Київський національний університет
імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Клаліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 113 Прикладна математика

на тему

Федеративне навчання і метод простої ітерації

Виконав студент IV курсу
Кутузов Владислав Павлович _____

Науковий керівник
доктор фізико-математичних наук
Семенов Володимир Вікторович _____

Засвідчую, що в цій роботі немає запозичень
з праць інших авторів без відповідних посилань.

Студент _____
Роботу розглянуто й допущено до
захисту на засіданні кафедри
обчислювальної математики
«____» _____ 2022 р.,
протокол № ____
Завідувач кафедри
С. І. Ляшко _____

Київ – 2022

Реферат

Обсяг роботи 42 сторінkb, 16 джерел посилання, 7 ілюстрацій, 0 таблиць, 2 додатки.

Ключові слова: ФЕДЕРАТИВНЕ НАВЧАННЯ, НЕРУХОМІ ТОЧКИ, ОПТИМІЗАЦІЯ, ЗБІЖНІСТЬ, ГРАДІЄНТНИЙ МЕТОД.

Об'єктом дослідження є алгоритми федеративного навчання, їх різновиди та властивості. Пошук нерухомих точок за допомогою простої ітерації.

Мета роботи: перевірити ефективність модифікацій до існуючий алгоритмів федеративного навчання на основі метода простої ітерації.

Результат: огляд понятть потрібних для аналізу алгоритмів. Оцінка ефективності існуючих алгоритмів федеративного навчання, що базуються на знаходженні нерухомих точок за допомогою метода простої ітерації. Розробка модифікацій існуючих алгоритмів та перевірка їх ефективності.

Зміст

1 Вступ

Ця робота про федеративне навчання. Федеративне навчання, це техніка машинного навчання, яка дозволяє навчати модель на децентралізованих пристроях з локальними даними, без обміну цими даними.

Класичне машинне навчання навчання різнится з федеративним, можливістю зібрати дані на одному пристрої.

У класичному навчанні, ми маємо доступ до пристрою з повним датасетом, на якому навчаємо модель. У федеративному, ми маємо доступ лише до частинок датасету, які зберігаються локально на пристроях. Причому ці пристрої не можуть передавати дані між собою, тому немає можливості зібрати весь датасет на одному з них.

Федеративним навчання займаються, бо проблема неможливості зібрати всі дані на одному пристрої — актуальна. Одним з класичних прикладів такої конфігурації, є мобільні телефони. Кожен телефон містить персональні дані користувача, які згідно з політикою конфіденційності, компанії не мають права зберігати у себе. Проте використати ці дані для покращення продукту, хочеться. Тому, наприклад Google[5], використовує наступну схему. Ваш пристрій завантажує поточну спільну модель. Навчаючись на даних вашого телефона, він покращує її. Зміни завантаженої моделі підсумовуються, як невелике оновлення. Тільки це оновлення моделі надсилається в хмару, де воно негайно усереднюється з оновленнями інших користувачів для покращення спільної моделі. Як результат, усі дані про залишаються на вашому пристрої, а модель покращується.

Коли зайшов на Wikipedia[6] по запиту "Federated learning"і проглянув джерела, то побачив, що 43 з 44 джерел, належать 2015-2022 року. Причому більше половини, це джерела 2019-2021 року. Це свідчить про те, що дана область активно розвивається в останні роки.

У даній роботі оглядається математика, для роботи з федеративним на-

вчанням. Порівнюються між собою алгоритми запропоновані в статті "From Local SGD to Local Fixed-Point Methods for Federated Learning"[3] та їх модифікації.

Модифікації направленні на отримання кращого результату навчання моделі, при меншій кількості коммунікацій між пристроєм та сервером. Зараз, це один з основних напрямків покращення наявних алгоритмів. Адже, коли в тебе мільйони простроїв, то такі коммунікації – значно уповільнють роботу розподіленої системи.

2 Основні поняття

2.1 Властивості операторів

Визначення 1. Оператор $\mathrm{T} : \mathbb{R}^n \to \mathbb{R}^n$ є **L-ліпиицевим** при $L \geq 0$, якщо $\forall x, y$ виконується нерівність

$$\| \mathrm{T}\, x - \mathrm{T}\, y\| \leq L\|x - y\| \tag{1}$$

Умову (1) називають **умовою Ліпшиця**. Найменше L, що задовольняє нерівність, називають **константою Ліпшиця**.

Якщо $L = 1$, то оператор називають **нерозтягуючим**. Нерозтягуючий оператор, гарантовано, не збільшує відстані між двома довільними точками.

Коли $L < 1$, оператор називають **стискаючим**. Такий оператор завжди зменшуватиме відстань, між двома точками.

Приклад. Найпростішим прикладом нерозтягуючого, є оператор тотожності $\mathrm{Id} : \mathbb{R}^n \to \mathbb{R}^n$. Для нього виконується $\| \mathrm{Id}\, x - \mathrm{Id}\, y\| = \|x - y\|$.

Приклад. Функція $f(x) = \sin(x)$ є нерозтягуючою. Щоб це довести, візмемо дві точки $a < b$. За теоремою про середнє, між ними завжди існуватиме $c \in (a, b)$, така що

$$\frac{f(b) - f(a)}{b - a} = f'(c).$$

Накинемо модулі з обох боків, потім перенесемо знаменник вправо і отримаємо

$$|f(b) - f(a)| = |f'(c)||b - a|$$

Значення $|f'(c)| = |\cos(c)| \leq 1$, для будь-якого c. Тому виконується

$$|f(b) - f(a)| \leq |b - a|$$

, відповідно $\sin(x)$ – нерозтягуюча функція. Аналогічна ситуація з $\cos(x)$.

Визначення 2. Для $\alpha \in (0, 1]$, оператор T називають **α-усередненим**, якщо $\mathrm{T} = \alpha\,\mathrm{N} + (1-\alpha)\,\mathrm{Id}$, для деякого нерозтягуючого оператора N, та оператора тотожності Id.

Фактично, усереднений оператор, це опукла комбінація нерозтягуючого і тотожного оператора. $\left(\frac{1}{2}\right)$-усереднений оператор, називають **строго нерозтягуючим**. Якщо α невідоме, то оператор просто називають **усередненим**.

Визначення 3. x^* – **нерухома точка** оператора T, якщо $\mathrm{T}\,x^* = x^*$. Множина нерухомих точок позначається $\operatorname{Fix}\mathrm{T}$.

Якщо взяти нерозтягуючий T, то з нього можна створити α-усереднений $\overline{\mathrm{T}}$. Для цього треба просто вибрати довільне $\alpha \in (0, 1)$ і за означенням

$$\overline{\mathrm{T}} = \alpha\,\mathrm{T} + (1-\alpha)\,\mathrm{Id}\,.$$

Важливо, що, T і $\overline{\mathrm{T}}$ матимуть однакові фіксовані точки $\operatorname{Fix}\mathrm{T} = \operatorname{Fix}\overline{\mathrm{T}}$. Ця властивість дуже корисна, адже усереднені оператори мають кращі гарантії збіжності, ніж нерозтягуючі. Далі

Властивість $\operatorname{Fix}\mathrm{T} = \operatorname{Fix}\overline{\mathrm{T}}$, легко обгрунтувати. Якщо $\mathrm{T}\,x^* = x^*$, то $\overline{\mathrm{T}}x^* = \alpha x^* + (1-\alpha)x^* = x^*$. Інакше також працює, якщо $\overline{\mathrm{T}}x^* = x^*$, то

$$\begin{aligned}
\overline{\mathrm{T}}x^* &= \alpha x^* + (1-\alpha)\,\mathrm{T}\,x^* \\
x^* &= \alpha x^* + (1-\alpha)\,\mathrm{T}\,x^* \\
(1-\alpha)x^* &= (1-\alpha)\,\mathrm{T}\,x^* \\
x^* &= \mathrm{T}\,x^*.
\end{aligned}$$

Приклад. Оператор $\mathrm{T} : \mathbb{R}^2 \to \mathbb{R}^2$ є $(3/4)$-усередненим з множиною нерухомих

точок $\operatorname{Fix} \mathrm{T} = \{(0, y) \mid y \in \mathbb{R}\}$.

$$\mathrm{T}\, x = \begin{bmatrix} -0.5 & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

Усередненість легко отримати з наступного представлення

$$\mathrm{T}\, x = \left(\frac{3}{4}\mathrm{N} + \frac{1}{4}\mathrm{Id}\right) x = \left(\frac{3}{4}\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) x$$

Де N, нерозтягуючий, бо $\|\mathrm{N}\, x - \mathrm{N}\, y\| = \|x - y\|$. Також легко перевірити, що для $x^* \in \operatorname{Fix} \mathrm{T}$, виконуватиметься $\mathrm{T}\, x^* = x^*$.

Відношення між класами стискаючих, усереднених і нерозтягуючих операторів, можна продемонструвати наступною картинкою.

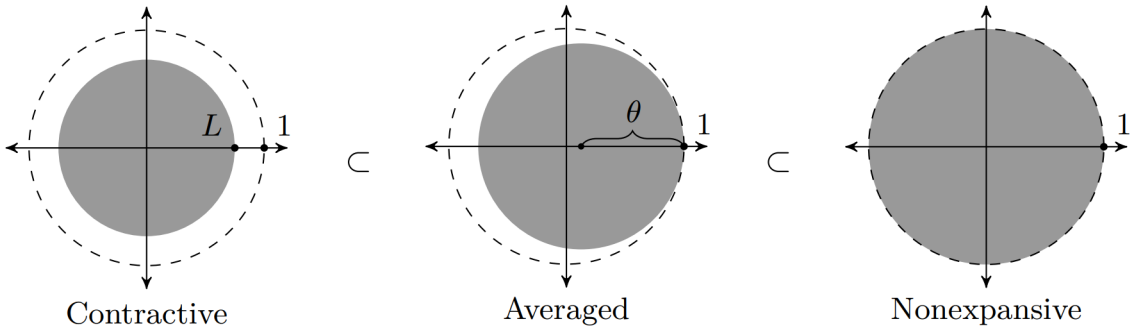


Рис. 1: Стискаючі(Contractive) $\subset$ усереднені(averaged) $\subset$ нерозтягуючі(nonexpansive). Іллюстрація взята з книги [1]

2.2 Проста ітерація (FPI)

Ітераційний процес н

$$x^{k+1} = \mathrm{T}\, x^k$$

називають **простою ітерацією**. У цій нотації, верхній індекс $k = 0, 1, 2 \ldots$, це номер ітерації. Не степінь!

Оператор $\mathrm{T} : \mathbb{R}^n \to \mathbb{R}^n$, x^0 – початкова точка. У англомовній літературі, цей метод простої ітерації називають fixed-point iteration (FPI), від fixed-point – нерухома точка.

Метод простої ітерації, зазвичай, використовують наступним чином:

- знаходять оператор, нерухомі точки якого є розв'язками задачі, що нас цікавить.
- показують, що проста ітерація збігається до нерухомої точки.

Приклад. Ми хочемо знайти корінь рівняння $x^2 - x - 1 = 0$. Для кореня має виконуватись:

$$x^2 = x + 1$$
$$x = 1 + \tfrac{1}{x}.$$

Візмемо $f(x) = 1 + \frac{1}{x}$, тоді пошук кореня – зводиться до пошуку нерухомої точки x^* функції f. Цю точку можна знайти простою ітерацією $x^{k+1} = f(x^k)$. Це і зробимо.

Нехай початок $x^0 = 3$, тоді ітераційний процесс буде

$$x^1 = 1.333, x^2 = 1.75, x^3 = 1.571, \ldots, x^{10} = 1.618, \ldots$$

Як видно, він збігається до одного з коренів рівняння, а саме $\frac{1}{2} + \frac{\sqrt{5}}{2} \approx 1.6180$.

Зрозуміло, що тут постає ціла низька питаннь про збіжність. Чи з будь-якої початкової точки вона буде? Який оператор забезпечує кращу збіжність? І так далі...

2.3 Збіжність простої ітерації

У загальному випадку, метод простої ітерації не збіжний. Але, за рахунок властивостей оператора T, ми можемо отримати деякі гарантії.

Твердження 1. *Якщо оператор* $\mathrm{T} : \mathbb{R}^n \to \mathbb{R}^n$ *стискаючий, то існує єдина нерухома точка* x^* *та проста ітерація* $x^{k+1} = \mathrm{T}\, x^k$ *збігатиметься до* x^*.

Доведення. Ми працюємо в метричному просторі $\left(\mathbb{R}^n, \sqrt{\sum_{i=1}^n (x_i - y_i)^2}\right)$, він є повним. Тому за теоремою Банаха (7.1), існуватиме єдина нерухома точка.

Маємо просту ітерацію $x^{k+1} = \mathrm{T}x^k$. Так як T стискаючий при $L \in (0, 1)$ а $\| \mathrm{T}\, x^k - \mathrm{T}\, x^{k-1}\| = \|x^{k+1} - x^k\|$ матимемо, що

$$\|x^{k+1} - x^k\| \leq L\|x^k - x^{k-1}\|.$$

Тобто з кожною ітерацією відстань $\|x^{k+1} - x^k\|$ зменшується. Враховуючи, що $\|x^{k+1} - x^k\| = \| \mathrm{T}\, x^k - x^k\|$ матимемо $\lim\limits_{k\to\infty} \| \mathrm{T}\, x^k - x^k\| = 0$

Для нерухомої точки x^* виконується $\| \mathrm{T}\, x^* - x^*\| = 0$. Відповідно чим ближче $\| \mathrm{T}\, x^k - x^k\|$ до нуля, тим ближче x^k до нерухомої точки. З того, що границя $\| \mathrm{T}\, x^k - x^k\|$ по k, прямує до нуля, отримаємо

$$x^k \to x^*.$$

□

Нажаль, зазвичай наш оператор не є стискаючим. Це дуже сильна умова. Більш доступною умовою є усередненість. Просту ітерацію для усередненого оператора, називають ітерацією Красносельського-Манна. Для такого є теоремка.

Теорема 2.1. *Якщо* $\mathrm{T} : \mathbb{R}^n \to \mathbb{R}^n$ *і* T – α*-усереднений з* $\alpha \in (0, 1)$ *і множина* $\mathrm{Fix}\,\mathrm{T}$ *непорожня. Тоді матимемо:*

1. *Ітерація* $x^{k+1} = \mathrm{T}\, x^k$ *з будь-якої початкової точки* x^0 *збігатиметься* $x^{k+1} \to x^*$ *до однієї з фіксованих точок* $x^* \in \mathrm{Fix}\,\mathrm{T}$.

2. $\forall x^*$ *відстань* $\left\|x^{k+1} - x^*\right\|$ *зменшується з кожним кроком*

$$\left\|x^{k+1} - x^*\right\|^2 - \left\|x^k - x^*\right\|^2 \leq -\frac{1-\alpha}{\alpha}\left\|x^{k+1} - x^k\right\|^2.$$

3. Швидкість збіжності алгоритму

$$\left\|x^{k+1} - x^k\right\|^2 \leq \frac{\alpha}{(k+1)(1-\alpha)}\|x^0 - x^*\|^2$$

$$\left\|x^{k+1} - x^k\right\|^2 = o(1/k)$$

1 та *3* обгрунтовують в розділі 2.4.2 підручника [1]. *2* це наслідок з твердженнь 4.46, 4.42 та 4.35 в книзі [2].

Приклад. Для оператора T з прикладу (2), ця теорема працює, бо він є (3/4)-усередненим. Для перевірки я взяв 3 різни точки і запустив просту ітерацію $x^{k+1} = \mathrm{T}\, x^k$. Далі, отримані результати.

- Початкова точка $x^0 = \begin{bmatrix} 8 \\ 5 \end{bmatrix}$. Для неї маємо ітерацію:

$$x^1 = \begin{bmatrix} -4 \\ 5 \end{bmatrix}, x^2 = \begin{bmatrix} 2 \\ 5 \end{bmatrix}, x^3 = \begin{bmatrix} -1 \\ 5 \end{bmatrix}, \ldots, x^{10} = \begin{bmatrix} 0,008 \\ 5 \end{bmatrix}, \ldots$$

$$x^k \to \begin{bmatrix} 0 \\ 5 \end{bmatrix} \quad \text{при} \quad k \to \infty$$

- Початкова точка $x^0 = \begin{bmatrix} -0.5 \\ 55 \end{bmatrix}$

$$x^1 = \begin{bmatrix} 0.25 \\ 55 \end{bmatrix}, x^2 = \begin{bmatrix} -0.125 \\ 55 \end{bmatrix}, x^3 = \begin{bmatrix} 0.0625 \\ 55 \end{bmatrix}, \ldots, x^{10} = \begin{bmatrix} 2 \cdot 10^{-4} \\ 55 \end{bmatrix}, \ldots$$

$$x^k \to \begin{bmatrix} 0 \\ 55 \end{bmatrix} \quad \text{при} \quad k \to \infty$$

- Початкова точка $x^0 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$$x^1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, x^3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \ldots, x^{10} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \ldots$$

Ми почали одразу з нерухомої точки, тому тут з кожною ітерацією значення не змінюється.

Загалом можна помітити, що починаючи з довільної точки $x^0 = \begin{bmatrix} a \\ b \end{bmatrix}$, проста ітерація збігатиметься до нерухомої точки $x^* = \begin{bmatrix} 0 \\ b \end{bmatrix}$.

2.4 Композиція операторів

Теорема 2.2. *Нехай* T_1 *і* T_2 *це* $\alpha_1, \alpha_2 \in (0, 1)$ *усереднені відповідно оператори. Кожен з них* $\mathbb{R}^n \to \mathbb{R}^n$*. Тоді їх композиція* $\mathrm{T}_1\,\mathrm{T}_2$ *буде* β*-усередненим оператором при*

$$\beta = \frac{\alpha_1 + \alpha_2 - 2\alpha_1\alpha_2}{1 - \alpha_1\alpha_2} \tag{3}$$

Доведення цієї теореми є у книзі [1], розділ 13.4.1.

На основі цієї теорему, можна отримати наступне твердження. Є корисним, коли маємо справу з багаторазовою композицією операторів однієї усередненості.

Твердження 2. *Нехай* $\mathrm{T} : \mathbb{R}^n \to \mathbb{R}^n$ *це* α*-усереднений оператор. Тоді його*

композиція $\mathrm{T}^k = \underbrace{\mathrm{T}\,\mathrm{T}\ldots\mathrm{T}}_{k}$ *буде* β*-усередненим оператором, при*

$$\beta = \frac{k\alpha}{1+(k-1)\alpha}, \quad k \in \mathbb{N} \tag{4}$$

Беспосередньо, багаторазова композиція виникає у простій ітерації $x^{k+1} = \mathrm{T}\,x^k$. За початкової точки x^0, наступні елементи це

$$\begin{aligned} x^1 &= \mathrm{T}\,x^0 \\ x^2 &= \mathrm{T}\,\mathrm{T}\,x^0 \\ x^3 &= \mathrm{T}\,\mathrm{T}\,\mathrm{T}\,x^0 \\ &\vdots \\ x^{k+1} &= \mathrm{T}^{k+1}\,x^0 \\ &\vdots \end{aligned}$$

2.5 Градієнтні оператори

Проста ітерація виду

$$x^{k+1} = x^k - \gamma\nabla f\left(x^k\right) \tag{5}$$

називається градієнтним методом, або градієнтним спуском. Значення γ – крок градієнтного методу. Оператор такої ітерації, це $\mathrm{T} = \mathrm{Id} - \alpha\nabla f$. Для аналізу градієнтних методів корисне наступне визначення.

Визначення 4. Градієнт ∇f називається L-ліпшицевим , якщо при $L > 0$

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\|, \forall x, y \tag{6}$$

Якщо у функції L-ліпшицевий градієнт, то саму функцію називають L-**гладкою**.

До уваги теорему [2.1], про збіжність усереднених операторів. Наступне твердження пов'язує її зі збіжністю градієнтного методу.

Твердження 3. *Градієнтний оператор* $\mathcal{F} = \mathrm{Id} + \frac{1}{L}\nabla f$ *з* L*-ліпшицевим градієнтом, буде* $(\frac{1}{2})$*-усередненим.*

Відповідно теореми [2.1], градієнтний метод з оператором $\mathcal{F} = \mathrm{Id} + \frac{1}{L}\nabla f$ збігатиметься до фіксованої точки $x^* = \mathcal{F}x^*$, якщо така існує $\mathrm{Fix}\,\mathcal{F} \neq \emptyset$. Таким чином ми отримали гарний варіант вибору кроку $\gamma = \frac{1}{L}$, градієнтного методу. Тепер треба зрозуміти, як можна оцінити значення L, ліпшицевого градієнту.

Твердження 4. *Якщо функція* $f : \mathbb{R}^n \to \mathbb{R}^n$ *опукла та двічі диференційована. Якщо* $\nabla^2 f$ *це невід'ємновизначена матриця, тоді*

$$\|\nabla^2 f\| \leq L \iff \|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{7}$$

Норму $\|\nabla^2 f\|$, можна оцінити зверху найбільшим власним значенням $\nabla^2 f$. Це класичний шлях до підходу вибору кроку γ.

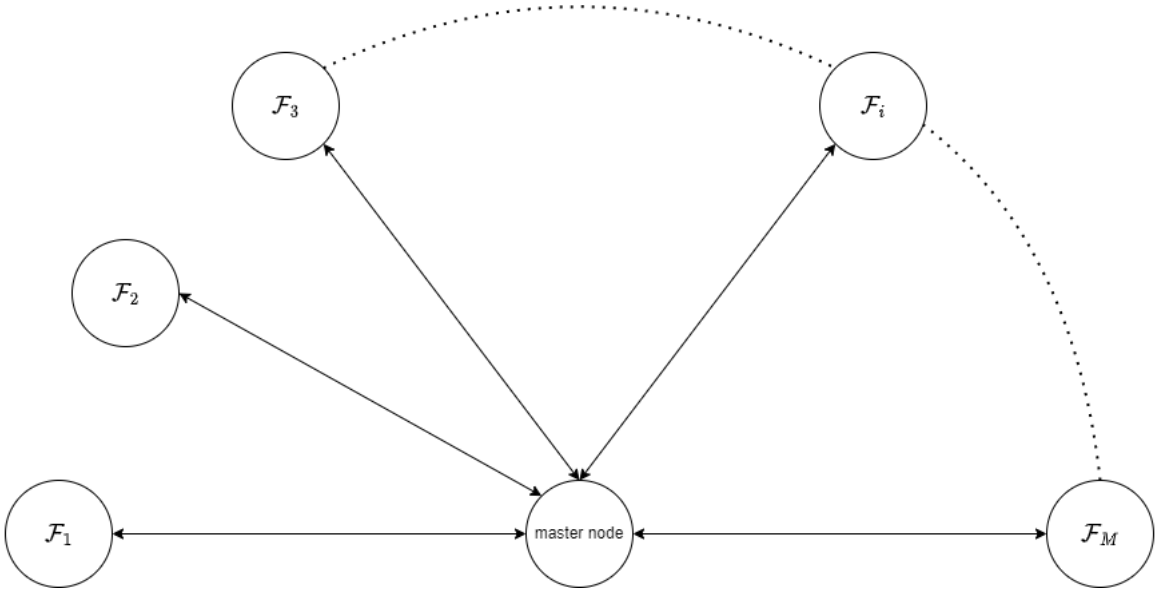
На практиці, для визначення кроку, використовують "Backtracking line search"[7]. Він вважається одним з найкращих методів. Коротко, він працює наступним чином[7, 8]:

- Починають з великого значення γ.
- Ділять γ навпіл, поки не виконуватиметься $f(x - \gamma\nabla f) \leq f(x) - \gamma\lambda\|\nabla f\|^2$. λ зазвичай обирають невелике, 0.1 або навіть 0.0001. Коли умова виконана, то поточне значення γ вибирається кроком методу.

3 Розподілена система

3.1 Загальний опис

До цього, ми розглядали один ітеаційний процесс $x^{k+1} = \mathrm{T}\, x^k$. Один ітераційний процесс, більше притаманний класичному машинному навчанню. У федеративному навчанні ж, таких ітераційних процессів багато. Кожен з них відбувається на своєму пристрої / вузлі / ноді, зі своїми локальними даними. Наступна картинка ілюструє цю архітектуру.



Тут $\mathcal{F}_i$ – локальний оператор, який задає просту ітерацію на пристої i. Якщо пристроїв M, то ітераційний процесс кожного з них можна описати так.

$$x_i^{k+1} = \mathcal{F}_i\, x_i^k$$
$$x_i^{k+1} = x_i^k + \gamma_i \nabla f_i(x_i^k)$$
$$i = 1, 2, \ldots, M$$

Тут видно, що кожний пристрій має свій поточний розв'язок x_i^k, свою функцію f_i, свій крок γ_i.

У федеративному навчанні, наша мета отримати "інтегральний"розв'язок. Розв'язок, який би підходив для всієї сукупності данних, а не для кожної окремої ноди. Для цього, з якоюсь періодичністю, всі ноди обмінюються

своїми розв'язками з master node. master node їх усереднює, і далі усереднений розв'язок транслює на всі ноди. Тобто, після усереднення, всі ноди отримують отримують один поточний розв'язок. Після отримання, кожна нода йде своїм шляхом, крокуючи своїм кроком γ_i, у бік свого градієнта ∇f_i. Так відбувається до моменту повторного усереднення. Тут важливо, що обмінюємося ми тільки розв'язками, обміну даними нема. Вони зберігаються локально.

Періодичність зв'язку всіх нод з master node, може задаватись по-різному. Один з найпростіших варіантів, це створити послідовність моментів t_n, в кожний з котрих ноди зв'язуватимуться з master. У реальному житті, послідовність може бути погодинна. Наприклад, і-тий пристрій кожну відправляє свій поточний розв'язок. Після відправки, з якоюсь затримкою, йому приходить усереднений розв'язок, з якого він і продовжує своє обчислення. Інший варіант зв'язку, це коли на сервері стоїть свій лічільник, який у певний момент часу з ймовірністю p відсилає сигнал всім пристроям на збір їх поточних розв'язків. У цій роботі ми матимемо алгоритм для обох варіантів зв'язку.

3.2 Математичний огляд

Розглянемо детально варіант, коли усереднення відбувається у певну послідовність моментів t_n[3]. Ця послідовність, задаватиметься періодичністю $H > 0$.

$$t_n = nH, n \in \mathbb{N}$$

У нас є буде загальний лічильник ітерацій $k = 0, 1, 2, \ldots$. У момент, коли $k \in t_n$ відбуватиметься усереднення

$$\overline{x}^k = \frac{1}{M} \sum_{i=1}^{M} x_i^k.$$

Після усереднення, наступна ітерації ноди i виглядатиме так

$$x_i^{k+1} = \overline{x}^k + \gamma_i \nabla f_i(\overline{x}^k)$$

. Для подальшого аналізу, запишемо оператор, що застосовується кожні H кроків.

$$\overline{\mathcal{F}} = \frac{1}{M} \sum_{i=1}^{M} (\mathcal{F}_i)^H \tag{8}$$

$\mathcal{F}_i$ – оператор, що відповідає за наступну ітерацію кожної ноди. H – композиція оператора з самим собою.

За періоду H, моменти зв'язку виглядатимуть $t_n = \{H, 2H, 3H, \ldots, nH, \ldots\}$. За допомогою оператора $\overline{\mathcal{F}}$, ітераційний процесс всіх нод можна подати у вигляді:

$$\overline{x}^{(n+1)H} = \overline{\mathcal{F}}(\overline{x}^{nH})$$

Теорема 3.1. *Нехай $t_n = \{H, 2H, 3H, \ldots, nH, \ldots\}$ і кожен з операторів $\mathcal{F}_i$ є α-усередненим. Тоді виконується наступне:*

- *Послідовність $\overline{x}^{(n+1)H}$ збігається до фіксованої точки x^* оператора $\overline{\mathcal{F}}$.*
- *Оператор $\overline{\mathcal{F}}$ є β-усередненим, з $\beta = \frac{H\alpha}{1+(H-1)\alpha}$.*
- *Відстань між $\overline{x}^{nH}$ та фіксованою точкою x^*, зменшується з кожним усередненням*

$$\left\|\overline{x}^{(n+1)H} - x^*\right\|^2 \leq \left\|\overline{x}^{nH} - x^*\right\|^2 - \frac{1-\beta}{\beta} \left\|\overline{x}^{(n+1)H} - \overline{x}^{nH}\right\|^2.$$

- *Відстань між двома сусідніми точками.*

$$\left\|\overline{x}^{(n+1)H} - \overline{x}^{nH}\right\|^2 \leq \frac{1}{\beta(1-\beta)(n+1)} \left\|\overline{x}^0 - x^*\right\|^2$$

- $$\left\| \overline{x}^{(n+1)H} - \overline{x}^{nH} \right\|^2 = o(1/n)$$

Так як заранння фіксованих точок ми не знаємо, то збіжність алгоритму вимірюють метрикою $\left\| T\left(x^k\right) - x^k \right\|$.

4 Огляд алгоритмів

4.1 Класичні

Algorithm 1 Циклічний зв'язок з master node [3]

Input: stepsize $\lambda > 0$, communication times t_n
Initialize: start point w_i^0 for each node
for $k = 1, 2, ..$ **do**
  for $i = 1, 2, ..., M$ **do**
    if $k - 1 \in t_n$ **then**
      Translate previous step averaging to nodes
      $w_i^{k-1} = \overline{w}^{k-1}$
    end if
    $w_i^k = (1 - \lambda) w_i^{k-1} + \lambda \mathcal{F}_i(w_i^{k-1})$
  end for
  if $k \in t_n$ **then**
    At master node gather w_i^k
    $\overline{w}_i^k = \frac{1}{M} \sum_{i=1}^{M} w_i^k$
  end if
end for

Algorithm 2 Випадковий зв'язок з master node [3]

Input: stepsize $\lambda > 0$, communication probability p
Initialize: start point w_i^0 for each node
for $k = 1, 2, ..$ **do**
  for $i = 1, 2, ..., M$ **do**
    if $t_{k-1} > 1 - p$ **then**
      Translate averaging to nodes
      $w_i^{k-1} = \overline{w}^{k-1}$
    end if
    $w_i^k = (1 - \lambda) w_i^{k-1} + \lambda \mathcal{F}_i(w_i^{k-1})$
  end for
  $t_k =$ generate random $t \in [0, 1]$
  if $t_k > 1 - p$ **then**
    At master node gather w_i^k
    $\overline{w}_i^k = \frac{1}{M} \sum_{i=1}^{M} w_i^k$
  end if
end for

Ідея цих алгоритмів, була описана у розділі [3.2]. Фактично, різниця між ними, лише у підході до часу синхронізації. У першому, є чітка послідовність моментів синхронізації. У другому, синхронізація відбувається з заданою ймовірністю.

4.2 Модифікація алгоритмів

Algorithm 3 Циклічний зв'язок рандомної підмножини пристроїв з master node [3]

Input: stepsize $\lambda > 0$, communication times t_n, percent of nodes for communication s
Initialize: start point w_i^0 for each node
for $k = 1, 2, ..$ **do**
  for $i = 1, 2, ..., M$ **do**
    if $k - 1 \in t_n$ **then**
      Translate previous step averaging to nodes
      $w_i^{k-1} = \overline{w}^{k-1}$
    end if
    $w_i^k = (1 - \lambda)w_i^{k-1} + \lambda \mathcal{F}_i(w_i^{k-1})$
  end for
  if $k \in t_n$ **then**
    Select random $r = \frac{s}{100} \cdot M$ nodes
    We connect all r nodes with master
    Average selected r nodes values
    $\overline{w}^k = \frac{1}{M} \sum_{i=1}^{M} w_i^k$
  end if
end for

Algorithm 4 Випадковий зв'язок рандомної підмножини пристроїв з master node [3]

Input: stepsize $\lambda > 0$, communication probability p
Initialize: start point w_i^0 for each node
for $k = 1, 2, ..$ **do**
  for $i = 1, 2, ..., M$ **do**
    if $t_{k-1} > 1 - p$ **then**
      Translate averaging to nodes
      $w_i^{k-1} = \overline{w}^{k-1}$
    end if
    $w_i^k = (1 - \lambda)w_i^{k-1} + \lambda \mathcal{F}_i(w_i^{k-1})$
  end for
  t_k = generate random $t \in [0, 1]$
  if $t_k > 1 - p$ **then**
    Select random $r = \frac{s}{100} \cdot M$ nodes
    We connect all r nodes with master
    Average selected r nodes values
    $\overline{w}^k = \frac{1}{M} \sum_{i=1}^{M} w_i^k$
  end if
end for

Ця модифікації двох попередніх алгоритмів, направлені на зменшення кількості нод, що зв'язуються з master. Тут ми задаємо додатковий параметр s. Це відсоток від всіх пристроїв, які зв'язуються з master. При кожному усередненні, вибирається навмання r нод. Очікується, що завдяки цьому

алгоритму можна суттє зекономити на кількості зв'язків, не сильно втрачаючи в швидкості збіжності. Результати роботи наведені у наступный секції.

Algorithm 5 Випадковий зв'язок з master node, певного відсотку ознак з найбільшим значенням.

Input: stepsize $\lambda > 0$, communication probability p, percent of largest components d
Initialize: start point w_i^0 for each node
for $k = 1, 2, ..$ **do**
  for $i = 1, 2, ..., M$ **do**
    if $t_{k-1} > 1 - p$ **then**
      Translate averaging to nodes
      $w_i^{k-1} = \overline{w}^{k-1}$
    end if
    $w_i^k = (1 - \lambda)w_i^{k-1} + \lambda \mathcal{F}_i(w_i^{k-1})$
  end for
  t_k = generate random $t \in [0, 1]$
  if $t_k > 1 - p$ **then**
    At master node gather w_i^k
    For each w_i^k saves values of $d\%$ largest positions. The others makes 0.
    $\overline{w}_i^k = \frac{1}{M} \sum_{i=1}^{M} w_i^k$
  end if
end for

Algorithm 6 Циклічний зв'язок рандомної підмножини пристроїв з master node [3]

Input: stepsize $\lambda > 0$, communication times t_n, percent of nodes for communication s
Initialize: start point w_i^0 for each node
for $k = 1, 2, ..$ **do**
  for $i = 1, 2, ..., M$ **do**
    if $k - 1 \in t_n$ **then**
      Translate previous step averaging to nodes
      $w_i^{k-1} = \overline{w}^{k-1}$
    end if
    $w_i^k = (1 - \lambda)w_i^{k-1} + \lambda \mathcal{F}_i(w_i^{k-1})$
  end for
  if $k \in t_n$ **then**
    Select random $r = \frac{s}{100} \cdot M$ nodes
    We connect all r nodes with master
    Average selected r nodes values
    $\overline{w}^k = \frac{1}{M} \sum_{i=1}^{M} w_i^k$
  end if
end for

Ці два алгоритми відрізняються від перших двох тим, що тут кожен пристрій не передає свій повний розв'язок. Він передає частину, що складається з певного відсотку найбільших по модулю компонент. Цей відсоток задається

на початку роботи програми. Наприклад, якщо розв'язок складається з 10 компонент, і ми вибрали передавати 50%, то master ноді передасться лише половина значень цього вектору. Причому, ця половина підібрана так, що містить у собі 50% найбільших значень вектора. Ми обираємо певний відсоток саме найбільших значень, бо вони вносять основний вклад в значення градієнту.

Загалом, ці дві модифікації ставлять своєю метою зменшення об'єму переданої інформації. У своє чергу це допоможе пришвидшити швидкість зв'язку між сервером там пристроєм.

5 Експеримент

Обчислювальний експеримент в перевірці алгоритмів [1, 2, 3, 4, 5, 6] на одному й тому ж датасеті. Мене цікавить, щоб при меншій кількості зв'язків з master node, алгоритм швидше збігався. У якості оцінки збіжності я піду природнім шляхом, взявши $\left\|\bar{x}^{(n+1)H} - \bar{x}^{nH}\right\|$.

Модель

У своєму експерименті я використовував логістичну регрессію. Це одна з найбільш популярних моделей, для розв'язку задач классифікації. У даному випадку, задачі бінарної классифікації.

На кожному пристрої є своя функція помилки f. За структурою, ця функція однакова для всіх пристроїв. Локальний датасет складається зі спостережень x_i. Відповіді на спостереженнях це $y_i \in \{-1, +1\}$. Коефіцієнти всередині регресії записані у вектор w.

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i w^T x_i\right)\right) + \lambda\|x\|^2. \tag{9}$$

На кожному пристрої відбувається ітераційний процесс

$$x^{k+1} = x^k - \gamma \nabla f(x^k) \tag{10}$$

де γ це розмір кроку.

При програмній реалізації, я скористувався пакетом *sklearn*[10]. Кількість нод $M = 10$.

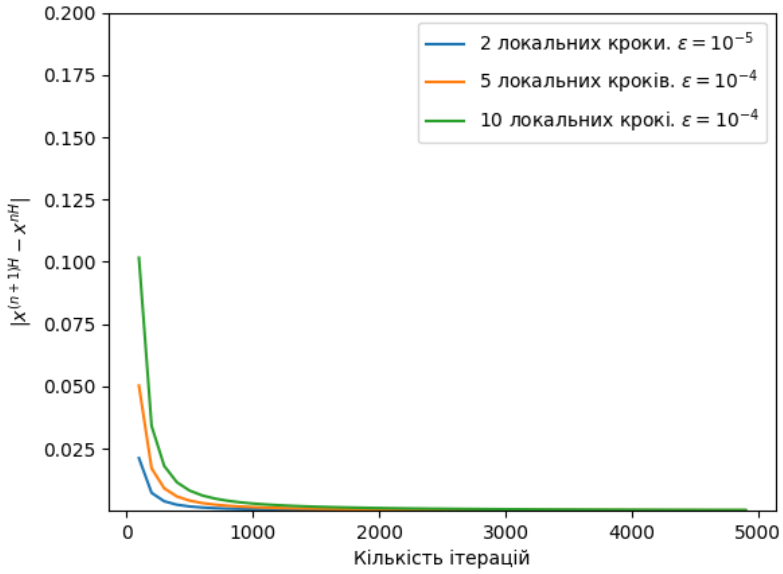
Дані

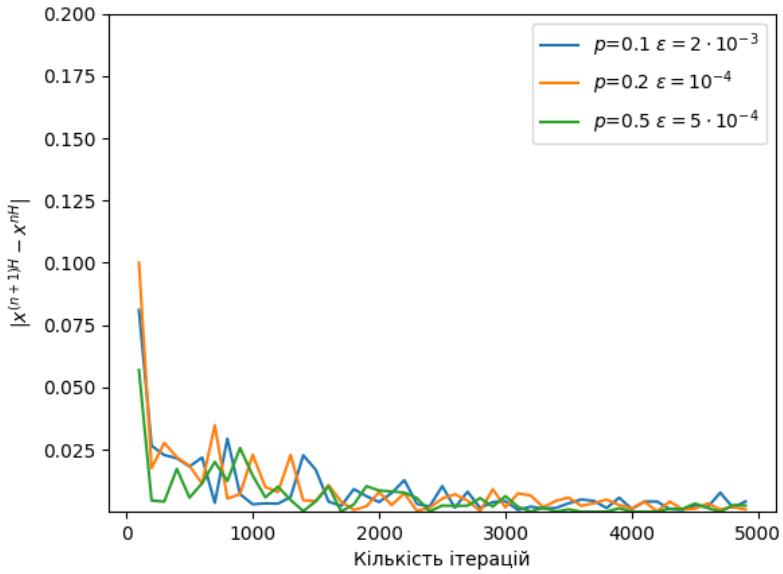
Я використовував датасет *a9a*. Він має 32 000 спостережнь з 123 параметрами. Призначений для тестування методів бінарної класифікації, тому

йде одразу передоброблений. Детальніше про нього тут[9].

Алгоритм 1-2

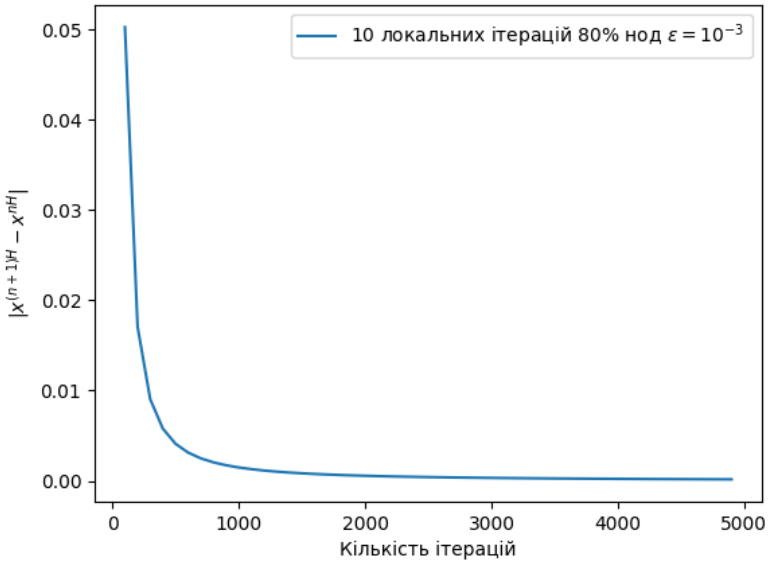
Алгоритми вже продемонстрували свою ефективність в статті [3]. Єдине, на чому варто зауважити увагу, що циклічний зв'язок з master node показує себе більш стабільним.

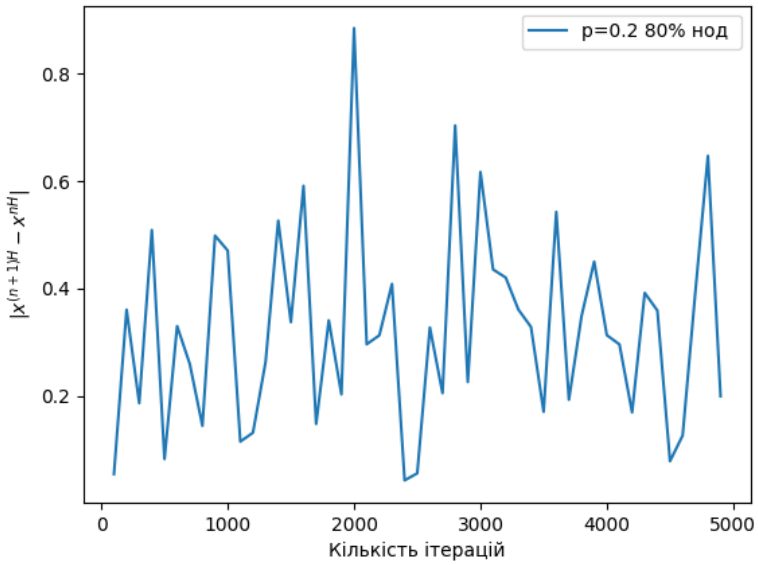




Алгоритм 3-4

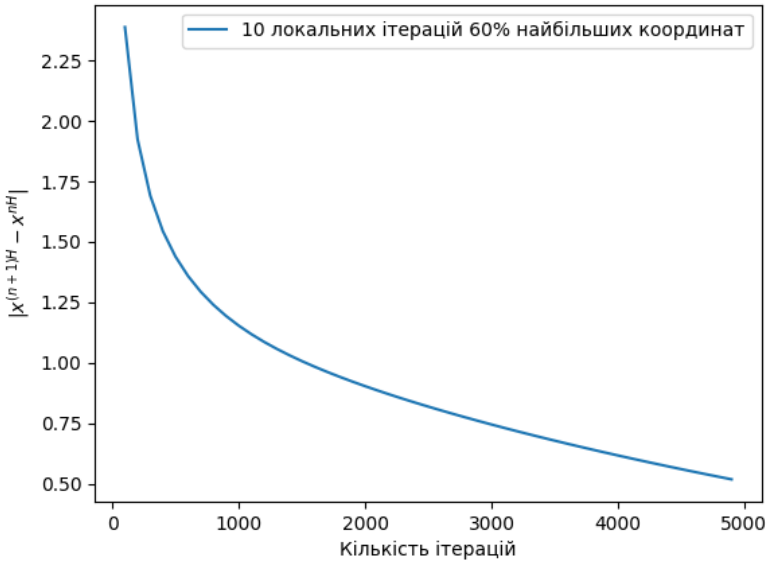
Модифікація, коли ми зв'язвуємо не всі ноди, а лише певний їх відсоток (підмножину). Причому підмножина вибирається випадково, з поверненням. Тобто, зараз у цій множині можуть бути присутніми дві однакові ноди. По графіку, алгоритм циклічного зв'язку видається стабільним, з гарною точністю. Алгоритм з випадковим зв'язком навпаки, волатильний. До того ж точність набагато менша.





Алгоритм 5-6

Модифікація, коли ми передаємо ноді не весь вектор, а лише його частину. Вона виявилась поганою. Насамперед, через сильне погіршення швидкості збіжності. У моїй реалізації, така економія точно не вартує подібної втрати точності.



6 Висновок

Алгоритм 1,2 були представлені у роботі [3], там показана їх ефективність. При відтворенні, я отримав схожу поведінку.

Алгоритм 3,4, це модифікаці 1, 2 що мають на меті зменшити кількість раундів коммунікації між master node і пристроями, не втративши помітно у точності. На представленому графіку алгоритма 3, обмін з сервером проводили 80% від всіх нод. При такому відсотку точність, постраждала не сильно. При цьому кількість пристроїв, що відправляють дані до сервера, зменшилася на 20%. У подальшій розробці цієї теми, ці алгоритми є сенс додатково дослідити. Протестувати, як зміниться їх поведінка при стрімкому збільшенні кількості нод. Як зміниться поведінка, якщо набирати ноди без повернення? Як алгоритм поводиться на різних датасетах?

Алгоритм 5,6 мали на меті зменшити об'єм даних, який кожен пристрій передає серверу. По експерименту, втрати точності виявилися дуже великими. Цей алгоритм є можна дослідити додатково. Рухатися у бік представлення даних, наприклад квантилізації. Коли дійсні ознаки розбиваються на проміжки, і далі бінарним значенням вказуємо проміжок знаходження. Можливо, коли кількість ознак значно зросте, то легше буде знехтувати значеннями.

7 Додаток А

Теорема 7.1 (Банаха)**.** *Будь-яке стискаюче відображення повного метричного простору* (X, d) *в себе має лише одну нерухому точку.*

Твердження 5. *Нехай* $\mathrm{T} : \mathbb{R}^n \to \mathbb{R}^n$ *це* α*-усереднений оператор. Тоді його композиція* $\mathrm{T}^k = \underbrace{\mathrm{T}\,\mathrm{T}\ldots\mathrm{T}}_{k}$ *буде* β*-усередненим оператором, при*

$$\beta = \frac{k\alpha}{1 + (k-1)\alpha}, \quad k \in \mathbb{N} \tag{11}$$

Доведення. Доведемо за допомогою математичної індукції.
$k = 1$. $\beta = \frac{\alpha}{1+(1-1)\alpha} = \alpha$.

$k = 2$. $\beta = \frac{2\alpha}{1+\alpha}$. Це збігається з результатом (3), якщо $\alpha_1 = \alpha_2$.

Індукційний перехід. Нехай твердження працює для $k-1$, тоді користуючись

$\mathrm{T}^k = \mathrm{T}^{k-1}\,\mathrm{T}$ і результатом (3) останньої теореми, матимемо:

$$\begin{aligned}
\beta &= \frac{\frac{(k-1)\alpha}{1+(k-2)\alpha} + \alpha - \frac{2(k-1)\alpha}{1+(k-2)\alpha}\cdot\alpha}{1 - \frac{(k-1)\alpha}{1+(k-2)\alpha}\cdot\alpha} \\
&= \frac{\alpha\left[\frac{k-1}{1+(k-2)\alpha} + 1 - \frac{2(k-1)\alpha}{1+(k-2)\alpha}\right]}{\frac{1+(k-2)\alpha-(k-1)\alpha^2}{1+(k-2)\alpha}} \\
&= \frac{\alpha\left[\frac{k-1+\quad 1+(k-2)\alpha\quad -2(k-1)\alpha}{1+(k-2)\alpha}\right]}{\frac{1+(k-2)\alpha-(k-1)\alpha^2}{1+(k-2)\alpha}} \\
&= \frac{\alpha\left[k+(k-2)\alpha-2(k-1)\alpha\right]}{1+(k-2)\alpha-(k-1)\alpha^2} \\
&= \frac{k\alpha - k\alpha^2}{1-\alpha+(k-1)\alpha-(k-1)\alpha^2} \\
&= \frac{k\alpha(1-\alpha)}{1-\alpha+(k-1)\alpha(1-\alpha)} \\
&= \frac{k\alpha(1-\alpha)}{(1-\alpha)(1+(k-1)\alpha)} \\
&= \frac{k\alpha}{1+(k-1)\alpha}.
\end{aligned}$$

Довели, що це працює для k. Доведення за математичною індукцією, завершено. □

Твердження 6. *Градієнтний оператор* $\mathcal{F} = \mathrm{Id} + \alpha\nabla f$ *де* $\alpha = \frac{1}{L}$ *при* L*-ліпшицевому* ∇f *є* $(\frac{1}{2})$*-усередненим.*

Доведення. $\mathrm{Id} + \alpha\nabla f = \frac{1}{2}\,\mathrm{Id} + \frac{1}{2}(\mathrm{Id} - \frac{2}{L}\nabla f)$. Треба показати, що $\mathrm{Id} - \frac{2}{L}\nabla f)$ є нестискаючим. Використаємо, що

$$\|\,\mathrm{T}\,x - \mathrm{T}\,y\|^2 \le \|x-y\|^2 \implies \|\,\mathrm{T}\,x - \mathrm{T}\,y\| \le \|x-y\|.$$

Відповідно

$$\begin{aligned}
\| \mathcal{F}\, x - \mathcal{F}\, y\|^2 &= \left\| x - \frac{2}{L}\nabla f(x) - \left(y - \frac{2}{L}\nabla f(y) \right) \right\|^2 \\
&= \| \, (x-y) - \frac{2}{L} \left(\nabla f(x) - \nabla f(y) \right) \|^2 \\
&= \|x-y\|^2 - \frac{4}{L}(x-y)^T(\nabla f(x) - \nabla f(y)) + \frac{4}{L^2}\|\nabla f(x) - \nabla f(y)\|^2 \\
&= \|x-y\|^2 - \frac{4}{L}\left((x-y)^T(\nabla f(x) - \nabla f(y)) - \frac{1}{L}\|\nabla f(x) - \nabla f(y)\|^2 \right) \\
&\text{використаємо } (x-y)^T \left(\nabla f(x) - \nabla f(y) \right) \leq L\|x-y\|^2 \text{ з [4]} \\
&\leq \|x-y\|^2 - \frac{4}{L}\left(L\|x-y\|^2 - \frac{1}{L}\|\nabla f(x) - \nabla f(y)\|^2 \right) \\
&\text{використаємо } \|\nabla f(x) - \nabla f(y)\|^2 \leq L^2\|x-y\|^2 \\
&\leq \|x-y\|^2 - \frac{4}{L}\left(L\|x-y\|^2 - \frac{1}{L}L^2\|x-y\|^2 \right) = \|x-y\|^2
\end{aligned}$$

□

8 Додаток В

```
import pandas as pd
import numpy as np
import random as rn
import copy
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

def alg1(dataframe, target_name,
                n_nodes,
                n_iter,
                conn_step  # each conn_step iteration is comm round
                ):

    parts = np.array_split(dataframe, n_nodes)

    models = []  # have unique model for each node
    for _ in range(n_nodes):
        sgd = SGDClassifier(loss='log',
        learning_rate="optimal",
        penalty="l2",
        alpha=0.0001,
        tol=0.0000001,
        shuffle=False)
        models.append(sgd)

    diff_approx_w = 0  # // w_k+1 - w_k //
    ws_old = np.array([])
    ws_avg = np.array([])

    k_n = np.arange(n_iter)
    t_n = np.arange(start=conn_step, stop=n_iter,
                    step=conn_step)  # communication times
```

```
print_k = np.arange(100, n_iter, 100)  # print times
for k in k_n:
    ws = []  # wieghts
    inters = []  # intercepts

    if k in print_k:
        diff_approx_w = np.linalg.norm(
        np.array(ws_old) - np.array(ws_avg)
        )
        print(f"{diff_approx_w},")

    if k in t_n:
        for i in range(n_nodes):
            ws.append(models[i].coef_[0])
            inters.append(models[i].intercept_)

        ws_old = copy.deepcopy(ws_avg)

        ws_avg = np.array(ws).sum(axis=0) /
            n_nodes
        inters_avg = np.array(inters).sum(axis=0) /
            n_nodes

        for i in range(n_nodes):
            models[i].coef_, models[i].intercept_ = (
                ws_avg, inters_avg
                )

    for i in range(n_nodes):
        X, y = (
            parts[i].drop(columns=target_name),
            parts[i][target_name]
        )
        models[i].partial_fit(X, y, classes=np.unique(y))
```

```
    return [ws_avg, inters_avg, diff_approx_w]

def alg2(dataframe, target_name,
                n_nodes,
                n_iter,
                conn_prob  # probability of communication
                ):

    assert 0 < conn_prob <= 1, "probability between 0 and 1"
    assert n_iter > 0
    assert n_nodes > 0

    parts = np.array_split(dataframe, n_nodes)

    models = []  # have unique model for each node
    for _ in range(n_nodes):
        sgd = SGDClassifier(loss='log',
        learning_rate="optimal",
        penalty="l2",
        alpha=0.0001,
        tol=0.0000001,
        shuffle=False)
        models.append(sgd)

    diff_approx_w = 0  # // w_k+1 - w_k //
    ws_old = np.array([])
    ws_avg = np.array([])

    rn_key = 0
    k_n = np.arange(n_iter)
    print_k = np.arange(100, n_iter, 100)
    for k in k_n:

        ws = []  # wieghts
        inters = []  # intercepts
```

```
        if k in print_k:
            diff_approx_w = (
                np.linalg.norm(np.array(ws_old) - np.array(ws_avg))
                    )
            print(f"{diff_approx_w},")

        rn_key = rn.random()
        if rn_key > 1-conn_prob:
            for i in range(n_nodes):
                ws.append(models[i].coef_[0])
                inters.append(models[i].intercept_)

            ws_old = copy.deepcopy(ws_avg)

            ws_avg = np.array(ws).sum(axis=0) / n_nodes
            inters_avg = np.array(inters).sum(axis=0) / n_nodes

            for i in range(n_nodes):
                models[i].coef_, models[i].intercept_ = (
                    ws_avg, inters_avg
                )


        for i in range(n_nodes):
            X, y = (
            parts[i].drop(columns=target_name),
            parts[i][target_name]
            )
            models[i].partial_fit(X, y, classes=np.unique(y))

    return [ws_avg , inters_avg, diff_approx_w]

def alg3(dataframe, target_name,
```

```
            n_nodes,
            n_iter,
            conn_step,  # each conn_step iteration is comm round
            subset_percent
    ):

parts = np.array_split(dataframe, n_nodes)

models = []  # have unique model for each node
for _ in range(n_nodes):
    sgd = SGDClassifier(loss='log',
    learning_rate="optimal",
    penalty="l2",
    alpha=0.0001,
    tol=0.0000001, shuffle=False)
    models.append(sgd)

n_subset_nodes = round(n_nodes * subset_percent / 100)
assert n_subset_nodes > 0  # at least one node shares info

diff_approx_w = 0  # || w_k+1 - w_k ||
ws_old = np.array([])
ws_avg = np.array([])

k_n = np.arange(n_iter)
t_n = np.arange(start=conn_step, stop=n_iter,
                step=conn_step)  # communication times
print_k = np.arange(100, n_iter, 100)  # print times
for k in k_n:
    ws = []  # wieghts
    inters = []  # intercepts

    if k in print_k:
        diff_approx_w = (
        np.linalg.norm(np.array(ws_old) - np.array(ws_avg))
```

```
            )
            print(f"{diff_approx_w},")

        if k in t_n:
            for i in range(n_nodes):
                ws.append(models[i].coef_[0])
                inters.append(models[i].intercept_)

            ws_old = copy.deepcopy(ws_avg)

            nodes_id = (
                [rn.randint(0, n_nodes - 1) for _ in range(n_subset_nodes)]
                )
            ws_subs = [ws[i] for i in nodes_id]
            inters_subs = [inters[i] for i in nodes_id]

            ws_avg = np.array(ws_subs).sum(axis=0) / n_nodes
            inters_avg = np.array(inters_subs).sum(axis=0) / n_nodes

            for i in range(n_nodes):
                models[i].coef_, models[i].intercept_ = (
                ws_avg, inters_avg
                )

        for i in range(n_nodes):
            X, y = (
            parts[i].drop(columns=target_name), parts[i][target_name]
            )
            models[i].partial_fit(X, y, classes=np.unique(y))

    return [ws_avg, inters_avg, diff_approx_w]

def alg4(dataframe, target_name,
                n_nodes,
                n_iter,
```

```
            conn_prob,  # probability of communication
            subset_percent):

assert 0 < conn_prob <= 1, "probability between 0 and 1"
assert n_iter > 0
assert n_nodes > 0
assert 0 < subset_percent <= 100

parts = np.array_split(dataframe, n_nodes)

models = []  # have unique model for each node
for _ in range(n_nodes):
    sgd = SGDClassifier(loss='log',
    learning_rate="optimal",
    penalty="l2",
    alpha=0.0001,
    tol=0.0000001,
    shuffle=False)
    models.append(sgd)

n_subset_nodes = round(n_nodes * subset_percent / 100)
assert n_subset_nodes > 0  # at least one node shares info

diff_approx_w = 0  # // w_k+1 - w_k //
ws_old = np.array([])
ws_avg = np.array([])

rn_key = 0
k_n = np.arange(n_iter)
print_k = np.arange(100, n_iter, 100)
for k in k_n:

    ws = []  # wieghts
    inters = []  # intercepts
```

```
if k in print_k:
    diff_approx_w = (
    np.linalg.norm(np.array(ws_old) - np.array(ws_avg))
    )
    print(f"{diff_approx_w},")

rn_key = rn.random()
if rn_key > 1-conn_prob:
    for i in range(n_nodes):
        ws.append(models[i].coef_[0])
        inters.append(models[i].intercept_)

    ws_old = copy.deepcopy(ws_avg)

    nodes_id = (
    [rn.randint(0, n_nodes - 1) for _ in range(n_subset_nodes)]
    )
    ws_subs = [ws[i] for i in nodes_id]
    inters_subs = [inters[i] for i in nodes_id]

    ws_avg = np.array(ws_subs).sum(axis=0) / n_nodes
    inters_avg = np.array(inters_subs).sum(axis=0) / n_nodes

    for i in range(n_nodes):
        models[i].coef_, models[i].intercept_ = (
            ws_avg, inters_avg
        )


for i in range(n_nodes):
    X, y = (
        parts[i].drop(columns=target_name), parts[i][target_name]
    )
    models[i].partial_fit(X, y, classes=np.unique(y))
```

```
    return [ws_avg , inters_avg, diff_approx_w]

def select_largest(arr, k):
    sorted_index_array = np.argsort(arr)
    sorted_array = arr[sorted_index_array]

    assert k <= len(sorted_array)
    largest_k = sorted_array[-k : ]

    j = 0
    for i in range(len(arr)):
        arr[i] += np.random.normal(0, .001)



    for i in range(len(largest_k)):
        if arr[i] not in largest_k:
            arr[i] = 0
        else:
            j += 1

    return arr

def alg5(dataframe, target_name,
                n_nodes,
                n_iter,
                conn_step,  # probability of communication
                features_percent):
        parts = np.array_split(dataframe, n_nodes)
        n_largest = int((features_percent / 100) * (dataframe.shape[1] - 1))

        models = []  # have unique model for each node
        for _ in range(n_nodes):
            sgd = SGDClassifier(loss='log',
            learning_rate="optimal",
            penalty="l2",
```

```
        alpha=0.0001,
        tol=0.0000001, shuffle=False)
        models.append(sgd)

diff_approx_w = 0  # || w_k+1 - w_k ||
ws_old = np.array([])
ws_avg = np.array([])

k_n = np.arange(n_iter)
t_n = np.arange(start=conn_step, stop=n_iter,
                step=conn_step)  # communication times
print_k = np.arange(100, n_iter, 100)  # print times
for k in k_n:
    ws = []  # wieghts
    inters = []  # intercepts

    if k in print_k:
        diff_approx_w = (
        np.linalg.norm(np.array(ws_old) - np.array(ws_avg))
        )
        print(f"{diff_approx_w},")

    if k in t_n:
        for i in range(n_nodes):
            ws.append(select_largest(models[i].coef_[0], n_largest))
            inters.append(models[i].intercept_)

        ws_old = copy.deepcopy(ws_avg)

        ws_avg = np.array(ws).sum(axis=0) / n_nodes
        inters_avg = np.array(inters).sum(axis=0) / n_nodes

        for i in range(n_nodes):
            models[i].coef_, models[i].intercept_ = (
                ws_avg, inters_avg
```

```
                    )

            for i in range(n_nodes):
                X, y = (
                parts[i].drop(columns=target_name),
                parts[i][target_name]
                )

                models[i].partial_fit(X, y, classes=np.unique(y))

        return [ws_avg, inters_avg, diff_approx_w]

def alg6(dataframe, target_name,
                n_nodes,
                n_iter,
                conn_prob, # probability of communication
                features_percent):

    assert 0 < conn_prob <= 1, "probability between 0 and 1"
    assert n_iter > 0
    assert n_nodes > 0
    assert 0 < subset_percent <= 100

    parts = np.array_split(dataframe, n_nodes)
    n_largest = int((features_percent / 100) * (dataframe.shape[1] - 1))

    models = [] # have unique model for each node
    for _ in range(n_nodes):
        sgd = SGDClassifier(loss='log',
        learning_rate="optimal",
        penalty="l2",
        alpha=0.0001,
        tol=0.0000001,
        shuffle=False)
        models.append(sgd)
```

```
n_subset_nodes = round(n_nodes * subset_percent / 100)
assert n_subset_nodes > 0  # at least one node shares info

diff_approx_w = 0  # || w_k+1 - w_k ||
ws_old = np.array([])
ws_avg = np.array([])

rn_key = 0
k_n = np.arange(n_iter)
print_k = np.arange(100, n_iter, 100)
for k in k_n:

    ws = []  # wieghts
    inters = []  # intercepts



    if k in print_k:
        diff_approx_w = (
        np.linalg.norm(np.array(ws_old) - np.array(ws_avg))
        )
        print(f"{diff_approx_w},")

    rn_key = rn.random()
    if rn_key > 1-conn_prob:
        for i in range(n_nodes):
            ws.append(select_largest(models[i].coef_[0], n_largest))
            inters.append(models[i].intercept_)

        ws_old = copy.deepcopy(ws_avg)

        ws_avg = np.array(ws).sum(axis=0) / n_nodes
        inters_avg = np.array(inters).sum(axis=0) / n_nodes

        for i in range(n_nodes):
```

```
                models[i].coef_, models[i].intercept_ = (
                ws_avg, inters_avg
                )


        for i in range(n_nodes):
            X, y = (
            parts[i].drop(columns=target_name),
            parts[i][target_name]
            )
            models[i].partial_fit(X, y, classes=np.unique(y))

    return [ws_avg , inters_avg, diff_approx_w]
```

Література

[1] Ernest K. Ryu & Wotao Yin (2022) *Large-Scale Convex Optimization via Monotone Operators*

[2] Bauschke, H. H. and Combettes, P. L. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, New York, 2nd edition, 2017

[3] Grigory M., Dmitry K., Elnur G., Laurent C., Peter R. (2020) *From Local SGD to Local Fixed-Point Methods for Federated Learning*

[4] Zhou, Xingyu. "On the Fenchel Duality between Strong Convexity and Lipschitz Continuous Gradient." arXiv preprint arXiv:1803.06573 (2018).

[5] Google AI Blog *Federated Learning: Collaborative Machine Learning without Centralized Training Data* `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`.

[6] Wikipedia, *Federated learning* `https://en.wikipedia.org/wiki/Federated_learning`

[7] Nick Goudl *Linesearch methods for unconstrained optimization. MSc course on nonlinear optimization*

[8] Mark Schmidt *Convex Optimization CMPT 419/726*

[9] LIBSVM Data: Classification (Binary Class) `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

[10] *sklearn.linear_model.SGDClassifier* `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.score`

[11] Lipschitz-constant gradient implies bounded eigenvalues on Hessian https://math.stackexchange.com/questions/1698812/lipschitz-constant-gradient-implies-bounded-eigenvalues-on-hessian

[12] Right Bregman *Nonexpansive Operators in Banach Spaces* https://ssabach.net.technion.ac.il/files/2015/12/MRS2012-1.pdf

[13] Ahmed Khaled and Peter Richt´arik *Better Theory for SGD in the Nonconvex World* (2020)

[14] Geoff Gordon and Ryan Tibshirani *Lecture 5: Gradient Desent Revisited* (2012)

[15] Lorenzo Rosasco *Lecture 14- Logistic Regression* Spring 2014

[16] Compositions and Convex Combinations of Averaged Nonexpansive Operators