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

Прогнозування часових рядів за допомогою деревоподібного бустингового алгоритму LightGBM у поєднанні з лінійною регресією

Текстова частина до курсової роботи
за спеціальністю „Прикладна Математика” 113

Керівник курсової роботи
к.т.н., доц. Жежерун О. П.

_____

(підпис)

“____”_____ 2022 р.

Виконав студент 3 р.н.
Кирпа М.Г.
“14” червня 2022 р.

м. Київ – 2022 рік

Календарний план виконання курсової роботи

Тема: Розробка нейронної мережі для розпізнавання графічних об’єктів

Календарний план виконання роботи:

№ п\п	Назва етапу дипломного проекту(роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень - листопад 2021р.	
2.	Огляд літератури за темою роботи	Листопад-лютий 2021р.	
3.	Аналіз підходів до прогнозування часових рядів	Лютий-березнь 2022р.	
4.	Деревоподібні алгоритми машинного навчання	Березень 2022р.	
5.	Написання текстової частини	Квітень 2022р.	
6.	Перегляд змісту роботи керівником	Травень 2022р.	
7.	Створення презентації	Червень 2022р.	
8.	Захист курсової роботи	Червень 2022р.	

Студент Кирпа М.Г. _____

Керівник Жежерун О.П. _____

“_____”_____2022 р.

Зміст

ВСТУП

Сучасний світ генерує терабайти даних щогодини, усі користувачі інтернету залишають свій слід куди б вони не заходили. Веб пошук, соціальні мережі, онлайн магазини, тощо - вся ця інформація є дуже важливою, як для комерційного. так і для соціального аналізу поведінки людей.

Звичайно ж постає логічне запитання, як саме оперувати цими масивами даних і яку вигоду можна отримати з них? Саме цим займається галузь під назвою “Машинне Навчання”, більшість алгоритмів якого вигадано багато років тому, але саме зараз людство створило необхідні пристрої що можуть швидко обробляти і допомагати аналізувати великі об’єми інформації. Тож пророкувати майбутнє можна навіть на базовому комп’ютері вдома.

Одне з найцікавіших відгалужень Машинного Навчання це прогнозування часових рядів, через те що людям завжди цікаво як саме буде змінюватися навколишній світ з часом і що на це впливає. Наразі розроблено величезну кількість алгоритмів що непогано справляються з подібною проблемою, але вони мають ряд недоліків.

Мета цієї курсової роботи - оглянути базовий метод прогнозування часових рядів і запропонувати новий підхід з деякими архітектурними відмінностями.

РОЗДІЛ 1: БАЗОВІ АЛГОРИТМИ ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ

1.1 Визначення часового ряду

Часовий ряд - це сукупність точок, в хронологічному порядку, рівновіддалених в часі, що являють собою значення будь-якого об’єкта або повторюваної події в минулому. Як приклад це може бути: погодинний курс валют, щодення кількість інфікованих на грип у світі, щомісячна кількість проданих товарів супермаркетом,

тощо.

1.2 Складові компоненти часового ряду

На перший погляд, може здаватися, що часовий ряд це хаотично розкидані точки у просторі і не дуже зрозуміло як саме його можна аналізувати, і тим паче будувати прогнози на основі цих даних. Тож для спрощення задачі у будь якому часовому ряді прийнято виділяти 3 компоненти:

- Тренд - основний довгостроковий напрямок часового ряду
- Сезонність - залежність від місяця, пори року, тощо
- Нерегулярність - несистематичні, короткострокові впливи

1.2.1 Тренд

Тренд - це довгостроковий рух часового ряду у просторі подій. Він є дуже важливою компонентою, допомогою якої можна легко визначити чи взагалі буде зменшуватися або ж навпаки зменшуватися числовий показник спростаігаємого об’єкта у перспективі.

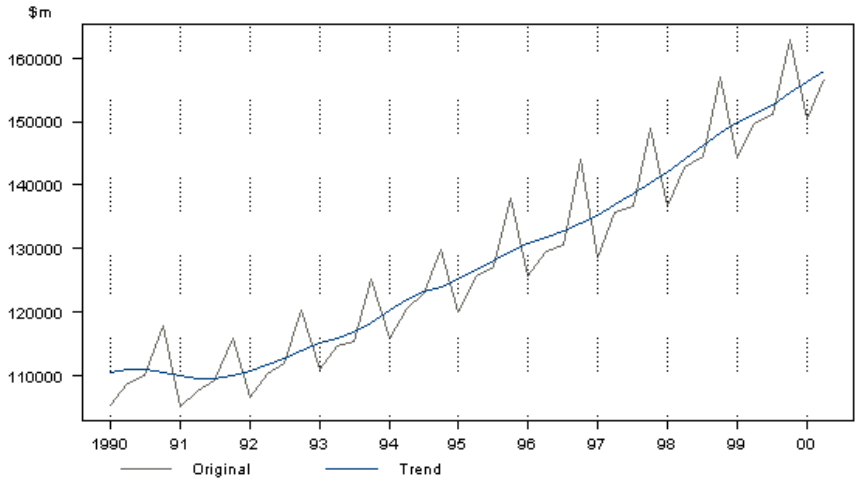

Рис. 1 Квартальний ВВП Південного Уельса

На рисунку 1 можна побачити - тренд це крива, що відображає поведінку часового ряду за весь проміжок спостережень, без додаткового шуму у вигляді сезонності та несистематичних змін.

1.2.2 Сезонність

Майже у всіх видах людської діяльності присутня сезонність. Людина не буде носити зимній одяг влітку, що в свою чергу впливає на попит у магазинах, або ж користуватись додатками для заповнення декларацій у не податковий період. Тож сезонні коливання у часових рядах можуть бути визначені як регулярні піки та спадання з приблизно однаковою амплітудою і послідовним напрямком.

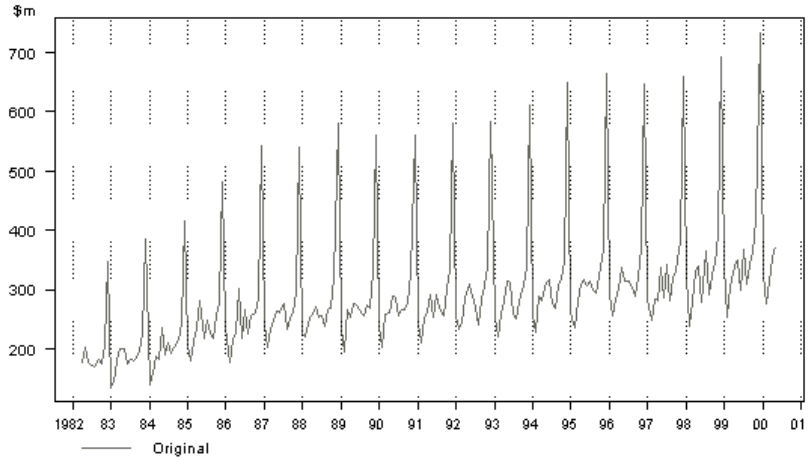

Рис. 2 Місячні продажі товарів у Південному Уельсі

З рисунку 2 видно, що в залежності від пори року сума на яку продано товари зростає і спадає відповідно, через ажіотаж перед Новим Роком, зазвичай, люди купують продукти значно активніше. Амплітуда сезонних коливань може змінюватись з часом залежно від попиту та зовнішніх чинників.

1.2.3 Нерегулярність

Остання, але не менш важлива компонента - нерегулярність. Це несистематична зміна у поведінці часового ряду під впливом зовнішніх факторів які неможливо передбачити, як от різка зміна погодних умов, соціально-політична ситуація, епідемія, тощо.

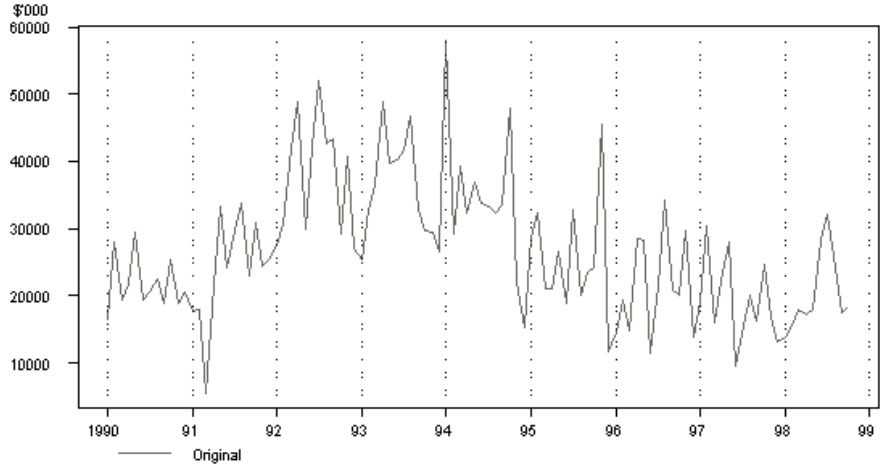

Рис. 3 Середньомісячна вартість квартир

Нерегулярна компонента залишається після прибирання сезонності та тренду з часового ряду.

1.2.4 Стаціонарність часового ряду

Більшість алгоритмів аналізу часових рядів спрацьовують тільки у випадку якщо ряд є стаціонарним, тобто його математичне очікування і дисперсія є постійними. Для того щоб досягти стаціонарності з ряду потрібно прибрати тренд, сезонність та нерегулярність, на практиці цього можна досягти за допомогою критеріїв Стьюдента і Фішера та подальших математичних перетворенн.

1.3 Моделі

Найпоширеніша модель що використовується для аналізу і прогнозування часових рядів - це модель авторегресії - інтегрованого ковзного середнього, або ж модель Бокса — Дженкінса.

1.3.1 Лінійна регресія

Перше що спадає на думку при аналізі двох змінних які пов’язані між собою і одна залежить від іншої це - прогнозування за допомогою лінійної регресії.

Простими словами лінійна регресія - це пошук найбільш оптимальної прямої що описує поведінку залежної змінної, з урахуванням поведінки незалежної. Більш формальне означення: лінійна регресія — це метод моделювання залежності між скалярною змінною y та векторною змінною X. Модель лінійної регресії для 2 змінних можна описати наступною математичною функцією:

$$y = c + bx$$

Де:

- x - незалежна змінна
- y - залежна змінна
- b - параметр який описує зв’язок між змінними
- c - випадкова похибка

У випадку з аналізом часових рядів лінійну регресію можна застосовувати задля пошуку тренд, але прогнози побудовані за допомогою такої моделі будуть дуже не точними. Тому її прийнято використовувати лише для поверхневого або ж додаткового аналізу.

1.3.2 Авторегресійна модель

Більш просунутий підхід з використанням регресійних моделей є прогнозування часового ряду за допомогою авторегресії. Якщо в лінійній регресії для декількох змінних для побудування моделі використовується лінійна комбінація значень незалежних змінних, то в авторегресії незалежними змінними виступають минулі значення незалежної змінної. Авторегресійна модель є надзвичайно гнучкою і зазвичай, гарно вправляється з більшістю стаціонарних часових рядів.

Авторегресію можна представити в наступному вигляді:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

Де:

- y_t - значення змінної у момент часу t
- y_{t-1}, y_{t-2}, ..., y_{t-p} - значення змінної у в попередні моменти часу
- c - початкова стала
- ε_t - випадкова похибка(білий шум)
- $\phi_1, \phi_2, \ldots, \phi_p$ - змінні що впливають на різні паттерни моделі
- p - порядок залежності від минулих значенньь моделі(скільки саме минулих спостережень впливають на розрахунок поточного)

  Модель позначається як AR(p).

1.3.3 Модель ковзного середнього

Ще одним підходом до прогнозування часових рядів є модель ковзного середнього - MA(q), але на відміну від авторегресії ця модель використовує попередні “помилки” у вигляді псевдо регресійної моделі, під помилками тут мається на увазі різниця між реальними і теоритичними значеннями змінної у.

Не слід також плутати змінне середнє з цієї моделі з ковзним середнім у методі зглажування часового ряду, так як тут ми прогнозуємо майбутнє значення у, а в методі зглажування нормалізуємо минулі значення.

Модель ковзного середнього являє собою:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

Де:

- y_t - значення змінної у момент часу t
- c - початкова стала
- ε_t, ε_{t-1}, ..., ε_{t-q} - випадкова похибка(білий шум) у поточний та попередні моменти часу
- $\theta_1, \theta_2, \ldots, \theta_p$ - змінні що впливають на різні паттерни моделі
- q - порядок залежності від минулих значенньь моделі

1.3.4 Модель авторегресії - ковзного середнього

За допомогою поєднання моделей AR та MA досягається значне підвищення точності прогнозування та аналізу часового ряду і таку модель називають ARMA.

Якщо ряд є стаціонарним, ARMA з високою ймовірність надасть точну оцінку майбутніх значень.

1.3.5 Модель авторегресії - інтегрованого ковзного середнього

Найпоширеніша модель, що використовується для аналізу і прогнозування часових рядів - це модель авторегресії - інтегрованого ковзного середнього(ARIMA), або ж модель Бокса — Дженкінса. Що в свою чергу є розширенням до моделі ARMA і має змогу обробляти нестаціонарні часові ряди. Можливість обробки нестаціонарних часових рядів досягається за допомогою різноманітних тестів, що виявляють наявність одиничних коренів і порядку інтегрування моделі.

В подальшому ряд стаціонаризують за допомогою різниці відповідного інтегрованого порядку(якщо порядок > 0) і вже для такого ряду застосовується модель ARMA.

З математичної точки зору модель ARIMA(p, d, q) виглядає наступним чином:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

Де:

- y'_t - диференційоване значення часового ряду в поточний момент(з порядком диференційованності d)
- інші компоненти з AR та MA

Якщо порядок інтегрованості часового ряду = 0, то ряд є стаціонарним і для аналізу застосовується модель ARMA

1.3.6 Модель сезонної авторегресії - інтегрованого ковзного середнього

Як ми бачимо з опису модель ARIMA достаньо вправно справляється з прогнозуванням часових рядів, але в неї є один великий недолік, ця модель приймає часовий ряд з припущенням що в ньому немає сезонної компоненти,

саме тут на допомогу приходить модель SARIMA або ж модель сезонної авторегресії - інтегрованого ковзного середнього.
Формально, SARIMA$(p,\ d,\ q)(P,\ D,\ Q)_m$ є такою ж як ARIMA$(p,\ d,\ q)$, але додатково треба підібрати сезонні компоненти авторегресії, інтегровності, та ковзного середнього.

1.3.7 Інші моделі

Звичайно ж для пронозування часових рядів існує велика кількість різноманітних статистичних алгоритмів таких як: Naive Bayes Model, FBprophet(from facebook) та ін. Всі вони використовують подібну структуру розділення часового ряду на компоненти і пронозування виключно на основі стаціонарного часового ряду.

Але недоліками таких моделей є пошук гіперпараметрів для стаціонарізації кожного часового ряду окремо, що може ускладнювати процесс аналізу у випадках коли в великих комерційних компаніях треба опрацьовувати сотні, а деколи й тисячі часових рядів за короткий проміжок часу.

РОЗДІЛ 2: ДЕРЕВОПОДІБНІ АЛГОРИТМИ

2.1 Визначення деревоподібних алгоритмів машинного навчання

Наразі у галузі машинного навчання, одні з найкращих алгоритмів, що опрацьовують табличні данні, визнано деревоподібні. Великою перевагою над різноманітними регресійними алгоритмами є можливість вправно знаходити нелінійні зв’язки у даних. Також такі алгоритми можуть вирішувати як класифікаційні проблеми, так і регресивні.

Подібні алгоритми називаються деревоподібнми, через те що вони побудовані на основі бінарного дерева приняття рішень.

2.2 Алгоритми

Звичайно деревоподібні алгоритми поділяються на різні категорії за рівнями складності та використими технологіям. У цьому підрозділі, я хочу розглянути:

- Дерево прийняття рішень
- Алгоритм поєднання декількох дерев на основі голосування
- Рандомний ліс дерев прийняття рішень
- AdaBoost
- Light GBM

Всі ці алгоритми використовують свої унікальні технології, але водночас наслідують риси бінарного дерева.

2.2.1 Дерево прийняття рішень

Походження дерева прийняття рішень достатньо важко визначити, вперше з’являються згадки про подібний алгоритм у роботі Рональда Фішера про дискримінаційний аналіз 1936 року. Також у психології 1960-х років використовували метод дерева прийняття рішень для моделювання людської концепції навчання. У 1972 році в проекті THAID Месенджера і Менделла з'явилося перше дерево класифікації. І вже у 1977 році професори статистики Берклі Лео Брейман і Чарльз Джоел Стоун разом з Джеромом Х. Фрідманом і

Річардом Олшеном зі Стенфордського університету розробили перший алгоритм дерева класифікації та регресії (CART).

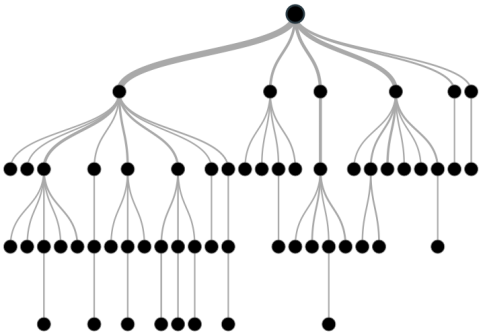

Рис. 4 Схема бінарного дерева

Його назва є такою через те, що виглядом воно нагадує перегорнуте гілками вниз дерево, як представлено на рисунку вище.

У своїй основі технологія, що використана задля реалізації ДПР - це послідовне розбиття набору даних на 2 або ж декілька однорідних підгруп за якоюсь властивістю. Параметрами в такій моделі є:

- глибина дерева - максимальна кількість рівнів листків
- мінімальна кількість спостережень в кожному листку

Дерево прийняття рішень може поділяти на групи як за категоріальною так і за числовою властивістю.

Як приклад представимо що в нас є вибірка з 100 людей обох статей у вікових групах від 0 до 18, від 18 до 60 та від 60 років(в кожній групі рівна кількість людей). На першому рівні буде створено 3 листки в кожномо за параметром вік: менше за 18, від 18 до 60, більше 60. На другому рівні дерево поділить кожну групу за статтю. Таким чином маємо дворівневе дерево прийняття рішень.

2.2.2 Алгоритм поєднання декількох дерев на основі голосування

Цей алгоритм не є чимось екстраондинарним, але має змогу значно покращити результати пронозування. Через те що дерева прийняття рішень, як і всі інші алгоритми, схильні до проблем з змішенням та дисперсією(під зміщенням мається на увазі: наскільки в середньому прогнозовані значення відрізняються від фактичного значення, дисперсією: наскільки різними будуть прогнози моделі в одній точці, якщо брати різні вибірки з однієї сукупності), є сенс в поєднанні

великої кількості дерев прийняття рішень, що використовують різні гіперпараметри при тренуванні(максимальну глибину, мінімальну кількість спостережень в кожному листку). Результати, методом голосування вирішують до якого класу, або ж значення найбільше наближена вибірка даних.

Цікавим є саме голосування, виділяють дві основні методики у машинному навчанні:

- Жорстке голосування(Hard Voting) - відбір відбувається на основі моди(найбільш зустрічаємого значення у результатах моделей)
- М'яке голосування(Soft Voting) - відбір відбувається на основі усереднення ймовірностей результату

У регресійних задачах зазвичай використовується м'яке голосування, де замість ймовірностей беруть числові результати моделей.

2.2.3 Рандомний ліс дерев прийняття рішень

Рандомний ліс прийняття рішень у 1995 році винайшов науковець з IBM Т. К. Хо. Хо встановив, що ліси дерев, які розщеплюються косими гіперплощинами, можуть отримати точність у міру зростання, не страждаючи від перетренованості, якщо ліси випадковим чином обмежуються, щоб бути чутливими лише до вибраних розмірів ознак.

На відміну від моделі голосування, рандомний ліс дерев використовує більш просунений метод комбінації моделей, а саме Bagging. Bagging - комплексна техніка, яка використовується для зменшення дисперсії наших прогнозів шляхом поєднання результатів кількох класифікаторів, змодельованих на різних підвибірках одного набору даних.

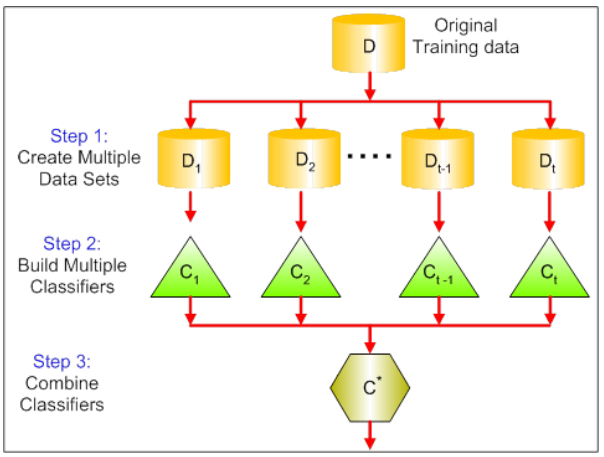

Рис 5. Bagging техніка

Різні стадії техніки bagging-у представлені на рисунку 5.

У подібній техніці результат залежить від кількості побудованих моделей, але якщо їх буде забагато, велика ймовірність зтикнутися з проблемою перенавчанн.

У науці про дані рандомний ліс дуже полюбляють і поважають через легку інтерпретабельність, можливість швидкої імплементації і зрозумілість гіперпараметрів. Так само як дерево, ліс має змогу обробляти як проблеми регресії так і класифікації, але все ще у випадку з регресією значення будуть у вигляді класі, тобто у подібних виборок будуть однакові спрогнозовані значення.

Також з позитивних сторін рандомного лісу можна виділити:

- вміє гарно обробляти великі виборки даних з великою розмірністю і після тренування можна вивести функцію що візуалізує які саме використані чинники найбільше вплипають на результат, тож в подальшому їх можна не використовувати, що значно полегшує роботу
- має вбудовані методи роботи з великою кількістю пропущених даних
- має вбудовані методи для роботи з незбалансованими даними

Наряду з гіперпараметрами унаслідуваними від дерева прийняття рішень тут ще можна виокримити наступні гіперпараметри:

- кількість дерев у моделі
- максимальна кількість використаних для побудування окремого дерева показників(стовпчикиків в вибірці)
- максимальна кількість використаних для побудування окремого дерева прикладів(рядочків в вибірці)

Єдиним великим недоліком при роботі з рандомним лісом є те, що модель формально є дуже незрозумілою зсередини, і якщо щось не виходить при тренуванні можна лише підібрати нові гіперпараметри і спробувати знову.

2.2.4 Алгоритм Adaboost

У часи коли комп’ютери розвинулися на стільки щоб справлятися з тисячами простих операцій за долі секунди з’явився новий підхід до побудування деревоподібних алгоритмів, а саме бустинг. Основна ідея бустингу полягає в

тому щоб перетворити “слабку модель” у “сильну”. Слабка гіпотеза або ж слабка модель визначається як та, чия продуктивність принаймні трохи краща, ніж випадковість. Покращення моделі полягає в ідеї фільтрації спостережень, залишаючи ті спостереження, з якими може впоратися слабка модель, і зосереджувати увагу на розробці нових “слабких навичок”, щоб справлятися з залишеними “важкими” спостереженнями.

Бустинговий алгоритм складається з 3 частин:

1. Функція втрат яку необхідно оптимізувати. Функція втрат зазвичай напряму залежить від вирішуємої задачі, однак вона обов’язково має бути дифиренційовною. У регресійних моделях це може бути середньоквадратичне відхилення, а у класифікаційних - логарифмічна функція втрат.

2. Слабка модель - у бустинговому алгоритмі використовют дерево прийняття рішень як слабку модель. Дерева будуються жадібним способом, що вибирає найкращі точки розбиття на основі показників чистоти таких як Джині(Gini - функція що показує наскільки гарно дерево було розбито)

$$Gini(t) = 1 - \sum_{i=1}^{j} P(i|t)^2$$

Формула функції Джині, де j - загальна кількість класів, P - коефіцієнт класу на і-тій ноді(листку)

Початково, як і в Adaboost використовувались тільки короткі дерева з 1 робиттям - що називались “пнем рішення”, але з часом так як комп’ютерні можливості збільшувалися почали використовувати й більш глибокі дерева.

3. Аддитивна модель - поступово додаються додаткові дерева, а початкові не змінюються, процедура градієнтного спуску використовується для мінімізації функції втрат при додаванні нових моделей.

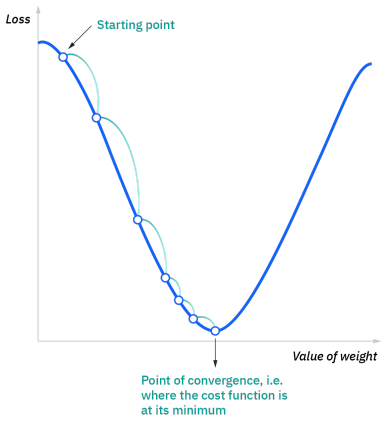

Рис. 6 Градієнтний спуск

Традиційно градієнтний спуск застосовується до параметрів, після чого розраховується втрата моделі, ваги оновлюються і помилка мінімізується. Тут все так само і як параметр виступають слабкі моделі, ми розраховуємо функцію втрат, на основі цього змінюємо параметри моделі і рухаємося в правильному напрямку мінімізуючи втрати. Після чого додаємо вихідні дані нового дерева до послідовності вже створених дерев задля покращення результату всієї моделі. Градієнтний бустинг проводиться доти, поки модель не досягає достатнього рівня, або ж не перестає покращуватися.

Першою реалізацією ідеї бустингу стала модель AdaBoost або ж Adaptive Boosting у 2003 році розроблена статистиками Й. Фрьойдом та Р. Шапіром, що виграли престижну нагороду Gödel Prize за свою роботу.

Процесс роботи AdaBoost можна роділити на наступні кроки:

1. Створюється слабка модель на основі збалансованих даних з однаковими вагами для кожної вибірки, як приклад, знизу представлена таблиця, що описує квартири в Києві, і треба спрогнозувати їх місцезнаходження:

кв. Метри	Район	Ціна в тис. $	Ваги
30	Голосіївський	55	1/4
45	Шевченківський	90	1/4
60	Дарницький	65	1/4

38	Шевченківський	75	1/4

2. Створюється пень рішення для кожної змінної, і проводиться аналіз наскільки добре кожен пень класифікує вибірки за цільовими класами.
3. Неправильно класифікованим зразкам призначається більша вага, щоб вони були правильно класифіковані в наступному пеньку рішення. Вага також призначається кожному класифікатору на основі точності класифікатора, що означає висока точність = велика вага!
4. Повторюємо кроки 2 та 3 поки всі точки не будуть правильно класифіковані, або не буде досягнуто максимальної кількості ітерацій

З математичної точки зору AdaBoost описано наступним чином:

1. Припустимо маємо наступні дані: $x_i \in R^n,\ y_i \in \{-1, 1\}$, - де: x_i - вибірка даних, n - кількість спостережень у наборі, y - результат(у цьому випадку бінарна класифікація)
2. Ми обчислюємо зважені вибірки для кожної точки даних. AdaBoost призначає вагу кожному навчальному прикладу, щоб визначити його значення в наборі навчальних даних. Спочатку всі точки даних матимуть однакову зважену вибірку w. $w = 1/N$

3. Після цього ми обчислюємо фактичний вплив цього класифікатора на класифікацію точок даних за формулою:

$$\alpha_t = \frac{1}{2} ln \frac{(1 - TotalError)}{TotalError}$$

Альфа - це те, наскільки цей пень матиме вплив на остаточну класифікацію. Total Error - це не що інше, як загальна кількість помилок для цього навчального набору, поділена на розмір навчального набору.

4. Після отримання фактичних значень Total Error для кожного пня настав час оновити ваги вибірки, які ми спочатку прийняли як 1/N для кожної точки даних. Ми зробимо це за такою формулою: $w_i = w_{i-1} * e^{+-a}$

Іншими словами, нова вага буде дорівнювати старій вазі, помноженому на число Ейлера, в степені плюс-мінус альфа (яке ми щойно обчислили).

AdaBoost є достатньо простим алгоритмом, і для його імплементації не треба налаштовувати велику кількість гіпер параметрів. Теоретично AdaBoost не схильний до перенавчання, хоча для цього немає конкретних доказів. Це може бути через те, що параметри не оптимізовані спільно — поетапна оцінка уповільнює процес навчання.

До недоліків AdaBoost можна віднести надзвичайну чутливість до зашумлених даних і викидів, тож дані мають бути якісними і без пропусків.

2.2.5 Алгоритм LightGBM

Алгоритм LightGBM або ж Light Gradient Boosting Machine розроблений у 2016 році науковцем з компанії Microsoft Гуоліном Ке, який в свою чергу при розробці зосереджувався на продуктивності та масштабованості своєї моделі. LightGBM формально наслідує всі основні принципи роботи AdaBoost, але з деякими покращеннями, тут використано новітню технологію Gradient-based One-Side Sampling (GOSS), цей метод, зменшує вибірки на основі градієнтів. Як ми знаємо, екземпляри з малими градієнтами добре тренуються (невелика помилка навчання), а екземпляри з великими градієнтами недостатньо навчені. Наївний підхід до зниження вибірки полягає в тому, щоб відкинути екземпляри з малими градієнтами, зосередившись виключно на екземплярах з великими градієнтами, але це змінило б розподіл даних. GOSS зберігає екземпляри з великими градієнтами, виконуючи випадкову вибірку для екземплярів з малими градієнтами.

Інтуїтивно зрозумілі кроки для розрахунку GOSS:

1. Відсортувати екземпляри за абсолютними градієнтами в порядку спадання
2. Вибрати верхні екземпляри a * 100%. (недостатньо навченині / великі градієнти)
3. Випадково вибрати b * 100% екземплярів з решти даних. Це зменшить внесок добре навчених прикладів на b ($b < 1$)
4. Без кроку 3 кількість зразків з малими градієнтами була б $1 - a$ (наразі це b). Щоб зберегти оригінальний розподіл, LightGBM посилює внесок вибірок з малими градієнтами на константу$(1 - a)/b$, щоб приділити більше уваги недостатньо навченим екземплярам. Це приділяє більше уваги недостатньо навченим екземплярам без значної зміни розподілу даних.

Ще у LightGBM використовується інший підхід до побудування дерев, замість росту в глибину, тут використано технологію росту по листкам. У глибиневій стратегії кожен вузол розбиває дані з пріоритетністю вузлів ближче до кореня дерев, коли стратегія на основі листків розвиває дерево, розбиваючи дані на вузли з найбільшою зміною у функції втрат. Такий підхід є достатньо небезпечним через перенавчання, але вбудовані функції регуляризації допомагають впоратися з даною проблемою
Також, у LightGBM є методи що гарно опрацьовують категоріальні дані, що спрощує пре-опрацювання вибірки і значно пришвидшує процес аналізу.
Так як LightGBM створений для прогнозування табличних даних, для аналізу часового ряду нам необхідно перетворити його у табличний формат з додаванням додаткових показників у вигляді лагів на основі реальних значень.
Лаги - це значення часового ряду у минулому з деяким кроком.
Оскільки модель може передбачити лише однокроковий прогноз, передбачене значення використовується для функції на наступному кроці, коли ми створюємо багатоетапне прогнозування, яке називається рекурсивним підходом для багатокрокового прогнозування.

2.3 Недолік деревоподібних алгоритмів при роботі з регресією

Як описано в минулому розділі деревоподібні алгоритми є дуже популярними і точними при роботі з табличними даними. Проте коли задача стосується регресії є великий недолік що стосується обмеження значень зверху у прогнозах, тобто при навчанні моделі, вона не зможе прогнозувати значення яких не було в тренувальній вибірці.

Що стосується часових рядів - алгоритм LightGBM не має можливості прогнозувати дані з трендом, тож для вирішення цієї проблеми є сенс додати просту модель лінійної регресії, що призначена саме для вирішення такої проблеми.

РОЗДІЛ 3: ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

Для проведення експерименту було використано мову програмування python - через величезну кількість бібліотек створених для аналізу даних.

Наразі однією з найпопулярніших галузей у сфері фінансів - є криптовалюта. І будь-яка криптовалюта, це часовий ряд через залежність ціни у часі. Тож порівняння моделей проведено на датасеті ціни криптовалюти Bitcoin з 01-12-2021 по 01-06-2022.

Для застосування моделі Sarima спочатку потрібно перевірити наш часовий ряд на стаціонарність. Задля цього у бібліотеці statsmodels є функція що застосовує Dickey-Fuller тест до нашого ряду і якщо p > 0.05 - ряд не є стаціонарним. Також можна побудувати 2 графіки ACF та PACF. ACF - відповідає за порядок ковзного середнього в моделі ARIMA, PACF - за порядок авторегресивної частини.

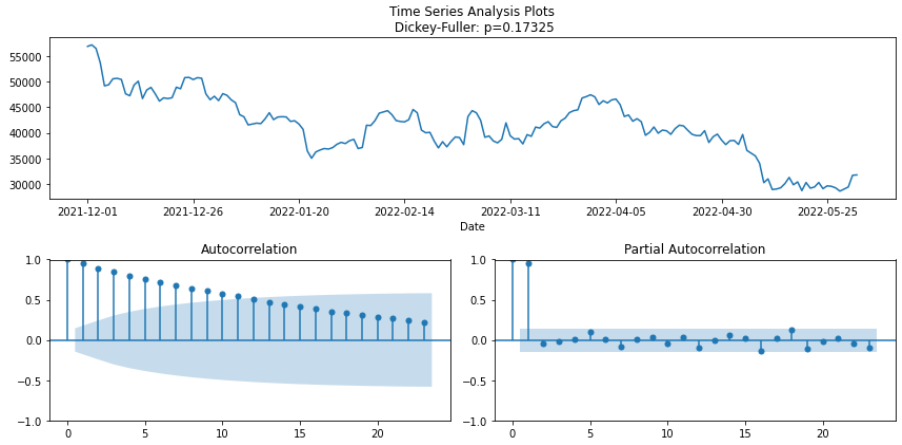

Рис. 7 Оригінальний часовий ряд ціни Bitcoin

Як бачимо з рисунку 7 значення p > 0.05, автокореляція плавно спадає, а частинна автокореляція має порядок 1. Тож для досягнення стаціонарності треба диференціювати часовий ряд з порядком 1, якщо б на графіку автокореляції ми бачили б піки, то порядок диференціювання дорівнював би порядковому номеру піку на графіку. Для диференціювання віднімемо від наявних значень часового ряду значення зміщені на 1(від сьогоднішньої ціни віднімаємо вчорашню). Після диференціювання застосуємо тест повторно.

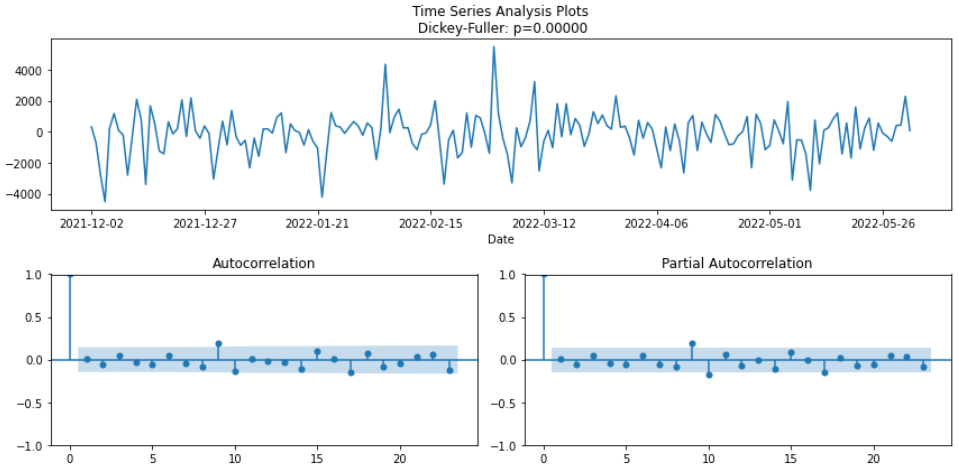

Рис. 8 Диференційований часовий ряд ціни Bitcoin

Як бачимо з рисунку 8, $p < 0.05$ - тож ряд є стаціонарним.

З бібліотеки sktime завантажимо модель AutoArima і передамо наступні параметри:

- start_p = 9 - порядок Авторегресії. Як можна побачити з графіку ACF(Autocorrelation) - порядковий номер останньої точки що не входить в проміжок часових лагів(синя смуга) - 9.
- max_p = 10 - start_p + порядок диференціювання.

Також поділимо датасет на тренувальну і тестувальну частину з коефіцієнтом 0.2(на 80% даних будемо навчати модел, на 20% даних валідувати отримані результати)

```
forecaster = AutoARIMA(start_p = 9, max_p = 10, suppress_warnings=True)
train.index = train.index.astype(int)
forecaster.fit(train['y'])
forecaster.summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	147
Model:	SARIMAX(9, 1, 2)	**Log Likelihood**	-1251.000
Date:	Sat, 04 Jun 2022	**AIC**	2526.000
Time:	12:23:58	**BIC**	2561.803
Sample:	0	**HQIC**	2540.548
	- 147		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0013	0.089	0.014	0.989	-0.173	0.175
ar.L2	-0.9931	0.076	-13.107	0.000	-1.142	-0.845
ar.L3	-0.0414	0.104	-0.398	0.691	-0.245	0.163
ar.L4	-0.1560	0.120	-1.301	0.193	-0.391	0.079
ar.L5	-0.0639	0.094	-0.681	0.496	-0.248	0.120
ar.L6	-0.0914	0.132	-0.694	0.487	-0.349	0.167
ar.L7	-0.0221	0.117	-0.189	0.850	-0.251	0.207
ar.L8	0.0136	0.084	0.161	0.872	-0.152	0.179
ar.L9	-0.0470	0.093	-0.508	0.611	-0.228	0.134
ma.L1	0.0099	0.037	0.266	0.790	-0.063	0.083
ma.L2	0.9764	0.052	18.719	0.000	0.874	1.079
sigma2	1.717e+06	3.79e-08	4.53e+13	0.000	1.72e+06	1.72e+06

Ljung-Box (L1) (Q):	0.06	**Jarque-Bera (JB):**	80.65
Prob(Q):	0.80	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.21	**Skew:**	0.60
Prob(H) (two-sided):	0.51	**Kurtosis:**	6.44

Рис. 9 Модель SARIMA

З рисунку 9 видно - найкращі параметри для моделі сезонної авторегресії диференційовного ковзного седенього - AR = 9, D = 1, MA = 2.

Застосуємо натреновану модель до тестового часового проміжку і перевіримо результат.

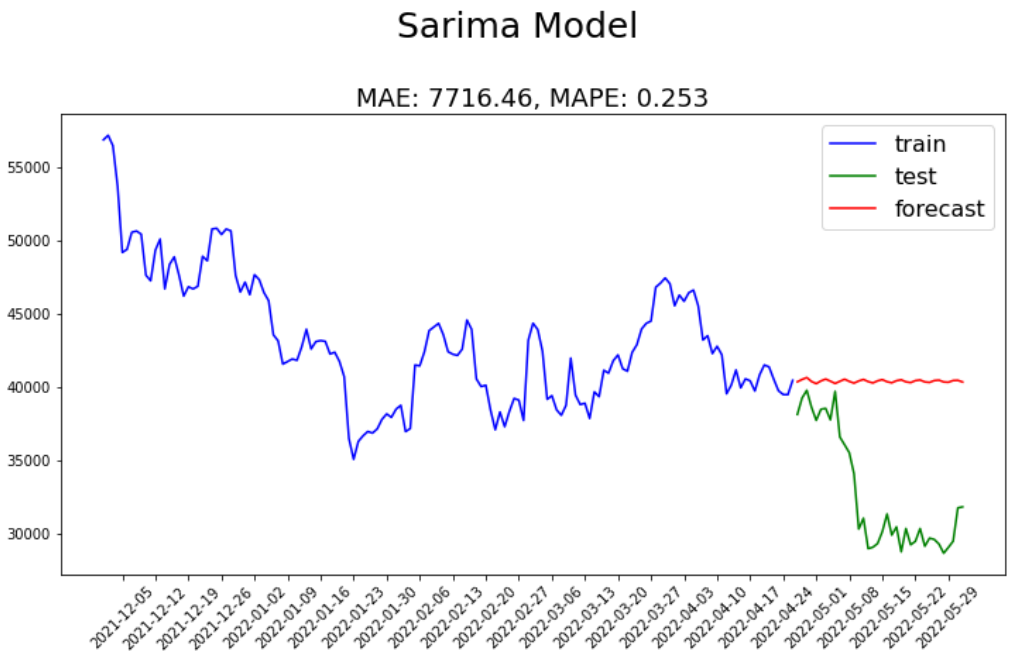

Рис. 10 Результати прогнозування ціни на Bitcoin за допомогою моделі SARIMA

Середня абсолютна помилка(MAE) складає 7716.46$, і середня абсолютна відсоткова помилка(MAPE) складає 25.3%.

Наступним кроком спрогнозуємо ціну Bitcoin за допомогою моделі LightGBM. Ця модель працює виключно з табличним формтом даних і може прогнозувати лише на 1 крок вперед, тож для її застосування необхідно перетворити данні у табличний формат, створити часові лаги(додаткові значення часового ряду на основі даних з минулого, з зазначеним кроком) і використати рекурсивний підхід прогнозування часових рядів(використовувати вже зпрогнозоване значення задля пронозування наступного). Для цього в бібліотеці sktime є методи ExpandingWindowSplitter, ForecastingGridSearchCV. ExpandingWindowSplitter - випадково бере частину тренувального датасету для навчання. ForecastingGridSearchCV - навчає модель з різними параметрами на основі даних з ExpandingWindowSplitter, і вибирає найкращу. Також створимо декілька допоміжних методів create_forecaster(window_length=5)та grid_serch_forecaster(train, test, forecaster, param_grid).

У методі create_forecaster(window_length=5) ініціалізуємо модель Light GBM, перетворимо датасет у табличний формат і створимо нову колонку даних з лагованими значеннями(за замовчуванням дані з 5 проміжків часу у минулому)

```python
def create_forecaster(window_length=5):

    # creating forecaster with LightGBM
    regressor = lgb.LGBMRegressor()
    forecaster = make_reduction(regressor, window_length=window_length, strategy="recursive")

    return forecaster
```

Рис. 11 Метод ініціалізації моделі Light Gradient Boosting Machine

У методі grid_serch_forecaster(train, test, forecaster, param_grid)ініціалізуємо ExpandingWindowSplitter та ForecastingGridSearchCV,натренуємо модель на тестових даних, виберемо найкращі параметри для моделі Light GBM, та застосуємо модель на тестових даних.

```python
def grid_serch_forecaster(train, test, forecaster, param_grid):

    # Grid search on window_length
    cv = ExpandingWindowSplitter(initial_window=int(len(train) * 0.7))
    gscv = ForecastingGridSearchCV(
        forecaster, strategy="refit", cv=cv, param_grid=param_grid
    )
    gscv.fit(train['y'])
    print(f"best params: {gscv.best_params_}")

    # forecasting
    fh=np.arange(len(test))+1
    y_pred = gscv.predict(fh=fh)
    mae, mape = plot_forecast(train.set_index('Date'), test, y_pred, 'Light GBM Model')

    return gscv.best_params_, mae, mape
```

Рис. 12 Метод підбору параметрів та прогнозування задопомогою моделі Light Gradient Boosting Machine

Викличемо методи з наступними параметрами param_grid = {"window_length": [7, 14, 21, 28, 35, 42, 49]} - кількість лагованих періодів у часовому ряді.

Найкращі параметри: {'window_length': 21}

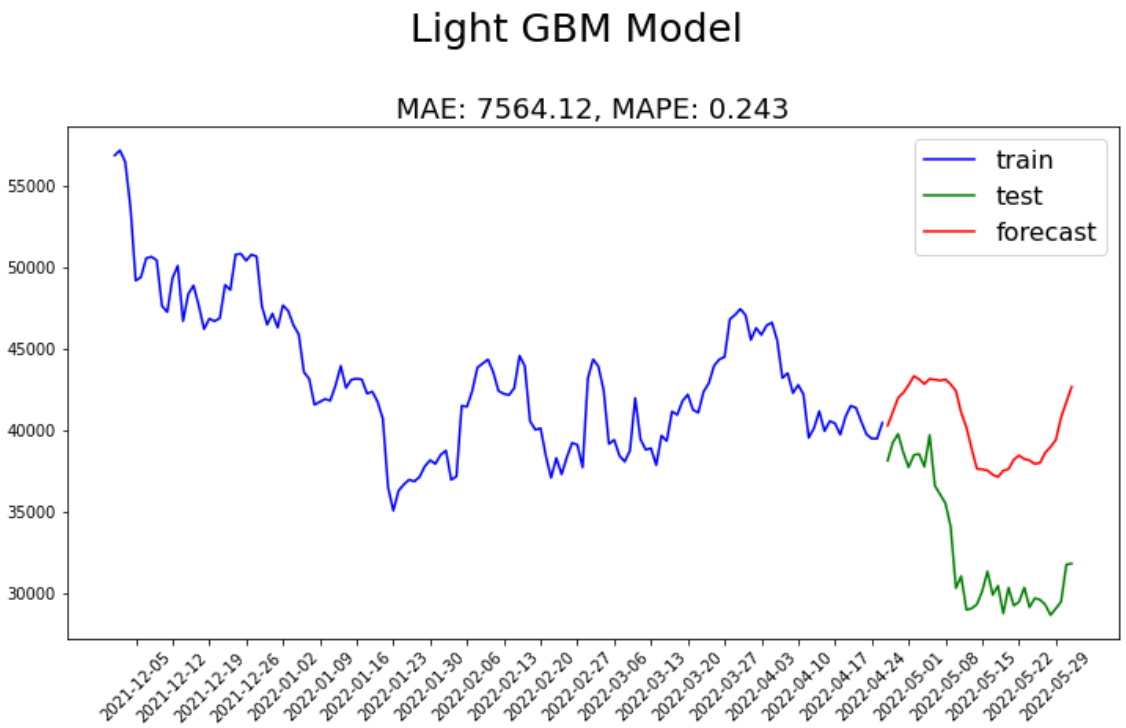

Рис. 13 Результати прогнозування ціни на Bitcoin за допомогою моделі Light Gradient Boosting Machine

З рисунку 13, середня абсолютна помилка(MAE) складає 7564.12$, і середня абсолютна відсоткова помилка(MAPE) складає 24.3%. Маємо покращення в середньому на 1 відсоток в порівнянні моделі Arima.

Але як було зазначено вище модель LGBM не має можливості прогнозувати дані з трендовою складовою, тож додамо ще один метод lgb_and_linreg_forecaster(lgb_model_params, data), що використовує найкращу модель Light GBM, тренує модель лінійної регресії з бібліотеки sklearn і усереднює результати обох моделей.

```python
def lgb_and_linreg_forecaster(lgb_model_params, data):
  data = data.reset_index()
  test_len = int(len(data) * 0.2)
  train, test = data.iloc[:-test_len], data.iloc[-test_len:]
  train_linear, test_linear = train.copy(), test.copy()
  train_linear['time'] = np.arange(len(train.index))
  X = train_linear.loc[:, ['time']]  # features
  y = train_linear.loc[:, 'y']  # target

  linreg_model = LinearRegression()
  linreg_model.fit(X, y)

  y_pred_linear = pd.Series(linreg_model.predict(\
                          test_linear.reset_index().drop(['Date', 'y'],
                          axis = 1)), index=test_linear.index)

  lgb_model = create_forecaster(lgb_model_params['window_length'])

  fh=np.arange(len(test))+1
  lgb_model.fit(train['y'])
  y_pred_lgb = lgb_model.predict(fh=fh)

  y_pred = (y_pred_linear + y_pred_lgb) / 2
  mae_ans, mape_ans = plot_forecast(train.set_index('Date'),
                                    test.set_index('Date'),
                                    y_pred,
                                    'Light GBM and Linear Regression Models')

  return mae_ans, mape_ans
```

Рис. 14 Метод підбору параметрів та прогнозування задопомогою моделі Light Gradient Boosting Machine у поєднанні з моделлю лінійної регресії

Після виклику останнього методу, маємо значне покращення у обох метриках.

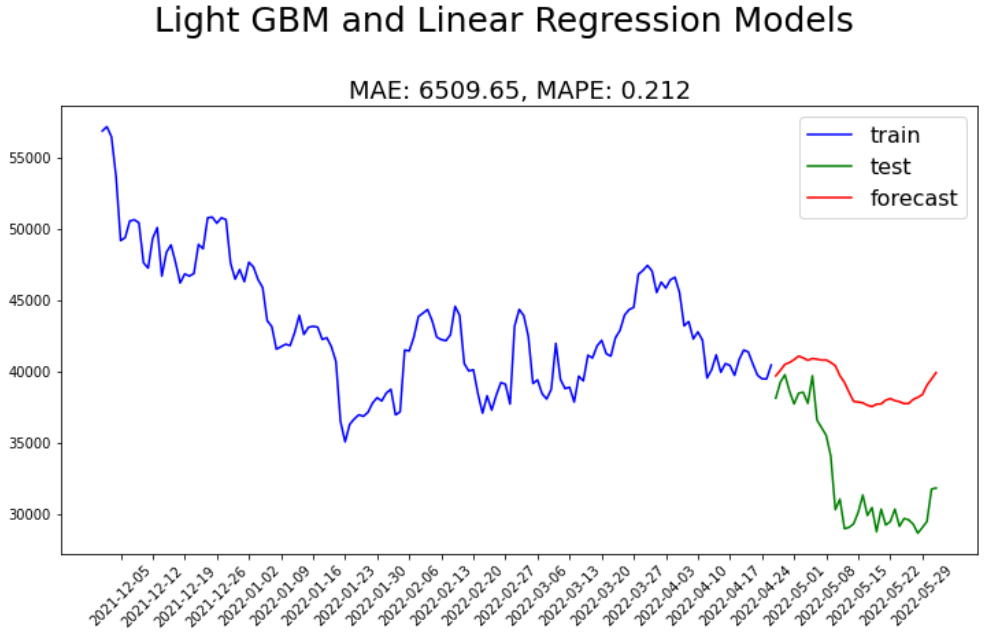

Рис. 15 Результати прогнозування ціни на Bitcoin за допомогою моделі Light Gradient Boosting Machine у поєднанні з моделлю лінійної регресії

У результаті порівняємо метрики кожного з представлених підходів:

	Model	Mean Absolute Error	Mean Absolute Percentage Error
0	Sarima	7716.457645	0.253268
1	LightGBM	7564.116979	0.242513
2	**Lightgbm & Linear Regression**	**6509.654067**	**0.212001**

Рис. 16 Результати прогнозування ціни на Bitcoin за допомогою різних алгоритмів

Як бачимо з рисунку 16, по обом метрикам виграє модель з використанням Light GBM у поєднанні з Linear Regression.

Висновок

Під час роботи було проведено аналіз і прогнозування часового ряду за допомогою декількох алгоритмів, тонкощі роботи з різними підходами, їх імлементації та оцінювання результатів.

Показано підхід до пошуку стаціонарності ряду, та підбору параметрів для застосування класичних методів.

Наглядно представлено переваги і недоліки кожного алгоритму.

Список джерел

1. Aurelien Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2nd Edition, 2019
2. Hyndman, R.J., & Athanasopoulos, G. *Forecasting: principles and practice*, 2nd edition, 2018
3. Tomonori Masui, Multi-step Time Series Forecasting with ARIMA, LightGBM, and Prophet, 2021
4. Galit Shmueli , Kenneth C. Lichtendahl Jr, Practical Time Series Forecasting with R: A Hands-On Guide [2nd Edition] (Practical Analytics), 2016
5. George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, Greta M. Ljung, Time Series Analysis: Forecasting and Control, 1970
6. Douglas C. Montgomery, Cheryl L. Jennings, Murat Kulahci, Introduction to Time Series Analysis and Forecasting, 2nd Edition, 2016