sprzężonego do rozwiązania powyższego układu równań należy użyć metodę, która działa poprawnie w przypadku macierzy nieokreślonych, np. metoda `MINRES` bądź `SYMMLQ` [60].

W części eksperymentalnej zbadano dwa modele — w pierwszym obliczono i zapamiętano iloczyn $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$, by wykorzystać go w kolejnych iteracjach (ten model nazwano `l2_logreg_wdbr`), a w drugim nie skorzystano z tej optymalizacji (ten z kolei nazwano `l2_logreg_woodbury`).

2.5 Regularyzacja ℓ^1

Idea zastosowania normy ℓ^1 w zadaniu regularyzacji i przycinania pojawiła się w latach 90. [83], a później ukuto dla niej nazwę „lasso” w kontekście regresji liniowej [75]. W przypadku regresji liniowej optymalizowana funkcja kosztu wygląda następująco:

$$Q(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle)^2 + \lambda\sum_{j=1}^{p}|w_j|, \tag{2.33}$$

natomiast w przypadku regresji logistycznej funkcja kosztu ma postać:

$$Q(\boldsymbol{w}) = \sum_{i=1}^{n}\log\left(1 + e^{-y_i\cdot\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle}\right) + \lambda\sum_{j=1}^{p}|w_j|. \tag{2.34}$$

Funkcja w punkcie $w_j = 0$ posiada ostrze (pochodna nie istnieje), co przyczynia się do powstawania rzadkich modeli (część parametrów modelu zostaje wyzerowana). Brak gładkości uniemożliwia bezpośrednie zastosowanie metod gradientowych, jednakże wprowadzenie pewnych modyfikacji niweluje to ograniczenie.

2.5.1 Subgradient

Zróżniczkowanie funkcji kosztu zawierającej karę ℓ^1 względem zmiennej w_j prowadzi do następujących przypadków:

$$\frac{\partial Q(\boldsymbol{w})}{\partial w_j} = \begin{cases} \frac{\partial L(\boldsymbol{w})}{\partial w_j} + \lambda, & w_j > 0 \\ \frac{\partial L(\boldsymbol{w})}{\partial w_j} - \lambda, & w_j < 0 \\ \frac{\partial L(\boldsymbol{w})}{\partial w_j} - \lambda, & w_j = 0 \ \wedge \ \frac{\partial L(\boldsymbol{w})}{\partial w_j} > \lambda \\ \frac{\partial L(\boldsymbol{w})}{\partial w_j} + \lambda, & w_j = 0 \ \wedge \ \frac{\partial L(\boldsymbol{w})}{\partial w_j} < -\lambda \\ 0, & \text{wpr.} \end{cases} \tag{2.35}$$

Jeśli $L(\boldsymbol{w})$ jest funkcją kwadratową, to pochodna funkcji $Q(\boldsymbol{w})$ jest funkcją kawałkami liniową (ang. piecewise linear). Dzięki tej obserwacji można skonstruować wydajny algorytm znajdujący minimum funkcji $Q(\boldsymbol{w})$, którego złożoność odpowiada złożoności zwyczajnej (bez regularyzacji) metody najmniejszych kwadratów [18]. Analogiczne podejście można zastosować w przypadku regresji logistycznej, formułując funkcję straty w postaci ważonej funkcji kwadratowej [49].

Wykorzystując fakt, że pochodna istnieje poza zerem (czyli poza osiami układu współrzędnych), można znaleźć minimum lokalnej aproksymacji kwadratowej, a następnie dokonać projekcji znalezionego rozwiązania na osie układu współrzędnych, aby część parametrów, które mają być rzeczywiście wyzerowane, zostały faktycznie wyzerowane [3, 26]. Alternatywne metody iteracyjnego ściągania-progowania można znaleźć w pracach [7, 11].

2.5.2 Spadek względem współrzędnych

Modyfikacja polega na optymalizacji jednego parametru (lub bloku parametrów) przy ustalonych pozostałych parametrach [21, 79]. Optymalizacji podlega albo bezpośrednio funkcja kosztu, albo jej lokalna aproksymacja kwadratowa [87]. W przypadku regresji liniowej optymalizacja pojedynczego parametru sprowadza się do miękkiego progowania[6]:

$$w_j^* \leftarrow \frac{\mathcal{S}_\lambda\left(\sum_i x_{i,j} \cdot \left(y_i - \sum_{k \neq j} w_k \cdot x_{i,k}\right)\right)}{\sum_i x_{i,j}^2}, \tag{2.36}$$

gdzie $\mathcal{S}_\lambda(z)$ jest operatorem miękkiego progowania:

$$\mathcal{S}_\lambda(z) = \operatorname{sgn}(z) \cdot \max(|z| - \lambda, 0). \tag{2.37}$$

[6] ang. soft thresholding

W tej postaci koszt korekty pojedynczego współczynnika wynosi $O(np)$. Można zauważyć, że:

$$\begin{aligned} y_i - \sum_{k \neq j} w_k \cdot x_{i,k} &= y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + w_j \cdot x_{i,j} \\ &= r_i + w_j \cdot x_{i,j}, \end{aligned} \tag{2.38}$$

gdzie $\boldsymbol{r}$ jest wektorem reszt [21]. Tym sposobem formuła (2.36) upraszcza się:

$$w_j^* \leftarrow \frac{\mathcal{S}_\lambda \left(\sum_i x_{i,j} \cdot r_i + w_j \cdot \sum_i x_{i,j}^2 \right)}{\sum_i x_{i,j}^2}, \tag{2.39}$$

redukując złożoność do $O(n)$. Wartości $\sum_i x_{i,j}^2$ można policzyć raz i zapamiętać. Korekta i-tej współrzędnej wektora reszt wygląda następująco:

$$r_i \leftarrow r_i + \left(w_j - w_j^* \right) \cdot x_{i,j}. \tag{2.40}$$

W przypadku regresji logistycznej, aproksymowanej lokalnie funkcją kwadratową, formuła na koretkę j-tego parametru ma postać:

$$w_j^* \leftarrow \frac{\mathcal{S}_\lambda \left(\sum_i x_{i,j} \cdot r_i + w_j \cdot \sum_i d_{i,i} \cdot x_{i,j}^2 \right)}{\sum_i d_{i,i} \cdot x_{i,j}^2}. \tag{2.41}$$

Na początku $\boldsymbol{r} = \boldsymbol{v}$ (2.8), $d_{i,i}$ to i-ty element macierzy diagonalnej $\boldsymbol{D}$ (2.9), zaś korekta i-tej współrzędnej wektora reszt wygląda następująco:

$$r_i \leftarrow r_i + \left(w_j - w_j^* \right) \cdot d_{i,i} \cdot x_{i,j}. \tag{2.42}$$

Aby dalej usprawnić proces minimalizacji, należy na początku wyznaczyć zbiór aktywnych (niezerowych) parametrów, a następnie operować jedynie na nich. Jeśli w trakcie obliczeń dany parametr osiągnie wartość 0, to jest on usuwany ze zbioru [21, 86]. Ponadto mieszanie kolejności optymalizowania poszczególnych parametrów modelu również zazwyczaj sprzyja skracaniu czasu uczenia [86].

Po znalezieniu minimum trzeba udać się do nowego punktu, obliczyć nowe wartości wag $\boldsymbol{v}$ i $\boldsymbol{D}$ i powtórzyć cały proces — podsumowuje to algorytm 4. Jako nowy punkt można wziąć znalezione minimum $\boldsymbol{w}^*$ bądź punkt leżący między poprzednim minimum $\boldsymbol{w}^{(k)}$ a znalezionym, minimalizujący wyrażenie (2.34). Eksperymenty pokazują, że to drugie podejście przeważa nad pierwszym [86].

W początkowych iteracjach nie trzeba dokładnie rozwiązywać problemu, toteż warunek stopu dla wewnętrznej pętli metody spadku względem współrzędnych powinien

Algorytm 4 Uczenie modelu regresji logistycznej z regularyzacją ℓ^1 za pomocą metody spadku względem współrzędnych.

Dane wejściowe: $\boldsymbol{X} = \mathbb{R}^{n \times p}$, $\boldsymbol{y}_{n \times 1}$
Inicjalizacja: $\boldsymbol{w}^{(0)} \leftarrow \boldsymbol{0}$, $\delta \leftarrow 1$, $k \leftarrow 0$
repeat
  Oblicz wartości $\boldsymbol{g}$, $\boldsymbol{D}$, $\boldsymbol{r}$ dla bieżącego $\boldsymbol{w}^{(k)}$.
  Wyznacz zbiór $\mathcal{J}$ indeksów aktywnych (niezerowych) parametrów.
  $\boldsymbol{w}^* \leftarrow \boldsymbol{w}^{(k)}$.
  $n_{\text{it}} \leftarrow 0$.
  repeat
    $\Delta_{\max} \leftarrow 0$.
    Wymieszaj kolejność indeksów w zbiorze $\mathcal{J}$.
    for all $j \in \mathcal{J}$ **do**
      $w_{\text{opt}} \leftarrow \frac{\mathcal{S}_\lambda\left(\sum_i x_{i,j} \cdot r_i + w_j^* \cdot \sum_i d_{i,i} \cdot x_{i,j}^2\right)}{\sum_i d_{i,i} \cdot x_{i,j}^2}$.
      $\boldsymbol{r} \leftarrow \boldsymbol{r} + \left(w_j^* - w_{\text{opt}}\right) \cdot \boldsymbol{D} \cdot \boldsymbol{x}_{:,j}$.
      $\Delta_{\max} \leftarrow \max\left(\Delta_{\max}, \left|w_j^* - w_{\text{opt}}\right|\right)$.
      $w_j^* \leftarrow w_{\text{opt}}$.
    end for
    $n_{\text{it}} \leftarrow n_{\text{it}} + 1$.
  until $\Delta_{\max} < \delta$
  if $n_{\text{it}} = 1$ **then**
    $\delta \leftarrow \frac{\delta}{4}$.
  end if
  $\beta \leftarrow \arg\min_t Q\left(t\boldsymbol{w}^* + (1-t)\,\boldsymbol{w}^{(k)}\right)$.
  $\boldsymbol{w}^{(k+1)} = \beta \boldsymbol{w}^* + (1-\beta)\,\boldsymbol{w}^{(k)}$.
  $k \leftarrow k + 1$.
until $\|\partial \boldsymbol{Q}\|_2 < \epsilon$

się zmieniać w razie potrzeby (gdy po jednym obiegu pętli maksymalna zmiana $|\Delta_{\max}|$ okazuje się mniejsza niż początkowy próg δ, to jest on zmniejszany czterokrotnie [86]).

W części eksperymentalnej przetestowano wersję algorytmu znajdującą się w bibliotece `liblinear` (nazwano ją `l1_logreg_liblinear`).

2.5.3 Przekształcenie w problem gładkiej optymalizacji

Inne podejście opiera się na zastąpieniu normy ℓ^1 liniowymi ograniczeniami. Zrealizować można to dwojako:

– zastępując zmienną nieograniczoną w_j parą zmiennych ograniczonych $w_j^+ - w_j^-$ [69]:

$$\begin{cases} \sum_{i=1}^{n} \log\left(1 + e^{-y_i \cdot \langle \boldsymbol{w}^+ - \boldsymbol{w}^-, \boldsymbol{x}_i \rangle}\right) + \lambda \sum_{j=1}^{p} w_j^+ + w_j^- \to \min \\ w_j^+ \geqslant 0,\ w_j^- \geqslant 0,\ j = 1, \ldots, p \end{cases} \tag{2.43}$$

Powyższy problem z ograniczeniami można rozwiązać np. za pomocą algorytmu `L-BFGS-B` [88]. W części eksperymentalnej model ten umieszczono pod nazwą `l1_logreg_lbfgsb`.

– wprowadzając zmienną u_j stanowiącą ograniczenie zmiennej w_j [46]:

$$\begin{cases} \sum_{i=1}^{n} \log\left(1 + e^{-y_i \cdot \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle}\right) + \lambda \sum_{j=1}^{p} u_j \to \min \\ -u_j \leqslant w_j \leqslant u_j,\ j = 1, \ldots, p \end{cases} \tag{2.44}$$

Niegładkie ograniczenie $-u_j \leqslant w_j \leqslant u_j,\ j = 1, \ldots, p$ można zastąpić gładką i wypukłą funkcją logarytmiczną, uzyskując w ten sposób problem bez ograniczeń, który następnie można rozwiązać za pomocą metody Newtona [46]. W części eksperymentalnej pod nazwą `l1_logreg_ipm` kryje się ten model.

Rozbicie pierwotnego problemu na dwa podproblemy

Kolejny sposób oparty na optymalizacji z ograniczeniami polega na sformułowaniu równoważnego problemu, w którym część gładka i niegładka są rozdzielone — mianowicie problemowi pierwotnemu

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \tag{2.45}$$

odpowiada problem z ograniczeniem równościowym

$$\begin{cases} L(\boldsymbol{w}) + \lambda \|\boldsymbol{z}\|_1 \to \min \\ \boldsymbol{w} - \boldsymbol{z} = 0 \end{cases}, \tag{2.46}$$

który można rozwiązać przy użyciu algorytmu ADMM[7] [10, 85]:

$$\boldsymbol{w}^{(k+1)} \leftarrow \arg\min_{\boldsymbol{w}} \left(L(\boldsymbol{w}) + \tfrac{\rho}{2} \left\| \boldsymbol{w} - \boldsymbol{z}^{(k)} + \boldsymbol{u}^{(k)} \right\|_2^2 \right), \tag{2.47}$$

$$\boldsymbol{z}^{(k+1)} \leftarrow S_{\frac{\lambda}{\rho}} \left(\boldsymbol{w}^{(k+1)} + \boldsymbol{u}^{(k)} \right), \tag{2.48}$$

$$\boldsymbol{u}^{(k+1)} \leftarrow \boldsymbol{u}^{(k)} + \boldsymbol{w}^{(k+1)} - \boldsymbol{z}^{(k+1)}. \tag{2.49}$$

[7] ang. alternating direction method of multipliers

gdzie $\rho > 0$. Minimalizacja funkcjonału wyrażonego za pomocą operatora bliskości[8] we wzorze (2.47) odpowiada rozwiązywaniu problemu z regularyzacją ℓ^2. Formuła (2.48) minimalizuje wyrażenie $\lambda\|\boldsymbol{z}\|_1 + \frac{\rho}{2}\|\boldsymbol{w}^{(k+1)} - \boldsymbol{z} + \boldsymbol{u}^{(k)}\|_2^2$. Na końcu korekcie ulega przeskalowana zmienna dualna $\boldsymbol{u}$ i całość powtarza się aż do spełnienia warunku stopu [10].

W celu rozwiązania (2.47) można ponownie posłużyć się rozkładem QR lub formułą Woodbury'ego. Dla rozkładu QR równanie (2.24) przybierze formę

$$\boldsymbol{d} = \boldsymbol{Q}_1^{\mathrm{T}} \left(\boldsymbol{L}_1^{\mathrm{T}} \boldsymbol{D} \boldsymbol{L}_1 + \rho \boldsymbol{I}\right)^{-1} \left(\boldsymbol{L}_1^{\mathrm{T}} \boldsymbol{v} - \rho \boldsymbol{Q}_1 \left(\boldsymbol{w} - \boldsymbol{z}^{(k)} + \boldsymbol{u}^{(k)}\right)\right), \tag{2.50}$$

a po transformacji wektorów $\boldsymbol{w}$, $\boldsymbol{z}^{(k)}$ i $\boldsymbol{u}^{(k)}$ do mniejszej przestrzeni można je zapisać następująco:

$$\hat{\boldsymbol{d}} = \left(\boldsymbol{L}_1^{\mathrm{T}} \boldsymbol{D} \boldsymbol{L}_1 + \rho \boldsymbol{I}\right)^{-1} \left(\boldsymbol{L}_1^{\mathrm{T}} \boldsymbol{v} - \rho \left(\hat{\boldsymbol{w}} - \hat{\boldsymbol{z}}^{(k)} + \hat{\boldsymbol{u}}^{(k)}\right)\right). \tag{2.51}$$

Z kolei dla formuły Woodbury'ego równanie (2.28) przyjmie postać

$$\boldsymbol{d} = \frac{1}{\rho} \boldsymbol{X}^{\mathrm{T}} (\boldsymbol{v} - \boldsymbol{\gamma}) - \left(\boldsymbol{w} - \boldsymbol{z}^{(k)} + \boldsymbol{u}^{(k)}\right), \tag{2.52}$$

gdzie $\boldsymbol{\gamma}$ stanowi rozwiązanie poniższego układu równań:

$$\left(\boldsymbol{D}^{-1} + \frac{1}{\rho} \boldsymbol{X} \boldsymbol{X}^{\mathrm{T}}\right) \boldsymbol{\gamma} = \frac{1}{\rho} \boldsymbol{X} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{v} - \boldsymbol{X} \left(\boldsymbol{w} - \boldsymbol{z}^{(k)} + \boldsymbol{u}^{(k)}\right). \tag{2.53}$$

Z uwagi na konieczność rzutowania do pierwotnej przestrzeni, rozwiązanie oparte na rozkładzie QR będzie mniej satysfakcjonujące pod względem złożoności obliczeniowej niż to oparte na formule Woodbury'ego, dlatego w części eksperymentalnej skupiono się na tym drugim podejściu.

Zastąpienie normy ℓ^1 jej górnym ograniczeniem

Jeszcze inne podejście opiera się na zastąpieniu normy ℓ^1 jej górnym ograniczeniem za pomocą funkcji kwadratowej [20, 47]:

$$\|\boldsymbol{w}\|_1 \leqslant \frac{1}{2} \sum_{j=1}^{p} \left(\frac{w_j^2}{|w'_j|} + |w'_j|\right), \tag{2.54}$$

gdzie powyższa nierówność jest prawdziwa dla dowolnego $\boldsymbol{w}'$, a równość zachodzi wtedy i tylko wtedy, gdy $\boldsymbol{w}' = \boldsymbol{w}$. Zastępując czynnik karzący w równaniu (2.34) otrzymujemy:

$$Q(\boldsymbol{w}) = \sum_{i=1}^{n} \log\left(1 + e^{-y_i \cdot \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle}\right) + \frac{\lambda}{2} \sum_{j=1}^{p} \left(\frac{w_j^2}{|w'_j|} + |w'_j|\right). \tag{2.55}$$

[8] ang. proximity operator

Kierunek poszukiwań $\boldsymbol{d}$ określa poniższe równanie:

$$\boldsymbol{d} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \boldsymbol{E}\right)^{-1}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{E}\boldsymbol{w}\right), \tag{2.56}$$

gdzie $\boldsymbol{E}$ jest macierzą diagonalną i $e_{j,j} = \frac{\lambda}{|w_j|}$ (przyjmując, że $\boldsymbol{w}' = \boldsymbol{w}$). Początkowy wektor $\boldsymbol{w}$ musi mieć niezerowe składowe, ponieważ raz wyzerowana składowa pozostanie zerem, a jeśli w trakcie optymalizacji wartość składowej $|w_j|$ spadnie poniżej precyzji maszynowej, to można ją bezpiecznie wyzerować. Dobry punkt startowy stanowi rozwiązanie problemu (2.13) [36]. Można także zacząć od wektora zerowego i znaleźć minimum w kierunku wskazanym przez antysubgradient, dzięki temu część współczynników może już na początku zostać wyeliminowana, co przyśpieszy początkowe iteracje — ten sposób inicjalizacji wykorzystano w eksperymencie.

Formuła (2.56) nie jest stabilna numerycznie, ponieważ w mianowniku mogą pojawić się wartości bliskie zera, jednakże istnieją warianty formuły stabilnej numerycznie [30, 47]:

$$\boldsymbol{d} = -\boldsymbol{E}^{-\frac{1}{2}}\left(\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X}\boldsymbol{E}^{-\frac{1}{2}} + \boldsymbol{I}\right)^{-1}\boldsymbol{E}^{-\frac{1}{2}} \cdot \partial\boldsymbol{Q} \tag{2.57}$$

$$= \left(\boldsymbol{I} - \boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{D}^{-1}\right)^{-1}\boldsymbol{X}\right) \cdot \left(\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{w}\right), \tag{2.58}$$

gdzie $\partial\boldsymbol{Q}$ oznacza subgradient policzony zgodnie ze wzorem (2.35) z tą różnicą, że dla $w_j = 0$ mamy $\frac{\partial Q(\boldsymbol{w})}{w_j} = 0$. W przypadku (2.57) macierz układu równań ma wymiary $p \times p$, natomiast w przypadku (2.58) macierz układu równań ma wymiary $n \times n$ (skorzystano z zależności

$$\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \boldsymbol{E}\right)^{-1} = \boldsymbol{E}^{-1} - \boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{D}^{-1}\right)^{-1}\boldsymbol{X}\boldsymbol{E}^{-1}$$

i macierz $\boldsymbol{E}^{-1}$ wyłączono z prawej strony, po czym wmnożono ją w antygradient funkcji). W obu wypadkach mamy do czynienia z macierzą dodatnio określoną, więc odwracanie macierzy można zastąpić rozwiązywaniem układu równań metodą gradientu sprzężonego, unikając przy tym jawnego formowania macierzy. Pochodną kierunkową dla przypadku (2.57) opisuje poniższa formuła:

$$\nabla Q\left(\boldsymbol{w}\right)^{\mathrm{T}}\boldsymbol{d} = \langle \partial\boldsymbol{Q}, \boldsymbol{d}\rangle \equiv \langle -\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v}, \boldsymbol{d}\rangle + \lambda\sum_{j=1}^{p}\operatorname{sgn}(w_j)\cdot d_j. \tag{2.59}$$

W części eksperymentalnej model opierający się na rozwiązywaniu (2.57) nazwano `l1_logreg_ordinary`, natomiast ten oparty na (2.58) nazwano `l1_logreg_woodbury`.

Porządkując wyrazy w równaniu (2.58) można je zapisać w następującej formie:

$$\boldsymbol{d} = \boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{v} - \left(\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{D}^{-1}\right)^{-1}\left(\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{X}\boldsymbol{w}\right)\right) - \boldsymbol{w}. \tag{2.60}$$

Do rozwiązania jest układ równań:

$$\left(\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{D}^{-1}\right)\boldsymbol{z} = \boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{X}\boldsymbol{w}. \tag{2.61}$$

Gdy $\boldsymbol{w}$ jest rzadki, należy ten fakt uwzględnić w obliczeniach, żeby nie wykonywać niepotrzebnych mnożeń przez 0 — iloczyn macierzowo-wektorowy $\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v}$ można rozpisać następująco:

$$\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} \equiv \sum_{j=1}^{p} e_{j,j}^{-1} \cdot \langle \boldsymbol{x}_{:,j}, \boldsymbol{v}\rangle \cdot \boldsymbol{x}_{:,j}, \tag{2.62}$$

z kolei działanie $\boldsymbol{a} = \boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v}$ można zapisać następująco:

$$a_j = e_{j,j}^{-1} \cdot \langle \boldsymbol{x}_{:,j}, \boldsymbol{v}\rangle, \tag{2.63}$$

gdzie $\boldsymbol{x}_{:,j}$ jest j-tą kolumną macierzy $\boldsymbol{X}$. Tym samym dla zwiększenia wydajności obliczeń macierz danych powinna być przechowywana w porządku kolumnowym.

Formalnie pochodną kierunkową funkcji $Q(\boldsymbol{w})$ względem kierunku $\boldsymbol{d}$ opisuje następujące równanie:

$$\nabla Q(\boldsymbol{w})^{\mathrm{T}}\boldsymbol{d} = -\boldsymbol{v}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{d}) + (\boldsymbol{v} - \boldsymbol{z})^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{w}) - \lambda\sum_{j}|w_j|, \tag{2.64}$$

jednakże można z powodzeniem korzystać z równania (2.59), zaś samą optymalizację funkcji (2.34) można zakończyć w momencie spełnienia poniższego warunku:

$$\sqrt{-\nabla Q(\boldsymbol{w})^{\mathrm{T}}\boldsymbol{d}} \leqslant \epsilon \tag{2.65}$$

W przypadku uwzględnienia wyrazu wolnego w_0, układ równań przybierze następującą postać:

$$\begin{bmatrix} c & \boldsymbol{b}^{\mathrm{T}} \\ \boldsymbol{b} & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \boldsymbol{E} \end{bmatrix} \cdot \begin{bmatrix} d_0 \\ \boldsymbol{d} \end{bmatrix} = \begin{bmatrix} \boldsymbol{1}^{\mathrm{T}}\boldsymbol{v} \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{E}\boldsymbol{w} \end{bmatrix}, \tag{2.66}$$

gdzie $\boldsymbol{b} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{1}$, $c = \boldsymbol{1}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{1}$. Przekształcając odpowiednio otrzymamy:

$$\begin{cases} d_0 = \frac{\boldsymbol{1}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{d}}{c} \\ \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \boldsymbol{E} - \frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}\right)\boldsymbol{d} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{E}\boldsymbol{w} - \frac{\boldsymbol{1}^{\mathrm{T}}\boldsymbol{v}}{c}\boldsymbol{b} \end{cases}. \tag{2.67}$$

W tym przypadku macierz symetryczna $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X}+\boldsymbol{E}-\frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$ może być nieokreślona, dlatego do rozwiązania powyższego układu równań należy posłużyć się np. metodą `MINRES` bądź `SYMMLQ` [60].

Formułę stabilną numerycznie można uzyskać w podobny sposób, jak miało to miejsce w przypadku (2.57):

$$\boldsymbol{d}=-\boldsymbol{E}^{-\frac{1}{2}}\left(\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{D}-\tfrac{1}{c}\boldsymbol{D}\mathbf{1}\mathbf{1}^{\mathrm{T}}\boldsymbol{D}\right)\boldsymbol{X}\boldsymbol{E}^{-\frac{1}{2}}+\boldsymbol{I}\right)^{-1}\boldsymbol{E}^{-\frac{1}{2}}\left(\partial\boldsymbol{Q}+\tfrac{\mathbf{1}^{\mathrm{T}}\boldsymbol{v}}{c}\boldsymbol{b}\right), \quad (2.68)$$

a do rozwiązania jest układ równań o wymiarach $p\times p$. Macierz $\boldsymbol{D}-\frac{1}{c}\boldsymbol{D}\mathbf{1}\mathbf{1}^{\mathrm{T}}\boldsymbol{D}$ jest osobliwa, więc w celu użycia formuły Woodbury'ego wprowadźmy macierz $\hat{\boldsymbol{E}}=\boldsymbol{E}-\frac{1}{c}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}$. Korzystając z formuły Shermana-Morrisona obliczymy $\hat{\boldsymbol{E}}^{-1}$ [70]:

$$\hat{\boldsymbol{E}}^{-1}=\boldsymbol{E}^{-1}+\frac{\boldsymbol{E}^{-1}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{E}^{-1}}{c-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{E}^{-1}\boldsymbol{b}}. \quad (2.69)$$

Po uporządkowaniu do rozwiązania jest układ równań podobny do (2.61):

$$\left(\boldsymbol{X}\hat{\boldsymbol{E}}^{-1}\boldsymbol{X}^{\mathrm{T}}+\boldsymbol{D}^{-1}\right)\boldsymbol{z}=\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v}-\boldsymbol{X}\boldsymbol{w}+\gamma\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}. \quad (2.70)$$

gdzie $\gamma=\frac{\langle\boldsymbol{v},\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}\rangle-\langle\boldsymbol{D}\mathbf{1},\boldsymbol{X}\boldsymbol{w}\rangle-\mathbf{1}^{\mathrm{T}}\boldsymbol{v}}{c-\langle\boldsymbol{D}\mathbf{1},\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}\rangle}$. Iloczyn macierzowo-wektorowy $\boldsymbol{X}\hat{\boldsymbol{E}}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{s}$ można rozpisać analogicznie:

$$\boldsymbol{X}\hat{\boldsymbol{E}}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{s}\equiv\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{s}+\frac{\langle\boldsymbol{s},\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}\rangle}{c-\langle\boldsymbol{D}\mathbf{1},\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}\rangle}\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}, \quad (2.71)$$

zaś rozwiązanie po uproszczeniu opisuje równanie:

$$\boldsymbol{d}=\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{v}-\boldsymbol{z}+\left(\gamma-\frac{\langle\boldsymbol{z},\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}\rangle}{c-\langle\boldsymbol{D}\mathbf{1},\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}\rangle}\right)\boldsymbol{D}\mathbf{1}\right)-\boldsymbol{w}. \quad (2.72)$$

Pochodną kierunkową można policzyć zgodnie z równaniem (2.59), pamiętając o dodaniu składnika $g_0\cdot d_0$. Włączenie wyrazu wolnego wiąże się z dodatkowym kosztem $O\left(\#\mathrm{nnz}\cdot p\right)$ obliczenia iloczynu $\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{b}$ i kilkoma iloczynami skalarnymi, gdzie $\#\mathrm{nnz}$ oznacza liczbę niezerowych parametrów

Zysk z przechowywania macierzy $\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}$ (analogicznie jak miało to miejsce w przypadku macierzy $\frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$) wydaje się wątpliwy, ponieważ po każdej zmianie parametrów trzeba wyliczyć korektę (zmienia się przecież macierz $\boldsymbol{E}^{-1}$), co wiąże się z kosztem rzędu $O\left(\#\mathrm{nnz}\cdot n\right)$. Dodatkowo sama macierz może mieć niepełny rząd (gdy liczba parametrów aktywnych będzie mniejsza niż n), zatem koszt iloczynu macierzowo-wektorowego zapewne będzie większy niż użycie formuły (2.62).

2.6 Regularyzacja „elastic net”

Liczba niezerowych parametrów w modelu z regularyzacją ℓ^1 zawsze nie przekroczy $\min(n, p)$, a gdy w zbiorze znajdują się grupy silnie skorelowanych atrybutów, to spośród nich zostanie zazwyczaj arbitralnie wybranych po jednym (z kolei regularyzacja ℓ^2 ściąga równomiernie całą grupę), co w pewnym stopniu ogranicza tę technikę jako narzędzie automatycznej selekcji istotnych cech. Powyższe ograniczenia likwiduje połączenie regularyzacji ℓ^2 i ℓ^1, co szerzej znane jest jako regularyzacja elastic net [89]. W przypadku regresji liniowej optymalizowana funkcja kosztu wygląda następująco:

$$Q(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle)^2 + \lambda\sum_{j=1}^{p}\alpha\cdot|w_j| + \frac{1-\alpha}{2}w_j^2, \tag{2.73}$$

natomiast w przypadku regresji logistycznej funkcja kosztu ma postać:

$$Q(\boldsymbol{w}) = \sum_{i=1}^{n}\log\left(1 + e^{-y_i\cdot\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle}\right) + \lambda\sum_{j=1}^{p}\alpha\cdot|w_j| + \frac{1-\alpha}{2}w_j^2, \tag{2.74}$$

gdzie $\alpha \in (0, 1)$ steruje wpływem poszczególnych kar.

2.6.1 Spadek względem współrzędnych

W przypadku regresji liniowej zmianie ulega jedynie równanie (2.39) [21]:

$$w_j^* \leftarrow \frac{\mathcal{S}_{\lambda\alpha}\left(\sum_i x_{i,j}\cdot r_i + w_j\cdot\sum_i x_{i,j}^2\right)}{\lambda\cdot(1-\alpha) + \sum_i x_{i,j}^2}, \tag{2.75}$$

a w przypadku regresji logistycznej równanie (2.41) przybiera postać:

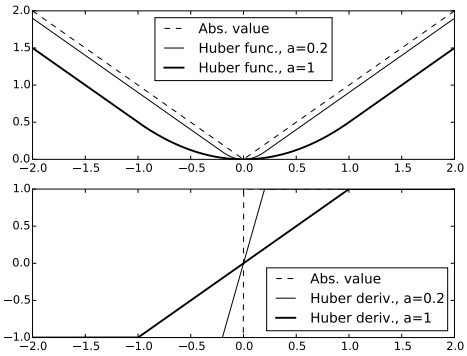
$$w_j^* \leftarrow \frac{\mathcal{S}_{\lambda\alpha}\left(\sum_i x_{i,j}\cdot r_i + w_j^*\cdot\sum_i d_{i,i}\cdot x_{i,j}^2\right)}{\lambda\cdot(1-\alpha) + \sum_i d_{i,i}\cdot x_{i,j}^2}. \tag{2.76}$$

2.6.2 Przekształcenie w problem gładkiej optymalizacji

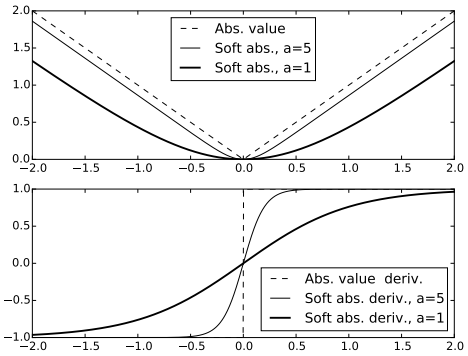
Analogicznie do układu (2.43) można zapisać układ równań w tym przypadku:

$$\begin{cases} \sum_{i=1}^{n}\log\left(1 + e^{-y_i\cdot\langle \boldsymbol{w}^+ - \boldsymbol{w}^-, \boldsymbol{x}_i\rangle}\right) + \lambda\frac{1-\alpha}{2}\|\boldsymbol{w}^+ - \boldsymbol{w}^-\|_2^2 + \lambda\alpha\sum_{j=1}^{p} w_j^+ + w_j^- \to \min \\ w_j^+ \geqslant 0,\ w_j^- \geqslant 0,\ j = 1, \ldots, p \end{cases} \tag{2.77}$$

i ponownie rozwiązać za pomocą algorytmu `L-BFGS-B`.



Rysunek 2.2: Funkcja Hubera [35] — gładka aproksymacja wartości bezwzględnej oraz jej pochodnq w porównaniu do wartości bezwzględnej wraz z jej pochodną.



Rysunek 2.3: Funkcja soft_abs — gładka aproksymacja wartości bezwzględnej oraz jej pochodna w porównaniu do funkcji wartości bezwzględnej wraz z jej pochodną.

Zastąpienie normy ℓ^1 jej górnym ograniczeniem

Ideę przedstawioną w sekcji 2.5.3 można z powodzeniem zastosować przy tej regularyzacji, zmieniając jedynie elementy macierzy $\boldsymbol{E}^{-1}$:

$$e_{j,j}^{-1} = \frac{|w_j|}{\lambda\alpha + \lambda \cdot (1-\alpha)\,|w_j|} \tag{2.78}$$

i równanie opisujące pochodną kierunkową:

$$\nabla Q\,(\boldsymbol{w})^{\mathrm{T}}\,\boldsymbol{d} = \langle -\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v}, \boldsymbol{d}\rangle + \lambda\,(1-\alpha)\,\langle \boldsymbol{w}, \boldsymbol{d}\rangle + \lambda\alpha \sum_{j=1}^{p} \operatorname{sgn}(w_j)\cdot d_j. \tag{2.79}$$

2.7 Aproksymacja normy ℓ^1

Kolejny sposób radzenia sobie z nieróżniczkowalnością normy ℓ^1 polega na zastąpieniu jej gładką aproksymacją. Powoduje to utratę rzadkości rozwiązania, jednakże efekt grupowania (ściągania w otoczenie zera) pozostaje zachowany. Jeżeli chcemy uzyskać rozwiązanie w pełni rzadkie, musimy po strojeniu wykonać progowanie współczynników poniżej arbitralnie przyjętej wartości.

W literaturze można natknąć się na przynajmniej trzy sposoby aproksymacji:

– w pierwszym funkcję w otoczeniu $[-\alpha, \alpha]$ przybliża się za pomocą funkcji kwadratowej (metoda Hubera [35]):

$$f_\alpha(x) = \begin{cases} \frac{1}{2\alpha}x^2, & |x| \leqslant \alpha \\ |x| - \frac{1}{2}\alpha, & \text{w.p.r.} \end{cases}. \tag{2.80}$$

Im α bliższe zeru, tym aproksymacja staje się dokładniejsza (rys. 2.2). Funkcja nie posiada ciągłej drugiej pochodnej.

– w drugim korzysta się z obserwacji, że tangens hiperboliczny przypomina funkcję signum, więc jej całka przybliża wartość bezwzględną [50] (funkcja oznaczna jako soft_abs, por. rys. 2.3):

$$f_\alpha(x) = \frac{1}{\alpha}\log\left(\cosh\left(\alpha x\right)\right) \equiv |x| + \frac{1}{\alpha}\log\left(\frac{1+e^{-2\alpha|x|}}{2}\right). \tag{2.81}$$

Zwiększenie wartości α poprawia dokładność przybliżenia. Funkcja ta jest ściśle wypukła oraz posiada ciągłe i gładkie pochodne, co pozwala na użycie gradientowych metod optymalizacji:

$$\frac{\partial Q}{\partial w_j} \equiv g_j = \frac{\partial L}{\partial w_j} + \lambda \tanh\left(\alpha w_j\right), \tag{2.82}$$

$$\frac{\partial^2 Q}{\partial \boldsymbol{w}\partial \boldsymbol{w}^{\mathrm{T}}} = \frac{\partial^2 L}{\partial \boldsymbol{w}\partial \boldsymbol{w}^{\mathrm{T}}} + \boldsymbol{E}, \tag{2.83}$$

gdzie $\boldsymbol{E}$ jest macierzą diagonalną i $e_{j,j} = \lambda\alpha\left(1-\tanh^2\left(\alpha w_j\right)\right)$. W przypadku $n < p$ można skorzystać z formuły Woodbury'ego w celu redukcji wymiarowości problemu — dla regresji logistycznej wzór na kierunek wygląda następująco:

$$\boldsymbol{d} = -\boldsymbol{E}^{-1}\boldsymbol{g} + \boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}}\left(\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{X}^{\mathrm{T}} + \boldsymbol{D}^{-1}\right)^{-1}\boldsymbol{X}\boldsymbol{E}^{-1}\boldsymbol{g}. \tag{2.84}$$

W analogiczny sposób można dołączyć wyraz wolny w_0.

W przypadku tego sposobu trzeba również zachować ostrożność, ponieważ na przekątnej macierzy $\boldsymbol{E}$ mogą pojawić się wartości bliskie zera.

– w trzecim przypadku wartość bezwzględną przybliża się za pomocą pierwiastka z funkcji kwadratowej:

$$f_\alpha(x) = \sqrt{x^2+\alpha}, \tag{2.85}$$

gdzie α jest pewną małą dodatnią liczbą.

2.8 Regularyzacja z użyciem pseudonorm ℓ^q, $0 \leqslant q < 1$,

Norma ℓ^1 znajduje się na granicy wypukłości — obniżając dalej wykładnik uzyskamy funkcję niewypukłą. Funkcja kosztu ma postać:

$$Q(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda\sum_{j=1}^{p}|w_j|^q. \tag{2.86}$$

Z uwagi na niewypukłość funkcji celu znalezione minimum będzie w ogólności lokalnym, a nie globalnym minimum. W dodatku pochodna kierunkowa nie istnieje, dlatego jako kryterium stopu można posłużyć się różnicą względną wartości funkcji między kolejnymi iteracjami.

2.8.1 Spadek względem współrzędnych

Rozpatrzmy najpierw ogólny przypadek jednowymiarowy:

$$f(x) = \frac{\mu}{2}(y-x)^2 + \lambda \cdot |x|^q,\ \mu > 0,\ q \in (0,1)\,. \tag{2.87}$$

Dla pewnej wartości $\lambda = \lambda_{\text{critical}}$ wartość funkcji w zerze równa się wartości funkcji w punkcie x_{critical} — innymi słowy istnieją dwa minima globalne. Aby znaleźć wartości $\lambda_{\text{critical}}$ i x_{critical} należy rozwiązać poniższy układ równań:

$$\begin{cases} f(0) = f(x_{\text{critical}}) \\ f'(x_{\text{critical}}) = 0 \end{cases}. \tag{2.88}$$

Rozwiązując otrzymamy $x_{\text{critical}} = \frac{2-2q}{2-q} \cdot y$ i $\lambda_{\text{critical}} = \mu \cdot (2-2q)^{1-q} \cdot \left(\frac{|y|}{2-q}\right)^{2-q}$. Gdy $\lambda > \lambda_{\text{critical}}$, to minimum globalne funkcji (2.87) znajduje się w zerze, a w przeciwnym razie należy skorzystać z numerycznej metody poszukiwania minimum, ponieważ nie istnieje analityczne rozwiązanie. W tym celu można użyć jednowymiarowej metody Newtona-Raphsona dzięki różniczkowalności funkcji dla $x \neq 0$:

$$\begin{cases} f(x)' = \mu \cdot (x-y) + \lambda \cdot q \cdot \operatorname{sgn}(x) \cdot |x|^{q-1}, \\ f(x)'' = \mu + \lambda \cdot q \cdot (q-1) \cdot |x|^{q-2}, \end{cases} \tag{2.89}$$

przyjmując $x_0 = y$. W przypadku $\lambda = \lambda_{\text{critical}}$ należy arbitralnie wskazać któreś z minimów.

Podstawiając (2.1) do (2.86) i różniczkując po zmiennej w_j otrzymamy:

$$\frac{\partial Q}{\partial w_j} = \sum_i x_{i,j}^2 \cdot \left(w_j - \frac{\sum_i x_{i,j} \cdot \left(y_i - \sum_{k \neq j} x_{i,k} w_k \right)}{\sum_i x_{i,j}^2} \right) + \lambda \cdot q \cdot \operatorname{sgn}\left(w_j\right) \cdot |w_j|^{q-1}. \tag{2.90}$$

Powyższe wyrażenie przypomina (2.89): $\mu = \sum_i x_{i,j}^2$, $x = w_j$, $y = \frac{\sum_i x_{i,j} \cdot \left(y_i - \sum_{k \neq j} x_{i,k} w_k\right)}{\sum_i x_{i,j}^2}$. Ponownie można skorzystać ze sztuczki (2.38) obniżającej koszt obliczenia wartości y. Z kolei dla regresji logistycznej w algorytmie 4 wystarczy jedynie podmienić $x = w_{\text{opt}}$, $y = \frac{\sum_i x_{i,j} \cdot r_i}{\sum_i d_{i,i} \cdot x_{i,j}^2} + w_j^*$ i wprowadzić $\mu = \sum_i d_{i,i} \cdot x_{i,j}^2$, po czym policzyć nową wartość w_{opt} dla j-tego parametru modelu.

Ponieważ mamy do czynienia z problemem niewypukłym, algorytm może nie zatrzymać się w skończonej liczbie kroków, a minimalizacja kierunkowa wymaga większej uwagi, aby znaleźć odpowiedni krok.

Przypadek szczególny $q = 0$

Przypadek ten sprowadza się do znalezienia najlepszego podzbioru atrybutów[9] [29]. Bywa on także nazywany twardym progowaniem przez analogię do miękkiego progowania w przypadku $q = 1$ [20]. Funkcja (2.87) w tym przypadku traci ciągłość w punkcie $x = 0$ i wygląda następująco:

$$f(x) = \begin{cases} \frac{\mu}{2}(y - x)^2 + \lambda, & x \neq 0, \\ \frac{\mu}{2} y^2, & x = 0. \end{cases} \tag{2.91}$$

Przyjmuje ona dwa minima: jedno w punkcie $x = 0$, a drugie w punkcie $x = y$. Wartość $\lambda_{\text{critical}}$ wynosi $\frac{\mu}{2} y^2$.

Przypadek szczególny $q = \frac{1}{2}$

W pracy [44] przeanalizowano dokładnie przypadek $q = \frac{1}{2}$, w którym minimalizację (2.87) można sprowadzić do minimalizacji wielomianu czwartego stopnia. Niech $x \geqslant 0$ i $t = x^{\frac{1}{2}}$ (przypadek $x < 0$ jest symetryczny):

$$f(t) = \frac{\mu}{2}\left(y - t^2\right)^2 + \lambda t. \tag{2.92}$$

Różniczkując względem t, a następnie dzieląc przez 2μ i przyrównując wynik do zera uzyskamy:

$$t^3 - yt + \frac{\lambda}{2\mu} = 0. \tag{2.93}$$

Powyższe równanie posiada trzy pierwiastki [58]:

$$\begin{cases} \alpha &= 2 \cdot \sqrt{\frac{y}{3}} \cdot \cos\theta, \\ \beta &= 2 \cdot \sqrt{\frac{y}{3}} \cdot \cos\left(\theta + \frac{2\pi}{3}\right), \\ \gamma &= 2 \cdot \sqrt{\frac{y}{3}} \cdot \cos\left(\theta + \frac{4\pi}{3}\right), \end{cases} \tag{2.94}$$

gdzie

$$\theta = \frac{1}{3} \cdot \arccos\left(-\frac{\lambda}{4\mu\left(\frac{y}{3}\right)^{\frac{3}{2}}}\right). \tag{2.95}$$

Interpretacja rozwiązań: α to lokalne minimum (2.92), z kolei γ to lokalne maksimum, natomiast $\beta < 0$. Pamiętając o podstawieniu $t = x^{\frac{1}{2}}$, wartość α należy podnieść do kwadratu, by otrzymać x.

[9] ang. best subset selection

2.8.2 Zastąpienie pseudonormy ℓ^q jej górnym ograniczeniem

Podobnie jak miało to miejsce w przypadku normy ℓ^1, możemy zastąpić pseudonormę ℓ^q funkcją kwadratową [40]:

$$Q(\boldsymbol{w}) = L(\boldsymbol{w}) + \frac{\lambda}{2}\sum_{j=1}^{p}\left(\frac{q \cdot w_j^2}{|w'_j|^{2-q}} + (2-q)\,|w'_j|^q\right) \tag{2.96}$$

i zapisać rozwiązanie dla regresji logistycznej w analogiczny sposób, jak w (2.56), zmieniając jedynie elementy macierzy diagonalnej $\boldsymbol{E}$, tj. $e_{j,j} = \frac{\lambda q}{|w_j|^{2-q}}$. Dalej należy postąpić analogicznie, jak przedstawiono to dla normy ℓ^1.

2.9 Uogólniona regularyzacja

Karze nie muszą podlegać wyłącznie pojedyncze współczynniki modelu — jeżeli np. podejrzewamy, że szukany model zawiera wiele płaskich obszarów, to możemy nałożyć karę ℓ^1 na różnice sąsiednich współczynników, co zminimalizuje ich wahanie się. Niech dana będzie macierz różnicy $\boldsymbol{F}_{(p-1)\times p}$ zbudowana następująco:

$$f_{i,j} = \begin{cases} -1, & j = i, \\ 1, & j = i+1, \\ 0, & wpr. \end{cases} \tag{2.97}$$

Powyższą macierz należy umieścić w funkcji kary (np. ℓ^1 lub ℓ^2) danej w tabeli 2.1, zaś samą funkcję w (2.12). Przyjmując następującą postać funkcji kosztu:

$$Q\,(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \|\boldsymbol{F}\boldsymbol{w}\|_1, \tag{2.98}$$

otrzymamy zagadnienie odszumiania[10], gdzie $\boldsymbol{y}$ to sygnał źródłowy, który chcemy odzyskać [65].

Normę ℓ^1 można z powodzeniem zastąpić normą ℓ^2, a różnice pierwszego rzędu różnicami wyższych rzędów [19], np. macierz $\boldsymbol{F}_{(p-2)\times p}$ obliczająca różnice drugiego rzędu ma postać:

$$f_{i,j} = \begin{cases} 1, & j = i \ \vee \ j = i+2, \\ -2, & j = i+1, \\ 0, & wpr. \end{cases} \tag{2.99}$$

W przypadku sygnałów dwuwymiarowych (np. obrazów) karać należy współczynniki w poszczególnych wierszach i kolumnach. Postać macierzy różnic $\boldsymbol{F}$ zależy od porządku,

[10] ang. total variation denoising

w jakim trzymano pierwotny sygnał, zanim przekształcono go w wektor, łącząc kolejno wiersze lub kolumny z sobą. Niech $(\hat{n}, \hat{p})$ oznacza pierwotną liczbę wierszy i kolumn sygnału (po wektoryzacji mamy $p = \hat{n}\hat{p}$):

– W porządku wierszowym $\boldsymbol{H} = \boldsymbol{I}_{\hat{n}} \otimes \hat{\boldsymbol{F}}_{(\hat{p}-1)\times\hat{p}}$, gdzie $\boldsymbol{H}$ to macierz obliczająca różnice elementów w wierszach, zaś $\hat{\boldsymbol{F}}_{(\hat{p}-1)\times\hat{p}}$ to macierz obliczająca różnice sygnału jednowymiarowego. Z kolei macierz $\boldsymbol{V}$ obliczającą różnice pierwszego rzędu w kolumnach buduje się następująco:

$$v_{i,j} = \begin{cases} -1, & j = i, \\ 1, & j = i + \hat{p}, \\ 0, & wpr. \end{cases} \tag{2.100}$$

– W porządku kolumnowym $\boldsymbol{V} = \boldsymbol{I}_{\hat{p}} \otimes \hat{\boldsymbol{F}}_{(\hat{n}-1)\times\hat{n}}$, natomiast macierz $\boldsymbol{H}$ definiuje się następująco:

$$h_{i,j} = \begin{cases} -1, & j = i, \\ 1, & j = i + \hat{n}, \\ 0, & wpr. \end{cases} \tag{2.101}$$

Ostatecznie macierz różnic $\boldsymbol{F}$ w przypadku dwuwymiarowym stanowi złożenie obu macierzy różnic, co można symbolicznie zapisać:

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{H} \\ \boldsymbol{V} \end{bmatrix}. \tag{2.102}$$

Do regularyzacji różnic można dodać jednoczesne karanie pojedynczych współczynników modelu[11] [76], co w przypadku normy ℓ^1 pozwoli wychwycić pojedyncze „piki” tudzież równiny.

[11] ang. fused lasso

3. Realizacja modeli nieliniowych

3.1 Uczenie sieci neuronowych

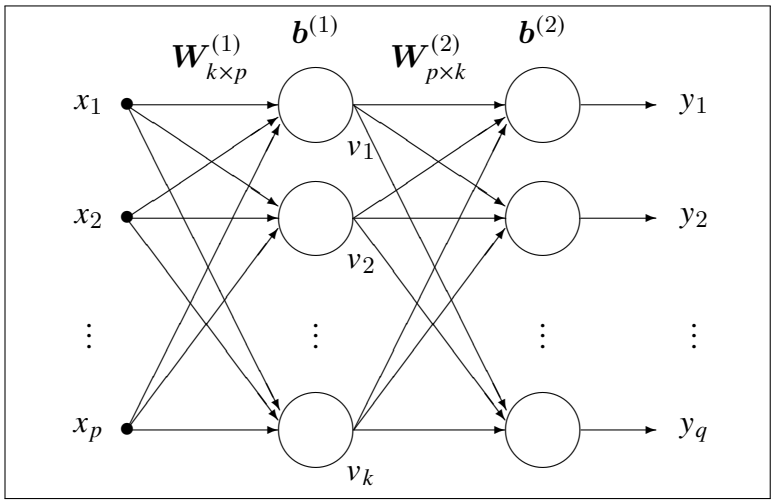
Rozpatrzmy prosty model sieci neuronowej z pojedynczą warstwą ukrytą złożoną z k neuronów ($k < p$), przedstawiony na rysunku 3.1. Wyjście z sieci opisuje wyrażenie:

$$y(\boldsymbol{x}) = \sigma_2\left(\boldsymbol{W}^2 \cdot \sigma_1\left(\boldsymbol{W}^1 \cdot \boldsymbol{x} + \boldsymbol{b}^1\right) + \boldsymbol{b}^2\right), \tag{3.1}$$

natomiast wyjście z warstwy pośredniej dane jest wzorem:

$$v(\boldsymbol{x}) = \sigma_1\left(\boldsymbol{W}^1 \cdot \boldsymbol{x} + \boldsymbol{b}^1\right), \tag{3.2}$$

gdzie $\sigma_1(\cdot)$ to funkcja aktywacji w warstwie ukrytej, a $\sigma_2(\cdot)$ to funkcja aktywacji w warstwie wyjściowej oraz $\mathbf{W}^1_{k\times p}$, $\mathbf{b}^1_{k\times 1}$, $\mathbf{W}^2_{q\times k}$ i $\mathbf{b}^2_{q\times 1}$ są odpowiednio wagami i wartościami progowymi dla pierwszej i drugiej warstwy. Rozważane w pracy funkcje aktywacji σ to identyczność oraz tangens hiperboliczny.



Rysunek 3.1: Sieć neuronowa z jedną warstwą ukrytą.

Sieć o takiej architekturze może spełniać różnorodne funkcje, począwszy od uniwersalnego klasyfikatora i regresora. W pracy przedstawione zostaną dwa zagadnienia wykorzystujące prezentowane techniki regularyzacji:

uczenie zrandomizowane — uczenie sieci wykorzystujące jedynie techniki strojenia klasyfikatorów/regresorów liniowych, w tym techniki opisane w poprzednich punktach.

rzadki autoenkoder — autoenkoder bazujący na dwuwarstwowej sieci neuronowej z wyzerowaną pewną liczbą wag w warstwie pośredniej i warstwie wyjściowej; taki autoenkoder jest pewną formą rzadkiego wariantu metody PCA lub pewną formą faktoryzacji macierzy danych.

3.2 Rzadki autoenkoder

Sformułowanie zagadnienia

Niech $\boldsymbol{X}$ oznacza macierz danych. Rozważmy procedurę PCA[1]:

$$\begin{aligned} \boldsymbol{C} &= \operatorname{cov}(\boldsymbol{X}) \\ [\boldsymbol{P}, \boldsymbol{\lambda}] &= \operatorname{eig}(C), \end{aligned} \tag{3.3}$$

gdzie $\boldsymbol{P}$ jest macierzą złożoną kolumnowo z wektorów własnych, a $\boldsymbol{\lambda}$ jest wektorem odpowiadających im wartości własnych. Wtedy mamy:

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{P}\boldsymbol{P}^T\boldsymbol{X} \\ \boldsymbol{X} &\approx \boldsymbol{P}_{:,1:k}\boldsymbol{P}^T_{:,1:k}\boldsymbol{X} \end{aligned} \tag{3.4}$$

a dla $k < p$ model ten może być traktowany jako najprostsza forma autoenkodera.

Metoda PCA nie jest jednak wolna od pewnych wad, np. minimalizacja wariancji składowych jest zależna od wartości odstających [15] oraz w standardowym podejściu większość zmiennych ma wpływ na każdą główną składową, a ich bezpośrednia interpretacja jest często kłopotliwa.

Przeciwieństwem tego podejścia są procedury rzadkie, które produkują składowe zależne tylko od małego podzbioru zmiennych wejściowych. Rzadkość komponentów składowych może być atrakcyjna z wielu punktów widzenia. Przykładowo stosując taką reprezentację składników można zaoszczędzić miejsce w pamięci i skrócić czas obliczeń. W przypadku specjalnych typów sygnałów o określonej strukturze zależności (jak obrazy, dźwięki lub dane chemometryczne), takie podejście zapewnia rozwiązania, które często

[1] ang. principal component analysis

można zinterpretować w sposób naturalny. Przykładowo jako rozwiązanie można uzyskać oddzielne obszary w danych, np. usta i oczy w przypadku obrazów twarzy.

Rzadkość jest zazwyczaj rozumiana jako liczba współczynników zerowych w rozwiązaniu lub jako względna liczba współczynników zerowych w odniesieniu do rozmiaru danych. Rzadkość rozwiązania może być zapewniona przez dodanie do funkcji błędu składnika regularyzacyjnego z karą ℓ^1 na współczynniki [29, 83]. Istnieje wiele różnych technik zapewniających rzadkość komponentów [15, 56]. Jednym z takich podejść jest uczenie słownikowe, które zaimplementowano w klasie `SparsePCA` w pakiecie `scikit-learn` [39, 53].

Sieć o architekturze przedstawionej na rysunku 3.1, gdy $p = q$ może pełnić funkcję autoenkodera, tj. struktury, której celem jest odtwarzanie danych wejściowych, uczonej zestawem danych $\{(\boldsymbol{x}_i, \boldsymbol{x}_i)\}_{i=1}^n$, gdzie n to liczba obserwacji, a p to liczba atrybutów. Sieć o takiej strukturze może być rozpatrywana jako wariant rzadkiej metody PCA [8, 56]. Przyjmijmy funkcje aktywacji identycznościowe, tzn. $\sigma_1(z) = z$, $\sigma_2(z) = z$. Rozważmy funkcję kosztu zawierającą składniki z karą ℓ^1 na współczynniki warstwy pośredniej i wyjściowej:

$$L\left(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}^1, \boldsymbol{W}^2\right) = \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{y}_i - f\left(\boldsymbol{x}_i\right)\|_2^2 + \lambda \cdot \left(\|\boldsymbol{W}^1\|_1 + \|\boldsymbol{W}^2\|_1\right). \tag{3.5}$$

Procedura uczenia

Wyniki przedstawione w tym rozdziale zostały opublikowane w pracy [62]. Rozważamy ogólne zadanie uczenia nadzorowanego: niech $\boldsymbol{X}_{p\times n} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ będzie częścią wejściową zbioru danych (obserwacje znajdują się w kolumnach), a $\boldsymbol{Y}_{p\times n} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$ wyjściową. Rozważmy następującą funkcję (część dodatnia, `relu`, por. sekcja 2.5.3):

$$(a_i)^+ = \max(a_i, 0) = \begin{cases} a_i, & \text{dla } a_i > 0 \\ 0, & \text{wpr.j} \end{cases},$$

podobnie możemy zdefiniować część ujemną: $(a_i)^- = (-a_i)^+$ oraz podobnie dla wektora $\mathbf{a} = [a_1, \ldots, a_p]$, używamy notacji: $\mathbf{a}^+ = [(a_1)^+, \ldots, (a_p)^+]$ i $\mathbf{a}^- = [(a_1)^-, \ldots, (a_p)^-]$. Korzystając z definicji otrzymamy (por. sekcja 2.5.3) $\mathbf{a} = \mathbf{a}^+ - \mathbf{a}^-$, czyli rozbito zmienną na różnicę części dodatniej i ujemnej, przy czym zarówno część ujemna, jak i dodatnia są nieujemne.

Rozważamy dwuwarstwową sieć neuronową z k neuronami w warstwie ukrytej, jak pokazano na rys. 3.1, wyjściem z sieci i wyjściem z warstwy pośredniej danymi wzorami (3.1)

oraz (3.2). Celem uczenia jest optymalizacja (minimalizacja) następującej funkcji:

$$\underset{\boldsymbol{W}^1,\boldsymbol{b}^1,\boldsymbol{W}^2,\boldsymbol{b}^2}{\arg\min} \frac{1}{2n}\sum_{i=1}^{n}\left\|\sigma\left(\boldsymbol{W}^2\sigma\left(\boldsymbol{W}^1\boldsymbol{x}_i+\boldsymbol{b}^1\right)+\mathbf{b}^2\right)-\boldsymbol{y}_i\right\|_2^2+\alpha\cdot\left(\|\boldsymbol{W}^1\|_1+\|\boldsymbol{W}^2\|_1\right), \quad (3.6)$$

gdzie α jest współczynnikiem regularyzacji. Ze względu na nieróżniczkowalność normy $\|\cdot\|_1$, zamieniamy problem optymalizacyjny (3.6) na równoważny problem optymalizacji funkcji gładkiej z ograniczeniami, używając tożsamości $|x| \equiv x^+ - x^-$ dla $x^+, x^- \geqslant 0$:

$$\begin{aligned}\underset{\boldsymbol{W}^{1+},\boldsymbol{W}^{1-},\boldsymbol{b}^1,\boldsymbol{W}^{2+},\boldsymbol{W}^{2-},\boldsymbol{b}^2}{\arg\min} \quad & \frac{1}{2n}\sum_{i=1}^{n}\left\|\sigma\left(\boldsymbol{W}_*^2\cdot\sigma\left(\boldsymbol{W}_*^1\boldsymbol{x}_i+\boldsymbol{b}^1\right)+\boldsymbol{b}^2\right)-\boldsymbol{y}_i\right\|_2^2+\\ & +\alpha\cdot\left(\mathbf{1}^T\boldsymbol{W}^{1+}\mathbf{1}+\mathbf{1}^T\boldsymbol{W}^{1-}\mathbf{1}+\mathbf{1}^T\boldsymbol{W}^{2+}\mathbf{1}+\mathbf{1}^T\boldsymbol{W}^{2-}\mathbf{1}\right),\\ & \boldsymbol{W}^{1+}\geqslant 0,\boldsymbol{W}^{1-}\geqslant 0,\boldsymbol{W}^{2+}\geqslant 0,\boldsymbol{W}^{2-}\geqslant 0,\end{aligned} \quad (3.7)$$

gdzie $\boldsymbol{W}_*^1 = \boldsymbol{W}^{1+} - \boldsymbol{W}^{1-}$, $\boldsymbol{W}_*^2 = \boldsymbol{W}^{2+} - \boldsymbol{W}^{2-}$, a operator wektorowy „$\geqslant$” jest stosowany do każdego elementu wektora/macierzy.

Do rozwiązania zadania (3.7) można użyć algorytmu `L-BFGS-B` [88]. Podobne podejście rozważano np. w [52, 69] w przypadku uczenia regresji logistycznej z karą ℓ^1 (por. sekcja 2.5.3).

Wykorzystując procedurę `L-BFGS-B` sieć można uczyć globalnie lub użyć następującej metody krokowej: zaczynając od jednego neuronu w warstwie ukrytej ($k = 1$) sieć neuronowa jest trenowana, a następnie poszerzeniu ulega bieżące rozwiązanie poprzez dodanie kolejnej jednostki — ta operacja wstawia do macierzy $\boldsymbol{W}^{1+}$ nowy wiersz i nową kolumnę do macierzy $\boldsymbol{W}^{2+}$ oraz nowy element do wektora $\boldsymbol{b}^1$, podobnie należy zrobić z macierzami $\boldsymbol{W}^{1-}$ i $\boldsymbol{W}^{2-}$. Wartości nowych współczynników można zainicjalizować losowo. Po dodaniu neuronu procedura uczenia jest ponawiana.

Początkowe wagi sieci neuronowej mogą być ustawione losowo, jednak w pracy inicjalizowano wagi $\boldsymbol{W}^1$ macierzą transformacji wyznaczoną za pomocą procedury PCA, a wagi $\boldsymbol{W}^2$ macierzą transponowaną, z kolei wektory $\boldsymbol{b}^1$ i $\mathbf{b}^2$ wykluczono z modelu.

Przedstawione rozwiązanie jest rozwiązaniem ogólnym, które można wykorzystać do uczenia dowolnych sieci neuronowych z rzadkimi ograniczeniami typy ℓ^1. Chcąc użyć taką sieć jako rzadki autokoder równoważny procedurze PCA, ustawiamy σ jako funkcję identycznościową (co oznacza liniową funkcję aktywacji) oraz dane wejściowe ustawiamy jako cel predykcji. k-ty wiersz $\mathbf{W}^1$ można interpretować jako k-tą główną składową, a $\mathbf{W}^2$ jest macierzą odwrotnej transformacji do wymiaru p.

Proponowane rozwiązanie może być porównane z procedurą `SparsePCA` z pakietu `scikit-learn` [39, 53], wykorzystującą uczenie słownikowe. Podejście to rozwiązuje następujący problem optymalizacji:

$$(\boldsymbol{U}^*, \boldsymbol{V}^*) = \arg\min_{U,V} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{V}\boldsymbol{U}\|_2^2 + \alpha\|\boldsymbol{V}\|_1 \quad \text{przy ograniczeniach: } \|\boldsymbol{u}_{j,:}\|_2 = 1 \text{ dla } 1 \leqslant j \leqslant k \tag{3.8}$$

Patrząc na (3.6) oraz (3.4) mamy $\|\boldsymbol{X} - \boldsymbol{W}^2\boldsymbol{W}^1\boldsymbol{X}\|_2^2$, gdzie $\boldsymbol{W}^2$ odpowiada $\boldsymbol{V}$ i jest traktowane jako słownik, a $\boldsymbol{W}^1\boldsymbol{X}$ odpowiada macierzy $\boldsymbol{U}$ i jest częścią kodująca (zakodowaną reprezentacją). W prezentowanym rzadkim autoenkoderze wymaga się, aby obie macierze wag były rzadkie oraz bardziej naturalne jest użycie $\boldsymbol{W}^1$ jako macierzy transformacji i $\boldsymbol{W}^2$ jako transformacji odwrotnej. W uczeniu słownikowym wymagane jest, aby słownik był rzadki (ograniczenia z normą ℓ^1)

3.3 Uczenie zrandomizowane, uczenie ekstremalne

Sieć neuronowa z jedną warstwą ukrytą i nieliniową funkcją aktywacji umożliwia wygenerowanie nieliniowej granicy decyzyjnej. Istnieje cały szereg wariantów strojenia takich struktur, głównie bazujących na idei wstecznej propagacji błędu.

Alternatywne podejście do uczenia polega na ustaleniu wag warstwy wejściowej w sposób losowy. Przepuszczenie próby uczącej przez tę warstwę skutkować będzie podniesieniem wymiarowości, o ile liczba neuronów będzie dużo większa od liczby atrybutów w danych uczących. Z kolei w przestrzeni o wyższym wymiarze dużo łatwiej znaleźć hiperpłaszczyznę dobrze separującą klasy decyzyjne (por. np. [80]). Przy zamrożonych losowych wagach warstwy ukrytej problem uczenia sprowadza się do strojenia tylko warstwy wyjściowej. Dzięki temu zabiegowi otrzymujemy liniowy klasyfikator wraz z losową warstwą transformująca, który generuje nieliniową granicę decyzyjną w oryginalnej przestrzeni.

Uczenie takie współcześnie spotykane jest pod nazwą uczenia ekstremalnego (ang. extreme learning) [17, 73, 90], jednak idea losowego wyboru parametrów warstw pośrednich sieci jest znana od lat dziewięćdziesiątych np. pod nazwami RVFL (Random Vector Functional Link) lub RFHNs (Randomly Fixed Hidden Neurons) [34, 37][2]. Wykorzystując techniki regularyzacji możemy bardzo efektywnie nastroić sieć, a korzystając z rzadkich regularyzacji możemy również znacznie zredukować rozmiar sieci i ustalić optymalną liczbę neuronów w warstwie pośredniej.

[2] por. ciekawy komentarz [82] zawierający historię rozwoju tej koncepcji.

Uczenie zrandomizowane wymaga jedynie prostego kroku losowania wag warstwy pośredniej, które ustala losowe odwzorowanie pomiędzy wejściem $\boldsymbol{X}$ a wyjściem z warstwy pośredniej $\boldsymbol{V}$ oraz nastrojenia warstwy wyjściowej na danych $\{(\boldsymbol{v}_i, \boldsymbol{y}_i)\}_{i=1}^n$, zamiast na parach $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$. Dobór wag warstwy pośredniej najlepiej zrealizować rozmieszczając je na tle danych wejściowych lub wybrać z danych wejściowych, biorąc pod uwagę geometryczną interpretację neuronu składającego się z części liniowej oraz nieliniowej funkcji aktywacji. Na podobnej idei bazują przekształcenia jądrowe (kernel trick), gdzie standardowo jako parametry przekształcenia wykorzystuje się cały zbiór uczący [80].

4. Część eksperymentalna

4.1 Środowisko testowe i szczegóły numeryczne

Na środowisko testowe składa się komputer z procesorem Intel® Core™ i7-10700 i pamięcią operacyjną o pojemności 128 GiB, pracujący pod kontrolą systemu operacyjnego Ubuntu 22.04.4 LTS. Zasadniczo eksperymenty przygotowano w języku `Python`, z poziomu którego wywoływane są fragmenty kodu związane z uczeniem modeli, napisane w `C++` dla zachowania maksymalnej wydajności. Wszystkie programy skompilowano przy użyciu narzędzi dostępnych w systemie Ubuntu, tj. `g++` lub `gfortran` (dla tych napisanych w języku `Fortran`) w wersji 11.4, z włączoną optymalizacją trzeciego stopnia za pomocą przełącznika `-O3`.

Wykorzystano bibliotekę `Armadillo` v14.0 [67] do realizacji obliczeń numerycznych w `C++`, ponieważ dostarcza jednolity interfejs dla macierzy gęstej (w porządku kolumnowym) i macierzy rzadkiej (w formacie skompresowanym kolumnowym `CSC`).

Implementacje niskopoziomowe procedur `BLAS` i `LAPACK` zaczerpnięto z biblioteki `Intel® Math Kernel Library` [38], którą zainstalowano za pomocą narzędzia `apt`. Bibliteka ta dla zwiększenia wydajności obliczeń korzysta z wielu rdzeni procesora, dlatego podczas eksperymentów wymuszono korzystanie tylko z jednego rdzenia, aby umożliwić uczciwe porównanie algorytmów.

Z biblioteki `liblinear` v2.47, która oryginalnie operuje na macierzach rzadkich w formacie skompresowanym wierszowym `CSR`, wzięto implementację przyciętej metody Newtona [23], do której dodano obsługę macierzy gęstej oraz typu zmiennoprzecinkowego pojedynczej precyzji. Z tej samem biblioteki wybrano implementację metody spadku względem współrzędnych dla regresji logistycznej z regularyzacją ℓ^1 [86], dodając również obsługę macierzy gęstych oraz typu zmiennoprzecinkowego pojedynczej precyzji.

Klasą bazową każdego modelu jest klasa abstrakcyjna `function` reprezentująca funkcję celu, zawierająca interfejs metod obliczających np. gradient funkcji celu bądź iloczyn macierzy Hessego funkcji celu przez dany wektor. Przygotowano także klasę `functionf`,

w której podmieniono typ zmiennoprzecinkowy podwójnej precyzji na typ zmiennoprzecinkowy pojedynczej precyzji. W pierwotnej implementacji metody Newtona z biblioteki `liblinear` należy do funkcji przekazać wskaźnik na obiekt klasy `function`, chcąc znaleźć optimum danej funkcji celu — korzystając z mechanizmu szablonów w C++ umożliwiono przekazywanie obiektów obu klas.

Właściwości modelu regresji logistycznej (spójrz na poniższy listing) należy ustawić za pomocą parametrów szablonu:

- pierwszy parametr włącza wyraz wolny (jeśli ustawiono go na `true`),
- drugi parametr aktywuje preconditioning,
- trzeci parametr umożliwia użycie dokładnej minimalizacji kierunkowej,
- czwarty parametr określa typ macierzy (albo `arma::Mat<>` dla macierzy gęstej, albo `arma::SpMat<>` dla macierzy rzadkiej).

Na podstawie użytego typu zmiennoprzecinkowego kompilator decyduje, którą wersję interfejsu odziedziczyć:

```
template<
    bool use_intercept,
    bool use_precon,
    bool use_exact_ls,
    typename matrix_class,
    typename el_type = std::remove_cvref_t<matrix_class>::elem_type
>
class logistic_regression : public std::conditional_t<
    std::is_same_v<double, el_type>,
    function,
    std::conditional_t<
        std::is_same_v<float, el_type>,
        functionf,
        void>>
{
// ...
};
```

Następnie w klasie `logistic_regression` dostarczono implementacje abstrakcyjnych metod, tak by reprezentowała podstawowy model regresji logistycznej bez regularyzacji. W kolejnych krokach wykorzystano za pośrednictwem dziedziczenia tę klasę do utworzenia pozostałych klas reprezentujących poszczególne modele.

Dokonano modyfikacji procedury `SYMMLQ`[1], dodając wsparcie dla typu zmiennoprzecinkowego pojedynczej precyzji oraz usuwając ograniczenie związane z procedurami `Aprod` i `Msolve` (przyjmowały one tylko trzy parametry: rozmiar macierzy `n`, wektor

[1]Kod źródłowy pochodzi z `https://web.stanford.edu/group/SOL/software/symmlq/symmlq-f77.zip`

wejściowy `x` i wektor wynikowy `y`), aby móc przekazywać obiekty reprezentujące funkcje celu w analogiczny sposób, jak miało to miejsce w implementacji metody Newtona w bibliotece `liblinear`.

W implementacji algorytmu `L-BFGS-B` v3.0[2] [88] pozbyto się zależności od biblioteki `LINPACK` na rzecz jej następcy, tj. biblioteki `LAPACK`, tzn. zastąpiono procedury `DPOFA` i `DTRSL` odpowiednio procedurami `DPOTRF` i `DTRTRS` [2] — w ten sposób całość obliczeń oparto na bibliotece `Intel® Math Kernel Library`. Istniejącą implementację niskim kosztem przerobiono w ten sposób, by działała również dla typu zmiennoprzecinkowego pojedynczej precyzji.

W implementacji metody wewnętrznego punktu[3] dodano opcję logowania postępu do pliku zgodnie z przyjętym schematem. Autorzy implementacji przyjęli wierszowy porządek macierzy danych, dlatego do pomiaru włączono czas potrzebny na transpozycję macierzy (transpozycja nie jest dokonywana w miejscu). Porzucono pomysł adaptacji metody do typu pojedynczej precyzji, ponieważ wiązało się to ze zmianami w wielu plikach.

4.2 Obliczanie iloczynu XX^{T} w przypadku macierzy rzadkiej

W modelu regresji logistycznej z regularyzacją ℓ^2 po zastosowaniu formuły Woodbury'ego w celu obliczenia macierzy odwrotnej do macierzy Hessego pojawił się składnik $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ o wymiarach $n \times n$. Intuicyjnie w pewnym zakresie powinna pojawić się korzyść z jednorazowego obliczenia tego iloczynu, by później móc się do niego odwoływać podczas rozwiązywania układu równań. Dla macierzy gęstej iloczyn ten oblicza procedura `?SYRK` z biblioteki `BLAS`, gdzie „?" należy zastąpić literą `D` dla podwójnej precyzji lub `S` dla pojedynczej precyzji, a w wyniku powstaje macierz dolno- bądź górnotrójkątna (macierz kwadratowa z wypełnionym dolnym bądź górnym trójkątem), ponieważ mamy do czynienia z macierzą symetryczną, więc nie ma potrzeby obliczania całej macierzy. By maksymalnie wykorzystać pamięć podręczną autorzy procedury[4] budują wynik kolumna po kolumnie.

W przypadku macierzy rzadkiej sposób obliczania się komplikuje z uwagi na różne

[2]Kod źródłowy pochodzi z `https://users.iems.northwestern.edu/~nocedal/Software/Lbfgsb.3.0.tar.gz`

[3]Kod źródłowy pochodzi z `https://web.stanford.edu/~boyd/l1_logreg/download/l1_logreg-0.8.2.tar.gz`

[4]Kod źródłowy znajduje się pod adresem `https://www.netlib.org/lapack/explore-html/dc/d05/dsyrk_8f_source.html`

sposoby przechowywania macierzy, dlatego przetestowano dwa podejścia dla kilku losowych macierzy rzadkich w porządku kolumnowym o różnym stopniu wypełnienia[5], przyjąwszy wynik jako macierz gęstą dolnotrójkątną:

– w pierwszym przypadku wykorzystano „naiwny" sposób obliczenia wyniku, polegający na sumowaniu iloczynów zewnętrznych[6] kolejnych kolumn macierzy $\boldsymbol{X}$:

$$\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} \equiv \sum_{j=1}^{p} \boldsymbol{x}_{:,j}\boldsymbol{x}_{:,j}^{\mathrm{T}}, \tag{4.1}$$

a podsumowuje to poniższy kod (`XXT` to tablica o długość $n \cdot n$):

```
for (arma::uword i = 0; i < X.n_cols; ++i)
    for (auto it = X.begin_col(i); it != X.end_col(i); ++it)
        for (auto jt = it; jt != X.end_col(i); ++jt)
            XXT[it.row()*X.n_rows + jt.row()] += *it * *jt;
```

Dla wysokiego stopnia wypełnienia macierzy $\boldsymbol{X}$ czas obliczeń może się wydłużyć z powodu wielokrotnego przechodzenia po macierzy wynikowej. W tabeli wynikowej temu sposobowi przypisano nr 1.

– w drugim przypadku zaadaptowano procedurę `?SYRK` zapisującą wynik w dolnym trójkącie, gdzie wynik liczony jest kolumna po kolumnie (`col_it` to alias dla typu iteratora pozwalającego przejść po kolumnie macierzy):

```
auto cmp = [](const auto &a, const auto &b) {
    if (a.row() == b.row())
        return a.col() > b.col();
    return a.row() > b.row();
};
std::priority_queue<col_it, std::vector<col_it>, decltype(cmp)> nzi(cmp);
for (arma::uword j = 0; j < X.n_cols; j++)
    if (X.begin_col(j).col() == j)
        nzi.push(X.begin_col(j));
while (!nzi.empty()) {
    auto col = nzi.top();
    nzi.pop();
    auto j = col.row(), k = col.col();
    for (auto it = col; it != X.end_col(k); ++it)
        XXT[it.row() + j*X.n_rows] += *col * (*it);
    ++col;
    if (col != X.end_col(k))
        nzi.push(col);
}
```

W tabeli wynikowej temu sposobowi przypisano nr 2.

[5] Stopień wypełnienia to iloraz liczby niezerowych elementów i iloczynu wymiarów macierzy $n \cdot p$.
[6] Działanie to w bibliotece `BLAS` realizuje procedura `?SYR`.

Tabela 4.1: Czas wyznaczania iloczynu $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ dla macierzy rzadkiej w podwójnej precyzji dla trzech opisanych sposobów (nr), przy zadanym wypełnieniu (sp), n — l. wierszy, p — l. kolumn.

		p=100000			p=1000000		
sp	nr	n=100	n=1000	n=10000	n=100	n=1000	n=10000
10^{-3}	1	0,001246	*0,005947*	*0,444052*	0,011596	*0,036657*	*2,44171*
	2	0,002971	0,045212	1,50472	0,039607	1,1049	18,1561
	3	*0,000806*	0,010594	0,895082	*0,010979*	0,233275	7,56482
10^{-2}	1	*0,001857*	*0,045611*	*13,3153*	*0,017567*	*0,417333*	134,202
	2	0,026391	0,633123	13,8742	0,567729	11,1309	181,413
	3	0,0046	0,215629	13,7634	0,113054	2,49794	*100,096*
10^{-1}	1	*0,019872*	*2,67712*	395,156	*0,199497*	*23,97*	4053,38
	2	0,389611	8,9242	*292,341*	5,07064	101,59	3181,89
	3	0,112088	6,10633	311,866	1,48863	62,9627	*2946,42*

Tabela 4.2: Czas wyznaczania iloczynu $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ dla macierzy rzadkiej w pojedynczej precyzji dla trzech opisanych sposobów (nr), przy zadanym wypełnieniu (sp), n — l. wierszy, p — l. kolumn.

		p=100000			p=1000000		
sp	nr	n=100	n=1000	n=10000	n=100	n=1000	n=10000
10^{-3}	1	0,001206	*0,004404*	*0,301092*	0,011767	*0,033401*	*2,01659*
	2	0,002877	0,042853	1,22071	0,039498	1,08033	16,8522
	3	*0,000741*	0,007922	0,71066	*0,010352*	0,234735	6,96401
10^{-2}	1	0,001849	*0,03921*	12,8991	*0,017701*	*0,362959*	130,754
	2	0,026887	0,616852	13,1467	0,56545	10,8668	179,486
	3	*0,004169*	0,191002	*12,63*	0,088646	2,41417	*94,5575*
10^{-1}	1	*0,020864*	*1,89437*	*243,705*	*0,206076*	*17,9507*	*2477,67*
	2	0,380083	8,25811	275,335	5,05641	97,327	3002,67
	3	0,095689	5,5231	263,007	1,32293	55,2086	2566,24

– sprawdzono też czas obliczenia wyrażenia `X*X.t()`, choć iloczyn macierzy rzadkich biblioteka `Armadillo` traktuje jako macierz rzadką. W tabeli wynikowej temu sposobowi przypisano nr 3.

W eksperymencie zmierzono czas (w sekundach) jednokrotnego policzenia wyniku dla losowo wygenerowanej macierzy liczb z przedziału $[0, 1)$ o podanych wymiarach i stopniu wypełnienia. Wynik eksperymentu przedstawiają tabele 4.1 i 4.2.

W większości przypadków sposób pierwszy („naiwny”) przewyższa pozostałe sposoby pod kątem czasu obliczeń. Mimo zaadaptowania kodu z biblioteki `BLAS` na potrzeby

macierzy rzadkiej, tylko w jednym przypadku (n=10000, p=100000, sp=10^{-1}, podwójna precyzja) udało się przewyższyć pozostałe sposoby — testowane macierze okazały się zbyt rzadkie, by wykorzystać potencjał proponowanego algorytmu.

Na podstawie wyników powyższego eksperymentu wybrano sposób pierwszy do realizacji iloczynu $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ w przypadku macierzy rzadkiej.

4.3 Porównanie implementacji metody gradientu sprzężonego

W pakiecie `scikit-learn` znajduje się klasa `LogisticRegression`, która trenuje model regresji logistycznej z regularyzacją ℓ^2 (odpowiada on modelowi `l2_logreg_ordinary`, por. tab. 4.5) m.in. za pomocą metody Newtona, w której kierunek poszukiwań wyznacza algorytm gradientu sprzężonego bez preconditioningu (parametr `solver='newton-cg'` w konstruktorze klasy). Analiza kodu źródłowego metody Newtona użytej do uczenia modelu `LogisticRegression` wykazała następujące różnice w stosunku do implementacji metody Newtona znajdującej się w bibliotece `liblinear`:

- warunek stopu metody Newtona to $\|\boldsymbol{g}\|_\infty \leqslant \epsilon$, gdzie $\boldsymbol{g}$ to gradient funkcji kosztu w bieżącej iteracji, a ϵ to zadana przez użytkownika dokładność — w pracy obserwowano wartość pochodnej kierunkowej, natomiast pierwotnie w `liblinear` optymalizacja kończy się w momencie wystąpienia warunku $\|\boldsymbol{g}\|_2 \leqslant tol \cdot \|\boldsymbol{g}_0\|_2$, gdzie $\boldsymbol{g}_0$ to gradient funkcji kosztu w punkcie $\mathbf{0}$,
- warunek stopu metody gradientu sprzężonego to $\|\boldsymbol{r}^{(i)}\|_1 \leqslant \min\left(\frac{1}{2}, \sqrt{\|\boldsymbol{g}\|_1}\right) \cdot \|\boldsymbol{g}\|_1$, gdzie $\boldsymbol{r}^{(i)}$ to wektor różnic zdefiniowany tak samo, jak w algorytmie 2 — w `liblinear` warunek stopu opiera się na badaniu stosunku redukcji wartości funkcji kwadratowej (aproksymującej funkcję celu) w bieżącym kroku do średniej redukcji, a próg wynosi $\min\left(\epsilon_{\mathrm{cg}}, \sqrt[4]{\boldsymbol{g}^{\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{g}}\right)$ [23], gdzie ϵ_{cg} to zadana przez użytkownika dokładność,
- w minimalizacji kierunkowej, oprócz kryterium dostatecznej redukcji wartości funkcji[7] (tak jak w algorytmie 1), autorzy skorzystali dodatkowo z kryterium wystarczającej krzywizny[8], a procedurę minimalizacji zaczerpnęli z implementacji algorytmu `L-BFGS-B` w języku `Fortran` — w `liblinear` użyto natomiast wrotnej minimalizacji kierunkowej ze współczynnikami $\alpha = 0{,}01$ i $\beta = 0{,}5$.

W celach porównawczych obu implementacji przygotowano eksperyment, w którym trenowano dwa modele regresji logistycznej:

[7] znanego również jako kryterium Armijo [59]

[8] w połączeniu z kryterium Armijo są to tzw. (silne) kryteria Wolfe'a [59]

– `LogisticRegression` wykorzystujący do uczenia implementację metody Newtona z pakietu `scikit-learn` (w tabeli oznaczono ją `newton-cg`),
– `l2_logreg_ordinary` wykorzystujący do uczenia implementację metody Newtona z biblioteki `liblinear` (w tabeli oznaczono ją `liblinear`),

a uczenia dokonano na zbiorach danych wymienionych w tabeli 4.4 przy ustalonej wartości $\lambda = 10^{-4}$. Jakościowo obie procedury uczenia i oba modele są matematycznie równoważne, tzn. zbiegają do tego samego optimum. Przetestowano dwa ustawienia modeli: uwzględniający wyraz wolny (`b+`) i bez wyrazu wolnego (`b-`). Sprawdzono zachowanie obu metod dla dwóch precyzji typu zmiennoprzecinkowego: podwójnej (oznaczono ją literą `d`) i pojedynczej (oznaczono ją literą `s`). Dla typu podwójnej precyzji użyto $\epsilon = 10^{-8}$, natomiast dla typu pojedynczej precyzji wybrano $\epsilon = 10^{-4}$. We własnej implementacji modelu (`l2_logreg_ordinary`) wyłączono preconditioning. Wyniki przedstawiono w tabeli 4.3.

Na pierwszy rzut oka widać, że dla zbiorów rzadkich metoda `newton-cg` z pakietu `scikit-learn` pod względem czasu wykonania działała znacznie szybciej od metody Newtona z biblioteki `liblinear`, choć w większości przypadków wykonała więcej iteracji. Przyczyną tego stanu rzeczy są prawdopodobnie różnice w obsłudze macierzy rzadkiej, a także różnice wymienione wcześniej. Z kolei dla macierzy gęstych widać uderzającą różnicę na korzyść implementacji wziętej z biblioteki `liblinear`. Zauważono również skrócenie czasu uczenia po przejściu na pojedynczą precyzję, jednak w przypadku metody `newton-cg` dla testowanych macierzy rzadkich nie zaobserwowano dwukrotnego przyspieszenia.

4.4 Przebieg eksperymentu dla modeli z regularyzacją ℓ^2 oraz ℓ^1

Na podstawie zaprezentowanych wyników, w dalszym toku badań, jako punkt referencyjny przy porównywaniu modeli regresji logistycznej z regularyzacją ℓ^2 przyjęto model `l2_logreg_ordinary`. Dzięki temu uzyskano jednolity szkielet pętli uczącej z jednolitym warunkiem stopu oraz z jednolitymi bibliotekami numerycznej algebry liniowej. W pętli tej wygodnie można podmieniać i badać poszczególne elementy mające wpływ na zbieżność, np. sposób minimalizacji kierunkowej, dodanie preconditioningu czy podmianę metody optymalizacji. W przypadku modeli z regularyzacją ℓ^1 różnice pomiędzy modelami są znaczne, toteż każdy model stanowi osobną klasę.

Do przetestowania algorytmów wykorzystano cztery zbiory testowe: dwa zestawy danych gęstych (`extreme` i `PCam16k`) oraz dwa zestawy danych w postaci rzadkiej (`RCV1`

Tabela 4.3: Porównanie czasów optymalizacji dla wybranych zbiorów danych. k oznacza liczbę iteracji metody Newtona, n_{cg} to skumulowana liczba iteracji algorytmu gradientu sprzężonego, f^* to końcowa wartość funkcji, a t to czas wyrażony w sekundach.

zbiór	model	metoda	k	n_{cg}	f^*	t
RCV1	b+d	liblinear	10	41	7059,81	2,68
		newton-cg	15	51	7059,81	**1,22**
	b+s	liblinear	7	27	7059,75	1,34
		newton-cg	10	30	7059,81	**0,99**
	b-d	liblinear	9	31	7070,56	2,18
		newton-cg	16	48	7070,56	**0,83**
	b-s	liblinear	9	28	7070,50	1,26
		newton-cg	10	28	7070,56	**0,84**
news20	b+d	liblinear	11	66	6482,03	6,76
		newton-cg	17	94	6482,03	**2,23**
	b+s	liblinear	12	57	6481,78	3,95
		newton-cg	11	65	6482,02	**2,71**
	b-d	liblinear	13	62	6493,09	6,59
		newton-cg	17	87	6493,09	**2,03**
	b-s	liblinear	10	51	6492,83	3,80
		newton-cg	10	47	6493,08	**1,88**
extreme	b+d	liblinear	13	88	1970,53	**12,92**
		newton-cg	18	129	1970,53	16,19
	b+s	liblinear	17	89	1970,53	**7,77**
		newton-cg	13	89	1970,53	10,10
	b-d	liblinear	12	80	1970,53	**11,82**
		newton-cg	17	101	1970,53	12,95
	b-s	liblinear	7	38	1970,53	**3,24**
		newton-cg	14	75	1970,53	11,27
PCam16k	b+d	liblinear	6	31	9678,65	**60,27**
		newton-cg	16	56	9678,65	103,15
	b+s	liblinear	11	42	9678,63	**20,53**
		newton-cg	13	52	9678,65	46,27
	b-d	liblinear	8	33	10419,44	**30,30**
		newton-cg	12	42	10419,44	109,83
	b-s	liblinear	7	28	10419,43	**13,36**
		newton-cg	7	25	10419,44	45,08

Tabela 4.4: Zestawy danych wykorzystane w eksperymentach.

Dane	Opis	Rozmiar $n \times p$	Typ
`extreme`	Dane syntetyczne, losowo wygenerowane 10000 cech w problemie dwóch spiral	10000×10000	gęste
`PCam16k`	Dane rzeczywiste pobrane z repozytorium OpenML komendą: `X, y = fetch_openml(data_id=42811, return_X_y= True)`	16000×27649	gęste
`RCV1`	Baza textminingowa Reuters RCV1 [51] po wektoryzacji, wybrana klasa „33”	23149×47236	rzadkie
`news20`	dane textminingowe po wektoryzacji zebrane z 20 grup dyskusyjnych zawierające ponad 18000 wpisów, pobrane komendą: `data = fetch_20newsgroups_vectorized(subset='all', return_X_y= False, normalize=True)` wybrana klasa „1”	18846×130107	rzadkie

i `news20`). Charakterystyka zestawów danych wykorzystywanych w eksperymentach została przedstawiona w tabeli 4.4. W trakcie eksperymentu pokazany będzie wpływ liczby próbek n oraz wartości parametru regularyzacyjnego λ na przebieg procesu uczenia. Na wykresach i w tabelach liczba próbek przedstawiona będzie w sposób znormalizowany, tj. w odniesieniu do liczby atrybutów (n/p). Wszystkie algorytmy zaprojektowane są pod przypadek $n \leqslant p$, natomiast zysk z proponowanych podejść widoczny jest dla mniejszych liczności prób $n \ll p$. W eksperymentach przyjęto minimalną liczność $n = 20$ próbek, natomiast wartość końcową uzależniono od rozmiaru dostępnych danych lub ograniczeń pamięciowych. Eksperymenty wykonano dla 10 wartości n/p rozmieszczonych w skali logarytmicznej. Oprócz liczby próbek na przebieg procesu uczenia ma wpływ parametr regularyzacyjny λ. Część eksperymentów wykonano przy arbitralnie ustalonej wartości $\lambda = 10^{-4}$ przy zmianie parametru n. Przedstawiono również eksperyment ukazujący wpływ parametru λ na przebieg procesu uczenia.

W eksperymentach wykorzystano algorytmy opisane szczegółowo w rozdziale 2. W przypadku regularyzacji ℓ^2 główny model referencyjny to `l2_logreg_ordinary`, zaś

Tabela 4.5: Modele wykorzystane w eksperymentach wraz z szacowaną złożonością obliczeniową pojedynczej iteracji metody Newtona dla przypadku gęstego w wariancie optymistycznym i pesymistycznym; n — liczba próbek, p — liczba atrybutów, k_i — liczba iteracji procedury gradientu sprzężonego (zwykle mniejsza od liczby wierszy/kolumn macierzy układu).

Model	Opis	Złożoność iteracji
`l2_logreg_ordinary`	metoda Newtona wraz liniowym gradientem sprzężonym, odpowiednik procedury `newton-cg` pakietu `scikit-learn`	$O(npk_1)$, $O(np^2)$
`l2_logreg_woodbury`	odwrotność macierzy $\lambda \boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X}$ wyznaczana jest w każdej iteracji za pomocą formuły Woodbury'ego, por. sekcja 2.4.2	$O(npk_2)$, $O(n^2p)$
`l2_logreg_wdbr`	odwrotność macierzy $\lambda \boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X}$ wyznaczana jest w każdej iteracji za pomocą formuły Woodbury'ego, przy wcześniejszym stablicowaniu macierzy $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$, por. sekcja 2.4.2; koszt stablicowania iloczynu $O(n^2p)$	$O(n^2k_3)$
`l2_logreg_qr`	transformacja macierzy $\boldsymbol{X}^{\mathrm{T}}$ za pomocą rozkładu QR, por. sekcja 2.4.1; sposób bardziej naturalny uwzględniając porządek kolumnowy macierzy $\boldsymbol{X}^{\mathrm{T}}$; koszt wyznaczenia rozkładu $O(n^2p)$	$O(n^2k_4)$
`l2_logreg_lq`	transformacja macierzy $\boldsymbol{X}$ za pomocą rozkładu LQ, por. sekcja 2.4.1; sposób bardziej naturalny, biorąc pod uwagę rozmiar macierzy ($n < p$); koszt wyznaczenia rozkładu $O(n^2p)$	$O(n^2k_5)$

pozostałe to własne implementacje autora: `l2_logreg_woodbury`, `l2_logreg_wdbr`, `l2_logreg_qr`, `l2_logreg_lq`, podsumowane w tabeli 4.5. Wszystkie procedury mają podobną strukturę: zewnętrzna pętla (kroki algorytmu uczenia) i wewnętrzna (iteracyjne rozwiązywanie odpowiedniego układu równań do wyznaczenia kolejnego kierunku) oraz skalarna optymalizacja kierunkowa. Wszystkie porównywane algorytmy mają ustawione identycznie warunek zatrzymania pętli iteracyjnej rozwiązującej układ równań liniowych

oraz pętli zewnętrznej zatrzymującej algorytm uczenia. Każdy algorytm zaczynał optymalizację z punktu początkowego $\boldsymbol{w} = \mathbf{0}$.

Od strony formalno-teoretycznej wszystkie te procedury powinny mieć równoważne iteracje pętli zewnętrznej w tym sensie, że w każdym kroku spadek wartości funkcji celu powinien następować o tę samą wartość. Rozwiązywane w każdym kroku układy równań są różne, stąd liczby iteracji algorytmów rozwiązujących układ równań liniowych mogą od siebie odbiegać, bo inne są także wymiarowości macierzy układów równań, jednakże wektory wag w każdej iteracji mogą być sprowadzone do postaci równoważnej (wspólnej przestrzeni oryginalnej), np. w przypadku metody z rozkładem QR/LQ konieczne jest przemnożenie wag w przestrzeni transformowanej przez macierz Q.

Przed uruchomieniem eksperymentów zweryfikowano empirycznie poprawność algorytmów na mniejszych zbiorach danych i wszystkie algorytmy dawały identyczne wyniki, zarówno jeżeli idzie o liczbę iteracji pętli zewnętrznej, jak i wartości funkcji celu w kolejnych iteracjach oraz wynik końcowy w postacji wag.

Od strony numerycznej na wykonanie algorytmu mają wpływ następujące elementy (por. sekcja 2.4):

procedura rozwiązywania układu równań liniowych: wykorzystywane są gradient sprzężony wraz z przyciętą metodą Newtona [23] oraz SYMMLQ [60];

bias użycie wyrazu wolnego (`b+`) lub model bez wyrazu wolnego (`b-`);

użycie dokładnej minimalizacji kierunkowej (`e+`) lub przybliżonej (`e-`);

preconditioner wykorzystanie tzw. preconditioningu (macierzy ściskającej) celem poprawy uwarunkowania macierzy układu i zbieżności procedury iteracyjnej rozwiązującej układ równań; możliwe wartości: `p+` (użycie preconditionera), `p-` (brak, układ oryginalny);

typ zmiennopozycyjny `d` (podwójna precyzja, 64 bity), `s` (pojedyncza precyzja, 32 bity) [14].

Jeden z eksperymentów porównawczych przedstawiono w tabeli 4.6. Tabela zawiera wyniki uzyskane na zbiorze `lsvt` (por. repozytorium OpenML, zbiór `id=1484`) o rozmiarze $n \times p = 126 \times 309$ przy ustawieniach $\lambda = 10^{-4}$ oraz identycznych pozostałych ustawieniach numerycznych. Jak można zauważyć, w pierwszej iteracji wyniki wszystkich algorytmów zgadzają się z dokładnością do wyświetlanej precyzji. W kolejnych iteracjach widać, że identyczne numerycznie są algorytmy `l2_logreg_ordinary` oraz `l2_logreg_QR`, mimo że drugi z algorytmów pracuje w przetransformowanej przestrzeni, tj. na danych macierzy $\boldsymbol{R}$ z rozkładu QR. Dla obu algorytmów bazujących na formule Woodbury'ego widać drobną rozbieżność numeryczną względem dwóch pozostałych algo-

Tabela 4.6: Porównanie zbieżności testowanych algorytmów na zbiorze `lsvt` (por. repozytorium OpenML zbiór `id=1484`) o rozmiarze $n \times p = 126 \times 309$ przy ustawieniach $\lambda = 10^{-4}$ oraz ustawieniach: `b-p-e-d` (bez wyrazu wolnego, bez preconditioningu, bez dokładnej minimalizacji kierunkowej w typie podwójnej precyzji); `it` oznacza numer iteracji, `gTd` rzut gradientu na nowy kierunek (pochodna kierunkowa), `f` wartość funkcji celu, `cg` liczba iteracji procedury rozwiązującej układ równań liniowych.

	l2_logreg_ordinary			l2_logreg_woodbury			l2_logreg_wdbr			l2_logreg_qr		
it	gTd	f	cg	gTd	f	cg	gTd	f	cg	gTd	f	cg
1	-15,7842	79,1355	4	-15,7842	79,1355	3	-15,7842	79,1355	3	-15,7842	79,1355	4
2	-0,287993	78,9777	4	-0,286014	78,9779	5	-0,287309	78,9778	5	-0,287993	78,9777	4
3	-7,879e-03	78,9737	4	-8,449e-03	78,9737	6	-7,731e-03	78,9738	6	-7,879e-03	78,9737	4
4	-6,324e-06	78,9737	4	-7,944e-05	78,9737	3	-2,655e-04	78,9737	2	-6,324e-06	78,9737	4
5	-3,981e-12	78,9737	3	-6,309e-08	78,9737	7	-7,165e-07	78,9737	7	-3,981e-12	78,9737	3
6				-1,811e-13	78,9737	3	-6,599e-14	78,9737	6			

rytmów, zwłaszcza w pochodnej kierunkowej $\boldsymbol{g}^{\mathrm{T}}\boldsymbol{d}$ (wyraźniejszą począwszy od czwartej iteracji), oraz oba algorytmy wykonały o jedną iteracje więcej, gdyż warunek stopu był ustalony na $|\boldsymbol{g}^{\mathrm{T}}\boldsymbol{d}| < 10^{-8}$. Algorytmy bazujące na formule Woodbury'ego mają do rozwiązania nieco inny układ równań liniowych, stąd liczby iteracji rozwiązujących dany układ (kolumna `cg`) mogą być różne. Ponadto eksperymenty prezentowane w dalszych sekcjach wskazują, że oba algorytmy działają nieco stabilniej przy włączonym preconditioningu. Patrząc na wartość funkcji celu (`f`) wszystkie algorytmu zachowują się identycznie z dokładnością do wyświetlanej precyzji.

4.5 Wyniki eksperymentów dla regularyzacji ℓ^2

W kolejnych tabelach i na wykresach przedstawiono 8 (tylko double) lub 16 wariantów algorytmu uczenia kodowanych odpowiednią konfiguracją parametrów. Przykładowo wyrażenie `b+p-e-d` oznacza model z wyrazem wolnym, bez użycia preconditioningu, bez dokładnej minimalizacji kierunkowej, a obliczenia w typie podwójnej precyzji. Podstawowym typem numerycznym jest 64 bitowy typ zmiennopozycyjny (double) [14]. Wszystkie eksperymenty wykonano również dla typu 32 bitowego (single), głównie po to, aby przebadać zachowanie algorytmów przy obniżonej precyzji, gdzie problemy numeryczne różnych wariantów są bardziej widoczne.

Tabele 4.7, 4.8, 4.9, 4.10 przedstawiają podsumowanie wyników eksperymentów dla różnej liczby próbek uczących (n) oraz wskazują obszary parametrów z przewagą proponowanego algorytmu `l2_logreg_wdbr` nad pozostałymi algorytmami.

Tabela 4.7: Podsumowanie wyników dla danych `extreme`, dla różnych rozmiarów zbioru danych (n). Przewagę algorytmu `l2_logreg_wdbr` nad pozostałymi oznaczono przez „+”. Typ zmiennopozycyjny (d/s), b (+/-) model z/bez wyrazu wolnego, p (+/-) model z/bez preconditionerem, e (+/-) model z/bez dokładnej minimalizacji kierunkowej.

type	b	p	n e	20	36	68	125	232	429	793	1465	2707	4999
d	+	+	+	-	-	+	+	+	+	+	+	+	+
			-	-	-	+	+	+	+	+	+	+	+
		-	+	-	-	+	+	+	+	+	+	+	+
			-	-	-	+	+	+	+	+	+	+	+
	-	+	+	-	-	-	+	+	+	+	+	+	+
			-	-	-	-	+	+	+	+	+	+	+
		-	+	-	-	-	+	+	+	+	+	+	+
			-	-	-	-	+	+	+	+	+	+	+
s	+	+	+	-	-	-	+	+	+	+	+	+	+
			-	-	-	-	+	+	+	+	+	+	+
		-	+	-	-	-	+	+	+	+	+	+	+
			-	-	-	-	+	+	+	+	+	+	+
	-	+	+	-	-	-	+	+	+	+	+	+	+
			-	-	-	-	+	+	+	+	+	+	+
		-	+	-	-	-	-	+	+	+	+	+	+
			-	-	-	-	+	+	+	+	+	+	+

Na rysunkach 4.1, 4.3, 4.5, 4.7 przedstawiono czasy wykonania poszczególnych algorytmów wraz ze wzrostem liczby próbek, uśrednione po dziesięciu powtórzeniach z losowym wyborem zestawu uczącego. Na wykresach wąsami zaznaczono również odchylenia standardowe wyników. Na wykresach na osi OX przedstawiono unormowaną liczbę próbek n/p. Wyniki przedstawiono tylko dla typu `double`. Ponadto dla danych rzadkich nie wykonano algorytmów z rozkładami QR/LQ, gdyż nie gwarantują one rzadkich macierzy $\boldsymbol{Q}$ i $\boldsymbol{R}$, a zastąpienie rzadkiej macierzy danych $\boldsymbol{X}$ przez gęste macierze $\boldsymbol{R}$ i $\boldsymbol{Q}$ wydaje się dalece nieefektywne pamięciowo.

Na rysunkach 4.2, 4.4, 4.6, 4.8 zwizualizowano tempo zbieżności algorytmów dla

Tabela 4.8: Podsumowanie wyników dla danych `PCam16k`, dla różnych rozmiarów zbioru danych (n). Przewagę algorytmu `l2_logreg_wdbr` nad pozostałymi oznaczono przez „+”. Typ zmiennopozycyjny (d/s), b (+/-) model z/bez wyrazu wolnego, p (+/-) model z/bez preconditionerem, e (+/-) model z/bez dokładnej minimalizacji kierunkowej.

typ	b	p	n e	20	38	75	147	286	557	1085	2112	4111	8000
d	+	+	+	-	-	+	+	+	+	+	+	+	-
			-	+	-	+	+	+	+	+	+	+	-
		-	+	+	+	+	+	+	+	+	+	+	-
			-	+	+	+	+	+	+	+	+	+	-
	-	+	+	+	-	+	+	+	+	+	+	+	+
			-	+	+	+	+	+	+	+	+	+	+
		-	+	-	-	+	+	+	+	+	+	+	-
			-	+	-	+	+	+	+	+	+	+	-
s	+	+	+	-	-	-	+	+	+	+	+	+	-
			-	-	-	-	+	+	+	+	+	+	-
		-	+	-	-	+	+	+	+	+	+	+	-
			-	-	-	+	+	+	+	+	+	+	+
	-	+	+	-	-	-	+	+	+	+	+	+	-
			-	-	-	-	+	+	+	+	+	+	-
		-	+	-	-	-	+	+	+	+	+	+	-
			-	-	-	-	+	+	+	+	+	+	-

wybranych rozmiarów danych uczących (n). Wyniki przedstawiono dla wszystkich 16 wariantów algorytmów. Mimo że wszystkie algorytmy powinny mieć zbliżone iteracje, widać pewne odstępstwa spowodowane błędami numerycznymi oraz uwarunkowaniem macierzy. Widać przykładowo, że wpływ preconditionera Jacobiego jest niekorzystny dla algorytmu `l2_logreg_ordinary`, natomiast korzystnie wpływa na czas wykonania pozostałych algorytmów (por. 4.4). Implementacja `l2_logreg_ordinary` znacznie lepiej zachowuje sie dla modelu bez wyrazu wolnego. Ogólnie zgodnie z oczekiwaniami widać zysk zarówno z zastosowania rozkładu QR, jak i formuły Woodbury'ego z tablicowaniem macierzy $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ w stosunku do algorytmu referencyjnego. Ponadto patrząc

Tabela 4.9: Podsumowanie wyników dla danych `news20`, dla różnych rozmiarów zbioru danych (n). Przewagę algorytmu `l2_logreg_wdbr` nad pozostałymi oznaczono przez „+”. Typ zmiennopozycyjny (d/s), b (+/-) model z/bez wyrazu wolnego, p (+/-) model z/bez preconditionerem, e (+/-) model z/bez dokładnej minimalizacji kierunkowej.

typ	b	p	n e	20	42	91	196	419	898	1922	4114	8805	18846
d	+	+	+	+	+	+	+	+	+	+	-	-	-
			-	+	+	+	+	+	+	+	-	-	-
		-	+	+	+	+	+	+	+	+	-	-	-
			-	+	+	+	+	+	+	+	-	-	-
	-	+	+	+	+	+	+	+	+	-	-	-	-
			-	+	+	+	+	+	+	+	-	-	-
		-	+	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	+	-	-	-	-	-	-
s	+	+	+	+	+	+	+	+	+	+	-	-	-
			-	+	+	+	+	+	+	+	-	-	-
		-	+	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-
	-	+	+	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-
		-	+	-	-	-	-	-	-	-	-	-	-
			-	-	-	-	-	-	-	-	-	-	-

na tabele i wykresy można zobaczyć, że dla małych prób procedura `l2_logreg_wdbr` działała najkorzystniej, z wyjątkiem pojedynczych przypadków bardzo małych prób, gdzie korzystniejszy był algorytm z rozkładem QR. Dla danych gęstych widać przewagę proponowanej procedury praktycznie w całym badanym zakresie (do $n/p = 0{,}3$ lub $n/p = 0{,}5$). Zgodnie z oczekiwaniami, rozkład QR był korzystniejszy niż rozkład LQ, ponieważ jest lepiej dostosowany do porządku kolumnowego, w jakim przechowywane były dane po transpozycji. Dla danych rzadkich zysk nie jest tak wyraźny, jak dla danych gęstych. Dokładna analiza złożoności obliczeniowej jest w tym przypadku zagadnieniem złożonym, gdyż zależy ona nie tylko od stopnia wypełnienia macierzy, ale i od stopnia wypełnienia

Tabela 4.10: Podsumowanie wyników dla danych RCV1, dla różnych rozmiarów zbioru danych (n). Przewagę algorytmu l2_logreg_wdbr nad pozostałymi oznaczono przez „+”. Typ zmiennopozycyjny (d/s), b (+/-) model z/bez wyrazu wolnego, p (+/-) model z/bez preconditionerem, e (+/-) model z/bez dokładnej minimalizacji kierunkowej.

typ	b	p	n e	19	42	95	210	462	1017	2233	4902	10760	23617
d	+	+	+	+	+	+	+	+	+	-	-	-	-
			-	+	+	+	+	+	+	-	-	-	-
		-	+	+	+	+	+	+	+	-	-	-	-
			-	-	+	+	+	+	+	-	-	-	-
	-	+	+	+	+	+	+	+	+	-	-	-	-
			-	+	+	+	+	+	+	-	-	-	-
		-	+	+	+	+	+	+	+	-	-	-	-
			-	+	+	+	+	+	+	-	-	-	-
s	+	+	+	+	+	+	+	+	+	+	-	-	-
			-	+	+	+	+	+	+	+	-	-	-
		-	+	-	-	-	-	-	+	-	-	-	-
			-	-	-	-	-	-	+	-	-	-	-
	-	+	+	+	+	+	-	-	+	+	-	-	-
			-	-	+	+	+	-	+	+	-	-	-
		-	+	-	-	-	-	-	+	-	-	-	-
			-	-	-	-	-	-	+	-	-	-	-

kolumn/wierszy oraz rzadkości wyniku (por. [1]).

Kolejny eksperyment pokazuje wpływ parametru λ na przebieg procesu uczenia. Eksperyment przeprowadzono zaczynając od dużej wartości parametru regularyzacyjnego i zmniejszając go kolejno $\lambda = 10^3, 10^2, \ldots, 10^{-4}$. Eksperyment był prowadzony w ten sposób, że wagi z poprzednio wyuczonego modelu stanowiły punkt startowy dla kolejnego modelu[9].

Dla dużych wartości λ w funkcji celu $Q(\boldsymbol{w}; \boldsymbol{dane}) = L(\boldsymbol{w}; \boldsymbol{dane}) + R(\boldsymbol{w})$ dominuje składnik kwadratowy R, dzięki czemu optymalizowana funkcja jest dobrze przybliżana za

[9] ang. warm start

pomocą lokalnej aproksymacji kwadratowej, co prowadzi do szybkiej zbieżności. Wraz ze spadkiem wartości λ zaczyna dominować składnik związany z funkcją wiarygodności L i problem optymalizacyjny staje się bardziej złożony. Ponieważ koszt iteracji w proponowanych algorytmach `l2_logreg_wdbr` oraz `l2_logreg_qr` jest mniejszy, więc zysk z zastosowaniarozkładu QR bądź formuły Woodbury'ego z tablicowaniem macierzy $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ będzie bardziej widoczny, zwłaszcza w sytuacji, gdy algorytm uczenia do zbieżności wymaga większej liczby iteracji. Należy podkreślić, że oba algorytmy przed wejściem do pętli algorytmu uczenia mają pewien narzut związany z policzeniem rozkładu QR lub wyznaczeniem macierzy $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$.

Na rysunkach 4.9, 4.10, 4.11 4.12 przedstawiono wyniki eksperymentu wyznaczania rodziny modeli dla parametru regularyzacyjnego zmieniającego się od dużych wartości do małych: $\lambda = 10^3, 10^2, \ldots, 10^{-4}$. Rysunki przedstawiają skumulowany czas eksperymentu w funkcji malejącej λ. Widać, że dla dużych wartości λ zgodnie z oczekiwaniami przewaga algorytmów `l2_logreg_wdbr` oraz `l2_logreg_qr` jest mniejsza, natomiast staje się ona wyraźniejsza dla coraz mniejszych wartości λ. Ponadto widać w większości obszarów parametru λ przewagę proponowanego algorytmu wykorzystującego formułę Woodbury'ego. Na gęstych danych przewaga obu algorytmów (zwłaszcza dla małych prób) jest wyraźna — algorytm bazujący na rozkładzie QR ma nieznaczną przewagę tylko dla bardzo małych prób oraz dużej wartości λ. Jedynie na rzadkim zborze `RCV1` widać przewagę referencyjnego modelu `l2_logreg_ordinary`, która maleje wraz ze spadkiem wartości λ. Widać również, że przewaga proponowanych algorytmów maleje wraz z podnoszeniem rozmiaru próby. Można również zauważyć, że dla coraz większych prób na zbiorach gęstych zauważalna jest nieznaczna przewaga algorytmu `l2_logreg_woodbury` nad algorytmem `l2_logreg_ordinary` i również pozostałymi. Patrząc na złożoność obliczeniową kroku iteracji wewnętrznej (rozwiązywanie układu równań liniowych przy wyznaczaniu nowego kierunku) trudno wytłumaczyć jednoznacznie to zachowanie. Oba kroki mają podobną złożoność obliczeniową ze wskazaniem na korzyść modelu `l2_logreg_ordinary` — jedyna różnica jest taka, że w przypadku formuły Woodbury'ego rozwiązywany jest układ o mniejszej wymiarowości ($n \times n$ zamiast $p \times p$), co może przełożyć się na mniejszą liczbę iteracji procedury iteracyjnej rozwiązującej układ równań.

Na rysunkach 4.13, 4.14, 4.15 4.16 przedstawiono wyniki eksperymentu wyznaczania rodziny modeli dla parametru regularyzacyjnego zmieniającego się od dużych wartości do małych: $\lambda = 10^3, 10^2, \ldots, 10^{-4}$. W odróżnieniu od poprzedniego eksperymentu, ten był prowadzony w ten sposób, że kolejny model startował zawsze od wag $\boldsymbol{w} = \boldsymbol{0}$. Rysunki przedstawiają czas uśredniony jednorazowego uczenia modelu dla zadanej wartości λ.

Tak jak w poprzednim przypadku, dla dużych wartości λ przewaga algorytmów `l2_logreg_wdbr` oraz `l2_logreg_qr` jest mniejsza, a staje się ona wyraźniejsza dla coraz mniejszych wartości λ. Ponadto można zaobserwować w większości obszarów parametru λ przewagę proponowanego algorytmu wykorzystującego formułę Woodbury'ego. Dla gęstych danych przewaga obu algorytmów jest wyraźniejsza — algorytm bazujący na rozkładzie QR ma nieznaczną przewagę jedynie dla bardzo małych prób. Dla rzadkich zbiorów danych pewien zysk był widoczny dla małych prób na zbiorze `news20`, zaś na zbiorze `RCV1` za każdym razem krócej trwało uczenie modelu referencyjnego. Pokazuje to, że rodzaj rzadkości danych ma istotny wpływ na zachowanie algorytmów.

Przykładowe zagregowane wyniki ilościowe przedstawiono w tabelach 4.11 i 4.12. W tabeli 4.11 umieszczono wyniki podsumowujące eksperyment dla zbioru `PCam16k` (por. wykres 4.14). Wyniki dotyczą przypadku modelu z wyrazem wolnym (`b+`): dla każdego z modeli wybrano najlepsze ustawienia parametrów `p` i `e`, po czym uśredniono czasy z dziesięciu powtórzeń eksperymentu. Analogiczne wyniki dla przypadku rzadkiego zbioru `news20` (por. wykres 4.15) przedstawia tabela 4.12.

Na podstawie przedstawionych wyników widać również, że przewaga algorytmów `l2_logreg_wdbr` oraz `l2_logreg_qr` maleje wraz z podnoszeniem rozmiaru próby, natomiast dla dużych prób na zbiorze gęstym obserwowana jest pewna drobna przewaga algorytmu `l2_logreg_woodbury` względem `l2_logreg_ordinary`. Porównując rozwiązywane układy równań dla modelu `l2_logreg_ordinary`:

$$\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{d} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \lambda\boldsymbol{w}, \tag{4.2}$$

oraz proponowanego modelu `l2_logreg_woodbury` (wersja bez tablicowania):

$$\left(\boldsymbol{D}^{-1} + \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\right)\boldsymbol{z} = \frac{1}{\lambda}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{X}\boldsymbol{w}. \tag{4.3}$$

widać, że złożoność obliczeniowa jest podobna $O(np)$, z przewagą dla formuły klasycznej — jedyne nasuwające się wyjaśnienie jest takie, że układy różnią się wymiarowością na korzyść obliczeń z użyciem formuły Woodbury'ego, tj. macierz układu ma wymiar $n \times n$ zamiast $p \times p$, jak w przypadku standardowym.

W eksperymentach przebadano również wpływ typu zmiennopozycyjnego na dokładność i czas obliczeń. Eksperymenty pokazują, że wszystkie algorytmy przy odpowiednich ustawieniach zachowują się generalnie stabilnie przy obniżonej precyzji, czasami wymagając większej liczby iteracji dla uzyskania zbieżności. Widoczny jest również krótszy czas obliczeń. Wniosek ten może mieć znaczenie przy wykonywaniu obliczeń z użyciem kart

Tabela 4.11: Podsumowanie wyników dla zbioru danych PCam16k — tabela przedstawia czas obliczeń dla najlepszych ustawień modelu, zagregowane po n, λ oraz rodzaju modelu.

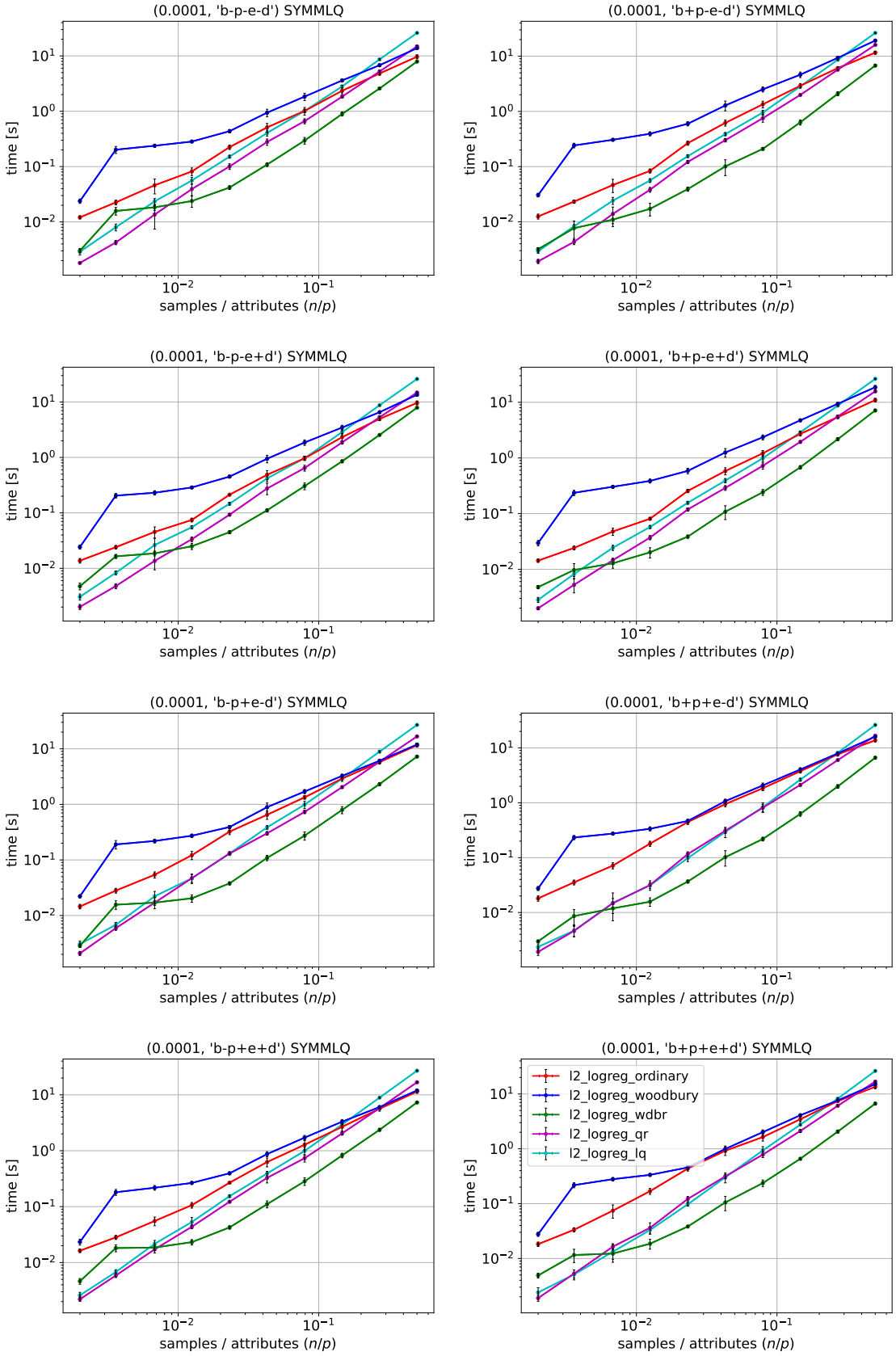
n	λ / model	10^{-4}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
20	l2_logreg_ordinary	0,0987	0,068	0,0471	0,0407	0,0298	0,0233	0,0169	0,0091
	l2_logreg_qr	0,0149	0,0191	0,0171	0,0188	0,0201	0,02	0,0181	0,0182
	l2_logreg_wdbr	0,0165	0,00979	0,00729	0,00756	0,00452	0,00373	0,00273	0,0019
	l2_logreg_woodbury	0,0975	0,0607	0,0401	0,0312	0,0245	0,0186	0,0135	0,00796
38	l2_logreg_ordinary	0,195	0,134	0,0856	0,0745	0,0525	0,0436	0,0311	0,0161
	l2_logreg_qr	0,0331	0,0279	0,0378	0,0352	0,0325	0,0402	0,0212	0,0339
	l2_logreg_wdbr	0,0746	0,0449	0,0507	0,034	0,0345	0,017	0,0148	0,00941
	l2_logreg_woodbury	0,509	0,262	0,194	0,142	0,111	0,0856	0,0612	0,0361
75	l2_logreg_ordinary	0,458	0,318	0,214	0,178	0,122	0,0964	0,0696	0,0364
	l2_logreg_qr	0,075	0,0826	0,0906	0,0913	0,0837	0,0871	0,0777	0,0942
	l2_logreg_wdbr	0,0934	0,0501	0,0618	0,0345	0,0237	0,0282	0,0177	0,0145
	l2_logreg_woodbury	0,596	0,335	0,291	0,218	0,165	0,126	0,0931	0,0479
147	l2_logreg_ordinary	1,17	0,738	0,51	0,453	0,297	0,224	0,161	0,0808
	l2_logreg_qr	0,18	0,254	0,253	0,262	0,264	0,262	0,264	0,264
	l2_logreg_wdbr	0,165	0,124	0,1	0,0797	0,0648	0,0505	0,0432	0,0284
	l2_logreg_woodbury	1,16	0,664	0,568	0,426	0,298	0,227	0,159	0,092
286	l2_logreg_ordinary	3,25	2,19	1,43	1,04	0,891	0,649	0,453	0,211
	l2_logreg_qr	0,666	0,693	0,678	0,695	0,679	0,688	0,692	0,688
	l2_logreg_wdbr	0,269	0,212	0,181	0,143	0,132	0,113	0,0901	0,0681
	l2_logreg_woodbury	2,93	1,67	1,35	0,881	0,706	0,551	0,384	0,211
557	l2_logreg_ordinary	4,96	3,41	2,25	1,76	1,26	0,934	0,659	0,335
	l2_logreg_qr	1,26	1,35	1,32	1,35	1,43	1,45	1,52	1,54
	l2_logreg_wdbr	0,566	0,392	0,391	0,276	0,296	0,269	0,235	0,197
	l2_logreg_woodbury	4,08	2,44	2,07	1,58	1,05	0,898	0,611	0,336
1085	l2_logreg_ordinary	9,96	5,99	4,4	3,84	2,67	1,92	1,32	0,647
	l2_logreg_qr	2,78	3,48	3,37	3,54	3,6	3,73	3,78	3,77
	l2_logreg_wdbr	1,56	1,02	1,1	1,05	0,906	0,899	0,781	0,704
	l2_logreg_woodbury	7,47	3,76	4,09	2,17	2,07	1,68	1,23	0,666
2112	l2_logreg_ordinary	20	12,8	8,59	6,59	4,93	3,4	2,53	1,31
	l2_logreg_qr	10	9,6	9,54	9,48	9,52	9,58	9,57	9,49
	l2_logreg_wdbr	4,67	4,01	3,71	3,49	3,26	3,13	2,76	2,62
	l2_logreg_woodbury	11,8	9,08	6,47	5,17	4,55	3,53	2,34	1,31
4111	l2_logreg_ordinary	41,6	27,7	17,7	13,9	11	7,74	5,46	2,64
	l2_logreg_qr	29,9	28,7	28,4	28,1	27,9	27,9	27,9	27,8
	l2_logreg_wdbr	15,8	13,4	13,4	12	11,9	11,1	11	10,6
	l2_logreg_woodbury	29,7	17	12,6	11,5	8,29	6,64	4,45	2,68
8000	l2_logreg_ordinary	87,2	52,4	36,3	31,3	20,9	15,4	11	5,33
	l2_logreg_qr	94,8	89,7	87,7	86,6	86,8	86,2	85,9	84,8
	l2_logreg_wdbr	45,9	44,7	44,1	41,7	42	40,4	40,8	39,4
	l2_logreg_woodbury	76,3	32,4	30,8	21	14,2	12,3	8,78	5,38

graficznych GP-GPU. W obliczeniach na kartach chętnie korzysta się z typów obniżonej precyzji, gdyż oszczędzamy na narzutach związanych z przesyłaniem danych, przesyłając

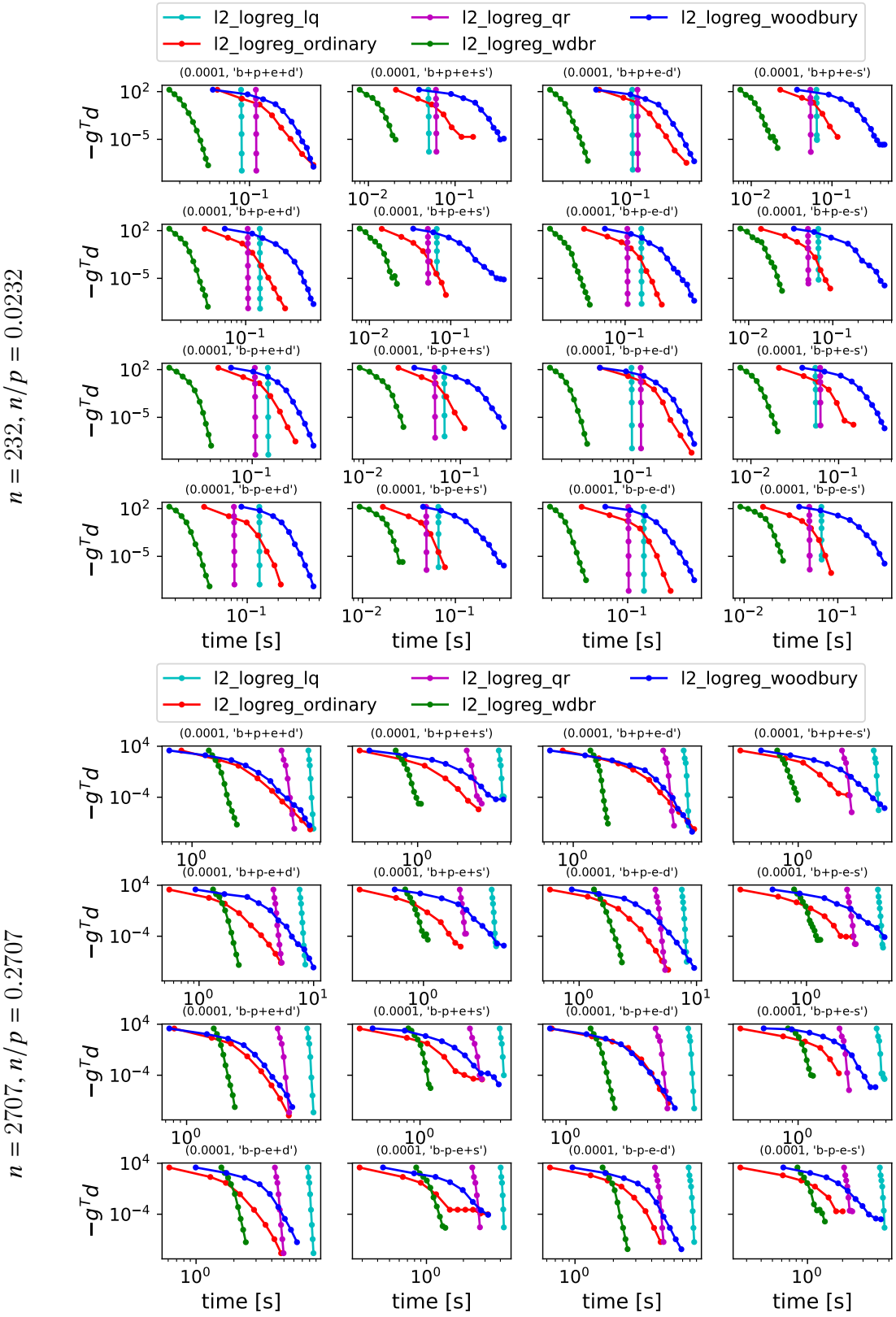
Tabela 4.12: Podsumowanie wyników dla zbioru danych `news20` — tabela przedstawia czas obliczeń dla najlepszych ustawień modelu, zagregowane po n, λ oraz rodzaju modelu.

n	λ / model	10^{-4}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
20	`l2_logreg_ordinary`	0,586	0,406	0,303	0,217	0,168	0,12	0,0968	0,0478
	`l2_logreg_wdbr`	0,321	0,271	0,211	0,155	0,122	0,0927	0,071	0,0419
	`l2_logreg_woodbury`	1,97	1,48	0,998	0,622	0,439	0,342	0,246	0,149
42	`l2_logreg_ordinary`	0,546	0,443	0,369	0,274	0,198	0,129	0,0955	0,054
	`l2_logreg_wdbr`	0,322	0,262	0,201	0,157	0,119	0,0927	0,0668	0,041
	`l2_logreg_woodbury`	2,22	1,66	0,998	0,63	0,444	0,346	0,246	0,142
91	`l2_logreg_ordinary`	0,625	0,46	0,315	0,227	0,173	0,121	0,0832	0,0468
	`l2_logreg_wdbr`	0,33	0,256	0,192	0,151	0,112	0,0855	0,0598	0,0343
	`l2_logreg_woodbury`	2,35	1,63	0,947	0,583	0,426	0,319	0,22	0,114
196	`l2_logreg_ordinary`	0,74	0,516	0,358	0,26	0,193	0,127	0,0833	0,0452
	`l2_logreg_wdbr`	0,334	0,259	0,198	0,156	0,116	0,0882	0,0621	0,0355
	`l2_logreg_woodbury`	2,7	1,65	0,953	0,637	0,433	0,327	0,22	0,112
419	`l2_logreg_ordinary`	0,795	0,549	0,375	0,269	0,196	0,13	0,0847	0,0458
	`l2_logreg_wdbr`	0,37	0,276	0,214	0,168	0,127	0,098	0,0696	0,0421
	`l2_logreg_woodbury`	2,92	1,72	0,95	0,654	0,447	0,321	0,214	0,108
898	`l2_logreg_ordinary`	0,912	0,604	0,405	0,284	0,207	0,134	0,0899	0,049
	`l2_logreg_wdbr`	0,432	0,328	0,258	0,208	0,163	0,132	0,102	0,0722
	`l2_logreg_woodbury`	3,15	1,79	1,03	0,696	0,478	0,35	0,234	0,122
1922	`l2_logreg_ordinary`	1,12	0,72	0,491	0,35	0,254	0,163	0,105	0,0527
	`l2_logreg_wdbr`	0,793	0,6	0,495	0,422	0,36	0,319	0,282	0,24
	`l2_logreg_woodbury`	3,72	2,01	1,14	0,769	0,53	0,393	0,261	0,131
4114	`l2_logreg_ordinary`	1,54	1	0,708	0,523	0,392	0,263	0,161	0,0995
	`l2_logreg_wdbr`	3,67	2,86	2,45	2,19	2,04	1,95	1,85	1,76
	`l2_logreg_woodbury`	4,58	2,48	1,41	0,951	0,668	0,509	0,351	0,19
8805	`l2_logreg_ordinary`	2,75	1,81	1,26	0,936	0,698	0,468	0,321	0,143
	`l2_logreg_wdbr`	21,4	17,6	15	13	12,1	11,3	10,9	10,1
	`l2_logreg_woodbury`	6,25	3,27	1,89	1,33	0,962	0,733	0,503	0,266
18846	`l2_logreg_ordinary`	5,88	4,03	2,81	2,11	1,57	1,05	0,698	0,37
	`l2_logreg_wdbr`	106	84,6	67,3	57,8	53	50,4	47,8	45,3
	`l2_logreg_woodbury`	11,2	5,82	3,5	2,45	1,73	1,32	0,895	0,453

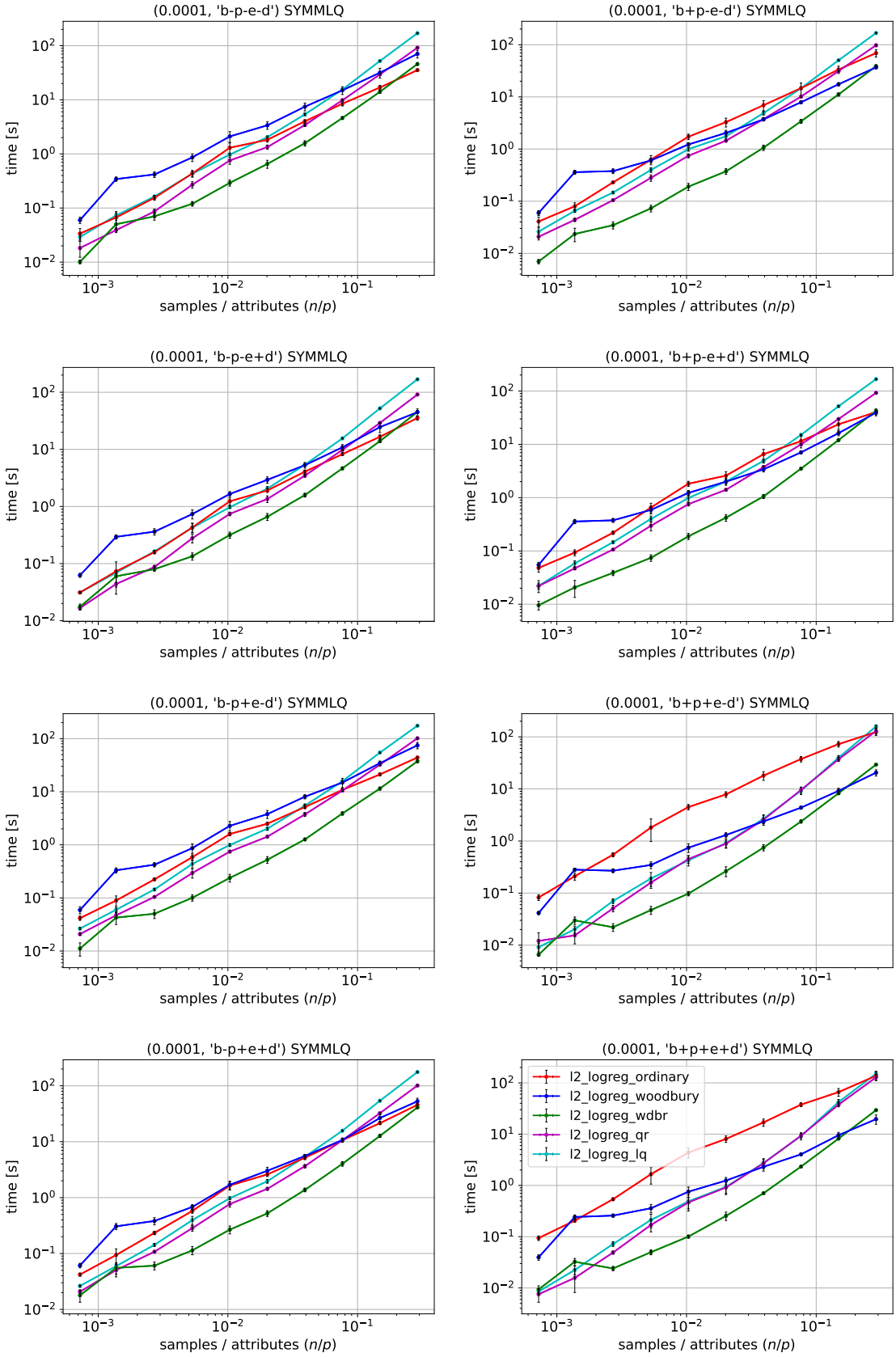
na kartę dane o mniejszym rozmiarze.



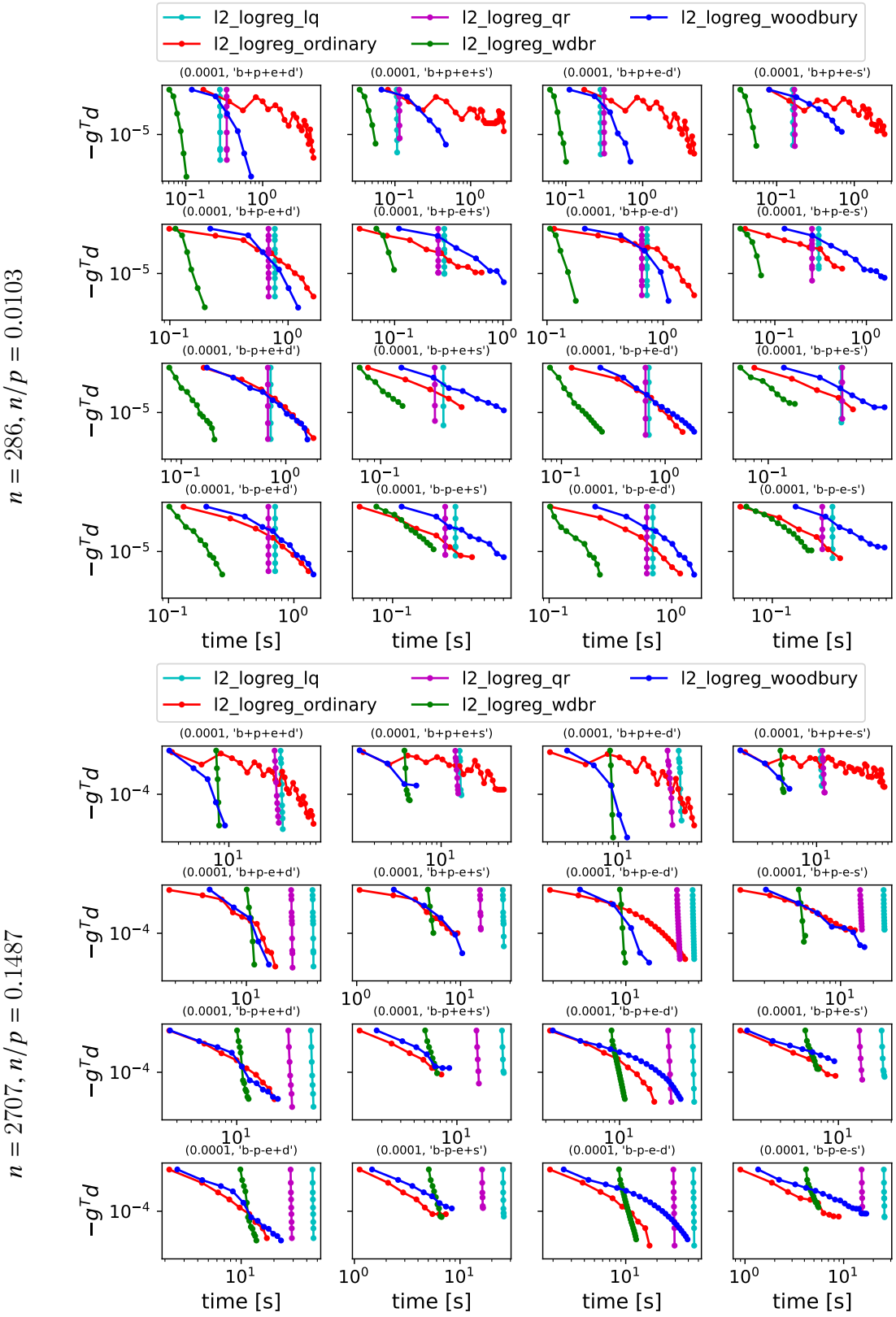
Rysunek 4.1: Porównanie czasów strojenia modeli w funkcji n/p. Dane gęste: `extreme`.



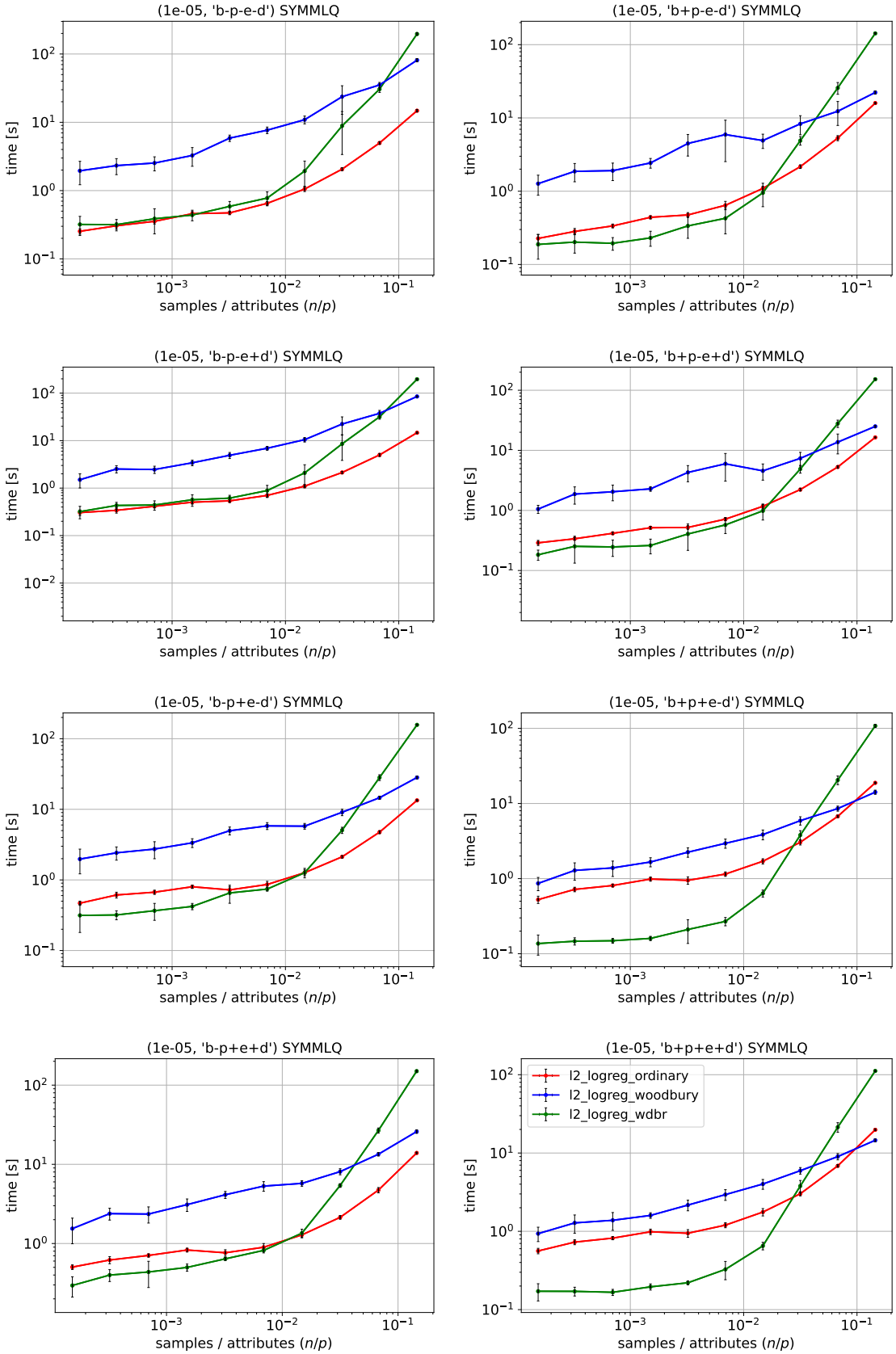
Rysunek 4.2: Porównanie zbieżności procedur dla różnej liczby próbek. Dane: `extreme`.



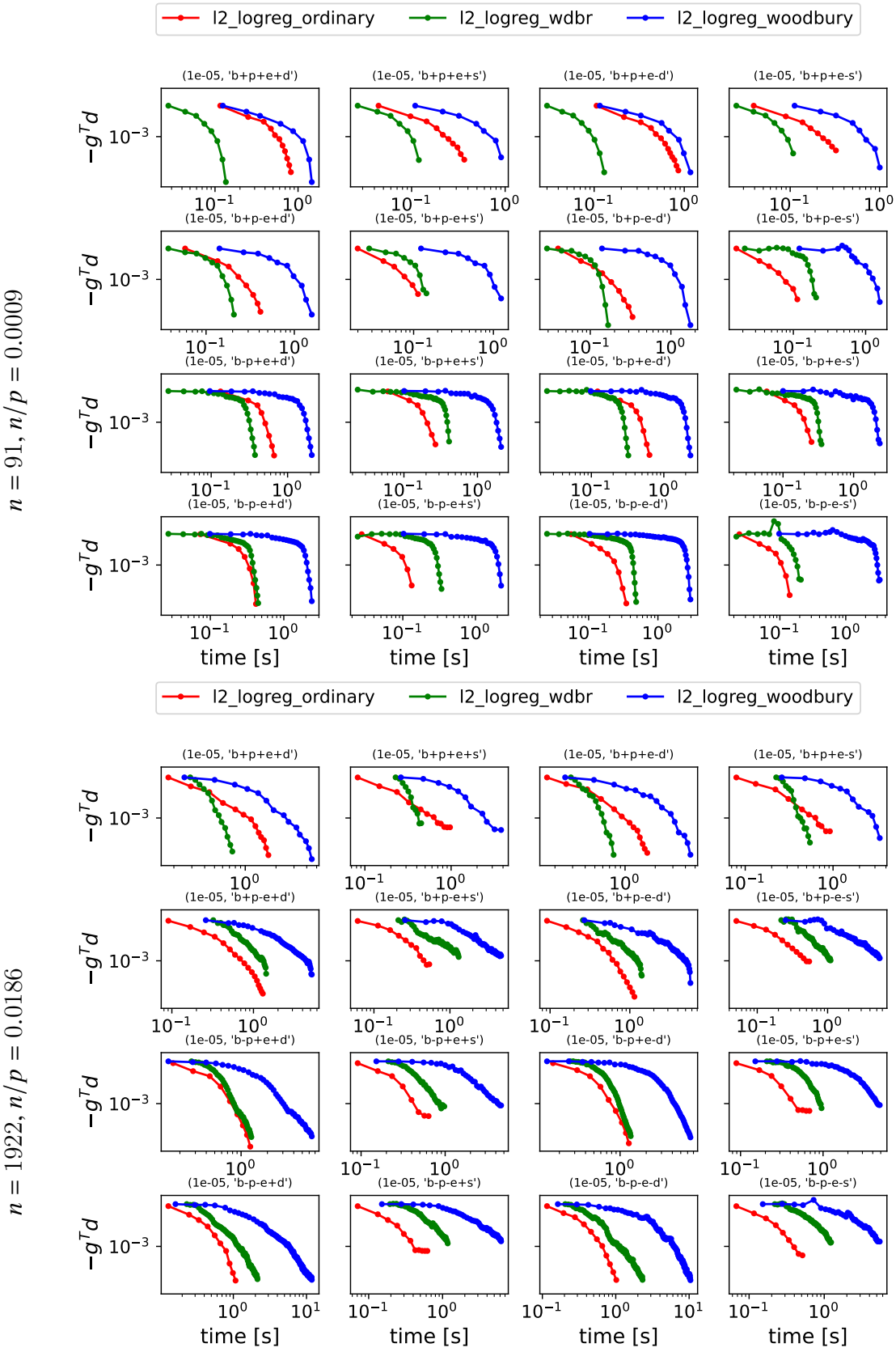
Rysunek 4.3: Porównanie czasów strojenia modeli w funkcji n/p. Dane gęste: `PCam16k`.



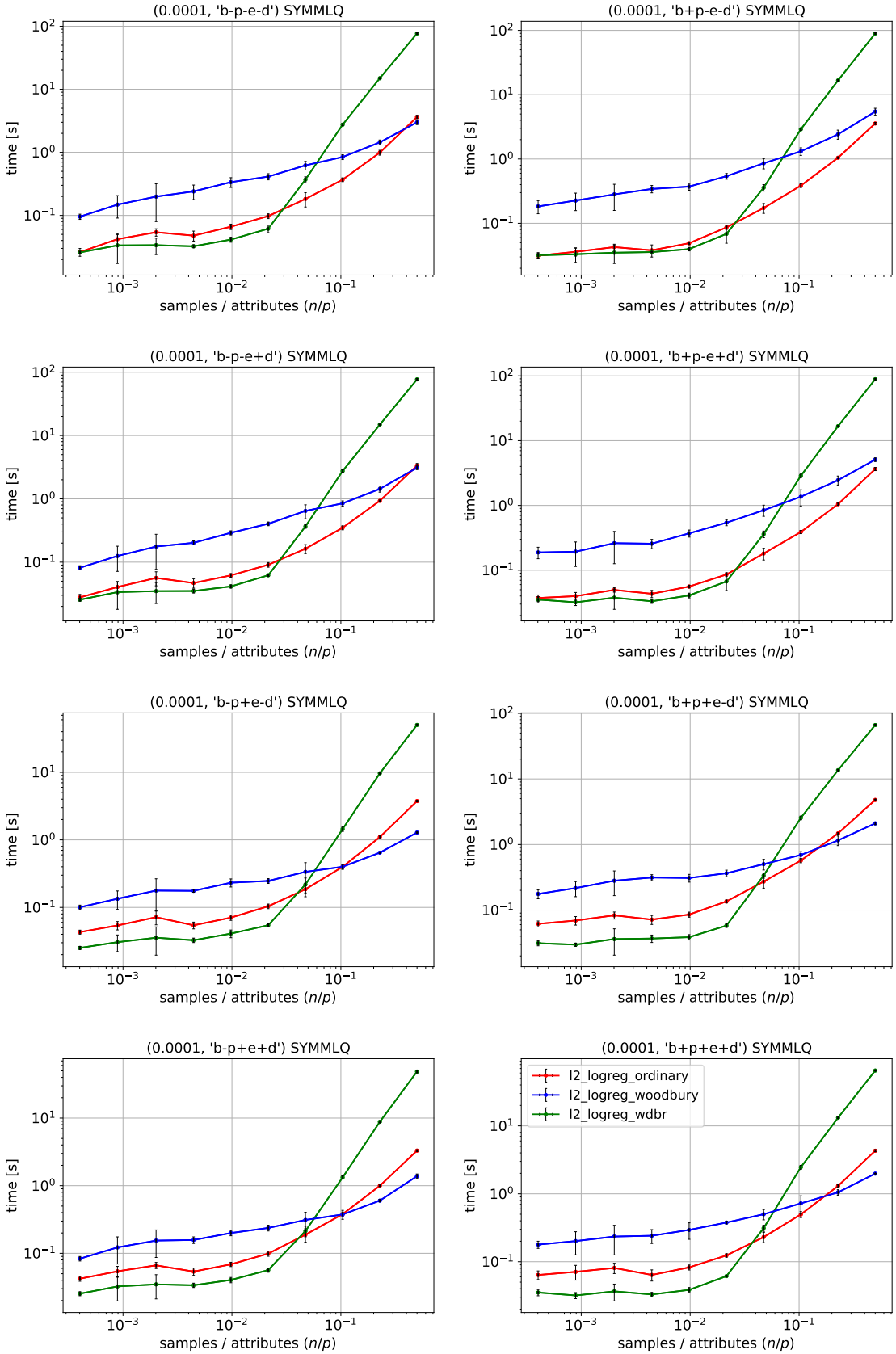
Rysunek 4.4: Porównanie zbieżności procedur dla różnej liczby próbek. Dane: `PCam16k`.



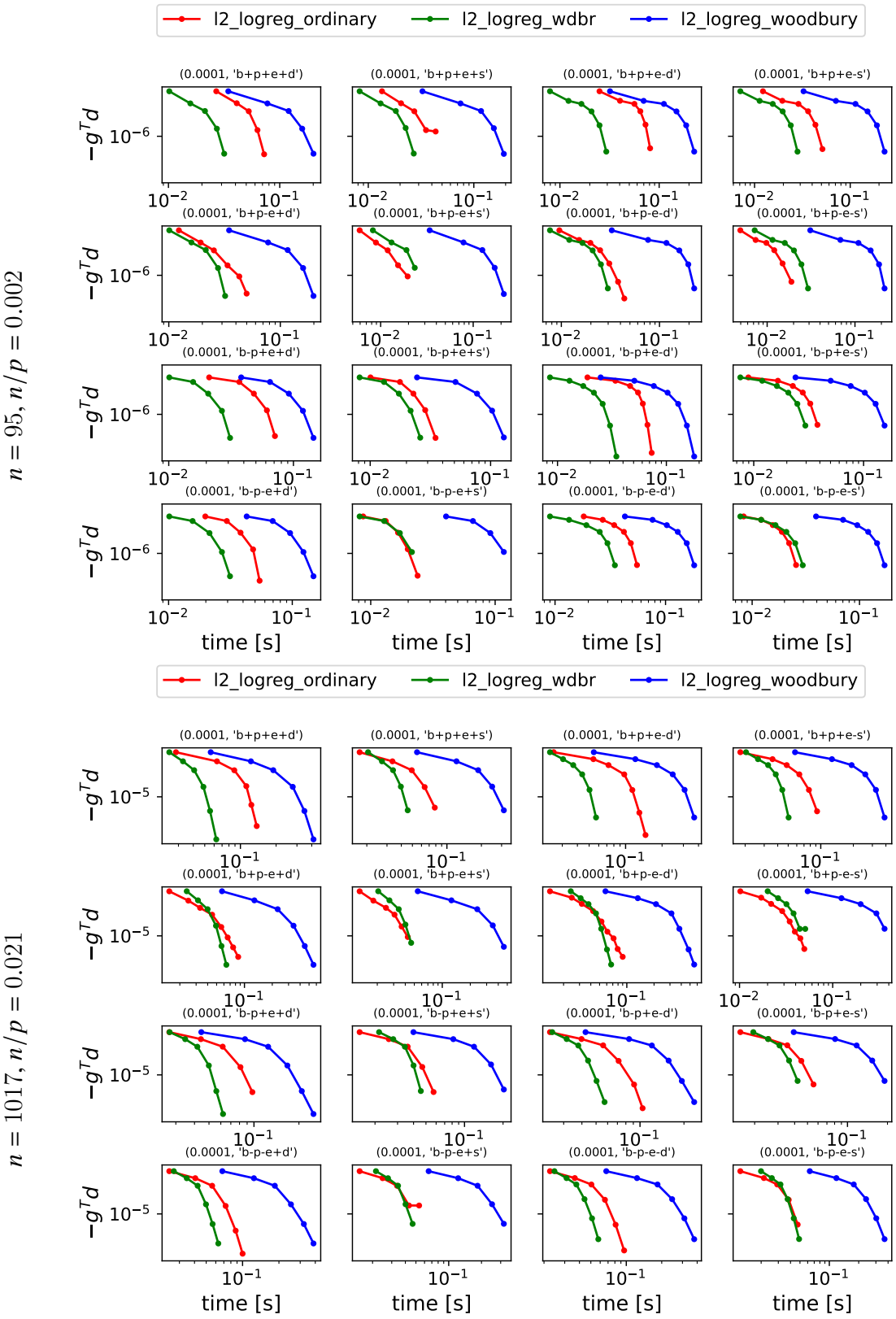
Rysunek 4.5: Porównanie czasów strojenia modeli w funkcji n/p. Dane rzadkie: `news20`.



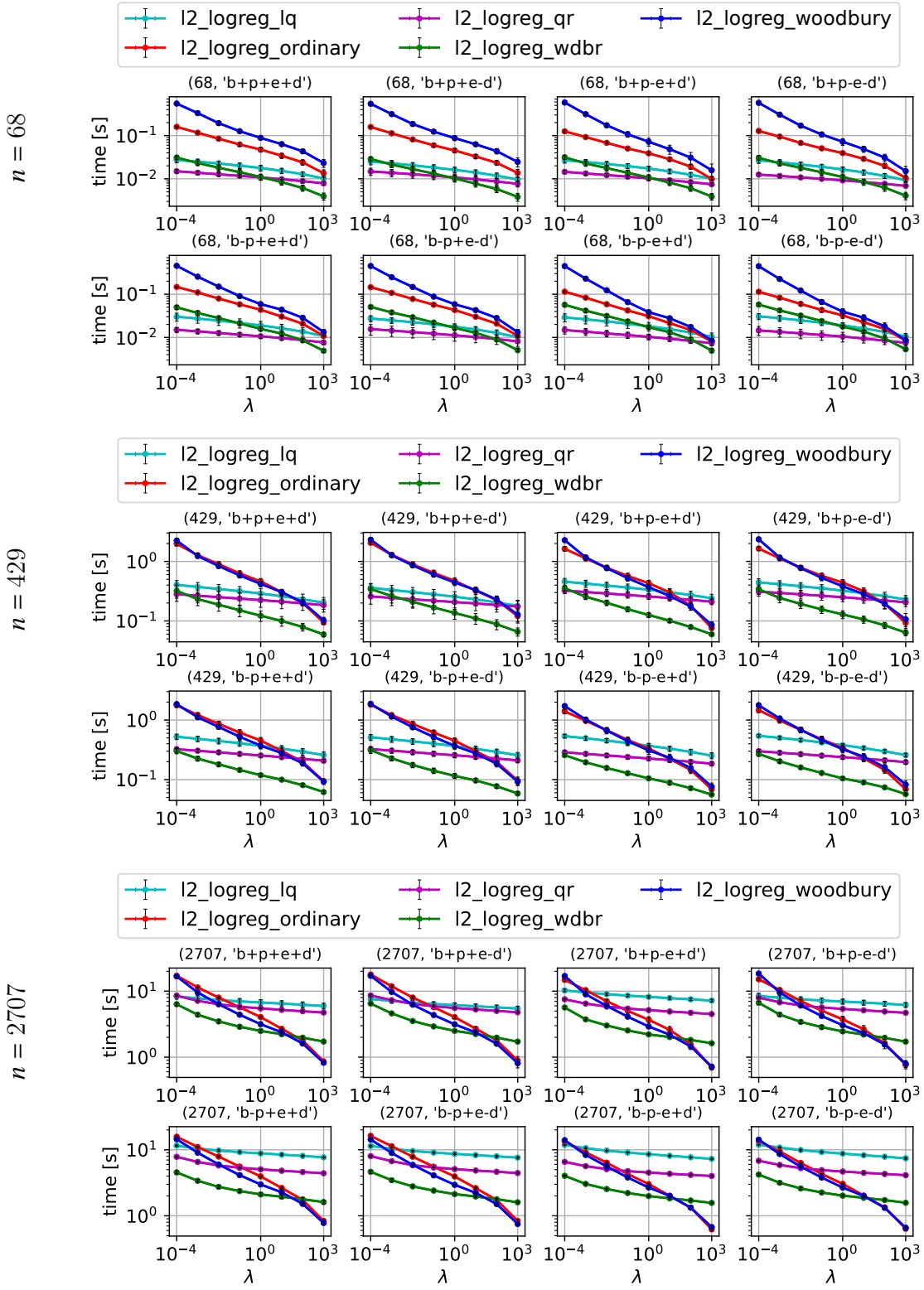
Rysunek 4.6: Porównanie zbieżności procedur dla różnej liczby próbek. Dane: `news20`.



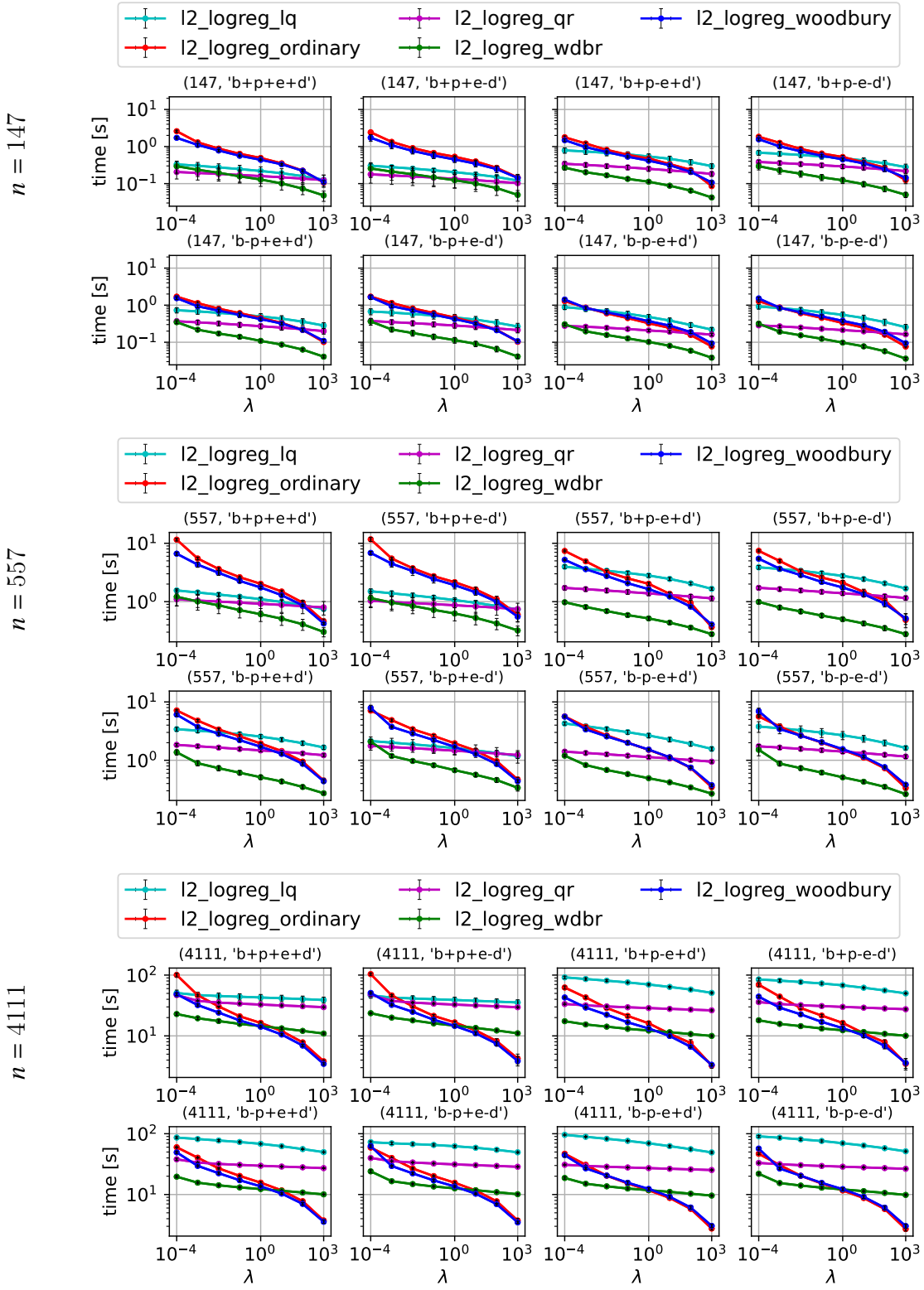
Rysunek 4.7: Porównanie czasów strojenia modeli w funkcji n/p. Dane rzadkie: RCV1.



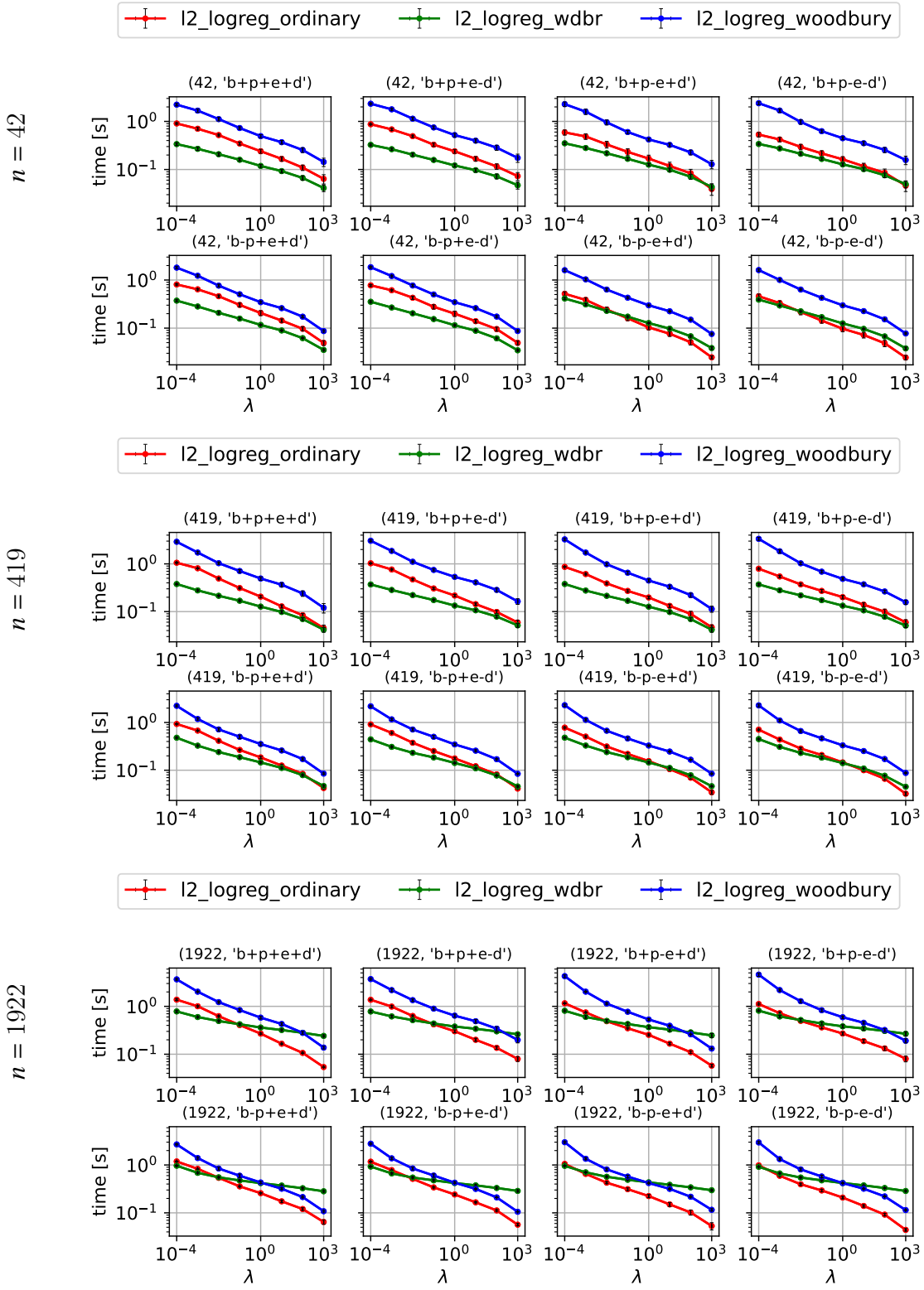
Rysunek 4.8: Porównanie zbieżności procedur dla różnej liczby próbek. Dane: RCV1.



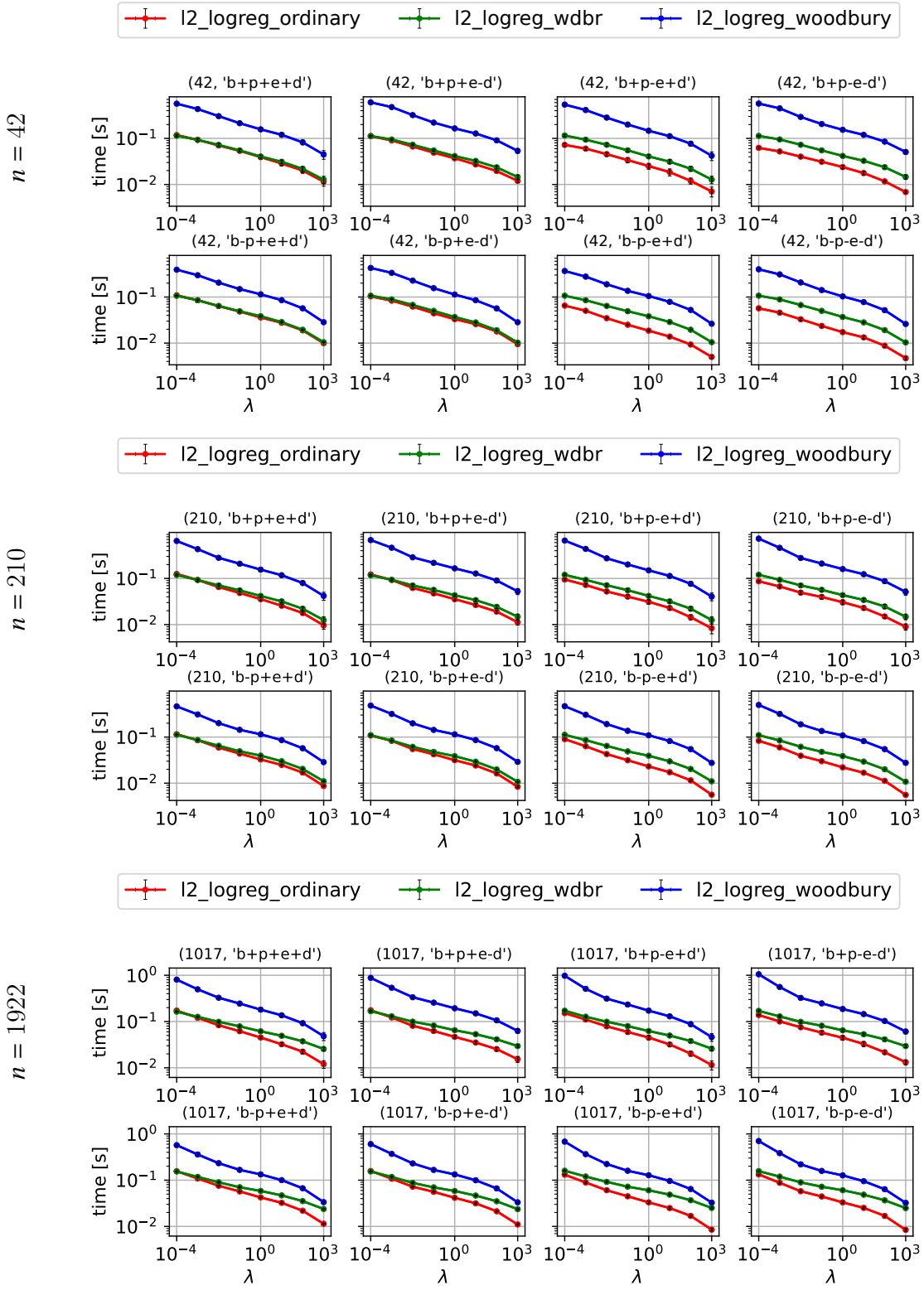
Rysunek 4.9: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od wag modelu poprzedniego. Liczba próbek kolejno od góry: $n = 68, 429, 2707$. Wykres przedstawia skumulowany czas obliczeń. Dane: `extreme`.



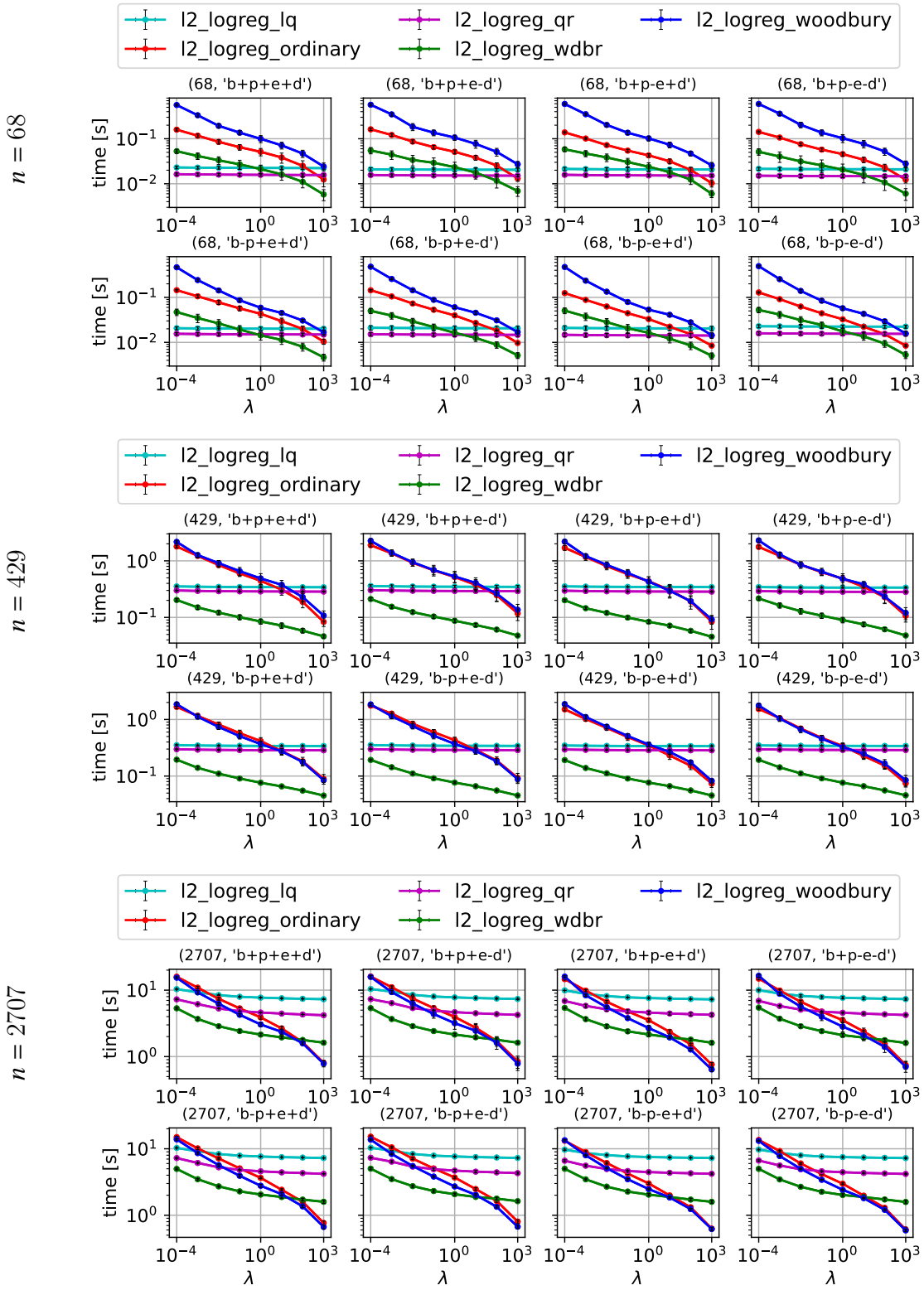
Rysunek 4.10: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od wag modelu poprzedniego. Liczba próbek kolejno od góry: $n = 147, 557, 4111$. Wykres przedstawia skumulowany czas obliczeń. Dane: `PCam16k`.



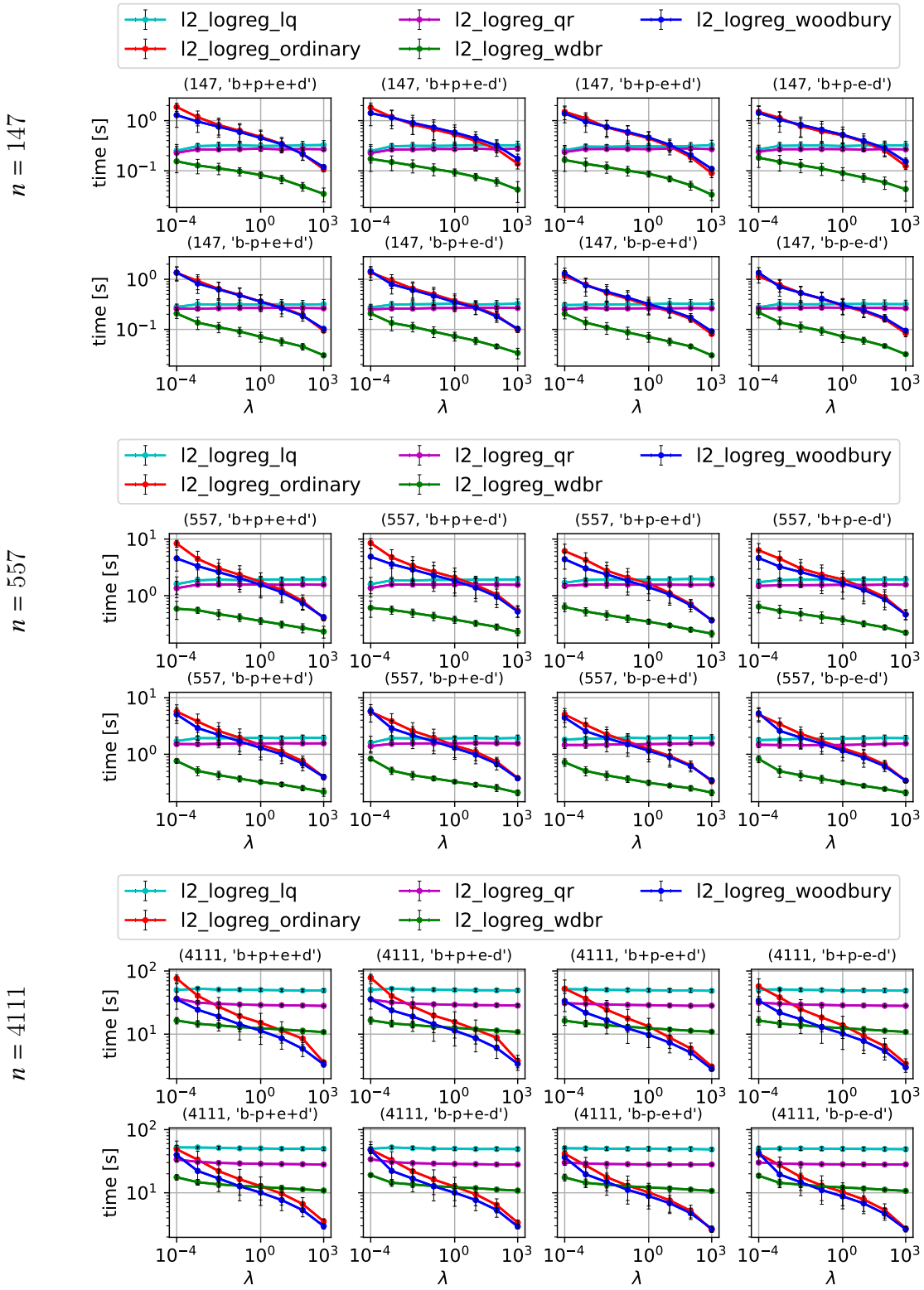
Rysunek 4.11: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od wag modelu poprzedniego. Liczba próbek kolejno od góry: $n = 42, 419, 1922$. Wykres przedstawia skumulowany czas obliczeń. Dane: `news20`.



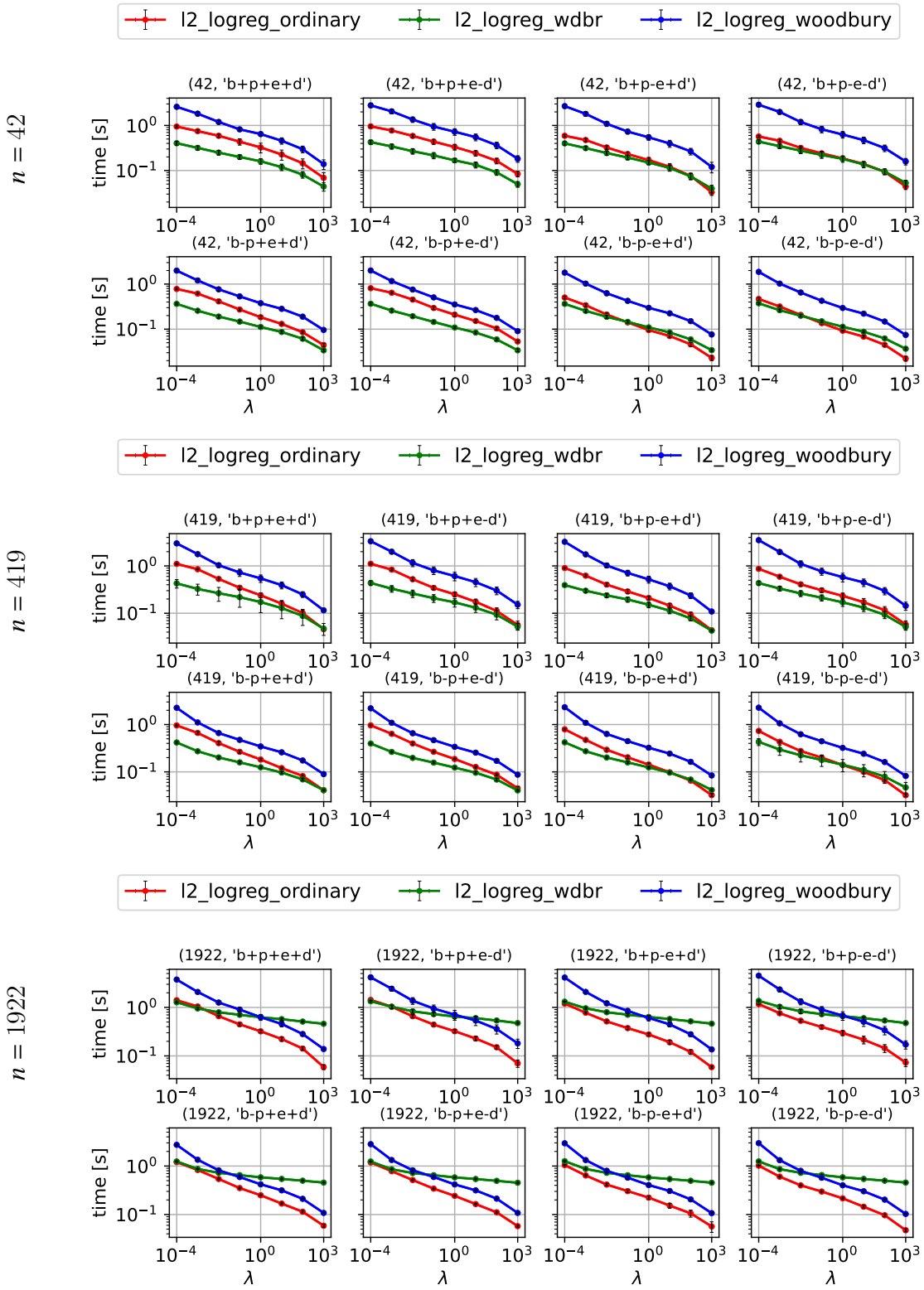
Rysunek 4.12: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od wag modelu poprzedniego. Liczba próbek kolejno od góry: $n = 42, 210, 1922$. Wykres przedstawia skumulowany czas obliczeń. Dane: RCV1.



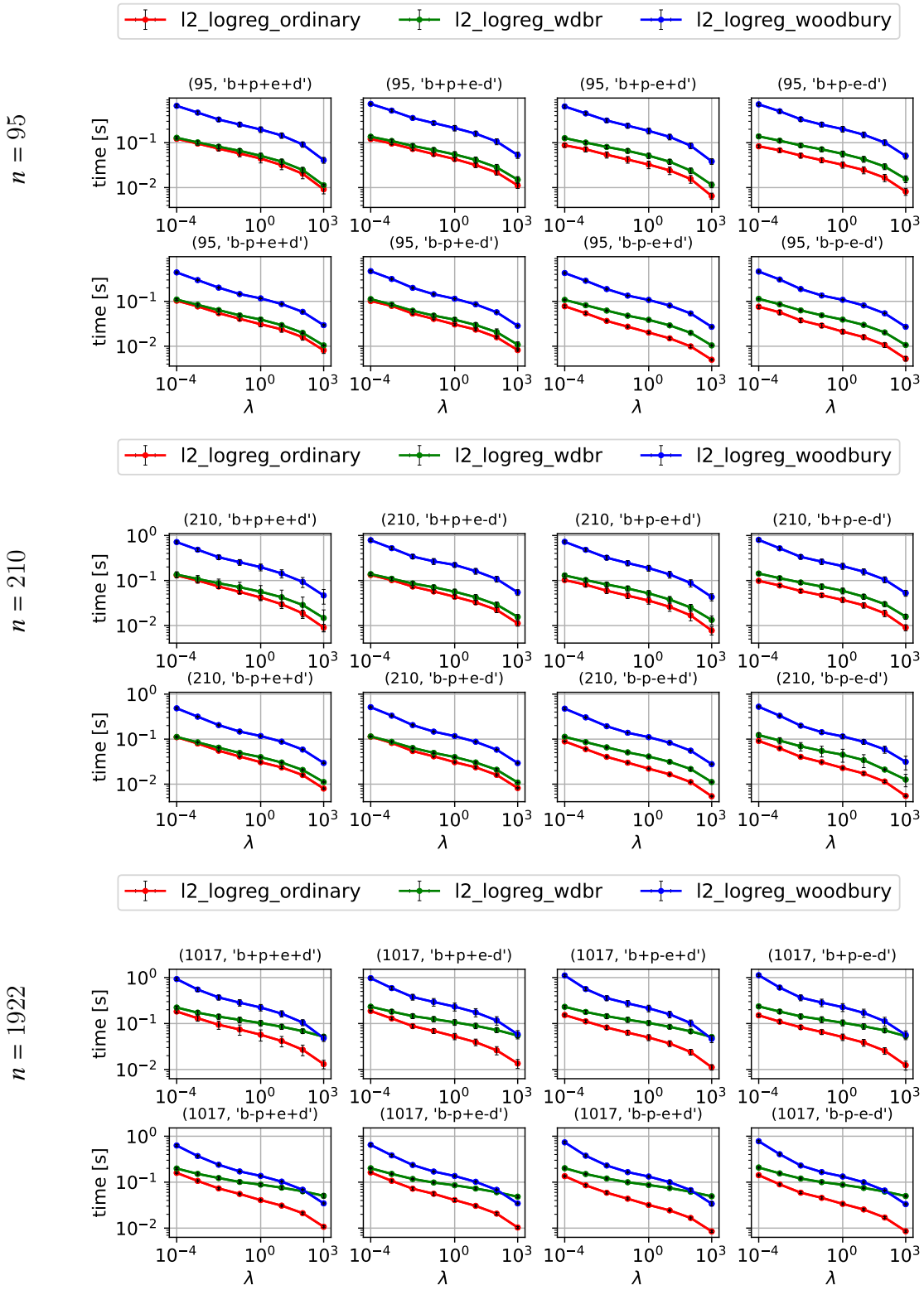
Rysunek 4.13: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od $\boldsymbol{w} = \mathbf{0}$. Liczba próbek kolejno od góry: $n = 68, 429, 2707$. Dane: `extreme`.



Rysunek 4.14: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od $\boldsymbol{w} = \mathbf{0}$. Liczba próbek kolejno od góry: $n = 147, 557, 4111$. Dane: `PCam16k`.



Rysunek 4.15: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od $\boldsymbol{w} = \mathbf{0}$. Liczba próbek kolejno od góry: $n = 42, 419, 1922$. Dane: `news20`.



Rysunek 4.16: Porównanie czasów wyznaczania ścieżki dla $\lambda = 10^3, 10^2, \ldots, 10^{-4}$; kolejny model startuje od $\boldsymbol{w} = \mathbf{0}$. Liczba próbek kolejno od góry: $n = 95, 210, 1922$. Dane: RCV1.

4.6 Wyniki eksperymentów dla regularyzacji ℓ^1

Eksperymenty z modelami z regularyzacją ℓ^1 wykonano w identycznym układzie, jak w przypadku opisanym we wcześniejszej sekcji. W eksperymentach porównywano wybrane procedury opisane w rozdziale 2 w sekcji 2.5, skrótowo podsumowane w tabeli 4.13 oraz wykorzystano te same zbiory danych.

Tabela 4.13: Procedury wykorzystane w eksperymentach z uczeniem modeli regresji logistycznej z regularyzacją ℓ^1.

Procedura	Opis
`l1_logreg_ipm`	metoda Newtona z ograniczeniami, rozwiązywany za pomocą metody wewnętrznego punktu (por. sekcja 2.5.3)
`l1_logreg_liblinear`	metoda spadku względem współrzędnych, procedura zaczerpnięta z pakietu `liblinear` (por. sekcja 2.5.2)
`l1_logreg_cd`	metoda spadku względem współrzędnych, własna implementacja w C++, nieznaczne różnice względem `liblinear` (por. sekcja 2.5.2)
`l1_logreg_lbfgsb`	metoda z wykorzystaniem gładkiej optymalizacji z ograniczeniami za pomocą algorytmu `L-BFGS-B` (por. sekcja 2.5.3)
`l1_logreg_ordinary`	metoda wykorzystująca górne ograniczenie normy ℓ^1 przez normę ℓ^2, kierunek znajdowany przez rozwiązywanie układu (2.57)
`l1_logreg_woodbury`	podobnie jak wyżej, kierunek znajdowany przez rozwiązywanie układu (2.58)
`elnet_lr_lbfgsb`	zaimplementowana metoda elastic net z wykorzystaniem gładkiej optymalizacji z ograniczeniami za pomocą algorytmu `L-BFGS-B` (por. sekcja 2.6), w eksperymentach z wyzerowaną karą typu ℓ^2 — przy tych ustawieniach model równoważny modelowi `l1_logreg_lbfgsb`

Tabela 4.14: Porównanie zbieżności testowanych algorytmów dla regularyzacji ℓ^1 na zbiorze `lsvt` (por. repozytorium OpenML zbiór `id=1484`) o rozmiarze $n \times p = 126 \times 309$ przy ustawieniach $\lambda = 10^{-4}$ oraz ustawieniach: `b-p-e-d` (bez wyrazu wolnego, bez preconditioningu, bez dokładnej minimalizacji kierunkowej w typie double); `it` onzacza numer iteracji, `gTd` rzut gradienu na nowy kierunek (pochodna kierunkowa), `f` wartość funkcji celu, `cg` liczba iteracji procedury rozwiązującej układ równań liniowych.

	`l1_logreg_ordinary`			`l1_logreg_woodbury`		
it	gTd	f	cg	gTd	f	cg
1	-0.170364	84.1824	1	-0.170364	84.1824	2
2	-0.0719646	84.1045	1	-0.0719593	84.1045	1
3	-0.0189374	84.0836	1	-0.0189363	84.0836	2
4	-0.0167302	84.0731	1	-0.0167284	84.0731	2
5	-0.00174617	84.0713	1	-0.00174603	84.0713	2
6	-0.00162302	84.0703	1	-0.00162284	84.0703	2
7	-0.000161771	84.0701	1	-0.000161757	84.0701	2
8	-6.70121e-05	84.07	1	-6.70074e-05	84.07	2
9	-5.8941e-05	84.07	1	-5.89344e-05	84.07	2
10	-6.1486e-06	84.07	1	-6.14812e-06	84.07	2
11	-5.74207e-06	84.07	1	-5.74141e-06	84.07	2
12	-5.70253e-07	84.07	1	-5.70205e-07	84.07	2
13	-2.35855e-07	84.07	1	-2.35838e-07	84.07	2
14	-2.08508e-07	84.07	1	-2.08484e-07	84.07	2
15	-2.16595e-08	84.07	1	-2.16578e-08	84.07	2
16	-2.0317e-08	84.07	1	-2.03147e-08	84.07	2
17	-2.01045e-09	84.07	1	-2.01028e-09	84.07	2
Wagi	$W_{186} = 2.73 \cdot 10^{-10}, W_{187} = 2.82 \cdot 10^{-10},$ $W_{188} = 0.569745$			$W_{186} = 2.73 \cdot 10^{-10}, W_{187} = 2.82 \cdot 10^{-10},$ $W_{188} = 0.569745$		

	`l1_logreg_lbfsgb`			`l1_logreg_liblinear`			`l1_logreg_cd`		
it	gTd	f	cg	gTd	f	cg	gTd	f	cg
1	-0.0932117	84.4821	3	-4.58855	-	1	-5.00323	85.5457	1
2	-0.114826	84.3672	1	-1.89621	-	3	-1.47852	84.07	3
3	-0.290918	84.0741	1	-1.88509e-08	-	1	-9.16914e-09	84.07	1
4	-0.00189289	84.0717	1						
5	-6.23783e-06	84.07	1						
Wagi	$W_{188} = 0.569721$			$W_{188} = 0.569745$			$W_{188} = 0.569745$		

Ponownie przed uruchomieniem eksperymentów zweryfikowano empirycznie poprawność algorytmów. Wyniki przykładowego eksperymentu porównawczego przedstawiono w tabeli 4.14. Tabela zawiera wyniki uzyskane na zbiorze `lsvt` (por. repozytorium OpenML zbiór `id=1484`) o rozmiarze $n \times p = 126 \times 309$ przy ustawieniach $\lambda = 10^{-4}$ oraz identycznych pozostałych ustawieniach. Z prezentowanych algorytmów porównywalne są dwa algorytmy bazujące na spadku względem współrzędnych, tj. `l1_logreg_liblinear` i `l1_logreg_cd`. W tym przypadku rozwiązanie posiada tylko jeden niezerowy współczynnik W_{188}, pozostałe zmienne są wyzerowane. Oba algorytmy dały dokładnie taki sam wynik z dokładnością do wyświetlanej precyzji. Bardzo podobny wynik (z dokładnością do 4 cyfr znaczących) oraz podobną liczbę iteracji wykonał algorytm `l1_logreg_lbfgsb`. Wszystkie z nich startowały z punktu początkowego $\boldsymbol{w} = \mathbf{0}$.

Dwa pozostałe algorytmy (`l1_logreg_woodbury` i `l1_logreg_ordinary`) zatrzymały się na takiej samej wartości funkcji celu, natomiast dodatkowe dwa współczynniki W_{186} i W_{187} okazały się niezerowe, chociaż ich wartość jest właściwie na poziomie zera numerycznego rzędu 10^{-10} — gdyby algorytmy działały dłużej, to te dwa współczynniki zostałyby wyzerowane. Oba algorytmy są równoważne (co można zauważyć śledząc iteracje w tabeli 4.14). Te algorytmy startowały z punktu początkowego $\boldsymbol{w} = \boldsymbol{d} \cdot \arg\min_t Q(t\boldsymbol{d})$, gdzie $\boldsymbol{d}$ to antysubgradient policzony w zerze. Długa ścieżka uczenia już na wstępie dyskwalifikuje te dwa podejścia, gdyż dla dużo większych zbiorów danych czas znacząco się wydłuży.

Pozostałe algorytmy, pomimo zbieżności do tej samej wartości funkcji celu oraz dawania numerycznie identycznych rozwiązań, nie są równoważne (jeżeli chodzi sposób działania i iteracje).

Na rysunku 4.17 przedstawiono czasy wykonania poszczególnych algorytmów wraz ze wzrostem liczby próbek, uśrednione po dziesięciu powtórzeniach z losowym wyborem zestawu uczącego. Na wykresach wąsami zaznaczono również odchylenia standardowe wyników. Na wykresach na osi OX przedstawiono unormowaną liczbę próbek n/p. Wyniki przedstawiono tylko dla typu `double`. Metoda `l1_logreg_ipm` wspiera jedynie konfigurację `b+p+e-d`, natomiast dwie pozostałe metody mogą uczyć modele z wyrazem wolnym lub bez niego, korzystając z przybliżonej minimalizacji kierunkowej (`b-`) i bez możliwości wykorzystania preconditioningu (`p-`).

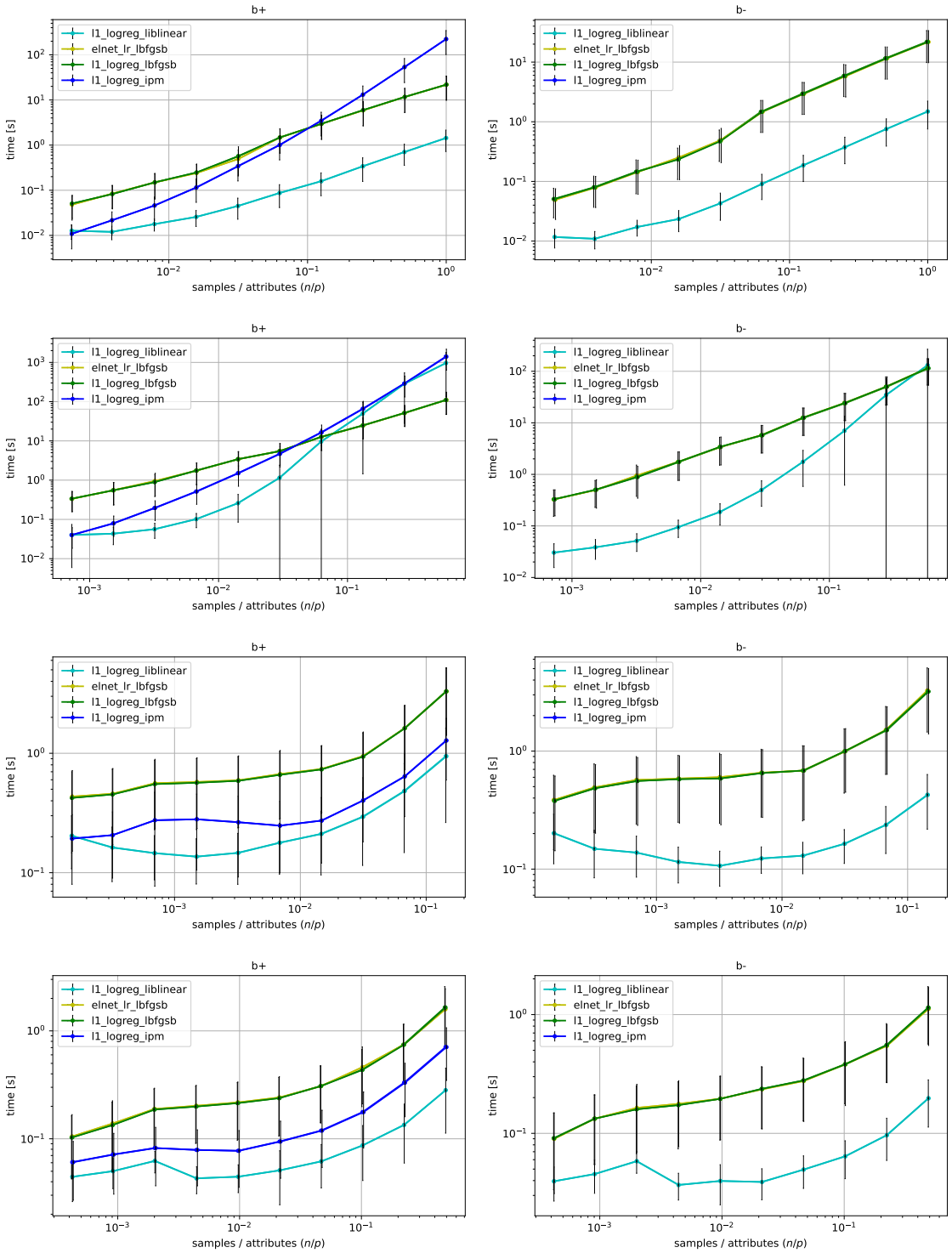
Na rysunkach 4.18, 4.19, 4.20, 4.21 zwizualizowano tempo zbieżności algorytmów dla wybranych rozmiarów danych uczących (n) i parametru $\lambda = 10^{-2} \cdot \lambda_{\max}$, gdzie $\lambda_{\max}$ to najmniejsza wartość, dla której wagi w modelu wynikowym wynoszą zero (z wyjątkiem wyrazu wolnego, który nie podlega regularyzacji). Wyniki przedstawiono dla wszyst-

kich dwóch wariantów algorytmów (z wyrazem wolnym i bez wyrazu wolnego). Każda z metod inaczej podchodzi do rozwiązania problemu, zatem mają różny koszt związany z wyznaczaniem nowego punktu. Kroki w metodzie spadku względem współrzędnych są szybkie, jednak trzeba ich wykonać więcej, by uzyskać satysfakcjonujący wynik. Z kolei pojedyncza iteracja w metodzie wewnętrznego punktu wymaga większego nakładu pracy, jednakże prowadzi to do szybszej zbieżności. Metoda oparta na algorytmie `L-BFGS-B` balansuje między tymi dwiema metodami pod kątem kosztu pojedynczej iteracji.

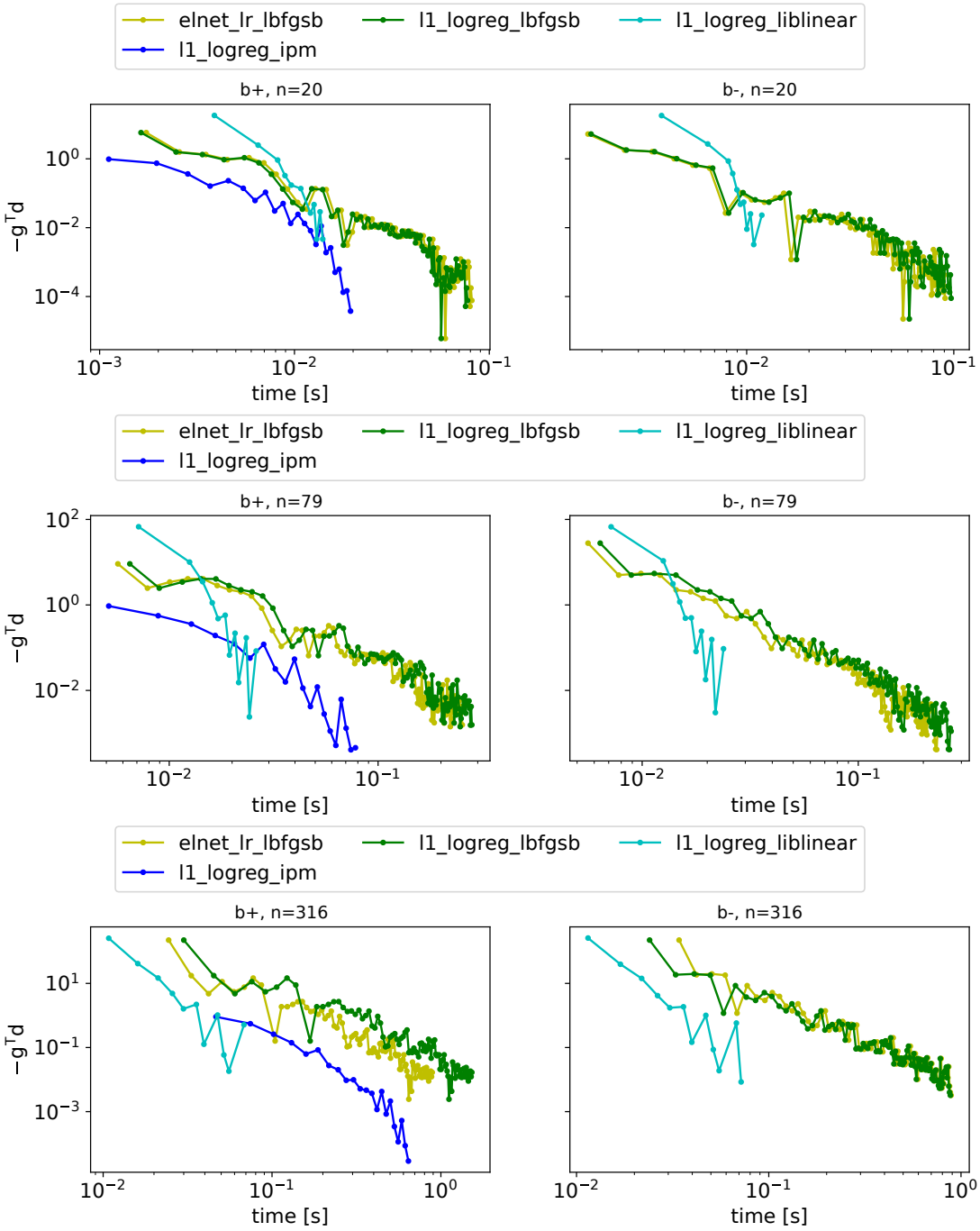
Patrząc na rysunki 4.18, 4.19, 4.20, 4.21 widać, że dla dostatecznie małolicznych zbiorów `l1_logreg_ipm` może być konkurencyjny w stosunku do `l1_logreg_liblinear`. Ogólnie różnica między tymi algorytmami (zwłaszcza na zbiorach rzadkich) nie jest duża (nieznacznie na korzyść `l1_logreg_liblinear`), natomiast dla większych wartości n/p na zbiorach gęstych staje się ona wyraźniejsza. Rysunek 4.17 nie oddaje dokładnie różnic miedzy algorytmami, ponieważ porównywane metody różnie definiują warunek stopu i zatrzymanie wspomnianych algorytmów następuje często później (bliżej optimum, przy niższej wartości funkcji celu), natomiast dla algorytmu `l1_logreg_liblinear` ustawiony jest on domyślnie wcześniej. Skutkuje to również tym, że algorytmy te zatrzymują się czasami w różniących się nieznacznie punktach końcowych (jeżeli chodzi o rzadkość modeli).

Patrząc na rysunek 4.17 można zauważyć, że dla zbioru gęstego `PCam16k` uczenie modelu z wyrazem wolnym oparte na algorytmie `L-BFGS-B` okazało się efektywniejsze dla większej próby (przy stosunku liczby próbek do liczby atrybutów na poziomie 10%). Zysk wytłumaczyć można następująco — przy większej liczbie próbek zależności między atrybutami (korelacje) w tym zbiorze zaczynają się ujawniać, przez co optymalizacja parametrów pojedynczo nie prowadzi do szybkiej zbieżności. Obniżając mocniej współczynnik λ spodziewać należy się wydłużenia czasu uczenia w metodzie opartej na spadku względem współrzędnych

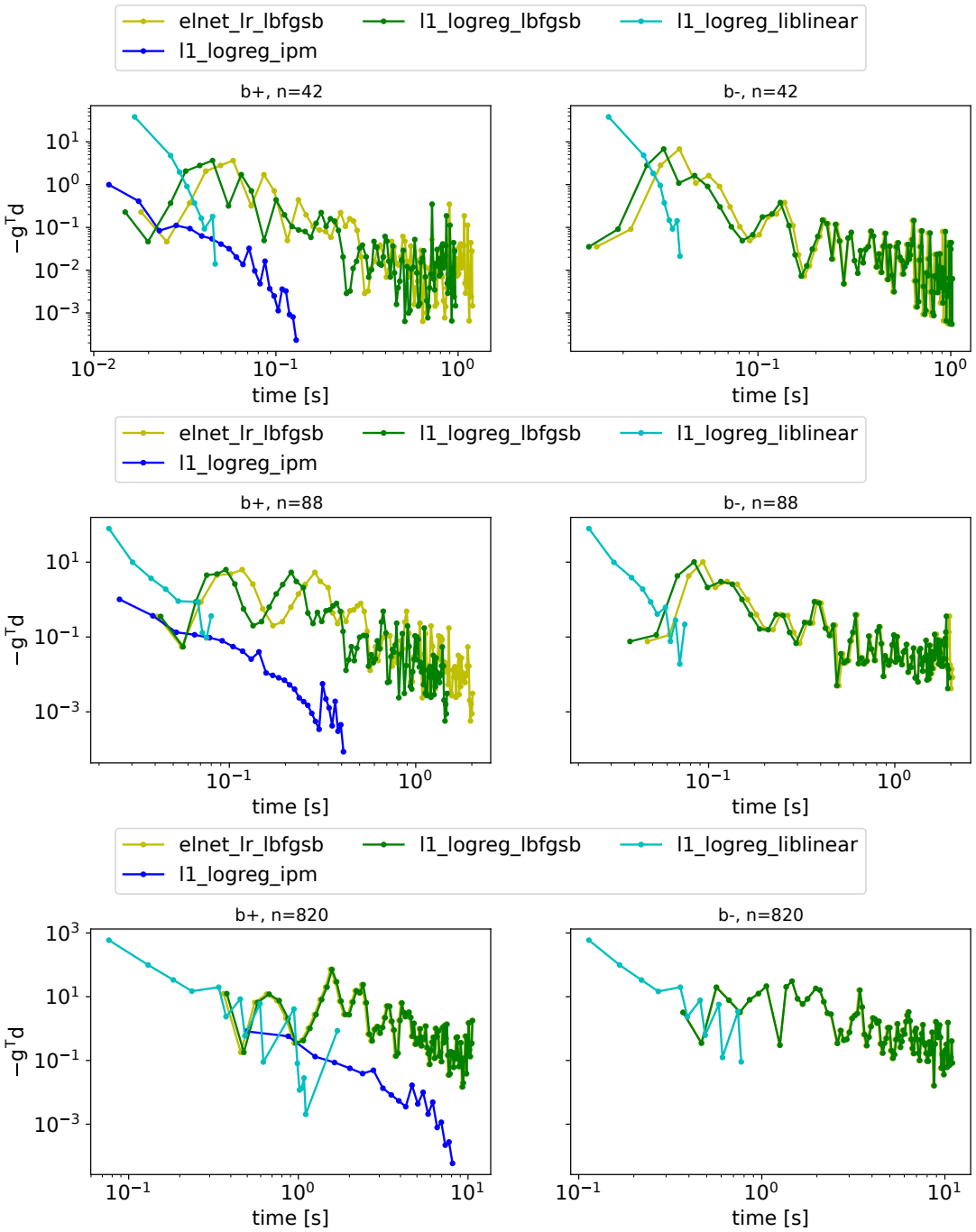
Z rysunku można odczytać także, że dla rzadkich zbiorów danych metoda `IPM` radzi sobie nieznacznie gorzej względem metody spadku względem współrzędnych — przy mniejszej wartości λ powinien być obserwowany zysk w stosunku do metody spadku względem współrzędnych.



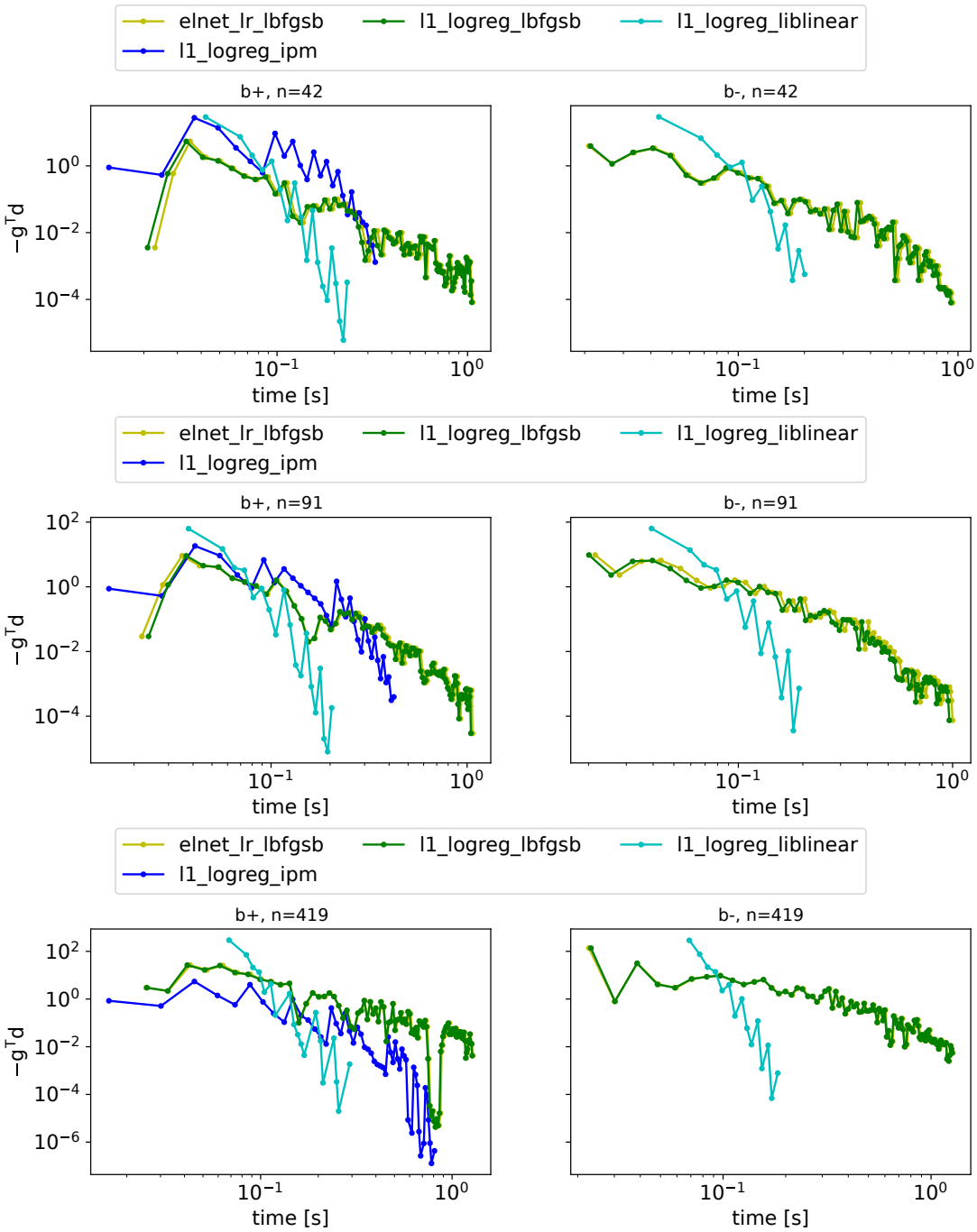
Rysunek 4.17: Porównanie czasów strojenia modeli w funkcji n/p. Kolejność zbiorów danych od góry: `extreme`, `PCam16k`, `news20`, `RCV1`.



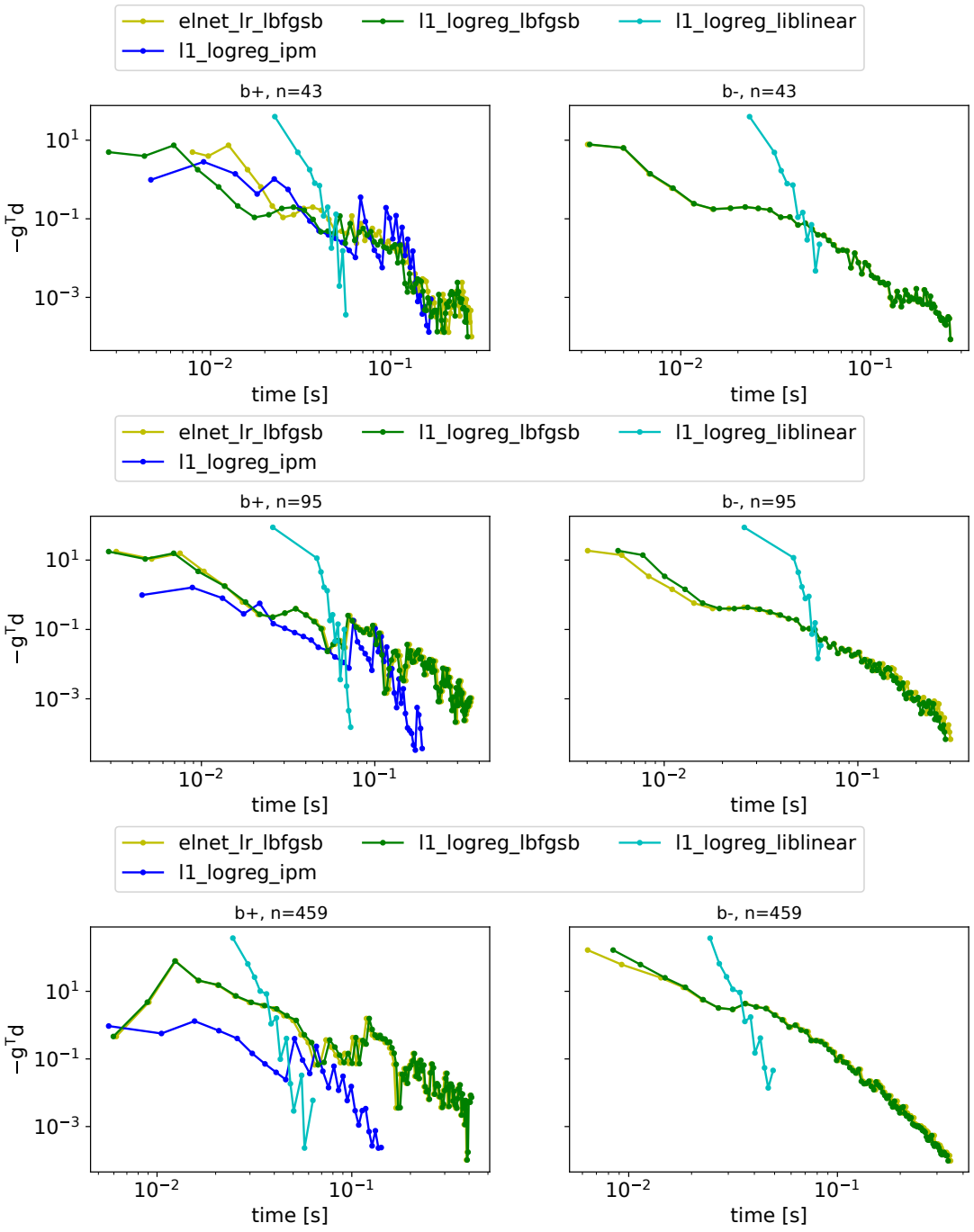
Rysunek 4.18: Porównanie zbieżności modeli na zbiorze `extreme` dla różnych rozmiarów danych. Lewa kolumna to model z wyrazem wolnym, a prawa kolumna to model bez wyrazu wolnego.



Rysunek 4.19: Porównanie zbieżności modeli na zbiorze PCam16k. Lewa kolumna to model z wyrazem wolnym, a prawa kolumna to model bez wyrazu wolnego.



Rysunek 4.20: Porównanie zbieżności modeli na zbiorze `news20`. Lewa kolumna to model z wyrazem wolnym, a prawa kolumna to model bez wyrazu wolnego.



Rysunek 4.21: Porównanie zbieżności modeli na zbiorze RCV1. Lewa kolumna to model z wyrazem wolnym, a prawa kolumna to model bez wyrazu wolnego.

5. Zastosowanie regularyzacji w modelach nieliniowych

5.1 Rzadki autoenkoder

5.1.1 Przygotowanie i opis eksperymentu

Model dwuwarstwowej sieci neuronowej uczonej z regularyzacją ℓ^1 o strukturze autoenkodera opisany w sekcji 3.2 przetestowano na zbiorach: `Gasoline`, `MNIST` oraz `Olivetti faces`. Eksperymenty przeprowadzono w środowisku `Python/scikit-learn`.

Głównym modelem odniesienia jest procedura `SparsePCA` z pakietu `scikit-learn` [61] w języku `Python`. Własna oryginalna struktura rzadkiego autoenkodera w eksperymentach oznaczona jako `neural_network` została zaimplementowana w języku `C++` i podpięta do środowiska `Python`, w którym przeprowadzono całość eksperymentów. Jako drugą metodę referencyjną wykorzystano metodę PCA, która jest oczywiście najszybszą z rozważanych metod, jednak generuje rozwiązania jakościowo znacznie odbiegające od pozostałych.

W eksperymencie porównywano czas obliczeń i jakość uzyskiwanych wyników (głównie rzadkość rozwiązania oraz dokładność odtworzenia) dla różnych parametrów regularyzacji α. W tym eksperymencie liczba dopasowanych komponentów (w przypadku sieci — neuronów warstwy ukrytej) została ustawiona arbitralnie na 50 dla `faces` i `MNIST` oraz na 10 dla `gasoline`. Metoda `SparsePCA` wykorzystuje odmienną technikę uczenia

Tabela 5.1: Podsumowanie parametrów zbiorów danych wykorzystanych w eksperymentach.

Dane	**l. próbek (n)**	**liczba atrbutów (p)**	**Rozstęp atrybutów**
Gasoline	60	401	[-0.084, 1.33]
MNIST	60000	784	[0, 255]
Olivetti faces	400	4096	[0, 1]

(uczenie słownikowe), przez co znaczenie parametru regularyzacyjnego α jest inne niż w proponowanej procedurze, stąd zamiast wartości parametru regularyzacyjnego punktem odniesienia w porównaniu wyników będzie rzadkość rozwiązania.

W eksperymentach porównawczych wykorzystano następujące zestawy danych:

1. `gasoline` — ten zestaw danych pochodzi z pakietu **pls** [41] języka `R` i zawiera 60 sygnałów widmowych bliskiej podczerwieni dla próbek benzyny, opisanych przez 401 atrybutów, odpowiadających długościom fal od 900 nm do 1700 nm. W zbiorze tym celem jest predykcja liczby oktanowej paliwa.
2. `MNIST` — zestaw danych MNIST [48] zawierający odręcznie pisane cyfry 0–9, który zawiera 60 000 obrazów treningowych i 10 000 obrazów testowych zapisanych na matrycach 28×28, co daje łącznie 784 atrybuty.
3. `Olivetti faces` — zestaw danych [12] pobrany z pakietu `scikit-learn` [61] zawiera 400 obrazów twarzy — 10 różnych obrazów dla każdej klasy. Obrazy znajdują się w macierzach o wymiarach 64×64, co daje łącznie 4096 atrybutów. Obrazy zostały wykonane w różnych warunkach, przy różnym oświetleniu, mimice i szczegółach twarzy (np. obecność okularów).

Dla wszystkich modeli przy walidacji wyników użyto osobnych zestawów testowych. Zbiór `MNIST` zawiera oryginalny zestaw testowy, pozostałe zestawy danych podzielono na część uczącą i testową w sposób losowy za pomocą funkcji `train_test_split` (z pakietu `scikit-learn`) w proporcji 1:1. Liczba iteracji i tolerancja dla algorytmów uczących zostały ustawione arbitralnie odpowiednio jako 1000 oraz 10^{-3}.

Oba rozwiązania były testowane w tych samych warunkach w modelu jednowątkowym. Jedyną równoległą częścią była n-krotna walidacja z jednym wątkiem na iterację. Eksperymenty zostały przeprowadzone na maszynie z procesorem Xeon E5-2699 v4 2,20 GHz i 128 GB pamięci RAM. Wykorzystano następującą procedurę testowania dopasowania:

```
for trial_num in [1, ..., number_of_trials]:
    x_train, x_test = train_test_split(X, test_size=0.5)
    for model in [SparsePCA, neural_network]:
        for alpha in list_of_alphas:
            model.fit(x_train)
            mse_train[trial_num] = score(model, x_train)
            mse_test[trial_num] = score(model, x_test)
score_train = mean(mse_train)
score_test = mean(mse_test)
```

Tabela 5.2: Szczegółowe porównanie proponowanego autoenkodera (`neural_network`) z procedurą `SparsePCA/scikit-learn` oraz standardową procedurą `PCA`. Rzadkość oznacza średnią liczbę zerowych elementów na komponent.

Data	neural-network					SparsePCA/scikit-learn					PCA		
	α	MSE train	MSE test	time	sparsity	α	MSE train	MSE test	time	sparsity	MSE train	MSE test	time
Gasoline	1.4e − 5	4.4e − 6	7.3e − 6	0.073	0.09	2.3e − 4	1.2e − 6	1.1e − 5	10.57	65.65			
	2.7e − 5	4.5e − 6	7.3e − 6	0.071	0.62	5.2e − 4	1.3e − 6	1.2e − 5	11.08	86.63			
	5.2e − 5	7.4e − 6	1.2e − 5	0.469	115.1	0.001	1.6e − 6	1.3e − 5	7.915	121.3	0.016	0.016	0.064
	1.0e − 4	1.7e − 5	2.4e − 5	1.205	339.7	0.003	2.4e − 6	1.4e − 5	8.058	174.5			
	1.9e − 4	2.3e − 5	3.0e − 5	1.268	374.8	0.006	4.6e − 6	2.1e − 5	7.459	225.9			
Olivetti faces	1.0e − 4	2.0e − 3	3.8e − 3	392.6	564.8	0.014	2.0e − 3	3.9e − 3	163.5	344.0			
	3.7e − 4	2.0e − 3	3.9e − 3	281.5	2212	0.032	2.3e − 3	3.8e − 3	2699	1332			
	7.2e − 4	2.1e − 3	4.0e − 3	268.3	2927	0.072	2.8e − 3	3.9e − 3	1542	2190	6.4e − 3	8.2e − 3	2.118
	1.4e − 3	2.3e − 3	4.1e − 3	248.4	3371	0.164	3.3e − 3	4.2e − 3	485.4	2817			
	2.7e − 3	2.9e − 3	4.4e − 3	228.9	3375	0.373	4.3e − 3	5.2e − 3	180.5	3347			
MNIST	0.019	766.1	770.9	2969	204.4	5.179	766.1	771.0	46.83	191.2			
	2.683	766.3	771.0	1271	362.5	13.89	766.3	771.1	46.08	230.7			
	19.31	769.2	773.7	1264	521.9	100.0	775.7	780.3	85.00	351.9	778.9	783.7	0.864
	138.9	825.9	829.5	1268	746.1	268.3	831.7	835.2	755.6	616.8			
	372.8	901.9	904.4	1250	770.7	1931	1118	1119	127.7	762.6			

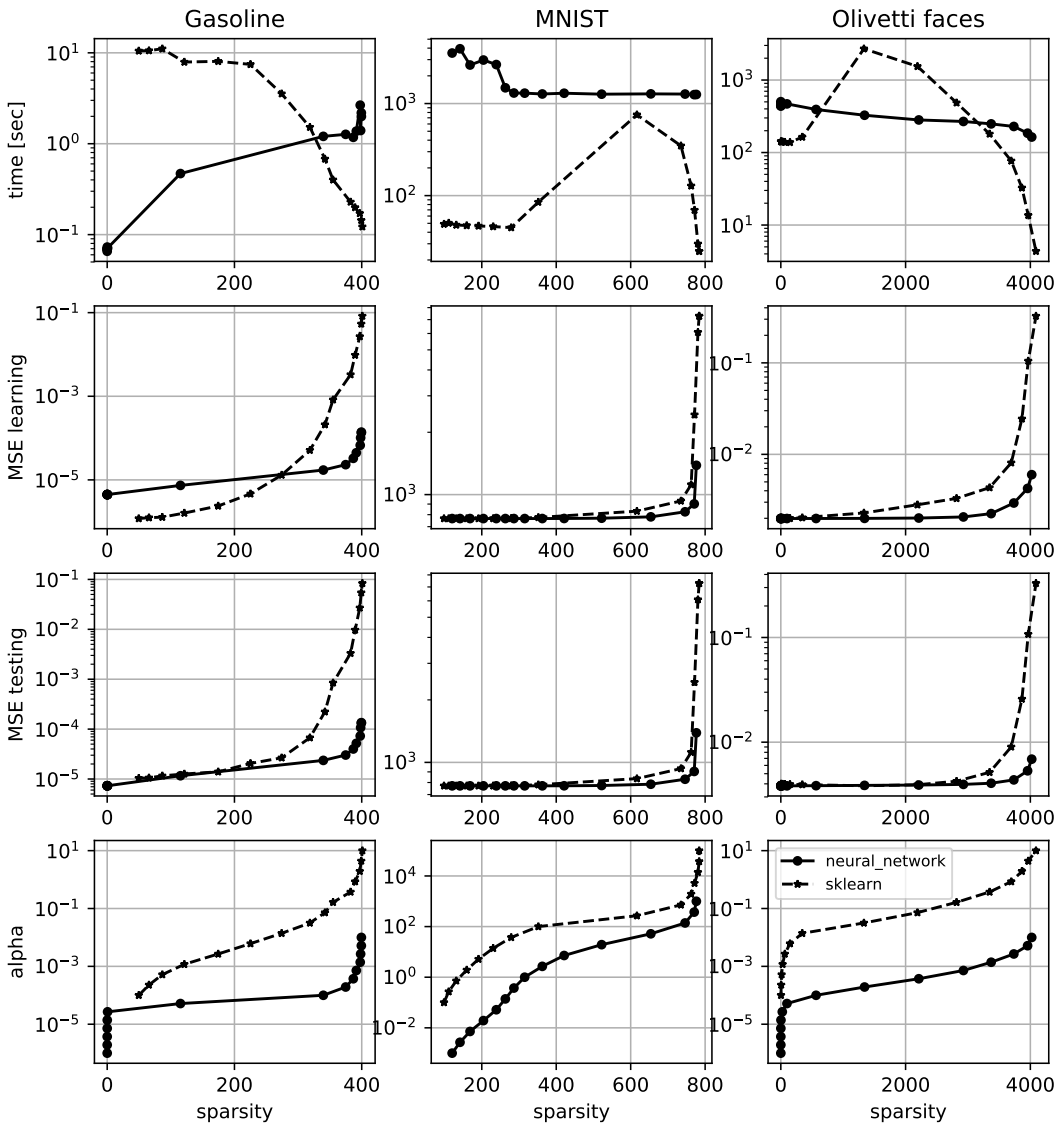
Wyniki prezentują odpowiednie miary jakości modeli (czas strojenia i jakość rekonstrukcji) w odniesieniu do rzadkości rozwiązań. Oznacza to, że dwa modele o tej samej rzadkości zostały porównane pod względem czasów obliczeń i jakości rekonstrukcji. Jako miarę jakości zastosowano standardowy średni błąd kwadratowy rekonstrukcji (`mse`). Rzadkość w części eksperymentalnej oznacza średnią liczbę zerowych współczynników na komponent.

5.1.2 Wyniki

Tabela 5.2 przedstawia porównanie proponowanego autoenkodera (`neural_network`) z procedurą `SparsePCA/scikit-learn` oraz standardową procedurą `PCA` (też z pakietu `scikit-learn`). Na rysunku 5.1 przedstawiono czasy obliczeń, jakość rekonstrukcji (tj. błąd MSE na danych uczących i testowych) oraz wartość parametru regularyzacyjnego α przy uzyskanej rzadkości.

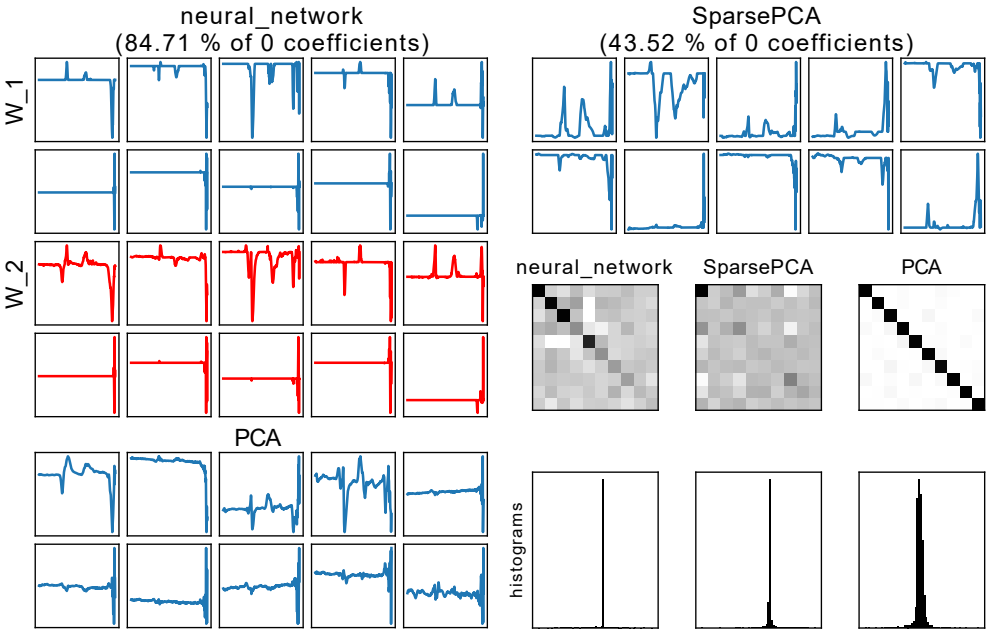
Jak można zauważyć na rysunku 5.1, pomimo że `SparsePCA` jest szybsze dla większej rzadkości, proponowane rozwiązanie (`neural_network`) zachowuje wyższą dokładność rekonstrukcji dla większej rzadkości, zarówno na zbiorze treningowym, jak i testowym. Ponadto istnieją obszary rzadkości, dla których proponowana procedura działa szybciej.

Porównanie komponentów (współczynników neuronów warstwy ukrytej) w zestawie danych `faces` jest przedstawione na rysunku: 5.6. Różne kolory czerwony/niebie-

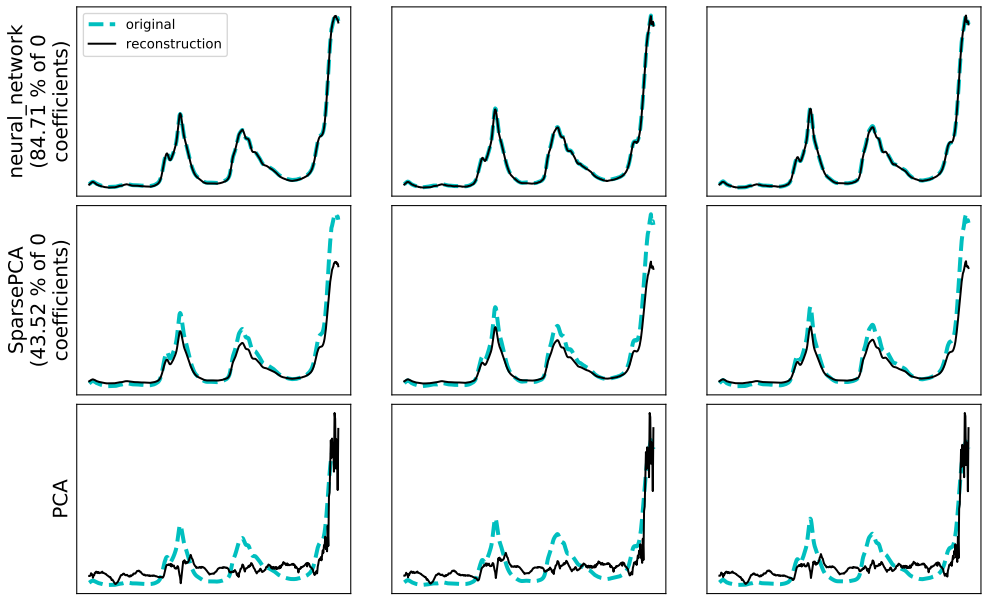


Rysunek 5.1: Porównanie proponowanego podejścia `neural_network` z procedurą `SparsePCA/scikit-learn`. Rzadkość oznacza średnią liczbę zerowych współczynników na składnik, α (alpha) jest parametrem regularyzacji. Od góry czas uczenia, błąd rekonstrukcji na danych uczących i testowych, parametr regularyzacyjny α.

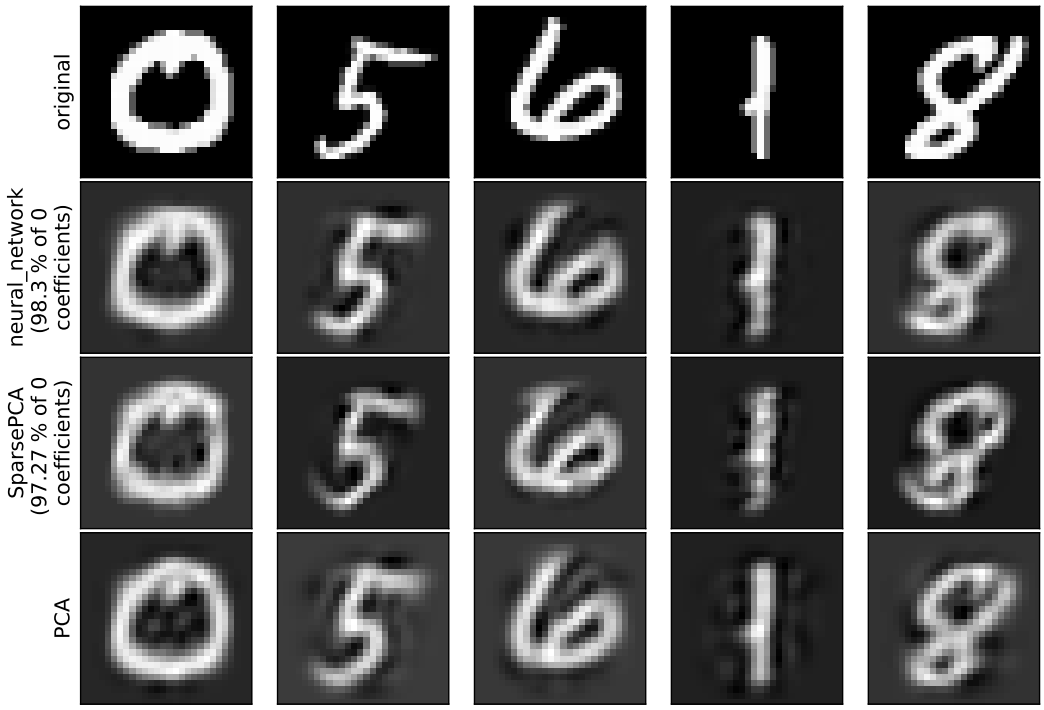
ski oznaczają współczynniki ujemne/dodatnie, a kolor biały oznacza dokładną wartość równą zero. Dodatkowo, dla porównania przedstawiono wyniki dla zwykłej procedury PCA oraz macierze kowariancji dla komponentów. Ortogonalność komponentów zwykłej procedury PCA jest gwarantowana teoretycznie, co potwierdzają również eksperymenty. Patrząc na wizualizacje macierzy kowariancji widzimy, że komponenty znalezione przez `neural_network` są bliskie ortogonalnym (wyraźniejsza przekątna), znacznie bardziej



Rysunek 5.2: Porównanie 10 komponentów wyznaczonych na zbiorze `gasoline`. Niebieski kolor oznacza komponenty ($\boldsymbol{W}^1$), czerwony to wektory używane do rekonstrukcji ($\boldsymbol{W}^2$). W dole po lewej składowe główne wyznaczone za pomocą procedury `PCA`. W dole po prawej wizualizacja macierzy kowariancji komponentów, a poniżej histogram wag dla poszczególnych modeli; uzyskana rzadkość: `neural_network` (93,47%), `SparsePCA/scikit-learn` (56,33%), `PCA` (0%).

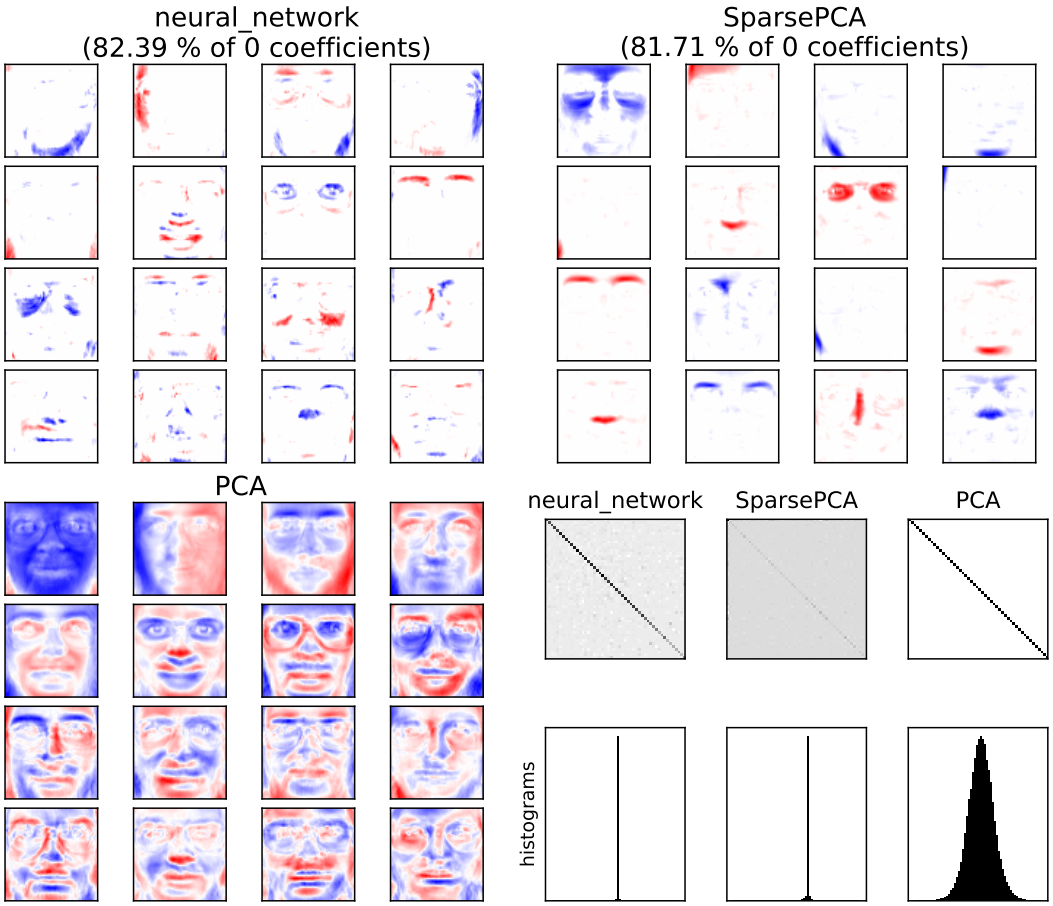


Rysunek 5.3: Dokładność rekonstrukcji dla zbioru `gasoline` — porównano modele `neural_network`, `SparsePCA` oraz model `PCA`. Oryginalne sygnały zaznaczono linią przerywaną, a rekonstrukcję linią ciągłą.



Rysunek 5.4: Porównanie jakości rekonstrukcji na zbiorze `MNIST`. Pierwszy wiersz zawiera oryginalne obrazy, poniżej rekonstrukcja przy użyciu kolejno modeli: `neural_network`, `SparsePCA` i `PCA`.

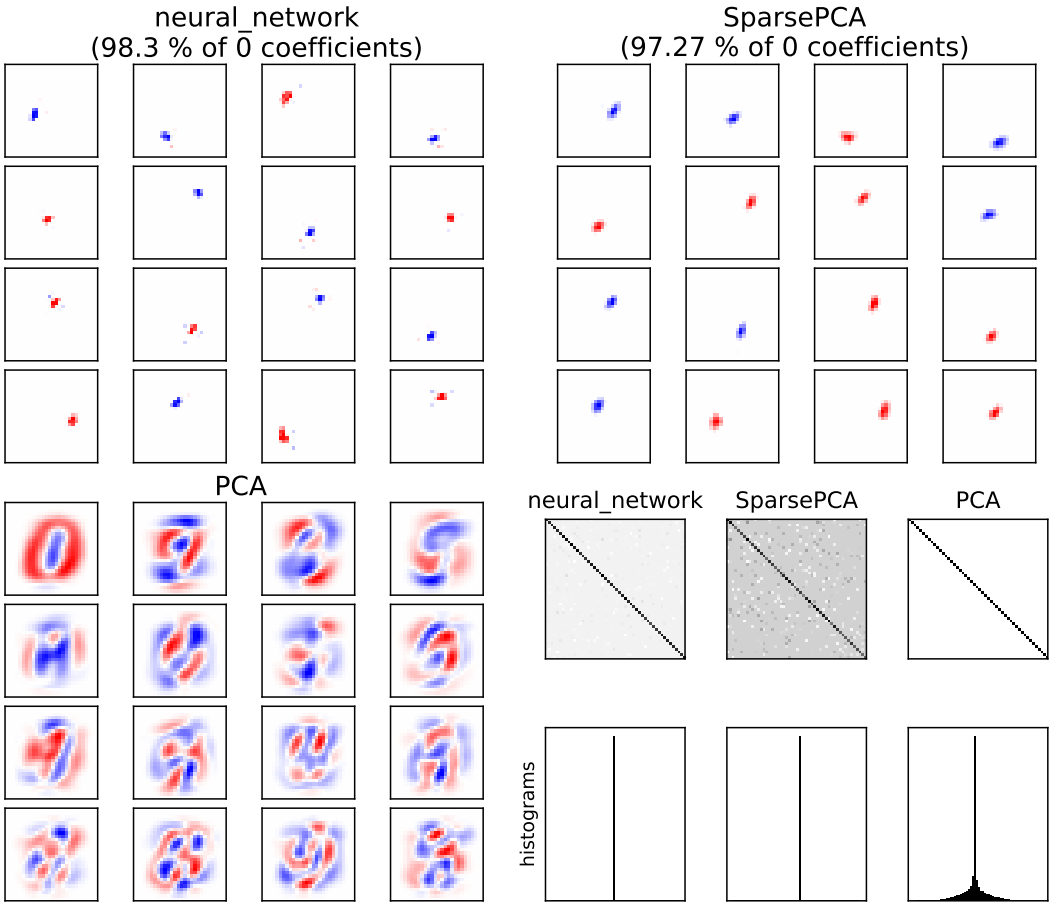
Rysunek 5.5: Porównanie jakości rekonstrukcji na zbiorze `Olivetti faces`. Pierwszy wiersz zawiera oryginalne obrazy, poniżej rekonstrukcja przy użyciu kolejno modeli: `neural_network`, `SparsePCA` i `PCA`.



Rysunek 5.6: Przykładowe 16 komponentów (spośród wszystkich 50) w zbiorze danych `Olivetti faces`. Kolor czerwony oznacza wartości ujemne, niebieski dodatnie, biały jest dokładnym zerem; lewy dolny róg: wyniki zwykłej procedury `PCA`; prawy dolny róg: macierze kowariancji dla komponentów oraz histogramy współczynników modeli. Uzyskana rzadkość: `neural_network` (82,39%), `SparsePCA/scikit-learn` (81,71%), `PCA` (0%).

niż te znalezione przez `SparsePCA`. Wyniki takie obserwowane są dla wszystkich trzech zbiorów: `gasoline`, `MNIST` i `Olivetti faces`.

Jakość rekonstrukcji dla danych widmowych `gasoline` jest przedstawiona na rysunku 5.3 (użyto 10 komponentów). Z kolei dla obrazów (dane `MNIST` i `Olivetti faces`) jakość rekonstrukcji przedstawiono na rysunkach 5.4 i 5.5 (w tym przypadku użyto 50 komponentów). Jak widać, nawet dla tak względnie małej liczby komponentów jakość jest całkiem dobra. Przy bardzo zbliżonej rzadkości proponowane podejście daje lepszą jakość rekonstrukcji w stosunku do metody `SparsePCA`. W przypadku `faces` patrząc na komponenty można zaobserwować nieco wyraźniejsze szczegóły, takie jak



Rysunek 5.7: Przykładowe 16 komponentów (spośród wszystkich 50) w zbiorze danych `MNIST`. Kolor czerwony oznacza wartości ujemne, niebieski dodatnie, biały jest dokładnym zerem. Lewy dolny róg: wyniki zwykłej procedury `PCA`, a w prawym dolnym rogu: macierze kowariancji dla komponentów oraz histogramy współczynników modeli. Uzyskana rzadkość: `neural_network` (98.3%), `SparsePCA/scikit-learn` (97.27%), `PCA` (0%).

okulary lub wąsy, broda czy obrys twarzy. W przypadku danych `gasoline` przy tej samej rzadkości (około 80%, tj. 320 wyzerowanych współczynnikach) jakość proponowanego podejścia `neural_network` jest znacznie lepsza. W przypadku zestawów danych `gasoline` i `faces`, posiadających stosunkowo mało próbek (ok. 10% liczby atrybutów), jakość rekonstrukcji obu rzadkich metod jest znacznie lepsza niż w przypadku standardowej procedury `PCA`, a przy większym zbiorze danych (jak `MNIST`) wpływ rzadkości na jakość rekonstrukcji jest mniejszy.

5.2 Klasyfikacja nieliniowa — uczenie zrandomizowane

Wyniki opisane w tej sekcji zostały opublikowane w pracy [42].

Zgodnie z opisem w sekcji 3.3 w tym eksperymencie wybieramy losowo współczynniki ukrytej warstwy $\mathbf{W}^1$ i $\boldsymbol{b}^1$, a liczbę neuronów w warstwie ukrytej ustalono na $k = 1000$. Mając ustalone wagi warstwy ukrytej wystarczy nauczyć współczynniki warstwy wyjściowej $\boldsymbol{W}^2$ i b^2, wykorzystując zamiast macierzy $\boldsymbol{X}_{n\times p}$ nową macierz przekształconych danych $\boldsymbol{V}_{n\times k}$, w której nowa obserwacja $\boldsymbol{v}_i$ odpowiada wynikowi $\tanh\left(\boldsymbol{W}^1 \cdot \boldsymbol{x}_i + \boldsymbol{b}^1\right)$. Eksperymenty wykonano w języku `Python`. Na potrzeby eksperymentów przygotowano klasę `ExtremeClassifier` (w paradygmacie `fit-predict` pakietu `scikit-learn`), która zależy od liczby neuronów w warstwie ukrytej k, rodzaju liniowego klasyfikatora wykorzystanego do strojenia warstwy wyjściowej i jego parametrów. W eksperymentach wszystkie klasyfikatory liniowe mają jednakowo inicjalizowaną warstwę uczącą. Wykorzystano osiem modeli liniowych:

`L1-Liblinear` model regresji logistycznej z karą ℓ^1 — klasa `LogisticRegression` z ustawieniami `penalty='l1'` oraz `solver='liblinear'`

`L1-QR` model z regularyzacją ℓ^1 opisany w sekcji 2.5.3 wykorzystujący wcześniejszy rozkład QR/LQ (podobnie do (2.60), por. [42])

`Lq-QR` model analogiczny do `L1-QR` wykorzystujący wcześniejszy rozkład QR/LQ, z regularyzacją w postaci pseudonormy ℓ^q, przy $q = 0{,}8$, por. [43, 44]

`L1-QR-soft` model zastępujący niegładką normę ℓ^1 jej gładkim przybliżeniem, opisany w sekcji 2.7, por. [45].

`L2-QR` model regresji logistycznej z regularyzacją ℓ^2 z wcześniejszym wykonaniem rozkładu QR, opisany w sekcji 2.4.1.

`L2-lbfgs` model regresji logistycznej z karą ℓ^2 — klasa `LogisticRegression` z ustawieniami `penalty='l2'` oraz `solver='lbfgs'`

`L2-NewtonCG` model regresji logistycznej z karą ℓ^2 — klasa `LogisticRegression` z ustawieniami `penalty='l2'` oraz `solver='newton-cg'`

`MLP-lbfgs` dodatkowo dla porównana dodano również nieliniowy model — perceptron wielowarstwowy (klasa `MLPClassifier`) z dwoma warstwami oraz z k neuronami w warstwie ukrytej. Model ten jest uczony w standardowy sposób przy użyciu dostępnego w pakiecie algorytmu `L-BFGS` (oznaczony jako `MLP-lbfgs`). Jest to model o strukturze równoważnej z pozostałymi modelami, z tą różnicą, że model ten stroi współczynniki obu warstw w procesie wstecznej propagacji — poprzednie modele losują i zamrażają współczynniki warstwy pośredniej. Model ten posiada

domyślnie ustawiony parametr $\alpha = 0{,}0001$ odpowiadający mechanizmowi „weight decay", czyli regularyzacji ℓ^2.

Model `L1-Liblinear` stanowi punkt odniesiena dla modeli rzadkich, a modele `L2-lbfgs, l2-netwoncg, MLP-lbfgs` stanowią punkt odniesienia dla modeli z regularyzacją ℓ^2.

W przypadku rzadkich funkcji kary tylko algorytmy `L1-Liblinear` i `L1-QR` dają ilościowo te same wyniki, jednak `L1-Liblinear` może działać kilkukrotnie szybciej. Inne modele (`L1-QR-soft, Lq-QR`) dają jakościowo różne wyniki. Algorytm `Lq-QR` dla $q = 0{,}8$ uzyskał najlepszą rzadkość i był również nieznacznie szybszy niż `L1-QR`. Pakiet `scikit-learn` udostępnia rozwiązania dla kar typu ℓ^2 i ℓ^1, lecz nie posiada takowego dla ogólnego przypadku ℓ^q.

Jako dane testowe wykorzystano trzy sztuczne zbiory:

1. `Chessboard3` Szachownica 3×3
2. `Chessboard4` Szachownica 4×4
3. `Spirals` Dwie zagnieżdżone w sobie spirale

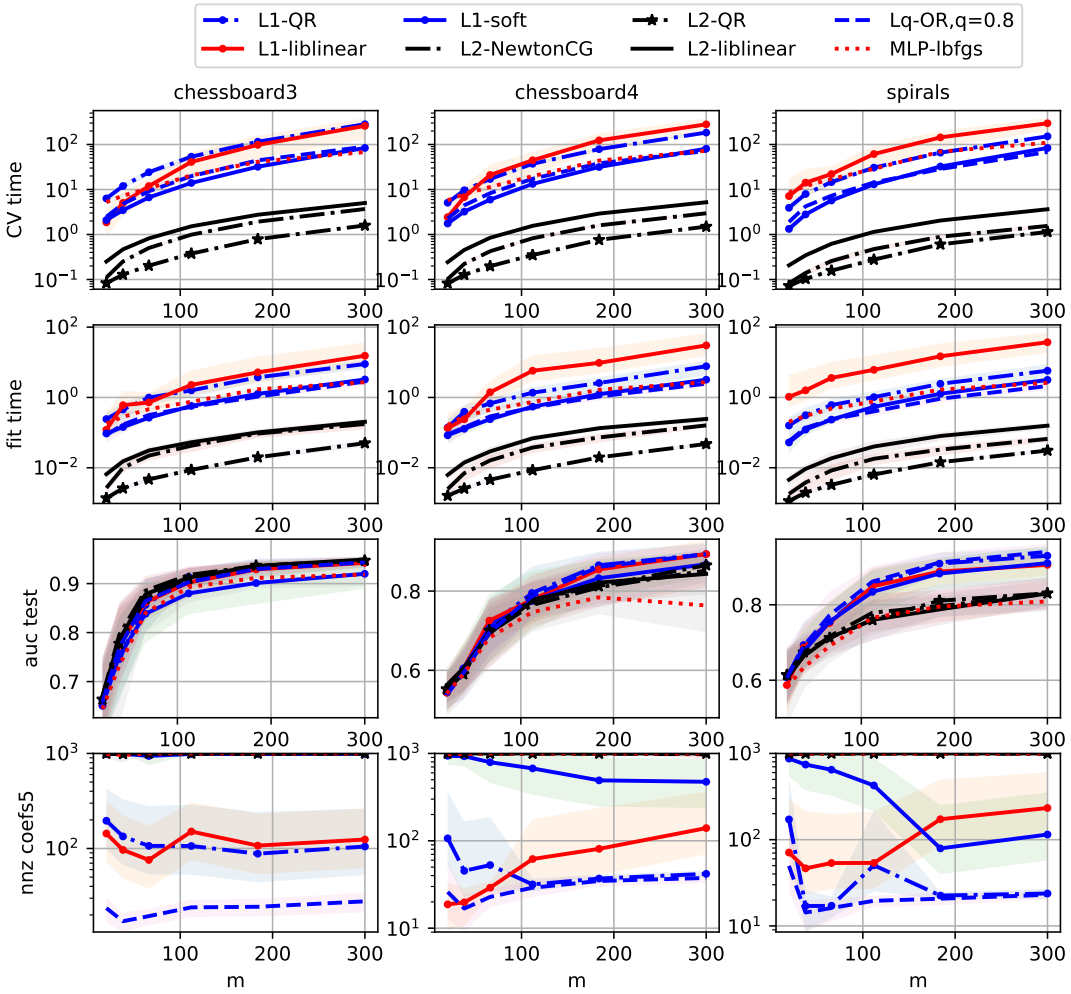
Eksperyment przebiegał następująco: zwiększano rozmiar próby uczącej od 20 do 300 próbek, dla każdego rozmiaru danych uczących i dla każdego klasyfikatora wybierano optymalną wartość parametru $C = 1/\lambda$ za pomocą walidacji krzyżowej, a w trakcie kolejnych iteracji notowano czas uczenia i testowania, błędy na zbiorze testowym oraz liczbę niezerowych współczynników (dotyczy modeli z karą $\ell^{0<q\leqslant 1}$). Wyniki zbiorcze eksperymentu przedstawiono na rysunku 5.8.

Jak widać w przypadku regularyzacji ℓ^2 rozwiązanie oparte na rozkładzie QR zawsze daje lepsze czasy dopasowania niż zwykłe „rozwiązywacze" dostępne w pakiecie `scikit-learn`. Czas dopasowania `L1-QR` jest 2–5 razy krótszy niż `L1-Liblinear`, szczególnie w przypadku szachownicy 4×4 i dwóch spirali — wynika to z faktu niedostosowania biblioteki do danych gęstych. Patrząc na jakość, widzimy, że modele rzadkie są zbliżonej jakości i okazały się znacznie lepsze niż modele z regularyzacją ℓ^2, zwłaszcza wyraźnie widać to dla danych `Chessborad4` i `Spirals`.

Dla zbioru `Spirals` najlepszy okazał się model `Lq-QR` i był to również model najrzadszy. Model `MLP-lbfgs` uzyskał najgorszą jakość dopasowania oraz porównywalny czas uczenia modeli rzadkich.

Eksperyment pokazuje, że użycie rozkładu QR można skutecznie połączyć z uczeniem zrandomizowanym (RVFL) z różnymi regularyzatorami. Co więcej, eksperymenty potwierdzają, że takie uczenie działa stabilniej niż zwykły algorytm uczenia sieci neuronowych, szczególnie w przypadku dużej liczby ukrytych neuronów. Przykładowe granice

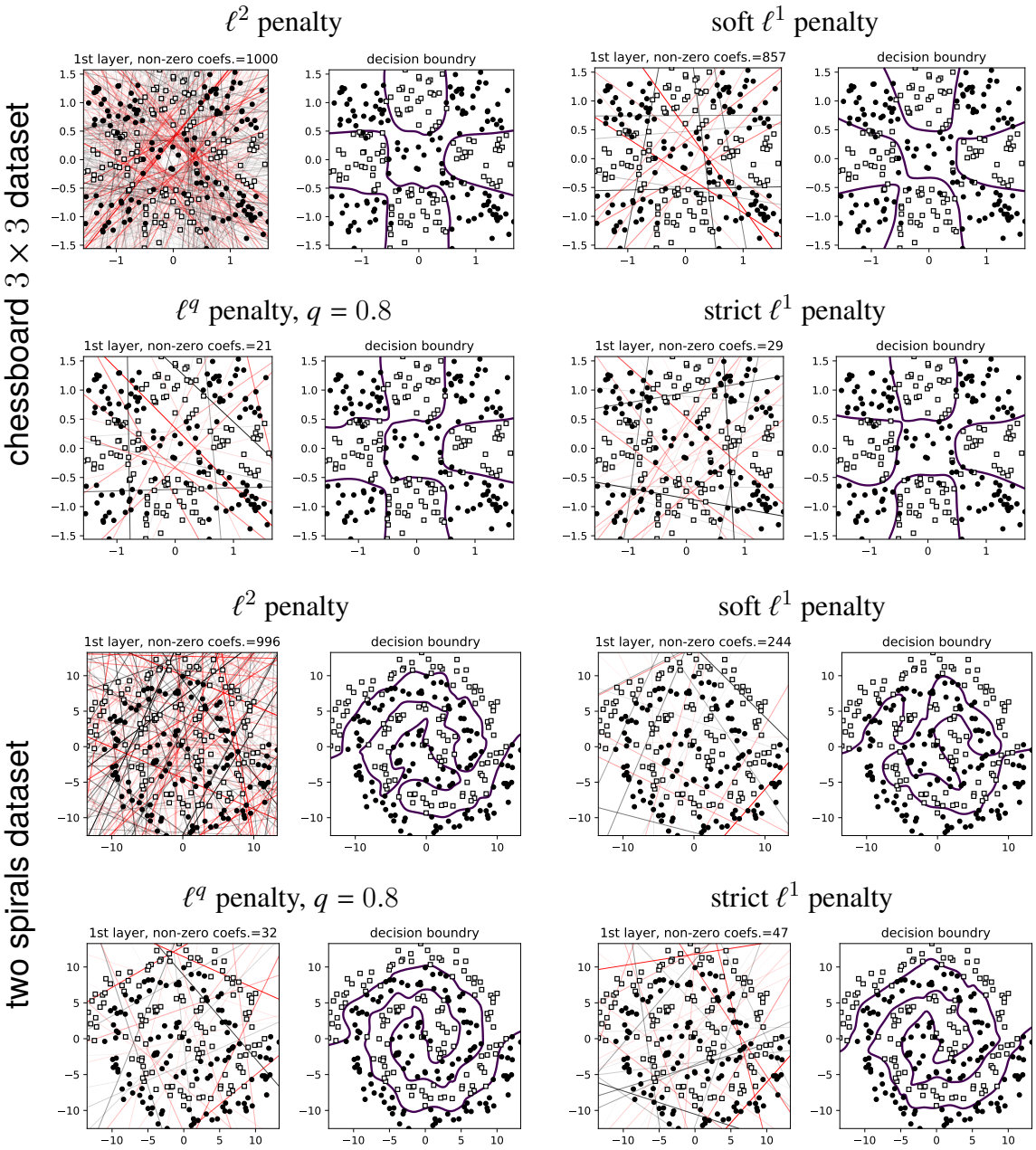
decyzyjne, rzadkość rozwiązania i wizualizacje wag neuronów ukrytych na tle danych wejściowych przedstawiono na rysunku 5.9, z którego można np. odczytać, że modele z rzadką regularyzacją redukują liczę neuronów w warstwie ukrytej z 1000 do ok. 20–50.



Rysunek 5.8: Wyniki eksperymentalne dla uczenia zrandomizowanego. Porównanie na sztucznych zbiorach danych. `CV time` to czas procedury walidacji krzyżowej wyboru prametru regularyzacyjnego, `fit time` to czas dopasowania dla znalezionego λ, `auc test` to miara AUC (pole pod krzywą ROC) na danych testowych, a nnz coefs to liczba niezerowych współczynników.

Przedstawione zastosowanie rozkładu QR macierzy w celu poprawy procedury Newtona uczenia modeli regresji logistycznej może być efektywnie wykorzystane w obu przypadkach, tj. w przypadku obrotowo niezmienniczym (z karą ℓ^2) oraz w przypadku modeli obrotowo zmiennych (z rzadkimi funkcjami kary)[1]. Wykonane eksperymenty wskazują, że użycie rozkładu QR może istotnie poprawić klasyczny algorytm Newton-CG dla regresji

[1]W tym drugim podejściu zastosowanie formuły Woodbury'ego będzie miało mniejszą złożoność obliczeniową, bo znika konieczność policzenia rozkładu. Artykuł powstał zanim podjęto bezpośrednią próbę



Rysunek 5.9: Przykładowe granice decyzyjne dla różnych funkcji składników regularyzacyjnych (ℓ^2, ℓ^1 z gładką aproksymacją funkcji wartości bezwzględnej, $\ell^{q=0,8}$, ℓ^1) dla różnych zbiorów testowych. Na rysunku (lewa strona) pokazano wizualizację współczynników pierwszej warstwy sieci neuronowej przedstawione jako linie — intensywność i kolor reprezentują wielkość i znak danego współczynnika.

z karą ℓ^2 w przypadku danych o rozmiarze $n \ll p$.

zastosowania formuły Woodbury'ego do uczenia modelu regresji logistycznej z karą ℓ^1 z góry ograniczoną przez karę ℓ^2.

Eksperymenty pokazały, że stosując rozkład QR i formułę Woodbury'ego możemy rozwiązać problem uczenia modelu regresji również z rzadkimi funkcjami kar. Zysk numeryczny z zastosowania rozkładu QR nie w tym przypadku tak spektakularny, jak w przypadku kary ℓ^2. W pracy [42] pokazano, że stosując rozkład QR uzyskujemy teoretyczną górną granicę lepszą niż w przypadku ogólnej procedury Newton-CG. Eksperymenty numeryczne wykazały, że w przypadku trudniejszych i skorelowanych danych (np. w przypadku uczenia ekstremalnego) oraz większych wartości parametru λ takie podejście może działać efektywniej niż metoda spadku względem współrzędnych (`L1-Liblinear`). Jednak należy przyznać, że w typowych i prostszych przypadkach metody oparte na spadku względem współrzędnych (w tym biblioteka `Liblinear`) są szybsze.

6. Podsumowanie i wnioski

W pracy przebadano rozmaite konfiguracje modelu regresji logistycznej, próbując przyspieszyć procedurę uczenia. Dla małych zbiorów danych wykazano teoretycznie i empirycznie, że w pewnym zakresie podejście zwyczajne (`l2_logreg_ordinary`, odpowiednik modelu `LogisticRegression` z ustawionym parametrem `solver='newton-cg'`) okazuje się mniej wydajne pod względem czasu uczenia od rozwiązania uzyskanego za pomocą formuły Woodbury'ego, co szczególnie dobrze widać dla zbiorów gęstych. O ile koncepcja wykorzystania rozkładu QR w zagadnieniu uczenia modeli liniowych (głównie w kontekście regresji) pojawia się w literaturze, o tyle analogiczne zastosowanie formuły Woodbury'ego wydaje się jeszcze niespotykane.

W przypadku rzadkich zbiorów danych analogiczny zysk dla modeli z karą ℓ^2 okazuje się trudniejszy do uzyskania. W eksperymentach było to obserwowane dla bardzo małych zbiorów. Dokładna analiza teoretyczna złożoności procedur w przypadku danych rzadkich wydaje się trudna, gdyż bardzo to zależy od rodzaju rzadkości macierzy wejściowych, ale również od rzadkości macierzy wynikowych powstających w wyniku kolejnych mnożeń, a także sposobów ich reprezentacji [1]. W przeprowadzanych eksperymentach korzystne okazało się przechowywanie wyników mimo wszystko w macierzach gęstych.

Pewna przewaga proponowanego podejścia może się ujawnić wtedy, gdy trzeba wielokrotnie uczyć model na tym samym zbiorze danych (np. podczas walidacji krzyżowej przy poszukiwaniu optymalnego parametru λ) — wtedy istotne znaczenie ma również kolejność parametrów λ_k, dla których chcemy nauczyć model. W takiej procedurze wartości λ powinny tworzyć malejący ciąg, zaś przy uczeniu kolejnego modelu rozsądnym założeniem jest skorzystanie z parametrów modelu nauczonego dla czynnika λ_{k-1}.

W ramach badań dotyczących modeli z regularyzacją ℓ^1 również zaproponowano model bazujący na rozkładzie QR oraz formule Woodbury'ego. Jednak zysk z tego podejścia, mimo teoretycznej atrakcyjności, nie okazał się lepszy od metod referencyjnych. W przypadku regresji logistycznej z regularyzacją ℓ^1 uważna implementacja metody spadku względem współrzędnych, korzystająca ze wszystkich udoskonaleń, radzi sobie zdecydo-

wanie najlepiej pomimo swej prostoty. Wszystkim metodom uczącym dodatkowo pomóc może wykluczenie na wstępie atrybutów, które przy zadanej wartości λ na pewno nie wejdą do wynikowego modelu [24, 77, 81]. W przypadku metody opartej na zastąpieniu normy ℓ^1 przez normę ℓ^2 największym mankamentem jest względnie długi czas potrzebny do uzyskania w pełni rzadkiego rozwiązania, ponieważ zerowanie współczynników zmierzających do zera powoduje chwilowe zaburzenie.

Dla części z procedur konieczna była również adaptacja do danych rzadkich, co ujawniło w eksperymentach pewne zalety metod wyższego rzędu (właśnie dla danych rzadkich) w stosunku metody bazującej na spadku względem współrzędnych. Jednym z wniosków dotyczących adaptacji algorytmów do danych rzadkich dotyczy rozkładu QR. Rozkład ten dla dużych danych jest problematyczny pamięciowo, a zwłaszcza konieczność wyznaczenia macierzy $\boldsymbol{Q}$, o której należy założyć, że jest gęsta, a jej rozmiar jest równy rozmiarowi danych. Problem ten nie występuje w przypadku zastosowania formuły Woodbury'ego.

W ostatnich wersjach pakietu `scikit-learn` (>1.2) wprowadzono dla regresji z karą ℓ^2 nową metodę strojenia `newton-cholesky`, jednak jest ona dostosowana do przypadku $n \gg p$ i zupełnie nieprzydatna w przypadkach rozważanych w pracy. Eksperymenty pokazuja, że nawet drobna modyfikacja istniejącego kodu pakietu `scikit-learn` (jak ta przedstawiona na stronie 22) może dać zysk obliczeniowy dla małych zbiorów danych.

W pracy przebadano szereg własności numerycznych algorytmów i nasuwają się tu pewne wnioski odnośnie ich konfiguracji. W przypadku algorytmów rozwiązujących zadanie uczenia w przestrzeni zredukowanej (metoda z rozkładem QR/LQ czy z formułą Woodbury'ego) zastosowanie preconditionera ma istotne znaczenie, natomiast w przypadku klasycznej procedury Newton-CG nie jest to kluczowe. Z rozważanych procedur gradientu sprzężonego i SYMMLQ, ta pierwsza, mimo że w pewnych konfiguracjach bywa szybsza, nie radzi sobie z problemami nieokreślonymi, a takie macierze czasami pojawiają się np. po zastosowaniu formuły Woodbury'ego uwzględniającego wyraz wolny.

Korzystając z paradygmatu extreme machine learning (zrandomizowanego ustalania wag warstwy wejściowej) można w prosty sposób na bazie istniejącego zbioru danych przygotować nowy, z dużo większą liczbą cech, gdzie granica separacji będzie łatwiejsza do znalezienia. W takim podejściu rzadkie modele liniowe są naturalnym i bardzo atrakcyjnym narzędziem selekcji cech przekształcenia zrandomizowanego. Przeprowadzone eksperymenty pokazały duże zalety takiego podejścia w porównaniu do standardowych algorytmów uczenia sieci wielowarstwowych (np. zwykła wsteczna propagacja czy `L-BFGS`). W przypadku przewymiarowanych sieci, standardowe metody bazujące na gradiencie prostym mają tendencję utykania w minimach lokalnych (np. część neuronów

warstwy ukrytej nie podlega procesowi uczenia). Ponadto uczenie takiego klasyfikatora z losową warstwą pośrednią jest znacznie szybsze, ponieważ wymaga tylko strojenia warstwy wyjściowej. Takie podejście może być łatwo zaadaptowane do głębokiego uczenia, np. wybierając parametry jąder w sposób losowy i strojąc warstwę wyjściową procedurami uczenia klasyfikatorów liniowych. Podejście to może stanowić punkt startowy przy douczaniu sieci.

Zastosowanie metod regularyzowanych, a zwłaszcza rzadkich regularyzacji (typu lasso czy fused lasso) może być wykorzystane do ekstrakcji cech posiadających pewne własności, atrakcyjne z punktu widzenia interpretacji wyników, np. wykryte piki w spektrogramie czy oczy, broda, okulary na zdjęciach twarzy. Mimo że nie jest to nowa obserwacja, to w ramach pracy udało się zaimplementować strukturę rzadkiego autoenkodera, który może służyć jako ekstraktor cech rzadkich. Opracowana implementacja okazała się w pewnych przypadkach konkurencyjna w stosunku do zaimplementowanego w pakiecie `scikit-learn` uczenia słownikowego.

W pracy duży nacisk położono na metody wyższego rzędu, jednak należy mieć świadomość, że współczesne uczenie maszynowe opiera się głównie na metodach pierwszego rzędu, tj. metodach gradientowych. Pakiety automatycznego różniczkowania (typu Tensorflow czy Torch), na których opierają się biblioteki głębokiego uczenia, mają zdefiniowaną funkcję wartości bezwzględnej (abs) wraz z jej pochodną w postaci funkcji signum (sign). Za pomocą tej funcji możemy z łatwością zbudować model wyposażony w regularyzację typu ℓ^1, jednak stosując typową metodę gradientu prostego (bez wyrafinowanej zmiany współczynnika uczenia) nie uzyskamy efektu zerowania się współczynników, a proces zbieżności w pobliże rozwiązania jest bardzo powolny. Ogólnie bezpośrednie stosowanie rzadkich regularyzacji w pakietach wydaje się kłopotliwe, choć jest możliwe.

Spis literatury

[1] Amir Abboud i in. “The Time Complexity of Fully Sparse Matrix Multiplication”. W: Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), s. 4670–4703. DOI: 10.1137/1.9781611977912.167. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611977912.167.

[2] E. Anderson i in. LAPACK Users’ Guide. Third. Society for Industrial i Applied Mathematics, 1999. DOI: 10.1137/1.9780898719604. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719604. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898719604.

[3] Galen Andrew i Jianfeng Gao. “Scalable Training of L_1-Regularized Log-Linear Models”. W: Proceedings of the 24th International Conference on Machine Learning. ICML ’07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, s. 33–40. ISBN: 9781595937933. DOI: 10.1145/1273496.1273501. URL: https://doi.org/10.1145/1273496.1273501.

[4] Martin Anthony i Peter L. Bartlett. Neural Network Learning: Theoretical Foundations. USA: Cambridge University Press, 2009.

[5] Richard Barrett i in. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Society for Industrial i Applied Mathematics, 1994. DOI: 10.1137/1.9781611971538. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611971538. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611971538.

[6] Peter L. Bartlett i Shahar Mendelson. “Rademacher and gaussian complexities: risk bounds and structural results”. W: J. Mach. Learn. Res. 3 (2003), 463–482.

[7] Amir Beck i Marc Teboulle. “A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems.” W: SIAM Journal on Imaging Sciences 2.1 (2009),

s. 183–202. URL: http://dblp.uni-trier.de/db/journals/siamis/siamis2.html#BeckT09.

[8] H. Bourlard i Y. Kamp. "Auto-association by multilayer perceptrons and singular value decomposition". W: Biological Cybernetics 59.4 (1988), s. 291–294. DOI: 10.1007/BF00332918. URL: https://doi.org/10.1007/BF00332918.

[9] Stephen Boyd i Lieven Vandenberghe. Convex Optimization. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787.

[10] Stephen Boyd i in. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". W: Foundations and Trends in Machine Learning 3.1 (sty. 2011), s. 1–122. ISSN: 1935-8237. DOI: 10.1561/2200000016. URL: http://dx.doi.org/10.1561/2200000016.

[11] Kristian Bredies, Dirk A. Lorenz i Stefan Reiterer. "Minimization of Non-Smooth, Non-Convex Functionals by Iterative Thresholding". W: Journal of Optimization Theory and Applications 165.1 (kw. 2015), s. 78–112. ISSN: 0022-3239. DOI: 10.1007/s10957-014-0614-7. URL: https://doi.org/10.1007/s10957-014-0614-7.

[12] AT&T Laboratories Cambridge. "The Database of Faces". W: (1994). URL: https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[13] C. Cortes i V. Vapnik. "Support Vector Networks". W: Machine Learning 20 (1995), s. 273–297.

[14] IEEE Standard for Floating-Point Arithmetic. Standard IEEE Std 754-2008. New York, NY, USA: IEEE Computer Society, 2008.

[15] Alexandre d'Aspremont i in. "A Direct Formulation for Sparse PCA Using Semidefinite Programming". W: CoRR cs.CE/0406021 (czer. 2004). DOI: 10.2139/ssrn.563524.

[16] Aaron Defazio, Francis Bach i Simon Lacoste-Julien. "SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives". W: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1. NIPS'14. Montreal, Canada: MIT Press, 2014, s. 1646–1654.

[17] B. Deng i in. “An Overview of Extreme Learning Machine”. W: 2019 4th International Conference on Control, Robotics and Cybernetics (CRC). IEEE, 2019, s. 189–195.

[18] Bradley Efron i in. “Least Angle Regression”. English. W: The Annals of Statistics 32.2 (2004), s. 407–451. ISSN: 00905364.

[19] Paul H. C. Eilers. “A Perfect Smoother”. W: Analytical Chemistry 75.14 (2003), s. 3631–3636. DOI: https://doi.org/10.1021/ac034173t.

[20] Jianqing Fan i Runze Li. “Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties”. W: Journal of the American Statistical Association 96.456 (2001), s. 1348–1360. DOI: 10.1198/016214501753382273.

[21] Jerome Friedman, Trevor Hastie i Rob Tibshirani. “Regularization Paths for Generalized Linear Models via Coordinate Descent”. W: Journal of Statistical Software, Articles 33.1 (2010), s. 1–22. ISSN: 1548-7660. DOI: 10.18637/jss.v033.i01. URL: https://www.jstatsoft.org/v033/i01.

[22] Leonardo Galli i Chih-Jen Lin. Supplementary materials for “A Study on Truncated Newton Methods for Linear Classification”. 2020. URL: "https://www.csie.ntu.edu.tw/~cjlin/papers/tncg/supplement.pdf".

[23] Leonardo Galli i Chih-Jen Lin. “A Study on Truncated Newton Methods for Linear Classification”. W: IEEE Transactions on Neural Networks and Learning Systems 33.7 (2022), s. 2828–2841. DOI: 10.1109/TNNLS.2020.3045836.

[24] Laurent Ghaoui, Vivian Viallon i Tarek Rabbani. “Safe Feature Elimination for the LASSO and Sparse Supervised Learning Problems”. W: CoRR abs/1009.3515 (wrz. 2010).

[25] Gene H. Golub i Charles F. Van Loan. Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. ISBN: 9781421407944.

[26] Pinghua Gong i Jieping Ye. “A Modified Orthant-Wise Limited Memory Quasi-Newton Method with Convergence Analysis”. W: Proceedings of the 32nd International Conference on Machine Learning. Red. Francis Bach i David Blei. T. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, s. 276–284. URL: https://proceedings.mlr.press/v37/gonga15.html.

[27] P. J. Green. “Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives (with discussion)”. W: Journal of the Royal Statistical Society, Series B, Methodological 46 (1984), s. 149–192.

[28] T. Hastie i R. Tibshirani. Expression arrays and the $p \gg n$ problem. 2003.

[29] Trevor Hastie, Robert Tibshirani i Jerome Friedman. The Elements of Statistical Learning. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[30] H. V. Henderson i S. R. Searle. “On Deriving the Inverse of a Sum of Matrices”. W: SIAM Review 23.1 (1981), s. 53–60. DOI: 10.1137/1023004.

[31] M. R. Hestenes i E. Stiefel. “Methods of Conjugate Gradients for Solving Linear Systems”. W: Journal of research of the National Bureau of Standards 49 (1952), s. 409–436.

[32] Tim Hesterberg i in. “Least angle and ℓ_1 penalized regression: A review”. W: Statistics Surveys 2.none (2008), s. 61 –93. DOI: 10.1214/08-SS035. URL: https://doi.org/10.1214/08-SS035.

[33] Arthur E. Hoerl i Robert W. Kennard. “Ridge Regression: Biased Estimation for Nonorthogonal Problems”. W: Technometrics 12.1 (1970), s. 55–67. DOI: 10.1080/00401706.1970.10488634.

[34] G.B. Huang, L. Chen i C.K. Siew. “Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Nodes”. W: IEEE Transactions on Neural Networks 17.4 (2006), 879—892.

[35] Peter J. Huber. “Robust estimation of a location parameter”. W: Annals of Mathematical Statistics 35.1 (mar. 1964), s. 73–101. ISSN: 0003-4851. DOI: 10.1214/aoms/1177703732. URL: http://dx.doi.org/10.1214/aoms/1177703732.

[36] R. Hunter i Runze Li. “Variable Selection Using MM Algorithm”. W: Annals of Statistics (2005), s. 1617–1642.

[37] B. Igelnik i Yoh-Han Pao. “Stochastic choice of basis functions in adaptive function approximation and the functional-link net”. W: Trans. Neur. Netw. 6.6 (1995), 1320–1329. ISSN: 1045-9227. DOI: 10.1109/72.471375. URL: https://doi.org/10.1109/72.471375.

[38] Intel. Intel® oneAPI Math Kernel Library Documentation. 2023. URL: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-documentation.html.

[39] Rodolphe Jenatton, Guillaume Obozinski i Francis Bach. Structured Sparse Principal Component Analysis. 2009. URL: https://hal.archives-ouvertes.fr/hal-00414158.

[40] Ata Kabán i Robert J. Durrant. "Learning with $L_{q<1}$ vs L_1-Norm Regularisation with Exponentially Many Irrelevant Features". W: Machine Learning and Knowledge Discovery in Databases. Red. Walter Daelemans, Bart Goethals i Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 580–596.

[41] John H. Kalivas. "Two Data Sets of Near Infrared Spectra". W: Chemometrics and Intelligent Laboratory Systems 37 (1997), s. 255–259.

[42] Jacek Klimaszewski i Marcin Korzeń. "Fitting Penalized Logistic Regression Models Using QR Factorization". W: Computational Science – ICCS 2020. Red. Valeria V. Krzhizhanovskaya i in. Cham: Springer International Publishing, 2020, s. 44–57. ISBN: 978-3-030-50417-5.

[43] Jacek Klimaszewski i Marcin Korzeń. "Image Smoothing Using ℓ^p Penalty for $0 \leq p \leq 1$ with Use of Alternating Minimization Algorithm". W: Progress in Computer Recognition Systems. Red. Robert Burduk, Marek Kurzynski i Michał Wozniak. Cham: Springer International Publishing, 2020, s. 224–234. ISBN: 978-3-030-19738-4.

[44] Jacek Klimaszewski i Marcin Korzeń. "Optimization of ℓ^p regularized Linear Models via Coordinate Descent". W: Schedae Informaticae 25 (2016), s. 61–72.

[45] Jacek Klimaszewski, Michał Sklyar i Marcin Korzeń. "Learning ℓ^1-penalized logistic regressions with smooth approximation". W: 2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA). 2017, s. 126–130. DOI: 10.1109/INISTA.2017.8001144.

[46] Kwangmoo Koh, Seung-Jean Kim i Stephen Boyd. "An Interior-Point Method for LargeScale ℓ_1-Regularized Logistic Regression". W: Journal of Machine Learning Research 8 (grud. 2007), s. 1519–1555. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1314498.1314550.

[47] B. Krishnapuram i in. “Sparse Multinomial Logistic Regression: Fast Algorithms and Generalization Bounds”. W: IEEE Transactions on Pattern Analysis and Machine Intelligence 27.6 (2005), s. 957–968.

[48] Yann LeCun i Corinna Cortes. “MNIST handwritten digit database”. W: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[49] Su-In Lee i in. “Efficient L_1 Regularized Logistic Regression”. W: AAAI. AAAI Press, 2006, s. 401–408. URL: http://dblp.uni-trier.de/db/conf/aaai/aaai2006.html#LeeLAN06.

[50] Yuh-Jye Lee i O.L. Mangasarian. “SSVM: A Smooth Support Vector Machine for Classification”. W: Computational Optimization and Applications 20.1 (2001), s. 5–22. ISSN: 1573-2894. DOI: 10.1023/A:1011215321374. URL: https://doi.org/10.1023/A:1011215321374.

[51] David D. Lewis i in. “RCV1: A New Benchmark Collection for Text Categorization Research”. W: J. Mach. Learn. Res. 5 (2004), 361–397. ISSN: 1532-4435.

[52] Chih-Jen Lin, Ruby C. Weng i S. Sathiya Keerthi. “Trust Region Newton Method for Logistic Regression.” W: Journal of Machine Learning Research 9 (22 kw. 2010), s. 627–650.

[53] Julien Mairal i in. “Online Dictionary Learning for Sparse Coding”. W: Proceedings of the 26th Annual International Conference on Machine Learning. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, s. 689–696. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553463. URL: http://doi.acm.org/10.1145/1553374.1553463.

[54] T. P. Minka. A comparison of numerical optimizers for logistic regression. 2003. URL: https://tminka.github.io/papers/logreg/minka-logreg.pdf.

[55] Kevin P. Murphy. Machine learning: a probabilistic perspective. Cambridge, Mass.: MIT Press, 2013.

[56] Andrew Ng. CS294A lecture notes: Sparse autoencoder. 2019. URL: https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf.

[57] Andrew Y. Ng. “Feature Selection, L_1 vs. L_2 Regularization, and Rotational Invariance”. W: Proceedings of the Twenty-first International Conference on Machine Learning. ICML ’04. Banff, Alberta, Canada: ACM, 2004, s. 78–. ISBN: 1-58113-

838-5. DOI: 10.1145/1015330.1015435. URL: http://doi.acm.org/10.1145/1015330.1015435.

[58] R. W. D. Nickalls. "A New Approach to Solving the Cubic: Cardan's Solution Revealed". W: The Mathematical Gazette 77.480 (1993), s. 354–359. ISSN: 00255572. URL: http://www.jstor.org/stable/3619777.

[59] Jorge Nocedal i Stephen J. Wright. Numerical Optimization. 2e. New York, NY, USA: Springer, 2006.

[60] C. C. Paige i M. A. Saunders. "Solution of Sparse Indefinite Systems of Linear Equations". W: SIAM Journal on Numerical Analysis 12.4 (1975), s. 617–629. DOI: 10.1137/0712047. eprint: https://doi.org/10.1137/0712047. URL: https://doi.org/10.1137/0712047.

[61] F. Pedregosa i in. "Scikit-learn: Machine Learning in Python". W: Journal of Machine Learning Research 12 (2011), s. 2825–2830.

[62] Jan Rodziewicz-Bielewicz, Jacek Klimaszewski i Marcin Korzeń. "Regularized Learning of Neural Network with Application to Sparse PCA". W: Artificial Intelligence and Soft Computing. Red. Leszek Rutkowski i in. Cham: Springer International Publishing, 2019, s. 203–214. ISBN: 978-3-030-20912-4.

[63] Volker Roth. "Probabilistic Discriminative Kernel Classifiers for Multi-Class Problems". W: Pattern Recognition. Red. Bernd Radig i Stefan Florczyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 246–253. ISBN: 978-3-540-45404-5.

[64] Donald B. Rubin. "The Calculation of Posterior Distributions by Data Augmentation: Comment: A Noniterative Sampling/Importance Resampling Alternative to the Data Augmentation Algorithm for Creating a Few Imputations When Fractions of Missing Information Are Modest: The SIR Algorithm". W: Journal of the American Statistical Association 82.398 (1987), s. 543–546. ISSN: 01621459. URL: http://www.jstor.org/stable/2289460.

[65] Leonid I. Rudin, Stanley Osher i Emad Fatemi. "Nonlinear Total Variation Based Noise Removal Algorithms". W: Proceedings of the Eleventh Annual International Conference of the Center for Nonlinear Studies on Experimental Mathematics: Computational Issues in Nonlinear Science. Los Alamos, New Mexico, USA: Elsevier North-Holland, Inc., 1992, s. 259–268.

[66] Yousef Saad. Iterative Methods for Sparse Linear Systems. Second. Other Titles in Applied Mathematics. SIAM, 2003. ISBN: 978-0-89871-534-7. DOI: 10.1137/1.9780898718003.

[67] Conrad Sanderson i Ryan Curtin. "Armadillo: a template-based C++ library for linear algebra". W: Journal of Open Source Software 1.2 (2016), s. 26. DOI: 10.21105/joss.00026. URL: https://doi.org/10.21105/joss.00026.

[68] Mark Schmidt, Nicolas Le Roux i Francis Bach. "Minimizing Finite Sums with the Stochastic Average Gradient". W: Mathematical Programming (2013).

[69] Shai Shalev-Shwartz i Ambuj Tewari. "Stochastic Methods for $\ell_1$regularized Loss Minimization". W: Journal of Machine Learning Research 12 (lip. 2011), s. 1865–1892. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2021059.

[70] Jack Sherman i Winifred J. Morrison. "Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix". W: The Annals of Mathematical Statistics 21.1 (1950), s. 124–127. DOI: 10.1214/aoms/1177729893. URL: https://doi.org/10.1214/aoms/1177729893.

[71] Connor Shorten i Taghi M Khoshgoftaar. "A survey on image data augmentation for deep learning". W: Journal of Big Data 6.1 (2019), s. 1–48.

[72] Nitish Srivastava i in. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". W: Journal of Machine Learning Research 15.56 (2014), s. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[73] Jiexiong Tang i Guang-Bin Huang. "Extreme Learning Machine for Multilayer Perceptron". W: IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS 27.4 (2016).

[74] Martin A. Tanner i Wing Hung Wong. "The Calculation of Posterior Distributions by Data Augmentation". W: Journal of the American Statistical Association 82.398 (1987), s. 528–540. ISSN: 01621459. URL: http://www.jstor.org/stable/2289457 (term. wiz. 09. 09. 2024).

[75] R. Tibshirani. "Regression Shrinkage and Selection via the Lasso". W: Journal of the Royal Statistical Society (Series B) 58 (1996), s. 267–288.

[76] Robert Tibshirani i in. "Sparsity and smoothness via the fused lasso". W: Journal of the Royal Statistical Society Series B (2005), s. 91–108.

[77] Robert Tibshirani i in. “Strong Rules for Discarding Predictors in Lasso-Type Problems”. W: Journal of the Royal Statistical Society Series B: Statistical Methodology 74.2 (list. 2011), s. 245–266. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2011.01004.x. eprint: https://academic.oup.com/jrsssb/article-pdf/74/2/245/49513900/jrsssb_74_2_245.pdf. URL: https://doi.org/10.1111/j.1467-9868.2011.01004.x.

[78] A. N. Tikhonov. “On the solution of ill-posed problems and the method of regularization”. W: Dokl. Akad. Nauk SSSR 151.3 (1963), s. 501–504.

[79] Paul Tseng i Sangwoon Yun. “A Coordinate Gradient Descent Method for Nonsmooth Separable Minimization”. W: Math. Program. 117 (mar. 2009), s. 387–423. DOI: 10.1007/s10107-007-0170-0.

[80] Vladimir N. Vapnik. The nature of statistical learning theory. Springer-Verlag New York, Inc., 1995. ISBN: 0-387-94559-8.

[81] Jie Wang i in. “A Safe Screening Rule for Sparse Logistic Regression”. W: Neural Information Processing Systems. 2013. URL: https://api.semanticscholar.org/CorpusID:2499725.

[82] L.P. Wang i C.R. Wan. “Comments on ‘The Extreme Learning Machine’”. W: IEEE Transactions on Neural Networks 19.8 (2008), s. 1494–1495.

[83] Peter M. Williams. “Bayesian Regularization and Pruning Using a Laplace Prior”. W: Neural Computation 7 (1995), s. 117–143.

[84] Max A Woodbury. Inverting modified matrices. Statistical Research Group, Memo. Rep. 42. Princeton, N. J: Princeton University, 1950.

[85] Gui-Bo Ye i Xiaohui Xie. “Split Bregman Method for Large Scale Fused Lasso”. W: Computational Statistics and Data Analysis 55.4 (kw. 2011), 1552–1569. ISSN: 0167-9473.

[86] Guo-Xun Yuan, Chia-Hua Ho i Chih-Jen Lin. “An Improved GLMNET for L1-regularized Logistic Regression”. W: Journal of Machine Learning Research 13.1 (czer. 2012), s. 1999–2030. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2503308.2343708.

[87] Guo-Xun Yuan i in. “A Comparison of Optimization Methods and Software for Large-Scale L1-Regularized Linear Classification”. W: Journal of Machine Learning Research 11 (grud. 2010), 3183–3234. ISSN: 1532-4435.

[88] Ciyou Zhu i in. “Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-scale Bound-constrained Optimization”. W: ACM Trans. Math. Softw. 23.4 (grud. 1997), s. 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: http://doi.acm.org/10.1145/279232.279236.

[89] Hui Zou i Trevor Hastie. “Regularization and variable selection via the Elastic Net”. W: Journal of the Royal Statistical Society, Series B 67 (2005), s. 301–320.

[90] A.M. Zyarah i D. Kudithipudi. “Extreme learning machine as a generalizable classification engine”. W: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017, s. 3371–3376.

Załącznik A

Do pracy załączono zestaw kodów źródłowych w językach `C++` i `Python`, zawierający wszystkie metody zaimplementowane w ramach pracy oraz większość eksperymentów. Przed uruchomieniem należy zainstalować pakiety zależne (m.in. `MKL`, `Armadillo`, `Python/scikit-learn`) oraz zbudować odpowiednie biblioteki wywołując polecenie `make`.