Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

Курсова робота
за спеціальністю „Прикладна математика” 113

Модель градієнтного бустінгу LightGBM

Керівник курсової роботи

Дрінь Світлана Сергіївна

Старший викладач, кандидат фізико-математичних наук.

_____

(підпис)

Виконала студентка 3-го курсу

спеціальності «Прикладна математика»

Вербівська Юлія Віталіївна

«__» _____ 2023 р.

Київ – 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., д.ф.-м.н.
_____Р. К. Чорней
(підпис)
«___»_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Вербівській Юлії Віталіївні факультету інформатики 3 курсу
Тема: **Модель градієнтного бустінгу LightGBM**

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
1. Огляд градієнтного бустінгу
2. Огляд LightGBM
3. Симуляція роботи LightGBM

Висновки
Список літератури
Додатки

Дата видачі „__” _____ 2022 р. Керівник ____
(підпис)
Завдання отримав _____
(підпис)

Тема: **Модель градієнтного бустінгу LightGBM**

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	затвердження	
2.	Обговорення деталей теми та розробка плану виконання	05.01.2023	
3.	Огляд технічної літератури за темою роботи та опрацювання матеріалів	10.10.2022	
4.	Написання теоретичної частини роботи	1.04.2023	
5.	Розробка практичної частини	1.05.2023	
6.	Аналіз отриманих результатів, обговорення остаточної доповіді з керівником	12.05.2023	
7.	Здача курсової роботи	15.05.2023	

Студент: Вербівська Ю. В.

Керівник: Дрінь С.С.

“__” _____ 2023

Зміст

Перелік прийнятих скорочень

ML – Machine Learning – машинне навчання

XGBoost – Extreme Gradient Boosting

LightGBM – Light Gradient Boosting Machine

GBM – Gradient Boosting Machine

CART – Classification And Regression Tree – дерево класифікації та регресії

IG – Information Gain

MSE – середньоквадратична помилка

MAE – середня абсолютна помилка

GBDT - Gradient Decision Tree Boosting

GOSS – Gradient-based One-Side Sample

EFB – Exclusive Feature Bundling

ВСТУП

Актуальність теми. Машинне навчання (надалі ML) докорінно змінило підхід до розв'язання багатьох проблем у різноманітних галузях, його здатність навчатися на історичних даних і роботи прогнози відкрила незнані раніше можливості для створення більш ефективних стратегій прийняття рішень, оптимізації виробництв та багато іншого.

Однією з популярних та потужних технік ML, що лежать в основі багатьох таких алгоритмів, є градієнтний бустінг. Його реалізації широко використовується у таких сферах, як:

- класифікація зображень
- обробка природньої мови
- прогнозуванні ціни акцій.

Однією з найбільш відомих реалізацій цього алгоритму є Light Gradient Boosting Machine (скорочено LightGBM), що була запропонована у 2016 командою розробників Microsoft на чолі з Гуолінем Ке, одним з авторів статі "LightGBM: A Highly Efficient Gradient Boosting Decision Tree"[1]. LightGBM швидко почав здобувати популярність через свою швидкість, точність та масштабованість.

Мета дослідження. Метою є дослідження алгоритму Light Gradient Boosting Machine та проведення симуляції його роботи на реальних даних.

Об'єкт дослідження. Алгоритм легкого градієнтного бустінгу (LightGBM).

Предмет дослідження. Дерева рішень, градієнтний бустінг, основні техніки застосовані у LightGBM, основні переваги алгоритму та способи використання.

Джерела дослідження. Електронні ресурси.

РОЗДІЛ 1. Від дерев рішень до машини градієнтного бустінгу

У цьому розділі розглянемо алгоритм Gradient Boosting Machine (надалі GBM) та його складові. Градієнтний бустінг – це ансамблевий метод ML, тобто він комбінує у собі декілька прогнозувальних моделей, що називають слабкими учнями, аби забезпечити більш точний результат. GBM зазвичай використовує дерева рішень (особливо CART – дерева класифікації та регресії) фіксованого розміру у ролі слабких учнів. Тому спочатку розглянемо дерева рішень.

1.1 Дерева рішень

Дерева рішень – це метод навчання з учителем, що застосовується для задач класифікації та регресії. Якщо шукана змінна приймає деяке дискретне значення з можливих (наприклад, вид рослини), то дерево рішень називають деревом класифікації, якщо шукана змінна неперервна (наприклад, ціна будинку) – деревом регресії.

Навчання з учителем, також відоме, як контрольоване навчання – це тип машинного навчання, за якого моделі навчаються на тренувальних наборах даних, що крім вхідних даних містять також і правильні вихідні.

1.1.1 Структура дерев рішень

Дерева рішень будуються за допомогою двох типів елементів: вузлів та гілок. На рисунку 1 показано загальну структуру дерева рішень. Тут можна побачити, що основними компонентами дерева рішень є кореневий вузол, у якому дані вперше розбиваються на два дочірніх вузла за фактором, що найкраще з усіх їх розділяє. Також є проміжні вузли, у яких дані розбиваються з врахуванням попередніх розбиттів. Останнім елементом є

кінцеві вузли, які також називають листками або ж листовими вузлами. Саме у них робляться прогнози.

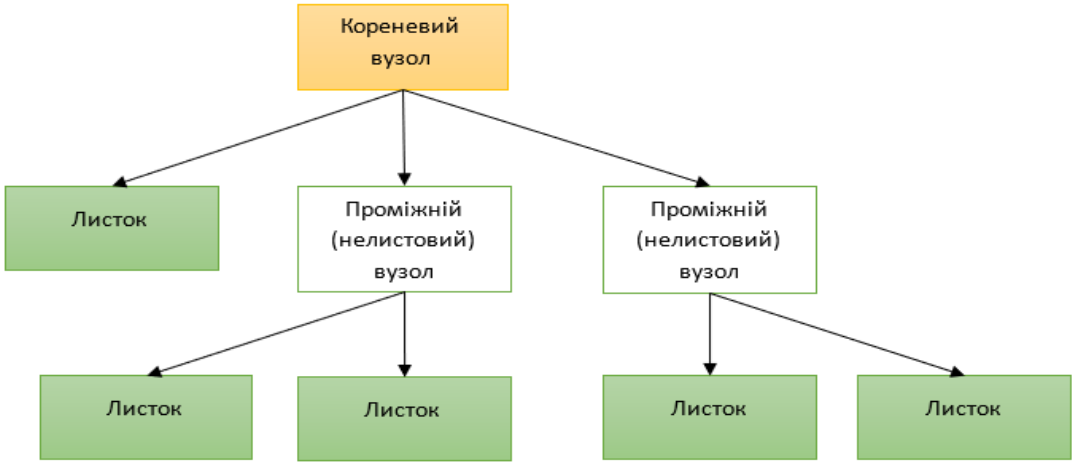

Рисунок 1 Структура дерева рішень

Кожен нелистовий вузол розбивається за деякою фактором x_i, що при розбитті на два дочірні вузли надає найкращу чистоту всередині цих вузлів. (рис 2)

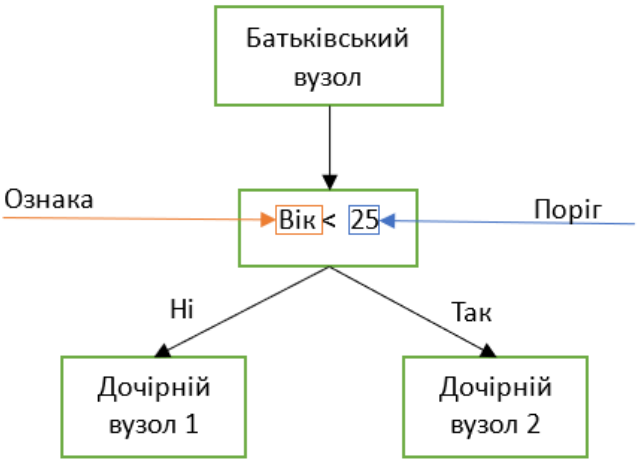

Рисунок 2 Розбиття батьківського вузла на дочірні

1.1.2 Математичне представлення дерев рішень

Розглянемо вектори тренувальних даних $x_i \in R^n, i = 1, \ldots, l$ та вектор цільових значень $y \in R^l$, дерево рішень рекурсивно розбиває простір факторів таким чином, щоб зразки з однаковими або схожими цільовими значеннями були згруповані разом.

Позначимо дані, що знаходяться у деякому вузлі m як Q_m з n_m кількістю елементів. Для кожного розбиття-кандидата $\theta = (j, t_m)$, що складається з фактора j та порогу t_m (дивитись рисунок 2) дані розбиваються на підмножини $Q_m^{left}(\theta)$ та $Q_m^{right}(\theta)$:

$$Q_m^{left}(\theta) = \{(x, y) | x_j \leq t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

Для задач класифікації використовуються функції неоднорідності, для задач регресії – функції втрат.

Критерії розбиття для задач класифікації

Нехай цільова змінна приймає ціле значення від 0 до K-1, тоді для деякого вузла Q_m дерева класифікації пропорція спостережень, що належать до класу k:

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} \boldsymbol{I}(y = k),$$

де **I** – індикаторна функція, що приймає значення 1, якщо $y = k$, і 0, якщо $y \neq k$. Якщо Q_m є кінцевим вузлом, то p_{mk} – передбачена ймовірність класу для цієї області, тобто частка елементів, що належать одному класі у листі.

Для визначення найкращого розбиття використовуються міри неоднорідності, що обраховують однорідність елементів у вузлах дерева. Зазвичай для задач класифікації використовують наступні міри неоднорідності або ж, як їх ще називають міри хаотичності або неоднорідності:

1. Міра неоднорідності Джині (Gini impurity):

$$Gini(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

   Це число від 0 до 0.5, що вказує, якою є ймовірність неправильної класифікації спостережень відповідно до розподілу класів у наборі даних. Розбиття з найменшим значенням міри неоднорідності Джині обирається у якості найкращого.

2. Ентропія:

$$Entropy(Q_m) = -\sum_k p_{mk}\, log(p_{mk})$$

   Ця міра неоднорідності приймає значення від 0 до 1 та вказує на рівень хаотичності даних, де 1 це максимальний ступінь хаотичності, а 0 – мінімальний. Фактично, якщо $Entropy(Q_m) = 0$, то це означає, що у вузлі Q_m знаходяться спостереження одного класу.

Критерії розбиття для задач регресії

У разі, якщо цільова змінна є неперервною, для визначення майбутніх точок розбиття вузла m деякого дерева рішень, зазвичай використовують

функції втрат. Зазвичай це середньоквадратична помилка (надалі MSE) або середня абсолютна помилка (надалі MAE). MSE визначає прогнозоване значення для кінцевих вузлів на основі вивченого середнього значення $\overline{y}_m$ вузла, тоді як MAE визначає його на основі медіани $median(y)_m$.

MSE:

$$H(Q_m) = \frac{1}{n_m} \sum_{y \in Q_m} (y - \overline{y}_m)^2, \text{ де } \overline{y}_m = \frac{1}{n_m} \sum_{y \in Q_m} y,$$

MAE:

$$H(Q_m) = \frac{1}{n_m} \sum_{y \in Q_m} |y - median(y)_m|, \text{ де } median(y)_m = \underset{y \in Q_m}{median}(y).$$

Важливим етапом при розв’язанні задач є оцінка якості розбиття у кожному вузлі. У деякому вузлі m якість розбиття обчислюється за допомогою формули $G(Q_m, \theta)$

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta)),$$

де n_m^{left} - кількість елементів у вузлі Q_m^{left}, а n_m^{right} – кількість елементів у вузлі Q_m^{right}. Для мінімізації неоднорідності або втрат обираються такі параметри

$$\theta^* = \text{argmin}_\theta G(Q_m, \theta)$$

Далі, доки не буде досягнута максимальна дозволена глибина дерева, $n_m < \min_{samples}$ або $n_m = 1$, алгоритм рекурсивно повторюється для підмножин $Q_m^{left}(\theta^*)$ та $Q_m^{right}(\theta^*)$[3].

1.2 Ансамблева модель дерев рішень

На рисунку 3 наведений приклад CART, що класифікує членів родини на тих, кому подобається певна гра X, а кому ні.

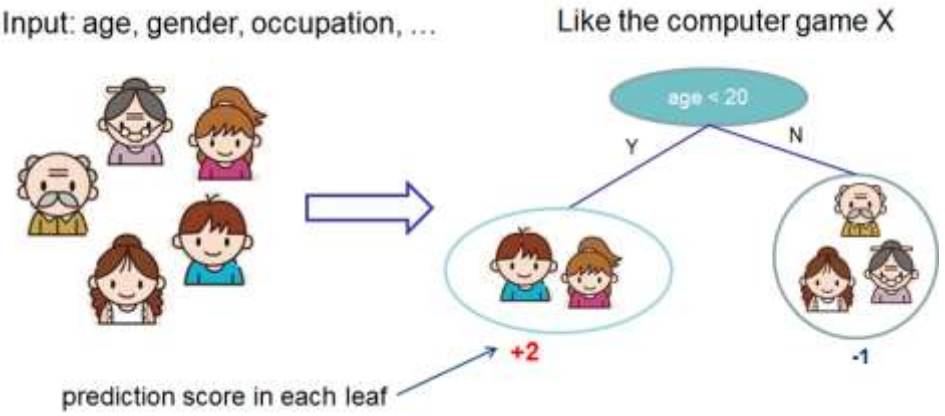

Рисунок 3 Приклад класифікації за допомогою дерева рішень, де Y – Yes, N- No [4]

Дерево розбиває членів сім'ї по листам, яким присвоює певну оцінку. CART відрізняється від звичайних дерев рішень саме тим, що крім розподілення даних на певні категорії по листкам, він додає реальні оцінки для листків (на рисунку 3, наприклад, у лівому листку ця оцінка дорівнює 2, а у правому -1). Це надає більш глибоку інтерпретацію, аніж звичайна класифікація.

Зазвичай одне дерево не надає достатньо точних результатів, щоб його можна було використовувати на практиці. Натомість використовуються ансамблеві моделі, що об'єднують прогнози декількох дерев разом. На рисунку 4 наведено приклад використання ансамблевої моделі, що включає в себе 2 дерева рішень, для вирішення згаданої вище задачі класифікації.

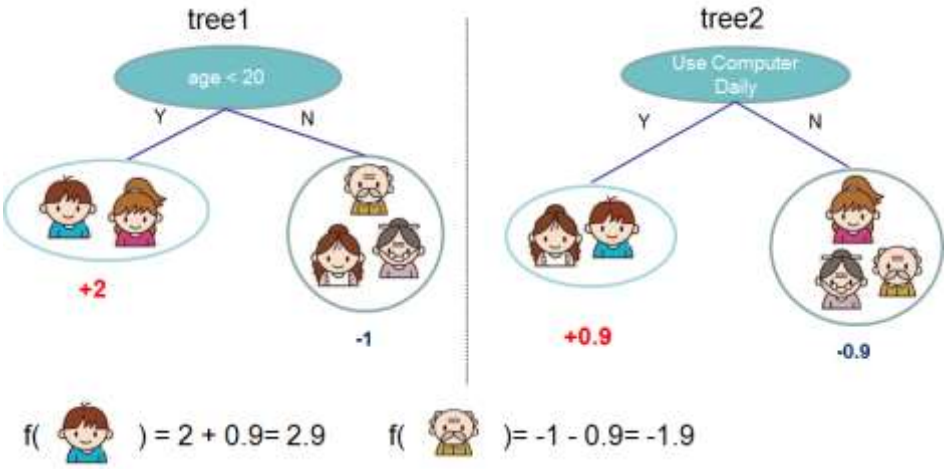

Рисунок 4 Приклад застосування ансамблю з двох дерев рішень [4]

Іншими словами, для набору даних з n елементів та l факторів $D = \{(x_i, y_i)\}$ ($|D| = n, x_i \in \mathbb{R}^l,\ y_i \in \mathbb{R}$), ансамблева модель, що використовує дерева рішень, як слабких учнів, використовує K адитивних функцій для передбачення результату.

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

де K – кількість дерев, $\mathcal{F} = \{f(x) = \omega_q\}(q: \mathbb{R}^l \to T, \omega \in \mathbb{R}^T)$ - простір дерев регресії (або CART), q позначає структуру кожного дерева, T – кількість листків у дереві, а кожна f_k відповідає незалежній структурі дерева q з ваги листків ω.

На відміну від дерев рішень, кожне дерево регресії містить неперервне значення на кожному листку, що позначає оцінку на i-ому листку, для її позначення використовується ω_i. У даному прикладі оцінки у відповідних

листках (позначених ω) дерев (позначених q) підсумовувалися для отримання остаточного прогнозу.

1.3 Бустінг дерев

Бустінг – це метод ансамблевого навчання, тобто така що послідовно об’єднує набір слабких учнів у точнішу модель, щоб мінімізувати помилки навчання.

Слабкі учні – це моделі, що мають низьку точність прогнозування, схожу на випадкове вгадування. Такі моделі схильні до надмірного припасування, тобто не можуть прогнозувати дані, що занадто відрізняються від початкових.

Узагальнена ідея, що лежить в основі більшості методів бустінгу – це послідовне навчання слабких учнів таким чином, щоб кожен наступний намагався виправити помилки минулого. Кожна наступна модель навчається на заново зважених тренувальних даних. Зазвичай неправильно класифіковані вхідні дані отримують більшу вагу, а ті, що були класифіковані правильно, її втрачають. Таким чином, майбутні слабкі учні зосереджують увагу на тих даних, що попередні не змогли класифікувати правильно. На рисунку 5 представлена інтуїтивна ілюстрація алгоритму бустінгу.

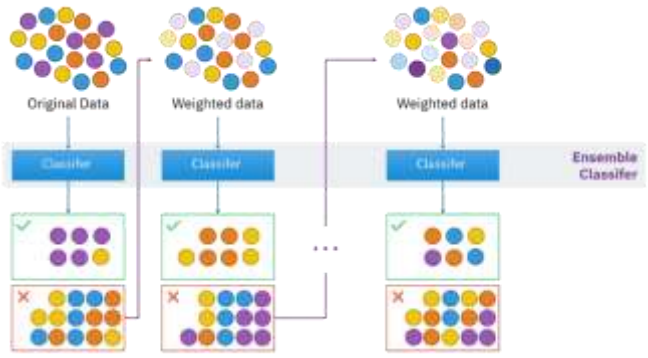

Рисунок 5 Інтуїтивна ілюстрація алгоритмів бустінгу
By Sirakorn - Own work,

Іншими словами, бустінгові моделі будуються у вигляді адитивної моделі: вивчене раніше фіксується, а нові дерева додаються послідовно. Значення прогнозу i-го елементу на кроці t записується у вигляді $\hat{y}_i^{(t)}$. Тоді маємо

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

1.4 Градієнтний бустінг, Gradient Boosting Machine

Ідея градієнтного бустінгу виникла завдяки спостереженню Лео Бреймана про те, що бустінг можна інтерпретувати, як алгоритм оптимізації відповідної функції вартості (з англ. cost function). У даного методу декілька назв, але найпоширенішою є Gradient Boosting Machine, термін у 1999 році запровадив Джером Фрідман, розробник одного з найперших алгоритмів явного регресійного бустінгу, у своїй статі “Greedy Function Approximation: A Gradient Boosting Machine”[2].

Градієнтний бустінг (Gradient boosting) - це техніка ML для задач регресії та класифікації, що створює модель прогнозування у вигляді ансамблю слабких учнів, як правило, дерев рішень, і покращує їх шляхом послідовного навчання нових на відповідних залишках від минулих.

Функція втрат – це математичний інструмент, що визначає, наскільки добре працює модель ML. Вона вимірює рівень помилок моделі, порівнюючи передбачені значення з правильним відповідями з тренувального набору

даних. Чим менше значення функції втрат, тим краще модель працює на тренувальному наборі даних.$L(y, \hat{y})$, де $\hat{y} = f(x)$ – це прогнози моделі.

Оскільки змінна y може бути, наприклад, бінарною або неперервною, може мати різний розподіл, а сама задача може бути задачею класифікації, регресії або ранжування, у різних випадках використовуються різні функції втрат. У випадку неперервності y найчастіше використовуються наступні функції втрат:

1. Gaussian loss або L_2 loss:

$$L(y, f) = (y - f)^2,$$

   що визначає умовне середнє.

2. Laplacian loss або L_1 loss. Ця функція визначає умовну медіану і є більш стійкою до викидів.:

$$L(y, f) = |y - f|.$$

3. L_q loss або Quantile loss:

$$L(y, f) = \begin{cases} (1 - \alpha) \cdot |y - f|, & \text{if } y - f \leq 0 \\ \alpha \cdot |y - f|, & \text{if } y - f > 0 \end{cases}, \alpha \in (0,1)$$

   Ця функція втрат замість середнього та медіани використовує квантилі. Таким чином $\alpha = 0.25$ відповідає 25%-квантилю. Ця функція асиметрична і гірше оцінює спостереження, що лежать праворуч від визначеного квантилю.

Градієнт – це вектор, що показує напрямок та швидкість найшвидшого зростання функції в точці. Для функції $f(x, y)$ градієнт може бути обчислений як вектор:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right),$$

де $\frac{\partial f}{\partial x}$ та $\frac{\partial f}{\partial y}$ – часткові похідні функції $f(x, y)$ по змінним x та y відповідно.

1.4.1 Алгоритм GBM

У загальному випадку задачу машинного навчання з учителем, тобто таку, за якої моделі навчаються на тренувальних наборах даних, що крім вхідних даних містять також і правильні вихідні, можна сформулювати наступним чином. Нехай маємо тренувальний набір даних, що складається з множини факторів x_i де $i = 1, \dots, l$ та множини цільових значень y, метою є мінімізація функції втрат $L(y, \hat{y})$, де $\hat{y} = f(x)$ – це прогнози моделі. Для GBM необхідно, щоб функція витрат була диференційованою.

Для оцінки функції $f(x)$ використаємо ітеративне наближення та отримаємо $\hat{f}(x)$. $\hat{f}(x)$ повинна найкраще мінімізувати задану функцію втрат.

$$y \approx \hat{f}(x), \qquad \hat{f}(x) = \underset{f(x)}{argmin}\, L\big(y, f(x)\big)$$

Розглянемо параметричну сім'ю функцій $f(x, \theta), \theta \in \mathbb{R}^d$. Таким чином маємо задачу оцінювання параметрів:

$$\hat{f}(x) = f(x, \hat{\theta}),$$
$$\hat{\theta} = \underset{\theta}{argmin}[L(y, f(x, \theta))]$$

Зазвичай простих аналітичних рішень для отримання оптимальних значень параметрів $\hat{\theta}$ майже не існує. Через це їх наближають ітеративно.
Наближення $\hat{\theta}$ за M кроків можна виписати у вигляді суми наступним чином:

$$\hat{\theta} = \sum_{i=1}^{M} \hat{\theta}_i$$

Нехай маємо N спостережень (x_i, y) $i = 1, \ldots, n$, за якими зменшуємо емпіричну функцію втрат

$$L_\theta(\hat{\theta}) = \sum_{i=1}^{N} L(y_i, f(x_i, \hat{\theta})).$$

Для мінімізації використовуємо алгоритм градієнтного спуску. Знаходимо градієнт $\nabla L_\theta(\hat{\theta})$ та додаємо ітеративні обчислення $\hat{\theta}_i$ (оскільки ми мінімізуємо втрати, додаємо знак мінус). Обираємо початкове наближення $\hat{\theta}_0$ і кількість ітерацій M.
Процедура оптимізації за градієнтним спуском має наступний вигляд:

1. Визначаємо початкове наближення параметрів $\hat{\theta} = \hat{\theta}_0$
2. Для кожної ітерації $t = 1, \ldots, M$:
    2.1. Обчислюємо градієнт функції втрат $\nabla L_\theta(\hat{\theta})$ за поточного наближення $\hat{\theta}$

$$\nabla L_\theta(\hat{\theta}) = \left[\frac{\partial L(y, f(x, \theta))}{\partial \theta}\right]_{\theta=\hat{\theta}}$$

    2.2. Встановлюємо поточне ітеративне наближення $\hat{\theta}_t$ на основі обчисленого градієнту

$$\hat{\theta}_t \leftarrow -\nabla L_\theta(\hat{\theta})$$

2.3. Оновлюємо наближення параметрів $\hat{\theta}$:

$$\hat{\theta} \leftarrow \hat{\theta} + \hat{\theta}_t = \sum_{i=0}^{t} \hat{\theta}_i$$

3. Зберігаємо результат наближення $\hat{\theta}$:

$$\hat{\theta} = \sum_{i=0}^{M} \hat{\theta}_i$$

4. Використовуємо знайдену функцію $\hat{f}(x) = f(x, \hat{\theta})$[5].

1.4.2 Класичний алгоритм GBM за Фрідманом

Оптимізація у методах бустінгу проводиться у функціональному просторі. Оцінка $\hat{f}$ за M ітерацій параметризована в адитивній функціональній формі:

$$\hat{f}(x) = \sum_{i=0}^{M} \hat{f}_i(x),$$

де $\hat{f}_0$ - це початкове припущення для оцінки $\hat{f}$, а $\hat{f}_i, i = 1, \ldots, M$ – прирости функцій, що також називають boosts. Обмежимося однією сім'єю функцій та введемо параметризовану функцію слабкого учня $\mathrm{h}(\mathrm{x}, \theta)$.

"Жадібний" поетапний підхід до розширення функції з слабкими учнями можна сформулювати наступним чином. На кожній ітерації

підбираємо оптимальний коефіцієнт $\rho \in \mathbb{R}$. На кроці t задача має наступний вигляд:

$$\hat{f}(x) = \sum_{i=0}^{t-1} \hat{f}_i\,(x),$$

$$(\rho_t, \theta_t) = \underset{\rho,\theta}{arg\,min}\, L\left(y, \hat{f}(x) + \rho \cdot h(x,\theta)\right),$$

$$\hat{f}_t(x) = \rho_t \cdot h(x, \theta_t).$$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{f}_t(x) = \sum_{i=0}^{t} \hat{f}_i\,(x)$$

Тут задачу описано у загальному вигляді. На практиці отримати розв'язок для оцінок параметрів за заданими функцією втрат $L\big(y, f(x,\theta)\big)$ та слабким учнем $h(x,\theta)$ буває дуже складно. Тому було запропоновано спосіб, що значно полегшує задачу, а саме використання градієнту функції втрат.

Оскільки градієнт вказує на напрямок найшвидшого зростання функції, то для того, щоб мінімізувати функцію втрат, ми маємо рухатися у напрямку, протилежному до напрямку градієнту. Для цього ми навчаємо моделі таким чином, щоб їхні прогнози були більш корельованими з від'ємним значенням градієнту.

Розглянемо алгоритм GBM, що було запропоновано Jerome Friedman у 1999[2].

На вхід алгоритму подається наступне:

- тренувальний набір даних $(\mathrm{x_i}, \mathrm{y})\ \mathrm{i} = 1, \ldots, \mathrm{n}$
- кількість ітерацій M
- обрана диференційована функція втрат $L(y, f(x))$
- обрана сім'я функцій алгоритмів слабких учнів $h(x,\theta)$

- додаткові гіперпараметри для $h(x, \theta)$ (для дерева рішень, наприклад, це може бути його глибина)

У якості початкового наближення використовується константне значення γ.

Алгоритм можна сформулювати наступним чином:

1. Задаємо початкові константні значення моделі:

$$\hat{f}_0(x) = \underset{\gamma}{arg\,min} \sum_{i=1}^{n} L(y_i, \gamma), \qquad \gamma \in \mathbb{R}.$$

2. Для кожної ітерації $t = 1, \ldots, M$, повторюємо:

2.1 Розраховуємо так звані псевдо-залишки r_t:

$$r_{it} = -\left[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\right]_{f(x)=\hat{f}(x)}, \text{ for } i = 1, \ldots, n.$$

2.2 Навчаємо слабкого учня $h_t(x, \theta)$ на псевдо-залишках, тобто на тренувальній множині $(x_i, r_{it}),\ i = 1, \ldots, n$.

2.3 Шукаємо оптимальний коефіцієнт ρ_t за поточного $h_t(x, \theta)$ відповідно до початкової функції втрат:

$$\rho_t = \underset{\rho}{arg\,min} \sum_{i=1}^{n} L\left(y_i, \hat{f}(x_i) + \rho \cdot h_t(x_i)\right) \qquad (1)$$

2.4 Оновлюємо модель:

$$\hat{f}_t(x) = \rho_t h_t(x_i)$$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{f}_t(x) = \sum_{i=0}^{t} \hat{f}_i\,(x) \qquad (2)$$

3. Отримуємо кінцеву модель $\hat{f}(x)$.

1.4.3 Градієнтний бустінг дерев рішень (надалі GBDT)

Градієнтний бустінг зазвичай використовує CART фіксованого розміру у ролі слабких учнів. Для такого випадку було запропоновано модифікацію, що покращує якість навчання кожного слабкого учня.

Класичний алгоритм градієнтного бустінгу навчає дерева рішень на псевдо-залишках. Якщо J_t – кількість кінцевих вузлів у дереві $h_t(x)$, то це дерево розбиває вхідні дані на J_t множин, що не перетинаються: $R_{1t}, \ldots, R_{J_t t}$ і прогнозує значення $b_{1t}, \ldots, b_{J_t t}$ для цих множин. Результат роботи дерева $h_t(x)$ можна записати наступним чином:

$$h_t(x) = \sum_{j=1}^{J_t} b_{jt} I_{R_{jt}}(x),$$

де $I_{R_{jt}}(x)$ – індикаторна функція, що приймає значення 1, якщо $x \in R_{jt}$, та 0, якщо $x \notin R_{jt}$.

Потім значення b_{jt} множаться на коефіцієнт ρ_t, що обирається за допомогою лінійного пошуку для мінімізації функції втрат (1), а сама модель оновлюється, як у (2).

Запропонована Jerome Friedman модифікація полягає у пошуку окремих оптимальних коефіцієнтів ρ_{jt} для кожної множини R_{jt}. У такому разі значення b_{jt} можна відкинути і отримаємо наступне правило оновлення моделі:

$$\hat{f}_t(x) \leftarrow \hat{f}_{t-1}(x) + \sum_{j=1}^{J_t} \rho_{jt} 1_{R_{jt}}(x),$$

$$\rho_{jt} = \underset{\rho}{arg\,min} \sum_{x_i \in R_{jt}} L\big(y_i, \hat{f}_{t-1}(x_i) + \rho\big).$$

Тут кількість кінцевих вузлів – це параметр моделі, що налаштовується вручну, за його допомогою контролюється максимально допустимий рівень взаємодії між змінними. Якщо $J_t = 2$ (таке дерево також називають пеньком рішень), то змінні не взаємодіють між собою. Зазвичай параметр $4 \leq J_t \leq 8$.

1.4.4 Регуляризація

У цьому підрозділі розглянемо так звані техніки регуляризації, що використовуються у GBM. Вони зменшують ефект надмірного припасування моделі до тренувальних даних.

Одним з параметрів регуляризації є кількість ітерацій M: збільшення M зменшує кількість помилок на тренувальних даних, але може призвести до надмірного припасування.

Ще одним з таких параметрів є глибина дерева: чим вище це значення, тим ймовірніше надмірне припасування до тренувальних даних. Також таким параметром є кількість спостережень у листку. Її обмежують і тоді під час побудови кандидати-розбиття дерева, за яких у вузлі буде менше спостережень, аніж вказане значення, ігноруються. Накладання цього обмеження допомагає зменшити дисперсію передбачень на листках.

Стиснення

У градієнтному бустінгу важливих методом регуляризації є стиснення (з англ. shrinkage). Цей метод змінює правило оновлення моделі на кроці t наступним чином:

$$\hat{f}_t(x) \leftarrow \hat{f}_{t-1}(x) \; + \; \nu \cdot \rho_t h_t(x), \qquad 0 < \nu \leq 1,$$

де ν – параметр, що називають швидкістю навчання (з англ. learning rate). Використання невеликої швидкості навчання (наприклад, $\nu < 0.1$) покращує узагальнюючу здатність моделі. Однак це досягається ціною збільшення обчислювального часу. Менша швидкість навчання вимагає більшої кількості ітерацій.

GBM на підвибірках

На кожній ітерації з повної вибірки тренувальних даних випадковим чином вибирається підвибірка, що потім використовується замість повної вибірки для навчання слабкого учня[6].

Розмір підвибірки – це відсоток s від розміру тренувальної вибірки. Коли $s = 1$, алгоритм є детермінованим, менші значення s додають випадковості в алгоритм і запобігають надмірному припасуванню, а також пришвидшують роботу алгоритму, адже дерева рішень навчаються на менших множинах даних на кожній ітерації.

РОЗДІЛ 2. Light Gradient Boosting

З початком появи великих даних, у яких кількість спостережень та факторів лише зростає, користувачі алгоритму GBDT все частіше стають перед вибором між точністю та ефективністю. Хоча GBDT й зараз активно використовується, зростання кількості даних, що обробляються моделлю показало необхідність у покращенні даного алгоритму. Саме це й запропонували розробники Microsoft на чолі з Guolin Ke у статті “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”[1].

LightGBM базується на GBM, але має певні відмінності від класичних реалізацій, про які поговоримо у цьому розділі.

2.1 Ріст дерева по листу

У більшості реалізаціях GBM, наприклад, в XGBoost дерева зростають за рівнем (level-wise), як показано на рисунку 6.

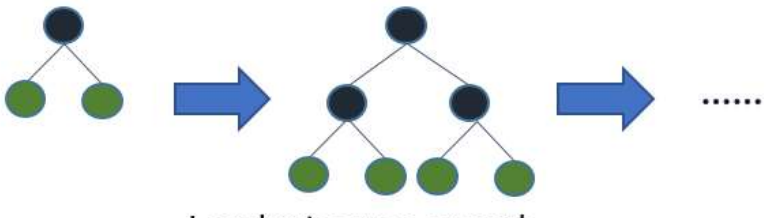

Рисунок 6 Ріст дерева за рівнем[9]

За leaf-wise зростання дерев найкраще розбиття розбиває вузол на дочірні на один рівень вниз, а самі дерева виходять симетричні.

Натомість у LightGBM дерева зростають за листками (leaf-wise), такий метод ще називають “перший найкращий”. На кожній ітерації роботи алгоритм обирає листок, що найкраще з можливих варіантів зменшує втрати, і розбиває лише цей листок, ігноруючи інші листки на тому рівні, як показано на рисунку 7.

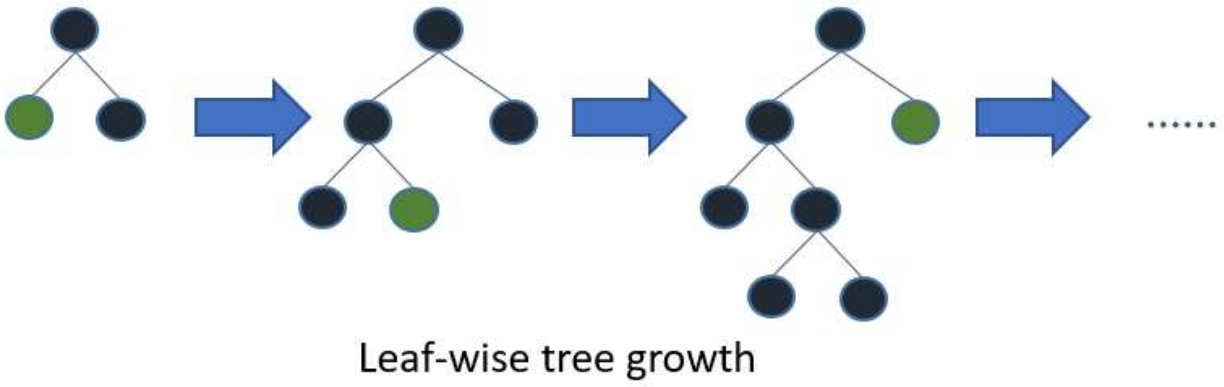

Рисунок 7 Зростання дерева за листками[9]

Така стратегія розбиття листків, як правило, призводить до менших втрат порівняно з level-wise. Також завдяки використанню такого підходу для побудови дерев рішень, алгоритм створює складніші дерева, що призводить до вищої точності. Втім, leaf-wise зростання дерев може призвести до надмірного припасування, особливо для невеликих наборів даних. Тому щоб зменшити вплив надмірного припасування LightGBM має параметр max_depth, тобто максимальну глибину дерева рішень, що обмежує кількість рівнів дерева [9].

2.2 Розбиття дерев рішень на основі гістограми

Найбільш трудомістка операція у роботі GBDT – навчання дерев рішень, за якого найбільше часу витрачається на визначення найкращих точок розбиття. Одним з найбільш популярних методів для навчання дерев рішень є алгоритм попереднього сортування (pre-sorted algorithm), що перераховує всі можливі точки розбиття за попередньо відсортованими значеннями факторів. Такий алгоритм, хоч і може знайти оптимальні точки розбиття, є неефективним за швидкістю та споживанням пам’яті.

LightGBM, натомість, використовує алгоритм на основі гістограми (histogram-based algorithm). Даний алгоритм групує неперервні значення факторів у дискретні групи, так звані біни, і використовує їх для побудови

гістограм факторів під час навчання, перетворюючи таким чином розподіл факторів у рівномірний для пошуку розбиття. Це значно пришвидшує роботу алгоритму та кількість потрібної пам’яті.

Розглянемо тренувальну множину з лише двома факторами x_1 та x_2, які є колонками:

$$\begin{bmatrix} 1.5 & 0 \\ 0 & 5.5 \\ 0.3 & 7 \\ 5.5 & 8.5 \end{bmatrix}$$

Для такої тренувальної множини, за алгоритмом на основі гістограми створиться гістограма рівної щільності для кожного фактору, замінюючи кожне окреме значення, індексом біну, до якого його було віднесено. Наприклад, якщо гістограма матиме 3 біни, то можемо отримати такий результат:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 2 \\ 2 & 2 \end{bmatrix}$$

Таким чином, за алгоритмом попереднього сортування модель перераховує всі можливі точки розбиття за попередньо відсортованими значеннями факторів, тобто швидкість пошуку найкращих точок розбиття пропорційна до $\#data \times \#feature$, де $\#data$ – кількість спостережень, а $\#feature$ – кількість факторів. Для алгоритму на основі гістограми швидкість пошуку найкращих точок розбиття буде пропорційною до $\#bin \times \#feature$, де $\#bin$ – кількість груп факторів. Оскільки $\#bin$ зазвичай має набагато менше значення, аніж $\#data$, найбільш трудомісткою частиною

роботи алгоритму буде саме побудова гістограми, що залежить від $\#data \times \#feature$ але не пошук точок розбиття [1].

Крім того, оскільки неперервні значення даний алгоритм заміняє на індекси бінів, до яких їх було віднесено, для зберігання гістограми потрібно менше пам'яті.

2.3 Gradient-based One-Side Sample

Gradient-based One-Side Sample (GOSS), тобто одностороння вибірка на основі градієнту – це новітня техніка ML, що була запропонована та реалізована розробниками LightGBM у даному алгоритмі. Основна ідея GOSS полягає у використанні значення градієнту для елементів даних, як індикатора їхньої важливості для навчання дерев.

Так як у багатьох алгоритмах ML вага елементів даних є індикатором їхньої важливості, а у GBDT початкової ваги для даних немає, це унеможливлювало застосування деяких методів для створення підвибірки для навчання моделі. Але, оскільки градієнт вказує на напрямок та темп найшвидшого зростання, розробники відзначили, що він несе й інформацію про важливість елементів даних[1]. Таким чином, елементи з невеликим значенням градієнту мають невелику тренувальну помилку і вважаються добре натренованими.

Для навчання моделі GOSS спочатку сортує елементи з початкової вибірки за значенням градієнту та відбирає $a \times 100\%$ елементів з найбільшим значенням градієнту, де $a \in (0,1)$, а потім випадковим чином обирає $b \times 100\%$, де $b \in (0,1)$, елементів з тих, що лишилися. Після цього, щоб початковий розподіл не був сильно змінений, GOSS підсилює вплив елементів з малим градієнтом за допомогою константи $\frac{1-a}{b}$. Таким чином, під час навчання модель зосереджується на недостатньо вивчених елементах, та витрачає менше ресурсів на менш важливі для побудови моделі елементи. Це

у свою чергу зменшує кількість елементів даних, що потрібні для тренування моделі, а відповідно збільшує її швидкість.

2.4 Exclusive Feature Bundling

У реальних застосунках часто трапляється так, що дані мають велику кількість факторів, що зазвичай є розрідженими, тобто мають багато невідомих або нульових значень. У такому випадку часто трапляються фактори, що є між собою або цілком, або майже взаємовиключними, тобто такими, що рідко приймають значення рівне 0 одночасно. Exclusive Feature Bundling (EFB) або пакування взаємовиключних факторів – це техніка розроблена саме для таких випадків. Ідея EFB полягає в об'єднанні взаємовиключних факторів в один (тобто групування), який називають bundle, або пакетом. За допомогою цієї техніки можна побудувати гістограми для пошуку точок розбиття з пакетів факторів, а не з окремих факторів, що зменшує складність побудови самої гістограми з пропорційної до $\#data \times \#feature$ до пропорційної до $\#data \times \#bundle$. Оскільки $\#bundle$ – кількість групувань, є меншим за кількість факторів $\#feature$, ми можемо значно прискорити навчання моделі без шкоди для точності.

2.5 Висновки

Завдяки технікам, що використовуються у LightGBM має наступні переваги відносно класичних реалізацій GBM:

1. Підвищені швидкість навчання та ефективність:
   Роботу моделі пришвидшують EFB та GOSS, що були запропоновані саме у LightGBM, а відповідно не зустрічаються в інших реалізаціях GBM. Якщо зазвичай у реалізаціях GBM використовується алгоритм

попереднього сортування для пошуку найкращих точок розбиття, то у LightGBM використовується алгоритм на основі гістограми, що значно пришвидшує цей пошук.

2. Зменшене використання пам’яті:

   Оскільки за допомогою технік, що застосовуються в LightGBM, кількість факторів та спостережень, що необхідно зберігати, зменшуються до кількості пакетів та бінів відповідно, для роботи алгоритму потрібно менше пам’яті.

3. Краща точність порівняно з іншими GBM:

   Якщо у класичних реалізаціях GBM застосовується level-wise підхід до побудови дерев рішень, то у LightGBM замість нього використовується leaf-wise підхід, що створює складніші дерева, аніж за level-wise підходу. Це призводить до вищої точності моделі.

4. Сумісність з великими за об’ємом спостережень та факторів наборами даних:

   Завдяки технікам, що використовуються у LightGBM, даний алгоритм має високу точність та швидкість при роботі з наборами даних з високо вимірним простором факторів та великою кількістю спостережень працює швидше аніж більшість реалізацій GBM.

У цілому, LightGBM – це потужний та перспективний інструмент ML, що забезпечує високу точність та ефективність. Завдяки своїм перевагам та можливостям, він займає одне з провідних місць серед алгоритмів машинного навчання та знаходить широке застосування в різних галузях.

РОЗДІЛ 3. Симуляція роботи LightGBM та його застосування на реальному прикладі

У даному розділі проведемо симуляцію роботи LightGBM та розглянемо результати його застосування на реальному наборі даних “Allstate Claims Severity”[10], що є у загальному доступі.

3.1 Симуляція роботи

Розглянемо результати роботи LightGBM на згенерованому нами наборі даних. Для цього згенеруємо 500 випадкових значень, розподілених нормально – це і буде єдиним фактором x у нашому наборі даних. Крім того, таким самим чином згенеруємо значення відхилень eps та визначимо значення цільової змінної $y = -0.5 + 1.5x\text{^}2 + eps$. Отримаємо набір даних, перші 8 значень з якого, мають наступний вигляд:

	x	y
0	-0.102957	-0.406468
1	-0.653903	0.759837
2	0.018394	-0.023626
3	-1.477241	1.623064
4	1.246558	3.474033
5	-1.612557	3.865045
6	0.593514	-0.245505
7	0.503616	1.556528
8	0.378210	1.214062

Рисунок 8 Набір даних, згенерований для симуляції роботи LightGBM

Фактор x та відхилення eps розподілені наступним чином:

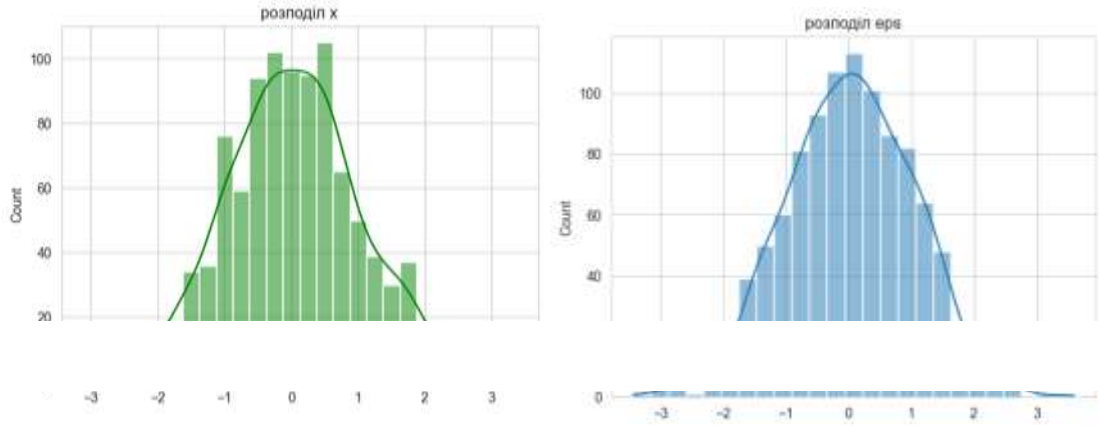

Рисунок 9 Розподіли фактору x та відхилення eps

За допомогою функції train_test_split розділимо даний набір даних на тестову, обсягом у 20% від початкового набору даних, та тренувальну, обсягом у 80% від початкового, множини. Тренуємо модель LightGBM на тренувальних даних та прогнозуємо значення цільової змінної y за тестовою множиною фактору x. Створимо набір даних, що містить значення x та y з тестової множини та прогнозовані нашою моделлю значення y. Отримаємо набір даних, який матиме наступний вигляд:

x	y_true	y_pred
-0.102957	-0.406468	-1.036616
0.018394	-0.023626	-0.244863
-0.281025	0.199165	-0.448511
-0.566719	-0.715689	-0.179172
0.479224	-0.997237	0.342910
-0.299514	-1.713546	-0.507860
-1.390391	2.972385	2.098895
0.840489	0.292515	0.434625

Рисунок 10 Набір даних, що містить значення фактору, дійсні значення цільової змінної та прогнозовані LightGBM

Розглянемо графік, що містить дійсні значення y з тестової множини та прогнозовані за допомогою LightGBM, також на ньому зображена гіпербола $y = -0.5 + 1.5x^2$, що відповідає за відношення x та y.

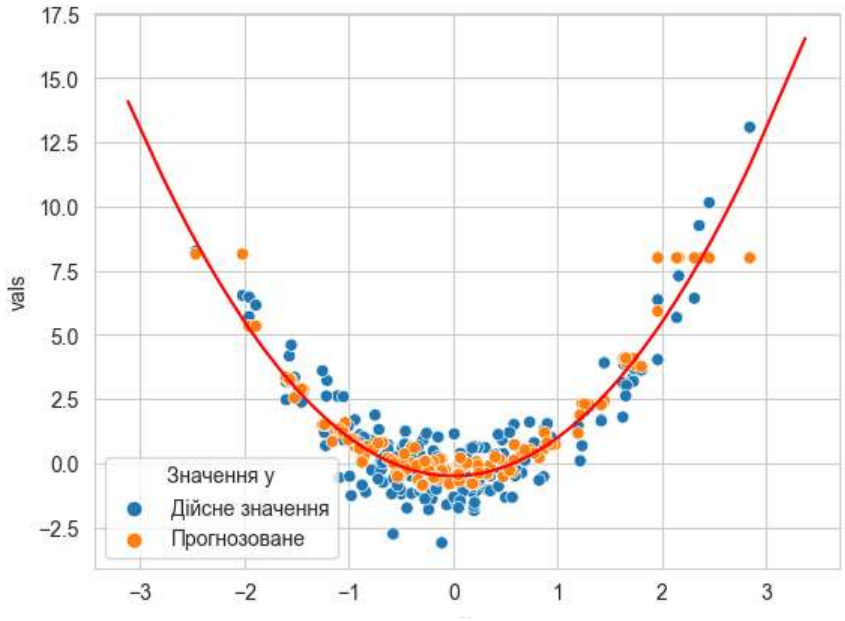

Рисунок 11 Графік відношення дійсного значення цільової змінної з прогнозованим

Як бачимо, прогнозовані моделлю значення відповідають такому ж відношенню, як і дійсні.

3.2 Опис набору даних для прикладу застосування LightGBM

Набір даних “Allstate Claims Severity” (з англ. “важкість заяв у компанії Allstate”) був оприлюднений компанією Allstate Insurance, що є одним з найбільших постачальників особистого страхування у США. Завданням, що було поставлене самою компанією у конкурсі на всесвітньо відомій платформі Kaggle[11] до цього набору даних є створення моделі, що зможе якомога точніше передбачити вартість збитків від страхових випадків.

“Allstate Claims Severity” налічує 188318 анонімізованих спостережень та 130 факторів, з яких 16 є неперервними, а 114 категоріальними. Фактори включають інформацію про поліс, транспортний засіб, деталі страхового покриття і т. ін., але оскільки вони є зашифрованими, сказати, який фактор за що відповідає, не можемо.

3.3 Попередня підготовка даних

Оскільки фактори є зашифрованими, тренуватимемо модель на усіх наявних. Після перевірки на наявність відсутніх значень з’ясували, що таких немає. Крім того було замінено значення категоріальних факторів з буквених на цілочисельні. Основні статистичні показники для цільової змінної продемонстровані на рисунку 8, а для неперервних факторів – на рисунку 9.

	loss
count	188318.000000
mean	3037.337686
std	2904.086186
min	0.670000
25%	1204.460000
50%	2115.570000
75%	3864.045000
max	121012.250000

Рисунок 12 Основні статистичні показники для цільової змінної "loss"

	count	mean	std	min	25%	50%	75%	max
cont1	188318.0	0.493861	0.187640	0.000016	0.346090	0.475784	0.623912	0.984975
cont2	188318.0	0.507188	0.207202	0.001149	0.358319	0.555782	0.681761	0.862654
cont3	188318.0	0.498918	0.202105	0.002634	0.336963	0.527991	0.634224	0.944251
cont4	188318.0	0.491812	0.211292	0.176921	0.327354	0.452887	0.652072	0.954297
cont5	188318.0	0.487428	0.209027	0.281143	0.281143	0.422268	0.643315	0.983674
cont6	188318.0	0.490945	0.205273	0.012683	0.336105	0.440945	0.655021	0.997162
cont7	188318.0	0.484970	0.178450	0.069503	0.350175	0.438285	0.591045	1.000000
cont8	188318.0	0.486437	0.199370	0.236880	0.312800	0.441060	0.623580	0.980200
cont9	188318.0	0.485506	0.181660	0.000080	0.358970	0.441450	0.566820	0.995400
cont10	188318.0	0.498066	0.185877	0.000000	0.364580	0.461190	0.614590	0.994980
cont11	188318.0	0.493511	0.209737	0.035321	0.310961	0.457203	0.678924	0.998742
cont12	188318.0	0.493150	0.209427	0.036232	0.311661	0.462286	0.675759	0.998484
cont13	188318.0	0.493138	0.212777	0.000228	0.315758	0.363547	0.689974	0.988494
cont14	188318.0	0.495717	0.222488	0.179722	0.294610	0.487403	0.724623	0.844848

Рисунок 13 *Основні статистичні показники неперервних факторів Allstate Claims Severity*

3.4 Прогнозування

Симуляція роботи моделі була проведена для LightGBM з використанням EFB та GOSS (надалі буде позначено як lgb), без використання GOSS (надалі efb_only) та без використання обох технік (lgb_baseline).

Для прогнозування набір даних “Allstate Claims Severity” було розділено на тренувальну підвибірку, що вміщує 150654 спостережень, тобто 80% від початкового набору даних, та на тестову, що вміщує 37664 спостереження, тобто 20% від початкового набору.

Підбором параметрів було з’ясовано, що найкращі результати модель отримує, якщо встановити наступні параметри, як у таблиці 1.

Таблиця 1 Спільні параметри для моделей

learning_rate	0.02
n_estimators	1000
num_leaves	50
colsample_bytree	0.4

Тут $learning\_rate$ відповідає швидкості навчання, про яку йшлося у розділі 1.4.4. $n\_estimators$ відповідає кількості дерев рішень, що будуть побудовані для досягнення фінальної моделі. num_leaves відповідає максимальній кількості листків у деревах рішень. $colsample\_bytree$ відповідає за відсоток колонок, що будуть використані при побудові кожного дерева, для цього параметру значення має бути від 0 до 1, де 1 означає, що для навчання дерева будуть використані усі колонки, а значення менше 1 відповідає випадковому вибору певної частини колонок.

Окремо для того, щоб у lgb_baseline не використовувалися техніки EFB та GOSS, маємо встановити параметр $enable\_bundle = False$, оскільки за замовчуванням LightGBM не використовує GOSS, інші параметри налаштовувати не потрібно. Натомість, щоб у lgb використовувалися і GOSS, і EFB, встановлюємо параметри $enable\_bundle = True$ і $boosting\_type = \text{“}goss\text{”}$.

Середній час тренування моделей

Після проведення симуляції роботи даних моделей 50 разів отримали наступні середні значення часу витраченого на тренування моделей:

Таблиця 2 Середні значення часу витраченого моделями на тренування

lgb_baseline	efb_only	lgb
20.809418	20.451197	18.921176

Як бачимо використання EFB разом з GOSS дійсно зменшує час необхідний для тренування моделі.

Оцінки якості прогнозування

Для визначення точності моделей було використано **кросвалідацію**, що є процес оцінки ефективності моделі шляхом поділу даних на кілька частин, так званих фолдів, для тренування та валідації. Кожен фолд використовується, як тестова множина, для моделі, що тренується на інших

фолдах. Це дозволяє отримати більш надійні оцінки ефективності моделі на нових даних.

Після проведення кросвалідації, для якої початковий набір даних було розбито на 5 фолдів, було отримано наступні результати:

Таблиця 3 Результати кросвалідації

Модель	lgb_baseline	efb_only	lgb
Середнє значення точності	0.579001	0.579001	0.580163
Мінімальне значення точності	0.574769	0.574769	0.573919
Максимальне значення точності	0.583748	0.583748	0.586471

Як бачимо, середня та мінімальна точність вища у моделі, що використовує і EFB, і GOSS.

Крім точності для оцінки якості прогнозування також використовуються MSE, тобто середньоквадратична помилка, та MAE, тобто середня абсолютна помилка, про які йшлося у розділі 1.1.2. Ці метрики надають інформацію про те, наскільки далеко прогнозовані значення відхиляються від фактичних. Моделі з низькими значеннями вважаються більш точними та ефективними.

Таблиця 4 Отримані значення MSE та MSE

	lgb_baseline	efb_only	lgb
MSE	3444629.54	3444629.54	3422112.13
MAE	1172.27	1172.27	1171.28

Знову бачимо, що результати отримані для моделі LightGBM, що використовує обидві техніки EFB та GOSS кращі.

Також розглянемо, як наші моделі спрогнозують цільову змінну для обраних 5 спостережень з тестової множини (таблиця 5) та з тренувальної множини (таблиця 6):

Таблиця 5 Прогнози моделей для 5 випадкових спостережень з тестової множини

	lgb_baseline	efb_only	lgb	Справжнє значення
№1	5127.031310	5127.031310	5319.305582	9065.99
№2	2178.757903	2178.757903	2059.621357	1255.38
№3	1302.902063	1302.902063	1316.760836	449.12
№4	1713.704338	1713.704338	1697.856329	1212.23
№5	8417.240213	8417.240213	8288.186408	11029.86

Таблиця 6 Прогнози моделей для 5 випадкових спостережень з тренувальної множини

	lgb_baseline	efb_only	lgb	Справжнє значення
№1	2138.404751	2138.404751	2131.968398	1238.14
№2	3682.084841	3682.084841	3616.554454	936.30
№3	2005.212995	2005.212995	1910.217051	2728.53
№4	4214.702819	4214.702819	4438.659189	4153.59
№5	1738.816590	1738.816590	1672.112917	1456.59

Як бачимо, прогнози в цілому відповідають вказаним вище значенням метрик MSE та MAE.

3.5 Висновки

Отже, за результатами проведеної симуляції роботи моделей LightGBM без використання технік GOSS та EFB, з використанням лише EFB та з використанням і EFB, і GOSS, ми можемо сказати, що GOSS та EFB дійсно збільшують точність та швидкість роботи моделі. Як можемо побачити за таблицями, наведеними вище, з використанням GOSS та EFB швидкість роботи збільшилася в середньому на 1.5-2 секунди, а точність зросла на 0.001, а також зменшилися MSE та MAE, що свідчить про більшу ефективність застосування обох технік у моделі.

Висновки

Отже, у цій роботі ми досліджували метод легкого градієнтного бустінгу (LightGBM), що є швидким та ефективним ансамблевим методом ML, який використовує дерева рішень у ролі слабких учнів та має у своїй основі алгоритм градієнтного спуску. Також було наведено результати симуляції роботи LightGBM на реальних даних з використанням мови програмування Python.

LightGBM – це потужний та популярний засіб машинного навчання, що застосовується, як для задач класифікації, так і для задач регресії у багатьох сферах. Основні переваги використання даного алгоритму полягають у наступному:

- Підвищені швидкість та точність
  Техніки, що використовуються у LightGBM значно пришвидшують його роботу та точність
- Менший об’єм необхідної для роботи пам’яті.
  За допомогою технік групування, що застосуються у LightGBM, зменшується й об’єм необхідної пам’яті.
- LightGBM сумісний з наборами даних, що мають велику кількість спостережень та факторів.

Дійсно, розглянувши результати проведеної симуляції роботи даного алгоритму, ми наочно переконалися у тому, що LightGBM швидко та відносно точно працює на наборах даних з великою кількістю спостережень та факторів.

Список використаної літератури

1. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree” Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu
2. Friedman J. H. (February 1999). “Greedy Function Approximation: A Gradient Boosting Machine”
3. 1.10. Decision Trees. Scikit-learn documentation https://scikit-learn.org/stable/modules/tree.html
4. “XGBoost: A Scalable Tree Boosting System” Tianqi Chen, Carlos Guestrin
5. “Gradient boosting machines, a tutorial” Alexey Natekin, Alois Knoll
6. “Stochastic Gradient Boosting” Jerome H. Friedman (1999)
7. “Decision tree learning” Wikipedia https://en.wikipedia.org/wiki/Decision_tree_learning
8. “What is boosting?” amazon https://aws.amazon.com/what-is/boosting/
9. LightGBM documentation https://lightgbm.readthedocs.io/en/v3.3.2/
10. “Allstate Claims Severity” dataset https://www.kaggle.com/competitions/allstate-claims-severity/overview
11. Kaggle https://www.kaggle.com/