Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Прогнозна модель товару без передісторії з використанням LightGBM**»

Виконала: студентка 4-го року навчання освітньої програми «Прикладна математика», спеціальності 113 Прикладна математика

Толокнова Варвара Вячеславівна

Керівник: Дрінь С.С., кандидат фіз.-мат. наук, доцент

Рецензент_____

(прізвище та ініціали)

Кваліфікаційна робота

захищена з оцінкою

_____

Секретар ЕК _____

«____» _____ 2023 р.

Київ – 2023

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

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., д.ф-м.н.
_____Б. В. Олійник
(підпис)
„____”_____202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту 4-го курсу бакалаврської програми, факультету інформатики
Толокновій Варварі Вячеславівній

Розробити модель для прогнозування ціни товару без історії

Вихідні дані:

- прогнозована ціна товару

Зміст ТЧ до бакалаврська роботи:

Зміст

Анотація

Вступ

1. Огляд теорії;
2. Аналіз алгоритму;
3. Практичне застосування;

Висновки

Література

Додатки

Дата видачі „___” _____ 2023 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Прогнозна модель товару без передісторії з використанням LightGBM

Календарний план виконання роботи:

№ п/п	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Визначення теми кваліфікаційної роботи.	29.10.2022	
2.	Визначення літератури для роботи.	07.02.2023	
3.	Огляд і аналіз літератури за темою роботи.	14.02.2023	
4.	Оформлення теоретичної частини кваліфікаційної роботи.	07.03.2023	
5.	Виконання практичного застосування вибраного методу.	31.03.2023	
6.	Оформлення практичної частини кваліфікаційної роботи.	12.04.2023	
7.	Створення презентації кваліфікаційної роботи.	18.05.2023	
8.	Захист кваліфікаційної роботи.	5.06.2023	

Студент _____

Керівник _____

“_____”_____

Зміст

Перелік умовних позначень

EFB - Exclusive Feature Bundling

GBDT - Gradient Boosting Decision Tree

GBM - Gradient-boosting machine

GBRT - Gradient Boosted Regression Trees

GOSS - Gradient-based One-Side Sampling

LightGBM - Light gradient-boosting machine

MAE – Mean absolute error

ML - Machine Learning

RF – Random forest

RMSE – Root mean square error

R^2- коефіцієнт детермінації

XGBoost - Extreme Gradient Boosting

ВСТУП

Сучасні технології та динамічні умови ринкової конкуренції ставлять перед компаніями значні виклики щодо ефективного управління продажами та встановлення оптимальних цін на їхню продукцію. Одним із складних завдань є визначення цін на новий товар, який не має передісторії продажів або історичних даних.

У цьому контексті виникає необхідність розробки прогнозних моделей, які допомагатимуть компаніям встановити оптимальну ціну на товари без попереднього досвіду. Одним із потужних інструментів для побудови таких моделей є алгоритм машинного навчання LightGBM (англ. Light Gradient Boosting Machine).

Актуальність. . Ціноутворення на продукт є однією з найбільших проблем для будь-якої компанії на ринку. Компанія повинна стежити за попитом споживачів і змінами вартості товарів і послуг в своїй сфері, щоб визначити найкращу ціну на свою продукцію. Збираючись вивести на ринок новий продукт, компанії стикаються з питанням встановлення найбільш ефективної ціни на товар без історії. Якщо ви поставите завищену ціну, то купівельна спроможність людей буде нижчою, і навпаки. Особливо важливо встановлення «правильної» ціни у кризовий час, адже це є гарантом успішного майбутнього компанії.

В економічній теорії кажуть: « ціни визначаються попитом і пропозицією. Якщо відносний попит на продукт зростає, споживачі будуть готові платити за нього більше. Їхні конкурентоспроможні пропозиції зобов’язуватимуть їх платити за це більше, а виробникам дозволять отримати за це більше»[1, с.32].

Метою роботи є встановлення ціни на товар без історії за його характеристиками та даними про сусідні схожі товари з використанням LightGBM.

Об'єктом дослідження є рекомендаційна система прогнозування ціни з використанням методу LightGBM.

Предметом дослідження є алгоритми прогнозування ціни.

Мета роботи зумовила наступне **наукове завдання** :

1. Розглянути методи та алгоритми для прогнозування ціни товару без історії.
2. Виконати прогнозування ціни за допомогою методу LightGBM.
3. Проаналізувати отримані результати.

При розв’язанні поставлених завдань застосовано пошуковий, описовий, аналітичний методи аналізу та узагальнення.

Робота, що носить реферативно-дослідницький характер складається із вступу, трьох пунктів основної частини, висновків, списку літератури та додатків. В першому розділі розглядається теоретичне підґрунтя для розгляду обраної моделі машинного навчання для прогнозування. Перелічено етапи роботи з розробки рекомендаційної системи для прогнозування ціни товару, пояснено такі необхідні означення як дерева рішень, бустинг, бегінг, випадковий ліс, градієнтний спуск, GBM, GBDT та алгоритми пошуку точок розбиття, зокрема гістограмний та алгоритм попереднього сортування. В другому розділі розглядається сама модель, її особливості, ефективність та новизна. Докладно пояснено нововведені алгоритми GOSS та EFB, їх мотивацію, особливості. В третьому розділі проведено розвідувальний аналіз набору даних, наведено приклад застосування моделі на базі даних та дані поняття для розуміння параметрів моделі. Оцінено модель на навчальних даних. Оцінку порівняно з моделлю XGBoost.

Під час написання роботи було використано джерела: наукова та публіцистична література, інтернет ресурси, відео ресурси.

РОЗДІЛ 1

Етапи виконання досліджень

На початку роботи з розробки рекомендаційної системи для прогнозування ціни товару слід зазначити етапи яким слідують при виконанні роботи:

1. Збір даних:

   Даний етап можна назвати одним з найважливіших, адже саме на ньому ви отримуєте дані на яких буде ґрунтуватися вся подальша робота. Саме зараз треба звернути увагу на різні атрибути, які можуть вплинути на ціну продукту, наприклад тип продукту, бренд, місцезнаходження, попит і пропозиція тощо.
2. Попередня обробка даних:

   На цьому етапі наші дані піддаються попередній обробці: чистці, нормалізації стандартизації, тобто ми приводимо наші дані в правильний та зручний для роботи з ними вигляд.
3. Вибір найбільш впливових атрибутів:

   За допомогою розвідувального аналізу ми обираємо ті змінні, які, ймовірно будуть мати значний вплив на прогноз ціни. Для зручності, неважливі змінні можна видалити з датасету.
4. Вибір модель машинного навчання (англ. Machine Learning, ML) для прогнозування:

   Від обраної моделі ML буде залежати вся пророблена робота. Точність, ефективність, швидкість тренувального процесу, можливість роботи з великими наборами даних без втрат – все це повинно бути враховано. Популярними моделями для проблем регресії, в які входить і прогнозування ціни, це лінійна регресія, випадковий ліс (англ. Random Forest) і посилення градієнта (англ. Gradient boosting).
5. Навчання моделі:

Користуючись обраною моделлю, навчити її, вказавши навчальні дані та цільову функцію.

6. Оцінка моделі:

   Етап оцінки продуктивності моделі за показниками середньої квадратичної помилки та р-квадрат.

7. Прогнозування:

   Використання навченої моделі безпосередньо для прогнозування ціни.

Для даної роботи було обрано модель LightGBM, адже фреймворк LightGBM має багато переваг порівняно з іншими моделями. Розробляли її як поліпшення однієї за найпопулярніших моделей – XGBoost. LightGBM використовується для скорочення назви Light gradient-boosting machine і має багато своїх плюсів, якщо порівнювати її з XGBoost. Основна відмінність між ними полягає в конструкції дерева, яка детально обговорюється в розділі 2.

LightGBM базується на алгоритмі градієнтного бустінгу дерев рішень та являється досить популярною в області машинного навчання та аналізу даних, адже є потужним інструментом для розв'язання завдань класифікації та регресії, здатним швидко та ефективно обробляти великі обсяги даних.

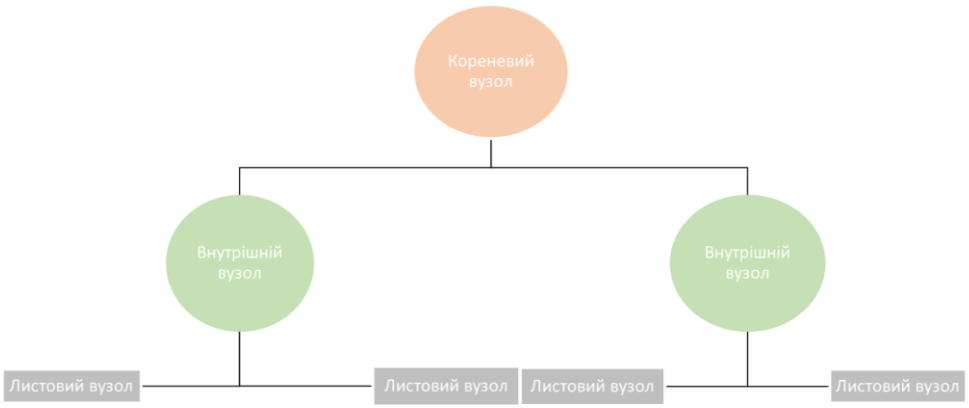
Основні поняття

Дерева рішень

Дерева рішень, більш відомі як Decision Trees, це модель машинного навчання, яка представляє прийняття рішень у вигляді дерева. У цій моделі кожен вузол дерева являє собою рішення, засноване на певній ознаці, а кожне ребро дерева являє собою зв'язок між ознакою і рішенням. Дерева рішень можуть використовуватись як для задач класифікації, так і для регресії. Зазвичай, їх метою є передбачення значення вихідної змінної на основі значень декількох вхідних змінних. Одним з переваг дерев рішень є їхня легка інтерпретованість, що допомагає визначити важливі ознаки, що впливають на процес прийняття рішень.

Дерева мають ієрархічну структуру. Дерево складається з кореневого вузла (англ. Root Node), гілок (англ. Branches), внутрішніх вузлів (англ. Internal Nodes) і листових вузлів (англ. Leaf Nodes). Наведемо просту схему

Схема 1



Як можна побачити на схемі 1 вище, в основі дерева рішень лежить кореневий вузол, який не має жодних вхідних гілок. Вихідні гілки кореневого вузла з'єднуються з внутрішніми вузлами, відомими як вузли прийняття рішень. Обидва типи вузлів розраховують оцінки для формування однорідних підмножин. Вони позначаються листовими вузлами або ж кінцевими вузлами. Листові вузли представляють усі можливі результати в наборі даних.

Аби краще розуміти алгоритм дерева рішень, необхідно зрозуміти, що таке «impurity». Impurity (коефіцієнт незбалансованості) - це міра нечистоти вузла дерева рішень, яка використовується для визначення того, які атрибути краще всього поділяти на дві групи (для класифікації) або розділяти на два діапазони (для регресії).

Одним з методів обчислення коефіцієнту незбалансованості є ***ентропія***.

Ентропія розраховується за формулою

$$Entropy(S) = -\sum_{c \in C} p(c) \log_2(p(c)),$$

де S представляє набір даних, для якого обчислюється ентропія, c представляє класи в наборі S, p(c) представляє частку точок даних, які належать до класу C, до загальної кількості точок даних у наборі S.

Значення ентропії можуть коливатися від 0 до 1. Якщо всі зразки в наборі даних належать до одного класу, тоді ентропія дорівнюватиме нулю. Якщо половину зразків класифікують як один клас, а іншу половину – до іншого класу, ентропія матиме найвище значення 1.

Ще одним методом обчислення незбалансованості є ***індекс Джинні***.

Індекс Джині - це ймовірність неправильної класифікації випадкової точки в наборі даних. Подібно до ентропії, якщо множина S є чистою (тобто належить до одного класу), то її домішка дорівнює нулю. Одиниця означає максимальну нерівність між елементами.

Припустимо, ми маємо $S = (x_{ij}, y_j), y_j \in 1, \dots, c$, де c це кількість різних класів.

Нехай $S_k \subseteq S$ де $S_k = \{(x_i, y) \in S : \text{якщо } y = k\}$ (усі входи з мітками k).

$S = S_1 \cup \dots \cup S_c \dots$

Визначимо:

$$p_k = \frac{|S_k|}{|S|} - \text{частка входів у } S \text{ з міткою } k$$

Звідси індекс Джині розраховується за формулою:

$$Gini\ Index = 1 - \sum_{k=1}^{c} p_k^2$$

Отже, коефіцієнт незбалансованості вимірює однорідність даних, і якщо дані однорідні, то вони належать до одного класу, а дерево рішень розділяється за однорідністю.

Беггінг (Bagging)

Bagging (Bootstrap Aggregating) - це техніка ансамблювання моделей машинного навчання, яка полягає в тому, щоб побудувати кілька однакових моделей на різних підмножинах тренувальних даних та об'єднати їх результати. Він зменшує варіативність (англ. variance) та покращує стійкість (англ. robustness) моделей машинного навчання.

Слабкий закон великих чисел говорить, що для незалежних і однаково розподілених випадкових величин x_i із середнім $\bar{x}$, ми маємо

$$\frac{1}{m}\sum_{i=1}^{m} x_i \rightarrow \bar{x} \text{ , де } m \rightarrow \infty$$

Застосуємо це до класифікаторів: генеруємо m навчальних наборів $D_1, D_2, \dots, D_n$, взятих з навчального набору за допомогою Bootstrap-процедури. Тренуємо класифікатор на кожному та усереднюємо результат:

$$\hat{h} = \frac{1}{m}\sum_{i=1}^{m} h_{D_i} \rightarrow \bar{h}, \qquad \text{де } m \rightarrow \infty$$

Ми називаємо таке середнє значення кількох класифікаторів ансамблем класифікаторів.

Тобто можемо виділити три етапи в процесі беггінгу:

- генеруємо m наборів даних $D_1, \dots, D_m$ з D із заміною.
- для кожного D_j тренуємо класифікатор $h_j(\,)$.
- остаточним класифікатором буде $h(x) = \frac{1}{m}\sum_{j=1}^{m} h_j(x)$.

На практиці більше m призводить до кращого ансамблю, однак у певний момент ми отримаєм меншу віддачу. Зауважимо, що встановлення надмірно високого значення m лише сповільнить роботу класифікатора, але не збільшить похибку класифікатора.

Основна перевага Bagging полягає в тому, що він зменшує ризик перенавантаження моделі, покращує її стійкість до шуму та забезпечує кращу точність передбачення. Крім того, використання Bagging може допомогти вирішити проблему нестабільності алгоритму при невеликих змінах у тренувальному наборі даних.

Бустінг (Boosting)

Boosting — це техніка, яка передбачає послідовне тренування класифікаторів в ансамблі. Відмінністю Boosting від Bagging є те, що у Boosting нові моделі звертають більшу увагу на помилки попередніх моделей.

Bagging вивчає базових учнів із незалежно завантажених підмножин даних, і, отже, ми можемо навчати всіх базових учнів одночасно в паралельному середовищі. Бустінг ж навчає базових учнів послідовно – моделі навчаються одна за одною. Тому навчання базових учнів паралельно неможливо.

Як працює бустінг:

- Присвоюємо кожному навчальному прикладу вагу таким чином, щоб сума ваг дорівнювала 1. На початку, всі ваги прикладів рівні.
- Тренуємо перший класифікатор і перевіряємо, на яких прикладах він помилився.
- Перерозподіляємо ваги таким чином, щоб «помилкові приклади» з попереднього пункту тепер мали більшу вагу, ніж інші (проте сума ваг все одно залишається 1).
- Тренуємо наступний класифікатор. Оскільки якість класифікації обчислюватиметься як зважена сума помилок, другий класифікатор буде зацікавлений «загладити» помилки першого класифікатора.
- Повторюємо процес, доки не натренуємо усі класифікатори.

  Математично матимемо

$$h(x) = \sum_{j=1}^{m} \rho_j h_j(x),$$

де ρ – ваги j-ого класифікатора

Ітеративно вдосконалюючи вагові коефіцієнти навчальних прикладів і навчаючи слабкі моделі для виправлення помилок попередніх, Boosting створює сильну модель, яка може точно класифікувати дані.

Зауваження: українською мовою термін "boosting" можна перекласти як "підсилення" або "усилення", а термін "bagging" як "фіксоване згладжування" або "мішкова модель". Однак у вітчизняній літературі часто використовується саме транслітерація англійських термінів "бустінг" та "беггінг".

Випадковий ліс (Random forest)

Алгоритм дерева рішень досить простий для розуміння та інтерпретації. Але вчені, які працюють з даними, здебільшого використовують саме випадковий ліс, адже одного дерева недостатньо для отримання ефективних результатів. Саме тут на допомогу приходить алгоритм випадкового лісу.

Випадковий ліс чи Random Forest (RF) - це суттєва модифікація методу беггінг, яка будує велику колекцію некорельованих дерев, а потім усереднює їх.

На багатьох задачах ефективність випадкових лісів дуже схожа на бустінг, і їх простіше навчати та налаштовувати. Як наслідок, випадкові ліси є популярними і реалізуються у різноманітних пакетах для роботи з данним.

Основна ідея беггінгу полягає в усередненні багатьох зашумлених, але приблизно незміщених моделей, а отже, у зменшенні дисперсії. Дерева є ідеальними кандидатами для беггінгу, оскільки вони можуть відображати складні структури взаємодії в даних. Оскільки дерева дуже зашумлені, вони значно виграють від усереднення. Більше того, оскільки кожне дерево, згенероване за допомогою беггінгу, має однаковий розподіл, математичне сподівання середнього значення В таких дерев є таким самим, як і сподівання будь-якого з них. Це означає, що зміщення дерев, згенерованих беггінгом, є таким же як і у окремих дерев, і єдиною надією на покращення є зменшення дисперсії.

Середнє значення B незалежних та однаково розподілених випадкових величин, кожна з яких має дисперсію σ^2, має дисперсію $\frac{1}{B}\ \sigma^2$. Якщо змінні просто однаково розподілені (але не обов'язково незалежні) з позитивною парною кореляцією ρ, то дисперсія середнього дорівнює

$$\rho\sigma^2\ +\ \frac{1\ -\ \rho}{B}\ \sigma^2. \qquad (1)$$

Зі збільшенням B другий член зникає, але перший залишається, а отже, розмір кореляції пар дерев, згенерованих за допомогою беггінгу, обмежує переваги усереднення. Одна з основних ідей випадкових лісів полягає у покращенні зменшення дисперсії шляхом зменшення кореляції між деревами, зберігаючи дисперсію на прийнятному рівні. Це досягається в процесі вирощування дерев шляхом випадкового вибору вхідних змінних. Зокрема, при вирощуванні дерева на завантаженому наборі даних:

Перед кожним розбиттям випадковим чином вибирається $m \leq p$ вхідних змінних як кандидатів на розбиття.

Зазвичай значення m дорівнює $\sqrt{p}$ або навіть 1. Після того, як вирощено B таких дерев $\{T(x;\Theta_b)\}^B{}_1$, предиктор випадкового лісу (регресії) має вигляд

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b\ =1}^{B} T(x;\Theta_b).$$

Θ_b описує b-те дерево випадкового лісу за допомогою змінних розбиття, точок розрізу на кожному вузлі та значень термінальних вузлів. Інтуїтивно, зменшення значення m зменшить кореляцію між будь-якою парою дерев у ансамблі, тому за формулою (1) зменшить дисперсію середнього.[2, с.587]

Градієнтний спуск (англ. Gradient Descent)

Метод градієнтного спуску ґрунтується на тому, що якщо функція багатьох змінних $F(x)$ визначена і диференційована в околиці точки a, то $F(x)$ зменшується найшвидше, якщо йти від a у напрямку негативного градієнта F в a,

$-\nabla F(a)$ (символом ∇ в математиці зазвичай позначається градієнт, який є вектором часткових похідних функції). З цього випливає, що якщо $a_{n+1} = a_n - \gamma \nabla F(a_n)$ для достатньо невеликого розміру кроку (англ. step size) або швидкості навчання (англ. learning rate) $\gamma \in \mathbb{R}_+$, тоді $F(a_n) \geq F(a_{n+1})$.[3]

Іншими словами, $\gamma \nabla F(a)$ віднімається від a, тому що ми хочемо рухатися проти градієнта, до локального мінімуму. Знаючи це, ми починаємо з x_0 – припущення щодо локального мінімуму F, розглядаємо послідовність $x_0, x_1, x_2 \ldots$ таких, що $x_{n+1} = x_n - \gamma_n \nabla F(x_n), n \geq 0$.

Отримуємо $F(x_0) \geq F(x_1) \geq F(x_2) \geq \ldots$, ми очікуємо послідовність x_n збігатись до локального мінімуму. Варто зазначити, що розмір кроку γ можна змінювати кожної ітерації.

Оскільки нас цікавить метод градієнтного спуску в машинному навчанні, слід зазначити що таке функція втрат (англ. loss function) та функція витрат (англ. cost function). Адже метою градієнтного спуску є мінімізація функції вартості або похибки між прогнозованим і фактичним *Y*. Для цього потрібні дві точки даних — напрямок і швидкість навчання. Ці фактори визначають обчислення часткових похідних майбутніх ітерацій, що дозволяє поступово досягати локального або глобального мінімуму.

Функція втрати вимірює похибку між прогнозованим результатом і фактичним результатом для одного навчального прикладу, тоді як функція витрат - похибку між прогнозованим і фактичним результатом для всього набору даних, а не лише для окремого прикладу. Іншими словами, функція втрат визначає, наскільки добре модель працює на навчальних даних, а функція вартості - наскільки добре модель працюватиме на нових, невідомих даних.

Функція витрат в градієнтному спуску вимірює її точність на кожній ітерації оновлення параметра. Модель продовжує коригувати параметри, щоб мінімізувати помилку, поки функція не наблизиться до нуля або дорівнюватиме йому.

Припустимо, що в наборі даних є загальна кількість N точок, і для всіх цих точок даних ми хочемо мінімізувати помилку. Отже, функція Cost буде повною квадратичною помилкою, тобто матиме вигляд

$$Cost = \frac{1}{N}\sum_{i=1}^{N}(Y' - Y)^2$$

Обчислення градієнтного спуску

Для прикладу візьмемо рівняння $y = mX+b$. Тут "m" і "b" - це його параметри. У процесі навчання їхні значення дещо зміняться. Позначимо цю невелику зміну через δ. Значення параметрів будуть оновлюватися, тобто маємо $m=m-\delta m$ і $b=b-\delta b$, відповідно. Наша мета полягає в тому, щоб знайти такі значення m і b в $y=mx+b$, для яких похибка буде мінімальною, тобто значення, які мінімізують функцію витрат.

Ми вже знаємо як виглядає функція втрат, проте вона також може позначатися як J, де J – функція втрат від m,b

$$J_{m,b} = \frac{1}{N}\sum_{i=1}^{N}(Y'_i - Y_i)^2$$

Часто, для спрощення замість $Y'_i - Y_i$ пишуть $Error_i$.

Тепер, застосувавши знання правил обчислення похідних до нашого початкового рівняння, ми можемо знайти похідну функції втрат для "m" і "b". Наша функція втрат :

$$J_{m,b} = \frac{1}{N}\sum_{i=1}^{N}(Error_i)^2$$

Для спрощення позбудемося знаку підсумовування. Частина підсумовування є дуже важливою, особливо в концепції стохастичного градієнтного спуску (SGD) на відміну від пакетного градієнтного спуску. Під час пакетного градієнтного спуску розглядаються помилки всіх навчальних прикладів одночасно, тоді як у

SGD - кожна помилка окремо. Однак для спрощення ми будемо вважати, що ми розглядаємо кожну помилку по одній.

$$\frac{\partial J}{\partial m} = 2 \times Error \times \frac{\partial}{\partial m} Error$$

$$\frac{\partial J}{\partial b} = 2 \times Error \times \frac{\partial}{\partial b} Error$$

Тепер обчислимо градієнт похибки як для m, так і для b :

$$\frac{\partial}{\partial m} Error = \frac{\partial}{\partial m} (Y' - Y) \qquad \frac{\partial}{\partial b} Error = \frac{\partial}{\partial b} (Y' - Y)$$

$$\frac{\partial}{\partial m} Error = \frac{\partial}{\partial m} (mX + b - Y), \qquad \frac{\partial}{\partial b} Error = \frac{\partial}{\partial b} (mX + b - Y),$$

де X, b, Y - константи де X, m, Y – константи

$$\frac{\partial}{\partial m} Error = X \qquad \frac{\partial}{\partial b} Error = 1$$

Далі ми вставляємо значення назад у функцію втрат і множимо її на швидкість навчання, тобто:

$$\frac{\partial J}{\partial m} = 2Error \times X \times Learning\ rate \qquad \frac{\partial J}{\partial b} = 2Error \times 1 \times Learning\ rate,$$

де другий елемент множення визначає напрямок мінімізації помилки, а $Learning\ rate$, тобто швидкість навчання, визначає довжину кроку, який треба зробити.

Цифра 2 в цих рівняннях не така вже й важлива, оскільки вона просто говорить про те, що ми маємо вдвічі більшу або вдвічі меншу швидкість навчання. Тож ми позбудемося й його теж.

Отже, зрештою, всі ці кроки привели нас до двох простих рівнянь, що представляють рівняння для градієнтного спуску:

$$\frac{\partial J}{\partial m} = Error \times X \times Learning\ rate \qquad \frac{\partial J}{\partial b} = Error \times Learning\ rate$$

оскільки m=m-δm та b=b-δb, маємо:

$$m^1 = m^0 - Error \times X \times Learning\ rate \quad b^1 = b^0 - Error \times Learning\ rate,$$

де m^1, b^1 - параметри наступної позиції; m^0, b^0 - параметри поточної позиції.

Отже, щоб знайти градієнт, ми ітераційно перебираємо наші точки даних, використовуючи нові значення m і b, і обчислюємо часткові похідні. Цей новий градієнт показує нам нахил нашої функції витрат у нашій поточній позиції і напрямок, в якому ми повинні рухатися, щоб оновити наші параметри. Розмір нашого оновлення контролюється швидкістю навчання.[4]

Gradient Boosting Machine

Gradient Boosting Machine - машина посилення градієнта, далі просто GBM. GBM - це метод машинного навчання, який застосовується для розв'язання задач класифікації та регресії. Він створює модель прогнозування шляхом поєднання багатьох слабких моделей прогнозування. GBM будує модель ітеративно, як і інші методи бустінгу, але є більш узагальненим, оскільки дозволяє оптимізувати будь-яку диференційовану функцію втрат.

Нехай ми маємо тренувальний набір $\{(x_i, y_i)\}_{i=1}^n$, диференційовану функцію втрат $L(y, F(x))$ та кількість ітерацій M.

Основний принцип роботи GBM можна розбити на наступні кроки:

1. Ініціалізуємо модель початковим значенням
   $F_0(x) = \underset{\gamma}{arg\,min} \sum_{i=1}^n L(y_i, \gamma)$, де
2. На кожній ітерації від m=1 до M виконується:
   a) Розраховуємо узагальнені або так звані псевдо-залишки r, які є градієнтом функції втрат L відносно прогнозованих значень F в попередній ітерації.
   $$r_{im} = -\left[\frac{dL\big(y_i, F(x_i)\big)}{dF(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{для } i = 1, \dots, n.$$
   b) Навчаємо базового учня (наприклад, дерево) $h_m(x)$ на псевдо-залишках, тобто тренуємо його на навчальному наборі $\{(x_i, y_i)\}_{i=1}^n$.

c) Обчислюємо множник γ_m, розв'язавши таку задачу одновимірної оптимізації (мінімізуємо функцію втрат):

$$\gamma_m = \underset{\gamma}{arg\,min} \sum_{i=1}^{n} L\big(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\big).$$

d) Оновлюємо модель:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. На виході отримуємо $F_M(x)$.

GBM зазвичай використовується разом з деревами рішень як базовими моделями, тому часто це поєднання називають саме Gradient Boosting Decision Trees (GBDT). Тобто можемо стверджувати, що GBM є різновидом ансамблю, а GBDT - окремий випадок, коли дерево використовується як оцінювач. Однак, GBM може використовувати і інші базові моделі, наприклад, лінійні моделі чи нейронні мережі. Але саме дерева рішень є найпоширенішим вибором для базових моделей в GBM.[2, с.360]

Gradient Boosting Decision Tree

GBDT (Gradient Boosting Decision Tree) — це модель ансамблю дерев рішень, які навчаються послідовно, є широко використовуваним алгоритмом машинного навчання завдяки своїй ефективності, точності та інтерпретованості, має чимало ефективних реалізацій, таких як XGBoost і pGBRT. GBDT поєднує потужність дерев рішень із технікою бустінгу, може виконувати різноманітні завдання машинного навчання, таких як багатокласова класифікація, прогнозування кліків, регресія та ранжування з високою точністю.

На кожній ітерації GBDT вивчає дерево рішень, підбираючи залишкові помилки (помилки до поточної ітерації). Це означає, що кожен наступний учень намагається дізнатися різницю між фактичним результатом і зваженою сумою прогнозів до попередньої ітерації. Помилки мінімізуються за допомогою градієнтного методу. Градієнт представляє напрямок найкрутішого спаду функції

втрат, який GBDT використовує для пошуку найкращих точок розбиття для створення дерева.

В GBDT навчання дерева рішень - найдорожча операція, а пошук оптимальних точок розбиття - найбільш трудомістке завдання.

Одним з найпопулярніших алгоритмів пошуку точок розбиття є алгоритм попереднього сортування (англ. *Pre-sorted algorithm*), який перераховує всі можливі точки розбиття за попередньо відсортованими значеннями ознак. Цей алгоритм простий і може знайти оптимальні точки розбиття, однак він неефективний як за швидкістю навчання, так і за споживанням пам'яті.

Іншим популярним алгоритмом є алгоритм на основі гістограми (англ. *Histogram based algorithm*). Замість того, щоб знаходити точки розбиття відсортованих значень ознак, алгоритм на основі гістограм розбиває неперервні значення ознак на дискретні відсіки і використовує ці відсіки для побудови гістограм ознак під час навчання.[5]

Надалі ми будемо розглядати модель, що використовує саме алгоритм на основі гістограм, оскільки він є ефективнішим як за споживанням пам'яті, так і за швидкістю навчання, тому й саме про нього розповідаємо більш детально.

Алгоритм на основі гістограм

Основна ідея алгоритму гістограми полягає в тому, аби дискретизувати послідовні власні значення на k цілих чисел і побудувати гістограму шириною k. При обході даних дискретизоване значення діє як індекс для накопичення статистики в гістограмі. Після одноразового обходу даних гістограма накопичує необхідну статистику і згодом проходить ще раз для пошуку оптимальної точки розбиття. Враховуючи, що алгоритм на основі гістограми зберігає дискретні біни замість неперервних значень ознак, пучок ознак можна побудувати, дозволивши взаємовиключним ознакам перебувати в певному діапазоні бінів. Цього можна досягти, збільшуючи зсув початкового значення ознаки.

Алгоритм гістограми все ж залишається недосконалим, адже оскільки ознаки дискретизовані, точка поділу не є суттєво точною, що, ймовірно, впливає на результат. Але результати на ряді наборів даних показують, що дискретизовані точки розбиття незначно впливають на кінцеву точність, і точність може збільшитися. Причина полягає в тому, що дерево рішень спочатку є слабкою моделлю, і не дуже важливо, наскільки точною є точка розбиття; більше того, не дуже точна точка розбиття має регуляризуючий ефект, здатний ефективно запобігти перенавчанню; навіть якщо помилка навчання одного дерева трохи більша, ніж у точно розбитого алгоритму, це не спричиняє суттєвої різниці в рамках градієнтного бустінгу. Ідея гістограмного алгоритму проілюстрована на малюнку А в додатках, а також нижче наведено псевдо код реалізації алгоритму.[6]

Алгоритм 1: Алгоритм на основі гістограми

Вхідні дані: *I* - навчальні дані, *d* - максимальна глибина

Вхідні дані: *m* - розмірність ознаки

nodeSet ← {0} ► вузли дерева на поточному рівні

rowSet ← {{0, 1, 2, ...}} ► індекси даних у вузлах дерева

for *i = 1* **to** *d* **do**

for *node* **in** *nodeSet* **do**

usedRows ← rowSet[node]

for *k = 1* **to** *m* **do**

H ← new Histogram() ► Побудова гістограми

for *j* **in** *usedRows* ***do***

bin ← I.f[k][j].bin

H[bin].y ← H[bin].y + I.y[j]

H[bin].n ← H[bin].n + 1

Знаходження найкращого розбиття гістограми H.

...

Оновлення rowSet та nodeSet відповідно до найкращих точок розбиття.

...

Як показано вище, алгоритм на основі гістограм знаходить найкращі точки розбиття на основі гістограм ознак. Він коштує O(#data × #feature) для побудови гістограми та O(#bin × #feature) для знаходження точок розбиття. Оскільки #bin зазвичай набагато менший за #data, побудова гістограми буде домінувати в обчислювальній складності. Якщо ми зможемо зменшити #data або #feature, ми зможемо суттєво пришвидшити навчання GBDT.

РОЗДІЛ 2

Ознайомлення з LightGBM.

В даному розділі ми розглядаємо безпосередньо принципи роботи LightGBM. В чому його переваги, що в нього унікального та як це працює.

LightGBM є одним з найновітніших типів GBDT. Дана модель була розроблена групою дослідників Microsoft у 2016 році. Розробляли її як поліпшення однієї за найпопулярніших моделей – XGBoost, відомою своєю швидкістю та надійністю для проектів класифікації кількох класів. Простіше кажучи, LightGBM представляє дві нові функції, яких немає в XGBoost. Варто зазначити, що LightGBM та XGBoost — це лише одиничні типи із кількох дерев рішень із підвищенням градієнта (GBDT).

Чому ж з’явилася потреба покращити XGBoost? Відповідь дуже тривіальна – ми хочемо ще більш ефективну та ще більш швидку реалізацію. Як ми зазначали в першому розділі, найбільш трудомістке завдання в GBDT це пошук оптимальних точок розбиття, їхня обчислювальна складність буде пропорційна як кількості ознак, так і кількості екземплярів. Адже дерева в GBDT навчаються послідовно шляхом оцінки залишкових помилок кожної ітерації та вдосконалення наступної, і кожної ітерації GBDT повинен обчислити приріст інформації для всіх екземплярів та врахувати всі можливі точки розділення при цьому. Через це, при роботі з великими даними ми стикаємось з проблемами через недостатню швидкість.

Щоб вирішити цю проблему, розробники Microsoft висунули просту ідею, яка полягає в тому, щоб зменшити кількість екземплярів даних і кількість характеристик.[6] Як результат було запропоновано дві нові техніки: Gradient-based One-Side Sampling (GOSS) - одностороння вибірка на основі градієнта та Exclusive Feature Bundling (EFB) - об’єднання ексклюзивних характеристик.

Що собою являє GOSS.

В GBDT відсутня індивідуальна вага для кожного екземпляра даних, проте розробники LightGBM виявили, що екземпляри з різними значеннями градієнту мають різну вагу при обчисленні інформаційного приросту.

Відповідно до визначення приросту інформації, екземпляри з більшими значеннями градієнту (тобто менш навчені зразки) мають більший вплив на обчислення інформаційного приросту. Тому, якщо необхідно зменшити обсяг вибірки даних з метою збереження точності обчислення інформаційного приросту, краще залишити екземпляри з більшими значеннями градієнту і випадковим чином відкинути екземпляри з меншими значеннями градієнту. Такий підхід дозволяє отримати більш точну оцінку інформаційного приросту, особливо якщо значення приросту інформації має великий діапазон [5].

Щоб збалансувати вплив розподілу даних, GOSS вводить постійний множник для екземплярів даних з малими градієнтами (Алгоритм 2), що дозволяє компенсувати їх відносний внесок у розподіл даних. Спочатку алгоритм сортує екземпляри даних за абсолютною величиною їхніх градієнтів та вибирає верхні $a \times 100\%$ з них. Потім з решти даних випадковим чином вибирається $b \times 100\%$ екземплярів. При обчисленні інформаційного приросту, GOSS підсилює відібрані дані з меншими градієнтами на постійну величину $\frac{1-a}{b}$, щоб приділити більше уваги менш навченим екземплярам, не змінюючи вихідного розподілу даних.[6, с. 4]

Для більш зручного розуміння алгоритму нижче наведено псевдо код, де чітко прописано кроки його реалізації.

Алгоритм 2: GOSS

Вхідні дані: I : навчальні дані, d : ітерації

Вхідні дані: a: частота дискретизації даних з великим градієнтом

Вхідні дані: b: частота дискретизації даних з меншим градієнтом

Вхідні дані: *loss* - функція втрат, *L* - моделі слабкого навчання ← {}, fact ← $\frac{1-a}{b}$,

topN ← a × len(I) , randN ← b × len(I)

for i = 1 **to** d **do**

preds ← models.predict(I)

g ← loss(I, preds), w ← {1,1,...}

sorted ← GetSortedIndices(abs(g))

topSet ← sorted[1:topN]

randSet ← RandomPick(sorted[topN:len(I)], randN)

usedSet ← topSet + randSet

w[randSet] × = fact ► Присвоїти вагу fact даним малого градієнта

newModel ← L(I[usedSet], − g[usedSet], w[usedSet])

models.append(newModel)

Підійдемо до питання з ,більш теоретичного боку, GBDT використовує дерева рішень для навчання функції з вхідного простору χ^s у градієнтний простір *G*. Припустимо, що ми маємо навчальну множину з n екземплярів $\{x_1 , x_2,... x_n\}$, де кожен x_i є вектором з розмірністю s у просторі χ^s. На кожній ітерації градієнтного підсилення ми обчислюємо від'ємні градієнти функції втрат відносно результати моделі, які позначаються як $\{g_1 , g_2,. . . g_n\}$.

Для побудови моделі дерева рішень кожен вузол розділяється за найбільш інформативною ознакою (з найбільшим інформаційним приростом). У випадку GBDT, інформаційний приріст, як правило, вимірюється дисперсією після розбиття, що визначається за наступним означенням.

Означення (Дисперсія після розбиття)

Нехай O є тренувальною вибіркою на фіксованому вузлі дерева прийняття рішень. Приріст дисперсії від розбиття ознаки j на точці d для цього вузла визначається як

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{\left(\sum_{\{x_i \in O:\, x_{ij} \le d\}} g_i\right)^2}{n^j_{l|O}(d)} + \frac{\left(\sum_{\{x_i \in O:\, x_{ij} > d\}} g_i\right)^2}{n^j_{r|O}(d)}\right),$$

де $n_O = \sum I[x_i \in O]$, $n^j_{l|O} = \sum I\left[x_i \in O : x_{ij} \le d\right]$ та

$$n^j_{r|O} = \sum I\left[x_i \in O : x_{ij} > d\right]$$

Для ознаки j, алгоритм дерева рішень обирає $d_j^* = argmax_d V_j(d)$ і обчислює найбільший приріст $V_j(d_j^*)$. Потім дані розбиваються за ознакою j^* в точці d_j^* на ліву та праву дочірні вершини.

У запропонованому розробниками Microsoft новому алгоритмі GOSS, спочатку відбираємо верхні $a \times 100\%$ навчальних прикладів з більшими градієнтами та створюємо з них підмножину A. З залишку навчальних прикладів, які мають менші градієнти, випадковим чином вибираємо підмножину B. Потім приклади з підмножини $A \cup B$ розбиваються відповідно до оціненого приросту дисперсії $\tilde{V}_j(d)$.

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{n^j_l(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n^j_r(d)}\right),$$

де $A_l = \{x_i \in A : x_{ij} \le d\}$, $A_r = \{x_i \in A : x_{ij} > d\}$,

$B_l = \{x_i \in B : x_{ij} \le d\}$, $B_r = \{x_i \in B : x_{ij} > d\}$, а коефіцієнт $\frac{1-a}{b}$ використовується для нормалізації суми градієнтів назад до розміру A^c.

Крім того, для методу GOSS справедлива така теорема

Теорема (Про точність навчання моделі)

Позначимо похибку апроксимації в GOSS через $\mathcal{E}(d) = \left|\tilde{V}_j(d) - V_j(d)\right|$ та $\bar{g}_l^j = \frac{\Sigma_{x_i \in (A \cup A^c)_l} |g_i|}{n_l^j(d)}$, $\bar{g}_r^j = \frac{\Sigma_{x_i \in (A \cup A^c)_r} |g_i|}{n_r^j(d)}$. З ймовірністю що найменш 1 – δ маємо

$$\mathcal{E}(d) \leq C_{a,b}^2 \ln {}^1\!/_\delta \cdot max\left\{\frac{1}{n_l^j(d)}, \frac{1}{n_r^j(d)}\right\} + 2DC_{a,b}\sqrt{\frac{\ln {}^1\!/_\delta}{n}}, \qquad (*)$$

де $C_{a,b} = \frac{1-a}{\sqrt{b}} max_{x_i \in A^c} |g_i|$, та $D = max\left(\bar{g}_l^j(d), \bar{g}_r^j(d)\right)$.

Дана теорема пояснює, що при використанні оцінених значень $\tilde{V}_j(d)$ на підмножині даних замість точних значень $V_j(d)$ на всіх даних, алгоритм GOSS здатен досягати майже такої ж точності, як і при використанні точних значень. Теорема надає оцінку похибки апроксимації (позначена як $\mathcal{E}(d)$), яка може бути контрольована. З ймовірністю не менше ніж $1 - \delta$, ця оцінка може бути обмежена значенням, який залежить від розміру підмножини даних та максимального значення градієнту.

Тобто, теорема показує, що алгоритм GOSS є ефективним у зменшенні обчислювальних витрат за рахунок використання підмножин даних та не втрачає значну точність порівняно з випадковим вибором даних.

Згідно з теоремою, ми маємо наступні твердження:

1) Асимптотичний відносний рівень апроксимації для GOSS дорівнює

$$\mathcal{O}\left(\frac{1}{n_l^j(d)} + \frac{1}{n_r^j(d)} + \frac{1}{\sqrt{n}}\right).$$

Якщо розбиття не є дуже незбалансованим, тобто $n_l^j(d) \geq \mathcal{O}\left(\sqrt{n}\right)$ та $n_r^j(d) \geq \mathcal{O}\left(\sqrt{n}\right)$,то другий член нерівності * буде домінувати в похибці апроксимації і зменшується до $\mathcal{O}\left(\sqrt{n}\right)$ при $n \to \infty$. Це означає, що при великій кількості даних апроксимація є достатньо точною.

2) Спробуємо проаналізувати загальну ефективність узагальнення в GOSS. Ми розглядаємо похибку узагальнення в GOSS як

$$\mathcal{E}_{gen}^{GOSS}(d) = \left|\tilde{V}_j(d) - V_*(d)\right|,$$

де $V_*(d)$ - справжній приріст дисперсії для базового розподілу.

Звідси ми маємо

$$\mathcal{E}_{gen}^{GOSS}(d) \leq \left|\tilde{V}_j(d) - V_j(d)\right| + \left|V_j(d) - V_*(d)\right| \triangleq \mathcal{E}_{GOSS}(d) + \mathcal{E}_{gen}(d)$$

Таким чином, якщо апроксимація GOSS є точною, то узагальнення за GOSS буде близькою до тієї, яка обчислюється за допомогою повного набору даних.

Підсумовуючи, кроки алгоритму GOSS пояснюються наступним чином:

Вхідні дані: навчальні дані, крок ітерації d, частота дискретизації даних з великим градієнтом - a, частота дискретизації даних з малим градієнтом - b, функція втрат та тип слабкого навчання;

Вихідні дані: навчений сильний учень;

i. Відсортувати точки вибірки у порядку спадання відповідно до їх абсолютних значень;
ii. Вибрати перші a×100% вибірок відсортованих результатів, щоб отримати підмножину точок вибірки з великим градієнтом;
iii. Для решти вибірки (1-a)×100% вибірок випадковим чином вибрати b×(1-a)×100% вибірок, щоб отримати множину точок вибірки з малим градієнтом;
iv. Об'єднати вибірку з великим градієнтом з вибірками з малим градієнтом;
v. Помножити вибірки з малим градієнтом на ваговий коефіцієнт;
vi. Використовувати згадані вибірки для навчання нового слабкого учня;
vii. Повторювати кроки з і по vi безперервно, доки не буде досягнуто заданої кількості ітерацій або збіжності.

Алгоритм GOSS дозволяє значно зменшити швидкість навчання моделі, зберігаючи точність навчання без зміни розподілу даних. У багатьох випадках

точність моделі, навченої за алгоритмом GOSS, вища, ніж за алгоритмом випадкової вибірки. Крім того, вибірка також збільшує різноманітність слабких учнів, тим самим потенційно покращуючи узагальнюючу здатність навченої моделі.

Що собою являє EFB.

Ми розглядаємо новий метод запропонований для ефективного зменшення кількості ознак.

Високовимірні дані зазвичай містять велику кількість ознак, що може призвести до проблеми перенавчання моделі. Однак, розрідженість простору ознак є типовим явищем. Розрідженність дозволяє розробити підхід до зменшення кількості ознак, майже без втрати інформації.

Важливою особливістю розрідженого простору ознак є те, що багато з них є взаємовиключними, тобто не приймають ненульових значень одночасно. Це означає, що можна безпечно об'єднати виключні ознаки в одну ознаку, яку можна назвати "пакетом виключних ознак" (англ. exclusive feature bundle).

За допомогою ретельно розробленого алгоритму сканування ознак можна побудувати гістограми ознак із пакетів виключних ознак, як і з окремих. Це дозволяє зменшити складність побудови гістограми з O(#data × #feature) до O(#data × #bundle), при цьому кількість пакетів виключних ознак #bundle значно менше за кількість окремих ознак #feature.

Таким чином, можна значно прискорити навчання градієнтного бустингу дерев рішень (англ. gradient boosting decision tree, GBDT), зберігаючи точність моделі.

Проте тут ми стикаємось з двома проблемами, які потрібно вирішити. По-перше, потрібно визначити, які ознаки слід об'єднати в один пакет. По-друге, потрібно знайти спосіб створення цього пакету.

Щодо першої проблеми, ми стикаємось з тим, що знаходження оптимальної стратегії об'єднання є NP-важкою задачею, тому точного розв'язку за поліноміальний час не існує. Однак, щоб знайти ефективний алгоритм наближення, ми можемо зводити задачу до розфарбовування графу, де вершини представляють об'єкти, а ребра вказують на те, які об'єкти можуть бути об'єднані. Жадібний алгоритм може дати досить точні результати для розфарбовування графу з постійним коефіцієнтом апроксимації.

Також, зазвичай існує багато ознак, які можуть бути об'єднані разом, бо вони рідко мають ненульові значення одночасно. Якщо наш алгоритм може дозволити деякі конфлікти, то можна отримати ще менше пакетів ознак і ще більшу обчислювальну ефективність. Якщо випадково забруднити невелику частину значень ознак, то це не сильно вплине на навчання, якщо максимальна частота конфліктів у кожному пакеті становить γ. За простими підрахунками, точність навчання не зменшиться більше, ніж на $\mathcal{O}\left([(1-\gamma)n]^{-2/3}\right)$, де γ - це загальна кількість ознак. Таким чином, вибір невеликого значення γ дозволить досягти гарного балансу між точністю та ефективністю

На основі вищезазначеного, ми маємо алгоритм для ексклюзивного пакетування ознак.

Псевдо алгоритм 3 наведено нижче.

Алгоритм 3: Жадібне пакування (Greedy Bundling)

Вхідні дані: *F*: ознаки, *K*: максимальна кількість конфліктів

Будуємо граф G

searchOrder ← G.sortByDegree()

bundles ← {}, bundlesConflict ← {}

for *i* **in** *searchOrder* **do**

needNew ← True

for *j = 1* **to** *len(bundles)* **do**

cnt ← ConflictCnt(bundles[j],F[i])

if *cnt + bundlesConflict[i]* ≤ *K* **then**

bundles[j].add(F[i]), needNew ← False

break

if *needNew* **then**

Додаємо F[i] як новий пакунок до пакунків

Вихідні дані: bundles

Алгоритм працює наступним чином:

1. Створюється граф G, де вузлами є об'єкти заданої множини , а ребра з'єднують об'єкти, які конфліктують між собою.
2. Граф G сортується за ступенем вузлів в порядку спадання.
3. Створюються порожні пакунки (bundles) та списки конфліктів між пакунками (bundlesConflict).
4. Для кожного вузла *i* відбувається наступне:
    - Перевіряється, чи можна додати вузол до наявних пакунків без конфліктів за допомогою перебору всіх наявних пакунків та порівняння кількості конфліктів (за допомогою функції ConflictCnt()).
    - Якщо вузол можна додати до наявних пакунків без конфліктів, то вузол додається до пакунку, а прапорець needNew ставиться в False
    - Якщо вузол не можна додати до жодного існуючого пакунку, то створюється новий пакунок та до нього додається вузол *i*.
5. Після проходження всіх вузлів, алгоритм повертає список bundles - пакунків, які містять об'єкти без конфліктів між собою

Часова складність Алгоритму 3 становить $O(\#\,\text{feature}^2)$. Це прийнятна складність для невеликої кількості ознак, але може стати проблемою, якщо ознак дуже багато. Для покращення ефективності пропонується використовувати більш ефективну стратегію впорядкування, яка полягає у впорядкуванні за кількістю ненульових значень, що схоже на впорядкування за ступенями, оскільки більша кількість ненульових значень зазвичай призводить до більшої ймовірності конфліктів. Ця стратегія допоможе підвищити ефективність алгоритму при великій кількості ознак.

Оскільки ми змінюємо лише стратегії впорядкування в Алгоритмі 3, деталі нового алгоритму опущено, адже все що потрібно це замінити G.sortByDegree() на G.sortNonZero().

Для вирішення другої проблеми нам необхідний ефективний метод об'єднання ознак в одному пакеті. Головною метою є забезпечення можливості відрізняти значення початкових ознак від пакунків. Оскільки алгоритм на основі гістограми зберігає дискретні біни замість неперервних значень ознак, ми можемо створити пакунок ознак, розмістивши унікальні ознаки в різних бінах.

Це можна реалізувати додавши зміщення до початкових значень ознак. Наприклад, припустимо, що у нас є два елементи у наборі елементів. Спочатку елемент A має значення [0, 10), а елемент B - [0, 20). Потім ми додаємо зсув 10 до значень ознаки B так, щоб уточнена ознака набувала значень з [10, 30). Після цього можна об'єднати ознаки A та B і використати набір ознак з діапазоном

[0, 30] для заміни вихідних ознак A та B.[6]

Для більш точного розуміння, можемо привести детальний приклад псевдо алгоритму:

Алгоритм 4: Об'єднання ексклюзивних ознак (Merge Exclusive Features)

Вхідні дані: *numData*: кількість даних

Вхідні дані: *F* : Один набір ексклюзивних ознак

binRanges ← {0}, totalBin ← 0

for *f* **in** *F* **do**

totalBin += f.numBin

binRanges.append(totalBin)

newBin ← newBin(numData)

for *i = 1* **to** *numData* **do**

newBin[i] ← 0

for *j = 1* **to** *len(F)* **do**

if F[j].bin[i] 6= 0 **then**

newBin[i] ← F[j].bin[i] + binRanges[j]

Вихідні дані: newBin, binRanges

Алгоритм EFB може об'єднати багато ексклюзивних ознак у значно меншу кількість щільних ознак, що дозволяє ефективно уникнути непотрібних обчислень для нульових значень ознак.

Підсумовуючи, кроки алгоритму EFB пояснюються наступним чином:

Вхідні дані: ознака F, максимальна кількість конфліктів K, граф G;

Вихідні дані: пакети функцій;

i. Будується граф з вагами на ребрах, ваги яких відповідають сумарним конфліктам між ознаками;
ii. Ознаки відсортовуються в порядку спадання їх степеня на графі;
iii. Відмічається відповідна ознака у впорядкованому списку і призначається до існуючого пакету з незначними конфліктами, або створюється новий пакет.

Алгоритм LightGBM

Як висновок всього нашого другого розділу ми маємо такий алгоритм роботи LightGBM [7]:

Нехай ми маємо тренувальний набір $D = \{(x_i, y_i)\}_{i=1}^{n}$, диференційовану функцію втрат $L(y, F(x))$ та кількість ітерацій M, великий коефіцієнт вибірки градієнтних даних a, невеликий коефіцієнт – b.

1. Комбінуємо функції, які є взаємовиключними (тобто функції, які ніколи не приймають ненульових значень одночасно) $x_i, i = 1, \dots, n$ за допомогою техніки EFB;
2. Ініціалізуємо модель початковим значенням

$$F_0(x) = arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

3. На кожній ітерації від m=1 до M виконується:
   a) Розрахуємо абсолютні значення градієнта:

$$r_i = -\left[\frac{dL\big(y_i, F(x_i)\big)}{dF(x_i)}\right]_{F(x)=F_{m-1}(x)} \text{ для } i = 1, \dots, n.$$

   b) Створюємо повторну вибірку даних за допомогою GOSS::

   $topN = a \times len(D)$; $randN = b \times len(D)$;

   $sorted = GetSortedIndices\big(abs(r)\big)$;

   $A = sorted[1{:}\,topN]$;

   $B = RandomPick(sorted[\,topN{:}\,len(D)], randiN)$;

   $\acute{D} = A + B$;

   c) Обчислюємо інформаційний приріст:

$$V_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} r_i + \frac{1-a}{b}\sum_{x_i \in B_l} r_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} r_i + \frac{1-a}{b}\sum_{x_i \in B_r} r_i\right)^2}{n_r^j(d)}\right)$$

   d) Будуємо нове дерево рішень $F_m(x)'$ на множині D'

   e) Оновлюємо модель:

$$F_m(x) = F_{m-1}(x) + F_m(x\,).$$

4. На виході отримуємо навчену модель $F_M(x)$

РОЗДІЛ 3

Набір даних для реалізації практичної частини для даної кваліфікаційної роботи був завантажений з сайту kaggle. Цей набір даних доступний на сайті під назвою Mercari Price Suggestion Challenge. Заснована у 2013 році, Mercari Inc. є японською компанією, яка управляє одним з найпопулярніших C2C - маркетплейсів на японському ринку. З самого початку діяльності Mercari ставила собі за мету створити популярну, легкодоступну платформу, на якій користувачі могли б продавати та купувати власні товари без обмежень у часі, просторі та цінах.

Дані вже розбиті на сети для тренування та тестування. Розмір файлу train.tsv = 322 МБ. Загальна кількість списків товарів = 1,482,535. Набір даних складається з наступних 7 характеристик:

- назва товару (name),
- ідентифікатор стану товару (item_condition_id),
- назва бренду (brand_name), категорії (category_name), доставка (shipping), опис товару (item_description) та ціна товару (price).

Функція категорій має 3 підкатегорії для кожного товару, таким чином перша підкатегорія є основною категорією, за нею йде відділ і товар категорія. Ціна складається з безперервного значення, яке знаходиться в діапазоні 0-2009. Ідентифікатор стану товару та доставка складаються з категорійних числових значень. Діапазон значень значень в ідентифікаторі стану товару від 1 до 5, тоді як доставка складається лише з 1 та 0. Значення 0 в доставці означає, що інтернет-магазин не сплатив жодних коштів за доставку, тоді як 1 означає, що інтернет-магазин заплатив гроші за доставку цього товару.

Розвідувальний аналіз даних

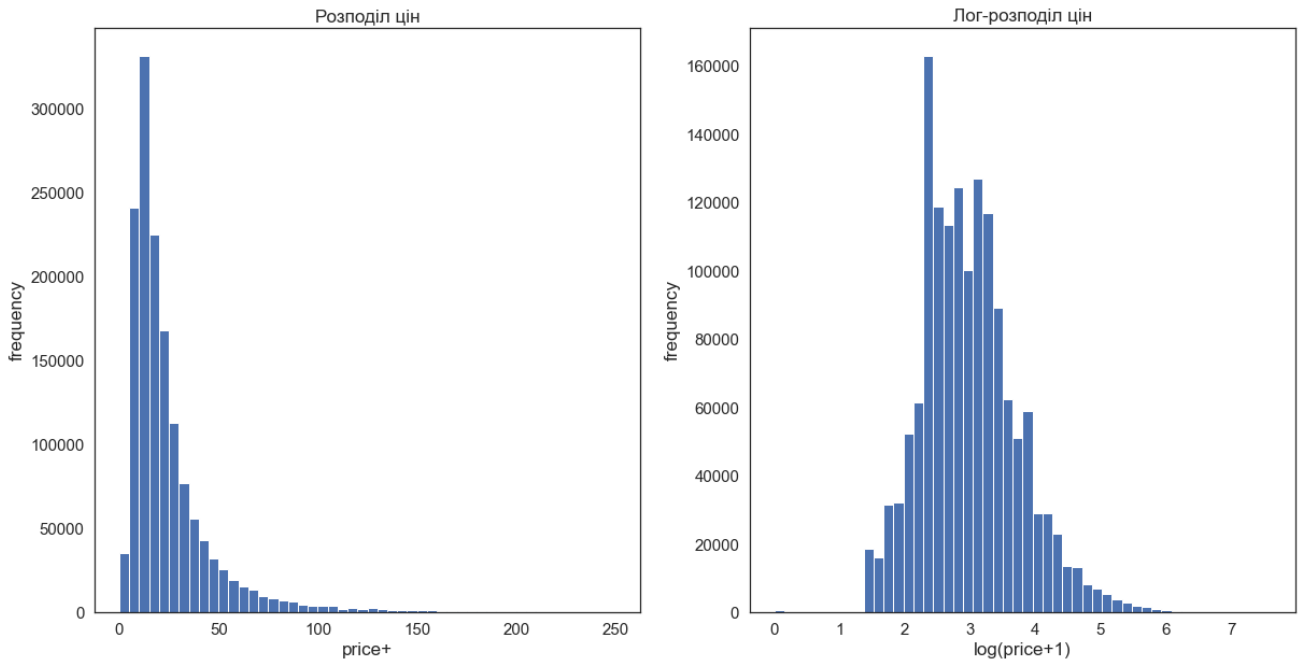
Розвідувальний наліз даних є найважливішою частиною перед проведенням будь-якого типу статистичного або математичного моделювання. Він допомагає

нам зрозуміти дані та отримати кращі уявлення про них. Тільки зробивши це, ми можемо застосувати до них повноцінну функціональну роботу.

Першим кроком буде вибір цільової змінної, якою очевидно, що буде «ціна».

Робимо стандартну перевірку нашої цільової змінної - ціни, яку ми пропонуємо продавцям на ринку Mercari. Медіанна ціна всіх товарів у тренінгу становить близько 267$, але з огляду на існування деяких екстремальних значень, що перевищують 100$, і максимального значення в 2009$, розподіл змінних сильно зміщений вліво. Тому ми зробили лог-перетворення ціни. Ми додали +1 до значення перед перетворенням, щоб уникнути нульових та від'ємних значень.

Гістограми обох розподілів наведено нижче:



Бачимо, що лог-перетворення значно покращило картину сприйняття розподілу даної змінної і цей розподіл близький до розподілу Гаусса або нормального, який має певні переваги перед початковим розподілом. А саме:

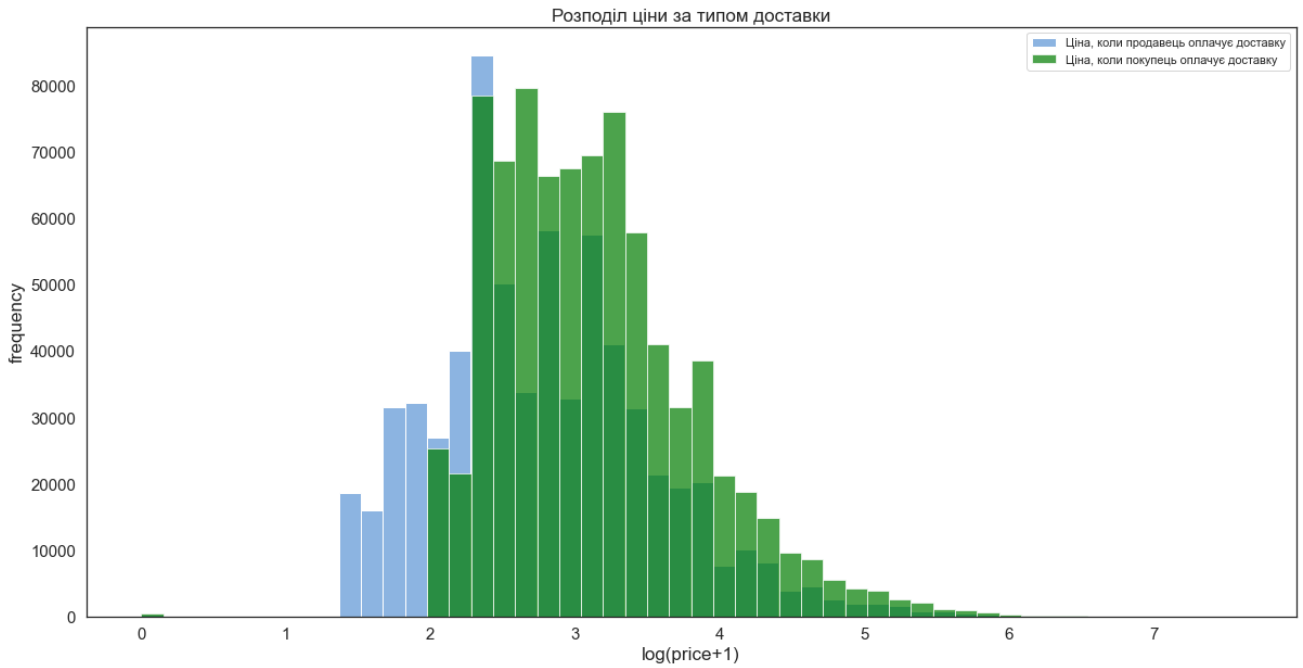
- Значення в гаусівському розподілі набагато більш рівномірно розподілені в усіх діапазонах, на відміну від логнормального розподілу.
- Це дає перевагу моделям на кшталт лінійної регресії, які мають припущення, що залежні змінні розподілені нормально. Цільові змінні з

гауссовим розподілом також забезпечують кращу продуктивність й іншим моделям.

Таким чином, ми будемо вважати log(ціна+1) нашою цільовою змінною.

Наступним нашим кроком буде огляд змінної «доставка».

Ми робимо гістограму розподілу змінної «ціна» та групуємо її за змінною «доставка». Тепер ми чітко бачимо, що витрати на доставку досить добре розподілені між продавцями та покупцями, причому більше половини витрат на доставку товарів оплачують продавці (55%, це ми побачили порахувавши кількість нуликів в змінній доставка, тобто кількість доставок за рахунок отримувача, і поділивши це значення на довжину нашого сету). Крім того, середня ціна, яку платять користувачі, яким доводиться оплачувати вартість доставки, нижча, ніж для тих, хто не потребує додаткових витрат на доставку. Це збігається з нашим сприйняттям того, що продавцям потрібна нижча ціна, щоб компенсувати додаткові витрати на доставку.



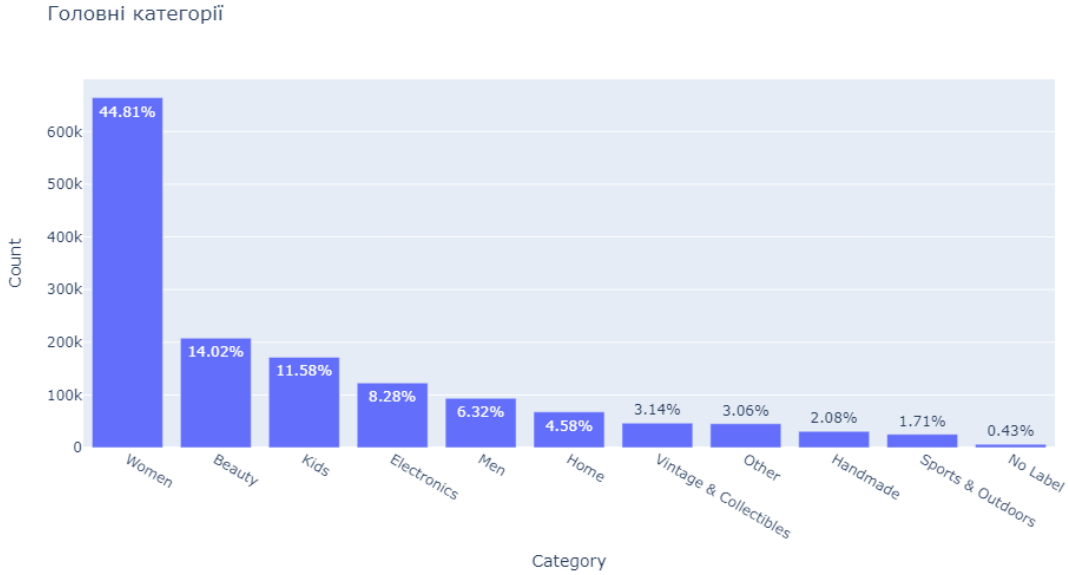
Зеленим на графіку позначено гістограму розподілу ціни для тих випадків, коли доставку оплачує сам покупець. Блакитним – коли доставка здійснюється за рахунок продавця.

Змінна категорій.

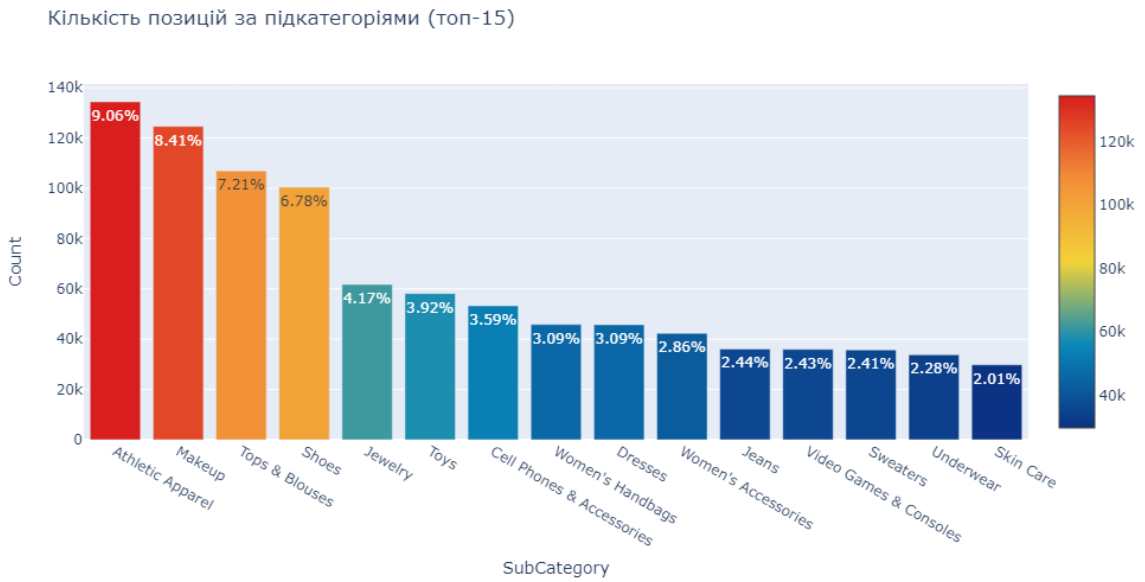
В даному датасеті снує близько 1287 унікальних категорій, в кожній з них ми завжди бачимо спочатку основну/загальну категорію, а потім ще дві конкретні підкатегорії (наприклад, Men/Tops/T-shirts). Але прицьому, є 6327 товарів, які не мають категорій. Для зручності ми розділимо категорії на три різні колонки. Пізніше ми побачимо, що ця інформація насправді досить важлива з точки зору продавця, і те, як ми обробляємо відсутню інформацію в колонці brand_name, вплине на прогноз моделі.

Оскільки підкатегорії дуже зручно розділені позначенням /, нам буде досить просто зробити наш розподіл. З одної змінної ‘category_name’ ми створили три нові: ‘general_cat’ - загальна категорія (Men), ‘subcat_1’ – перші підкатегорії (Tops), ‘subcat_2’ – другі підкатегорії (T-shirts).

Загалом ми маємо 11 основних категорій (114 у перших підкатегоріях і 871 у других підкатегоріях): жіночі товари та товари для краси як дві найпопулярніші категорії (майже 60% спостережень), за ними йдуть дитячі товари та електроніка. Можемо побачити це на наведеному нижче графіку. (Графіки ми будуємо за допомогою бібліотеки plotly – дуже зручної бібліотеки для візуалізації даних)



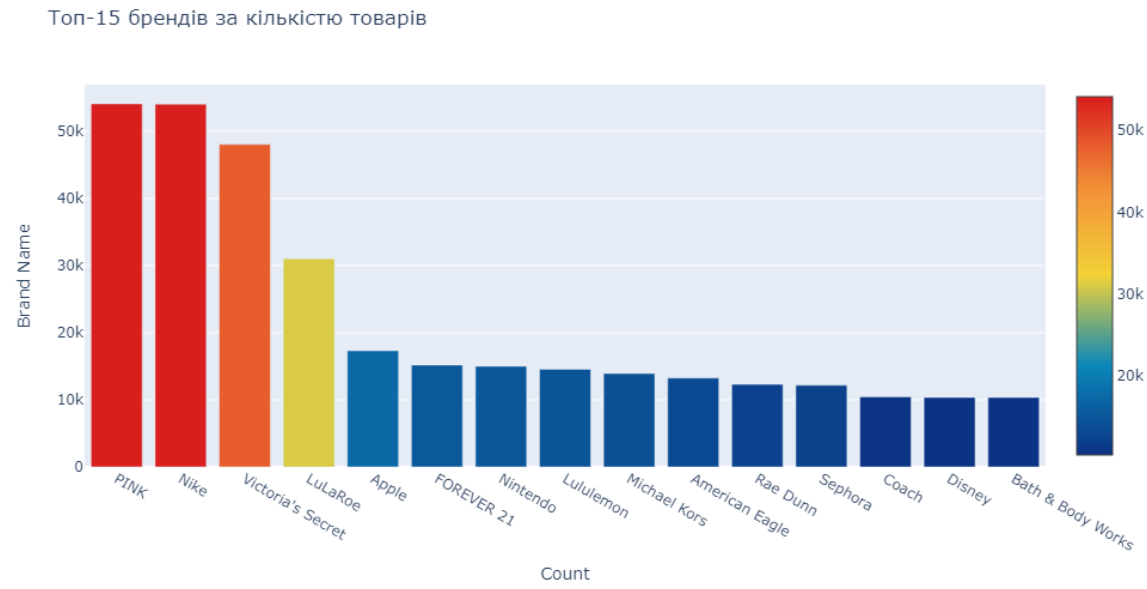
Також зробимо розподіл для перших підкатегорій.



Можемо побачити, що з точки зору ціноутворення, всі категорії досить добре розподілені, немає жодної категорії з екстраординарним рівнем цін.

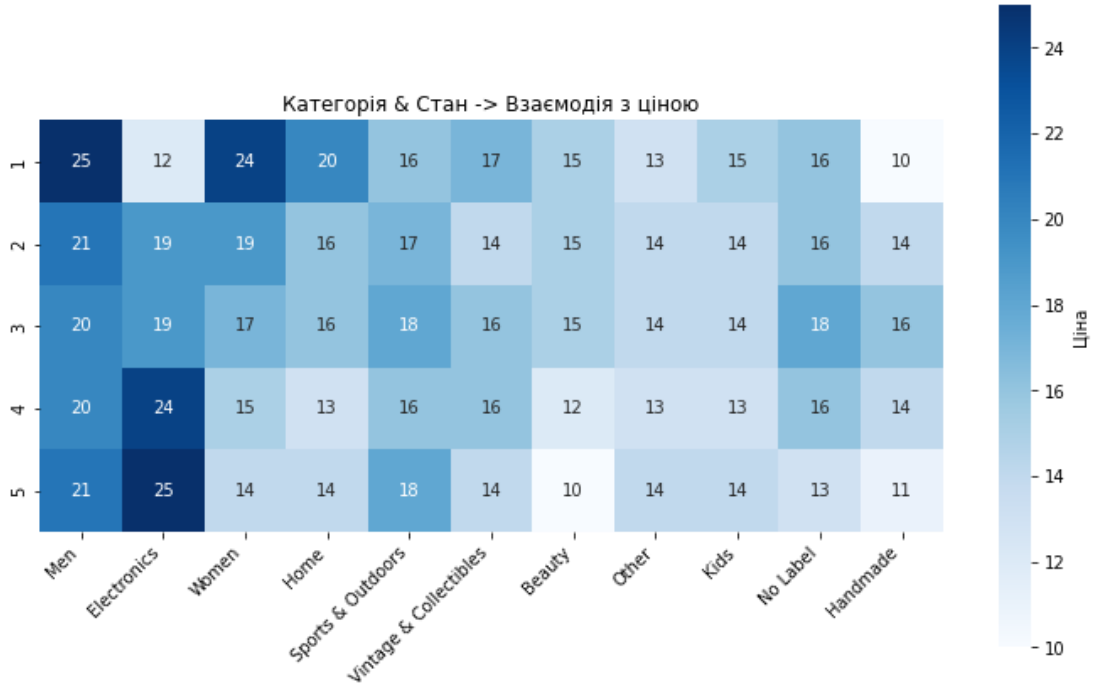
Далі подивимось на змінну назви бренду.

У навчальному наборі наших даних 4809 унікальних торгових марок. Але при цьому, 632682 товарів взагалі не мають бренду. Побудуємо графік розподілу для брендів. Візьмемо топ-15 брендів за кількістю товарів.



Легко побачити, що найбільше пропозицій таких брендів як: PINK – 54.08 тисячі, Nike – 54.04 тисячі , Victoria’s Secret – 48 тисяч, та LulaRoe – 31 тисяча.

Подивимось на взаємодії. Взаємодія між категорією та станом товару показує, що товари категорій «Чоловічі», «Жіночі» та «Дім» продаються найдорожче в стані 1 (стан «Новий»). Електроніка продається найдорожче зі станом 4 і 5 (стан «задовільний» і «поганий»).



Застосування

Приступаємо до створення нашої моделі.

Спочатку, додаємо до нашого тренувального сету тесувальний за допомогою функції cз бібліотеки pandas - concat.

Для ефективнішої роботи моделі, ми повинні підготувати наші данні, а саме:

- Позбутись від всіх найменування, що мають пропущенні данні.

Ми визначили, що змінні category_name, brand_name та item_description мають пропущенні значення. Тому використавши кастомні функції handle_missing_inplace та cutting ми позбуваємось тих рядків нашого набору даних, які мають NA.

- Перетворити тип змінних category_name, brand_name та item_condition_id на категоріальний.

Далі ми застосовуємо CountVectorizer для name та general_cat, subcat_1, subcat_2. CountVectorizer - це метод перетворення тексту в числові дані, полегшує використання текстових даних безпосередньо в моделях машинного та глибинного навчання. Ми також застосовуємо TfidfVectorizer для item_description. TfidfVectorizer — це широко використовуваний метод вилучення ознак тексту в NLP (Natural language processing), який перетворює колекцію необроблених текстових документів на матрицю числових ознак. Це розшифровується як "Term Frequency-Inverse Document Frequency Vectorizer" (Термін Частотно-інверсний векторизатор частоти документа) часто пишуть просто скорочення TF-IDF. Він працює, обчислюючи частоту кожного слова в документі та призначаючи вагу кожному слову на основі того, як часто воно в ньому зустрічається.

Після того, як всі змінні очищені та попередньо оброблені, ми повинні перетворити їх у розріджену матрицю. Матриця CSR (Compressed Sparse Row matrix) - це структура даних, яка використовується для ефективного

представлення розріджених матриць у стислому вигляді. У CSR-матриці ненульові елементи матриці зберігаються у трьох одновимірних масивах: один для значень, один для індексів стовпців і один для вказівників рядків. Це дозволяє пришвидшити операції з матрицями та зменшити використання пам'яті порівняно зі щільним представленням матриці.

Далі ми знову ділимо наш набір на тестувальний та тренувальний, користуючись заздалегідь збереженою довжиною початкового тренувального набору.

Для того, аби оцінити нашу модель тестувальний набір даних ми також розділяємо на тренувальний та валідаційний. Схема розділення набору даних в дослідженнях додана до додатку Б.

Тепер ми підійшли безпосередньо до використання LightGBM.

Імпортуємо: import lightgbm as lgb. Потім нам необхідно створити набір даних для роботи моделі, використовуємо функцію lgb.Dataset(X, y).

Наступним кроком ми встановлюємо параметри моделі, params = {'boosting_type' : 'gbdt', "enable_bundle": "true", "data_sample_strategy":"goss", 'learning_rate': 0.75,'objective': 'regression','max_depth': 3,'num_leaves': 100,'verbosity': -1,'metric': 'RMSE'. Як 'objective' ми встановили 'regression', що в принципі є дефолтним параметром, адже ми маємо задачу регресії, а не класифікації. Наприклад, якби змінна яку ми хочемо спрогнозувати приймала значення один або нуль ми б встановили ‘binary’. Показово, ми також прописали тип бустінгу - 'gbdt', але він також є таким за замовчуванням. Додаємо як метод семплювання "data_sample_strategy" наш метод GOSS. Також додаємо "enable_bundle": "true", тобто кажемо, що и використовуємо техніку EFB. Як оцінювальний параметр 'metric' ми обрали 'RMSE'. RMSE - це середньоквадратична помилка. Чим нижчий показник, тим краща продуктивність. Вибір цієї метрики обумовлений наступними причинами,

- Вона стійка до викидів.

- Вона є інваріантною до масштабу.
- Занижена оцінка карається більше, ніж завищена.

Це має сенс з точки зору бізнесу, тому що в ідеалі компанія хотіла б продавати продукцію за досить високою ціною. Це також підвищує задоволеність клієнтів (продавців), а також комісію, яку компанія може отримати за кожен продаж. Інша причина полягає в тому, що розподіл цін є сильно зміщеним вправо (це ми побачили в розвідувальному аналізі). Це означає, що існує багато продуктів за низькими цінами порівняно з продуктами за дуже високими цінами.

RMSE має формулу:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

де n – це загальна кількість спостережень в наборі даних, $\hat{y}_i$ - зпрогнозована ціна, y_i – реальна ціна.[8]

Також ми встановили максимальна кількість листь, які може мати одне дерево – 100.

Тепер можемо тренувати нашу модель. Користуємось функцією lgb.train(param, train_data, num_round). Далі робимо передбачення та отримуємо шукані значення bst.predict(data).

Аби оцінити роботу створеної моделі, імпортуємо за бібліотеки sklearn.metrics такі методи як r2_score, mean_absolute_error, mean_squared_error. За їх допомогою ми знайдемо R^2 , MAE та RMSE відповідно, де середня абсолютна помилка (англ. Mean absolute error, MAE) розраховується за формулою:

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}|,$$

а коефіцієнт детермінації (англ. Coefficient of determination, R^2)за наступною формулою:

$$R^2=1-\frac{SSR}{SST}=1-\frac{\sum(y_i-\hat{y})^2}{\sum(y_i-\bar{y})^2}$$

Для порівняння ми також реалізували модель XGBoost, про цю модель ми згадували в попередніх розділах, адже саме вона була модернізована до вигляду LightGBM.

Слід зазначити, що в нашому експерименті ми використовували класичні реалізації обох методів, з ідентичними параметрами.

Отримали таку порівняльну таблицю оцінок моделей тренованих на навчальних за 1000 ітерацій.

Metric	**LGBM**	**XGBoost**	**LGBM+XGBoost**
RMSE	0.47667	0.47778	0.47441
MAE	0.35779	0.35897	0.35612
R-squared	0.59461	0.59274	0.59845

На тренування LightGBM було витрачено 810.2477 секунд, що дорівнює приблизно 15 хвилинам. Для XGBoost знадобилось 2116.6875 секунд, що становить приблизно 35 хвилину.

Можемо зробити висновок, що на нашому наборі даних LightGBM показує більш гарні оцінки. В останньому стовпчику нашої таблиці ми маємо оцінку комбінованої моделі LightGBM + XGBoost (методом підбору були обрані коефіцієнти 0.9 та 0.1 відповідно). Як бачимо, комбінована модель має найкращі показники. В неї найнижчі помилки та найбільший коефіцієнт детермінації.

Однак, ми бачимо що самостійна модель LightGBM має майже такі ж оцінки що й комбінована, але за значно коротший час, тому ми обираємо саме її як фінальну модель для виконання нашої мети – передбачення ціни товару без історії.

Як результат нашої роботи ми отримаємо оцінену модель прогнозування ціни для наших даних, а також таблицю «Айді товару без ціни – прогнозована ціна». На фото в Додатку В виведено перші п’ять елементів даної таблиці.

ВИСНОВКИ

В результаті проведеного дослідження було побудовано модель для прогнозування ціни на товар без історії за його характеристиками та даними про сусідні схожі товари з використанням методу LightGBM на прикладі набору даних з Mercari - одного з найпопулярніших C2C -маркетплейсів на японському ринку. Ми показали в чому полягає новизна методу LightGBM та як працюють ці нові алгоритми, наведено псевдо-коди.

Отримані результати показали, що модель з використанням методу LightGBM, забезпечує достатньо високу точність прогнозування ціни товару без історії. Таким чином, дослідження підтверджує ефективність використання методу LightGBM для прогнозування ціни. Результати дослідження можуть бути корисними для компаній, які бажають розробити ефективні моделі для прогнозування ціни товарів без історії.

ЛІТЕРАТУРА

Основна література:

1. Marshall A. Principles of economics. London: Macmillan, 1890. 870p
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman. (2008). The Elements of Statistical Learning Data Mining, Inference, and Prediction (2nd ed., p.745). Springer.
3. Градієнтний спуск. (n.d.). Енциклопедія https://uk.zahn-info-portal.de/wiki/Gradient_descent
4. Pandey P. Understanding the Mathematics behind Gradient Descent. Medium. URL: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e.
5. Sharma, A. (2018, October 15). *What makes lightGBM lightning fast?* Medium. https://towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e
6. Ke, Guolin; Meng, Qi; Finley, Thomas; Wang, Taifeng; Chen, Wei; Ma, Weidong; Ye, Qiwei; Liu, Tie-Yan (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems.
7. Dunbray, N., Rane, R., Nimje, S., Katade, J., & Mavale, S. (2021). A Novel Prediction Model for Diabetes Detection Using Gridsearch and A Voting Classifier between Lightgbm and KNN. In 2021 2nd Global Conference for Advancement in Technology (GCAT). IEEE. https://doi.org/10.1109/gcat52182.2021.9587551
8. Akshita Chugh. Mae, mse, rmse, coefficient of determination, adjusted r squared — which metric is better? https://medium.com/analytics-vidhya/ mae- mse-rmse- coefficient- of- determination- adjusted- r- squared- whichmetric-is-better-cd0326a5697e

Додаткова література:

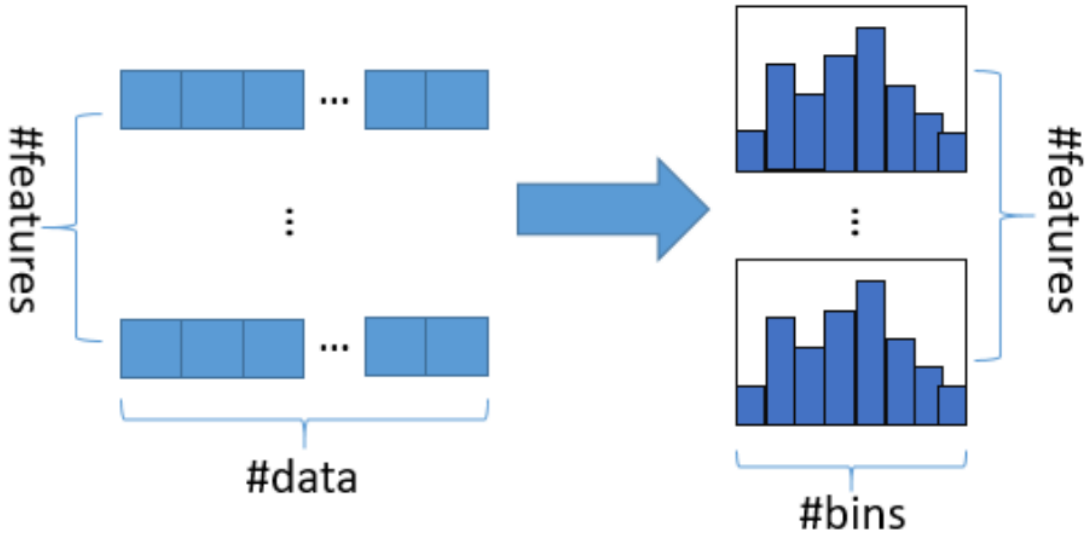
9. LightGBM documentation https://lightgbm.readthedocs.io/en/latest/Features.html
10. Understanding customers better through neural network embeddings - Adam Hornsby, 2018 https://anhornsby.github.io/embeddings-retail/#/
11. Terence Shin (2021). Understanding Feature Importance and How to Implement it in Python https://towardsdatascience.com/understanding-feature-importance-and-how-to-implement-it-in-python-ff0287b20285#:~:text=Feature%20Importance%20refers%20to%20techniques,to%20predict%20a%20certain%20variable
12. Echo Yang, David Chen (11 Feb 2020). Tree-Math, LightGBM. https://github.com/YC-Coder-Chen/Tree-Math/blob/master/LightGBM.md
13. Catherine Lovering. How does pricing variables affect a business? September 2018. How to Calculate Profit Maximization (chron.com)
14. *Price Prediction: How Machine Learning Can Help You Grow Your Sales*. (n.d.). DLabs.AI.https://dlabs.ai/blog/price-prediction-how-machine-learning-can-help-you-grow-your-sales/#:~:text=Price%20prediction%20uses%20an%20algorithm,price%20forecasting%20or%20predictive%20pricing.
15. *What is Gradient Descent? | IBM*. (n.d.). IBM - Deutschland | IBM. https://www.ibm.com/topics/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates.
16. *Histogram-Based Gradient Boosting Ensembles in Python - MachineLearningMastery.com*. (n.d.). MachineLearningMastery.com.https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/

17. Luo,S., & Chen,T. (2020). Two Derivative Algorithms of Gradient Boosting Decision Tree for Silicon Content in Blast Furnace System Prediction. In *School of Statistics, Jiangxi University of Finance and Economics, Nanchang 330013, China*.https://www.researchgate.net/publication/346577317_Two_Derivative_Algorithms_of_Gradient_Boosting_Decision_Tree_for_Silicon_Content_in_Blast_Furnace_System_Prediction
18. LightGBM - HackMD. (n.d.). HackMD. https://hackmd.io/@WangJengYun/HyLtemyxI

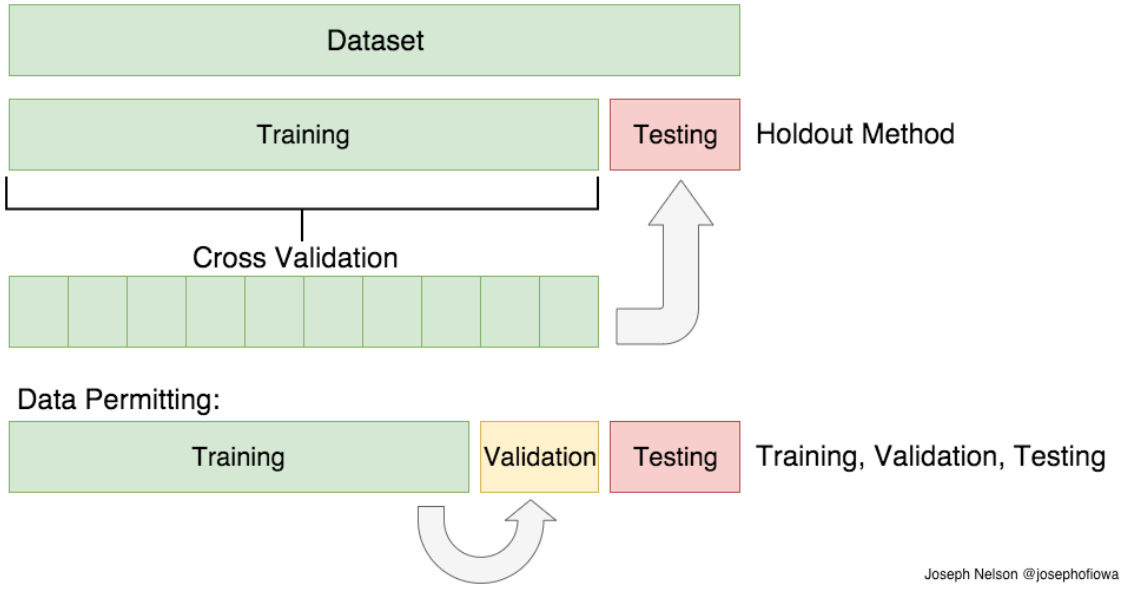
ДОДАТКИ

Додаток А

(Візуалізація алгоритму на основі гістограм)



Додаток Б



Додаток В

test_id	price
0	6.143482
1	6.819507
2	76.655691
3	12.996811
4	9.138563