КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії і технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Інформатика
на тему:

ДОСЛІДЖЕННЯ ВІДТОКУ КОРИСТУВАЧІВ ЗА ДОПОМОГОЮ МОДЕЛЕЙ КЛАСИФІКАЦІЇ ОБ'ЄКТІВ ПРИ НЕЗБАЛАНСОВАНИХ ДАНИХ

Виконала студентка 5-го курсу
заочної форми навчання
Клімчук Анастасія Романівна

_____
(підпис)

Науковий керівник: доцент к-ри ТТП
Панченко Тарас Володимирович

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень
з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на
засіданні кафедри теорії і технології
програмування
«____» _____ 20__ р.,
протокол No ___
Завідувач кафедри
Нікітченко М.С. _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 50 сторінок, 19 ілюстрацій, 1 таблиця, 10 джерел посилань.

КЛАСИФІКАЦІЯ ОБ’ЄКТІВ, ВІДТІК КОРИСТУВАЧІВ, МОДЕЛЬ МАШИННОГО НАВЧАННЯ, НЕЗБАЛАНСОВАНИЙ ДАТАСЕТ, GRADIENT BOOSTING, АНАСАМБЛЕВІ МЕТОДИ.

Метою роботи є зниження показника відтоку клієнтів. Для досягнення поставленої мети пропонується розробити програмний додаток, що використовує алгоритм машинного навчання, який буде виокремлювати сегмент користувачів схильних до відтоку, тобто схильних до припинення подальшої співпраці з бізнесом. Методи розроблення: комп’ютерне моделювання, моделювання та навчання моделі, тестування моделі на нових даних. Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки PyCharm CE, веб-інтерфейс користувача для jupyter проектів JupyterLab, мова програмування Python.

Функціональні вимоги до додатку:

- Добре пояснювальний алгоритм машинного навчання, що адаптований до роботи з дуже несиметричними вхідними даними, зважаючи на особливості заданої задачі;
- Відсоток правильно визначенних клієнтів у сегменті відтоку має перевищувати 90 відсотків;
- рішення повинно бути виконано за допомогою інструментів та технологій, які буде легко інтегрувати у систему маркетингу для подальшої праці з сегментом, дані про користувачів з сегменту відтоку повинні оновлюватись автоматично;

Результати роботи: була розроблена модель класифікації машинного навчання, що розділяє клієнтів на два сегменти: схильних до відтоку та лояльних.

Необхідний результат досягається завдяки аналізу предметної області та вивчення механізму відтоку користувачів. Також окрема увага була приділена вибору добре пояснювальної математичної моделі та вибору технології розробки.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

GB(Gradient Boosting) - градієнтний бустинг;

SGD (Stochastic Gradient Descent) – стохастичний градієнтний спуск;

BSD (Berkeley Software Distribution) – Дистрибутив програм Берклі;

CNN (Convolutional Neural Network) – згорткова нейронна мережа;

API (Application Programming Interface) – програмний інтерфейс;

ROC (Receiver Operating Characteristic) – графік, що дозволяє оцінити якість бінарної класифікації;

FPR (False Positive Rate) – хибно позитивна відповідь;

TPR (True Positive Rate) – дійсно позитивна відповідь;

AUC (Area Under The Curve) – область під кривою;

CE – Community dition.

ВСТУП

Відтік клієнтів – це відсоток клієнтів, які припинили користуватися продуктом або послугою компанії протягом певного періоду часу. Показник відтоку клієнтів або коефіцієнт відтоку клієнтів – це математичний розрахунок відсотка клієнтів, які малоймовірно будуть використовувати послуги компанії надалі.

Відтік клієнтів відбувається, коли клієнти вирішують не продовжувати купувати продукти/послуги в організації та припиняють свою асоціацію. Це невід'ємний параметр для організації, оскільки придбання нового клієнта може коштувати майже в 5 разів дорожче, ніж утримання існуючого. Відтік клієнтів може виявитися перешкодою для експоненціально зростаючої організації, і слід визначити стратегію утримання клієнтів, щоб уникнути збільшення показників відтоку. Клієнти, які не схильні до відтоку називаються лояльними.

Для визначення стратегії утримання клієнтів необхідно розуміти фактори та умови які впливають на вирішення припинення співпраці клієнта та компанії. Тому при вирішенні цієї задачі необхідно спиратися на алгоритми машинного навчання, які добре пояснюються та можуть відповісти на питання: чим сегменти відтоку та лояльності відрізняються між собою, тим самим надаючи гіпотези для утримання клієнтів схильних до відтоку.

Машинне навчання – це метод аналізу даних, який автоматизує побудову аналітичної моделі. Це розділ штучного інтелекту, заснований на ідеї, що системи можуть вчитися на даних, визначати закономірності та приймати рішення з мінімальним втручанням людини.

Метою даної роботи є виявлення сегменту користувачів схильних до відтоку із застосуванням алгоритмів машинного навчання. Застосований алгоритм має добре пояснюватись та відповідати на питання як утримати схильних до відтоку клієнтів зберігши при цьому витрати компанії.

Проблему виявлення клієнтів схильних до відтоку можна розглядати як

проблему класифікації об’єктів при незбалансованому датасеті. Оптимальна мережа для виконання класифікації на незбалансованому датасеті даних користувачів досліджується і реалізується за допомогою open-source бібліотеки машинного навчання, а саме Scikit-learn.

Об’єктом роботи є процес розв’язування класифікації користувачів на сегменти відтоку та лояльності.

Предметом роботи є алгоритм машинного навчання для розв’язування задачі виявлення сегменту користувачів схильних до відтоку.

РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ

1.1 Опис проблеми

Відтік клієнтів є справжньою проблемою для багатьох компаній, оскільки він показує, наскільки добре або погано вони вміють утримувати клієнтів на своєму боці. Повна вартість відтоку включає як втрачений дохід, так і витрати на маркетинг, пов'язані із заміною цих клієнтів новими. Здатність передбачити, що певний клієнт піддається високому ризику відтоку, хоча ще є час, щоб щось з цим зробити, є величезним додатковим потенційним джерелом доходу для кожного онлайн-бізнесу. Окрім прямих втрат доходу, які є результатом відмови клієнта від бізнесу, витрати на початкове придбання цього клієнта, можливо, ще не були покриті витратами клієнта на сьогоднішній день. Іншими словами, придбання цього клієнта насправді могло бути втратою інвестицією. Крім того, завжди важче і дорожче придбати нового клієнта, ніж утримати поточного клієнта, який платить. Зменшення відтоку є ключовою бізнес-ціллю кожного онлайн-бізнесу.

Для цього є дві причини:

1. По-перше, найбільше проблем викликає фінансовий аспект відтоку. За даними Forrester [1], придбання нових клієнтів коштує в 5 разів дорожче, ніж утримання наявних. Підведення нового клієнта до рівня існуючого буде коштувати в 16 разів дорожче.

2. Друга причина полягає в тому, що чим більше клієнтів утримує бізнес, тим більше він отримує дохід.

Наприклад, у звіті Гарвардської бізнес-школи стверджується[2], що в середньому збільшення рівня утримання клієнтів на 5% призводить до збільшення прибутку на 25-95%. І левова частка – 65% бізнесу компанії надходить від наявних клієнтів.

Ті самі істини розкриває KPMG[3], яка виявила, що утримання клієнтів є основним чинником прибутку компанії.

Розглянемо детальніше як саме компанії втрачають кошти через користувачів, які припиняють співпрацю.

Наприклад, якщо компанія залучає 10 клієнтів щороку, які купують товари та послуги на суму 3000 UAH, протягом 3 років із коефіцієнтом відтоку 0%, заробіток становитиме (3000 UAH * 10) + (3000 UAH * 20) + (3000 UAH * 30) = 180000 UAH.

Однак, коли з’являється коефіцієнт відтоку бізнесу, все стає складнішим. Тепер візьмемо середній показник відтоку бізнесу 30% і тепер компанія отримає (3000 UAH * 10) + (3000 UAH * (10*0.7 + 10)) + (3000 UAH * (17*0.7 + 10)) = 4810 UAH.

Іншими словами, відтік клієнтів обійшовся бізнесу в 1190 доларів, що за 3 роки коштує компанії приблизно 20% загального доходу. До цього до втрат компанії також слід занести витрати на залучення клієнтів, які пішли.

Відслідковування поведінки користувачів та схильність їх відтоку не тільки зберігає кошти компанії, це також допомагає зберігати контроль якості продукту або послуги.

Розглянемо індустрію потокового телебачення, де Netflix і Hulu є конкурентами. Дослідник ринку Park Associates повідомляє, що Netflix втратив близько 9% своєї бази передплатників за останні 12 місяців у порівнянні з приблизно 50% відтоку в Hulu [4]. Якби Hulu зміг провести кращій аналіз того, що відштовхує клієнтів, і знизити рівень відтоку на пару відсоткових пунктів, він став би більш конкурентоспроможним.

Отже, цілком зрозуміло, що зосередження на зменшенні відтоку клієнтів є першорядною задачею для кожного бізнесу, оскільки утримувати своїх клієнтів вигідно.

1.2 Різновиди причин виникнення проблеми

Основні причини відтоку клієнтів.

Існує безліч причин, які можуть призвести до відтоку клієнтів, і вони не завжди відомі. Давайте спробуємо зрозуміти деякі поширені причини відтоку клієнтів, з якими стикається кожна організація, і способи боротьби з ними.

Основні причини виникнення відтоку клієнтів показані на рис.1.1:

Рисунок 1.1 – Основні причини відтоку клієнтів

Розглянемо детальніше кожну з них:

1) Погана адаптація клієнта.

Однією з причин відтоку клієнтів є те, що клієнти не вважають продукт цінним. Якщо споживачі не вважають продукти, послуги чи програмне забезпечення цінними, це може бути пов’язано з тим, що вони не можуть знайти чи зрозуміти функції. Це відбувається в основному через погану адаптацію клієнта, коли компанія не проводить належним чином знайомство своїх нових

клієнтів з тим, як використовувати продукти та послуги.

Відмінний процес адаптації клієнтів є життєво важливим для забезпечення задоволеності клієнтів. Навпаки, погана адаптація може вплинути на зростання бізнесу негативним чином. Навчання споживачів збереже їх лояльність до бізнесу. Наприклад, Slack використовує підказку, щоб інформувати користувачів про їхні функції.

Згідно з дослідженням Harvard Business Review[2], зосередження на пропозиціях щодо адаптації може мати значний позитивний вплив на поновлення клієнтів.

2) Слабкі відносини з клієнтами.

Причина відтоку клієнтів, через те, що до думки клієнтів не дослухаються. Взаємодія з клієнтами та підтримка здорових відносин з ними має бути постійним процесом. Це не одноразова діяльність. Дотримування проактивного підходу та прислухання до голосу клієнтів, необхідно, щоб вони відчували, що їхню думку цінують.

3) Погане обслуговування клієнтів.

Майже 9 з 10 клієнтів покидають бізнес через поганий досвід.

Погана служба підтримки клієнтів може змусити клієнтів піти. Якщо компанія надає чудові продукти та послуги, але не зосереджується на підтримці клієнтів, у довгостроковій перспективі це буде коштувати їй більших грошей. Наприклад, якщо клієнту доведеться пройти довгий процес, щоб підключитися до представника клієнта, це може призвести до високого рівня відтоку.

4) Не зрозумілий інтерфейс

Багато компаній з часом розуміють, що споживачі завантажують програму, встановлюють її, але через деякий час припиняють їх використовувати. Яка причина цього? Це може бути тому, що вони не знають, як користуватися продуктом, або функції продукту занадто складні для розуміння. Не зрозумілий інтерфейс - одна з найголовніших причин відтоку нових користувачів.

5) Ігнорування скарг клієнтів.

Не реагування та неналежне розглядання скарг призводить до повторних

скаржень і змушує користувачів відчувати себе більш розчарованими і в кінцевому підсумку призводить до відтоку.

Скарги відіграють життєво важливу роль для виявлення проблем у продуктах і послугах.

Відтік клієнтів – це важливий показник, який безпосередньо пов’язаний з доходом і надає статистичні дані про лояльних клієнтів; тому кожна компанія повинна розуміти проблеми, які призводять до відтоку, і виправляти їх. Основна мета моделі прогнозування відтоку клієнтів полягає в тому, щоб утримати клієнтів із найвищим ризиком відтоку шляхом активної взаємодії з ними. Для того, щоб виявляти користувачів схильних до відтоку застосовують методи машинного навчання. Машинне навчання є ідеальними рішеннями для утримання клієнтів і запобігання відтоку, адже інформація та аналіз на основі даних клієнтів можуть надходити в бізнес-системи та інформаційні панелі компанії автоматично, що дозволить їм бути активними та гарантувати, що їх клієнти продовжать використовувати продукти/послуги.

1.3 Існуючі рішення проблеми

У наш час прийнято використовувати передові методи машинного навчання, щоб якомога точніше прогнозувати ймовірність відтоку клієнтів. Однак гарне рішення для запобігання відтоку вимагає не тільки точності. Зіткнувшись із проблемою відтоку, передбачення відтоку без пояснення не може забезпечити велику цінність для бізнесу. Важливо розуміти, звідки береться відтік і які дії слід вжити, щоб його запобігти.

Прогноз відтоку зазвичай розглядається як проблема класифікації, класифікуючи клієнтів на два кластери: схильні до відтоку та лояльні.

Розглянемо існуючі рішення для вирішення задач класифікації:

1) Логістична регресія – є легкою відправною точкою. Цей алгоритм легко пояснити та реалізувати, тому його легше продати бізнесу. За машинним

навчанням багатьох великих компаній стоїть проста логістична регресія. Коли є дуже багато даних, логістична регресія може дуже швидко навчити модель. Але для компаній, в яких об’єм даних вимірюється менше ніж сотнями тисяч строк, для логістичної регресії може бути недостатньо даних для отримання хорошого результату.

2) Наївний Байєс – зазвичай використовується з класифікацією тексту, і він добре працює при розв’язанні для кількох класів. Замість того, щоб просто давати так/ні, він може класифікувати клієнтів як низький/середній/високий ризик. Але має таку ж саму проблему як і логістична регресія - для гарного результату, треба дуже багато даних, яких в компанії може не бути.

3) Випадковий ліс - може дати хороші результати з меншою кількістю даних, тому це одна з найкращих моделей класифікації для передбачення відтоку. І це може бути ідеальним варіантом, якщо використовуються необроблені дані, але критична слабкість цієї моделі полягає в тому, що вона не обробляє динамічні дані. Тобто можуть виникнути проблеми з такими даними, як «використання клієнтами за останні 30 днів», які постійно змінюються.

4) Дерева рішень, підсилені градієнтом (Gradient-boosted decision trees) - дерева рішень підсені градієнтом є популярним методом для розв’язування задач прогнозування як в області класифікації, так і в області регресії. Цей підхід покращує процес навчання за рахунок спрощення мети та зменшення кількості ітерацій для досягнення достатньо оптимального рішення. Цей тип моделей неодноразово зарекомендував себе на різноманітних змаганнях, які оцінювали як точність, так і ефективність. Також їх можна використовувати при невеликих об’ємах даних і при несбалансованому датасеті.

На жаль, більшість методів моделювання прогнозу відтоку покладаються на кількісне визначення ризику на основі статичних даних і метрик, тобто інформації про клієнта, яким він чи вона існує зараз. Найпоширеніші моделі прогнозування відтоку базуються на старих методах статистики та аналізу даних, таких як логістична регресія та інші методи бінарного моделювання. Ці підходи пропонують певну цінність і можуть визначити певний відсоток клієнтів із

ризиком відтоку, але вони є відносно неточними і погано працюють з невеликою та асиметричною вибіркою даних.

Найкращою моделлю в цьому випадку є дерева рішень підсилені градієнтом. Також, основною задачею є не лише виявлення максимально точно користувачів схильних до відтоку, а й створити добре пояснювальний алгоритм, який надасть гіпотези щодо дій необхідних для зменшення відтоку.

РОЗДІЛ 2 АНАЛІЗ ПРОБЛЕМИ

2.1 Проектування робочого процесу прогнозування відтоку

Процес розробки системи, що базується на машинному навчанні для прогнозування відтоку користувачів є стандартним як і для більшості проектів, де використовується машинне навчання і включає наступні кроки:

1) Визначення проблеми та мети: важливо зрозуміти, які висновки необхідно отримати від аналізу та прогнозу.

2) Встановлення джерел даних: необхідно з’ясувати доступність усіх джерел даних, які будуть необхідні, на етапі моделювання.

3) Підготовка, дослідження та попередня обробка даних: необроблені історичні дані для вирішення проблеми та побудови прогнозних моделей потрібно трансформувати у формат, придатний для алгоритмів машинного навчання. Цей крок також може покращити загальні результати за рахунок підвищення якості даних.

4) Моделювання та тестування. Це охоплює розробку та перевірку продуктивності моделей прогнозування відтоку клієнтів із різними алгоритмами машинного навчання та різними параметрами.

5) Розгортання та моніторинг: це останній етап застосування машинного навчання для прогнозування відтоку користувачів. Модель можна або інтегрувати в існуюче програмне забезпечення, або створити нову оновлювальну програму.

Загальна схема процесу розробки моделі для прогнозування відтоку показана на рисунку 2.1:

Рисунок 2.1 – Схема робочого процесу при побудові моделі

Модель навчається на основі історії відтоку клієнтів. Історія відтоку клієнтів – це період події для функцій X і вікно діяльності для цільової змінної, зображено на рисунку 2.2.

Рисунок 2.2 – Схема збору даних для історії відтоку клієнтів

Щомісяця активна клієнтська база передається в прогнозну модель

машинного навчання, щоб повернути ймовірність відтоку для кожного клієнта.

Список клієнтів необхідно буде згрупувати за ймовірністю відтоку в групи ризику. Користувачів з груп ризику з середньою ймовірністю відтоку більше ніж за 80 % необхідно передати у форматі csv у відділи утримання клієнтів для подальшої взаємодії.

Клієнти, у яких є дуже низька ймовірність відтоку - лояльні клієнти. Ніяких заходів щодо них не вживають.

Проблеми побудови ефективної моделі відтоку

Ось основні проблеми, які можуть ускладнити створення ефективної моделі відтоку:

1) Неточні або шумні дані клієнтів;
2) Слабкий пояснювальний аналіз відтоку;
3) Брак інформації та знань предметної області;
4) Відсутність послідовного вибору відповідного підходу до моделювання відтоку;
5) Хибний вибір показників для перевірки ефективності моделі відтоку;
6) Швидка зміна концепції, заснована на змінах моделей поведінки клієнтів, що сприяють відтоку;
7) Дисбалансовані дані.

2.2 Розгляд моделі та механізму реалізації

Спираючись на висновки розділу 1.3 було обрано для реалізації моделі пошуку сегменту користувачів схильних до відтоку моделі дерева рішень підсиленні градієнтом. В цьому розділі розглянемо більш детально історію виникнення ансамблевого методу підсилення градієнтом, суть і принцип його роботи та наявні реалізації.

Підсилення градієнтом або gradient boosting є одним з найпотужніших методів для побудови прогнозних моделей.

2.2.1 Історія походження методу підсилення градієнтом

Ідея підсилення, далі бустінг, виникла з ідеї про те, чи можна модифікувати «слабкого лернера», тобто модель або гіпотезу ставати краще.

Майкл Кернс у його праці Думки про підвищення гіпотез (1988) сформулював ціль як «Проблему Бустінгу Гіпотез» або «Hypothesis Boosting Problem», сформулювавши мету з практичної точки зору так:

«… ефективний алгоритм для перетворення відносно поганих гіпотез у дуже хороші гіпотези» [5].

Слабка гіпотеза або слабкий лернер визначається як той, чий перформанс принаймні трохи кращий, ніж отримання результату випадковим чином.

Ці ідеї ґрунтувалися на роботі Леслі Валіанта щодо «Probably Approximately Correct (PAC) learning», тобто «ймовірно приблизно правильного навчання», що стало основою для дослідження складності проблем машинного навчання:

«Ідея полягає в тому, щоб використовувати метод слабкого навчання кілька разів, щоб отримати послідовність гіпотез, кожна з яких зосереджена на прикладах, які для попередніх гіпотез виявились складними та були неправильно класифіковані. … Зауважте, однак, зовсім не очевидно, як це можна зробити» [6].

2.2.2 AdaBoost – перший алгоритм бустинга

Першою реалізацією бустинга, який мав великий успіх у застосуванні, був Adaptive Boosting або скорочено AdaBoost. Бустінг відноситься до загальної проблеми створення дуже точного правила прогнозування шляхом поєднання грубих і помірно неточних правил [7].

Слабкі лернери в AdaBoost – це дерева рішень з єдиним розділенням, які через їх тривіальність називають пнями рішень. AdaBoost працює шляхом зважування спостережень, надаючи більшої ваги прикладам, які важко класифікувати, і менше тим, які вже добре оброблені. Послідовно додаються нові

слабкі лернери, або під-моделі, які зосереджують своє навчання на складніших моделях. Це означає, що вибірки, які важко класифікувати, отримують все більші ваги, поки алгоритм не визначить модель, яка правильно класифікує ці вибірки [8].

Прогнози робляться більшістю голосів прогнозів слабких учнів, зважених відповідно до їхньої індивідуальної точності (accuracy). Найуспішніша форма алгоритму AdaBoost була для задач, бінарної класифікації і називалася AdaBoost.M1.

2.2.3 Узагальнення AdaBoost до Градієнтного бустинга

AdaBoost і пов’язані з ним алгоритми були перероблені в статистичний фреймворк, спочатку Брейманом, назвавши їх алгоритмами ARCing.

Arcing – це абревіатура від Adaptive Reweighting and Combining. Кожен крок в алгоритмі arcing складається із зваженої мінімізації з подальшим переобчисленням [класифікаторів] і [зважених вхідних даних] [9].

Цей фреймворк надалі розвинув Фрідмен і назвав його «Gradient Boosting Machines». Пізніше отримав назву «Градієнтний Бустинг» або «Дерева Градієнтного бустинга».

Статистичний фреймворк обробляє бустинг як числову проблему оптимізації, мета якої мінімізувати функцію втрати моделі додаванням слабких під-моделей використовуючи градієнтний спуск.

Узагальнення дозволило використовувати довільні диференційовані функції втрат, розширюючи техніку за межі проблем бінарної класифікації для підтримки регресії, класифікації за багатьма класами тощо.

2.2.4 Принцип роботи методу градієнтного бустинга

Градієнтний бустінг включає в себе три основні елементи:

1) Функцію втрат для оптимізації.

2) Слабкі під-моделі для прогнозування.

3) Адитивна модель для додавання слабких під-моделей для мінімізації функції втрат.

Функція втрат залежить від типу розв'язуваної задачі. Вона має бути диференційованою. Наприклад, для задачі регресії може бути використаний метод найменших квадратів як функція втрат, а для задачі класифікація може використовуватись логарифмічна функція як функція втрати.

Перевага методу градієнтного бустинга полягає в тому, що новий алгоритм бустинга не потрібно виводити для кожної функції втрат, яку ми хочемо використати, натомість це досить загальна структура, щоб можна було використовувати будь-яку диференційовану функцію втрат.

Слабкі під-моделі. Дерева рішень використовуються як слабкі під-моделі, тобто слабкі одиниці навчання у складі загальної моделі, у градієнтному бустингу. Зокрема, використовуються дерева регресії, які виводять реальні значення для розщеплень і ці вихідні дані можна просумувати, дозволяючи додавати вихідні дані наступних моделей і «коригувати» залишки в передбаченнях. Дерева будуються на основі жадібного алгоритму, наприклад Джіні, вибираючи найкращі точки розділення для мінімізування функції втрат. Спочатку, як наприклад, у випадку AdaBoost, використовувалися дуже короткі дерева рішень, які мали лише одне розділення, яке називалося пеньком рішень. Більші дерева зазвичай можна використовувати з рівнями 4-8. Загальною практикою є обмеження слабких під-моделей певними способами, такими як максимальна кількість шарів, вузлів, розщеплень або листових вузлів. Це робиться для того, щоб під-моделі залишалися слабкими, але все ще могли бути сконструйовані в жадібний спосіб.

Адитивна модель. Дерева додаються в модель по одному, а вже додані дерева в моделі не змінюються. Для мінімізації втрат при додаванні дерев

використовується процедура градієнтного спуску. Традиційно градієнтний спуск використовується для мінімізації набору параметрів, таких як коефіцієнти в рівнянні регресії або ваги в нейронній мережі. Після обчислення помилки або втрат ваги оновлюються, щоб мінімізувати цю помилку. Замість параметрів ми маємо слабкі під-моделі або, точніше, дерева рішень. Після розрахунку втрат, щоб виконати процедуру градієнтного спуску, ми повинні додати до моделі дерево, яке зменшує втрати (тобто слідувати за градієнтом). Ми робимо це, параметризуючи дерево, потім змінюємо параметри дерева і рухаємося вправо, зменшуючи залишкові втрати. Зазвичай цей підхід називається функціональним градієнтним спуском або градієнтним спуском з функціями.

«Одним із способів створення зваженої комбінації класифікаторів, яка оптимізує [вартість], є градієнтний спуск у функціональному просторі» [10].

Вихідні дані для нового дерева потім додаються до вихідних даних існуючої послідовності дерев, щоб виправити або покращити кінцевий результат моделі. Додається фіксована кількість дерев або навчання припиняється, коли втрата досягає прийнятного рівня або більше не покращується набір даних зовнішньої перевірки.

Сам алгоритм має дуже наочну візуальну інтерпретацію інтуіції зважування спостережень, що стоїть за ним. Розглянемо іграшковий приклад задачі класифікації, у якій спробуємо на кожній ітерації алгоритму розділити дані деревом глибини 1 (так званим «пнем»). На перших двох ітераціях ми побачимо таку картинку (рис. 2.3):

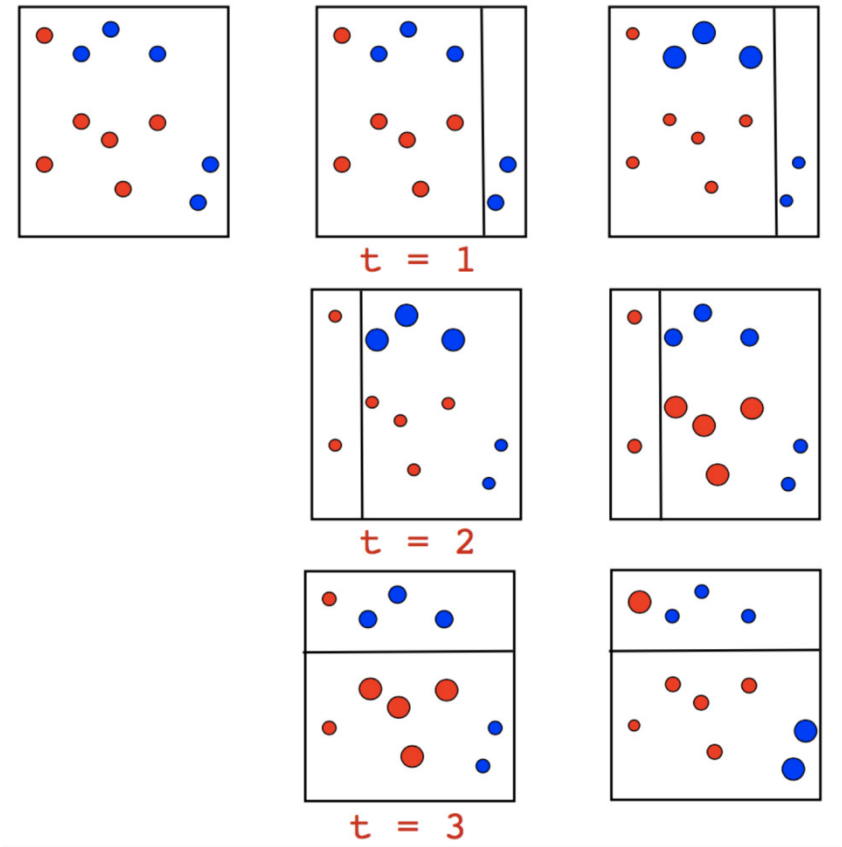

Рисунок 2.3 – Розділення даних деревом рішень глибиною 1

Розмір точки відповідає отриманій їй вазі за помилкове передбачення. І ми бачимо, як на кожній ітерації ці ваги зростають – пні не можуть поодинці впоратися з таким завданням. Однак, коли ми зробимо зважене голосування раніше побудованих пнів, ми отримаємо поділ, який ми шукаємо (рис. 2.4):

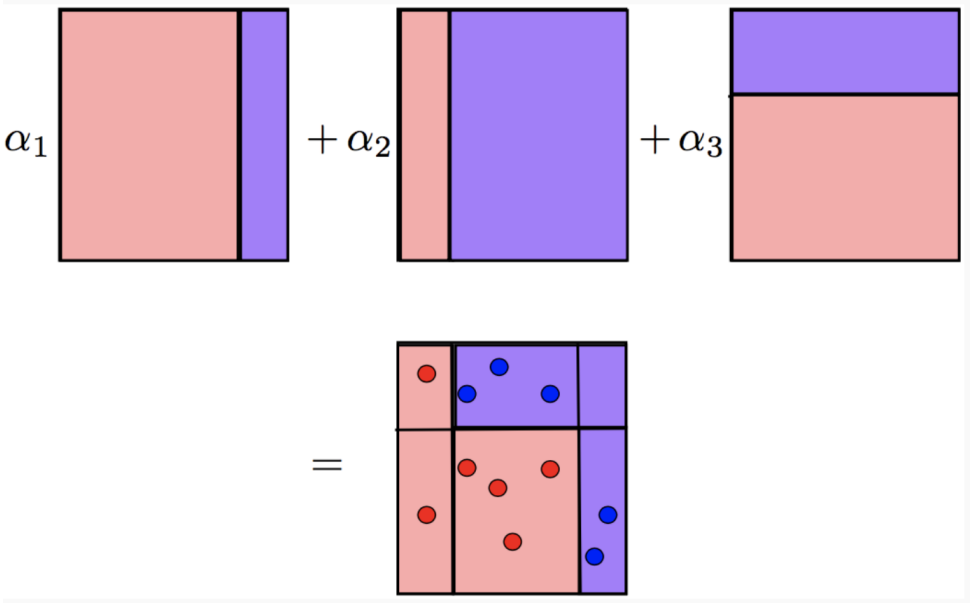

Рисунок 2.4 – Ансамбль зі зважених рішень під-моделей

2.3 Постановка ML задачі

Ми будемо вирішувати задачу відновлення функції у загальному контексті навчання з учителем. У нас буде набір пар ознак x і цільових змінних y, $\{(x_i, y_i)\}_{i=1,\dots n}$ на якому ми будемо відновлювати залежність виду $y = f(x)$. Відновлювати будемо наближенням $\hat{f}(x)$, а для розуміння, яке наближення краще, у нас також буде функція втрат $L(y,\ f)$, яку ми будемо мінімізувати:

$$y \approx \hat{f}(x),$$
$$\hat{f}(x) = arg\ min\ L(y,\ f(x))_{f(x)} \tag{2.1}$$

Поки що ми не робимо жодних припущень ні про тип залежності $f(x)$, ні про модель наших наближень $\hat{f}(x)$, ні про розподіл цільової змінної y. Окрім того, як було зазначено вище, що функція втрат має бути диференційованою. Так як набір даних обмежений, перепишемо все в термінах маточікування. А саме, будемо шукати наші наближення $\hat{f}(x)$ так, щоб у середньому мінімізувати функцію втрат на тих даних, що є:

$$\hat{f}(x) = arg\ min\ E_{x,y}[L(y,\ f(x))]_{f(x)} \tag{2.2}$$

В машинному навчанні зазвичай обмежують простір пошуку якимось конкретним параметризованим сімейством функцій $f(x, \theta),\ \theta \in R^d$.

Це дуже спрощує задачу, тому що вона зводиться до вже цілком вирішуваної оптимізації значень параметрів:

$$\hat{f}(x) = f(x, \hat{\theta}),$$

$$\hat{\theta} = arg\,min\ E_{x,y}[L(y,\ f(x,\ \theta))]_{\theta} \tag{2.3}$$

Аналітичні рішення для отримання оптимальних параметрів $\hat{\theta}$ в один рядок існують досить рідко, тому параметри наближають ітеративно.

Спочатку нам треба виписати емпіричну функцію втрат $L_{\theta}(\hat{\theta})$, що показує, наскільки добре ми їх оцінили за наявними у нас даними. Також випишемо наше наближення $\hat{\theta}$ за M ітерацій у вигляді суми:

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

$$L_{\theta}(\hat{\theta}) = \sum_{i=1}^{N} L(y_{i},\ f(x_{i},\ \hat{\theta})), \tag{2.4}$$

Залишилося тільки взяти відповідний ітеративний алгоритм, яким ми мінімізуватимемо $L_{\theta}(\hat{\theta})$. Найпростіший варіант, що часто використовується, – градієнтний спуск. Для нього потрібно виписати градієнт $\nabla L_{\theta}(\hat{\theta})$ та додавати наші ітеративні оцінки $\hat{\theta}_{i}$ вздовж нього зі знаком мінус. Залишилось ініціалізувати перше наближення $\hat{\theta}_{0}$ і обрати, скільки ітерацій M ми будемо продовжувати цю процедуру. Цей алгоритм виглядатиме так:

1) Ініціалізувати початкове наближення параметрів $\hat{\theta} = \hat{\theta}_{0}$.

2) Для кожної ітерації $t = 1, \ldots, M$ повторювати:

а) Порахувати градієнт функції втрат $\nabla L_{\theta}(\hat{\theta})$ при поточному наближенні $\hat{\theta}$

$$\nabla L_{\theta}(\hat{\theta}) = \left[\frac{\partial L(f(x,\theta))}{\partial \theta}\right]_{\theta=\hat{\theta}} \tag{2.5}$$

b) Задати поточне ітеративне наближення $\hat{\theta}_t$ на основі порахованого градієнта

$$\hat{\theta}_t \leftarrow - \nabla L_\theta(\hat{\theta}) \tag{2.6}$$

c) Оновити наближення параметрів $\hat{\theta}$:

$$\hat{\theta} \leftarrow \hat{\theta} + \hat{\theta}_t = \sum_{i=0}^{t} \hat{\theta}_i \tag{2.7}$$

3) Зберегти підсумкове наближення $\hat{\theta}$

$$\hat{\theta} = \sum_{i=0}^{M} \hat{\theta}_i, \tag{2.8}$$

4) Використати знайдену функцію $\hat{f}(x) = f(x, \hat{\theta})$ за призначенням.

Функціональний градієнтний спуск.

Припустимо, що ми можемо проводити оптимізацію в функціональному просторі та ітеративно шукати наближення $\hat{f}(x)$ у вигляді функцій. Випишемо наше наближення у вигляді суми інкрементальних покращень, кожне з яких є функцією. Для зручності одразу будемо рахувати цю суму, починаючи з початкового наближення $\hat{f}_0(x)$:

$$\hat{f}_0(x) = \sum_{i=0}^{M} \hat{f}_i(x) \tag{2.9}$$

Щоб вирішити задачу, нам все одно доведеться обмежити свій пошук якимсь сімейством функцій $\hat{f}(x) = h(x, \theta)$. Відразу врахуємо, що на кожному кроці для функцій нам буде необхідно підбирати оптимальний коефіцієнт $\rho \in R$. Для кроку t задача виглядає так:

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

$$(\rho_t, \theta_t) = arg_{\rho,\theta} min\, E_{x,y}[L(y, \hat{f}(x) + \rho \cdot h(x, \theta))],$$

$$\hat{f}_t(x) = \rho_t \cdot h(x, \theta_t) \qquad (2.10)$$

Ми виписали задачу в загальному вигляді для того щоб навчати які завгодно моделі $h(x, \theta)$ відносно яких завгодно функцій втрат $L(y, f(x, \theta))$. На практиці це дуже складно, тому був придуманий простий спосіб звести завдання до чогось більш практично вирішуваного.

Знаючи вираз градієнта функції втрат, ми можемо порахувати його значення на наших даних. Тож будемо навчати моделі так, щоб наші передбачення були найбільш скорельованими з цим градієнтом (зі знаком мінус). Тобто будемо вирішувати завдання МНК-регресії (методу найменших квадратів), намагаючись виправляти передбачення щодо цих залишків. І для класифікації, і для регресії, і для ранжування ми весь час будемо мінімізуватимемо квадрат різниці між псевдо-залишками r і нашими передбаченнями. Для кроку t задача в підсумку буде виглядати так:

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

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

$$\theta_t = arg_\theta min \sum_{i=1}^{n} (r_{it} - h(x_i, \theta))^2,$$

$$\rho_t = arg_\rho min \sum_{i=1}^{n} L(y_i, \hat{f}(x_i) + \rho \cdot h(x_i, \theta_t)) \quad (2.11)$$

Тепер у нас є все необхідне, щоб виписати алгоритм GBM, запропонований Джером Фрідманом. Ми так само вирішуємо загальне завдання навчання з учителем. На вхід алгоритму потрібно зібрати кілька складових:

1) набір даних $\{(x_i, y_i)\}_{i=1,\dots,n}$;

2) число ітерацій M;

3) вибір функції втрат $L(y, f)$ з виписаним градієнтом;

4) вибір сімейства функцій базових алгоритмів $h(x, \theta)$, з процедурою їх навчання;

5) додаткові гіперпараметри $h(x, \theta)$, наприклад, глибина дерева у деревах рішень.

Залишилося задати початкове приближення $f_0(x)$. Для простоти, в якості ініціалізації використовують константне значення γ. Його, а також оптимальний коефіцієнт ρ знаходять бінарним пошуком, або іншим лінійним алгоритмом пошуку щодо вихідної функції втрат.

Отже, алгоритм GBM:

1) Ініціалізувати GBM постійним значенням:

$$\hat{f}(x) = \hat{f}_0, \hat{f}_0 = \gamma, \gamma \in R$$
$$\hat{f}_0 = arg_\gamma min \sum_{i=1}^{n} L(y_i, \gamma) \quad (2.12)$$

2) Для кожної ітерації $t = 1, \dots, M$ повторити:

а) Порахувати псевдо-залишки r_t

$$r_{it} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}(x)}, \; for\; i = 1, \dots, n \quad (2.13)$$

b) Побудувати новий базовий алгоритм $h_t(x)$ як регресію на псевдо-залишках $\{(x_i, r_{it})\}_{i=1,\dots,n}$

c) Знайти оптимальний коефіцієнт ρ_t при $h_t(x)$ відносно вихідної функції втрат

$$\rho_t = arg_\rho min \sum_{i=1}^{n} L(y_i, \hat{f}(x_i) + \rho \cdot h(x_i, \theta)) \quad (2.14)$$

d) Зберегти $\hat{f}_t(x) = \rho_t \cdot h_t(x)$

e) Оновити поточне наближення $\hat{f}(x)$

$$\hat{f}(x) \leftarrow \hat{f}(x) + \hat{f}_t(x) = \sum_{i=0}^{t} \hat{f}_i(x) \quad (2.15)$$

3) Скомпонувати підсумкову модель GBM $\hat{f}(x)$

$$\hat{f}(x) = \sum_{i=0}^{M} \hat{f}_i(x) \quad (2.16)$$

Розглянемо більш детально функції втрат для вирішення нашої задачі бінарної класифікації для виявлення сегменту відтоку користувачів.

Функція втрат у дереві посилення градієнта для бінарної класифікації.

Для бінарної класифікації загальний підхід полягає у тому, щоб побудувати деяку модель $\hat{y} = f(x)$ і прийняти її логіт як передбачення ймовірності:

$$\hat{p} = \frac{1}{1+e^{-\hat{y}}} \quad (2.17)$$

Тут x представляє собою предіктор(и), вихід y – це двійкове значення (або 0, або 1), $\hat{y}$ приймає будь-яке реальне значення на $(-\infty, +\infty)$, та $\hat{p}$ є реальним значенням від 0 до 1, інтерпретується як ймовірність y бути 1. Важливо зазначити що $\hat{y}$ не намагається тут передбачити бінарний вихід y, тому що ці значення не порівнянні ні в значенні, ні в величинах. Передбачення - це $\hat{p}$, ймовірність $y = 1$ (а не саме значення у), тоді як $\hat{y}$ є лише деяким проміжним способом моделювання.

Це легко вивести:

$$\hat{y} = log\frac{\hat{p}}{1-\hat{p}} \tag{2.18}$$

Функція (1), яка перетворює $\hat{y}$ на $\hat{p}$, називається логістичною; його обернена, тобто (2), яка перетворює $\hat{p}$ в $\hat{y}$, називається логітом.

Найвідомішим прикладом є логістична регресія, в якій $\hat{y} = f(x)$ – є лінійною моделлю x. Загалом, f є адитивною моделлю. Під цим ми маємо на увазі, що fможна побудувати підсумовуючи декілька компонентів, наприклад ряд базових функцій або регресійних дерев. Оскільки діапазон f дорівнює $(-\infty, \infty)$, немає жорстких обмежень для компонентів або підсумовування. Що призводить до можливості застосування дерев для посилення градієнта для класифікації. У таких алгоритмах базовими навчальними моделямі - під-моделями, є регресійні дерева, підсумовування яких є $\hat{y}$. Вони не можуть бути деревами класифікації, кожне з яких забезпечує передбачення ймовірності $\hat{p}$, оскільки $\hat{p}$ не можна скласти, щоб отримати корисну величину. Ми виконуємо логіт-перетворення лише тоді, коли збираємося надати ймовірний прогноз на основі $\hat{y}$, але це перетворення насправді не є частиною моделі бустингу дерев.

Давайте візьмемо модель після m-ї ітерації, тобто F_m, і запишемо:

$$\hat{y}_i = F_m(x_i),\ i = 1, \ldots, n \tag{2.19}$$

Як зазначено вище, передбачення $\hat{y}_i$ не є передбаченням бінарного виходу y_i, і це не є проблемою; це всього лише проміжний пристрій для моделювання. Нам потрібно лише уточнити функцію втрат і розробити її градієнт по відношенню до $\hat{y}_i$.

Втрата перехресної ентропії є стандартним вибором:

$$L_i = -y_i \cdot log\,\hat{p}_i - (1 - y_i) \cdot log(1 - \hat{p}_i) \tag{2.20}$$

Ми все ще можемо використовувати величину $\hat{p}$ як перетворення $\hat{y}$, але нам потрібно думати про втрату переважно як функцію $\hat{y}$, а не $\hat{p}$. Давайте спробуємо реформувати втрату таким чином, щоб допомогти виразити її в термінах $\hat{y}$:

$$L_i = -y_i \cdot log(\hat{p}_i(1 - \hat{p}_i)) - log(1 - \hat{p}_i) \tag{2.21}$$

Використовуючи формули 1 та 2, отримуємо:

$$L_i = log(1 + e^{\hat{y}_i}) - y_i \hat{y}_i \tag{2.22}$$

Функція втрат тепер виражається бінарною відповіддю і логарифмічними шансами.

Візьмемо похідну:

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{\partial \sum_j L_j}{\partial \hat{y}_i} = \frac{\partial L_i}{\partial \hat{y}_i} = \frac{e^{\hat{y}_i}}{1 + e^{\hat{y}_i}} - y_i = \hat{p}_i - y_i \tag{2.23}$$

Тепер візьмемо похідну від (3),

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{\partial(-y_i \cdot log\, \hat{p}_i - (1-y_i) \cdot log(1-\hat{p}_i))}{\partial \hat{y}_i} = \hat{p}_i - y_i \tag{2.24}$$

Давайте зробимо ще один крок і з'ясуємо другу похідну:

$$\frac{\partial^2 L}{\partial \hat{y}_i^2} = \frac{\partial^2 L_i}{\partial \hat{y}_i^2} = \frac{\partial(\hat{p}_i - y_i)}{\partial \hat{y}_i} = \frac{\partial \hat{p}_i}{\partial \hat{y}_i} = \hat{p}_i(1 - \hat{p}_i) \tag{2.25}$$

Класифікація градієнтним бустінгом в бібліотекі scikit-learn

Давайте зрозуміємо, як працює класифікація градієнтним бустінгом в scikit-learn, бібліотекі для python. Є багато деталей; Я зосередилася на високому рівні навколо функції втрат та ітерації моделі.

Модульний документ має чудовий опис:

```
"""Gradient Boosted Regression Trees

This module contains methods for fitting gradient boosted regression trees for
both classification and regression.

The module structure is the following:

- The ``BaseGradientBoosting`` base class implements a common ``fit`` method
  for all the estimators in the module. Regression and classification
  only differ in the concrete ``LossFunction`` used.

- ``GradientBoostingClassifier`` implements gradient boosting for
  classification problems.

- ``GradientBoostingRegressor`` implements gradient boosting for
  regression problems.
"""
```

Я збираюся детально дослідити клас GradientBoostingClassifier і зрозуміти його ініціалізатор (__init__), а також основні методи навчання та прогнозування. Хоча код може обробляти багатокласову класифікацію, я буду відстежувати його поведінку лише для бінарної класифікації, для вирішення моєї задачі.

Якщо не зазначено інше, усі класи, досліджені нижче, знаходяться в модулі sklearn.ensemble.gradient_boosting.

__init__

Документ дає гарне резюме:

```
class GradientBoostingClassifier(BaseGradientBoosting, ClassifierMixin):
    """Gradient Boosting for classification.

    GB builds an additive model in a
    forward stage-wise fashion; it allows for the optimization of
    arbitrary differentiable loss functions. In each stage ``n_classes_``
    regression trees are fit on the negative gradient of the
    binomial or multinomial deviance loss function. Binary classification
    is a special case where only a single regression tree is induced.
    """
```

Тут особливо цікаві наступні параметри:

```
"""
Parameters
----------
loss : {'deviance', 'exponential'}, optional (default='deviance')
    loss function to be optimized. 'deviance' refers to
    deviance (= logistic regression) for classification
    with probabilistic outputs. For loss 'exponential' gradient
    boosting recovers the AdaBoost algorithm.

learning_rate : float, optional (default=0.1)
    learning rate shrinks the contribution of each tree by `learning_rate`.
    There is a trade-off between learning_rate and n_estimators.

n_estimators : int (default=100)
    The number of boosting stages to perform. Gradient boosting
    is fairly robust to over-fitting so a large number usually
    results in better performance.

subsample : float, optional (default=1.0)
    The fraction of samples to be used for fitting the individual base
    learners. If smaller than 1.0 this results in Stochastic Gradient
    Boosting. `subsample` interacts with the parameter `n_estimators`.
    Choosing `subsample < 1.0` leads to a reduction of variance
    and an increase in bias.
```

```
criterion : string, optional (default="friedman_mse")
    The function to measure the quality of a split. Supported criteria
    are "friedman_mse" for the mean squared error with improvement
    score by Friedman, "mse" for mean squared error, and "mae" for
    the mean absolute error. The default value of "friedman_mse" is
    generally the best as it can provide a better approximation in
    some cases.

max_depth : integer, optional (default=3)
    maximum depth of the individual regression estimators. The maximum
    depth limits the number of nodes in the tree. Tune this parameter
    for best performance; the best value depends on the interaction
    of the input variables.
```

```
max_features : int, float, string or None, optional (default=None)
    The number of features to consider when looking for the best split:

    - If int, then consider `max_features` features at each split.
    - If float, then `max_features` is a fraction and
      `int(max_features * n_features)` features are considered at each
      split.
    - If "auto", then `max_features=sqrt(n_features)`.
    - If "sqrt", then `max_features=sqrt(n_features)`.
    - If "log2", then `max_features=log2(n_features)`.
    - If None, then `max_features=n_features`.

    Choosing `max_features < n_features` leads to a reduction of variance
    and an increase in bias.
"""
```

Функція втрат за замовчуванням – це «deviance». Для бінарної класифікації це BinomialDeviance (встановлено в BaseGradientBoosting._check_params).

```
class BinomialDeviance(ClassificationLossFunction):

    def __call__(self, y, pred, sample_weight=None):
        """Compute the deviance (= 2 * negative log-likelihood).

        Parameters
        ----------
        y : array, shape (n_samples,)
            True labels

        pred : array, shape (n_samples,)
            Predicted labels

        sample_weight : array-like, shape (n_samples,), optional
            Sample weights.
        """
        # logaddexp(0, v) == log(1.0 + exp(v))
        pred = pred.ravel()
        if sample_weight is None:
            return -2.0 * np.mean((y * pred) - np.logaddexp(0.0, pred))
        else:
            return (-2.0 / sample_weight.sum() *
                    np.sum(sample_weight * ((y * pred) - np.logaddexp(0.0, pred))))
```

```
def negative_gradient(self, y, pred, **kargs):
    """Compute the residual (= negative gradient).

    Parameters
    ----------
    y : array, shape (n_samples,)
        True labels

    pred : array, shape (n_samples,)
        Predicted labels
    """
    return y - expit(pred.ravel())
```

Тут pred – це $\hat{y}$ у попередньому розділі. Функція втрат, обчислена за допомогою __call__, узгоджується з формулою (5). Градієнт, обчислений за допомогою negative_gradient, узгоджується з формулою (6). (expit – це логістична функція, імпортована з scipy.special.)

Параметр subsample вмикає Bagging-like поведінку, але вимкнено за замовчуванням.

Параметр max_features вмикає Random-Forest поведінку, але вимкнено за замовчуванням.

fit

GradientBoostingClassifier.fit безпосередньо використовує батьківський BaseGradientBoosting.fit.

Пропускаючи всі перевірки працездатності та більшість сценаріїв нестандартних або не бінарних класів, найважливішими кроками є такі:

1) Задати self.classes_ = [0, 1] і self.n_classes_ = 2.

2) Задати self.loss_ = BinomialDeviance(2). Після цього параметра self.loss_.K дорівнює 1, кількість дерев регресії, які потрібно викликати.

3) Задати self.init_ = self.loss_.init_estimator(), що ефективно виконує self.init_ = LogOddsEstimator().

4) Натренувати початкову модель: self.init_.fit(X, y). Ефектом цієї операції є self.init_.prior = np.log(pos / neg), де pos і neg – це кількість 1 і 0 відповідно у вихідному векторі. Це значення є $log\frac{p}{1-p}$.

5) Зробить початкові прогнози: y_pred = self.init_.predict(X). Результатом цього присвоєння є те, що y_pred є вектором стовпця довжини y, заповненим значенням self.init_.prior.

6) Почніть ітерації (див. BaseGradientBoosting._fit_stages). Робочою одиницею в _fit_ є метод _fit_stage, який викликається один раз на кожній ітерації, приймаючи X, y, y_pred і повертаючи оновлений y_pred, щоб використовувати його на наступній ітерації. X і y є вхідними даними, і вони не змінюються під час

ітерацій.

Для бінарної класифікації _fit_stage відповідає sklearn.tree. DecisionTreeRegressor; більшість роботи делегується останньому класу.

Основні кроки в _fit_stage такі:

1) Обчислюється залишок = self.loss_.negative_gradient(y, y_pred). Цей залишок є $y - \hat{p}$, тобто формула (7) зі знаком мінус.

2) Навчається DecisionTreeRegressor з X як x і residual як y. Цікавим параметром, що передається в DecisionTreeRegressor, є criterion=self.criterion, який за замовчуванням є 'friedman_mse'. Цей критерій використовує середню квадратичну помилку балом покращення за Фрідманом для оцінки потенційних розколів.

3) Дерево, побудоване на попередньому кроці, вирішує розбиття вузлів. Тепер ігнорується значення листка та повторно визначають їх у self.loss_.update_terminal_regions. Це робить дві речі:

a) Визначає значення кожного листка передбачення як: np.sum(residual) / np.sum((y - residual) * (1 - y + residual)). Сумується за вибіркою даних, які потрапляють у поточний лист. Оскільки залишок дорівнює $y - \hat{p}$, значення цього листка дорівнює $\frac{\sum_i (y_i - \hat{p}_i)}{\sum_i \hat{p}_i (1 - \hat{p}_i)}$, де x_i потрапляє на відповідний лист. Ця формула взята з оригінальної роботи Фрідмана. Чисельник – це сума залишків (або негативних градієнтів); знаменник є сумою других похідних, див. формулу (8) вище.

b) Збільшує значення y_pred спостереження i на self.learning_rate, помножене на значення листку, до якого потрапляє це спостереження. Це крок «підвищення». learning rate відповідає за швидкість навчання, зменшує прогноз нового дерева, щоб запобігти перенавчанню. _fit_stage повертає оновлений вектор y_pred є значенням.

У цьому підрозділі ми розглянули теоретичний та практичний підхід щодо алгоритму посилення градієнта. Бустинг градієнта неодноразово доводився як одна з найпотужніших методик для побудови прогнозних моделей як для

класифікації, так і для регресії. Ми визначили основні механізми та математичні моделі, вибрали бібліотеку за допомогою якої будемо реалізовувати алгоритм і можемо переходити до розробки архітектури.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ АНСАМБЕВОГО МЕТОДУ

3.1 Обробка вхідних даних

Першим, що ми зробимо для реалізації даного алгоритму буде аналіз набору даних.

Нашим завданням є передбачення користувачів ride-hailing застосунку, схильних до відтоку, а саме водіїв додатку Уклон, які збираються припинити співпрацю.

Набір даних складається з набору числових даних, щодо роботи водіїв сервісу Уклон за останні 4 тижні до цільового тижня, на якому ми будемо робити передбачення. Нашою цільовою групою є «ліди», тобто постійні користувачі, які в середньому виконують 17 поїздок на тиждень та зареєстровані більше ніж за два місяці до початку досліджень. Набір даних для тренування складається з 30860 рядків та 52 колонок.

Тренувальний набір даних містить:

1) Id - захешований ідентифікатор водія;

2) Week - обернений порядковий номер тижня з чотирьох тижнів, що досліджуються. 0 - найближчий тиждень до цільового тижня;

3) Колонки V1, V2 … V22 та P1, P2, P27 - числові характеристики, щодо роботі водіїв за 4 тижні;

4) target - значення цільової мітки, тобто, якщо 1 – цей водій відноситься до сегменту відтоку, якщо 0 - то відноситься до сегменту лояльних.

Валідаційний набір даних містить:

1) Id - захешований ідентифікатор водія;

2) Week - обернений порядковий номер тижня;

3) Колонки V1, V2 … V22 та P1, P2, P27 - числові характеристики, щодо роботі водіїв за 4 тижні;

Для кожного Id з валідаційного набору даних треба передбачити «предікт»,

тобто передбачення, 1 або 0, до якого сегменту відноситься водій з цим Id. Необхідно спрогнозувати факт відношення до сегменту відтоку або лояльності, без прив'язки до періоду (тижня).

Препроцессинг вхідних даних будемо робити в середовищі jupyter lab.

Отже подивимось на тренувальний набір (рис. 3.1):

```
train_data.head(5)
```

	Id	Week	V1	V2	V3	V4	V5	V6	V7	V8	...	P19	P20	P21	P22	P23	P24	P25	P26	P27	target
0	-6536978109522202983	0	0.000000	0.0	0.000000	0.011364	0.006579	0.0311	0.013158	0.006579	...	15.400000	4.82	8.911111	200.0	0.000000	4.820000	0.333333	6712.0	0.333333	0.0
1	-6536978109522202983	1	0.000000	0.0	0.000000	0.011364	0.006579	0.0311	0.013158	0.006579	...	17.016667	4.82	8.292754	510.0	0.000000	4.820000	0.652174	4873.0	0.391304	0.0
2	-6536978109522202983	2	0.000000	0.0	0.000000	0.011364	0.006579	0.0311	0.013158	0.006579	...	41.100000	4.82	9.753030	190.0	0.000000	4.820000	0.500000	2175.0	0.181818	0.0
3	-6536978109522202983	3	0.000000	0.0	0.000000	0.011364	0.006579	0.0311	0.013158	0.006579	...	33.850000	4.82	10.520614	382.0	0.026316	4.814737	0.447368	4093.0	0.394737	0.0
4	-1744017237843019509	0	0.051282	1.0	0.014363	0.018308	0.037632	0.0000	0.180246	0.103449	...	31.866667	4.94	9.438406	777.0	0.021739	4.908478	0.673913	1296.0	0.239130	0.0

5 rows × 52 columns

```
train_data.sample(5)
```

	Id	Week	V1	V2	V3	V4	V5	V6	V7	V8	...	P19	P20	P21	P22	P23	P24	P25	P26	P27	target
30614	4706660506804653928	2	0.052632	1.0	NaN	NaN	NaN	NaN	NaN	NaN	...	9.066667	4.93	7.788889	100.0	0.0	4.93	0.000000	4486.0	0.000000	1.0
14643	2980655829525145468	3	0.025641	1.0	0.000000	0.000000	0.00000	0.0	0.201389	0.101389	...	17.183333	4.95	7.288333	294.0	0.0	4.95	0.850000	6332.0	0.600000	0.0
25862	-4373018893793908397	2	0.074074	2.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
16584	-3693136671825236106	0	0.025641	1.0	0.011364	0.000000	0.04125	0.0	0.080091	0.163456	...	9.400000	4.94	6.030303	280.0	0.0	4.94	0.863636	2873.0	0.181818	0.0
4028	8121205248966392981	0	0.000000	0.0	0.012920	0.029714	0.00000	0.0	0.189073	0.071636	...	40.583333	4.99	8.976761	240.0	0.0	4.99	0.464789	590.0	0.253521	0.0

5 rows × 52 columns

Рисунок 3.1 – Відображення фрагменту тренувальних даних

Можемо помітити, що для одного Id - є чотири записи, що відповідають його характеристиці за кожен з чотирьох тижнів. Також з’являється гіпотеза (1), що колонки V1-V22 можуть бути загальними характеристиками водіїв, так як вони від тижня до тижня не відрізняються. Це можуть бути такі дані, як кількість днів користування застосунком до початку досліджуваних 4-ох тижнів; кількість неробочих тижнів до початку досліджень; рейтинг; середній дохід тощо.

Також за допомогою функції sample(5), яка обирає випадковим чином рядки з датасету, бачимо, що набір даних містить NaN значення. Також, бачимо, що дані не нормовані, отже з часом нам треба буде їх стандартизувати.

Перевіримо гіпотезу щодо «загальних даних» позначених V1-V22та відобразимо це на рисунку 3.2:

```
list_columns = list(train_data.T.loc['V1':'V22'].index) + ['Id']
```

```
(train_data[list_columns].groupby('Id').mean()/train_data[list_columns].groupby('Id').first()).mean()
```

```
V1     1.0
V2     1.0
V3     1.0
V4     1.0
V5     1.0
V6     1.0
V7     1.0
V8     1.0
V9     1.0
V10    1.0
V11    1.0
V12    1.0
V13    1.0
V14    1.0
V15    1.0
V16    1.0
V17    1.0
V18    1.0
V19    1.0
V20    1.0
V21    1.0
V22    1.0
dtype: float64
```

Рисунок 3.2 – Перевірка Гіпотези 1

З цього можемо зробити висновок, що дійсно колонки V1-V22 статичні для кожного з тижнів. Так, як нам треба звести дані до вигляду: один Id - один рядок даних за 4 тижні, будемо враховувати, що колонки V1-V22 статічні.

Перевіримо, чи є в даних безкінечні значення та NaN (рис. 3.3):

```
train_data.isin([np.inf, -np.inf]).sum().sum()
```

```
0
```

```
train_data.isna().sum()
```

```
Id           0
Week         0
V1          48
V2          48
V3        7100
V4        7100
V5        7100
V6        7100
V7        7100
V8        7100
V9        7100
V10       7100
```

Рисунок 3.3 – Перевірка даних на безкінечність та відсутність

Безкінечних даних, які б могли бути отримані при ділення на 0 або -0 немає. Є відсутні дані – NaN, заповнемо їх нулями. Якщо перформанс на виході буде поганим, більш детально повернемося до цього питання та спробуємо заповнити відсутні дані за допомогою алгоритму найближчих сусідів.

Тепер розбиремо варіанти, як можна дані за 4 тижні звести до одного рядку.

Найпростіший варіант – взяти середнє за 4 тижні. Але, так як для коректної роботи моделі треба врахувати зміну поведінки протягом 4 тижнів, правильним буде використати інформацію по кожному тижню окремо. Отже, трансформуємо колонки та тижні в один рядок, наприклад (рис. 3.4):

Id	Week	P1
4706660506804653928	0	5
4706660506804653928	1	6
4706660506804653928	2	2
4706660506804653928	3	10

Id	P1_week_0	P1_week_1	P1_week_2	P1_week_3
4706660506804653928	5	6	2	10

Рисунок 3.4 – Трансформація вхідних даних в рядок

Стикуємо дані [V1, …, V2] та [P1_week_0, … P27_week_3] – отримуємо вихідний датасет (рис. 3.5):

train

Id	target	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	P25_week_2	P25_week_3	P26_week_0	P26_week_1	P26_week_2	P26_week_3
-9217315602200933985	0.0	0.363636	7.0	0.071160	0.078476	0.000000	0.000000	0.268198	0.166650	0.000000	...	0.000000	0.000000	1374.0	3085.0	821.0	1241.0
-9213274821386443078	1.0	0.025641	1.0	0.014706	0.000000	0.000000	0.000000	0.000000	0.000000	0.284314	...	0.941176	0.833333	2929.0	3968.0	2353.0	4073.0
-9208710115890940128	0.0	0.564103	22.0	0.015915	0.023058	0.066416	0.015915	0.202506	0.079574	0.105388	...	0.578947	0.474576	5334.0	3196.0	5535.0	1647.0
-9205482426769313939	0.0	0.000000	0.0	0.071182	0.035808	0.007812	0.007812	0.082472	0.118727	0.071311	...	0.714286	0.540541	2754.0	7642.0	5211.0	3983.0
-9204742353296425998	0.0	0.076923	1.0	0.000000	0.183557	0.000000	0.000000	0.315973	0.067457	0.006098	...	0.470588	0.195122	6408.0	6314.0	3766.0	3387.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9215456666947967643	0.0	0.000000	0.0	0.005435	0.454189	0.021786	0.000000	0.292128	0.011249	0.016018	...	0.695652	0.693878	2700.0	1727.0	2484.0	2078.0
9217675734888214114	0.0	0.051282	2.0	0.032596	0.079697	0.010870	0.000000	0.170562	0.083644	0.069876	...	0.339286	0.333333	1756.0	2598.0	2734.0	3629.0
9220558206402716117	0.0	0.051282	2.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.492308	1496.0	1146.0	0.0	136.0
9222188311406951236	0.0	0.000000	0.0	0.000000	0.085813	0.000000	0.000000	0.224405	0.000000	0.081151	...	0.000000	0.000000	3178.0	2120.0	1993.0	5942.0
9222566437483689898	0.0	0.051282	1.0	0.035890	0.014838	0.010990	0.000000	0.063873	0.097475	0.123089	...	1.000000	1.000000	4006.0	3218.0	3116.0	3103.0

7715 rows × 131 columns

Рисунок 3.5 – Зведені дані до вигляду Id - один рядок даних

Переходимо до стандартизації даних. Стандартизація простору ознак є загальною вимогою для ефективного навчання моделей, оскільки більшість з них є чутливими до того, як масштабуються вхідні вектори. У багатьох наборах даних функції мають різний масштаб і різні діапазони значень, що призводить до збільшення часу навчання. Тому корисно стандартизувати всі об'єкти таким чином, щоб кожен об'єкт перетворювався в стандартний нормальний розподіл шляхом видалення серенього значення кожного об'єкта, а потім масштабування його шляхом ділення на їх стандартні відхилення. Для нормалізації оберемо дані, які знаходяться в діапазоні більше ніж від 0 до 1, та використаєм preprocessing.normalize з бібліотеки sklearn (рис. 3.6):

```
to_normalize = list(train.describe().T[train.describe().T['max'] > 1].index)
```

```
norm_train = train.copy()
```

```
norm_train[to_normalize] = preprocessing.normalize(norm_train[to_normalize], axis=1)
```

```
norm_train.sample(5)
```

Id	target	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	P25_week_2
-7511299306446866268	1.0	0.000000	0.000000	0.031250	0.000000	0.000000	0.0	0.125000	0.161458	0.015625	...	0.500000
-8901521022512353938	0.0	0.076923	0.000071	0.007812	0.000000	0.000000	0.0	0.000000	0.000000	0.960693	...	0.638889
3742033105195017581	0.0	0.000000	0.000000	0.282431	0.000000	0.000000	0.0	0.000000	0.000000	0.116794	...	0.540541
-6848755946015712594	1.0	0.051282	0.000010	0.058333	0.008333	0.022222	0.0	0.041667	0.008333	0.069444	...	0.000000
-6636068365645490912	0.0	0.055556	0.000031	0.016667	0.023423	0.010417	0.0	0.126325	0.209936	0.056250	...	0.666667

Рисунок 3.6 – Нормалізація даних

Перевіримо збалансованість даних (рис. 3.7):

```
: norm_train.value_counts('target')

: target
  0.0    6379
  1.0    1336
```

Рисунок 3.7 – Співвідношення сегментів

Як бачимо, сегмент схильних до відтоку водіїв в декілька разів менший за лояльних, це буде необхідно врахувати в подальших обчисленнях та при виборі метрики для оцінювання точності моделі.

У наборі даних ми розглядаємо сегмент відтоку як позитивний клас, а сегмент лояльних - як негативний клас. Існують чотири прогнозуючі результати, які можна сформулювати в матриці 2 × 2, як показано в табл. 3.1. Стовпець представляє прогнозовану мітку, а рядок представляє справжню мітку. TN – це число лояльних, які правильно класифіковані (True Negative), FP – це кількість лояльних водіїв, які неправильно класифіковані як схильні до відтоку (False Positive), FN – кількість схильних до відтоку, які неправильно класифіковані як лояльні (False Negative), а TP – це число схильних до відтоку, які правильно класифіковані (True Positive).

Таблиця 3.1 – Чотири прогнозуючі результати

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Ефективність алгоритму навчання моделі зазвичай оцінюють з використанням прогнозної точності, яка визначається як точність = (TP + TN) / (TP + FP + TN + FN). Але маючи незбалансовані дані, краще використовувати

ROC криву - це стандартна методика оцінки продуктивності моделі для двійковій класифікації (рис. 3.8):

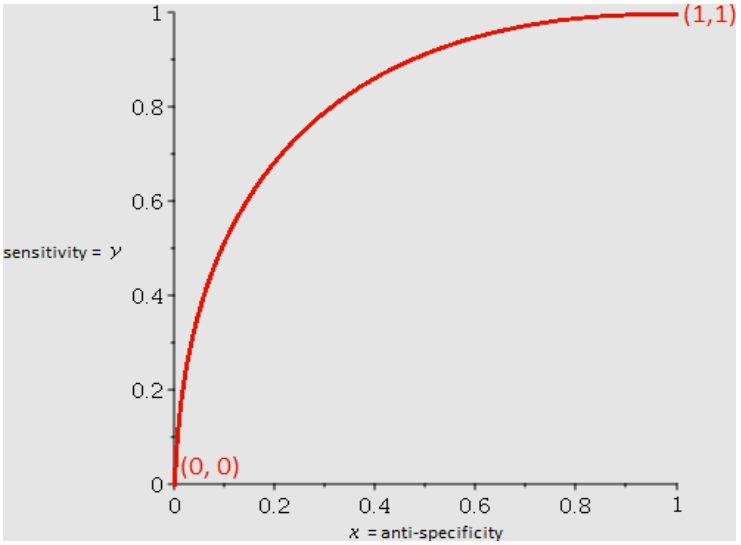

Рисунок 3.8 – ROC-крива

ROC-крива описує характеристики моделі, які представлені співвідношенням між істинної позитивної оцінкою (TPR) і помилкової позитивної оцінкою (FPR). У просторі ROC вісь X визначається як $\%FP = \frac{FP}{(TN+FP)}$, а вісь Y визначається як $\%TP = \frac{TP}{(TT+FN)}$. Найкраще прогнозування відбувається в точці (0,1) в просторі ROC, що представляє всі позитивні класи, класифіковані правильно, і жодні негативні класи не класифікуються як позитивні класи. Чим ближче крива ROC до верхнього лівого кута, тим вище точність моделі. Тому криві ROC можуть легко відобразити порівняння характеристик між різними моделями. Площа під кривою ROC (AUC) - ефективний показник для підведення підсумків роботи кривої ROC. Як правило, крива ROC з великим AUC показує кращу модель продуктивності.

Тож точність нашої моделі будемо оцінювати по метриці AUC.

3.2 Реалізація Градієнтного бустингу

Отже, враховуючи все вищесказане побудуємо модель градієнтного бустингу. Спочатку розбиваємо навчальний датасет на тренувальний і тестовій. Це можна зробити за допомогою функції train_test_split з бібліотеки sklearn.model_selection (рис. 3.9).

```
# Use a utility from sklearn to split and shuffle our dataset.
train_, test_ = train_test_split(norm_train, test_size=0.3)


# Form np arrays of labels and features.
train_df_labels = train_.pop('target')
train_labels = np.array(train_df_labels)


test_df_labels = test_.pop('target')
test_labels = np.array(test_df_labels)

train_features = np.array(train_)
test_features = np.array(test_)
```

Рисунок 3.9 – Розбиття вибірки даних на тренувальні і навчальні

Використаємо клас GradientBoostingClassifier з sklearn.ensemble бібліотеки для створення екземпляру класа градієнтного бустингу з початковими параметрами враховуючи специфіку поставленої задачі (рис. 3.10):

```
gradient_booster = GradientBoostingClassifier(learning_rate=0.1,
                                              criterion= 'friedman_mse',
                                              loss= 'log_loss',
                                              max_depth= 3,
                                              min_samples_leaf= 1,
                                              min_samples_split= 2)
```

Рисунок 3.10 – Створення екземпляру класу ансамблевого методу

Тепер тренуєм модель (рис. 3.11):

```
gradient_booster.fit(X_train,y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

Рисунок 3.11 – Використання методу fit на тренувальних даних

Перевіремо точність моделі за допомогою precision та recall та побудуємо ROC-криву (рис. 3.12).

```
print(classification_report(y_train,gradient_booster.predict(X_train)))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98      5089
         1.0       0.93      0.84      0.88      1083

    accuracy                           0.96      6172
   macro avg       0.95      0.91      0.93      6172
weighted avg       0.96      0.96      0.96      6172
```

```
metrics.plot_roc_curve(gradient_booster, X_train,y_train)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f09f061ca00>
```

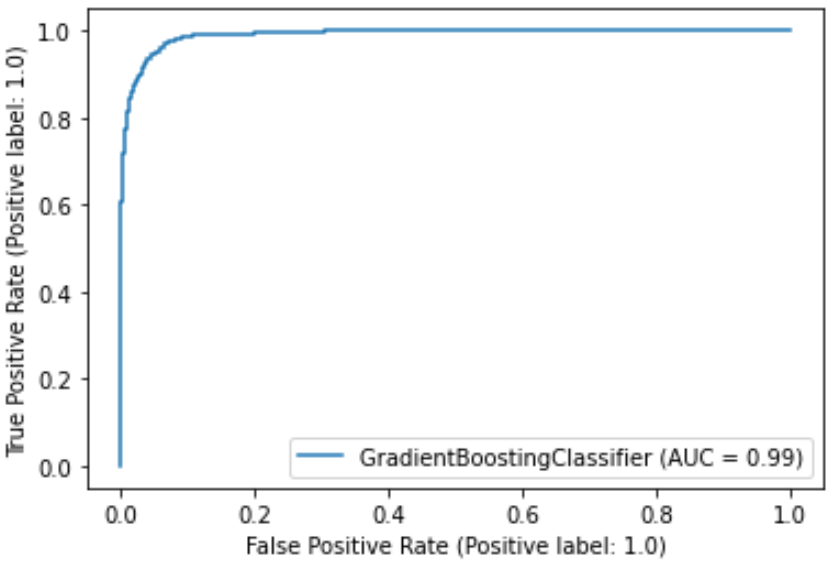

Рисунок 3.12 – Перевірка точності моделі

Значення AUC 99 %, модель натренована дуже добре. Перевіримо, чи не перевчилася модель, для цього побудуємо ROC-криву для тестової вибірки та перевіримо з тренувальною (рис. 3.13).

```
import matplotlib.pyplot as plt
from sklearn import metrics, model_selection
metrics.plot_roc_curve(gradient_booster, X_val, y_val)

plt.show()
```

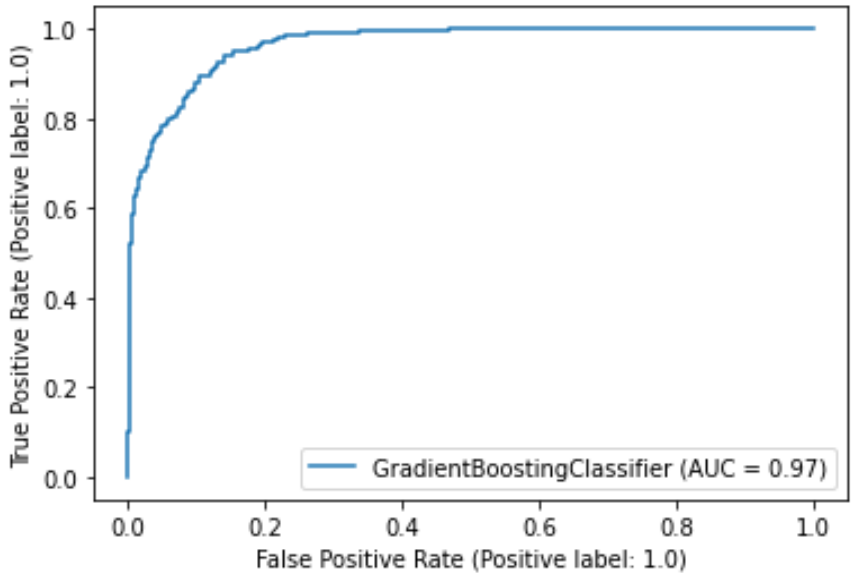

Рисунок 3.13 – Перевірка на перенавчання

Отже на тестових даних AUC складає 97 %, що задовольняє умови задачі.

Тепер необхідно побудувати «feature importance» для того, щоб знайти, на які колонки модель спиралася найчастіше при вирішенні задачі. Так ми можемо зрозуміти, як саме впливають ті чи інші фактори на поведінку групи, яку досліджують. Використаємо бібліотеку shap для розрахунку shap values. Це допоможе нам дізнатися не тільки важливість, але і як саме впливає: негативно чи позитивно.

Червоне забарвлення мають позитивний клас – тобто клас схильний до відтоку, синє – лояльні. Розташування праворуч від центру – означає негативний

вплив на клас, ліворуч – позитивний (рис. 3.14).

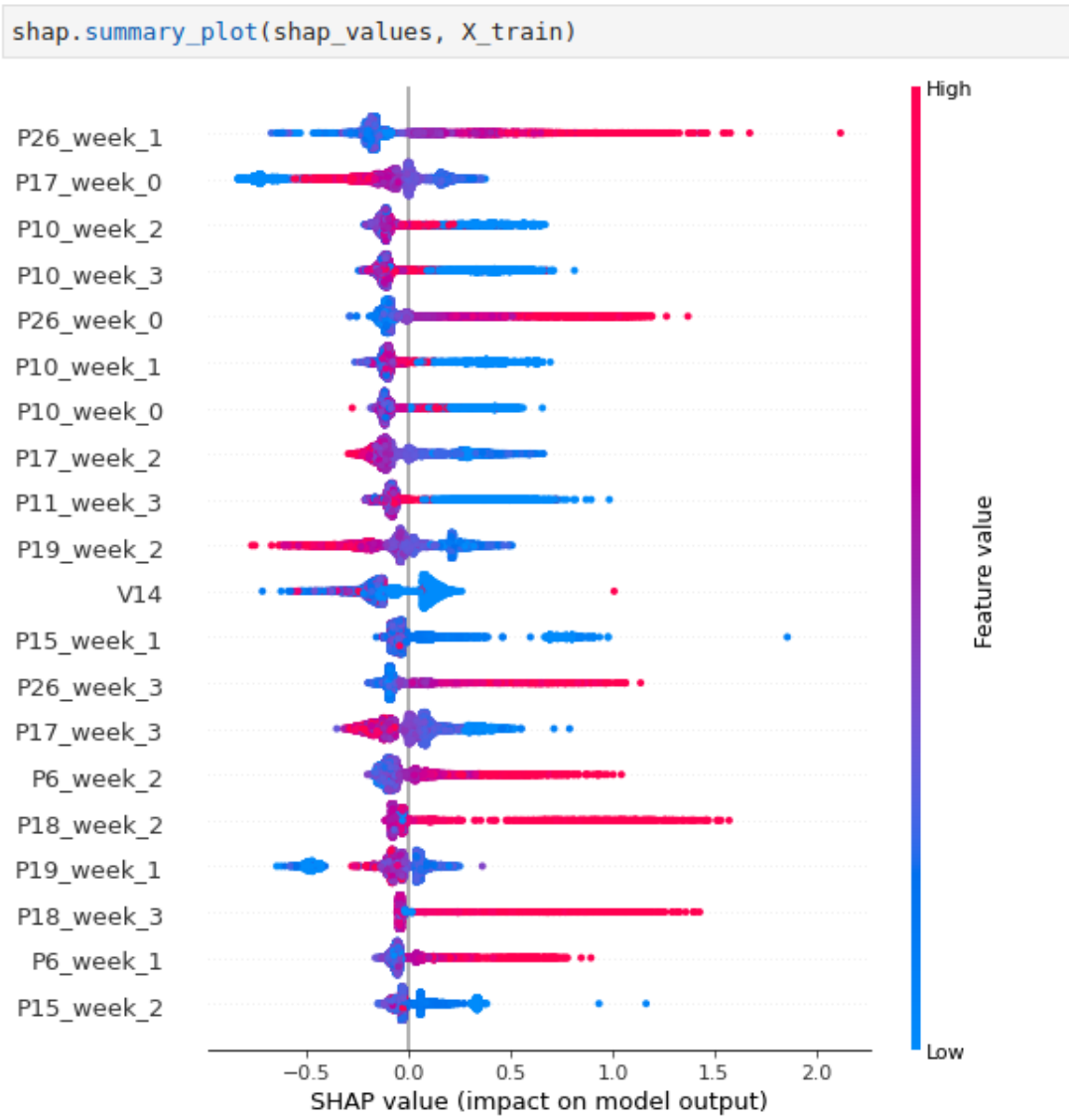

Рисунок 3.14 – Shap values для дослідження важливості колонок

Далі переходимо до пункту: «Користувачів з груп ризику з середньою ймовірністю відтоку більше ніж за 80 % необхідно передати у форматі csv у відділи утримання клієнтів для подальшої взаємодії.»

Для цього порахуємо ймовірність відтоку, за допомогою функції predict_proba, та присвоємо кожному Id групу, до якої він відноситься. Далі згрупуємо Id в групи ризику та перевіримо який дійсний відсоток відтоку в кожній групі.

```
pd.merge(train_df_labels.reset_index(name='target'), answers_test, on='Id').groupby('percent_group')['target'].mean()
```

```
percent_group
(-0.000535, 0.1]    0.000000
(0.1, 0.2]          0.000000
(0.2, 0.3]          0.000000
(0.3, 0.4]          0.000000
(0.4, 0.499]        0.000000
(0.499, 0.599]      0.034884
(0.599, 0.699]      0.301205
(0.699, 0.799]      0.552083
(0.799, 0.898]      0.847826
(0.898, 0.998]      0.976090
Name: target, dtype: float64
```

Рисунок 3.14 – Групи ризику відтоку

Як бачимо з рисунку 3.14 в групу з передбаченням 0.8 і вище попадають сегмент схильний до відтоку з ймовірністю 84.7 і вище. Тобто користувачів, в яких предікт більший ніж 80 % ми зберемо в csv файл та передамо відділу утримання клієнтів для подальшої роботи з цім сегментом.

На цьому наша робота виконана, зберігаємо модель для подальшого звернення її в даг для автоматичної роботи раз в місяць.

Докладно реалізацію ансамблевого методу на основі sklearn подано у Додатку А.

ВИСНОВКИ

Поставленої мети було досягнуто, у ході виконання кваліфікаційної роботи, а саме було розглянуте питання автоматичного виявлення сегменту користувачів схильного до відтоку та було обрано стратегію та алгоритм для вирішення задачі. Були розглянуті основні причини відтоку, та були прийняті до уваги складнощі, що виникають через специфіку вхідних даних та особливостей поставленої задачі.

На основі даних, які предоставила компанія Уклон, був розроблений ансамблевий алгоритм градієнтного бустингу з використанням різних технологій для виявлення водіїв схильних до відтоку. Було проведено підготовка вихідних результатів для передачі спеціалізованому відділу для подальшої праці з виявленими клієнтами.

Були використані мови програмування Python та бібліотеки sklearn, shap, pandas і так далі, інструменти є легко масштабованими та не викликають труднощів під час інтеграції у даг для автоматичного запуску.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Forrester. Zhi-Ying Barry. Reduce Customer Churn And Drive Banking Customer Loyalty In Australia – 2020: [Електронний ресурс] / Режим доступу: https://www.forrester.com/report/reduce-customer-churn-and-drive-banking-customer-loyalty-in-australia/RES175330?ref_search=0_1653247712279

2. Harvard Business School report. The Economics of E-Loyalty. – 2000: [Електронний ресурс] / Режим доступу: https://hbswk.hbs.edu/archive/the-economics-of-e-loyalty

3. KPMG. Global Customer Experience Excellence report – 2021: [Електронний ресурс] / Режим доступу: https://home.kpmg/xx/en/home/services/advisory/management-consulting/customer-first-insights/global-customer-experience-excellence-report-2021.html

4. Brett Sappington. Adoption, Churn, and the Risky Lives of OTT Video Services. – 2015: [Електронний ресурс] / Режим доступу: https://www.parksassociates.com/blog/article/pr-0414016

5. Michael K., Thoughts on Hypothesis Boosting. – 1988.

6. Leslie Valiant's. Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World. – 2013. – P. 152.

7. A decision-theoretic generalization of on-line learning and an application to boosting. – 1995.

8. Applied Predictive Modeling. – 2013. – P. 369.

9. Prediction Games and Arching Algorithms. – 1997. – P. 27

10. Boosting Algorithms as Gradient Descent in Function Space. – 1999. – P.12

ДОДАТОК А

Градієнтний бустинг на основі Skikit-learn

```
import warnings
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn import metrics, model_selection
from sklearn.inspection import permutation_importance

train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

train_data.groupby('Id').mean().target.value_counts()
train_data.head(5)
train_data.sample(5)
list_columns = list(train_data.T.loc['V1':'V22'].index) + ['Id']
(train_data[list_columns].groupby('Id').mean()/train_data[list_columns].groupby('Id').fir
st()).mean()
train_data.isin([np.inf, -np.inf]).sum().sum()
train_data = train_data.fillna(0)
staked = []
```

```
for c in list(train_copy.T.loc['P1':'P27'].index):
    res = train_copy.pivot(index='Id', columns=['Week'], values=c)
    res.columns = [f'{c}_week_{i}' for i in res.columns]
    staked.append(res)
train_df = pd.concat(staked, axis=1)
train_df2 = train_data[['target'] + list_columns].groupby('Id').first()
train = pd.concat([train_df2, train_df], axis=1)
to_normalize = list(train.describe().T[train.describe().T['max'] > 1].index)
norm_train = train.copy()
norm_train[to_normalize] = preprocessing.normalize(norm_train[to_normalize],
axis=1)
#build model
X_train, y_train = train_test_split(norm_train, test_size=0.3)
gradient_booster = GradientBoostingClassifier(learning_rate=0.1,
                                              criterion= 'friedman_mse',
                                              loss= 'log_loss',
                                              max_depth= 3,
                                              min_samples_leaf= 1,
                                              min_samples_split= 2)
gradient_booster.get_params()
gradient_booster.fit(X_train,y_train)
print(classification_report(y_train,gradient_booster.predict(X_train)))
metrics.plot_roc_curve(gradient_booster, X_val, y_val)
plt.show()
metrics.plot_roc_curve(gradient_booster, X_train,y_train)
importances = pd.DataFrame(np.array([pd.Series(X_train.columns),
pd.Series(gradient_booster.feature_importances_*100./np.sum(gradient_booster.feature
_importances_))]).transpose(),
                           columns=[«Name», «Importance»]).sort_values('Importance',
ascending=False)
```

```
probas_test = gradient_booster.predict_proba(X_val)
answers_test = pd.DataFrame(probas_test, columns=['first_pred', 'second_pred' ])
answers_test['Id'] = list(X_val.index)
answers_test['percent_group'] = pd.cut(answers_test['second_pred'], 10)
print(answers_test)
answers_test[answers_test['second_pred'] > 0.9]
explainer = shap.TreeExplainer(gradient_booster)
shap_values = explainer.shap_values(X_val)
shap.summary_plot(shap_values, X_val)
from sklearn.model_selection import StratifiedKFold
parameters = {
   «n_estimators»:[50,100,200,250],
   «max_depth»:[1,2,3,4],
   «learning_rate»:[0.01, 0.05, 0.1, 1]
}
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
gbc = GradientBoostingClassifier()
cv = GridSearchCV(gbc,parameters,cv=5)
cv.fit(train_features,train_label.values.ravel())
folds = 3
param_comb = 5
skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 777)
random_search = RandomizedSearchCV(gbc, param_distributions=parameters,
scoring='accuracy', n_jobs=4, cv=skf.split(X_train,y_train), verbose=3,
random_state=777 )
random_search.fit(X_train,y_train)
print(random_search.best_score_)

resul = answers_test[answers_test['second_pred'] > 0.8].to_csv(‘DATA/result.csv’)
```