НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Моделі машинного навчання в задачах аналізу здатності технічних індикаторів прогнозувати ціни криптовалют»

Виконав:
студент IV курсу, групи КА-11
Бедлінський Кірілл Ігорович _____

Керівник:
завідувач кафедри ММСА, к.т.н., доцент,
Тимощук Оксана Леонідівна _____

Консультант з економічного розділу:
доцент, к.е.н, Рощина Надія Василівна _____

Консультант з нормконтролю:
к. ф.-м. н., Статкевич Віталій Михайлович _____

Рецензент:
д.т.н., доцент ІТГІП НАН України,
Терентьєв Олександр Миколайович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Бедлінському Кіріллу Ігоровичу

1. Тема роботи «Моделі машинного навчання в задачах аналізу здатності технічних індикаторів прогнозувати ціни криптовалют», керівник роботи Тимощук Оксана Леонідівна, завідувач кафедри ММСА, к.т.н, доцент, затверджені наказом по університету від «_26_»___05____ 2025 р. №1759-с.

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи: ринкові дані криптовалюти біткоїн з платформи Yahoo Finance.

4. Зміст роботи: Розділ 1 Дослідження предметної області. Розділ 2 Теоретичні основи алгоритмів машинного навчання, технічних індикаторів та аналізу їх прогностичної здатності. Розділ 3 Програмна реалізація та аналіз результатів. Розділ 4 Функціонально-вартісний аналіз.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): ілюстрації результатів створення навчальної вибірки даних, ілюстрації результатів очищення навчальної вибірки, ілюстрації роботи моделей машинного навчання, ілюстрації графіків метрик оцінювання, ілюстрації графіка прибутковості.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Функціонально-вартісний аналіз	Рощина Н. В., доцент, к.е.н.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання бакалаврської дипломної роботи (БДР)	Термін виконання етапів роботи	Примітка
1	Затвердження теми дипломної роботи. Ознайомлення зі структурою БДР та державним стандартом ДСТУ 3008:2015.	14.04.2025	Виконано
2	Опрацювання літератури за темою. Визначення методології дослідження. Формулювання задачі дослідження	21.04.2025	Виконано
3	Робота над теоретичною частиною БДР.	28.04.2025	Виконано
4	Реалізація практичної частини БДР.	05.05.2025	Виконано
5	Аналіз результатів дослідження.	12.05.2025	Виконано
6	Виконання функціонально-вартісного аналізу	19.05.2025	Виконано
7	Оформлення пояснювальної записки	02.06.2025	Виконано
8	Оформлення презентації для демонстрації	16.06.2025	Виконано

Студент Кірілл БЕДЛІНСЬКИЙ

Керівник Оксана ТИМОЩУК

РЕФЕРАТ

Дипломна робота: 149 с., 45 рис., 22 табл., 2 дод., 28 джерел.

МАШИННЕ НАВЧАННЯ, ПРОГНОСТИЧНА ЗДАТНІСТЬ, ТЕХНІЧНІ ІНДИКАТОРИ, РИНОК КРИПТОВАЛЮТ, ЗАДАЧІ КЛАСИФІКАЦІЇ.

Об’єкт дослідження – процеси прогнозування зміни тренду ціни криптовалюти.

Предмет дослідження – моделі машинного навчання в аналізі предикативної здатності технічних індикаторів.

Мета роботи – розробка та експериментальна оцінка ефективності моделей машинного навчання для прогнозування динаміки цін криптовалют із використанням набору технічних індикаторів.

Результат роботи – є розроблене програмне рішення, яке виконує тренування моделі машинного навчання під технічний індикатор та яке показує ефективність обраного індикатору. Проведено порівняння результатів, отриманих різними розглянутими підходами.

Програмне рішення було розроблено у середовищі Jupiter Notebook із використанням мови програмування Python. В роботі було використано ринкові дані з платформи Yahoo Finance.

ABSTRACT

Thesis: 149 p., 45 fig., 22 tables, 2 appendices, 28 references.

MACHINE LEARNING, PREDICTIVE POWER, TECHNICAL INDICATORS, CRYPTOCURRENCY MARKET, CLASSIFICATION TASKS.

Object of the study – the processes of forecasting cryptocurrency price trend changes.

Subject of the study – machine learning models used to analyze the predictive power of technical indicators.

Purpose of the study – to develop and experimentally evaluate the effectiveness of machine learning models for forecasting cryptocurrency price dynamics using a set of technical indicators.

Result of the study – a software solution was developed that trains a machine learning model based on a specific technical indicator, this solution also demonstrates the indicator's predictive effectiveness. A comparative analysis of the results obtained using different approaches was conducted.

The software solution was developed in the Jupiter Notebook environment using the Python programming language. The study utilized market data retrieved from the Yahoo Finance platform.

ЗМІСТ

ВСТУП

За останнє десятиліття ринок криптовалют значно зріс і перетворився з малопомітного явища в галузь, яка є однією з найпопулярніших та яка найбільш розвивається у світі. Станом на травень 2025 року сягає майже 3 трильйона доларів США. Об'єми ринку можна порівняти з фондовими біржами деяких країн або з ринком цінних металів.

Однією з ключових рис криптовалютного ринку є висока волатильність. Це створює як додаткові можливості для інвесторів, так і додаткові ризики. Тому потреба в надійних інструментах аналізу ринку є суттєвою. Одним з варіантів аналізувати рух ринку є технічні індикатори. Гравці ринку вже давно використовують їх, так як технічні індикатори є простими і не потребують багато ресурсів для обчислення.

Проте торгівля без використання сучасних засобів сильно гальмує увесь процес, що може спотворювати справжні можливості стратегії. Історично аналізувати дані ринку у реальному часі без участі експерта було неможливим. З розвитком машинного навчання та ростом обчислювальних потужностей, для інвесторів відкрились нові можливості та нові інструменти.

Метою цієї роботи є розробка та експериментальна оцінка ефективності моделей машинного навчання для прогнозування динаміки цін криптовалют із використанням набору технічних індикаторів. У роботі буде здійснено збір даних, їх обробка та формування індикаторів а також навчання різних моделей машинного навчання для прогнозування руху цін криптовалют. Серед розглянутих моделей будуть такі алгоритми як Random Forest, XGBoost а також метод опорних векторів.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

За останнє десятиліття ринок криптовалют значно зріс і перетворився з малопомітного явища в галузь, яка є однією з найпопулярніших та яка найбільш розвивається у світі. Станом на травень 2025 року сягає майже 3 трильйона доларів США. Об'єми ринку можна порівняти з фондовими біржами деяких країн або з ринком цінних металів.

Однією з ключових рис криптовалютного ринку є висока волатильність. Щоденні коливання цін криптовалют з найбільшою капіталізацією можуть перевищувати 5-10%, а іноді становлять й двозначні значення. Це створює як додаткові можливості для інвесторів, так і додаткові ризики. Тому потреба в надійних інструментах аналізу ринку є суттєвою.

Одним з варіантів аналізувати рух ринку є технічні індикатори. Гравці ринку вже давно використовують їх, так як технічні індикатори є простими і не потребують багато ресурсів для обчислення. Тим не менш, технічні індикатори мають і мінуси. Вони ігнорують нелінійні рухи ринку, часто видають неправильні сигнали, особливо в умовах, коли ринок не має вираженого тренду, деякі з них є досить спізнілими особливо в умовах активної динаміки ринку, тощо. Також деякі з них мають між собою високу кореляцію, що може негативно впливати на торгівельні стратегії.

Проте торгівля без використання сучасних засобів сильно гальмує увесь процес, що може спотворювати справжні можливості стратегії. Історично аналізувати дані ринку у реальному часі без участі експерта було неможливим. З розвитком машинного навчання та ростом обчислювальних потужностей, для інвесторів відкрились нові можливості та нові інструменти. Проте застосування методів машинного навчання на значеннях цін є досить

неефективним. Тому технічні індикаторі відкривають нові варіанти більш ефективного застосування методів машинного навчання для побудування більш прибуткових торгівельних стратегій. Поєднання машинного навчання та технічних індикаторів може виправити недоліки цих концепцій.

Саме тому правильно обрати кращі технічні індикатори та найбільш ефективні моделі машинного навчання є дуже важливою задачею у побудові торгівельної стратегії з використанням цих інструментів.

1.2 Історичні та сучасні методи аналізу ринкових індикаторів моделями машинного навчання

1.2.1 Історія використання машинного навчання для аналізу ринкових індикаторів

Історія розвитку машинного навчання як галузі почалось у середині 20 століття. У 1956 році на конференції Dartmouth Summer Research Project on Artificial Intelligence було офіційно визнано цю галузь та вже у наступному десятилітті почались з’являтися експертні системи, які були першими спробами автоматизації аналізу ринку [1]. У 1990 році Т. Кімото, К. Асакава, М. Йода та М. Такеока вперше застосували багатошаровий персептрон для прогнозування індексу Nikkei [2], а вже в 1992 році на основі вже існуючого методу опорних векторів, Б. Босер, І.Гійон та В.Вапник розробили спосіб створення нелінійних классифікаторів за допомогою ядрового трюку [3]. Цей метод потім стали використовувати для класифікації тренду ринку за допомогою технічних індикаторів.

У 1997 році Ю. Шмідхубер та С. Хохрейтер запропонували алгоритм довгої короткочасної пам’яті (LSTM), як архітектуру рекурентних нейронних

мереж. У своїй роботі вони розробили механізм постійного потоку помилки, що дозволило краще виявляти залежності довжиною понад тисячу кроків у часі. Також у 1999 році Ф. Ґерс, Ю. Шмідхубер та Ф. Каммінс додали до архітектури забувальний гейт який дозволяє керувати об’ємом пам’яті, що дуже покращило стабільність в умовах безперервного потоку нових даних, що характерно для умов ринку.

У 1999 році Тін Кам Хо запропонував метод випадкового підпростору який згодом ліг в основу ансамблевого алгоритму випадкових лісів. Цей метод базується на принципі навчання декількох класифікаторів на різних підмножинах ознак та об'єднанні прогнозів цих класифікаторів [4]. З середини 2000 року ETF та хедж-фонди почали впровадження ансамблевих алгоритмів в аналіз технічних індикаторів. Це одразу дало свої результати – алгоритм добре себе показав при комбінуванні декількох індикаторів та дуже знизив кількість неправильних сигналів. Проте такі моделі потребують значно більшої кількості ресурсів, порівняно з одиночними та часто перенавчаються.

У 2014 році Тянци Чен разом розробив метод екстремального градієнтного бустингу (XGBoost). Цей метод є оптимізованою версією градієнтного бустингу, який також використовує регуляризацію, що допомагає запобігти перенавчанню. Також серед переваг цього методу можна виділити паралельну обробку, що значно пришвидшує тренування, що було значною проблемою алгоритму випадкового лісу до цього, особливо на великих об’ємах даних, що часто зустрічається у ринкових даних.

З 2017 року по сьогоднішній день відбувається активний розвиток нових методів машинного навчання – трансформерів. Трансформери – це архітектура глибоких нейронних мереж, яка була вперше представлена у 2017 року науковцями компанії гугл. Її особливість полягає в тому, що вони базуються на механізмі уваги, який дозволяє їм визначати та фокусуватись на біль важливих частинах вхідних даних [5]. Наразі транфсормери здебільшого використовуються для поєднання технічних індикаторів з альтернативними

даними, такими як наприклад новини, або дані з соціальних мереж, що допомагає їм розуміти увесь контекст ринку.

1.2.2 Сучасні практики використання машинного навчання серед гравців ринку

Сьогодні майже усі гравці ринку використовують машинне навчання для збільшення свого прибутку. Наприклад, великі інвестиційні банки застосовують машинне навчання для підвищення ефективності своїх операцій та поступового нарощення своєї присутності на криптовалютного ринку. Так, наприклад, одні з найбільших у світі банків JPMorgan Chase, Wells Fargo та Deutsche Bank активно розвивають свою інфраструктуру для торгівлі криптовалютними активами та вже пропонують своїм клієнтам деякі можливості для прибутку від цифрових активів.

Хедж фонди, які є провідними гравцями, активно впроваджують глибоке машинне навчання для аналізу поточного стану ринку та торгівельних агентів, основаних на різних видах машинного навчання.

Американська інвестиційна компанія Bridgewater у 2024 році запустила фонд загальною вартістю 2 мільярди доларів США, в якому алгоритми машинного навчання є ядром ухвалення рішень. Після запуску в 2023 році лабораторії AIA, компанія почала розробку стратегії на основі великих мовних моделей та інших моделей машинного навчання, що допомогло збільшити свої прибутки.

Хедж фонд Point 72 також активно впроваджує машинне навчання в свої торгівельні стратегії. Компанія використовує здебільшого ансамблеві алгоритми, наприклад алгоритм випадкових лісів, екстремальний градієнтний бустинг та експериментує з LSTM. Також компанія активно застосовує трансформери, для аналізу таких альтернативних даних як новини,

погодні дані та макроекономічні показники. Це допомагає вчасно виявляти зміни в настрої ринку та адаптувати свої стратегії до поточних умов. Завдяки цьому компанія отримує конкурентоспроможні прибутки, що робить її одним з лідерів систематичних інвестицій у цю галузь.

Компанія DRW Trading Group є однією з найбільших компаній у світі, що торгують активами і також активно розвиває напрямок машинного навчання. Вона активно використовує такі алгоритми машинного навчання як випадкові ліси, конволюційні нейронні мережі та екстремальний градієнтний бустинг для прогнозування цін активів і створення торгівельних стратегій.

В умовах високої конкуренції використання моделей машинного навчання для аналізу ринкових показників різних активів є досить важливим для існування багатьох компаній в цьому секторі. Саме тому майже усі компанії зараз активно інвестують в цю галузь.

1.3 Проблеми та обмеження використання моделей машинного навчання в аналізі ринкових індикаторів

При аналізі будь яких ринків варто враховувати що вони включають у себе багато різних умов, які є необхідними при спробах передбачення. Цифрові активи є відносно новим різновидом активів, регуляторні державні механізми ще не сформовані і тому їх непередбачуваність є набагато більшою ніж у інших.

У побудуванні моделей машинного навчання в аналізі ринкових індикаторів також виникають і проблеми.

1. Якість та кількість даних. Результати аналізу потребують наявності повних даних, які відповідають справжній ринковій ситуації. Такі дані як ціна та об’єми торгівлі досить легко знайти у відкритому доступі,

проте при побудуванні індикаторів часто використовуються дані, що не відносяться напряму до досліджуваного активу, наприклад індекс страху та жадібності, тренди по пошуковим запитам, кількість активних користувачів, кількість відкритих контрактів, відношення ринкової капіталізації до реалізованої тощо. Також при торгівлі новими криптовалютними активами виникає проблема відсутності достатньої кількості історичних даних для проведення аналізу. Також, хоча багато сервісів пропонують безкоштовний доступ до історичних даних, їх якість не завжди є високою, тобто вони можуть містити пропуски, неправильні дані та зміщені дані.

2. Проблема перенавчання моделей машинного навчання. При тренуванні моделей машинного навчання часто виникає перенавчання, коли модель настільки точно відтворює випадкові коливання та шум у даних, що на реальних даних вона може давати зовсім погані результати. Ця проблема особливо притаманна більш складним моделям, як ансамблі та глибокі нейронні мережі. Через дисбаланс між репрезентативними даними та усіма іншими, для навчання модель часто має обмежу кількість прикладів, тому в цьому випадку часто виникає проблема перенавченості.
3. Непередбачуваність цін цифрових активів. Так як ринок цифрових активів – відносно новий, то регуляторні органи ще не сформувались. Через це коливання цін на криптовалюти можуть бути хаотичними та непередбачуваними, що створює шум для моделей машинного навчання. Модель може вибудувати неправильні зв’язки на основі цих коливань і мати низьку результативність.

4. Зовнішні фактори. Цифрові активи через свою природу є дуже залежними від соціального сприйняття, тому їх ціна може реагувати на деякі зовнішні фактори, такі як новини, повідомлення від відомих людей, регуляторна політика держав та шахрайство з використанням

цих активів. При аналізі технічних індикаторів неможливо врахувати ці фактори, що робить деякі коливання незрозумілими для інтерпретації моделей машинного навчання.

Висновки до розділу 1

У цьому розділі було розглянуто актуальність та важливість використання моделей машинного навчання для торгівлі цифровими активами та розглянуто історію створення різних методів машинного навчання та проаналізовано нинішній стан використання їх у сфері інвестицій та торгівлі. Також було розглянуто приклади компаній, що вже використовують їх у своїй роботі, що ще раз підкреслює важливість їх дослідження для більш ефективної торгівлі.

Також було перелічено проблеми та обмеження моделей машинного навчання при аналізі технічних індикаторів. Серед перелічених проблем є як і ті, що можна вирішити шляхом підбору правильного методу та оптимізації параметрів моделі, так і ті, що неможливо вирішити у зв’язку з неможливістю отримання точних даних.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ, ТЕХНІЧНИХ ІНДИКАТОРІВ ТА АНАЛІЗУ ЇХ ПРОГНОСТИЧНОЇ ЗДАТНОСТІ

2.1 Теоретичні основи індикаторів технічного аналізу

Технічні індикатори – це математичні розрахунки на основі ринкових даних (ціни, обсягу тощо), які допомагають гравцям ринку аналізувати ринковий стан і визначати моменти для угод. Існують сотні індикаторів – від простих ковзних середніх до складних осциляторів – кожен з яких виконує свою функцію. Деякі вимірюють тренд, інші – імпульс, ще інші – волатильність чи обсяги торгів. Загалом технічні індикатори прийнято поділяти на трендові індикатори, індикатори імпульсу (осцилятори), індикатори волатильності та індикатори об’єму [6].

Трендові індикатори відслідковують напрямок та зміни тренду, тобто довгострокового руху ціни. Як правило, це засоби згладжування цінового ряду, що дозволяють виявити домінуючий напрям. Як приклад можна навести різні ковзні середні (Simple Moving Average, Exponential MA тощо) та індикатори на їх основі, наприклад MACD. Трендові індикатори допомагають визначити фазу ринку, зростаючий чи спадаючий тренд і реагують із певним запізненням, підтверджуючи поточну тенденцію [6].

Індикатори імпульсу (осцилятори) коливаються в заданому діапазоні та відображають імпульс руху ціни чи ступінь перекупленості або перепроданості актива. Вони порівнюють поточну ціну з її діапазоном за певний період, сигналізуючи про можливе послаблення тренду чи розворот. Яскравими прикладами є індекс відносної сили (RSI), стохастичний осцилятор тощо. Значення осциляторів зазвичай нормалізовані, що спрощує

інтерпретацію: високі значення вказують на перекупленість, низькі – на перепроданість актива [6].

Індикатори волатильності вимірюють мінливість цін та діапазон коливань ринку. Такі показники відображають, наскільки швидко і як сильно змінюється ціна. Наприклад, смуги Болліндже́ра (Bollinger Bands), що поєднують змінне середнє та стандартне відхилення ціни: відстань між верхньою і нижньою смугою розширюється при зростанні волатильності та звужується при її падінні. Індикатори волатильності дозволяють оцінити ризикованість ринку та можливі точки прориву діапазонів [7].

Індикатори об’єму ґрунтуються на обсягах торгів і показують інтенсивність торгівлі для підтвердження рухів ціни. Вони допомагають зрозуміти, наскільки сильним є тиск покупців чи продавців під час цінових змін. Наприклад індикатор On-Balance Volume (OBV), індекс грошового потоку (MFI) тощо. Індикатори об’єму часто слугують фільтром: наприклад, зростання OBV підтверджує зростаючий рух ціни, а падіння OBV при зростанні ціни може попереджати про слабкість тренду.

Розглянемо детальніше обрані для аналізу індикатори.

1. Смуги Болліндже́ра – це популярний індикатор волатильності, розроблений Джоном Болліндже́ром. Він складається з трьох смуг: середньої, верхньої та нижньої. Середня смуга зазвичай є простою ковзною середньою ціни за певний період. Верхня та нижня смуги відстають від середньої на певну задану кількість стандартних відхилень ціни (як правило, два стандартних відхилення) [7]. Формально центральна смуга визначається наступним чином:

$$M_t = SMA_n(P_t),$$

де SMA – проста ковзна середня ціни P за n періодів. Нижня смуга визначається як:

$$L_t = M_t - k * \sigma_t,$$

де σ_t– стандартне відхилення ціни за той самий період, k – коефіцієнт відступу. Аналогічним чином визначається і верхня смуга:

$$U_t = M_t + k * \sigma_t,$$

де σ_t– стандартне відхилення ціни за той самий період, k – коефіцієнт відступу.

Смуги Болліндже́ра формують діапазон, в межах якого коливається ціна; ширина цього діапазону динамічно відображає волатильність ринку. Якщо ринкова волатильність зростає, стандартне відхилення збільшується і відстань між смугами розширюється; при зниженні волатильності смуги зближуються [8]. З точки зору інтерпретації, коли ціна підходить до верхньої смуги або прориває її, можливий сигнал про перекупленість або скору корекцію вниз. Навпаки, вихід ціни до нижньої смуги може сигналізувати про перепроданість та потенційний розворот вгору. Важливо зазначити, що смуги Болліндже́ра самі по собі не прогнозують напрям руху, а вказують на рівні аномальної волатильності [8].

2. Стохастичний осцилятор – класичний індикатор імпульсу, який був розроблений Джорджем Лейном у 1950-х роках. Він показує положення поточної ціни відносно її мінімуму та максимуму за вибраний період і таким чином оцінює, чи не наближається ціна до крайніх значень свого діапазону. Основна ідея полягає в тому, що при висхідному тренді ціни закриття мають тенденцію наближатися до верхньої межі недавнього діапазону, а при низхідному – до нижньої . Стохастичний осцилятор складається з двох ліній: %K – швидкої стохастичної лінії, та %D – повільної (сигнальної) лінії, яка є згладженим середнім %K [9]. Формалізовано лінія %K визначається наступним чином:

$$\%K = \frac{P_{close} - \min(P_n)}{\max(P_n) - \min(P_n)},$$

де P_{close} – поточна ціна закриття свічки, $\min(P_n)$ – мінімум ціни за останні n періодів, $\max(P_n)$ – максимум ціни за останні n періодів. Лінія %D визначається як:

$$\%D = \frac{\%K_1 + \%K_2 + \%K_3 + \ldots + \%n}{n},$$

де %K = швидка стохастична лінія, n – вікно періодів. Стохастичний осцилятор набуває значень від 0 до 100. Високе значення означає, що ціна знаходиться близько до верхнього екстремуму періоду (сильний імпульс вгору), тоді як низьке – близько до нижнього екстремуму (сильний імпульс вниз). Традиційно рівень %K > 80% розглядається як зона перекупленості ринку, а %K < 20% – як зона перепроданості. Перетин швидкої лінії %K вниз через рівень 80 може слугувати сигналом до продажу (ослаблення висхідного імпульсу), а перетин вгору через рівень 20 – сигналом до купівлі (закінчення спадного імпульсу). Також аналізують перетини ліній: коли %K перетинає %D зверху вниз, це може вказувати на початок зниження (сигнал "продавати"), а перетин знизу вгору – на можливий розворот вгору ("купувати") [9].

Стохастичний осцилятор описують як індикатор, що функціонує на основі рівнів підтримки та опору, вимірюючи поточну ціну відносно діапазону за період. Його основна мета – прогнозувати розвороти тренду шляхом виявлення моментів, коли ціна досягла крайньої верхньої або нижньої межі свого недавнього діапазону.

3. Параболічний індикатор зупинки і розвороту (Parabolic SAR) – трендовий індикатор, розроблений Дж. У. Веллсом Вайлдером. На

відміну від осциляторів, Parabolic SAR прив'язаний безпосередньо до цінового графіка і відображається у вигляді серії точок (парабол), що розташовуються по той чи інший бік від цінових барів. Принцип дії цього індикатора полягає в тому, що при висхідному тренді точки розміщуються нижче ціни і поступово підтягуються вгору, а при низхідному тренді – вище ціни і спускаються вниз слідом за нею. У моменти, коли тенденція змінюється на протилежну, точки перевертаються на інший бік графіка. Перехід точки через ціну інтерпретується як сигнал до виходу з позиції і одночасного відкриття протилежної позиції [8].

Формалізація індикатора базується на наступній рекурентній формулі:

$$SAR_{t+1} = SAR_t + \alpha(EP_t - SAR_t),$$

де EP_t – найвища ціна при висхідному або найнижча при висхідному тренді, α – коефіцієнт прискорення SAR. Вайлдер рекомендував ініціалізувати α = 0.02 і збільшувати цей фактор на 0.02 при кожному оновленні екстремуму EP тренду, встановивши максимальне його значення 0.20. Таким чином, чим довше триває тренд і чим більше нових екстремумів він встановлює, тим швидше Parabolic SAR наздоганяє ціну по параболічній траєкторії. Якщо в черговому періоді розраховане значення SAR виявляється по інший бік від цінового бару (тобто перевищує попередній мінімум при висхідному русі чи опускається нижче попереднього максимуму при спадному русі), це слугує ознакою зміни тренду – індикатор перестрибує на протилежний бік цінового графіка, і розрахунок продовжується вже для нового тренду.

З точки зору ринкового аналізу, Parabolic SAR виступає інструментом для визначення напрямку тренду та генерування

торгівельних сигналів розвороту. Поки точки індикатора залишаються нижче ціни, зберігається висхідний тренд, а коли ж точки розташовані вище ціни, домінує низхідний тренд. Важливо зазначити, що цей індикатор надійно працює саме в умовах чітко вираженого тренду, тоді як у боковому ринку він може давати багато хибних спрацьовувань [8].

Розглянуті технічні індикатори – смуги Болліджера, стохастичний осцилятор та Parabolic SAR – належать до різних категорій і забезпечують різні аспекти аналізу ринку. Смуги Болліджера відображають волатильність та можливі екстремальні відхилення ціни, стохастичний осцилятор вимірює відносну позицію ціни в діапазоні для прогнозування розворотів, а Parabolic SAR сигналізує про напрям і зміну тренду.

2.2 Теоретичні основи методу Random Forest

Метод Random Forest (випадковий ліс) належить до ансамблевих методів машинного навчання, що базуються на сукупності великої кількості дерев рішень. Формально ансамбль можна визначити як:

$$F(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x),$$

де $f_i(x)$ – результат окремого дерева рішень, а N – кількість дерев у ансамблі.

Ідея методу полягає у формуванні численних дерев рішень, кожне з яких будується за певною підмножиною даних та змінних, після чого результат прогнозу визначається на основі агрегування результатів цих дерев [10]. Дерева рішень є одним з базових алгоритмів машинного навчання, які застосовуються для задач класифікації та регресії. Структуру дерева рішень можна формалізувати як:

$$G = (V, E),$$

де V – множина вершин вузлів, а E – множина ребер (гілок). Кожен внутрішній вузол відповідає перевірці певної ознаки X_j, гілки представляють можливі результати такої перевірки, а листові вузли – кінцеві рішення або прогнози $f(x)$. Основною перевагою дерев рішень є їхня інтерпретованість, прозорість у прийнятті рішень та простота реалізації.

Однак, окремі дерева рішень часто схильні до проблеми перенавчання (overfitting), коли модель стає надто чутливою до шуму в навчальних даних, що призводить до зниження ефективності її роботи на нових, невідомих даних. Для подолання цього недоліку застосовують ансамблеві методи, серед яких значну популярність здобув саме алгоритм Random Forest.

Розглянемо принципи, на яких базується алгоритм Random Forest.

1. Метод Bagging – випадкове повторне відбирання підмножин навчальної вибірки із поверненням. Кожне дерево навчається на власній підвибірці даних, що забезпечує різноманітність дерев та знижує ймовірність перенавчання [11].
2. Випадковий вибір ознак – на кожному етапі побудови дерева випадковим чином обирається обмежена кількість ознак із загального набору змінних, серед яких визначається найкращий критерій поділу. Це дозволяє додатково посилити різноманітність дерев у складі ансамблю [11].

Для визначення найкращого критерію поділу часто використовується індекс Джині (Gini impurity), який формалізується наступним чином:

$$Gini(D) = 1 - \sum_{c=1}^{C} p_c^2,$$

де p_c – ймовірність належності об'єкта вибірки до класу c, а C – кількість класів.

У задачах класифікації остаточний результат Random Forest визначається через голосування більшості:

$$F(x) = \text{argmax}_c \sum_{i=1}^{N} I(f_i(x) = c),$$

де I – індикаторна функція, яка визначається наступним чином:

$$I(z) = \begin{cases} 1, \text{якщо умова } z \text{ виконується} \\ 0, \text{якщо умова } z \text{ не виконується} \end{cases}$$

Завдяки таким особливостям, метод Random Forest є одним із найбільш ефективних, стійких до шуму та популярних алгоритмів машинного навчання, який широко застосовується у вирішенні задач аналізу та прогнозування.

2.3 Теоретичні основи методу XGBoost

XGBoost (екстремальний градієнтний бустинг) є ансамблевим методом машинного навчання, що базується на послідовному побудуванні моделей, які мінімізують помилку прогнозу шляхом поступового вдосконалення попередніх моделей [12]. Метод належить до класу алгоритмів градієнтного бустингу, який застосовується як для задач класифікації, так і для задач регресії.

Формально, модель XGBoost можна описати наступним чином:

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

де $\hat{y}_i$ – прогнозований результат для об'єкта x_i, K – кількість дерев у ансамблі, f_k – k-е дерево рішення, F – простір можливих дерев рішень.

Цільова функція, яку мінімізує XGBoost, має вигляд:

$$\mathrm{Obj}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k),$$

де $l(y_i, \hat{y}_i)$ – функція втрат (наприклад, квадратична втрата, логістична втрата тощо), а $\Omega(f_k)$ – регуляризаційний член, який запобігає перенавчанню моделі:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \left|\left|w\right|\right|^2,$$

де γ – штраф за додавання нових листів, T – кількість листів у дереві, λ – коефіцієнт регуляризації, w – вектор ваг листових вузлів дерева [12].

Ключовою особливістю XGBoost є те, що алгоритм побудови дерев використовує метод градієнтного спуску для оптимізації параметрів моделі. При кожному кроці алгоритм додає нове дерево, яке максимально ефективно коригує помилки прогнозування попередніх дерев [12]. Формально, нове дерево додається наступним чином:

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

де $\hat{y}_i^{(t-1)}$ – прогноз попереднього ансамблю дерев, а $f_t(x_i)$ – дерево, яке мінімізує втрати попереднього кроку [12].

Алгоритм XGBoost має низку особливостей, таких як ефективна робота з великими наборами даних, вбудована регуляризація, паралельність обчислень, а також широкі можливості налаштування гіперпараметрів [12]. Завдяки високій точності прогнозування та швидкості роботи, XGBoost став одним із найпопулярніших алгоритмів у сучасному машинному навчанні.

2.4 Теоретичні основи методу опорних векторів

Метод опорних векторів (Support Vector Machine, SVM) – це алгоритм машинного навчання, який широко використовується для вирішення задач класифікації та регресії. Основна ідея методу полягає у побудові оптимальної гіперплощини, яка максимально розділяє дані різних класів із найбільшим можливим відступом [13]. Формально, задачу лінійної класифікації можна визначити наступним чином. Припустимо, що існує навчальна вибірка:

$$D = (x_i, y_i)_{(i=1)}^{n}, x_i \in \mathbb{R}^m, y_i \in -1,1,$$

де x_i – вектор ознак об'єкта, y_i – клас об'єкта (-1 або 1), n – кількість об'єктів у навчальній вибірці, m – кількість ознак. Лінійна гіперплощина задається рівнянням:

$$w^T x + b = 0,$$

де w – вектор нормалі до гіперплощини, b – зсув гіперплощини відносно початку координат [13]. Оптимізаційна задача для знаходження оптимальної гіперплощини формулюється так:

$$\min_{w,b} \frac{1}{2} \left|\left|w\right|\right|^2,$$

за умов обмеження:

$$y_i(w^T x_i + b) \geq 1, \ i = 1, \dots, n.$$

Ця задача вирішується за допомогою методу множників Лагранжа. Вводиться функція Лагранжа:

$$L(w, b, \alpha) = \frac{1}{2}\left||w|\right|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1],$$

де α_i – множники Лагранжа. Необхідні умови оптимальності (умови Каруша-Куна-Таккера, ККТ) визначаються як:

$$\frac{\partial L}{\partial w} = 0, \frac{\partial L}{\partial b} = 0,\ \alpha_i [y_i(w^T x_i + b) - 1] = 0.$$

З цих умов випливає, що:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i\ ,\ \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Таким чином, рішення отримуємо наступним чином:

$$f(x) = sign\left(\sum_{i=1}^{n} \alpha_i y_i \left(x_i^T x\right) + b\right),$$

де α_i – множники Лагранжа, які визначають вагу кожного об'єкта навчальної вибірки. Об'єкти, для яких $\alpha_i > 0$, називаються опорними векторами, оскільки саме вони визначають положення оптимальної гіперплощини [13].

Для випадків нелінійно-роздільних множин використовується метод ядрових функцій (kernel trick). Замість скалярного добутку $x_t^T x_j$ використовується функція ядра $K\left(x_i, x_j\right)$, що дозволяє будувати нелінійні гіперплощини у просторі більш високої розмірності [14, С. 405–517]. Типовими ядровими функціями є:

1) поліноміальне ядро: $K\left(x_i, x_j\right) = \left(x_i^T x_j + c\right)^d$;

2) Гаусове (RBF) ядро: $K(x_i, x_j) = \exp\left(-\gamma \left|\left|x_i - x_j\right|\right|^2\right)$;

3) сигмоїдальне ядро: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + c)$.

Метод опорних векторів характеризується такими властивостями, як роботою з високорозмірними даними, ефективністю, стійкістю до перенавчання та до шуму в даних.

2.5 Порівняння моделей машинного навчання

Розглянуті три алгоритми відрізняються насамперед способом контролю складності моделі. Random Forest зменшує дисперсію, будуючи багато незалежних дерев-чернеток на різних підмножинах даних і ознак; їхнє усереднення створює стійкий до шуму прогноз без явної регуляризації. XGBoost, навпаки, послідовно додає дерева, кожне з яких коригує градієнт помилки попередніх, поєднуючи гнучкість слабких моделей із жорсткою ℓ_2-регуляризацією та малими кроками бустингу, тому здатен до високої точності, але потребує тонкого налаштування. SVM шукає гіперплощину з максимальною маржею й переносить питання регуляризації у вибір ядра та параметра C; це забезпечує чітку геометричну інтерпретацію та хорошу здатність узагальнюватись, але за ціною квадратичної складності на великих вибірках.

2.6 Метрики аналізу прогностичних здатності технічних індикаторів

Оцінка ефективності класифікаційних алгоритмів є невід’ємною складовою емпіричного дослідження в галузі машинного навчання, оскільки саме вона забезпечує науково обґрунтовану перевірку гіпотези щодо здатності

моделі узагальнювати закономірності, приховані у вибірці. До класичних метрик класифікації належать такі метрики як точність (англ. precision), повнота (англ. recall) та гармонійне середнє точності та повноти (англ. F1-score). Формалізовано ці метрики можна представити в наступному вигляді:

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN},$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall},$$

де TP – кількість істинно позитивних передбачень, FP – кількість хибно позитивних передбачень [15]. Тобто precision вимірює частку коректно передбачених позитивних випадків серед усіх випадків, віднесених моделлю до позитивного класу. Високе значення precision свідчить про те, що модель рідко дає хибні спрацьовування [16]. Ця метрика є особливо корисною, коли вартість помилкового спрацювання висока (наприклад, діагностика рідкісних захворювань, або відкриття угоди). Recall оцінює здатність моделі виявляти всі наявні позитивні випадки в даних. Високе значення recall означає, що модель виявляє більшість реальних позитивних прикладів [16]. Ця метрика є важливою в задачах, де пропуск позитивного випадку є великою втратою (наприклад, виявлення шкідливого програмного забезпечення, де упущення загрози неприпустиме). F1-score представляє собою гармонійне середнє між precision і recall, забезпечуючи компроміс між обома метриками [16].

Проте у ситуаціях, коли мітки класів є дуже незбалансованими, ці метрики можуть виявитись дуже низькими чи дуже вискоими, що не буде відповідати дійсності, наприклад, висока «загальна точність» може зумовлюватися переважанням одного класу, а надмірно низькі показники precision чи recall – бути наслідком нетипової вибірки [15]. Тому доцільно розглянути також метрики Balanced Accuracy та Average Precision score.

Balanced Accuracy є узагальненням точності з поправкою на нерівномірний розподіл класів. Формально вона визначається як середнє арифметичне двох базових precision: та recall:

$$Balanced\ Accuracy\ = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right).$$

Average Precision Score узагальнює здатність класифікатора розташовувати позитивні приклади вище за негативні, незалежно від обраного порогу. Визначається як площа під кривою precision–recall (PR-curve):

$$Average\ Precision = \sum_{n=1}^{N}(R_n - R_{n-1})P_n,$$

де P_n – precision на кроці n, R_n – recall на кроці n. Площа під кривою варіюється від 0 до 1; близьке до 1 значення свідчить про те, що модель стабільно підтримує високий precision навіть за високого recall. Ця метрика особливо цінна, коли потрібно порівняти класифікатори за всіма можливими порогами та оцінити їх здатність правильно ранжувати приклади [17].

Також в якості альтернативних метрик оцінки доцільно застосувати економічні метрики. Вони незалежні від незбалансованості класів та відображають реальні ринкові результати, тому оцінка цих метрик є прямою ціллю цієї роботи. Розглянемо основні економічні метрики.

1. Максимальна кумулятивна дохідність. Максимальна кумулятивна дохідність вимірює найбільше зростання вартості портфеля за весь період спостереження, починаючи від базового рівня до найвищої точки. Тобто цей показник показує потенційну максимальну прибутковість стратегії за досліджуваний інтервал і є орієнтиром для порівняння з іншими підходами [18].
2. Максимальне просідання. Максимальне просідання характеризує найбільше відхилення портфеля вниз від локального максимуму до наступного локального мінімуму. Тобто ця метрика показує глибину найсуттєвішого збитку в періоді. Чим вона вища, тим більший потенційний капіталовий ризик.

3. Середній прибуток на угоду. Середній прибуток на угоду показує наскільки великий прибуток було отримано в середньому з прибуткових угоди.
4. Середній збиток на угоду. Середній збиток на угоду показує наскільки великий збиток було отримано в середньому зі збиткових угод;
5. Частка виграшних угод. Частка виграшних угод показує яку долю з усіх угод становили прибуткові угоди.
6. Коефіцієнт Шарпа. Коефіцієнт Шарпа є однією з базових метрик, що дозволяють кількісно оцінити співвідношення винагороди за ризик до волатильності портфеля [19]. Формалізація виглядає наступним чином:

$$S = \frac{E\left[R_p - R_f\right]}{\sigma_{p-r}},$$

де $E\left[R_p - R_f\right]$ – середня надлишкова дохідність портфеля над безризиковою ставкою за розглянутий інтервал. Надлишкова дохідність розраховується як різниця між фактичною дохідністю портфеля R_p та безризиковою ставкою R_f [19].

Коефіцієнт Шарпа спирається на кілька ключових припущень, які в окремих умовах можуть обмежувати його корисність. Ця метрика оптимально інтерпретується за умови симетричного розподілу надлишкових дохідностей; при асиметрії, що часто спостерігається в криптовалютах, використання стандартного відхилення може призводити до недооцінки або переоцінки ризиків [19]. Також коефіцієнт Шарпа трактує волатильність як лінійну міру ризику, однаково враховуючи підйоми й просідання вартості активів, тоді як інвестор зазвичай більше турбується саме про втрати.

Висновки до розділу 2

У цьому розділі було розглянуто теоретичні засади різних видів технічних індикаторів а саме трендових, індикаторів імпульсу, індикаторів волатильності та індикаторів об’єму. Було розглянуто детально індикатори смуги Болліджера, стохастичний осцилятор та параболічний індикатор зупинки та розвороту, та виділено їх сильні та слабкі сторони.

Також було обрано три ключові алгоритми машинного навчання – Random Forest, XGBoost та Support Vector Machine. Random Forest забезпечує стійкість до шуму та надмірної складності за рахунок ансамблю незалежних дерев, навчених на випадково відібраних підвибірках даних і ознак. XGBoost, в свою чергу, реалізує послідовний градієнтний бустинг із жорсткою L2-регуляризацією, що дозволяє досягати високої точності навіть на неоднорідних вибірках. Метод опорних векторів (SVM) відрізняється чіткою геометричною інтерпретацією та здатністю працювати з нелінійними зв’язками через ядрові перетворення, однак вимагає обережного підбору параметрів і може стикатися з обчислювальними обмеженнями на великих наборах даних.

Окрім цього, для об’єктивного порівняння моделей, було розглянуто використання комбінованого набору метрик, що включає як класичні показники якості класифікації (Precision, Recall, F1-score), так і розширені та економічні метрики. Економічні показники, а саме кумулятивна дохідність, максимальне просідання та коефіцієнт Шарпа враховують також фінансовий результат із ризиком, що дає змогу безпосередньо співвідносити прогнозну здатність моделей із їх практичною ефективністю на валютному ринку. Таким чином, поєднання цих метрик забезпечує всебічну оцінку обраних алгоритмів і їх придатність для реального трейдингу.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Вступ

Програмну реалізацію було виконано мовою програмування Python 3. Завдяки великому обсягу інструментів, представлених цією мовою, вона надає широкі можливості для машинного навчання, аналізу та візуалізації даних. Серед переваг цієї мови програмування також можна визначити кросплатформеність, що надає можливість запускати код на багатьох системах, простота мови, як для сприйняття, так і для написання коду, що значно прискорює реалізацію алгоритмів. Серед недоліків можна визначити повільність, що є характерною особливістю високорівневих мов програмування.

Розглянемо використані бібліотеки для побудування моделей машинного навчання та візуалізації.

1. NumPy. NumPy є фундаментальною бібліотекою, для роботи з багатовимірними числовими масивами. Вона є досить ефективною та обширною, що дозволяє швидко застосовувати математичні операції з лінійної алгебри, математичного аналізу та інших розділів математики. Також, вона є складовою багатьох інших бібліотек задля забезпечення стабільного та максимально ефективного функціонування [20].
2. Pandas. Pandas є основною бібліотекою Python, яка дозволяє швидко, зручно та ефективно працювати з даними. Вона підтримує одновимірні та двовимірні масиви, що можуть містити в собі різні типи даних, наприклад дата та час, булеві значення, текст, числа тощо. Також бібліотека має вбудовані функції для ефективного імпорту та експорту даних з різних видів джерел, очищення й трансформації даних,

агрегації та групування. Вона також є дуже зручною у роботі з часовими рядами, що є необхідним у реалізації цієї роботи [21].

3. Scikit learn. Scikit learn є однією з найпопулярніших бібліотек в машинному навчанні на Python. Завдяки обширній кількості вбудованих інструментів, вона надає можливості деякої попередньої обробки даних, наприклад feature-engineering, побудування моделей машинного навчання як з вчителем, так і без вчителя та певній візуалізації даних. Ця бібліотека є стандартизованою, тому дані після обробки цією бібліотекою можна використовувати у багатьох інших, наприклад для візуалізації даних. Вона має вже реалізовані рішення для задач регресії та класифікації, а також для нормування даних та пошуку оптимальних гіперпараметрів [22].
4. XGBoost. XGBoost є ефективною реалізацією методу градієнтного бустингу, що широко використовується в задачах класифікації та регресії. Вона має вбудовані методи регуляризації, підтримкою важливості ознак та гнучкістю у налаштуванні гіперпараметрів. Вона також є досить високопродуктивною, що значно економить час при навчанні моделей машинного навчання [23].
5. Matplotlib. Matplotlib є популярною бібліотекою для візуалізації даних в Python. Вона пропонує готові методи для побудови багатьох видів графіків різної складності. Завдяки цій бібліотеці, можна побудувати лінійні графіки, графіки статистичних розподілів, візуалізацію часових рядів тощо. Вона також є уніфікованою, тому підтримує інтеграцію з популярними бібліотеками, наприклад Pandas. Окрім цього, бібліотека надає широкі можливості налаштування всіх елементів графіка, від кольору та стилю ліній, до підписів осей та заголовків [24].
6. Plotly. Plotly є однією з найпопулярніших бібліотек для побудови інтерактивних графіків, яка надає можливості масштабування та кастомізації, а також фільтрування та отримання підказок при наведені.

Вона є особливо популярною для роботи з часовими рядами, зокрема з даними свічок активів, завдяки своїй інтерактивності та гнучкості [25].

3.2 Опис, отримання та обробка вхідних даних. Створення цільових змінних

3.2.1 Опис та отримання даних

Для навчання моделей було зібрано дані по криптовалюті Bitcoin, у торгівельній парі з Tether USDT, що є аналогом долару США. Дані було викачано з відкритого ресурсу Yahoo Finance, що надає безкоштовний доступ до ринкових даних різних видів активів. Дані було отримано у щохвилинному інтервалі у виді Open, High, Low, Close, Volume, що відповідає початковій ціні за хвилинний інтервал, максимальній та мінімальній ціні, останній ціні за інтервал, а також об’єму торгів за цей інтервал. Для отримання даних було використано бібліотеку yfinance, яка дозволяє за допомогою мови Python легко викачати ці дані. Період отриманих даних – з 01.01.2022 по 08.05.2025 [26].

3.2.2 Обробка вхідних даних та розрахунок індикаторів

Отримані дані було очищено від помилкових та пустих даних для покращення процесу навчання моделей. Також дані було сортовано за часом, задля правильного розрахунку технічних індикаторів. В якості досліджуваних технічних індикаторів було проведено розрахунок та додано

до основних даних смуги Болліндже́ра, стохастичний осцилятор та параболічний індикатор зупинки та розвороту. Усі індикатори було розраховано за допомогою бібліотеки ta – найпопулярнішої бібліотеки для технічного аналізу [27].

Смуги Болліндже́ра є трендовим індикатором, так як розраховуються на основі значень ціни закриття, що містить в собі тренд. Це робить часовий ряд нестаціонарним, що створює додаткові проблеми для моделей:

1) модель вчиться на патернах, які більше не повторяться;
2) тренувальні та тестувальні дані матимуть різні розподіли даних.

Тому з даних смуг Болліндже́ра було виключено трендову складову шляхом віднімання ковзної середньої смуг Болліндже́ра за 1440 попередніх свічок, що є еквівалентом 24 годин. Окрім цього, для стохастичного осцилятору було додано попередні два значення осцилятору для кожної свічки та обчислено різницю між значенням свічки та попереднім значенням осцилятора. Аналогічно для параболічного індикатору зупинки та розвороту було розраховано змінні відхилення між поточною ціною та лінією індикатору у стандартизованих одиницях, нормалізовані швидкість зміни індикатору та темп зміни, тобто першу і другу похідну, а також довжину безперервного періоду, під час якого значення ціни залишається з одного боку від лінії індикатора.

В якості цільової змінної для тренування моделей машинного навчання було обрано точки розвороту руху ціни, тобто екстремуми графіків. Ці точки було визначено за допомогою методу find_peaks бібліотеки SciPy. SciPy є однією с найбільших наукових бібліотек, що має вбудовані інструменти для розв’язання великого спектру задач лінійної алгебри, численних методів та задач оптимізації. Метод find_peaks призначений для виявлення локальних мінімумів та максимумів в одномірних масивах даних. Він використовує просте порівняння різниць із сусідніми значеннями. В результаті було додано цільову змінну signal, що може містити значення buy (купити), або sell (продати), що відповідає локальним мінімумам та локальним максимумам.

Для свічок, значення яких не є екстремумами, значення signal є невизначеним [28]. Таким чином для моделей машинного навчання ставиться задача класифікації свічок на виявлення розвороту або не розвороту руху ціни. Оброблені дані BTCUSDT для смуг Болліджера зображені на рисунку 3.1, дані для стохастичного осцилятору зображені на рисунку 3.2, дані для параболічного індикатору розвороту і зупинки зображені на рисунку 3.3.

time	open	high	low	close	volume	bb_high	bb_mid	bb_low	bb_high_mean_	bb_mid_mean_1h	bb_low_mean_1h
2025-05-08 20:08:00+00:00	101767.31	101767.31	101620	101622.5	18.02	2539.04	2268.99	1998.94	2220.67	2230.02	2239.37
2025-05-08 20:09:00+00:00	101622.5	101682.25	101622.5	101641.85	13.44	2568.13	2280.70	1993.28	2223.48	2228.03	2232.57
2025-05-08 20:10:00+00:00	101641.85	101648.18	101510.58	101526.73	30.95	2579.72	2290.26	2000.80	2226.63	2226.45	2226.28
2025-05-08 20:11:00+00:00	101526.74	101548	101507.93	101507.93	9.69	2587.16	2300.84	2014.53	2229.95	2225.28	2220.60
2025-05-08 20:12:00+00:00	101507.94	101547.15	101507.94	101516.22	15.43	2593.57	2311.84	2030.12	2233.61	2224.50	2215.38
2025-05-08 20:13:00+00:00	101516.23	101516.23	101489.73	101489.74	4.89	2594.30	2323.01	2051.71	2237.61	2224.14	2210.66
2025-05-08 20:14:00+00:00	101489.73	101489.74	101434.37	101434.38	9.93	2586.24	2333.45	2080.66	2241.64	2224.25	2206.85
2025-05-08 20:15:00+00:00	101434.38	101434.39	101290.42	101353.3	22.99	2564.52	2343.62	2122.72	2245.54	2224.81	2204.09
2025-05-08 20:16:00+00:00	101353.29	101353.3	101320.87	101333.34	10.43	2532.64	2354.31	2175.97	2249.11	2225.83	2202.54
2025-05-08 20:17:00+00:00	101333.33	101385.07	101333.33	101373.05	18.43	2501.22	2364.61	2228.00	2252.49	2227.35	2202.22

Рисунок 3.1 – оброблені дані BTCUSDT для смуг Болліджера

time	open	high	low	close	volume	is_peak	is_trough	is_reverse	signal	stoch_k	stoch_d	stoch_k_la	stoch_d_la	stoch_k_lag2	stoch_d_lag2	stoch_k_diff1	stoch_d_diff1	stoch_k_diff2	stoch_d_diff2
2025-05-0	101767.31	101767.31	101620.00	101622.50	18.02	FALSE	FALSE	FALSE	none	75.46	78.69	92.26	74.82	100.00	70.68	-16.80	3.86	-24.53	8.01
2025-05-0	101622.50	101682.25	101622.50	101641.85	13.44	FALSE	FALSE	FALSE	none	77.71	83.28	75.46	78.69	92.26	74.82	2.24	4.59	-14.55	8.46
2025-05-0	101641.85	101648.18	101510.58	101526.73	30.95	FALSE	FALSE	FALSE	none	64.35	85.98	77.71	83.28	75.46	78.69	-13.36	2.70	-11.11	7.29
2025-05-0	101526.74	101548.00	101507.93	101507.93	9.69	FALSE	FALSE	FALSE	none	62.17	87.61	64.35	85.98	77.71	83.28	-2.18	1.63	-15.54	4.33
2025-05-0	101507.94	101547.15	101507.94	101516.22	15.43	FALSE	FALSE	FALSE	none	63.14	86.06	62.17	87.61	64.35	85.98	0.96	-1.56	-1.22	0.07
2025-05-0	101516.23	101516.23	101489.73	101489.74	4.89	FALSE	FALSE	FALSE	none	60.06	82.98	63.14	86.06	62.17	87.61	-3.07	-3.07	-2.11	-4.63
2025-05-0	101489.73	101489.74	101434.37	101434.38	9.93	FALSE	FALSE	FALSE	none	53.64	79.49	60.06	82.98	63.14	86.06	-6.42	-3.49	-9.49	-6.56
2025-05-0	101434.38	101434.39	101290.42	101353.30	22.99	FALSE	FALSE	FALSE	none	44.23	75.20	53.64	79.49	60.06	82.98	-9.41	-4.29	-15.83	-7.78
2025-05-0	101353.29	101353.30	101320.87	101333.34	10.43	FALSE	FALSE	FALSE	none	41.92	70.88	44.23	75.20	53.64	79.49	-2.32	-4.32	-11.72	-8.61
2025-05-0	101333.33	101385.07	101333.33	101373.05	18.43	FALSE	FALSE	FALSE	none	46.53	67.49	41.92	70.88	44.23	75.20	4.61	-3.39	2.29	-7.71

Рисунок 3.2 – оброблені дані BTCUSDT для стохастичного осцилятора

time	open	high	low	close	volume	is_peak	is_trough	is_reverse	is_peak_or	is_trough_	signal	psar_gap_diff1	psar_gap_std	psar_delta1_std	psar_delta2_std	psar_dir	psar_run_len
2025-05-0	101767.3	101767.3	101620	101622.5	18.01763	FALSE	FALSE	FALSE	FALSE	FALSE	none	-233.07	-2.40	1.62	3.15	1	10
2025-05-0	101622.5	101682.3	101622.5	101641.9	13.43952	FALSE	FALSE	FALSE	FALSE	FALSE	none	-54.79	-0.56	1.34	2.94	1	11
2025-05-0	101641.9	101648.2	101510.6	101526.7	30.94586	FALSE	FALSE	FALSE	FALSE	FALSE	none	-177.40	-1.77	1.12	2.44	1	12
2025-05-0	101526.7	101548	101507.9	101507.9	9.68921	FALSE	FALSE	FALSE	FALSE	FALSE	none	-345.76	-3.15	4.70	5.59	0	1
2025-05-0	101507.9	101547.2	101507.9	101516.2	15.42629	FALSE	FALSE	FALSE	FALSE	FALSE	none	14.81	0.14	-0.09	4.62	0	2
2025-05-0	101516.2	101516.2	101489.7	101489.7	4.88929	FALSE	FALSE	FALSE	FALSE	FALSE	none	-20.09	-0.18	-0.09	-0.19	0	3
2025-05-0	101489.7	101489.7	101434.4	101434.4	9.93008	FALSE	FALSE	FALSE	FALSE	FALSE	none	-42.11	-0.39	-0.19	-0.28	0	4
2025-05-0	101434.4	101434.4	101290.4	101353.3	22.99153	FALSE	FALSE	FALSE	FALSE	FALSE	none	-58.67	-0.54	-0.32	-0.52	0	5
2025-05-0	101353.3	101353.3	101320.9	101333.3	10.43344	FALSE	FALSE	FALSE	FALSE	FALSE	none	19.64	0.18	-0.57	-0.89	0	6
2025-05-0	101333.3	101385.1	101333.3	101373.1	18.42903	FALSE	FALSE	FALSE	FALSE	FALSE	none	76.14	0.69	-0.52	-1.09	0	7

Рисунок 3.3 – оброблені дані BTCUSDT для параболічного індикатору розвороту і зупинки

Одразу з цим виникає проблема дисбалансу класів, адже моментів розвороту ціни набагато менше ніж моментів без розвороту. Для вирішення цієї проблеми було вирішено тренувати моделі машинного навчання на вручну збалансованих даних, тобто на масиві свічок з розворотом та такої ж кількості свічок без розвороту. Також, для збільшення тренувального набору даних, було штучно збільшено набір даних шляхом додання 2 свічок до

моменту розвороту та 10 свічок після розвороту. Таким чином модель матиме більше даних, проте може трохи пізніше класифікувати екстремуми. Графік свічок з об’ємом та виділеними точками розвороту зображено на рисунку 3.4.

Рисунок 3.4 – графік BTCUSDT з позначеними точками розвороту та об’ємом

Дані було поділено на тренувальну та тестову вибірку у відношенні 4 до 1. Таким чином, у тренувальному наборі дані у періоді з 01.01.2022 до 06.09.2024, а в тестовому з 06.09.2024 до 08.05.2025.

3.3 Тренування та результати обраних моделей машинного навчання

Для аналізу здатності технічних індикаторів прогнозувати ціни криптовалют було обрано такі алгоритми машинного навчання як алгоритм

випадкових лісів (Random Forest), алгоритм екстремального градієнтного бустингу (XGBoost) та метод опорних векторів (SVM). Вибір декількох моделей забезпечити більше обширні результати аналізу здатності технічних індикаторів прогнозувати ціни криптовалют. Моделі мають принципово різний підхід до навчання, що дозволяє у повній мірі порівняти прогностичну здатність технічних індикаторів та забезпечує більшу надійність висновків.

3.3.1 Random Forest

3.3.1.1 Смуги Болліндже ра

Для порівняння моделей машинного навчання в задачах аналізу здатності технічних індикаторів прогнозувати ціни криптовалют однією з використаних моделей був алгоритм випадкових лісів. Для оптимального пошуку гіперпараметрів було використано бібліотеку optuna, яка є однією з найбільш ефективних бібліотек для їх пошуку. Вона використовує байєсову оптимізацію з використанням алгоритму Tree-structured Estimator. Реалізація будує на кожному кроці дві емпіричні щільності гіперпараметрів – з низькою та з великою помилками. На новому кроці пропонує дві нові комбінації, які можуть належати до цих щільностей.

В якості метрики, щодо якої оцінювався процес навчання було обрано precision_macro. Метрика precision, яка визначає кількість правильно передбачених позитивних сигналів з усіх сигналів які були класифіковані позитивними, є ключовою в задачах прогнозування, адже інші метрики класифікації можуть видавати результати, які не відповідають дійсності, особливо враховуючи дисбаланс класів. Наприклад, при дисбалансі класів, метрика accuracy, кількість правильно передбачених значень, є зазвичай

досить високою, що досягається шляхом правильного визначення більшого з класів. Так як кількість свічок, які не є екстремумами, переважає, то використання метрики precision є доцільним. Метрика recall, яка визначає кількість правильно визначених позитивних сигналів з усіх реальних позитивних сигналів, може також спотворювати результати, так як у торгівлі активами краще пропустити вигідний момент ніж отримати хибний сигнал. Також була використана специфікація метрики precision_macro, що дозволяє визначати precision для мультикласової цільової змінної.

Розглянемо гіперпараметри, які було оптимізовано:

1) max_depth – обмежує максимальну глибинку кожного дерева;
2) n_estimators – визначає максимальну кількість дерев у ансамблі;
3) min_samples_split – задає мінімальну кількість зразків у вузлі, які необхідні для його розбиття;
4) min_samples_leaf – визначає мінімальну кількість зразків, які будуть міститись у листовому вузлі;
5) max_features – визначає яку частину ознак потрібно обрати для побудови кожного розбиття вузла;
6) criterion – визначає критерій якості розбиття, індекс Джині або ентропію.

При оптимізації моделі на даних смуг Болліджера в результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) max_depth = 25;
2) n_estimators = 129;
3) min_samples_split = 15;
4) min_samples_leaf = 16;
5) max_features = sqrt;
6) criterion = gini.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.786.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.1.

Таблиця 3.1 – Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.03	0.88	0.06	2769
None	1	0.44	0.61	346966
Sell	0.02	0.91	0.04	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 90% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.5 можна побачити що сигнали формують кластери сигналів, що йдуть підряд, проте загалом сигнали мають вірний подальший напрямок руху цін.

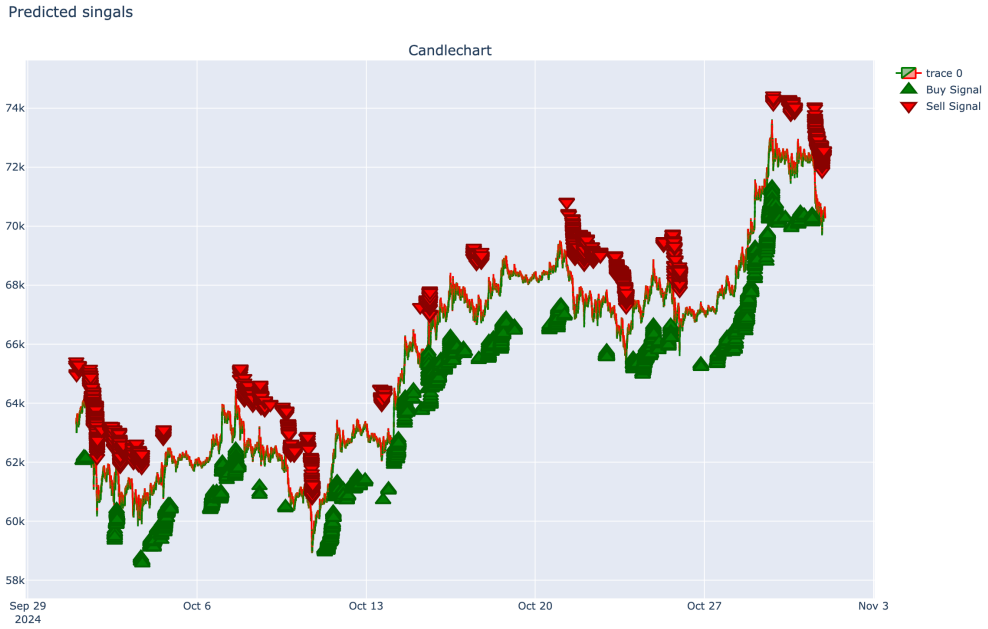

Рисунок 3.5 – Графік ціни із сигналами на основі смуг Болліджера, які формують кластери

Для подолання цієї проблеми було визначено кластери, як сигнали що знаходяться поряд та знаходяться на визначмо мінімальну відстань між сигналами в 3 години, або 180 свічок. Після фільтрації отримано сигнали, які зображені на рисунку 3.6.

Рисунок 3.6 – Графік ціни із фільтрованими сигналами на основі смуг Болліндже́ра

Було побудовано також графік ROC-кривих, щоб оцінити здатність моделі відрізняти кожен клас від інших при зміні порогу класифікації. Його особливістю є те, що він показує компроміс між recall та precision. Графік представлений на рисунку 3.7. Горизонтальна вісь означає частку помилкових спрацювань, вертикальна означає частку правильно класифікованих позитивних випадків. При аналізі графіку оцінюються площа під ROC-кривою. Чим ближча вона до 1, тим краща здатність моделі відрізняти даний клас від інших. На графіку можна побачити, що модель краще розрізняє класи buy та sell від прикладів класу none. Також побудовано криву micro-average, що об’єднує всі позитивні та всі негативні приклади та будує ROC-криву за сукупністю всіх передбачень. Отримане значення 0.62 значить, що модель має недостатню здатність забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

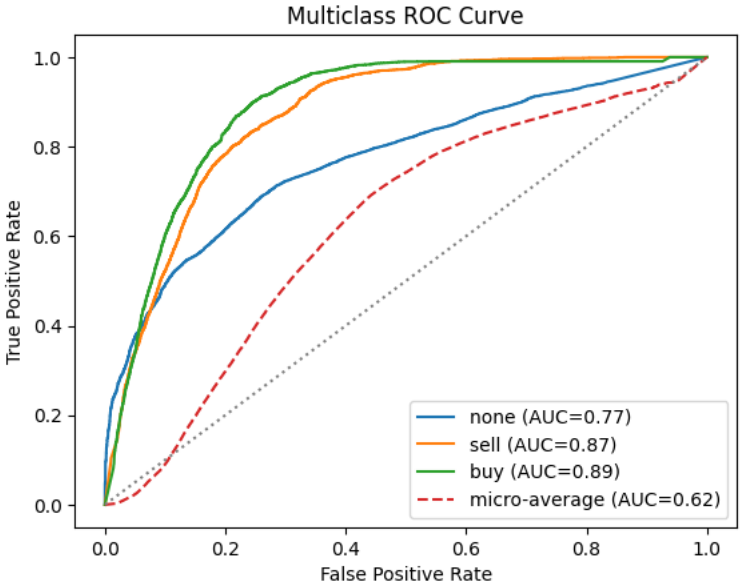

Рисунок 3.7 – Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.2. Значення Balanced Accuracy представляє збалансовану кількість правильно виявлених даних, незалежно від розміру класів, тобто модель в середньому правильно виявляє 74% прикладів кожного класу. F1 є гармонійним середнім precision та recall. Його значення значить, що модель не в змозі одночасно підтримувати високий recall та precision для кожного з класів. Average Precision score представляє площу під кривою Precision-Recall. Її значення значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.2 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.742	0.237	0.36

Також було створене тестування на ринкових даних, яке симулює угоди покупки й продажу. Було протестовано передбачення моделі машинного навчання на економічних показниках. Максимальний прибуток за період становив 6.902%, тоді як максимальне просідання портфеля досягло -11.147%, а найбільше денне просідання склало -11.022%. Середній прибуток з однієї

угоди становив 0.219%, а середній збиток становить -0.232%. У середньому прибутковий день приносив 0.348%, тоді як збитковий – втрату 0.407%. Частка виграшних угод дорівнювала 53.556%, що свідчить про незначну перевагу прибуткових угод. Коефіцієнт Шарпа, розрахований на денній основі, становив лише 0.324, що говорить про дуже низьке співвідношення прибутковості до ризику. Найдовший період просідання тривав 69 днів. У середньому здійснювалося 2.939 угоди на день, або приблизно 79.667 угоди на місяць. На рисунку 3.8 зображено графік кумулятивної прибутковості.

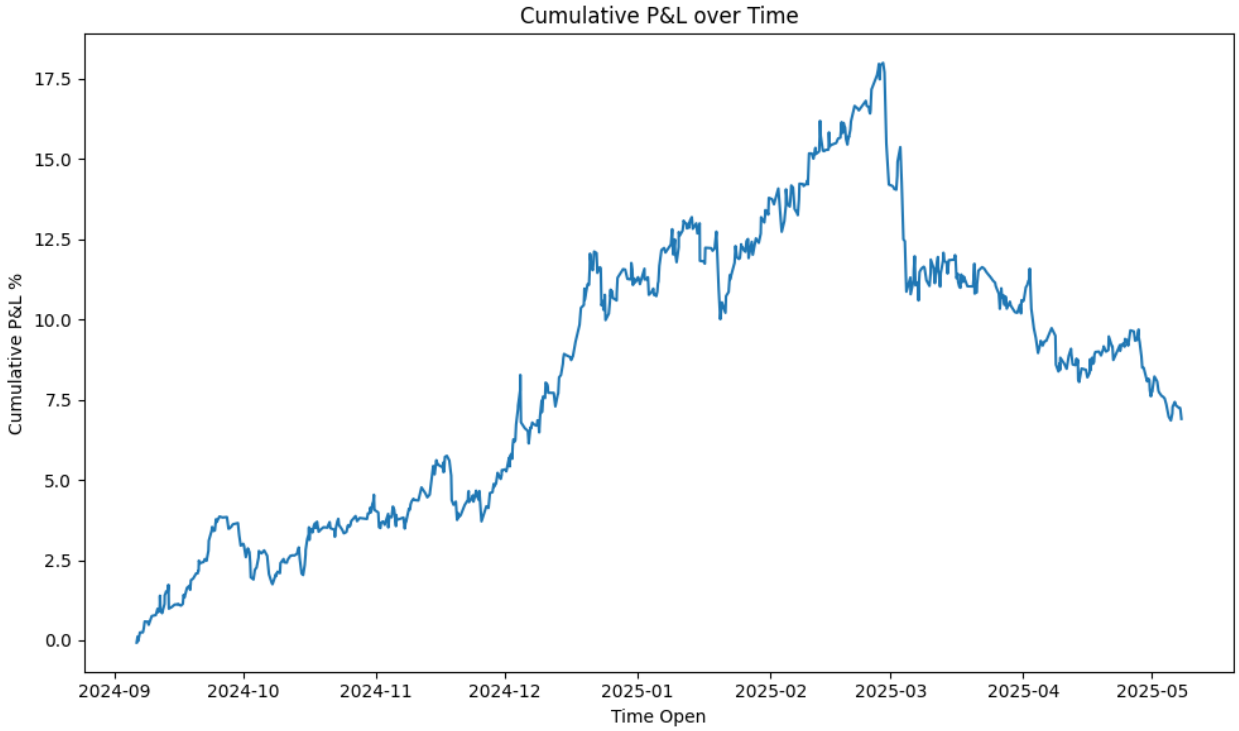

Рисунок 3.8 – Графік кумулятивної прибутковості

3.3.1.2 Стохастичний осцилятор

Аналогічно до смуг Болліндже́ра, при оптимізації моделі випадкових лісів на стохастичному осциляторі також було використано бібліотеку optuna,

а як цільову метрику для оптимізації було взято precision_macro. В результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) max_depth = 9;
2) n_estimators = 311;
3) min_samples_split = 13;
4) min_samples_leaf = 16;
5) max_features = log2;
6) criterion = gini.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.7.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.3.

Таблиця 3.3 – Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.02	0.94	0.04	2769
None	1	0.36	0.53	346966
Sell	0.02	0.91	0.04	271

Аналогічно до смуг Боллінджера, через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 80% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесені сигналів на графік ціни можна так само побачити кластери сигналів, які зображені на рисунку 3.9, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.10.

Рисунок 3.9 – Графік ціни із сигналами на основі стохастичного осцилятора, які формують кластери

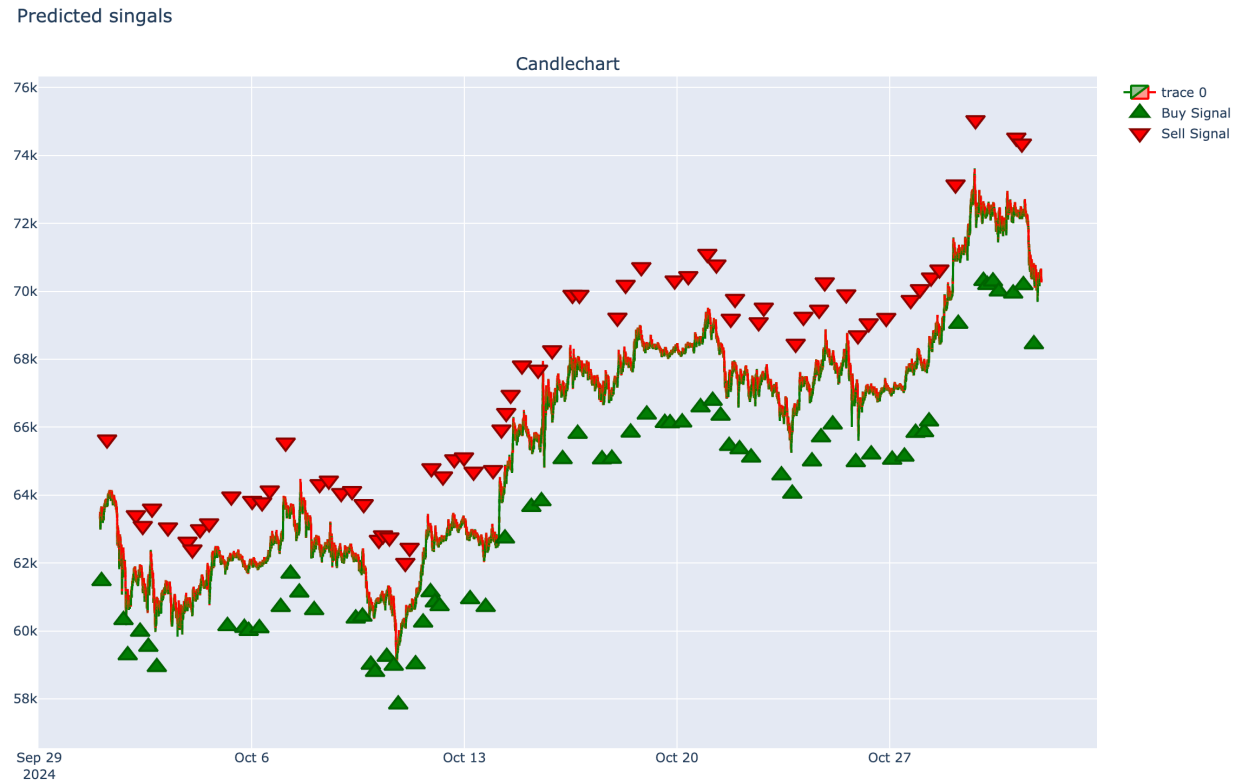

Рисунок 3.10 – Графік ціни із фільтрованими сигналами на основі стохастичного осцилятора

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.11. На графіку можна побачити, що модель краще розрізняє класи buy та sell від прикладів класу none. Також побудовано криву micro-average. Отримане значення 0.62 значить, що модель має недостатню здатність забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

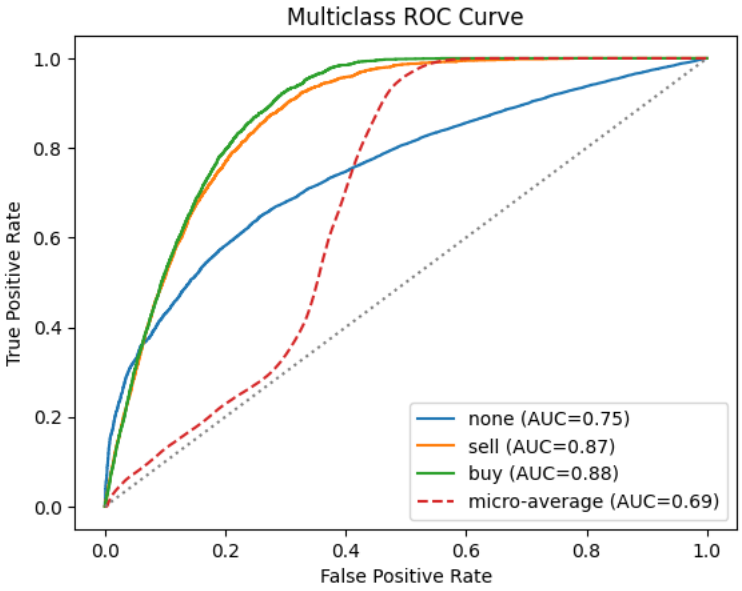

Рисунок 3.11– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.4. Значення Balanced Accuracy означає, що модель в середньому правильно виявляє 67% прикладів кожного класу. Значення F1 є досить низьким, тобто модель не в змозі одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.4 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.670	0.139	0.365

Аналогічно до смуг Болліджера було створене тестування на ринкових даних, яке симулює угоди покупки й продажу. Було протестовано передбачення моделей машинного навчання на економічних показниках. Максимальний прибуток за період склав 34.635%, тоді як максимальне просідання портфеля досягло -5.356%, а найбільше денне просідання становило -3.646%. Середній прибуток з однієї угоди дорівнював 0.209%, тоді як середній збиток – -0.203%. У середньому, прибутковий день приносив 0.596%, а збитковий – -0.507%. Частка успішних угод становила 53.936%, що свідчить про помірну перевагу виграшних операцій. Коефіцієнт Шарпа розрахований на денній основі становив 0.669, що вказує на помірне співвідношення прибутковості до ризику. Найдовший період просідання тривав 36 днів. У середньому здійснювалося 7.31 угоди на день або 199 угод на місяць. На рисунку 3.12 зображено графік кумулятивної прибутковості.

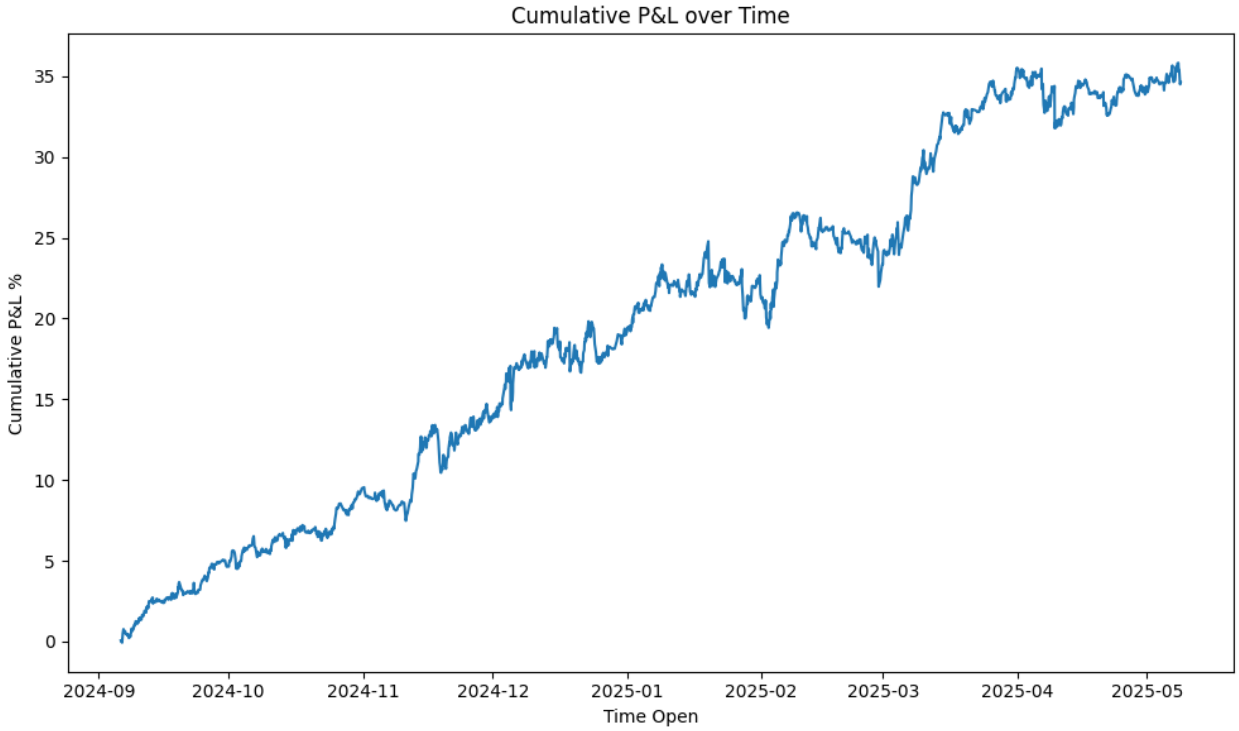

Рисунок 3.12 – Графік кумулятивної прибутковості

3.3.1.3 Параболічний індикатор зупинки та розвороту

Аналогічно до попередніх індикаторів, при оптимізації моделі випадкових лісів на параболічному індикаторі зупинки та розвороту (ParabolicSAR) також було використано бібліотеку optuna, а як цільову метрику для оптимізації було взято precision_macro. В результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) max_depth = 2;
2) n_estimators = 413;
3) min_samples_split = 5;
4) min_samples_leaf = 10;
5) max_features = None;
6) criterion = gini.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.63.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.5.

Таблиця 3.5 – Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.02	0.59	0.04	2556
None	0.99	0.52	0.68	346966
Sell	0.02	0.66	0.03	2508

Аналогічно до попередніх індикаторів, через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 72% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесені сигналів на графік ціни можна так само побачити кластери сигналів, які зображені на рисунку 3.13, тому аналогічним шляхом

було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.14.

Рисунок 3.13 – Графік ціни із сигналами на основі параболічного індикатору зупинки та розвороту, які формують кластери

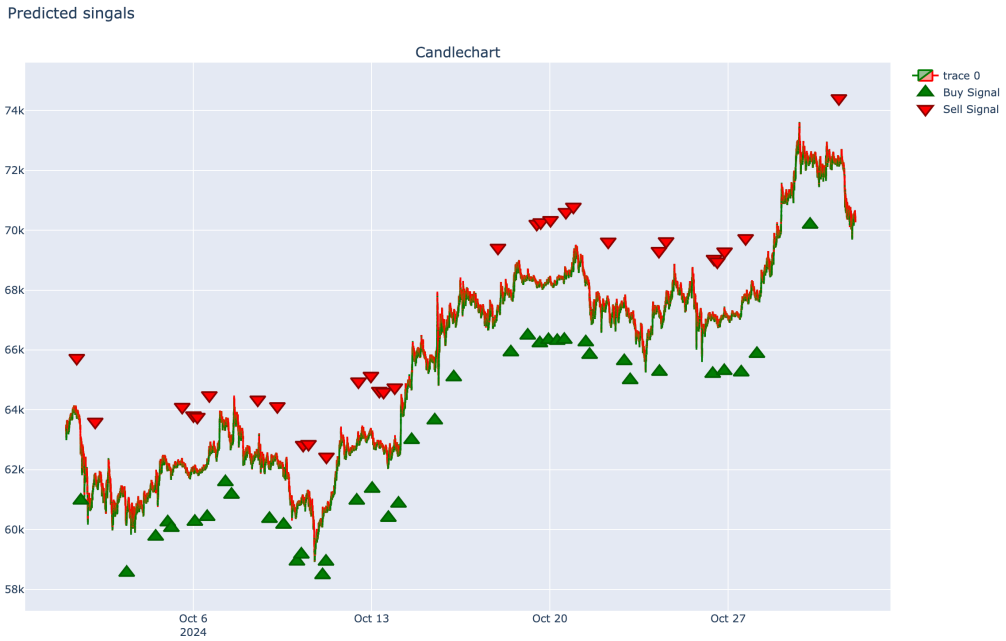

Рисунок 3.14 – Графік ціни із фільтрованими сигналами на основі параболічного індикатору зупинки та розвороту

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.15. На графіку можна побачити, що модель краще розрізняє класи buy та sell від прикладів класу none. Також побудовано криву micro-average. Отримане значення 0.62 значить, що модель має недостатню здатність забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

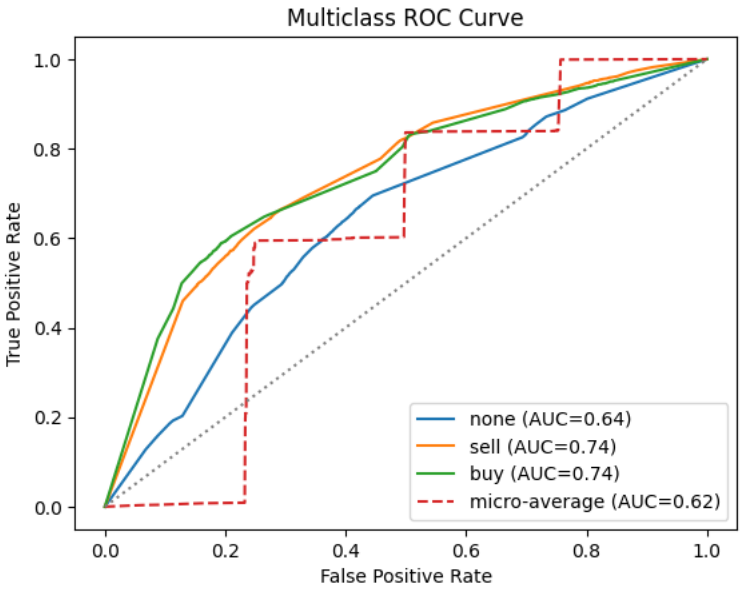

Рисунок 3.15– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.6. Значення Balanced Accuracy означає, що модель в середньому правильно виявляє 58% прикладів кожного класу. Значення F1 є досить низьким, тобто модель не в змозі одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.6 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.588	0.252	0.344

Аналогічно до попередніх індикаторів було створене тестування на ринкових даних, яке симулює угоди покупки й продажу. Було протестовано передбачення моделей машинного навчання на економічних показниках. Максимальний прибуток за період склав -0.123%, тобто стратегія завершила тестування зі загальним збитком. Максимальне просідання портфеля досягло -3.763%, а найбільше денне просідання становило -3.629%. Середній прибуток з однієї угоди складав 0.117%, тоді як середній збиток – -0.112%. У середньому прибутковий день давав 0.213%, а збитковий призводив до втрати 0.166%. Частка виграшних угод дорівнювала 48.669%, тобто прибуткових угод було трохи менше половини. Коефіцієнт Шарпа, розрахований на денній основі, склав -0.01, що свідчить про відсутність стабільної прибутковості при наявному ризику. Найдовший період просідання тривав 204 дні, що вказує на суттєве та тривале зниження капіталу без повного відновлення. У середньому виконувалась 1.073 угоди на день, або близько 29.222 угоди на місяць. На рисунку 3.16 зображено графік кумулятивної прибутковості.

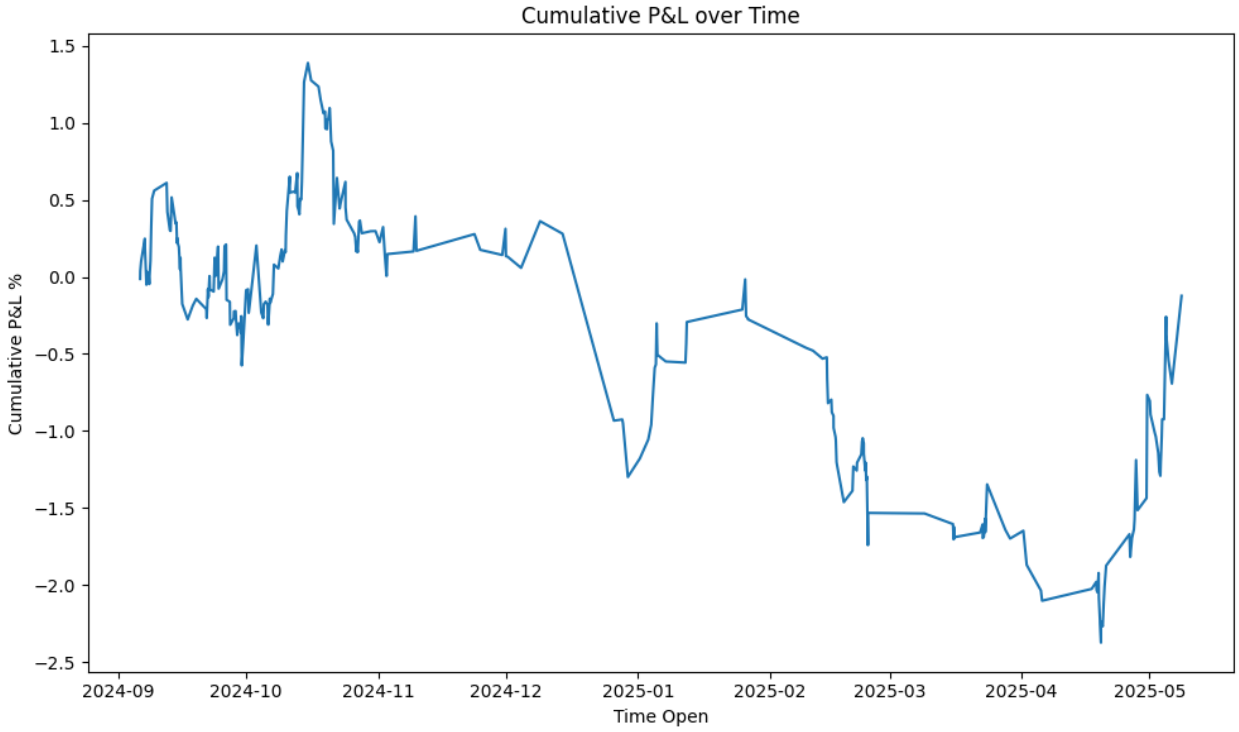

Рисунок 3.16 – Графік кумулятивної прибутковості

3.3.2 XGBoost

3.3.2.1 Смуги Болліндженра

Також було використано алгоритм екстремального градієнтного бустингу (XGBoost) для порівняння моделей машинного навчання в задачах аналізу здатності технічних індикаторів прогнозувати ціни криптовалют. Для оптимального пошуку гіперпараметрів було використано бібліотеку optuna. В якості метрики, щодо якої оцінювався процес навчання було обрано precision_macro. Також ця специфікація метрики дозволяє визначати precision для мультикласової цільової змінної.

Розглянемо оптимізовані гіперпараметри:

1) n_estimators – визначає максимальну кількість дерев у ансамблі;
2) max_depth – визначає максимальну глибину кожного дерева;
3) learning_rate – масштабує внесок кожного дерева в остаточний прогноз;
4) subsample – задає випадкову підвибірку рядків для побудови кожного дерева;
5) colsample_bytree – задає випадкову підвибірку колонок для побудови кожного дерева;
6) gamma – визначає поріг для здійснення розгалуження.
7) reg_alpha – коефіцієнт регуляризації L1;
8) reg_lambda – коефіцієнт регуляризації L2;
9) eval_Metric – визначає критерій оцінки якості на проміжних ітераціях.

При оптимізації моделі на даних смуг Болліндженра в результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) n_estimators = 456;
2) max_depth = 10;
3) learning_rate = 0.0024378009388622765;

4) subsample = 0.5139366604497112;
5) colsample_bytree = 0.7314300328446461;
6) gamma = 2.3854216057294115;
7) reg_alpha = 0.0001767680202145661;
8) reg_lambda = 7.002213771722228e-06;
9) eval_metric = mlogloss.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.799.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.7.

Таблиця 3.7 – Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.03	0.88	0.06	2769
None	1	0.45	0.62	346966
Sell	0.02	0.92	0.04	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 72% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.17 можна побачити що сигнали формують кластери сигналів, що йдуть підряд, проте у порівнянні з алгоритмом випадкових лісів, має не настільки великі кластери.

Проте кластери залишилися, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.18.

Рисунок 3.17 – Графік ціни із сигналами на основі смуг Болліджера, які формують кластери

Рисунок 3.18 – Графік ціни із фільтрованими сигналами на основі смуг Болліджера

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.19. На графіку можна побачити, що модель краще розрізняє класи buy та sell від прикладів класу none. Також побудовано криву micro-average. Отримане значення 0.59 значить, що модель має недостатню здатність забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

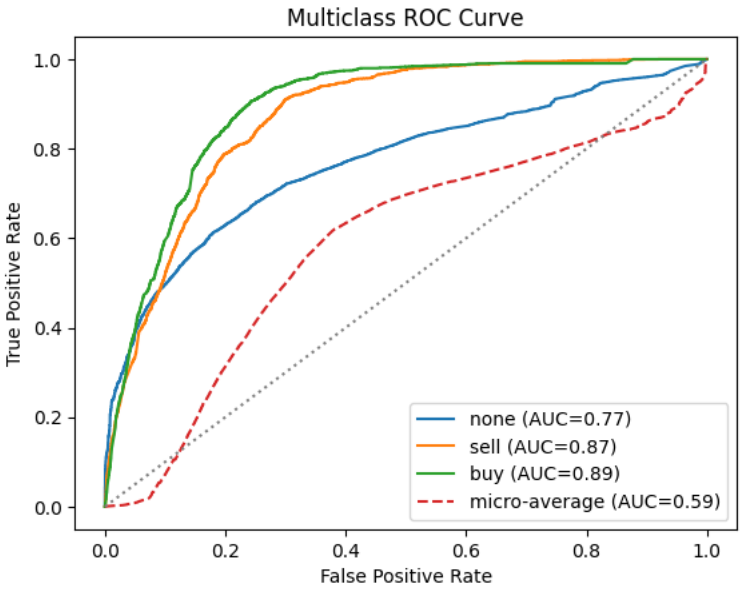

Рисунок 3.19– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.8. Значення Balanced Accuracy означає, що модель в середньому правильно виявляє 75% прикладів кожного класу. Значення F1 є досить низьким, тобто модель не в змозі одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.8 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.75	0.24	0.366

Було протестовано передбачення моделей машинного навчання на економічних показниках за допомогою тестування на ринкових даних, яке симулює угоди покупки й продажу. Максимальний прибуток за період становив 6.552%, тоді як максимальне просідання портфеля досягло -6.647%, а найбільше денне просідання склало -6.458%. Середній прибуток з однієї угоди становив 0.3%, тоді як середній збиток – -0.282%. У середньому прибутковий день давав 0.48%, а збитковий – втрату 0.434%. Частка виграшних угод становила 50.547%, що свідчить про майже однакову кількість прибуткових і збиткових операцій. Коефіцієнт Шарпа, розрахований на денній основі, склав 0.148, що говорить про дуже низьке співвідношення прибутковості до ризику. Найдовший період просідання тривав 72 дні. У середньому виконувалося 2.237 угоди на день, або приблизно 60.889 угоди на місяць. На рисунку 3.20 зображено графік кумулятивної прибутковості.

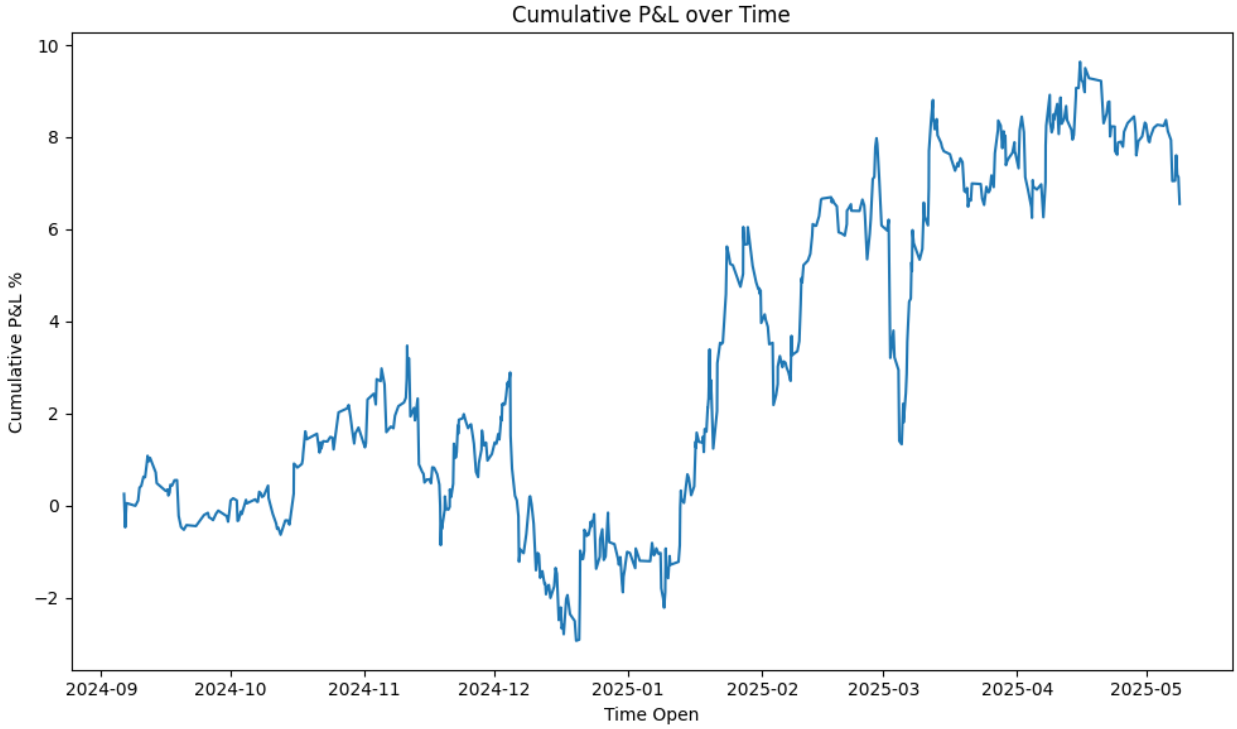

Рисунок 3.20 – Графік кумулятивної прибутковості

3.3.2.2 Стохастичний осцилятор

Аналогічно до попередніх індикаторів, при оптимізації моделі екстремального градієнтного бустингу також було використано бібліотеку optuna, а як цільову метрику для оптимізації було взято precision_macro. В результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) n_estimators = 138;
2) max_depth = 7;
3) learning_rate = 0.2985659496075293;
4) subsample = 0.823008561471604;
5) colsample_bytree = 0.5385156886954809;
6) gamma = 4.037751529597815;
7) reg_alpha = 0.05521150399359868;
8) reg_lambda = 0.10463488374623346;
9) eval_metric = mlogloss.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.701.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.9.

Таблиця 3.9 – Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.02	0.94	0.04	2769
None	1	0.36	0.53	346966
Sell	0.02	0.90	0.04	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 80% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.21 можна побачити що сигнали формують кластери сигналів, що йдуть підряд, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.22.

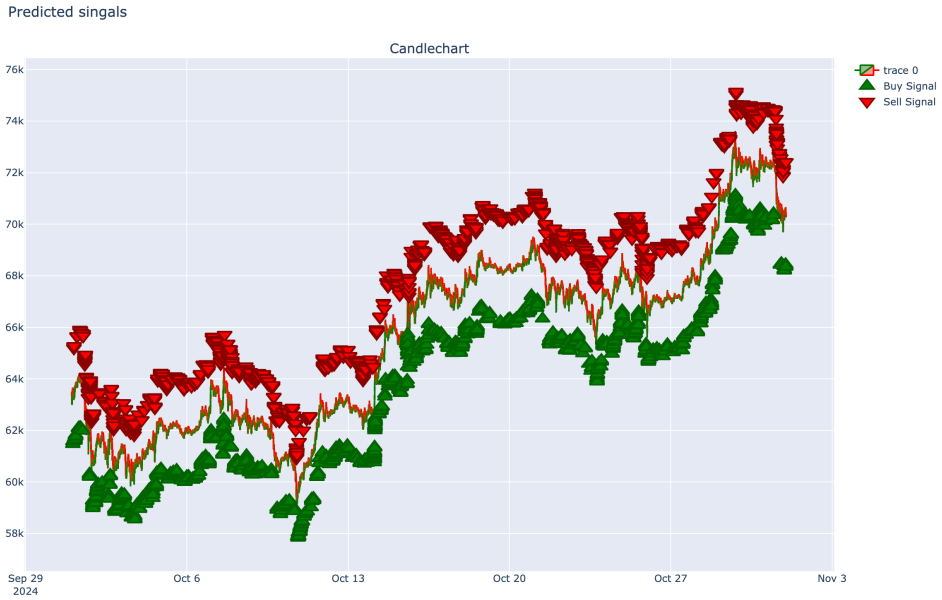

Рисунок 3.21 – Графік ціни із сигналами на основі стохастичного осцилятору, які формують кластери

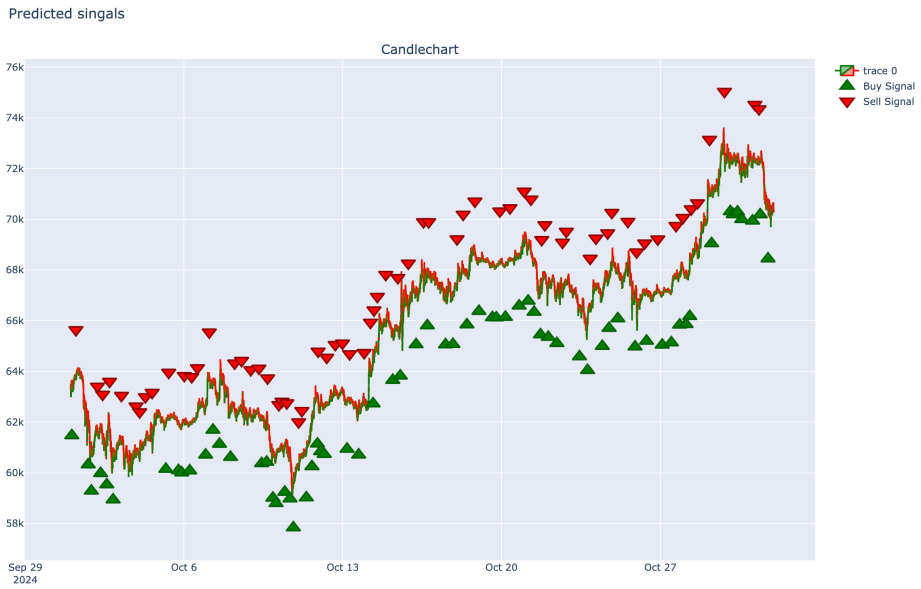

Рисунок 3.22 – Графік ціни із фільтрованими сигналами на основі стохастичного осцилятору

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.23. На графіку можна побачити, що модель найкраще розпізнає класи buy та sell, найгірше клас none. Також побудовано криву micro-average. Отримане значення 0.69 значить, що модель має недостатню здатність забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

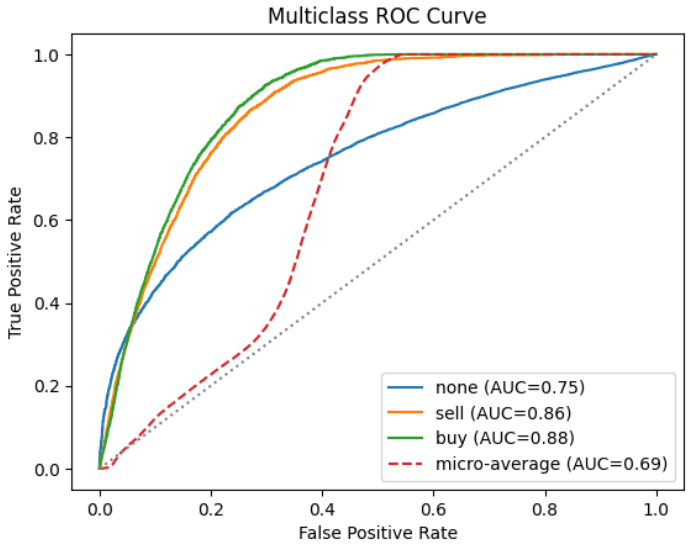

Рисунок 3.23– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.10. Значення Balanced Accuracy означає, що модель в середньому правильно виявляє 73,7% прикладів кожного класу. Значення F1 є досить низьким, тобто модель не в змозі одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.10 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.737	0.24	0.366

Було протестовано передбачення моделей машинного навчання на економічних показниках за допомогою тестування на ринкових даних, яке симулює угоди покупки й продажу. Максимальний прибуток за період склав 29.932%, тоді як максимальне просідання портфеля становило -4.263%, а найбільше денне просідання – -3.793%. Середній прибуток з однієї угоди дорівнював 0.213%, а середній збиток – -0.217%. У середньому прибутковий день забезпечував 0.597%, тоді як збитковий призводив до втрати 0.501%. Рівень успішних угод становив 54.508%, що свідчить про помірну перевагу прибуткових операцій. Коефіцієнт Шарпа, розрахований на денній основі склав 0.573, що свідчить про відносно непогану ефективність з урахуванням ризику. Найдовший період просідання тривав 26 днів. У середньому виконувалося 7.016 угоди на день, або приблизно 191 угода на місяць. На рисунку 3.24 зображено графік кумулятивної прибутковості.

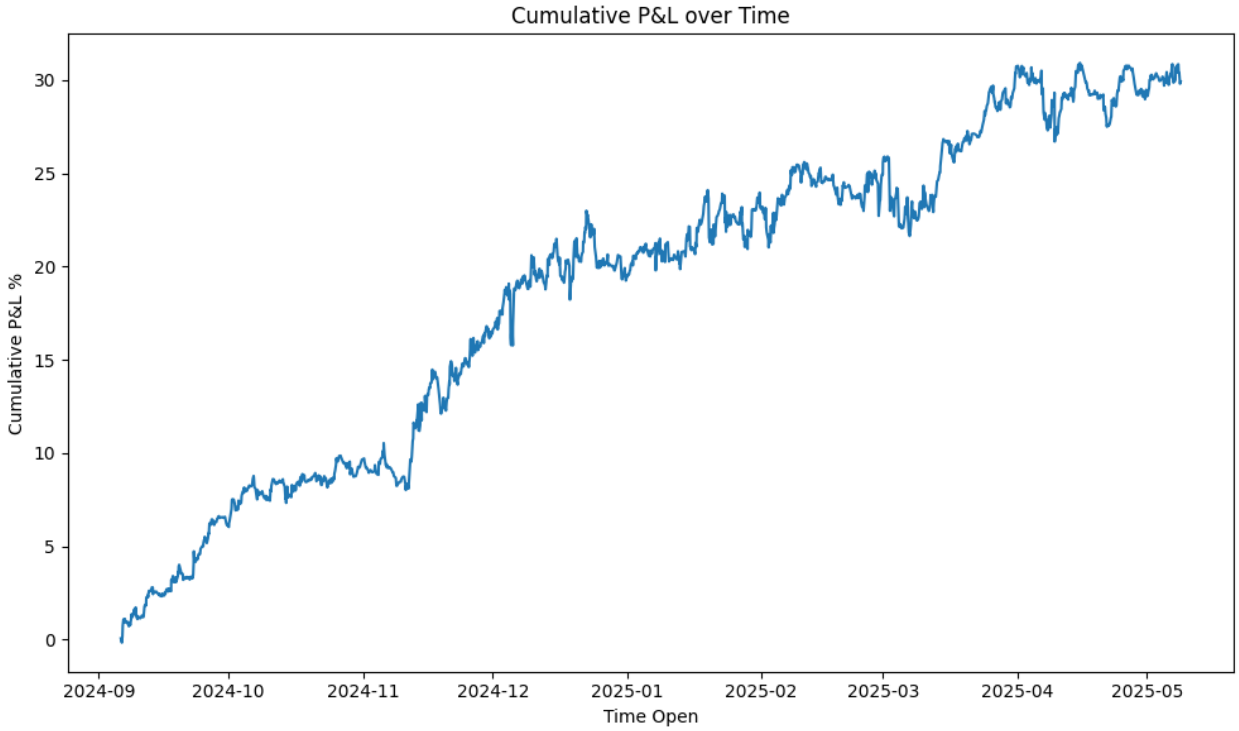

Рисунок 3.24 – Графік кумулятивної прибутковості

3.3.2.3 Параболічний індикатор зупинки та розвороту

Аналогічно до попередніх індикаторів, при оптимізації моделі екстремального градієнтного бустингу також було використано бібліотеку optuna, а як цільову метрику для оптимізації було взято precision_macro. В результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) n_estimators = 148;
2) max_depth = 6;
3) learning_rate = 0.038017616479346 08;
4) subsample = 0.661080796758976;
5) colsample_bytree = 0.9258964300014942;
6) gamma = 4.7843897609480415;
7) reg_alpha = 0.28889824326849883;
8) reg_lambda = 6.910621950534368e-06;
9) eval_metric = mlogloss.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.518.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.11.

Таблиця 3.11– Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.02	0.92	0.04	2769
None	0.99	0.02	0.04	346966
Sell	0.01	0.61	0.02	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 90% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.25 можна побачити що сигнали формують кластери сигналів, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.26.

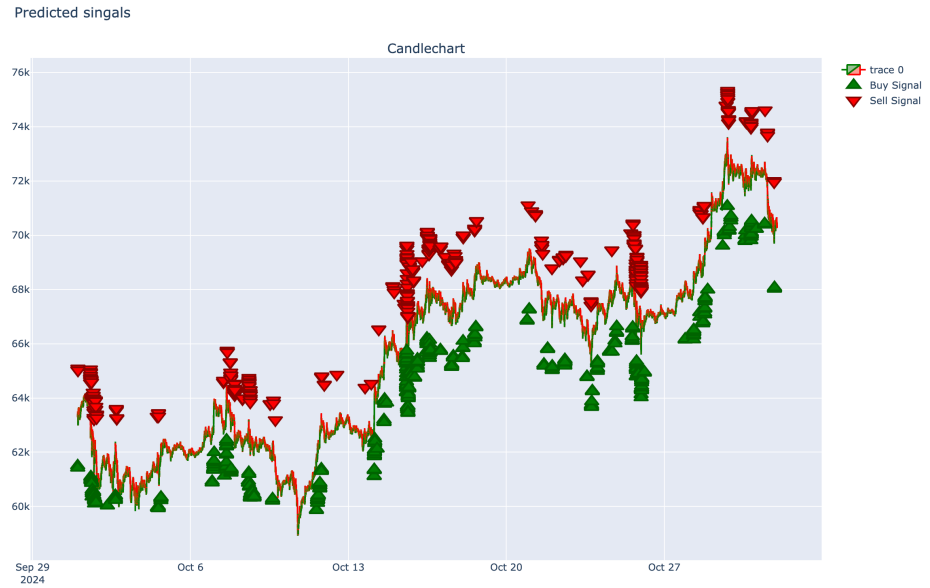

Рисунок 3.25 – Графік ціни із сигналами на основі Parabolic SAR, які формують кластери

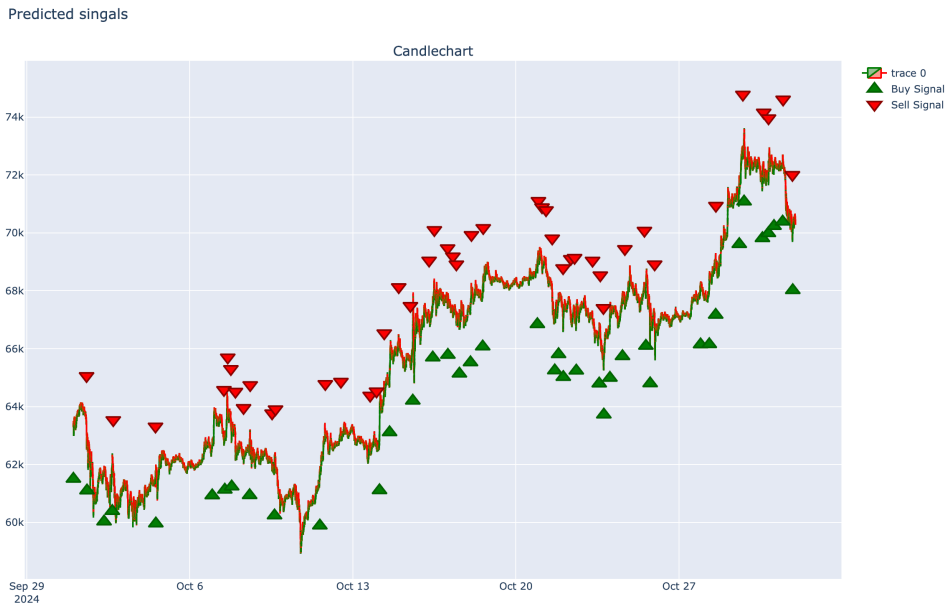

Рисунок 3.26 – Графік ціни із фільтрованими сигналами на основі Parabolic SAR

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.27. На графіку можна побачити, що модель найкраще розпізнає класи buy та sell, найгірше клас none. Також побудовано криву micro-average. Отримане значення 0.18 значить, що модель має дуже низьку здатність забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

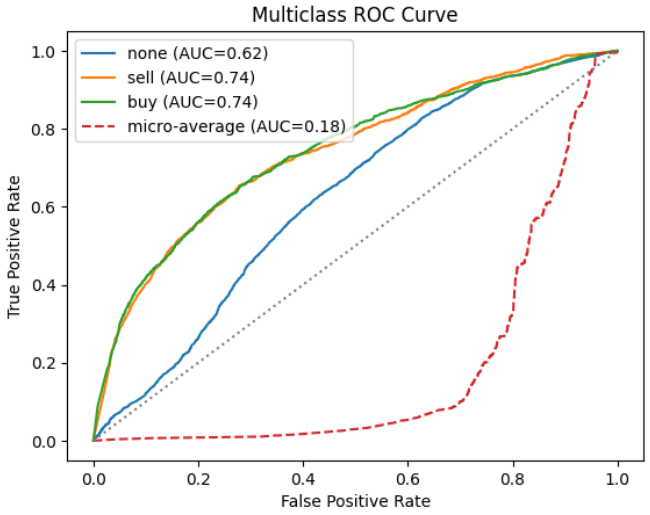

Рисунок 3.27– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.10. Значення Balanced Accuracy 51,7% означає, що рівень вгадування моделі приблизно відповідає випадковому. Значення F1 є дуже низьким, тобто модель не в змозі одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.12 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.517	0.03	0.348

Було протестовано передбачення моделей машинного навчання на економічних показниках за допомогою тестування на ринкових даних, яке симулює угоди покупки й продажу. Максимальний прибуток за період становив 3.702%, тоді як максимальне просідання портфеля досягло -5.959%, а найбільше денне просідання склало -4.867%. Середній прибуток з однієї угоди становив 0.271%, а середній збиток – -0.238%. У середньому прибутковий день приносив 0.522%, тоді як збитковий – втрату 0.379%. Частка виграшних угод дорівнювала 47.716%, тобто прибуткових операцій було менше половини. Коефіцієнт Шарпа склав 0.097, що свідчить про низьку ефективність стратегії з урахуванням ризику. Найдовший період просідання тривав 89 днів. У середньому виконувалося 3.397 угоди на день, або приблизно 87.556 угоди на місяць. На рисунку 3.28 зображено графік кумулятивної прибутковості.

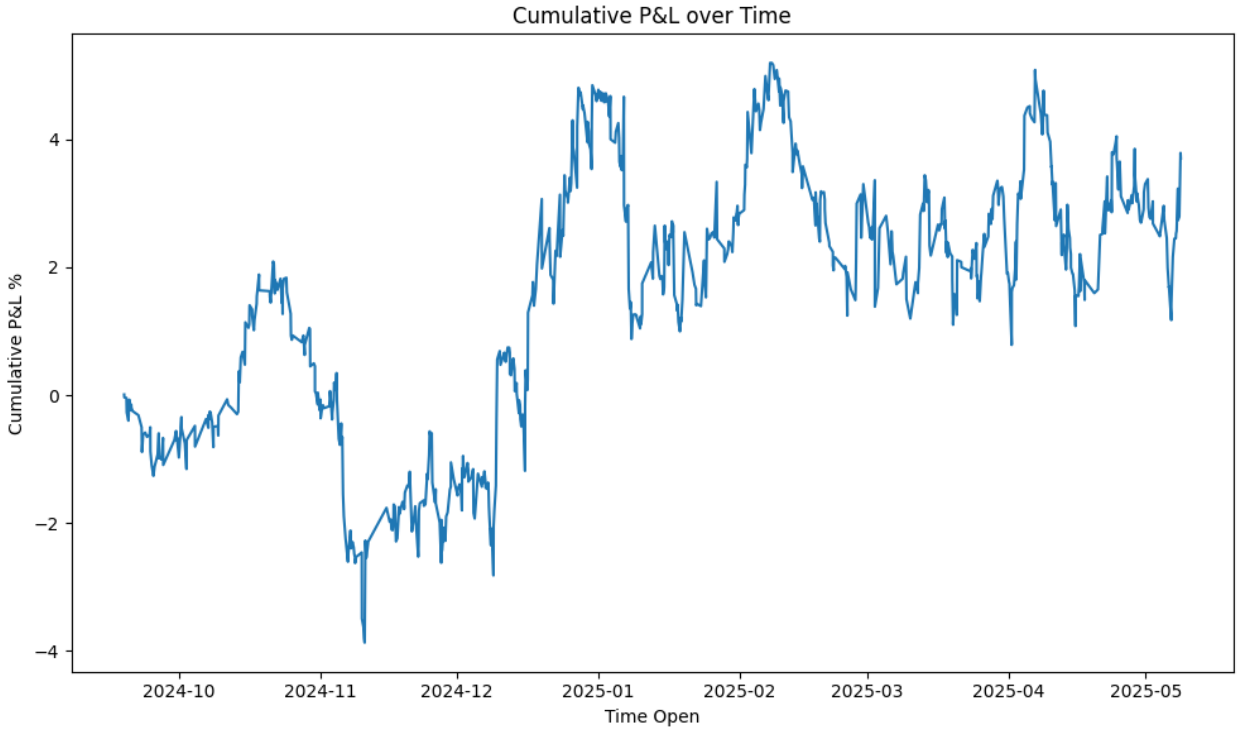

Рисунок 3.28 – Графік кумулятивної прибутковості

3.3.3 SVM

3.3.3.1 Смуги Болліджера

Останнім алгоритмом який розглядається для порівняння моделей машинного навчання в задачах аналізу здатності технічних індикаторів прогнозувати ціни криптовалют є метод опорних векторів. Для оптимального пошуку гіперпараметрів було також використано бібліотеку optuna. В якості метрики, щодо якої оцінювався процес навчання було обрано precision_macro. Також ця специфікація метрики дозволяє визначати precision для мультикласової цільової змінної.

Було оптимізовано наступні гіперпараметри:

1) c – визначає коефіцієнт регуляризації, який балансує між максимальною шириною розділяючої гіперплощини та сумарним штрафом за помилки класифікації;
2) kernel – задає тип ядра, який перетворює вхідний простір ознак у простір більшої вимірності, де вже можливо розподілити класу за допомогою лінійного зв’язку;
3) gamma – визначає для ядра rbf силу впливу одиничного навчального прикладу.

Також метод опорних векторів потребує нормалізації вхідних даних, тому також перед навчанням було використано нормалізацію даних, що реалізовано за допомогою методу scaler з бібліотеки Scikit Learn. Також для спрощення перевірки тестувальних даних процес передбачення та нормалізації вхідних було застосовано метод Pipeline, з бібліотеки Scikit Learn.

При оптимізації моделі на даних смуг Болліджера в результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) c = 98.4620527605875;
2) kernel = rbf;
3) gamma = 0.0015572238947615872.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.834.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.13.

Таблиця 3.13– Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.04	0.86	0.08	2769
None	1	0.60	0.75	346966
Sell	0.03	0.91	0.06	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 80% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.29 можна побачити що сигнали формують кластери сигналів, що йдуть підряд, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.30.

Рисунок 3.29 – Графік ціни із сигналами на основі смуг Болліджера, які формують кластери

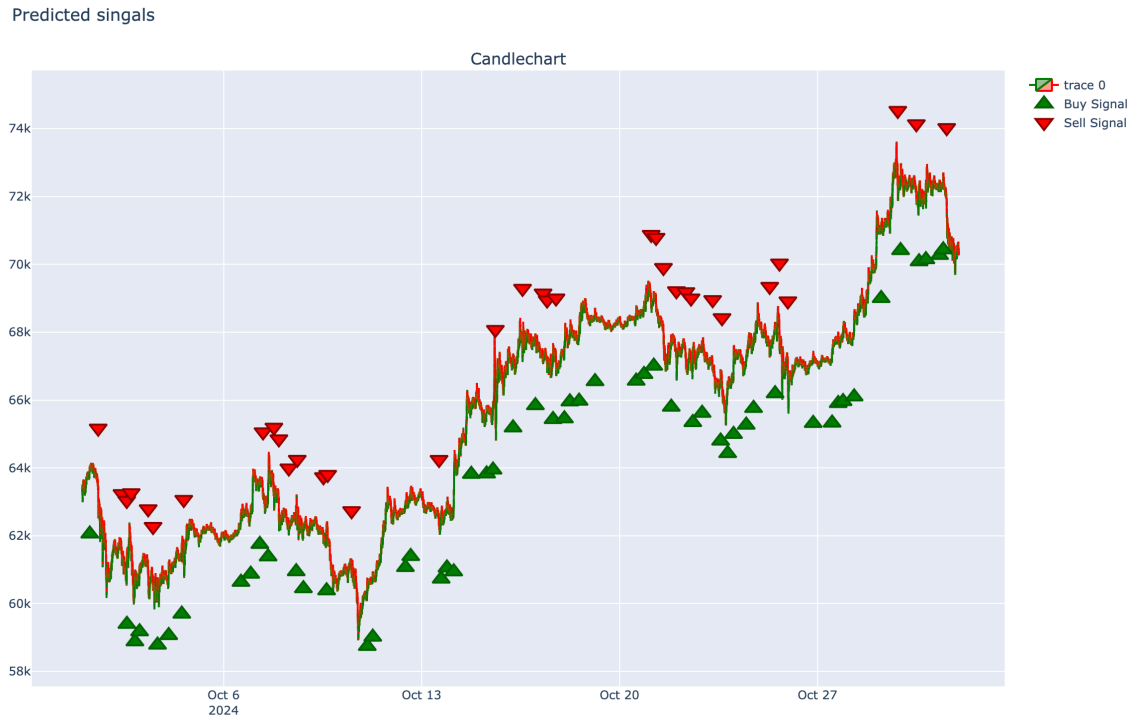

Рисунок 3.30 – Графік ціни із фільтрованими сигналами на основі смуг Болліджера

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.31. На графіку можна побачити, що модель найкраще розпізнає класи buy та sell, найгірше клас none. Також побудовано криву micro-average. Отримане значення 0.75 значить, що модель має досить високу здатність забезпечувати оптимальний баланс між чутливістю та специфічністю.

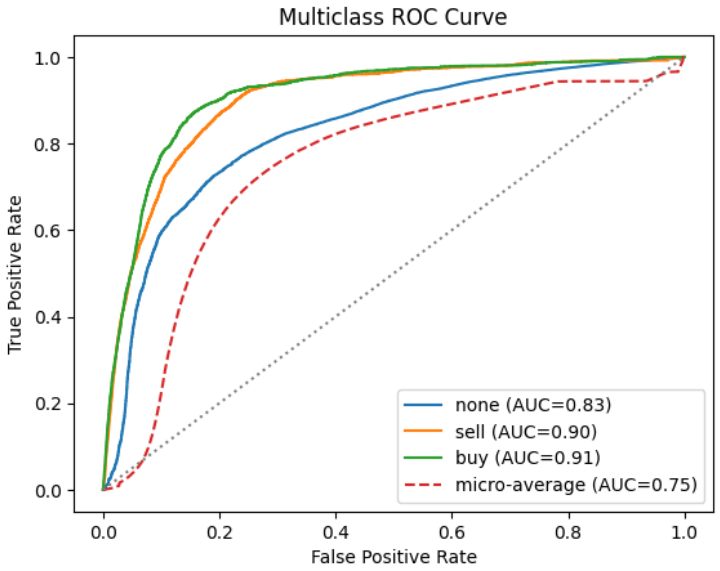

Рисунок 3.31– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.14. Значення Balanced Accuracy означає, що модель в досить правильно виявляє 79,4% прикладів кожного класу Значення F1 є невеликим, тобто модель не в змозі стабільно одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.14 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.794	0.3	0.382

Було протестовано передбачення моделей машинного навчання на економічних показниках за допомогою тестування на ринкових даних, яке симулює угоди покупки й продажу. Максимальний прибуток за період становив 20.706%, тоді як максимальне просідання портфеля досягло -8.371%, а найбільше денне просідання склало -7.836%. Середній прибуток з однієї угоди становив 0.277%, а середній збиток – -0.229%. У середньому прибутковий день забезпечував 0.532%, тоді як збитковий призводив до втрати 0.377%. Частка успішних угод дорівнювала 50.515%, що вказує на приблизно рівну кількість виграшних і збиткових операцій. Коефіцієнт Шарпа становив 0.476, що свідчить про помірне співвідношення прибутковості до ризику. Найдовший період просідання тривав 116 днів. У середньому виконувалося 3.18 угоди на день, або приблизно 86.222 угоди на місяць. На рисунку 3.32 зображено графік кумулятивної прибутковості.

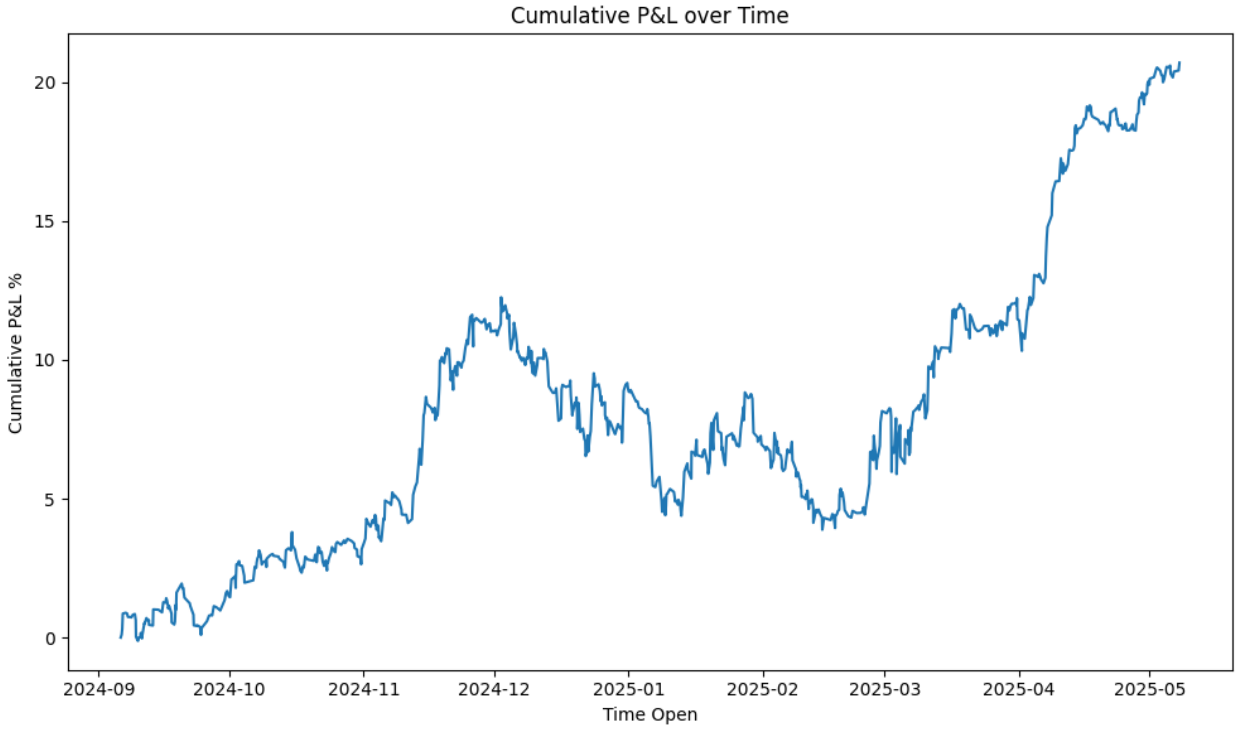

Рисунок 3.32 – Графік кумулятивної прибутковості

3.3.3.2 Стохастичний осцилятор

Аналогічно до попередніх індикаторів, при оптимізації методу опорних векторів також було використано бібліотеку optuna, а як цільову метрику для оптимізації було взято precision_macro. В результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) c = 89.49356782718716;
2) kernel = rbf;
3) gamma = 0.010515536484887244.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.724.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.15.

Таблиця 3.15– Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.02	0.96	0.04	2769
None	1	0.31	0.47	346966
Sell	0.02	0.94	0.04	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 80% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.33 можна побачити що сигнали формують кластери сигналів, що йдуть підряд, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.34.

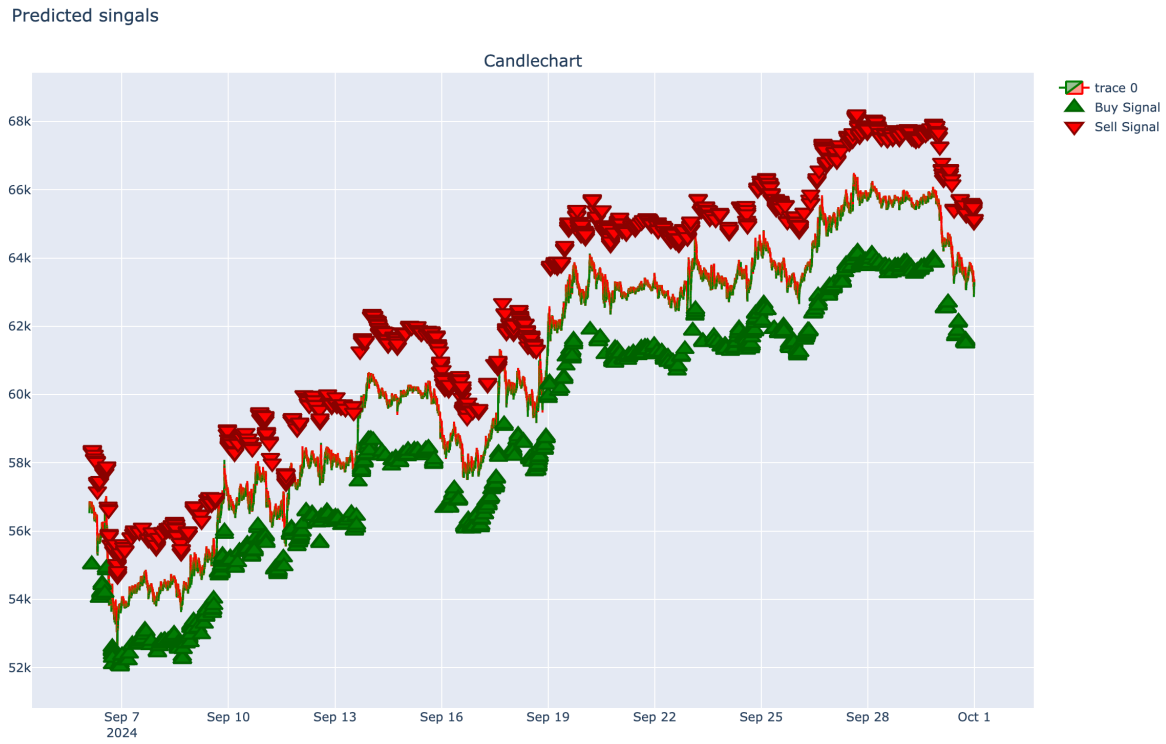

Рисунок 3.33 – Графік ціни із сигналами на основі стохастичного осцилятора, які формують кластери

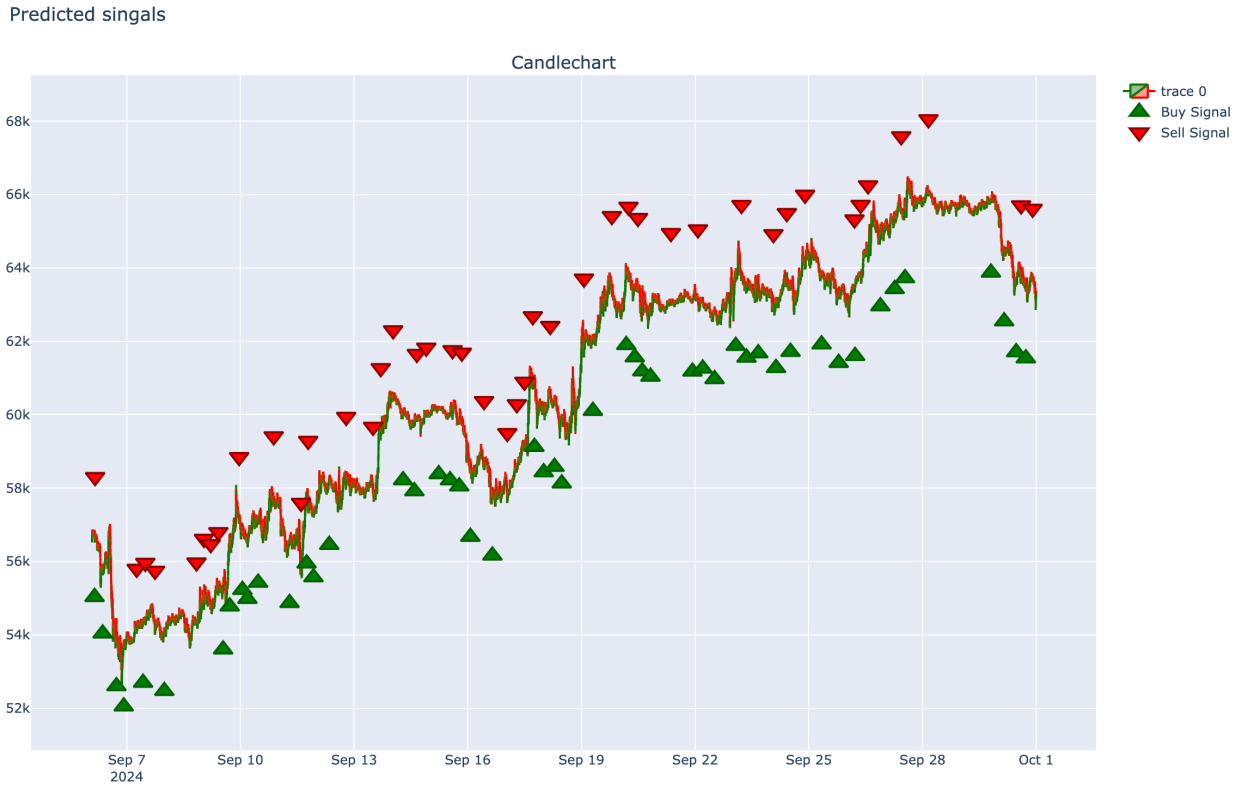

Рисунок 3.34 – Графік ціни із фільтрованими сигналами на основі стохастичного осцилятора

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.35. На графіку можна побачити, що модель найкраще розпізнає класи buy та sell, найгірше клас none. Також побудовано криву micro-average. Отримане значення 0.69 значить, що модель має досить високу здатність забезпечувати оптимальний баланс між чутливістю та специфічністю.

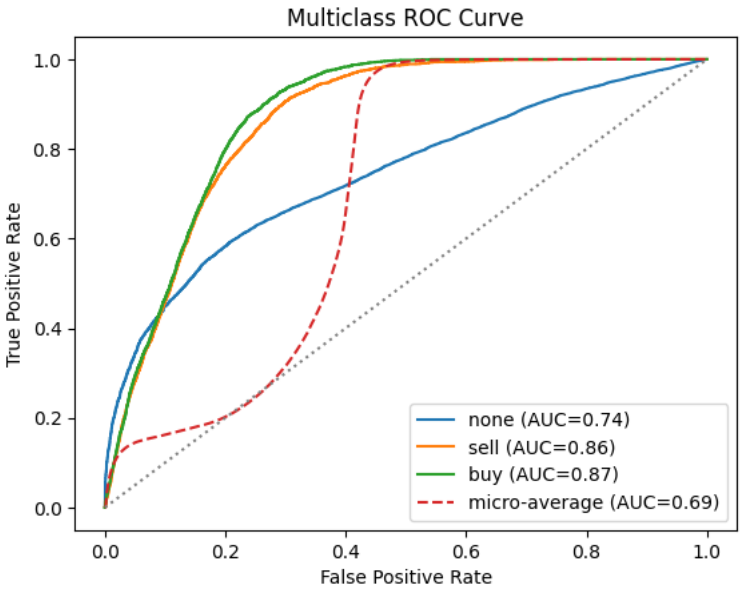

Рисунок 3.35– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.16. Значення Balanced Accuracy означає, що модель в непогано виявляє 73,5% прикладів кожного класу Значення F1 є невеликим, тобто модель не в змозі стабільно одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.16 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.735	0.185	0.356

Було протестовано передбачення моделей машинного навчання на економічних показниках за допомогою тестування на ринкових даних, яке симулює угоди покупки й продажу. Максимальний прибуток за період становив 28.496%, тоді як максимальне просідання портфеля досягло -7.537%, а найбільше денне просідання склало -6.778%. Середній прибуток з однієї угоди дорівнював 0.226%, а середній збиток – -0.215%. У середньому прибутковий день забезпечував 0.585%, тоді як збитковий день призводив до втрати 0.478%. Частка виграшних угод становила 52.16%, що свідчить про незначну перевагу прибуткових операцій. Коефіцієнт Шарпа склав 0.553, що вказує на помірну ефективність стратегії з урахуванням ризику. Найдовший період просідання тривав 43 дні. У середньому виконувалося 7.841 угоди на день, або приблизно 213.444 угоди на місяць. На рисунку 3.36 зображено графік кумулятивної прибутковості.

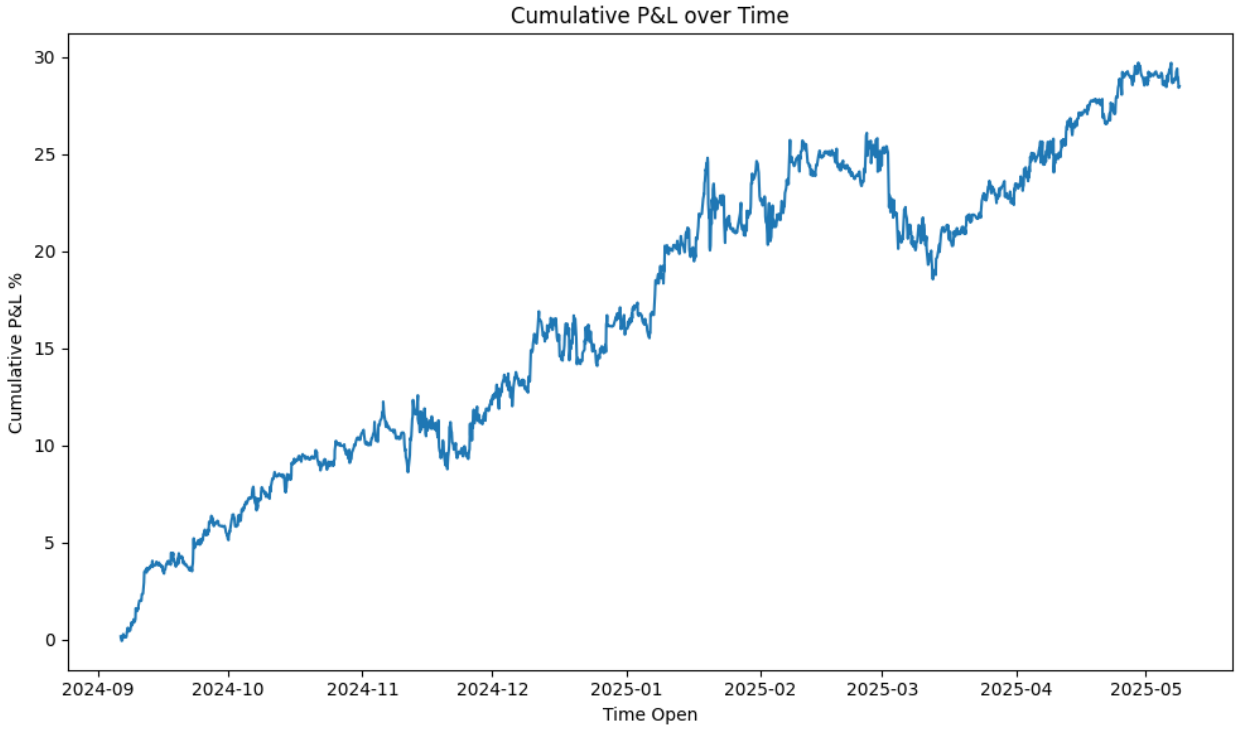

Рисунок 3.36 – Графік кумулятивної прибутковості

3.3.3.3 Параболічний індикатор зупинки та розвороту

Аналогічно до попередніх індикаторів, при оптимізації методу опорних векторів також було використано бібліотеку optuna, а як цільову метрику для оптимізації було взято precision_macro. В результаті оптимізації у 100 спроб було визначено наступні гіперпараметри:

1) c = 28.53039739101402;
2) kernel = rbf;
3) gamma = 0.05276132888741098.

Значення метрики при оптимізації на цих гіперпараметрах склало 0.553.

При використанні моделі на тестовій вибірці даних Bitcoin, модель досягла результатів, які представлені у таблиці 3.17.

Таблиця 3.17 – Метрики моделі на тестових даних

Signal	Precision	Recall	F1-score	Support
Buy	0.01	0.17	0.02	2769
None	0.99	0.55	0.75	346966
Sell	0.01	0.55	0.03	2717

Через незбалансованість класів метрики precision для сигналів на покупку та продаж виявились дуже низькими. Для покращення результатів моделі, було взято тільки передбачення з ймовірністю більше ніж 75% приналежності до класу buy або sell. Усі інші класифікуємо як none.

При нанесенні сигналів на графік ціни, на рисунку 3.37 можна побачити що сигнали формують кластери сигналів, що йдуть підряд, тому аналогічним шляхом було очищено кластери і залишено тільки перші сигнали розворотів у кластері, або перші за 3 години (180 свічок). Сигнали без кластерів зображено на рисунку 3.38.

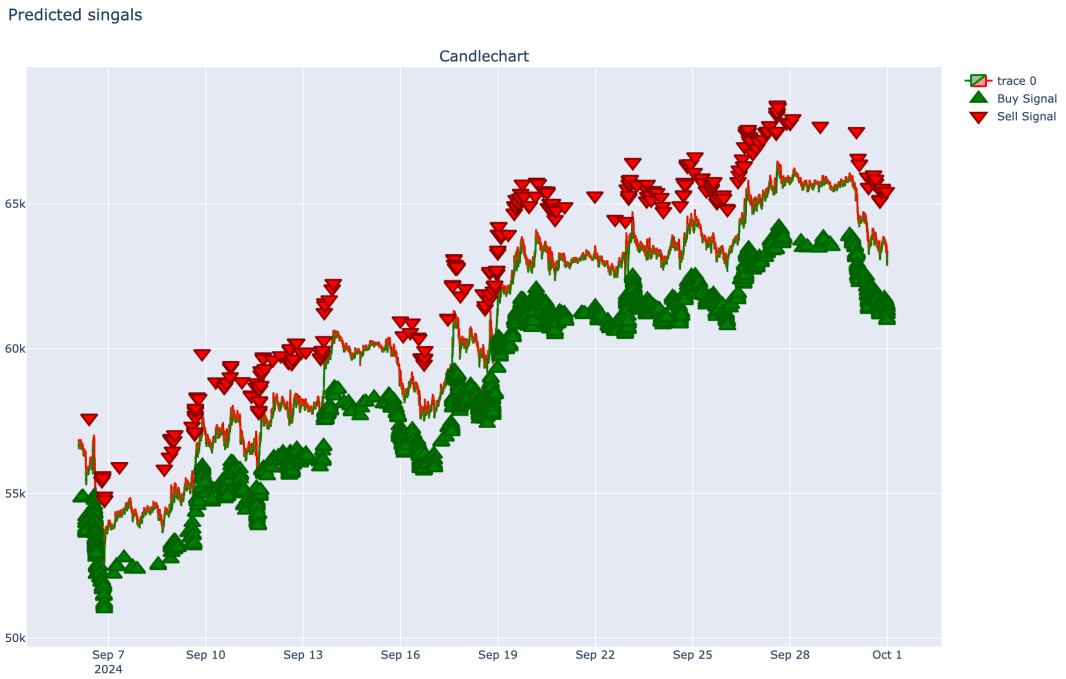

Рисунок 3.37 – Графік ціни із сигналами на основі Parabolic SAR, які формують кластери

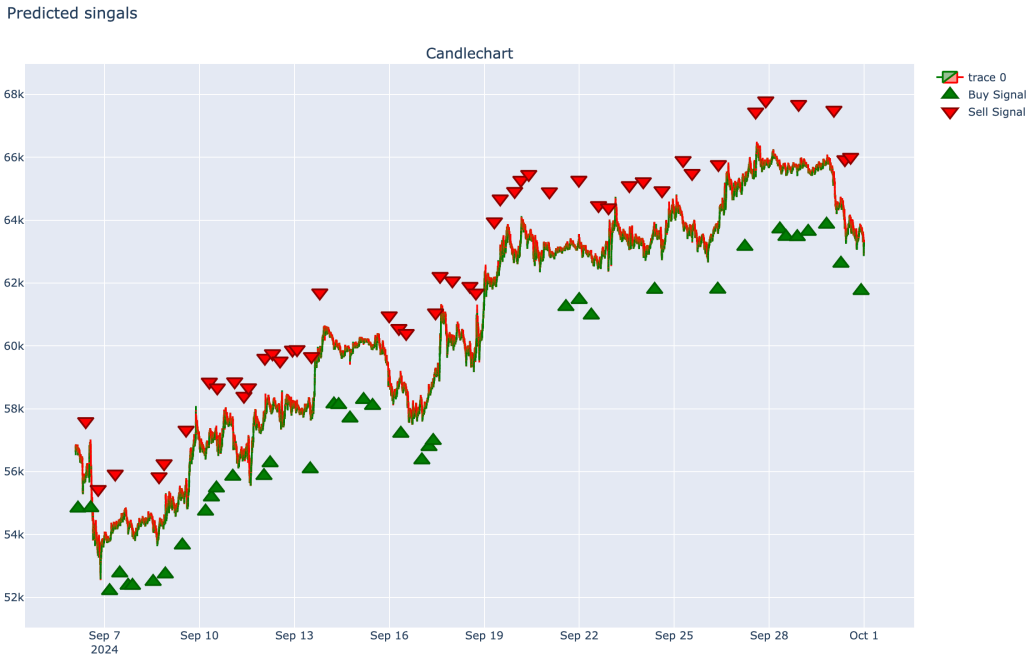

Рисунок 3.38 – Графік ціни із фільтрованими сигналами на основі Parabolic SAR

Було побудовано також графік ROC-кривих, який представлений на рисунку 3.39. На графіку можна побачити, що модель найкраще розпізнає класи buy та sell, найгірше клас none. Також побудовано криву micro-average. Отримане значення 0.18 значить, що модель не здатна забезпечувати оптимальний баланс між чутливістю та специфічністю, особливо щодо менших класів.

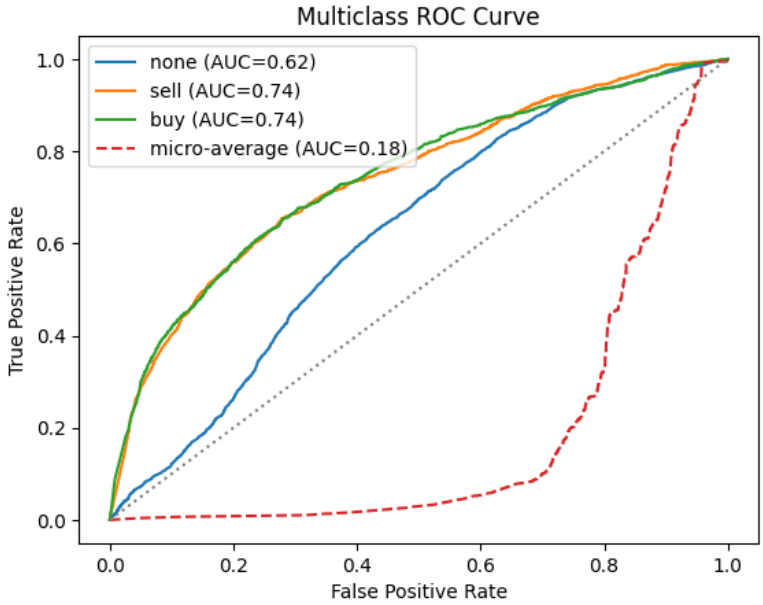

Рисунок 3.39– Графік ROC-кривих

За цією моделлю розглянуто також такі метрики, як Balanced Accuracy, F1 та Average Precision score у таблиці 3.16. Значення Balanced Accuracy означає, що модель в непогано виявляє 44,6% прикладів кожного класу Значення F1 є невеликим, тобто модель не в змозі стабільно одночасно підтримувати високий recall та precision для кожного з класів. Значення Average Precision значить, що при високих значеннях precision маємо низький recall і навпаки.

Таблиця 3.16 – Метрики Balanced Accuracy, F1, Average Precision Score

Balanced Accuracy	F1	Average Precision score
0.446	0.268	0.348

Було протестовано передбачення моделей машинного навчання на економічних показниках за допомогою тестування на ринкових даних, яке симулює угоди покупки й продажу. За результатами тестування стратегія продемонструвала максимальний прибуток 1.359%, тоді як максимальне просідання портфеля досягло -9.684%, а найбільше денне просідання склало -9.141%. Середній прибуток з однієї угоди становив 0.262%, а середній збиток – -0.271%. У середньому за день прибуткові сесії приносили 0.405%, тоді як збиткові завершувалися втратою 0.418%. Частка виграшних угод дорівнювала 51.194%, що свідчить про незначну перевагу прибуткових операцій. Коефіцієнт Шарпа при розрахунку на денних даних становив лише 0.033, вказуючи на майже відсутню надбавку за прийнятий ризик. Найдовший період просідання тривав 164 дні. У середньому виконується 2.735 угоди на день, або близько 74.444 угод на місяць. На рисунку 3.40 зображено графік кумулятивної прибутковості.

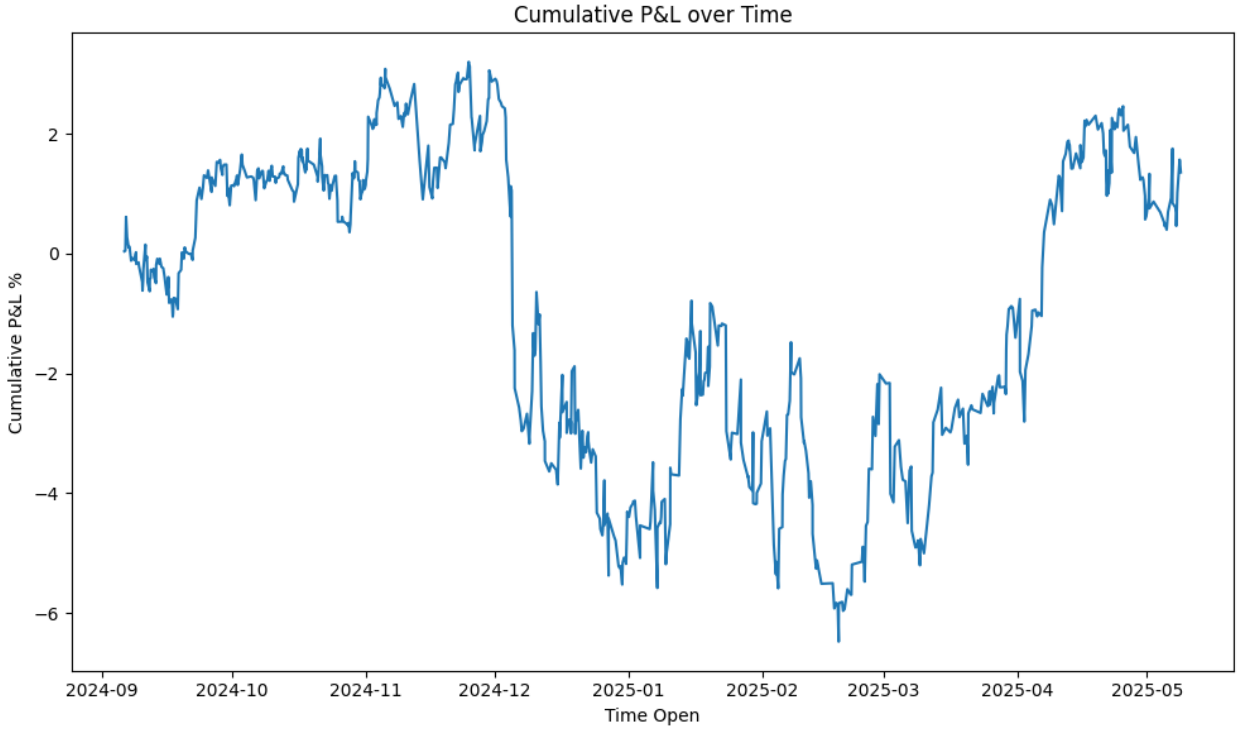

Рисунок 3.40 – Графік кумулятивної прибутковості

3.4 Порівняння та аналіз результатів

Дані, що були отримані в результаті тестування обраних моделей машинного навчання можна проаналізувати та порівняти їх, для подальших висновків про здатність обраних технічних індикаторів передбачати рух цін.

Такі стандартні метрики оцінки якості класифікації, як precision та f1 виявилась досить низькими на всіх тестувальних даних, що можна пояснити недостатньою збалансованістю класів. Цільові значення none мають переважну більшість, що робить моделі менш чутливими до класифікації значень buy та sell. Саме тому було прийнято рішення оцінювати модель також на метриках balanced accuracy, f1, average precision score та на економічних метриках.

Найкращі результати за метриками оцінки якості класифікації продемонстрував стохастичний осцилятор, особливо з використанням таких моделей як XGBoost та метод опорних векторів. Для XGBoost площі під ROC-кривими виявились досить високою, щоб сказати, що модель добре розпізнає класи buy та sell з усіх елементів цих класів (AUC 0.88 та 0.86 відповідно), а клас none гірше (AUC 0.75). Агрегована крива micro-average має AUC 0.69, що є краще ніж випадкове вгадування, проте все одно недостатньо оптимальний баланс між precision та recall. Збалансована за кількістю елементів класу точність становила 73.7%, а збалансовані метрики F1 та precision становили 0.24 та 0.366 відповідно. Для методу опорних векторів площі під ROC-кривими виявились трохи гіршими ніж при використанні XGBoost, проте все одне досить високими, щоб сказати, що модель добре розпізнає класи buy та sell з усіх елементів цих класів (AUC 0.87 та 0.86 відповідно), а клас none гірше (AUC 0.74). Площа під micro-average кривою виявилась також ж самою як і при використанні XGBoost. Збалансована за кількістю елементів класу точність становила 73.5%, а

збалансовані метрики F1 та precision становили 0.185 та 0.356 відповідно, що є приблизно однаковими показниками як і при застосуванні XGBoost.

Окрім цього, гарні результати за метриками оцінки якості класифікації продемонстрували смуги Болліндже́ра при використанні алгоритму Random Forest. Площі під ROC-кривими становили 0.90 та 0.91 для класів buy та sell відповідно, а для none – 0.83. Площа під micro-average кривою становила 0.75, що є найкращим показником серед усіх розглянутих моделей та індикаторів. Збалансована за кількістю елементів класу точність становила 79.4%, а збалансовані метрики F1 та precision становили 0.3 та 0.382 відповідно, що є кращими показниками ніж для стохастичного осцилятору.

Найгірші результати за метриками оцінки якості класифікації продемонстрував параболічний індикатор зупинки та розвороту при використанні алгоритмів XGBoost та опорних векторів. Для XGBoost площі під ROC-кривими становили 0.74 та 0.74 для класів buy та sell відповідно, а для none – 0.62. Площа під micro-average кривою становила всього 0.18, що є достатнім, щоб стверджувати що модель не здатна забезпечити оптимальний баланс між precision та recall. Площа під micro-average кривою виявилась також ж самою як і при використанні XGBoost. Збалансована за кількістю елементів класу точність становила 44.6%, а збалансовані метрики F1 та precision становили 0.268 та 0.348 відповідно.

Розглядаючи економічні метрики, було отримано найкращі результати для стохастичного індикатора незалежно від моделі, на якій було виконано передбачення. Для алгоритму випадкових лісів при тестуванні на історичному проміжку 244 днів було досягнуто прибутку 34.635%, при максимальному просіданні 5.356%. Кількість прибуткових угод була більшою ніж кількість збиткових на 3.935% а середній прибутковий день приносив більше ніж середній збитковий. Коефіцієнт Шарпа становив 0.669, що є гарним показником, так як він був розрахований на денній основі. Для XGBoost при тестуванні на тому самому проміжку було досягнуто прибутку 29.932%, при максимальному просіданні 4.263%. Кількість прибуткових угод

була більшою ніж кількість збиткових на 4.508% а середній прибутковий день приносив більше ніж середній збитковий. Коефіцієнт Шарпа становив 0.573, що є також гарним показником. Для методу опорних векторів при тестуванні на тому самому проміжку було досягнуто прибутку 28.496%, при максимальному просіданні 7.537%. Кількість прибуткових угод була більшою ніж кількість збиткових на 2.16% а середній прибутковий день приносив більше ніж середній збитковий. Коефіцієнт Шарпа становив 0.553, що є також гарним показником.

Також непогані результати показало тестування при використанні смуг Болліджера з використанням методу опорних векторів. При тестуванні на тому самому проміжку було досягнуто прибутку 20.706%, при максимальному просіданні 7.836%. Кількість прибуткових угод була більшою ніж кількість збиткових на 0.515% а середній прибутковий день приносив більше ніж середній збитковий. Коефіцієнт Шарпа становив 0.476, що є помірним значенням прибутковості до ризику.

Найгірші результати вийшли при тестуванні параболічного індикатору зупинки та розвороту на всіх моделях. При використанні Random Forest тестування показало збитковість індикатору, а коефіцієнт Шарпа склав всього -0.01. При використанні XGBoost тестування показало прибутковість становила 3.702%, а коефіцієнт Шарпа склав всього 0.097, що є досить низьким значенням навіть для денного розрахунку. При використанні методу опорних векторів тестування показало прибутковість 1.359%, а коефіцієнт Шарпа склав всього 0.033.

В середньому найкращі показники як на метриках оцінки якості класифікації, так і на метриках економічного аналізу показав стохастичний осцилятор, що може свідчити про його високу здатність прогнозування цін. Було отримано також непогані результати при використанні смуг Болліджера та методу опорних векторів, хоча й трохи гірші. Смуги Болліджера базуються на припущенні нормального розподілу цінових відхилень навколо ковзного середнього, тому при різкому та активному руху

ціни моделям машинного навчання може бути складно побудувати зв’язки на основі цих даних. Також дані смуг Болліджера, як і всі ковзні середні трохи відстають, тому моделі можуть надавати деякі передбачення трохи запізно.

Найгірші показники як на метриках оцінки якості класифікації, так і на метриках економічного аналізу показав параболічний індикатор зупинки та розвороту незалежно від обраної моделі машинного навчання. Це обумовлено тим, що цей індикатор також трохи відстає, що ускладнює побудування зав’язків для моделей, а також його чутливість залежить від значень кроку і максимальної швидкості, що потребують дуже точного налаштування в залежності від інтервалу даних, вольтинності та поточного настрою ринку.

Висновки до розділу 3

У цьому розділі було реалізовано та натреновано моделі машинного навчання для аналізу здатності технічних індикаторів прогнозувати рух цін криптовалют.

Було обрано мову програмування Python, через її універсальність та вбудованим функціям, які спрощують процес написання коду. Також було зібрано дані криптовалюти з ресурсу Yahoo Finance в періоді з 01.01.2022 до 08.05.2025. Дані було очищено і впорядковано а також розраховано обрані для дослідження технічні індикатори – смуги Болліджера, стохастичний осцилятор та параболічний індикатор зупинки та розвороту. Для формування тренувальної вибірки було визначено точки розвороту тренду та розширено їх шляхом додання сусідніх точок.

На даних кожного індикатора було побудовано по три моделі машинного навчання:

1) Random Forest;

2) XGBoost;

3) метод опорних векторів.

Оцінювання було виконано як за класичними метриками оцінювання якості класифікації, так і за економічними метриками, щоб врахувати помилки, які виникають при тестування на даних, що містять дисбаланс класів. Найвищу здатність прогнозувати продемонстрував стохастичний осцилятор, не залежно від використаної моделі машинного навчання, смуги Болліндже́ра при використанні з методом опорних векторів дали помірні результати, тоді як параболічний індикатор зупинки та розвороту виявився вкрай неефективним незалежно від алгоритму.

Практична частина показала, наскільки ефективно можна використовувати моделі машинного навчання для отримання торгових сигналів та оцінки здатності технічних індикаторів прогнозувати рух цін.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

У цьому розділі здійснюється функціонально-вартісний аналіз (ФВА) ключових етапів обробки даних та тренування моделей машинного навчання для оцінки здатності технічних індикаторів передбачати рух цін криптовалют. У сучасних дослідженнях машинного навчання зростає увага до оцінювання не лише точності прогнозів, а й ефективності витрат на реалізацію аналітичних компонентів. Функціонально-вартісний аналіз виступає системним інструментом, що дозволяє зіставити функціональні можливості розробленого рішення із ресурсними та фінансовими витратами, пов’язаними з його втіленням. Застосування ФВА у контексті прогнозування цін криптовалют є надзвичайно важливим через високу обчислювальну складність моделей і значні обсяги даних, які потребують обробки в реальному часі.

4.1 Постановка задачі проєктування

У роботі використовується метод ФВА для проведення техніко-економічного аналізу, тобто оптимізуються витрати та ефективність системи оцінки предикативної спроможності технічних індикаторів за допомогою моделей машинного навчання. Аналіз повинен зіставляти ключові аналітичні функції, тобто збір даних, обчислення індикаторів, навчання та валідацію моделей – із відповідними ресурсними та часовими витратами. Отримані результати дозволять виділити економічно обґрунтований набір індикаторів і параметрів моделей, який забезпечить оптимальний баланс між витратами на обчислювальні ресурси та якістю прогнозів.

Технічні вимоги до програмного продукту є наступними:

1) функціонування на персональних комп’ютерах із стандартним набором компонентів;
2) зручність та зрозумілість результатів аналізу предикативної здатності для користувача;
3) швидкість обробки даних та доступ до інформації в реальному часі;
4) можливість зручного масштабування та обслуговування;
5) мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – це розробка можливого програмного продукту який дозволяє оцінити здатність обраних технічних індикаторів передбачати рух цін криптовалют. Беручи до уваги основу цієї функції, основними функціями можливого програмного продукту є:

1. F_1 – вибір середовища розробки;
2. F_2 – виконання обчислень;
3. F_3 – виведення інформації в графічному та текстовому виді для користувача.

Кожна з цих функцій має декілька можливих варіантів:

1) Функція F_1:
   a. PyCharm;
   б. Jupiter Notebook.
2) Функція F_2:
   a. Самостійне створення функцій для обчислення технічних індикаторів та самостійне створення моделей машинного навчання ;
   б. Використання сторонніх бібліотек.
3) Функція F_3:
   a. Seaborn;

б. Matplotlib.

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рисунку 4.1.

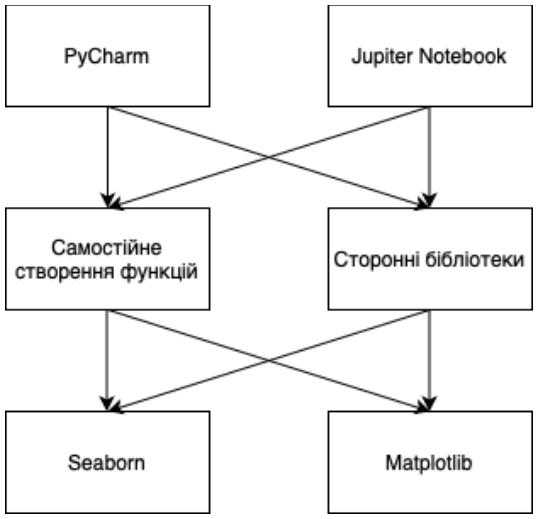

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину усіх можливих варіацій основних функцій. В таблиці 4.1 представлено позитивно-негативну матрицю.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	а	Легше управління віртуальними оточеннями та вбудована підтримка систем контролю версій.	Менш оптимізоване використання оперативної пам’яті, складна конфігурація та обмеженість безкоштовної версії.
	б	Швидкість розробки та інтерактивність. Краща зрозумілість для користувача	Обмежена підтримка тестування. Відсутність підтримки системи контролю версій.

Продовження таблиці 4.1

F_2	а	Гнучкість реалізації та прозорість коду. Відсутність проблем із залежностями бібліотек	Більша складність реалізації та оптимізації. Можливість помилок.
	б	Швидкість розробки та оптимізація продуктивності.	Обмежена гнучкість та залежність від поточної версії бібліотеки.
F_3	а	Краща зрозумілість , більша кількість вбудованих візуалізацій та підтримка інших популярних бібліотек	Обмежена гнучкість, залежність від версії та гріша продуктивність.
	б	Максимальна гнучкість налаштування графіків, сумісність з іншими бібліотеками та оптимальне використання ресурсів.	Складність налаштування зрозумілих графіків та відсутність готових візуалізацій.

На основі аналізу позитивно-негативної матриці можна зробити висновок про недоцільність використання деяких варіантів реалізації функцій, так як вони не відповідають зазначеним у морфологічній карті задачам. Основними варіантами реалізації для основних функцій є:

1) F_1: доцільно надати перевагу більш оптимальному за споживанням ресурсів та повністю безкоштовному варіанту, а також з більш зручною візуалізацією для користувача, тому приймається варіант Б;

2) F_2 доцільно надати перевагу більш надійному та оптимальному за складністю розробки варіанту, та варіанту, який є більш продуктивним, тому приймається варіант Б;

3) F_3: Програма допускає обрання обох варіантів. Можливо використати і варіант А, і варіант Б.

Отже, буде розглядатися два варіанти можливої реалізації програмного продукту:

1) F_1Б - F_2Б - F_3А;

2) F_1Б - F_2Б - F_3Б.

4.3 Обґрунтування системи параметрів програмного продукту

На основі розглянутих даних, можна визначити основні параметри вибору, які будуть використовуватися для розрахунку коефіцієнта технічного рівня програмного продукту. Для характеристики програмного продукту ми використовуємо наступні параметри:

1) X1 – Швидкодія мови програмування;
2) X2 – Обсяг пам'яті для обчислень та збереження даних;
3) X3 – Час навчання даних;
4) X4 – Потенційний обсяг програмного коду.

Гірші, середні та кращі значення параметрів вибираються на основі вимог замовника та умов, що характеризують використання ПП, як це наведено у таблиці 4.2:

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	1000	2000	5000
Об’єм пам’яті	X2	Мб	256	128	64
Час навчання даних	X3	с	1000	500	100
Потенційний обсяг програмного коду	X4	кількість рядків коду	1000	500	250

За даними таблиці 4.2 на рисунках 4.2 - 4.5 зображено графічні характеристики параметрів.

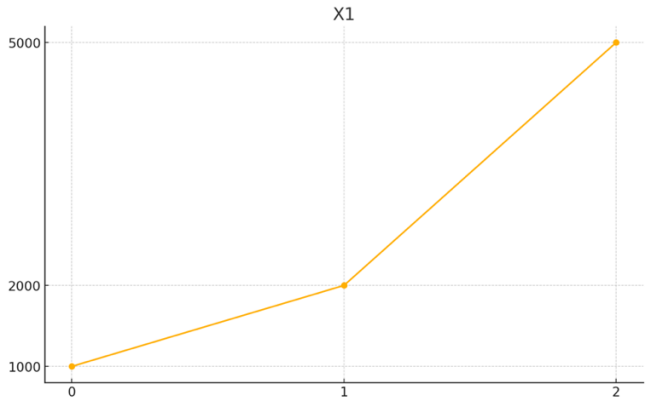

Рисунок 4.2 – Швидкодія мови програмування (оп/мс), X1

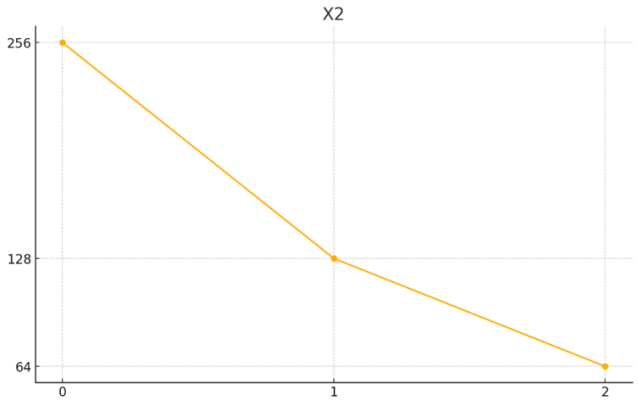

Рисунок 4.3 – Об’єм обчислювальних ресурсів (мб), X2

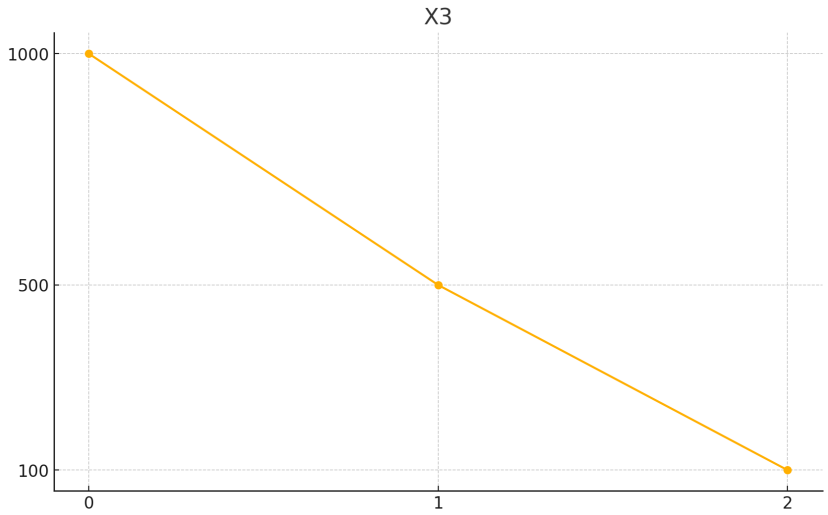

Рисунок 4.4 Час навчання моделі (хв), X3

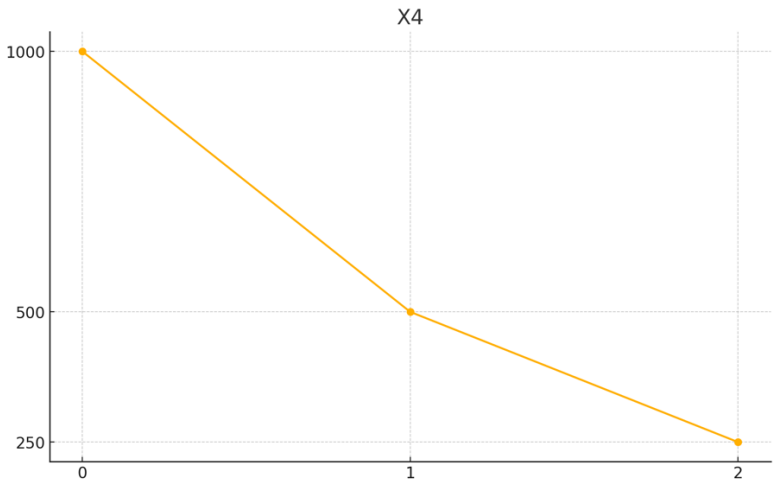

Рисунок 4.5 Потенційний об’єм програмного коду, X4

4.4 Аналіз експертного оцінювання параметрів

Після проведення ґрунтовного обговорення й аналізу кожен із семи експертів визначає відносну значимість параметрів з урахуванням мети розробки програмного продукту, здатного забезпечувати максимальну точність при налаштуванні адаптивних прогностичних моделей та обчисленні прогнозних значень. Оцінка значущості здійснюється методом попарного

порівняння. До експертної комісії входить сім фахівців. Процедура розрахунку коефіцієнтів значимості передбачає такі етапи:

1) рангів;
2) перевірку придатності експертних оцінок для подальшого
3) використання;
4) визначення оцінки попарного пріоритету параметрів;
5) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
		1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	1	1	2	1	1	1	2	9	-8.5	72.25
X2	Об’єм пам’яті	4	2	1	2	2	2	1	14	-3.5	12.25
X3	Час навчання даних	2	3	3	3	3	3	3	20	2.5	6.25
X4	Потенційний обсяг програмного коду	3	4	4	4	4	4	4	27	9.5	90.25
	Сума	10	10	10	10	10	10	10	70	0	181

Для перевірки степені достовірності експертних оцінок визначено наступні параметри:

1) сума рангів кожного з параметрів і загальна сума рангів (4.1), (4.2):

$$R_i = \sum_{j=1}^{N} r_{ij}\,, \tag{4.1}$$

$$R_{ij} = \frac{Nn(n+1)}{2} = 70, \tag{4.2}$$

де N – число експертів, n – кількість параметрів;

2) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5, \tag{4.3}$$

3) відхилення суми рангів кожного параметра від середньої суми рангів, сума яких повинна дорівнювати 0:

$$\Delta_i = R_i - T, \tag{4.4}$$

4) Загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 181. \tag{4.5}$$

Розрахування коефіцієнту узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 181}{7^2(4^3 - 4)} = 0.74 > W_k = 0.67. \tag{4.6}$$

Ранжування можна вважати достовірним, оскільки знайдений коефіцієнт узгодженості перевищує нормативний, що дорівнює 0,67. За результатами ранжування проведемо попарне порівняння всіх параметрів, результати представлені у таблиці 4.4:

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	>	<	<	<	<	<	0.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	<	<	<	<	<	<	<	<	0.5

Продовження таблиці 4.4

X2 і X3	>	<	<	<	<	<	<	<	0.5
X2 і X4	>	<	<	<	<	>	<	<	0.5
X3 і X4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги і-го параметра над j-тим, a_{ij} визначається по формулі (4.7):

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.7)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{\text{ві}}$ за наступними формулами (4.8) та (4.9):

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.8)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.9)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх. На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.10)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (4.11)$$

З таблиці 4.5 випливає, що різниця між значеннями коефіцієнтів вагомості не перевищує 2%, тому додаткові ітерації не потрібні.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0.5	0.5	0.5	2.5	0.16	9.25	0.16	34.13	0.16
X2	1.5	1	0.5	0.5	3.5	0.22	12.25	0.21	44.98	0.21
X3	1.5	1.5	1	0.5	4.5	0.28	16.25	0.28	59.13	0.27
X4	1.5	1.5	1.5	1	5.5	0.34	21.25	0.36	77.88	0.36
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функції

Абсолютні значення параметрів $X2$ (об’єм пам’яті для обчислень та збереження даних), $X3$ (Час навчання даних) та $X4$ (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП. Абсолютне значення параметра $X1$ (швидкодія мови програмування) обрано не найгіршим. Для кожного варіанта реалізації основних функцій визначаємо рівень якості окремо. Коефіцієнт технічного рівня для кожного варіанта реалізації ПП (таблиця 4.6) розраховується за наступною формулою (4.12):

$$K_K(j) = \sum_{i=1}^{n} K_{ві,j} B_{i,j}\,, \tag{4.12}$$

де n – кількість параметрів, $K_{ві,j}$ – коефіцієнт вагомості і-го параметра, $B_{i,j}$ – оцінка і-го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	Б	X1	450	12	0.16	1.92
F_2	Б	X2	100	17	0.21	3.57
F_3	А	X3	150	15	0.27	4.05
	Б	X4	800	20	0.36	7.2

За даними з таблиці 4.6 за формулою (4.13):

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.13)$$

Визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1.91 + 3.57 + 4.05 = 9.53,$$

$$K_{K2} = 1.92 + 3.57 + 7.2 = 12.69.$$

З розрахунків випливає, що кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1) розробка проекту програмного продукту;
2) розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1, а в завданні 2 – до групи 3.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється за наступною формулою (4.14):

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.14)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 40 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.1. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 40 \cdot 1.1 \cdot 0.8 = 35.2 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто T_P = 45 людино-днів, $K_П$ = 1.2, $K_{СК}$ = 1, $K_{СТ}$ = 0.9:

$$T_2 = 45 \cdot 1.2 \cdot 0.9 = 48.6 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (35.2 + 48.6 + 9.53) \cdot 8 = 746.64 \text{ людино-годин.}$$

$$T_{II} = (35.2 + 48.6 + 12.69) \cdot 8 = 771.92 \text{ людино-годин.}$$

У розробці бере участь один програміст з окладом 30000 грн та один експерт з аналітики з окладом 25000. Визначимо середню зарплату за годину за формулою (4.14):

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.}, \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{30000 + 25000}{3 \cdot 21 \cdot 8} = 109.13 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою (4.16):

$$С_{ЗП} = С_ч \cdot T_i \cdot К_Д, \quad (4.16)$$

де $С_ч$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 109.13 \cdot 746.64 \cdot 1.1 = 89628.90$ грн.

II. $С_{ЗП} = 109.13 \cdot 771.92 \cdot 1.1 = 92663.59$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 89628.90 \cdot 0.22 = 19718.36$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 92663.59 \cdot 0.22 = 20385.99$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0.3 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0.2 = 72000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 72000 \cdot (1 + 0.2) = 86400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 86400 \cdot 0.22 = 19008 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.1 \cdot 0.25 \cdot 25000 = 6875 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.1 \cdot 25000 \cdot 0.05 = 1375 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 10 – 10) \cdot 8 \cdot 0.75 = 1446 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1446 \cdot 0.25 \cdot 0.9 \cdot 9.43 = 3068.05 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 25000 \cdot 0.67 = 11222.5 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$$С_{ЕКС} = 86400 + 19008 + 6875 + 1375 + 3068.05 + 11222.5 = 127948.55 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 127948.55 / 1446 = 88.48 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 88.48 \cdot 746.64 = 66062.70$ грн.

II. $С_М = 88.48 \cdot 771.92 = 68299.48$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0.67, \quad (4.19)$$

I. $С_Н = 89628.90 \cdot 0.67 = 60051.36$ грн.

II. $С_Н = 92663.59 \cdot 0.67 = 62084.61$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 89628.90 + 19718.36 + 66062.70 + 60051.36 = 235461.32$ грн.

II. $С_{ПП} = 92663.59 + 20385.99 + 68299.48 + 62084.61 = 243433.67$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = \frac{9.53}{235461.32} = 4.04737 \cdot 10^{-5},$$

$$К_{ТЕР2} = \frac{12.69}{243433.67} = 5.21292 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 5.21292 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, які залишились після першого відбору двох варіантів виконання програмного

комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР2} = 5.21292 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

1) вибір середовища розробки – Jupiter Notebook;
2) виконання обчислень – Сторонні бібліотеки;
3) виведення інформації в графічному та текстовому виді для користувача – Matplotlib

Даний варіант виконання програмного комплексу дає користувачу зручну реалізацію програми, зрозумілий інтерфейс, швидку роботу системи і ефективний та інформативний аналітичний модуль.

Висновки до розділу 4

В четвертому розділі було проведено функціонально-вартісний аналіз програмного продукту. Було здійснено системне оцінювання його ключових функціональних компонентів з урахуванням їх значущості, вартості реалізації та впливу на загальну ефективність програмного забезпечення. Проаналізовано низку альтернативних варіантів реалізації окремих функцій з метою вибору найбільш доцільного рішення, що відповідає вимогам економічної обґрунтованості, технічної доцільності та забезпечує сумісність із подальшими етапами розробки.

За підсумками аналізу було ідентифіковано основні функціональні характеристики програмного продукту, сформовано перелік параметрів, що визначають його технічні та експлуатаційні властивості, а також обґрунтовано вибір оптимального варіанту реалізації. Отримані результати створюють аналітичну базу для подальших етапів проєктування та впровадження програмного забезпечення.

ВИСНОВКИ

У цій роботі було проведено дослідження моделей машинного навчання в задачах аналізу здатності технічних індикаторів прогнозувати ціни криптовалют. В процесі дослідження було проаналізовано його актуальність та історичний контекст. Також було розглянуто вже існуючі реалізації на ринку із детальним аналізом їх методів. Аналіз існуючих реалізації показав необхідність у якісному відборі технічних індикаторів та попит на це з боку криптовалютної сфери.

В теоретичній частині було розглянуто теоретичні засади індикаторів технічного аналізу, їх види та цілі. Такі технічні індикатори як смуги Болліндже́ра, стохастичний осцилятор та параболічний індикатор зупинки і розвороту виявились доцільними для використання. Також було проведено огляд теоретичних основ обраних методів машинного навчання, їхні переваги та недоліки. Розглянуті методи Random Forest, XGBoost та метод опорних векторів продемонстрували доцільність їх використання у роботі. Окрім цього, було розглянуто метрики аналізу прогностичної здатності технічних індикаторів, серед яких було розглянуто класичні метрики оцінки класифікації та економічні метрики.

В практичній частині було описано збір та обробку даних, а також розрахунок обраних технічних індикаторів. Було створено навчальну вибірку, натреновано моделі машинного навчання, оброблено вихідні дані та проаналізовано результати. За результатами прогнозування був здійснений аналіз за допомогою класичних метрик оцінювання класифікації, та економічних метрик, щоб забезпечити максимально всебічний аналіз результатів.

У четвертому розділі було проведено функціонально вартісний аналіз можливого програмного продукту, в якому було визначено основні функції та обрано оптимальні параметри. Також було проведено експертний аналіз, за

допомогою якого було зроблено висновок щодо найкращого варіанту реалізації.

Таким чином, результати роботи показали успішність використання моделей машинного навчання в задачах аналізу здатності технічних індикаторів передбачати рух цін криптовалют. В якості подальшого розвитку можна виділити такий напрямок, як побудування інтелектуальної системи підтримки прийняття рішень на основі цього аналізу для використання у власних та комерційних цілях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. McCarthy J., Minsky M.L., Rochester N., Shannon C.E. A proposal for the Dartmouth Summer Research Project on Artificial Intelligence. URL: https://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html (Last accessed: 24.05.2025).
2. Kimoto T., Asakawa K., Yoda M., Takeoka M. Stock market prediction system with modular neural networks. In: Proc. 1990 International Joint Conference on Neural Networks (IJCNN'90), San Diego, CA, USA, 1990, Vol. 1, P. 1–6.
3. Boser B. E., Guyon I. M., Vapnik V. N. A Training Algorithm for Optimal Margin Classifiers. Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT'92). Pittsburgh, PA: *ACM Press*, 1992. P. 144–152.
4. Ho T. K. The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1998. URL: https://ieeexplore.ieee.org/document/709601 (Last accessed: 24.05.2025)
5. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need. In: Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, December 4–9, 2017. P. 5998–6008 URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (Last accessed: 25.05.2025)
6. Altrady. Types of Technical Indicators URL: https://www.altrady.com/crypto-trading/technical-analysis/types-of-technical-indicators (Last accessed: 24.05.2025)
7. Lakhwan D., Dave A. Determining the most efficient technical indicator of investing in financial markets based on trends, volume, momentum and

volatility. Myśl Ekonomiczna i Polityczna. 2020; No. 3 Vol. 70. P. 64 – 137. URL: https://bazekon.uek.krakow.pl/rekord/171628292

8. Prasetijo A. B., Saputro T. A., Windasari I. P., Windarto Y. E. Buy/sell signal detection in stock trading with Bollinger Bands and Parabolic SAR: With web application for proofing trading strategy. Proceedings of the 2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Piscataway NJ, 2017, P. 41–44. DOI: http://dx.doi.org/10.1109/ICITACEE.2017.8257672
9. Paik C., Choi J., Vaquero I.U. Algorithm-based low-frequency trading using a stochastic oscillator, Williams %R, and trading volume for the S&P 500. Journal of Risk and Financial Management. 2024; Vol. 17 No.11 Art. 501 DOI: http://dx.doi.org/10.3390/jrfm17110501
10. Breiman L. Random Forests. Machine Learning. 2001; Vol. 45. P. 5–32. doi: https://doi.org/10.1023/A:1010933404324
11. Scornet E., Biau G., Vert J.-P. Consistency of Random Forests. Annals of Statistics. 2015. Vol. 43, No. 4. P. 1716–1741, DOI: http://dx.doi.org/10.1214/15-AOS1321
12. Chen T., Guestrin C. XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. San Francisco, CA. ACM; 2016. P. 785–794. DOI: http://dx.doi.org/10.1145/2939672.2939785.
13. Cortes C., Vapnik V.N. Support-Vector Networks. Machine Learning. 1995. Vol. 20. P. 273–297. DOI: http://dx.doi.org/10.1007/BF00994018
14. Schölkopf B., Smola A. J. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge, MA: *MIT Press*, 2002, 626 p.
15. He J., Makarov R.N., Tuero J., Wang Z. Performance evaluation metric for statistical learning trading strategies. *Data Science in Finance and Economics*. 2024. Vol. 4, No. 4. P. 570–600. DOI: http://dx.doi.org/10.3934/DSFE.2024024

16. GeeksforGeeks. Understanding the Confusion Matrix in Machine Learning URL: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/ (Last accessed: 24.05.2025).
17. Zhu M. Recall, Precision and Average Precision. Technical Report, Department of Statistics and Actuarial Science, University of Waterloo, 2004. URL: https://www.researchgate.net/publication/228874142_Recall_precision_and_average_precision.
18. Yang C., Zhai J., Li H. The upper bound of cumulative return of a trading series. PLOS ONE. 2022. DOI: http://dx.doi.org/10.1371/journal.pone.0267239
19. Sharpe W. F. The Sharpe Ratio. Journal of Portfolio Management. 1994. Vol. 21, No. 1. P. 49–58. DOI: http://dx.doi.org/10.3905/jpm.1994.409501
20. NumPy Documentation. URL: https://numpy.org/doc/stable/ (Last accessed: 18.05.2025).
21. Pandas Documentation. URL: https://pandas.pydata.org/docs/ (Last accessed: 18.05.2025).
22. Scikit-learn User Guide. URL: https://scikit-learn.org/stable/user_guide.html (Last accessed: 19.05.2025).
23. XGBoost Documentation URL: https://xgboost.readthedocs.io/en/release_3.0.0/index.html (Last accessed: 18.05.2025).
24. Matplotlib User Guide. URL: https://matplotlib.org/stable/users/index (Last accessed: 18.05.2025).
25. Plotly Technologies Inc. Dash by Plotly. URL: https://dash.plotly.com (Last accessed: 18.05.2025).
26. Roussi R. YFinance – Yahoo! Finance Market Data Downloader: https://ranaroussi.github.io/yfinance/ (Last accessed: 18.05.2025).
27. Technical Analysis Library in Python Developers. URL: https://technical-analysis-library-in-python.readthedocs.io/en/latest/ (Last accessed: 18.05.2025).

28. SciPy Developers. SciPy Reference Guide. URL: https://docs.scipy.org/doc/scipy/reference/index.html (Last accessed: 18.05.2025).

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

bollinger_bands.py

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import ta
from scipy.signal import find_peaks
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from joblib import dump, load
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import TimeSeriesSplit, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, StratifiedKFold,cross_val_score
from sklearn.preprocessing import StandardScaler
from ta.volatility import BollingerBands
from ta.momentum import StochasticOscillator
from ta.trend import PSARIndicator
from xgboost import XGBClassifier, callback as xgb_callback
import optuna
from tqdm import tqdm
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score,
average_precision_score
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.ensemble import BalancedRandomForestClassifier
from optuna.samplers import TPESampler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from optuna_integration import TFKerasPruningCallback
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, precision_recall_curve, auc
btc_df = pd.read_csv('data/btcusdt_1m_2.csv')
btc_df.time = pd.to_datetime(btc_df.time, utc=True)
print(btc_df.time.max())
print(btc_df.time.min())
prices_full = btc_df['close'].values
peaks,  _ = find_peaks(prices_full,  distance=0.5 * 24 * 60, prominence=1000, width=3)
troughs, _ = find_peaks(-prices_full, distance=0.5 * 24 * 60, prominence=1000, width=3)
btc_df['is_peak']   = btc_df.index.to_series().isin(peaks)
btc_df['is_trough'] = btc_df.index.to_series().isin(troughs)
btc_df['is_reverse'] = (btc_df['is_peak'] | btc_df['is_trough'])
btc_df['is_peak_orig'] = btc_df['is_peak']
btc_df['is_trough_orig'] = btc_df['is_trough']
btc_df['is_peak'] = False
btc_df['is_trough'] = False
btc_df['is_reverse'] = False
for idx in btc_df.index[btc_df['is_peak_orig'] | btc_df['is_trough_orig']]:
    start = max(idx - 2, 0)
    end = min(idx + 10, len(btc_df))
```

```
    btc_df.loc[start:end, 'is_reverse'] = True
    if btc_df.at[idx, 'is_peak_orig']:
        btc_df.loc[start:end, 'is_peak'] = True
    if btc_df.at[idx, 'is_trough_orig']:
        btc_df.loc[start:end, 'is_trough'] = True
btc_df['signal'] = np.where(
    btc_df['is_peak'], 'sell',
    np.where(btc_df['is_trough'], 'buy', 'none')
)
def plot_1(df):
    temp = df.copy()
    fig = make_subplots(
        rows=2, cols=1, shared_xaxes=True,
        row_heights=[0.7, 0.3], vertical_spacing=0.02,
        subplot_titles=('Candlechart','Volume')
    )
    fig.add_trace(
        go.Candlestick(
            x=temp['time'], open=temp['open'], high=temp['high'],
            low=temp['low'], close=temp['close'],
            increasing_line_color='green',
            decreasing_line_color='red'
        ),
        row=1, col=1
    )
    troughs = temp[temp['is_trough'] == True]
    if not troughs.empty:
        fig.add_trace(
            go.Scatter(
                x=troughs['time'],
                y=troughs['low'] * 0.97,
                mode='markers',
                marker=dict(
                    symbol='triangle-up',
                    size=15,
                    color='green',
                    line=dict(width=2, color='darkgreen')
                ),
                name='Trough (is_trough)',
                hovertemplate='Time: %{x}<br>Price: %{y}<extra></extra>'
            ), row=1, col=1
        )
    peaks = temp[temp['is_peak'] == True]
    if not peaks.empty:
        fig.add_trace(
            go.Scatter(
                x=peaks['time'],
                y=peaks['high'] * 1.03,
                mode='markers',
                marker=dict(
                    symbol='triangle-down',
                    size=15,
                    color='red',
                    line=dict(width=2, color='darkred')
                ),
                name='Peak (is_peak)',
                hovertemplate='Time: %{x}<br>Price: %{y}<extra></extra>'
```

```
            ), row=1, col=1
        )
    fig.add_trace(go.Scatter(
        x=temp['time'],
        y=temp['volume'],
        mode='lines',
        line=dict(color='blue', width=1),
        name='Volume'
    ), row=2, col=1)
    fig.update_layout(
        title='Local minima/maxima over BB signal',
        width=1200, height=800, hovermode='x unified',
        xaxis_rangeslider_visible=False
    )
    fig.update_xaxes(type='date', row=1, col=1)
    fig.update_xaxes(type='date', row=2, col=1)
    fig.show()
plot_1(btc_df[btc_df.time >= '2025-04-08 20:09:00+00:00'])
bollinger = BollingerBands(
    close=btc_df['close'],
    window=20,
    window_dev=2,
    fillna=False
)
btc_df['bb_high']     = bollinger.bollinger_hband()
btc_df['bb_mid']      = bollinger.bollinger_mavg()
btc_df['bb_low']      = bollinger.bollinger_lband()
btc_df['bb_high'] = btc_df['bb_high'] - btc_df['bb_high'].rolling(window=24 * 60).mean()
btc_df['bb_mid'] = btc_df['bb_mid'] - btc_df['bb_mid'].rolling(window=24 * 60).mean()
btc_df['bb_low'] = btc_df['bb_low'] - btc_df['bb_low'].rolling(window=24 * 60).mean()
btc_df['bb_high_mean_1h'] = btc_df['bb_high'].rolling(window=60).mean()
btc_df['bb_mid_mean_1h']  = btc_df['bb_mid'].rolling(window=60).mean()
btc_df['bb_low_mean_1h']  = btc_df['bb_low'].rolling(window=60).mean()
btc_df.tail(10)
btc_df_train_part = 0.8
n_total   = len(btc_df)
n_train   = int(n_total * btc_df_train_part)
btc_df_train = btc_df.iloc[:n_train].copy()
btc_df_test  = btc_df.iloc[n_train:].copy()
btc_df.tail(10)
btc_df_test.time.max()
total_needed     = 5000
min_chunk_size   = 50
max_chunk_size   = 200
working_df = btc_df_train.reset_index(names='orig_idx')
mask_non_rev = ~working_df['is_reverse']
working_df['grp'] = mask_non_rev.ne(mask_non_rev.shift()).cumsum()
runs = working_df.loc[mask_non_rev].groupby('grp')
runs_info = runs.size().reset_index(name='length')
valid_runs = runs_info[
    (runs_info['length'] >= min_chunk_size)
].copy().sample(frac=1, random_state=42).reset_index(drop=True)
selected_chunks = []
remaining = total_needed
for _, row in valid_runs.iterrows():
    block = runs.get_group(row['grp'])
    L = row['length']
```

```
    if L > max_chunk_size:
        block = block.iloc[:max_chunk_size]
        L = max_chunk_size
    if L < min_chunk_size:
        continue
    take_n = min(L, remaining)
    selected_chunks.append(block.iloc[:take_n])
    remaining -= take_n
    if remaining == 0:
        break
non_reverse_chunks = pd.concat(selected_chunks)
only_true = working_df.loc[working_df['is_reverse']].copy()
train_df = pd.concat([only_true, non_reverse_chunks], ignore_index=True)
train_df = train_df.sort_values('orig_idx').drop(columns=['orig_idx', 'grp'])
train_df.dropna(inplace=True)
mask_col = ['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h', 'bb_mid_mean_1h', 'bb_low_mean_1h']
X_train = train_df[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h', 'bb_mid_mean_1h',
'bb_low_mean_1h']]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 500),
        'max_depth': trial.suggest_int('max_depth', 2, 30),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 20),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', None]),
        'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy']),
    }
    clf = RandomForestClassifier(**params, random_state=42, n_jobs=-1)
    tscv = TimeSeriesSplit(n_splits=3)
    precision_macro_scorer = make_scorer(
    precision_score,
    average='macro',
    zero_division=0
)
    scores = cross_val_score(
        clf,
        X_train,
        y_train_enc,
        cv=tscv,
        scoring=precision_macro_scorer,
        n_jobs=-1
    )
    return scores.mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
def plot_classification_report(y_true, y_pred):
    report = classification_report(y_true, y_pred, output_dict=True)
    df = pd.DataFrame(report).T
    fig, ax = plt.subplots(figsize=(8, 6), facecolor='white')
    ax.axis('off')
    tbl = ax.table(
        cellText=np.round(df.values, 2),
        rowLabels=df.index,
        colLabels=df.columns,
```

```
        cellLoc='center',
        loc='center'
    )
    tbl.auto_set_font_size(False)
    tbl.set_fontsize(12)
    tbl.scale(1, 1.5)
    plt.tight_layout()
    plt.show()
best_model = load('models/bollinger_bands/random_forest/model_btc_test.joblib')
X_test = btc_df_test[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h', 'bb_mid_mean_1h',
'bb_low_mean_1h']]
y_test = btc_df_test['signal']
y_test_enc  = y_test.map(mapping)
y_pred = best_model.predict(X_test)
y_probas = best_model.predict_proba(X_test)
inv_mapping = {v: k for k, v in mapping.items()}
y_test_decoded = y_test_enc.map(inv_mapping)
y_pred_decoded = pd.Series(y_pred, index=y_test_enc.index).map(inv_mapping)
print(classification_report(y_test_decoded, y_pred_decoded))
proba_all = best_model.predict_proba(btc_df_test[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h',
'bb_mid_mean_1h', 'bb_low_mean_1h']])
pred = best_model.predict(btc_df_test[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h',
'bb_mid_mean_1h', 'bb_low_mean_1h']])
proba_class_0 = proba_all[:, 0]
proba_class_1 = proba_all[:, 1]
proba_class_2 = proba_all[:, 2]
btc_df_test['true_proba1'] = proba_class_1
btc_df_test['true_proba2'] = proba_class_2
btc_df_test['predicted_signal'] = np.where(btc_df_test['true_proba1'] >= 0.9, 'sell',
                                  np.where(btc_df_test['true_proba2'] >= 0.9, 'buy', None))
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
btc_df_high_conf.predicted_signal.value_counts()
def plot_res(df, start_time, end_time):
    temp = df[(df.time > start_time) & (df.time < end_time)].copy()
    fig = make_subplots(
        rows=1, cols=1,
        shared_xaxes=True,
        vertical_spacing=0.02,
        subplot_titles=('Candlechart',)
    )
    fig.add_trace(
        go.Candlestick(
            x=temp['time'], open=temp['open'], high=temp['high'],
            low=temp['low'], close=temp['close'],
            increasing_line_color='green',
            decreasing_line_color='red'
        ),
        row=1, col=1
    )
    buy_signals = temp[temp['predicted_signal_clean'] == 'buy']
    if not buy_signals.empty:
        fig.add_trace(
            go.Scatter(
                x=buy_signals['time'],
                y=buy_signals['low'] * 0.97,
                mode='markers',
                marker=dict(
```

```
                symbol='triangle-up',
                size=15,
                color='green',
                line=dict(width=2, color='darkgreen')
            ),
            name='Buy Signal',
            customdata=buy_signals['true_proba2'],
            hovertemplate=(
                'PredictProbe1: %{customdata}<br>'
                'Time: %{x}<br>'
                'Price: %{y}<extra></extra>'
            )
        ), row=1, col=1
    )
sell_signals = temp[temp['predicted_signal_clean'] == 'sell']
if not sell_signals.empty:
    fig.add_trace(
        go.Scatter(
            x=sell_signals['time'],
            y=sell_signals['high'] * 1.03,
            mode='markers',
            marker=dict(
                symbol='triangle-down',
                size=15,
                color='red',
                line=dict(width=2, color='darkred')
            ),
            name='Sell Signal',
            customdata=sell_signals['true_proba1'],
            hovertemplate=(
                'PredictProbe1: %{customdata}<br>'
                'Time: %{x}<br>'
                'Price: %{y}<extra></extra>'
            )
        ), row=1, col=1
    )
fig.update_layout(
    title='Predicted singals',
    width=1200, height=800, hovermode='x unified',
    xaxis_rangeslider_visible=False
)
fig.update_xaxes(type='date', row=1, col=1)
fig.show()
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
```

```
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import roc_auc_score
roc_auc_macro = roc_auc_score(y_test_enc, y_probas, multi_class="ovo", average="macro")
roc_auc_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
X_train = train_df[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h', 'bb_mid_mean_1h',
'bb_low_mean_1h']]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    params = {
        'n_estimators':       trial.suggest_int('n_estimators', 50, 500),
        'max_depth':          trial.suggest_int('max_depth', 3, 15),
        'learning_rate':      trial.suggest_float('learning_rate', 1e-3, 0.3, log=True),
        'subsample':          trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree':   trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'gamma':              trial.suggest_float('gamma', 0.0, 5.0),
        'reg_alpha':          trial.suggest_float('reg_alpha', 1e-8, 1.0, log=True),
        'reg_lambda':         trial.suggest_float('reg_lambda', 1e-8, 1.0, log=True),
        'eval_metric':        'mlogloss',
        'verbosity':          0
    }
    clf = XGBClassifier(**params, random_state=42, n_jobs=-1)
    tscv = TimeSeriesSplit(n_splits=3)
    scores = cross_val_score(
        clf,
        X_train,
        y_train_enc,
        cv=tscv,
        scoring='precision_macro',
        n_jobs=-1
    )
```

```
    return scores.mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
print("Лучшие гиперпараметры:", study.best_params)
best_params = study.best_params.copy()
clf_best = XGBClassifier(**best_params, random_state=42, n_jobs=-1)
clf_best.fit(X_train, y_train_enc)
X_test = btc_df_test[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h', 'bb_mid_mean_1h',
'bb_low_mean_1h']]
y_test = btc_df_test['signal']
y_test_enc  = y_test.map(mapping)
y_pred = clf_best.predict(X_test)
y_probas = clf_best.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred))
X_test = btc_df_test[['bb_high', 'bb_mid', 'bb_low',
                      'bb_high_mean_1h', 'bb_mid_mean_1h', 'bb_low_mean_1h']]
proba_all   = clf_best.predict_proba(X_test)
btc_df_test['true_proba0'] = proba_all[:, 0]
btc_df_test['true_proba1'] = proba_all[:, 1]
btc_df_test['true_proba2']  = proba_all[:, 2]
threshold = 0.72
btc_df_test['predicted_signal'] = None
btc_df_test.loc[btc_df_test['true_proba1']  >= threshold, 'predicted_signal'] = 'buy'
mask_sell = btc_df_test['predicted_signal'].isna() & (btc_df_test['true_proba2'] >= threshold)
btc_df_test.loc[mask_sell, 'predicted_signal'] = 'sell'
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
print(btc_df_high_conf['predicted_signal'].value_counts())
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
```

```
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
X_train = train_df[['bb_high', 'bb_mid', 'bb_low', 'bb_high_mean_1h', 'bb_mid_mean_1h',
'bb_low_mean_1h']]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    C     = trial.suggest_float('C', 1e-2, 100, log=True)
    gamma = trial.suggest_float('gamma', 1e-3, 1, log=True)
    model = Pipeline([
        ('scale', StandardScaler()),
        ('svc',   SVC(C=C, gamma=gamma, kernel='rbf',
                  class_weight='balanced', random_state=42))
    ])
    cv = TimeSeriesSplit(n_splits=5)
    score = cross_val_score(model, X_train, y_train,
                        cv=cv,
                        scoring='precision_macro',
                        n_jobs=-1).mean()
    return score
sampler = TPESampler(seed=42)
study   = optuna.create_study(direction='maximize', sampler=sampler)
study.optimize(objective, n_trials=100, timeout=600)
print('Лучший trial:', study.best_trial.number)
print('Лучший precision_macro  :', study.best_trial.value)
print('Параметры  :', study.best_params)
svc_pipe = load('models/bollinger_bands/svc/btc_svc_scaled_test.pkl')
X_test  = btc_df_test[['bb_high', 'bb_mid', 'bb_low',
                   'bb_high_mean_1h', 'bb_mid_mean_1h', 'bb_low_mean_1h']]
y_test  = btc_df_test['signal']
y_test_enc = y_test.map(mapping)
y_pred = svc_pipe.predict(X_test)
y_probas = svc_pipe.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred, target_names=['none','sell','buy']))
proba_all  =  svc_pipe.predict_proba(btc_df_test[['bb_high',  'bb_mid',  'bb_low',  'bb_high_mean_1h',
'bb_mid_mean_1h', 'bb_low_mean_1h']])
proba_class_0 = proba_all[:, 0]
proba_class_1 = proba_all[:, 1]
proba_class_2 = proba_all[:, 2]
btc_df_test['true_proba1'] = proba_class_1
btc_df_test['true_proba2'] = proba_class_2
btc_df_test['predicted_signal'] = np.where(btc_df_test['true_proba1'] >= 0.8, 'sell',
                                np.where(btc_df_test['true_proba2'] >= 0.8, 'buy', None))
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
btc_df_high_conf.predicted_signal.value_counts()
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
```

```
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import roc_auc_score
roc_auc_macro = roc_auc_score(y_test_enc, y_probas, multi_class="ovo", average="macro")
roc_auc_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(y_test_enc, y_pred)
mcc
class Backtest:
    def __init__(self, model, test_market_data, start_balance):
        self.model = model
        self.test_market_data = test_market_data.copy()
        self.trades = pd.DataFrame({
            'open_time': pd.Series([], dtype='datetime64[ns, UTC]'),
            'open_price': pd.Series([], dtype='float'),
            'close_price': pd.Series([], dtype='float'),
            'close_time': pd.Series([], dtype='datetime64[ns, UTC]'),
            'side': pd.Series([], dtype='str'),
        })
        self.balance = start_balance
        self.balance_history = pd.DataFrame({
                'trade_id': pd.Series([], dtype='int'),
```

```
            'time': pd.Series([], dtype='datetime64[ns, Etc/GMT+5]'),
            'balance': pd.Series([], dtype='float'),
            'affect': pd.Series([], dtype='str'),
            'coef': pd.Series([], dtype='float')
            })
    def predict(self, min_probability):
        proba_all = self.model.predict_proba(self.test_market_data[mask_col])
        proba_class_1 = proba_all[:, 1]
        proba_class_2 = proba_all[:, 2]
        self.test_market_data['true_proba1'] = proba_class_1
        self.test_market_data['true_proba2'] = proba_class_2
        self.test_market_data['predicted_signal'] = np.where(self.test_market_data['true_proba1'] >= min_probability, 'sell',
                                np.where(self.test_market_data['true_proba2'] >= min_probability, 'buy', None))
        threshold = pd.Timedelta(hours=3)
        self.test_market_data['predicted_signal_clean'] = self.test_market_data['predicted_signal']
        for sig in ['buy', 'sell']:
            mask = self.test_market_data['predicted_signal_clean'] == sig
            times = self.test_market_data.loc[mask, 'time']
            dt = times.diff()
            drop_idx = dt[dt <= threshold].index
            self.test_market_data.loc[drop_idx, 'predicted_signal_clean'] = None
    def calculate_trades(self, target_close):
        self.trades = self.trades.iloc[0:0]
        mask_predicted_signal = self.test_market_data.predicted_signal_clean.isin(['sell', 'buy'])
        for index, row in tqdm(self.test_market_data[mask_predicted_signal].iterrows(), total=self.test_market_data[mask_predicted_signal].shape[0]):
            open_time = row.time
            open_price = row.open
            end_time = open_time + pd.Timedelta(hours=2)
            future = self.test_market_data[
                (self.test_market_data.time > open_time) &
                (self.test_market_data.time <= end_time)
            ]
            if row.predicted_signal_clean == 'buy':
                target = open_price * (1 + target_close/100)
                hit = future[future.high >= target]
            else:
                target = open_price * (1 - target_close/100)
                hit = future[future.low <= target]
            if not hit.empty:
                close_price = target
                close_time = hit.iloc[0].time
            else:
                if not future.empty:
                    close_price = future.iloc[-1].close
                    close_time = future.iloc[-1].time
                else:
                    close_price = open_price
                    close_time = open_time
            new_row = {
                'open_time': open_time,
                'open_price': open_price,
                'close_time': close_time,
                'close_price': close_price,
                'side': row.predicted_signal_clean
```

```
            }
            self.trades = pd.concat(
                [self.trades, pd.DataFrame([new_row], columns=self.trades.columns)],
                ignore_index=True
            )
    def plot(self, min_datetime, max_datetime):
            plot_res(self.test_market_data, min_datetime, max_datetime)
model1 = load('models/bollinger_bands/xgboost/btc_xgb_test.joblib')
test = Backtest(model1, btc_df_test, 1000)
test.predict(min_probability=0.67)
test.calculate_trades(target_close=2)
trades = test.trades.copy()
trades = trades.rename(columns={
    'side':       'side',
    'open_time':   'time_open',
    'open_price':  'price_open',
    'close_time':  'time_close',
    'close_price': 'price_close'
})
trades['symbol']          = 'BTC'
trades['order_weight']    = 1
trades['is_maker_open']   = False
trades['is_maker_close']  = False
trades['amount']          = 0.0005
trades['volume_usd']      = trades['amount'] * trades['price_open']
direction = np.where(trades['side'].str.lower() == 'buy', 1, -1)
trades['pl_pct'] = direction * (trades['price_close'] - trades['price_open']) / trades['price_open']
trades['pl_usd'] = trades['pl_pct'] * trades['volume_usd'] * np.where(
    trades['side'].str.lower() == 'buy',
    1,
    -1
)
class PortfolioAnalysis:
    required_columns = [
        'symbol', 'side', 'time_open', 'price_open',
        'time_close', 'price_close', 'order_weight',
        'is_maker_open', 'is_maker_close','volume_usd','amount','pl_usd'
    ]
    def __init__(self, df):
        self.validate_data(df)
        self.df = df.copy()
        self.df['time_open'] = pd.to_datetime(self.df['time_open'])
        if 'time_close' in self.df.columns:
            self.df['time_close'] = pd.to_datetime(self.df['time_close'])
    def validate_data(self, df):
        missing_columns = [col for col in self.required_columns if col not in df.columns]
        if missing_columns:
            raise ValueError(f"Input DataFrame is missing the following required columns: {',
'.join(missing_columns)}")
    def resample_data(self, period):
        df_copy = self.df.copy()
        df_copy.set_index('time_open', inplace=True)
        result = df_copy.resample(period).agg({'pl_usd': 'sum'}).reset_index()
        return result
    def days_convert(self):
        return self.resample_data('D')
    def months_convert(self):
```

```
        return self.resample_data('M')
    def years_convert(self):
        return self.resample_data('Y')
    def profit_sum(self):
        return round(self.df['pl_usd'].sum(), 3)
    def max_drawdown(self, df=None):
        if df is None:
            df = self.df.copy()
        df['cumulative_sum_pl'] = df['pl_usd'].cumsum()
        df['drawdown'] = df['cumulative_sum_pl'] - df['cumulative_sum_pl'].cummax()
        max_drawdown_value = df['drawdown'].min()
        return round(max_drawdown_value, 3)
    def max_daily_drawdown(self):
        daily_data = self.days_convert()
        return self.max_drawdown(daily_data)
    def win_rate_deals(self):
        win_rate = 100 * len(self.df[self.df['pl_usd'] > 0]) / len(self.df)
        return round(win_rate, 3)
    def win_rate_days(self):
        daily_data = self.days_convert()
        win_rate = 100 * len(daily_data[daily_data['pl_usd'] > 0]) / len(daily_data)
        return round(win_rate, 3)
    def average_profit(self, df=None, column='pl_usd'):
        if df is None:
            df = self.df
        return round(df[df[column] > 0][column].mean(), 3)
    def average_loss(self, df=None, column='pl_usd'):
        if df is None:
            df = self.df
        return round(df[df[column] < 0][column].mean(), 3)
    def sharpe_ratio(self):
        monthly_returns = self.days_convert()['pl_usd']
        average_return = np.mean(monthly_returns)
        standard_deviation = np.std(monthly_returns)
        sharpe_ratio_value = average_return / standard_deviation * np.sqrt(12)
        return round(sharpe_ratio_value, 3)
    def market_limit(self):
        results = {}
        columns = {
            'is_maker_open': 'maker_open',
            'is_maker_close': 'maker_close'
        }
        for column, column_name in columns.items():
            true_count = self.df[column].sum()
            false_count = len(self.df) - true_count
            total_count = len(self.df)
            true_percentage = (true_count / total_count) * 100
            false_percentage = (false_count / total_count) * 100
            results[column_name] = {
                'True': round(true_percentage, 2),
                'False': round(false_percentage, 2)
            }
        formatted_result = f"maker_open {results['maker_open']['True']}% maker close
{results['maker_close']['True']}%"
        return formatted_result
    def longest_drawdown_period(self):
        df = self.days_convert()
```

```
        df['cumulative_sum_pl'] = df['pl_usd'].cumsum()
        df['drawdown'] = df['cumulative_sum_pl'] - df['cumulative_sum_pl'].cummax()
        df['in_drawdown'] = df['drawdown'] < 0
        longest_drawdown_period = 0
        current_drawdown_start = None
        for index, row in df.iterrows():
            if row['in_drawdown']:
                if current_drawdown_start is None:
                    current_drawdown_start = row['time_open']
            else:
                if current_drawdown_start is not None:
                    drawdown_period = row['time_open'] - current_drawdown_start
                    longest_drawdown_period = max(longest_drawdown_period, drawdown_period.days)
                    current_drawdown_start = None
        if current_drawdown_start is not None:
            drawdown_period = df['time_open'].iloc[-1] - current_drawdown_start
            longest_drawdown_period = max(longest_drawdown_period, drawdown_period.days)
        return longest_drawdown_period
    def average_deal_time(self):
        self.df['deal_time'] = (self.df['time_close'] - self.df['time_open']).dt.total_seconds() / 60
        return round(self.df['deal_time'].mean(), 3)
    def average_deals_per_day(self):
        first_date = self.df['time_open'].min()
        last_date  = self.df['time_open'].max()
        total_days = (last_date.normalize() - first_date.normalize()).days + 1
        total_deals = self.df.shape[0]
        return round(total_deals / total_days, 3)
    def average_deals_per_month(self):
        first_date = self.df['time_open'].min()
        last_date  = self.df['time_open'].max()
        total_months = (last_date.year - first_date.year) * 12 + (last_date.month - first_date.month) + 1
        total_deals = self.df.shape[0]
        return round(total_deals / total_months, 3)
    def start_end_backtest(self):
        start = self.df['time_open'].iloc[0]
        end = self.df['time_open'].iloc[-1]
        duration = end - start
        return f"{start}\nBacktest end: {end}\nBacktest duration: {duration}"
    def stats(self):
        print('Max profit:', f'{self.profit_sum()}%')
        print('Max drawdown:', f'{self.max_drawdown()}%')
        print('Max daily drawdown:', f'{self.max_daily_drawdown()}%')
        print('Average deal profit:', f'{self.average_profit()}%')
        print('Average deal loss:', f'{self.average_loss()}%')
        print('Average daily profit:', f'{self.average_profit(self.days_convert())}%')
        print('Average daily loss:', f'{self.average_loss(self.days_convert())}%')
        print('Deals win rate:', f'{self.win_rate_deals()}%')
        print('Sharpe ratio:', self.sharpe_ratio())
        print('Longest drawdown period:', f'{self.longest_drawdown_period()} days')
        print('Backtest start:', self.start_end_backtest())
        print('Average Deals per day:', self.average_deals_per_day())
        print('Average Deals per month:', self.average_deals_per_month())
analysis = PortfolioAnalysis(trades)
analysis.stats()
temp = trades.sort_values('time_open')
temp['cum_pl_usd'] = temp['pl_usd'].cumsum()
plt.figure(figsize=(10, 6))
```

```
plt.plot(temp['time_open'], temp['cum_pl_usd'])
plt.xlabel('Time Open')
plt.ylabel('Cumulative P&L %')
plt.title('Cumulative P&L over Time')
plt.tight_layout()
plt.show()
def economic_analysis(model_name, min_probability, close_time):
    model1 = load(model_name)
    test = Backtest(model1, btc_df_test, 1000)
    test.predict(min_probability=min_probability)
    test.calculate_trades(target_close=close_time)
    trades = test.trades.copy()
    trades = trades.rename(columns={
        'side':       'side',
        'open_time':   'time_open',
        'open_price':  'price_open',
        'close_time':  'time_close',
        'close_price': 'price_close'
    })
    trades['symbol']          = 'BTC'
    trades['order_weight']    = 1
    trades['is_maker_open']   = False
    trades['is_maker_close']  = True
    trades['amount']          = 0.0005
    trades['volume_usd']      = trades['amount'] * trades['price_open']
    trades['pl_pct']          = (trades.price_close - trades.price_open) / trades.price_open
    trades['pl_usd'] = trades['pl_pct'] * trades['volume_usd'] * np.where(
        trades['side'].str.lower() == 'buy',
        1,
        -1
    )
    analysis = PortfolioAnalysis(trades)
    analysis.stats()
    temp = trades.sort_values('time_open')
    temp['cum_pl_usd'] = temp['pl_usd'].cumsum()
    plt.figure(figsize=(10, 6))
    plt.plot(temp['time_open'], temp['cum_pl_usd'])
    plt.xlabel('Time Open')
    plt.ylabel('Cumulative P&L %')
    plt.title('Cumulative P&L over Time')
    plt.tight_layout()
    plt.show()
    display(trades)
economic_analysis(model_name='models/bollinger_bands/random_forest/model_btc_test.joblib',
min_probability=0.8, close_time=1)
economic_analysis(model_name='models/bollinger_bands/xgboost/btc_xgb_test.joblib',
min_probability=0.72, close_time=2)
economic_analysis(model_name='models/bollinger_bands/svc/btc_svc_scaled_test.pkl',
min_probability=0.8, close_time=2)
```

stoh_osc.py

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
import ta
from scipy.signal import find_peaks
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from joblib import dump, load
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import TimeSeriesSplit, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, StratifiedKFold,cross_val_score
from sklearn.preprocessing import StandardScaler
from ta.volatility import BollingerBands
from xgboost import XGBClassifier, callback as xgb_callback
import optuna
from tqdm import tqdm
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score,
average_precision_score
from sklearn.metrics import roc_curve, precision_recall_curve, auc
from sklearn.preprocessing import label_binarize
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.ensemble import BalancedRandomForestClassifier
from optuna.samplers import TPESampler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from ta.momentum import StochasticOscillator
btc_df = pd.read_csv('data/btcusdt_1m_2.csv')
btc_df.time = pd.to_datetime(btc_df.time, utc=True)
prices_full = btc_df['close'].values
peaks,  _ = find_peaks(prices_full,  distance=0.5 * 24 * 60, prominence=1000, width=3)
troughs, _ = find_peaks(-prices_full, distance=0.5 * 24 * 60, prominence=1000, width=3)
btc_df['is_peak']   = btc_df.index.to_series().isin(peaks)
btc_df['is_trough'] = btc_df.index.to_series().isin(troughs)
btc_df['is_reverse'] = (btc_df['is_peak'] | btc_df['is_trough'])
btc_df['is_peak_orig'] = btc_df['is_peak']
btc_df['is_trough_orig'] = btc_df['is_trough']
btc_df['is_peak'] = False
btc_df['is_trough'] = False
btc_df['is_reverse'] = False
for idx in btc_df.index[btc_df['is_peak_orig'] | btc_df['is_trough_orig']]:
    start = max(idx - 2, 0)
    end = min(idx + 10, len(btc_df))
    btc_df.loc[start:end, 'is_reverse'] = True
    if btc_df.at[idx, 'is_peak_orig']:
        btc_df.loc[start:end, 'is_peak'] = True
    if btc_df.at[idx, 'is_trough_orig']:
        btc_df.loc[start:end, 'is_trough'] = True
btc_df['signal'] = np.where(
    btc_df['is_peak'], 'sell',
    np.where(btc_df['is_trough'], 'buy', 'none')
)
stoch = StochasticOscillator(
    high=btc_df['high'],
    low=btc_df['low'],
    close=btc_df['close'],
    window=60,
    smooth_window=13,
```

```
    fillna=False
)
btc_df['stoch_k'] = stoch.stoch()
btc_df['stoch_d'] = stoch.stoch_signal()
btc_df[f'stoch_k_lag1'] = btc_df['stoch_k'].shift(1)
btc_df[f'stoch_d_lag1'] = btc_df['stoch_d'].shift(1)
btc_df[f'stoch_k_lag2'] = btc_df['stoch_k'].shift(2)
btc_df[f'stoch_d_lag2'] = btc_df['stoch_d'].shift(2)
btc_df['stoch_k_diff1'] = btc_df['stoch_k'] - btc_df['stoch_k_lag1']
btc_df['stoch_d_diff1'] = btc_df['stoch_d'] - btc_df['stoch_d_lag1']
btc_df['stoch_k_diff2'] = btc_df['stoch_k'] - btc_df['stoch_k_lag2']
btc_df['stoch_d_diff2'] = btc_df['stoch_d'] - btc_df['stoch_d_lag2']
btc_df_train_part = 0.8
n_total   = len(btc_df)
n_train   = int(n_total * btc_df_train_part)
btc_df_train = btc_df.iloc[:n_train].copy()
btc_df_test  = btc_df.iloc[n_train:].copy()
total_needed     = 5000
min_chunk_size   = 50
max_chunk_size   = 200
working_df = btc_df_train.reset_index(names='orig_idx')
mask_non_rev = ~working_df['is_reverse']
working_df['grp'] = mask_non_rev.ne(mask_non_rev.shift()).cumsum()
runs = working_df.loc[mask_non_rev].groupby('grp')
runs_info = runs.size().reset_index(name='length')
valid_runs = runs_info[
    (runs_info['length'] >= min_chunk_size)
].copy().sample(frac=1, random_state=42).reset_index(drop=True)
selected_chunks = []
remaining = total_needed
for _, row in valid_runs.iterrows():
    block = runs.get_group(row['grp'])
    L = row['length']
    if L > max_chunk_size:
        block = block.iloc[:max_chunk_size]
        L = max_chunk_size
    if L < min_chunk_size:
        continue
    take_n = min(L, remaining)
    selected_chunks.append(block.iloc[:take_n])
    remaining -= take_n
    if remaining == 0:
        break
non_reverse_chunks = pd.concat(selected_chunks)
only_true = working_df.loc[working_df['is_reverse']].copy()
train_df = pd.concat([only_true, non_reverse_chunks], ignore_index=True)
train_df = train_df.sort_values('orig_idx').drop(columns=['orig_idx', 'grp'])
train_df.dropna(inplace=True)
mask_col = ['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']
X_train = train_df[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    params = {
```

```
        'n_estimators': trial.suggest_int('n_estimators', 50, 500),
        'max_depth': trial.suggest_int('max_depth', 2, 30),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 20),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', None]),
        'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy']),
    }
    clf = RandomForestClassifier(**params, random_state=42, n_jobs=-1)
    tscv = TimeSeriesSplit(n_splits=3)
    precision_macro_scorer = make_scorer(
    precision_score,
    average='macro',
    zero_division=0
)
    scores = cross_val_score(
        clf,
        X_train,
        y_train_enc,
        cv=tscv,
        scoring=precision_macro_scorer,
        n_jobs=-1
    )
    return scores.mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
best_model = load('models/stoh_osc/random_forest/model_test_btc.joblib')
best_model.fit(X_train, y_train_enc)
X_test = btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_test = btc_df_test['signal']
y_test_enc  = y_test.map(mapping)
y_pred = best_model.predict(X_test)
y_probas = best_model.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred))
proba_all = best_model.predict_proba(btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1',
'stoch_k_lag2', 'stoch_d_lag2', 'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']])
pred = best_model.predict(btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2',
'stoch_d_lag2', 'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']])
proba_class_0 = proba_all[:, 0]
proba_class_1 = proba_all[:, 1]
proba_class_2 = proba_all[:, 2]
btc_df_test['true_proba1'] = proba_class_1
btc_df_test['true_proba2'] = proba_class_2
btc_df_test['predicted_signal'] = np.where(btc_df_test['true_proba1'] >= 0.85, 'sell',
                                np.where(btc_df_test['true_proba2'] >= 0.85, 'buy', None))
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
btc_df_high_conf.predicted_signal.value_counts()
def plot_res(df, start_time, end_time):
    temp = df[(df.time > start_time) & (df.time < end_time)].copy()
    fig = make_subplots(
        rows=1, cols=1,
        shared_xaxes=True,
        vertical_spacing=0.02,
        subplot_titles=('Candlechart',)
    )
    fig.add_trace(
        go.Candlestick(
```

```
        x=temp['time'], open=temp['open'], high=temp['high'],
        low=temp['low'], close=temp['close'],
        increasing_line_color='green',
        decreasing_line_color='red'
    ),
    row=1, col=1
)
buy_signals = temp[temp['predicted_signal_clean'] == 'buy']
if not buy_signals.empty:
    fig.add_trace(
        go.Scatter(
            x=buy_signals['time'],
            y=buy_signals['low'] * 0.97,
            mode='markers',
            marker=dict(
                symbol='triangle-up',
                size=15,
                color='green',
                line=dict(width=2, color='darkgreen')
            ),
            name='Buy Signal',
            customdata=buy_signals['true_proba2'],
            hovertemplate=(
                'PredictProbe1: %{customdata}<br>'
                'Time: %{x}<br>'
                'Price: %{y}<extra></extra>'
            )
        ), row=1, col=1
    )
sell_signals = temp[temp['predicted_signal_clean'] == 'sell']
if not sell_signals.empty:
    fig.add_trace(
        go.Scatter(
            x=sell_signals['time'],
            y=sell_signals['high'] * 1.03,
            mode='markers',
            marker=dict(
                symbol='triangle-down',
                size=15,
                color='red',
                line=dict(width=2, color='darkred')
            ),
            name='Sell Signal',
            customdata=sell_signals['true_proba1'],
            hovertemplate=(
                'PredictProbe1: %{customdata}<br>'
                'Time: %{x}<br>'
                'Price: %{y}<extra></extra>'
            )
        ), row=1, col=1
    )
fig.update_layout(
    title='Predicted singals',
    width=1200, height=800, hovermode='x unified',
    xaxis_rangeslider_visible=False
)
fig.update_xaxes(type='date', row=1, col=1)
```

```
    fig.show()
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
y_test_enc.value_counts()
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import roc_auc_score
roc_auc_macro = roc_auc_score(y_test_enc, y_probas, multi_class="ovo", average="macro")
roc_auc_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(y_test_enc, y_pred)
mcc
X_train = train_df[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    params = {
        'n_estimators':      trial.suggest_int('n_estimators', 50, 500),
        'max_depth':         trial.suggest_int('max_depth', 3, 15),
```

```
        'learning_rate':       trial.suggest_float('learning_rate', 1e-3, 0.3, log=True),
        'subsample':           trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree':    trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'gamma':               trial.suggest_float('gamma', 0.0, 5.0),
        'reg_alpha':           trial.suggest_float('reg_alpha', 1e-8, 1.0, log=True),
        'reg_lambda':          trial.suggest_float('reg_lambda', 1e-8, 1.0, log=True),
        'eval_metric':         'mlogloss',
        'verbosity':           0
    }
    clf = XGBClassifier(**params, n_jobs=-1)
    tscv = TimeSeriesSplit(n_splits=3)
    scores = cross_val_score(
        clf,
        X_train,
        y_train_enc,
        cv=tscv,
        scoring='precision_macro',
        n_jobs=-1
    )
    return scores.mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
clf_best = load('models/stoh_osc/xgboost/btc_xgb_test.joblib')
X_test = btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_test = btc_df_test['signal']
y_test_enc  = y_test.map(mapping)
y_pred = clf_best.predict(X_test)
y_probas = clf_best.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred))
X_test = btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
proba_all   = clf_best.predict_proba(X_test)
btc_df_test['true_proba0'] = proba_all[:, 0]
btc_df_test['true_proba1'] = proba_all[:, 1]
btc_df_test['true_proba2']  = proba_all[:, 2]
threshold = 0.8
btc_df_test['predicted_signal'] = None
btc_df_test.loc[btc_df_test['true_proba2']  >= threshold, 'predicted_signal'] = 'buy'
mask_sell = btc_df_test['predicted_signal'].isna() & (btc_df_test['true_proba1'] >= threshold)
btc_df_test.loc[mask_sell, 'predicted_signal'] = 'sell'
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
print(btc_df_high_conf['predicted_signal'].value_counts())
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
```

```
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import roc_auc_score
roc_auc_macro = roc_auc_score(y_test_enc, y_probas, multi_class="ovo", average="macro")
roc_auc_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(y_test_enc, y_pred)
mcc
X_train = train_df[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
y_test_enc  = y_test.map(mapping)
def objective(trial):
    C     = trial.suggest_float('C', 1e-2, 100, log=True)
    gamma = trial.suggest_float('gamma', 1e-3, 1, log=True)
    model = Pipeline([
        ('scale', StandardScaler()),
        ('svc',   SVC(C=C, gamma=gamma, kernel='rbf',
                  class_weight='balanced', random_state=42))
    ])
    cv = TimeSeriesSplit(n_splits=5)
    score = cross_val_score(model, X_train, y_train,
                        cv=cv,
                        scoring='precision_macro',
                        n_jobs=-1).mean()
    return score
sampler = TPESampler(seed=42)
study   = optuna.create_study(direction='maximize', sampler=sampler)
study.optimize(objective, n_trials=100, timeout=600)
print('Лучший trial:', study.best_trial.number)
print('Лучший precision_macro  :', study.best_trial.value)
```

```
print('Параметры  :', study.best_params)
svc_pipe = load('models/stoh_osc/svc/btc_svc_scaled_test.pkl')
X_test  = btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_test  = btc_df_test['signal']
y_test_enc = y_test.map(mapping)
y_pred = svc_pipe.predict(X_test)
print(classification_report(y_test_enc, y_pred, target_names=['none','sell','buy']))
svc_pipe = load('models/stoh_osc/svc/btc_svc_scaled_test.pkl')
X_test  = btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2', 'stoch_d_lag2',
'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']]
y_test  = btc_df_test['signal']
y_test_enc = y_test.map(mapping)
y_pred = svc_pipe.predict(X_test)
y_probas = svc_pipe.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred, target_names=['none','sell','buy']))
proba_all = svc_pipe.predict_proba(btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1',
'stoch_k_lag2', 'stoch_d_lag2', 'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']])
pred = svc_pipe.predict(btc_df_test[['stoch_k', 'stoch_d', 'stoch_k_lag1', 'stoch_d_lag1', 'stoch_k_lag2',
'stoch_d_lag2', 'stoch_k_diff1', 'stoch_d_diff1', 'stoch_k_diff2', 'stoch_d_diff2']])
proba_class_0 = proba_all[:, 0]
proba_class_1 = proba_all[:, 1]
proba_class_2 = proba_all[:, 2]
btc_df_test['true_proba1'] = proba_class_1
btc_df_test['true_proba2'] = proba_class_2
btc_df_test['predicted_signal'] = np.where(btc_df_test['true_proba1'] >= 0.8, 'sell',
                                np.where(btc_df_test['true_proba2'] >= 0.8, 'buy', None))
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
btc_df_high_conf.predicted_signal.value_counts()
btc_df_high_conf
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-09-01 00:00:00', '2024-10-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
```

```
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import roc_auc_score
roc_auc_macro = roc_auc_score(y_test_enc, y_probas, multi_class="ovo", average="macro")
roc_auc_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(y_test_enc, y_pred)
mcc
btc_df_test.columns
class Backtest:
    def __init__(self, model, test_market_data, start_balance):
        self.model = model
        self.test_market_data = test_market_data.copy()
        self.trades = pd.DataFrame({
            'open_time': pd.Series([], dtype='datetime64[ns, UTC]'),
            'open_price': pd.Series([], dtype='float'),
            'close_price': pd.Series([], dtype='float'),
            'close_time': pd.Series([], dtype='datetime64[ns, UTC]'),
            'side': pd.Series([], dtype='str'),
        })
        self.balance = start_balance
        self.balance_history = pd.DataFrame({
                'trade_id': pd.Series([], dtype='int'),
                'time': pd.Series([], dtype='datetime64[ns, Etc/GMT+5]'),
                'balance': pd.Series([], dtype='float'),
                'affect': pd.Series([], dtype='str'),
                'coef': pd.Series([], dtype='float')
                })
    def predict(self, min_probability):
        proba_all = self.model.predict_proba(self.test_market_data[mask_col])
        proba_class_0 = proba_all[:, 0]
        proba_class_1 = proba_all[:, 1]
        proba_class_2 = proba_all[:, 2]
        self.test_market_data['true_proba0'] = proba_class_0
        self.test_market_data['true_proba1'] = proba_class_1
        self.test_market_data['true_proba2'] = proba_class_2
        self.test_market_data['predicted_signal'] = np.where(self.test_market_data['true_proba1'] >=
min_probability, 'sell',
                                np.where(self.test_market_data['true_proba2'] >= min_probability, 'buy',
None))
        threshold = pd.Timedelta(hours=2)
        self.test_market_data['predicted_signal_clean'] = self.test_market_data['predicted_signal']
        for sig in ['buy', 'sell']:
            mask = self.test_market_data['predicted_signal_clean'] == sig
            times = self.test_market_data.loc[mask, 'time']
            dt = times.diff()
```

```
            drop_idx = dt[dt <= threshold].index
            self.test_market_data.loc[drop_idx, 'predicted_signal_clean'] = None
    def calculate_trades(self, target_close):
        self.trades = self.trades.iloc[0:0]
        mask_predicted_signal = self.test_market_data.predicted_signal_clean.isin(['sell', 'buy'])
        for index, row in tqdm(self.test_market_data[mask_predicted_signal].iterrows(),
total=self.test_market_data[mask_predicted_signal].shape[0]):
            open_time = row.time
            open_price = row.open
            end_time = open_time + pd.Timedelta(hours=2)
            future = self.test_market_data[
                (self.test_market_data.time > open_time) &
                (self.test_market_data.time <= end_time)
            ]
            if row.predicted_signal_clean == 'buy':
                target = open_price * (1 + target_close/100)
                hit = future[future.high >= target]
            else:
                target = open_price * (1 - target_close/100)
                hit = future[future.low <= target]
            if not hit.empty:
                close_price = target
                close_time = hit.iloc[0].time
            else:
                if not future.empty:
                    close_price = future.iloc[-1].close
                    close_time = future.iloc[-1].time
                else:
                    close_price = open_price
                    close_time = open_time
            new_row = {
                'open_time': open_time,
                'open_price': open_price,
                'close_time': close_time,
                'close_price': close_price,
                'side': row.predicted_signal_clean
            }
            self.trades = pd.concat(
                [self.trades, pd.DataFrame([new_row], columns=self.trades.columns)],
                ignore_index=True
            )
    def plot(self, min_datetime, max_datetime):
            plot_res(self.test_market_data, min_datetime, max_datetime)
model1 = load('models/stoh_osc/random_forest/model_test_btc.joblib')
test = Backtest(model1, btc_df_test, 1000)
test.predict(min_probability=0.67)
test.test_market_data[~test.test_market_data.predicted_signal.isna()]
test.calculate_trades(target_close=2)
trades = test.trades.copy()
trades = trades.rename(columns={
    'side':       'side',
    'open_time':   'time_open',
    'open_price':  'price_open',
    'close_time':  'time_close',
    'close_price': 'price_close'
})
trades['symbol']          = 'BTC'
```

```
trades['order_weight']     = 1
trades['is_maker_open']    = False
trades['is_maker_close']   = True
trades['amount']           = 0.0005
trades['volume_usd']       = trades['amount'] * trades['price_open']
trades['pl_pct']           = (trades.price_close - trades.price_open) / trades.price_open
trades['pl_usd'] = trades['pl_pct'] * trades['volume_usd'] * np.where(
    trades['side'].str.lower() == 'buy',
    1,
    -1
)
trades
class PortfolioAnalysis:
    required_columns = [
        'symbol', 'side', 'time_open', 'price_open',
        'time_close', 'price_close', 'order_weight',
        'is_maker_open', 'is_maker_close','volume_usd','amount','pl_usd'
    ]
    def __init__(self, df):
        self.validate_data(df)
        self.df = df.copy()
        self.df['time_open'] = pd.to_datetime(self.df['time_open'])
        if 'time_close' in self.df.columns:
            self.df['time_close'] = pd.to_datetime(self.df['time_close'])
    def validate_data(self, df):
        missing_columns = [col for col in self.required_columns if col not in df.columns]
        if missing_columns:
            raise ValueError(f"Input DataFrame is missing the following required columns: {',
'.join(missing_columns)}")
    def resample_data(self, period):
        df_copy = self.df.copy()
        df_copy.set_index('time_open', inplace=True)
        result = df_copy.resample(period).agg({'pl_usd': 'sum'}).reset_index()
        return result
    def days_convert(self):
        return self.resample_data('D')
    def months_convert(self):
        return self.resample_data('M')
    def years_convert(self):
        return self.resample_data('Y')
    def profit_sum(self):
        return round(self.df['pl_usd'].sum(), 3)
    def max_drawdown(self, df=None):
        if df is None:
            df = self.df.copy()
        df['cumulative_sum_pl'] = df['pl_usd'].cumsum()
        df['drawdown'] = df['cumulative_sum_pl'] - df['cumulative_sum_pl'].cummax()
        max_drawdown_value = df['drawdown'].min()
        return round(max_drawdown_value, 3)
    def max_daily_drawdown(self):
        daily_data = self.days_convert()
        return self.max_drawdown(daily_data)
    def win_rate_deals(self):
        win_rate = 100 * len(self.df[self.df['pl_usd'] > 0]) / len(self.df)
        return round(win_rate, 3)
    def win_rate_days(self):
        daily_data = self.days_convert()
```

```
        win_rate = 100 * len(daily_data[daily_data['pl_usd'] > 0]) / len(daily_data)
        return round(win_rate, 3)
    def average_profit(self, df=None, column='pl_usd'):
        if df is None:
            df = self.df
        return round(df[df[column] > 0][column].mean(), 3)
    def average_loss(self, df=None, column='pl_usd'):
        if df is None:
            df = self.df
        return round(df[df[column] < 0][column].mean(), 3)
    def sharpe_ratio(self):
        monthly_returns = self.days_convert()['pl_usd']
        average_return = np.mean(monthly_returns)
        standard_deviation = np.std(monthly_returns)
        sharpe_ratio_value = average_return / standard_deviation * np.sqrt(12)
        return round(sharpe_ratio_value, 3)
    def market_limit(self):
        results = {}
        columns = {
            'is_maker_open': 'maker_open',
            'is_maker_close': 'maker_close'
        }
        for column, column_name in columns.items():
            true_count = self.df[column].sum()
            false_count = len(self.df) - true_count
            total_count = len(self.df)
            true_percentage = (true_count / total_count) * 100
            false_percentage = (false_count / total_count) * 100
            results[column_name] = {
                'True': round(true_percentage, 2),
                'False': round(false_percentage, 2)
            }
        formatted_result = f"maker_open {results['maker_open']['True']}% maker close
{results['maker_close']['True']}%"
        return formatted_result
    def longest_drawdown_period(self):
        df = self.days_convert()
        df['cumulative_sum_pl'] = df['pl_usd'].cumsum()
        df['drawdown'] = df['cumulative_sum_pl'] - df['cumulative_sum_pl'].cummax()
        df['in_drawdown'] = df['drawdown'] < 0
        longest_drawdown_period = 0
        current_drawdown_start = None
        for index, row in df.iterrows():
            if row['in_drawdown']:
                if current_drawdown_start is None:
                    current_drawdown_start = row['time_open']
            else:
                if current_drawdown_start is not None:
                    drawdown_period = row['time_open'] - current_drawdown_start
                    longest_drawdown_period = max(longest_drawdown_period, drawdown_period.days)
                    current_drawdown_start = None
        if current_drawdown_start is not None:
            drawdown_period = df['time_open'].iloc[-1] - current_drawdown_start
            longest_drawdown_period = max(longest_drawdown_period, drawdown_period.days)
        return longest_drawdown_period
    def average_deal_time(self):
        self.df['deal_time'] = (self.df['time_close'] - self.df['time_open']).dt.total_seconds() / 60
```

```
        return round(self.df['deal_time'].mean(), 3)
    def average_deals_per_day(self):
        first_date = self.df['time_open'].min()
        last_date  = self.df['time_open'].max()
        total_days = (last_date.normalize() - first_date.normalize()).days + 1
        total_deals = self.df.shape[0]
        return round(total_deals / total_days, 3)
    def average_deals_per_month(self):
        first_date = self.df['time_open'].min()
        last_date  = self.df['time_open'].max()
        total_months = (last_date.year - first_date.year) * 12 + (last_date.month - first_date.month) + 1
        total_deals = self.df.shape[0]
        return round(total_deals / total_months, 3)
    def start_end_backtest(self):
        start = self.df['time_open'].iloc[0]
        end = self.df['time_open'].iloc[-1]
        duration = end - start
        return f"{start}\nBacktest end: {end}\nBacktest duration: {duration}"
    def stats(self):
        print('Max profit:', f'{self.profit_sum()}%')
        print('Max drawdown:', f'{self.max_drawdown()}%')
        print('Max daily drawdown:', f'{self.max_daily_drawdown()}%')
        print('Average deal profit:', f'{self.average_profit()}%')
        print('Average deal loss:', f'{self.average_loss()}%')
        print('Average daily profit:', f'{self.average_profit(self.days_convert())}%')
        print('Average daily loss:', f'{self.average_loss(self.days_convert())}%')
        print('Deals win rate:', f'{self.win_rate_deals()}%')
        print('Sharpe ratio:', self.sharpe_ratio())
        print('Longest drawdown period:', f'{self.longest_drawdown_period()} days')
        print('Backtest start:', self.start_end_backtest())
        print('Average Deals per day:', self.average_deals_per_day())
        print('Average Deals per month:', self.average_deals_per_month())
analysis = PortfolioAnalysis(trades)
analysis.stats()
temp = trades.sort_values('time_open')
temp['cum_pl_usd'] = temp['pl_usd'].cumsum()
plt.figure(figsize=(10, 6))
plt.plot(temp['time_open'], temp['cum_pl_usd'])
plt.xlabel('Time Open')
plt.ylabel('Cumulative P&L (USD)')
plt.title('Cumulative P&L over Time')
plt.tight_layout()
plt.show()
def economic_analysis(model_name, min_probability, close_time):
    model1 = load(model_name)
    test = Backtest(model1, btc_df_test, 1000)
    test.predict(min_probability=min_probability)
    test.calculate_trades(target_close=close_time)
    trades = test.trades.copy()
    trades = trades.rename(columns={
        'side':       'side',
        'open_time':   'time_open',
        'open_price':  'price_open',
        'close_time':  'time_close',
        'close_price': 'price_close'
    })
    trades['symbol']          = 'BTC'
```

```
    trades['order_weight']     = 1
    trades['is_maker_open']    = False
    trades['is_maker_close']   = True
    trades['amount']           = 0.0005
    trades['volume_usd']       = trades['amount'] * trades['price_open']
    trades['pl_pct']           = (trades.price_close - trades.price_open) / trades.price_open
    trades['pl_usd'] = trades['pl_pct'] * trades['volume_usd'] * np.where(
        trades['side'].str.lower() == 'buy',
        1,
        -1
    )
    analysis = PortfolioAnalysis(trades)
    analysis.stats()
    temp = trades.sort_values('time_open')
    temp['cum_pl_usd'] = temp['pl_usd'].cumsum()
    plt.figure(figsize=(10, 6))
    plt.plot(temp['time_open'], temp['cum_pl_usd'])
    plt.xlabel('Time Open')
    plt.ylabel('Cumulative P&L %')
    plt.title('Cumulative P&L over Time')
    plt.tight_layout()
    plt.show()
    display(trades)
economic_analysis(model_name='models/stoh_osc/random_forest/model_test_btc.joblib',
min_probability=0.8, close_time=1)
economic_analysis(model_name='models/stoh_osc/xgboost/btc_xgb_test.joblib',      min_probability=0.8,
close_time=2)
economic_analysis(model_name='models/stoh_osc/svc/btc_svc_scaled_test.pkl',      min_probability=0.8,
close_time=2)
```

parabolicsar.py

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import ta
from scipy.signal import find_peaks
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from joblib import dump, load
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import TimeSeriesSplit, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, StratifiedKFold,cross_val_score
from sklearn.preprocessing import StandardScaler
from ta.volatility import BollingerBands
from xgboost import XGBClassifier, callback as xgb_callback
import optuna
from tqdm import tqdm
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score,
average_precision_score
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.ensemble import BalancedRandomForestClassifier
```

```
from optuna.samplers import TPESampler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from ta.trend import PSARIndicator
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, precision_recall_curve, auc
btc_df = pd.read_csv('data/btcusdt_1m_2.csv')
btc_df.time = pd.to_datetime(btc_df.time, utc=True)
prices_full = btc_df['close'].values
peaks,   _ = find_peaks(prices_full,  distance=0.5 * 24 * 60, prominence=1000, width=3)
troughs, _ = find_peaks(-prices_full, distance=0.5 * 24 * 60, prominence=1000, width=3)
btc_df['is_peak']   = btc_df.index.to_series().isin(peaks)
btc_df['is_trough'] = btc_df.index.to_series().isin(troughs)
btc_df['is_reverse'] = (btc_df['is_peak'] | btc_df['is_trough'])
btc_df['is_peak_orig'] = btc_df['is_peak']
btc_df['is_trough_orig'] = btc_df['is_trough']
btc_df['is_peak'] = False
btc_df['is_trough'] = False
btc_df['is_reverse'] = False
for idx in btc_df.index[btc_df['is_peak_orig'] | btc_df['is_trough_orig']]:
    start = max(idx - 10, 0)
    end = min(idx + 1, len(btc_df))
    btc_df.loc[start:end, 'is_reverse'] = True
    if btc_df.at[idx, 'is_peak_orig']:
        btc_df.loc[start:end, 'is_peak'] = True
    if btc_df.at[idx, 'is_trough_orig']:
        btc_df.loc[start:end, 'is_trough'] = True
btc_df['signal'] = np.where(
    btc_df['is_peak'], 'sell',
    np.where(btc_df['is_trough'], 'buy', 'none')
)
psar_ind = PSARIndicator(
    high=btc_df['high'],
    low=btc_df['low'],
    close=btc_df['close'],
    step=0.02,
    max_step=0.2
)
btc_df['psar']       = psar_ind.psar()
btc_df['psar_up']    = psar_ind.psar_up()
btc_df['psar_dir'] = (~btc_df['psar_up'].isna()).astype(int)
btc_df['psar_lag1']      = btc_df['psar'].shift(1)
btc_df['psar_dir_lag1']   = btc_df['psar_dir'].shift(1)
btc_df['psar_lag2']      = btc_df['psar'].shift(2)
btc_df['psar_dir_lag2']   = btc_df['psar_dir'].shift(2)
btc_df['psar_lag3']      = btc_df['psar'].shift(3)
btc_df['psar_dir_lag3']   = btc_df['psar_dir'].shift(3)
btc_df['psar_lag4']      = btc_df['psar'].shift(4)
btc_df['psar_dir_lag4']   = btc_df['psar_dir'].shift(4)
btc_df['psar_delta1']      = btc_df['psar']      - btc_df['psar_lag1']
btc_df['psar_dir_delta1']   = btc_df['psar_dir']   - btc_df['psar_dir_lag1']
btc_df['psar_delta2']      = btc_df['psar_lag1']      - btc_df['psar_lag2']
btc_df['psar_dir_delta2']   = btc_df['psar_dir_lag1']   - btc_df['psar_dir_lag2']
btc_df.drop('psar_up', axis=1, inplace=True)
btc_df_train_part = 0.8
n_total   = len(btc_df)
n_train   = int(n_total * btc_df_train_part)
```

```
btc_df_train = btc_df.iloc[:n_train].copy()
btc_df_test  = btc_df.iloc[n_train:].copy()
total_needed     = 5000
min_chunk_size   = 50
max_chunk_size   = 200
working_df = btc_df_train.reset_index(names='orig_idx')
mask_non_rev = ~working_df['is_reverse']
working_df['grp'] = mask_non_rev.ne(mask_non_rev.shift()).cumsum()
runs = working_df.loc[mask_non_rev].groupby('grp')
runs_info = runs.size().reset_index(name='length')
valid_runs = runs_info[
    (runs_info['length'] >= min_chunk_size)
].copy().sample(frac=1, random_state=42).reset_index(drop=True)
selected_chunks = []
remaining = total_needed
for _, row in valid_runs.iterrows():
    block = runs.get_group(row['grp'])
    L = row['length']
    if L > max_chunk_size:
        block = block.iloc[:max_chunk_size]
        L = max_chunk_size
    if L < min_chunk_size:
        continue
    take_n = min(L, remaining)
    selected_chunks.append(block.iloc[:take_n])
    remaining -= take_n
    if remaining == 0:
        break
non_reverse_chunks = pd.concat(selected_chunks)
only_true = working_df.loc[working_df['is_reverse']].copy()
train_df = pd.concat([only_true, non_reverse_chunks], ignore_index=True)
train_df = train_df.sort_values('orig_idx').drop(columns=['orig_idx', 'grp'])
train_df.dropna(inplace=True)
mask_col=['psar', 'psar_dir', 'psar_lag1', 'psar_dir_lag1', 'psar_lag2', 'psar_dir_lag2', 'psar_lag3',
'psar_dir_lag3', 'psar_lag4', 'psar_dir_lag4', 'psar_delta1', 'psar_dir_delta1', 'psar_delta2', 'psar_dir_delta2']
X_train = train_df[mask_col]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 500),
        'max_depth': trial.suggest_int('max_depth', 2, 30),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 20),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', None]),
        'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy']),
    }
    clf = RandomForestClassifier(**params, random_state=42, n_jobs=-1)
    tscv = TimeSeriesSplit(n_splits=3)
    precision_macro_scorer = make_scorer(
    precision_score,
    average='macro',
    zero_division=0
)
    scores = cross_val_score(
        clf,
```

```
        X_train,
        y_train_enc,
        cv=tscv,
        scoring=precision_macro_scorer,
        n_jobs=-1
    )
    return scores.mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
best_model = RandomForestClassifier(**study.best_params, random_state=42, n_jobs=-1)
best_model.fit(X_train, y_train_enc)
X_test = btc_df_test[mask_col]
y_test = btc_df_test['signal']
y_test_enc  = y_test.map(mapping)
y_pred = best_model.predict(X_test)
y_probas = best_model.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred))
proba_all = best_model.predict_proba(X_test)
pred = best_model.predict(X_test)
proba_class_0 = proba_all[:, 0]
proba_class_1 = proba_all[:, 1]
proba_class_2 = proba_all[:, 2]
btc_df_test['true_proba1'] = proba_class_1
btc_df_test['true_proba2'] = proba_class_2
btc_df_test['predicted_signal'] = np.where(btc_df_test['true_proba1'] >= 0.72, 'sell',
                                np.where(btc_df_test['true_proba2'] >= 0.72, 'buy', None))
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
btc_df_high_conf.predicted_signal.value_counts()
def plot_res(df, start_time, end_time):
    temp = df[(df.time > start_time) & (df.time < end_time)].copy()
    fig = make_subplots(
        rows=1, cols=1,
        shared_xaxes=True,
        vertical_spacing=0.02,
        subplot_titles=('Candlechart',)
    )
    fig.add_trace(
        go.Candlestick(
            x=temp['time'], open=temp['open'], high=temp['high'],
            low=temp['low'], close=temp['close'],
            increasing_line_color='green',
            decreasing_line_color='red'
        ),
        row=1, col=1
    )
    buy_signals = temp[temp['predicted_signal_clean'] == 'buy']
    if not buy_signals.empty:
        fig.add_trace(
            go.Scatter(
                x=buy_signals['time'],
                y=buy_signals['low'] * 0.97,
                mode='markers',
                marker=dict(
                    symbol='triangle-up',
                    size=15,
                    color='green',
                    line=dict(width=2, color='darkgreen')
```

```
            ),
            name='Buy Signal',
            customdata=buy_signals['true_proba2'],
            hovertemplate=(
                'PredictProbe1: %{customdata}<br>'
                'Time: %{x}<br>'
                'Price: %{y}<extra></extra>'
            )
        ), row=1, col=1
    )
    sell_signals = temp[temp['predicted_signal_clean'] == 'sell']
    if not sell_signals.empty:
        fig.add_trace(
            go.Scatter(
                x=sell_signals['time'],
                y=sell_signals['high'] * 1.03,
                mode='markers',
                marker=dict(
                    symbol='triangle-down',
                    size=15,
                    color='red',
                    line=dict(width=2, color='darkred')
                ),
                name='Sell Signal',
                customdata=sell_signals['true_proba1'],
                hovertemplate=(
                    'PredictProbe1: %{customdata}<br>'
                    'Time: %{x}<br>'
                    'Price: %{y}<extra></extra>'
                )
            ), row=1, col=1
        )
    fig.update_layout(
        title='Predicted singals',
        width=1200, height=800, hovermode='x unified',
        xaxis_rangeslider_visible=False
    )
    fig.update_xaxes(type='date', row=1, col=1)
    fig.show()
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
X_train = train_df[mask_col]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
y_train_enc = y_train.map(mapping)
def objective(trial):
    params = {
        'n_estimators':       trial.suggest_int('n_estimators', 50, 500),
        'max_depth':          trial.suggest_int('max_depth', 3, 15),
        'learning_rate':      trial.suggest_float('learning_rate', 1e-3, 0.3, log=True),
        'subsample':          trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree':    trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'gamma':              trial.suggest_float('gamma', 0.0, 5.0),
        'reg_alpha':          trial.suggest_float('reg_alpha', 1e-8, 1.0, log=True),
        'reg_lambda':         trial.suggest_float('reg_lambda', 1e-8, 1.0, log=True),
        'eval_metric':        'mlogloss',
        'verbosity':          0
    }
    clf = XGBClassifier(**params, n_jobs=-1)
    tscv = TimeSeriesSplit(n_splits=3)
    scores = cross_val_score(
        clf,
        X_train,
        y_train_enc,
        cv=tscv,
        scoring='precision_macro',
        n_jobs=-1
    )
    return scores.mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
clf_best = load('models/parabolicSAR/xgboost/btc_xgb_test2.joblib')
X_test = btc_df_test[mask_col]
y_test = btc_df_test['signal']
y_test_enc  = y_test.map(mapping)
y_pred = clf_best.predict(X_test)
```

```
y_probas = clf_best.predict_proba(X_test)
print(classification_report(y_test_enc, y_pred))
X_test = btc_df_test[mask_col]
proba_all   = clf_best.predict_proba(X_test)
btc_df_test['proba_none'] = proba_all[:, 0]
btc_df_test['true_proba1'] = proba_all[:, 1]
btc_df_test['true_proba2']  = proba_all[:, 2]
threshold = 0.9
btc_df_test['predicted_signal'] = None
btc_df_test.loc[btc_df_test['true_proba2']  >= threshold, 'predicted_signal'] = 'buy'
mask_sell = btc_df_test['predicted_signal'].isna() & (btc_df_test['true_proba1'] >= threshold)
btc_df_test.loc[mask_sell, 'predicted_signal'] = 'sell'
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
print(btc_df_high_conf['predicted_signal'].value_counts())
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-10-01 00:00:00', '2024-11-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
X_train = train_df[mask_col]
y_train = train_df['signal']
mapping = {'none': 0, 'sell': 1, 'buy': 2}
```

```
y_train_enc = y_train.map(mapping)
scorer = make_scorer(
    precision_score,
    average='macro',
    zero_division=0
)
def objective(trial):
    C     = trial.suggest_float('C', 1e-2, 100,   log=True)
    gamma = trial.suggest_float('gamma', 1e-3, 1, log=True)
    model = Pipeline([
        ('scale', StandardScaler()),
        ('svc', SVC(
            C=C,
            gamma=gamma,
            kernel='rbf',
            class_weight='balanced',
            probability=False,
            random_state=42
        ))
    ])
    cv = TimeSeriesSplit(n_splits=5)
    scores = cross_val_score(
        model,
        X_train,
        y_train_enc,
        cv=cv,
        scoring=scorer,
        n_jobs=-1
    )
    return scores.mean()
sampler = TPESampler(seed=42)
study   = optuna.create_study(direction='maximize', sampler=sampler)
study.optimize(objective, n_trials=100, timeout=600)
print('Лучший trial:', study.best_trial.number)
print('Лучший precision_macro  :', study.best_trial.value)
print('Параметры  :', study.best_params)
print("Лучшие гиперпараметры:", study.best_params)
best_params = study.best_params.copy()
svc_pipe = Pipeline([
    ('scale', StandardScaler()),
    ('svc',   SVC(**best_params, probability=True, random_state=42))
])
svc_pipe.fit(X_train, y_train_enc)
X_test  = btc_df_test[mask_col]
y_test  = btc_df_test['signal']
y_test_enc = y_test.map(mapping)
y_pred = svc_pipe.predict(X_test)
print(classification_report(y_test_enc, y_pred, target_names=['none','sell','buy']))
proba_all = svc_pipe.predict_proba(btc_df_test[mask_col])
pred = svc_pipe.predict(btc_df_test[mask_col])
proba_class_0 = proba_all[:, 0]
proba_class_1 = proba_all[:, 1]
proba_class_2 = proba_all[:, 2]
btc_df_test['true_proba1'] = proba_class_1
btc_df_test['true_proba2'] = proba_class_2
btc_df_test['predicted_signal'] = np.where(btc_df_test['true_proba1'] >= 0.75, 'sell',
                                  np.where(btc_df_test['true_proba2'] >= 0.75, 'buy', None))
```

```
btc_df_high_conf = btc_df_test[btc_df_test['predicted_signal'].notna()].copy()
btc_df_high_conf.predicted_signal.value_counts()
btc_df_high_conf
threshold = pd.Timedelta(hours=3)
btc_df_test['predicted_signal_clean'] = btc_df_test['predicted_signal']
for sig in ['buy', 'sell']:
    mask = btc_df_test['predicted_signal_clean'] == sig
    times = btc_df_test.loc[mask, 'time']
    dt = times.diff()
    drop_idx = dt[dt <= threshold].index
    btc_df_test.loc[drop_idx, 'predicted_signal_clean'] = None
plot_res(btc_df_test, '2024-09-01 00:00:00', '2024-10-01 00:00:00')
inv_mapping = {v: k for k, v in mapping.items()}
y_test_bin = label_binarize(y_test_enc, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]
fpr = {}
tpr = {}
roc_auc = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_probas[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_probas.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
plt.figure()
for i in range(n_classes):
    class_name = inv_mapping[i]
    plt.plot(fpr[i], tpr[i], label=f"{class_name} (AUC={roc_auc[i]:.2f})")
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (AUC={roc_auc['micro']:.2f})", linestyle="--")
plt.plot([0, 1], [0, 1], linestyle=":", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Multiclass ROC Curve")
plt.legend()
plt.show()
from sklearn.metrics import balanced_accuracy_score
bal_acc = balanced_accuracy_score(y_test_enc, y_pred)
bal_acc
from sklearn.metrics import f1_score
f1_macro = f1_score(y_test_enc, y_pred, average="macro")
f1_macro
from sklearn.metrics import average_precision_score
ap_macro = average_precision_score(y_test_enc, y_probas, average="macro")
ap_macro
class Backtest:
    def __init__(self, model, test_market_data, start_balance):
        self.model = model
        self.test_market_data = test_market_data.copy()
        self.trades = pd.DataFrame({
            'open_time': pd.Series([], dtype='datetime64[ns, UTC]'),
            'open_price': pd.Series([], dtype='float'),
            'close_price': pd.Series([], dtype='float'),
            'close_time': pd.Series([], dtype='datetime64[ns, UTC]'),
            'side': pd.Series([], dtype='str'),
        })
        self.balance = start_balance
        self.balance_history = pd.DataFrame({
                'trade_id': pd.Series([], dtype='int'),
```

```
            'time': pd.Series([], dtype='datetime64[ns, Etc/GMT+5]'),
            'balance': pd.Series([], dtype='float'),
            'affect': pd.Series([], dtype='str'),
            'coef': pd.Series([], dtype='float')
            })
    def predict(self, min_probability):
        proba_all = self.model.predict_proba(self.test_market_data[mask_col])
        proba_class_1 = proba_all[:, 1]
        proba_class_2 = proba_all[:, 2]
        self.test_market_data['true_proba1'] = proba_class_1
        self.test_market_data['true_proba2'] = proba_class_2
        self.test_market_data['predicted_signal'] = np.where(self.test_market_data['true_proba1'] >= min_probability, 'sell',
                                    np.where(self.test_market_data['true_proba2'] >= min_probability, 'buy', None))
        threshold = pd.Timedelta(hours=3)
        self.test_market_data['predicted_signal_clean'] = self.test_market_data['predicted_signal']
        for sig in ['buy', 'sell']:
            mask = self.test_market_data['predicted_signal_clean'] == sig
            times = self.test_market_data.loc[mask, 'time']
            dt = times.diff()
            drop_idx = dt[dt <= threshold].index
            self.test_market_data.loc[drop_idx, 'predicted_signal_clean'] = None
    def calculate_trades(self, target_close):
        self.trades = self.trades.iloc[0:0]
        mask_predicted_signal = self.test_market_data.predicted_signal_clean.isin(['sell', 'buy'])
        for index, row in tqdm(self.test_market_data[mask_predicted_signal].iterrows(), total=self.test_market_data[mask_predicted_signal].shape[0]):
            open_time = row.time
            open_price = row.open
            end_time = open_time + pd.Timedelta(hours=2)
            future = self.test_market_data[
                (self.test_market_data.time > open_time) &
                (self.test_market_data.time <= end_time)
            ]
            if row.predicted_signal_clean == 'buy':
                target = open_price * (1 + target_close/100)
                hit = future[future.high >= target]
            else:
                target = open_price * (1 - target_close/100)
                hit = future[future.low <= target]
            if not hit.empty:
                close_price = target
                close_time = hit.iloc[0].time
            else:
                if not future.empty:
                    close_price = future.iloc[-1].close
                    close_time = future.iloc[-1].time
                else:
                    close_price = open_price
                    close_time = open_time
            new_row = {
                'open_time': open_time,
                'open_price': open_price,
                'close_time': close_time,
                'close_price': close_price,
                'side': row.predicted_signal_clean
```

```
            }
            self.trades = pd.concat(
                [self.trades, pd.DataFrame([new_row], columns=self.trades.columns)],
                ignore_index=True
            )
    def plot(self, min_datetime, max_datetime):
            plot_res(self.test_market_data, min_datetime, max_datetime)
model1 = load('models/parabolicSAR/random_forest/model_btc_best.joblib')
test = Backtest(model1, btc_df_test, 1000)
test.predict(min_probability=0.7)
test.calculate_trades(target_close=2)
test.trades
trades = test.trades.copy()
trades = trades.rename(columns={
    'side':       'side',
    'open_time':   'time_open',
    'open_price':  'price_open',
    'close_time':  'time_close',
    'close_price': 'price_close'
})
trades['symbol']          = 'BTC'
trades['order_weight']    = 1
trades['is_maker_open']   = False
trades['is_maker_close']  = False
trades['amount']          = 0.0005
trades['volume_usd']      = trades['amount'] * trades['price_open']
trades['pl_pct']          = (trades.price_close - trades.price_open) / trades.price_open
trades['pl_usd'] = trades['pl_pct'] * trades['volume_usd'] * np.where(
    trades['side'].str.lower() == 'buy',
    1,
    -1
)
class PortfolioAnalysis:
    required_columns = [
        'symbol', 'side', 'time_open', 'price_open',
        'time_close', 'price_close', 'order_weight',
        'is_maker_open', 'is_maker_close','volume_usd','amount','pl_usd'
    ]
    def __init__(self, df):
        self.validate_data(df)
        self.df = df.copy()
        self.df['time_open'] = pd.to_datetime(self.df['time_open'])
        if 'time_close' in self.df.columns:
            self.df['time_close'] = pd.to_datetime(self.df['time_close'])
    def validate_data(self, df):
        missing_columns = [col for col in self.required_columns if col not in df.columns]
        if missing_columns:
            raise ValueError(f"Input DataFrame is missing the following required columns: {',
'.join(missing_columns)}")
    def resample_data(self, period):
        df_copy = self.df.copy()
        df_copy.set_index('time_open', inplace=True)
        result = df_copy.resample(period).agg({'pl_usd': 'sum'}).reset_index()
        return result
    def days_convert(self):
        return self.resample_data('D')
    def months_convert(self):
```

```
        return self.resample_data('M')
    def years_convert(self):
        return self.resample_data('Y')
    def profit_sum(self):
        return round(self.df['pl_usd'].sum(), 3)
    def max_drawdown(self, df=None):
        if df is None:
            df = self.df.copy()
        df['cumulative_sum_pl'] = df['pl_usd'].cumsum()
        df['drawdown'] = df['cumulative_sum_pl'] - df['cumulative_sum_pl'].cummax()
        max_drawdown_value = df['drawdown'].min()
        return round(max_drawdown_value, 3)
    def max_daily_drawdown(self):
        daily_data = self.days_convert()
        return self.max_drawdown(daily_data)
    def win_rate_deals(self):
        win_rate = 100 * len(self.df[self.df['pl_usd'] > 0]) / len(self.df)
        return round(win_rate, 3)
    def win_rate_days(self):
        daily_data = self.days_convert()
        win_rate = 100 * len(daily_data[daily_data['pl_usd'] > 0]) / len(daily_data)
        return round(win_rate, 3)
    def average_profit(self, df=None, column='pl_usd'):
        if df is None:
            df = self.df
        return round(df[df[column] > 0][column].mean(), 3)
    def average_loss(self, df=None, column='pl_usd'):
        if df is None:
            df = self.df
        return round(df[df[column] < 0][column].mean(), 3)
    def sharpe_ratio(self):
        monthly_returns = self.days_convert()['pl_usd']
        average_return = np.mean(monthly_returns)
        standard_deviation = np.std(monthly_returns)
        sharpe_ratio_value = average_return / standard_deviation * np.sqrt(12)
        return round(sharpe_ratio_value, 3)
    def market_limit(self):
        results = {}
        columns = {
            'is_maker_open': 'maker_open',
            'is_maker_close': 'maker_close'
        }
        for column, column_name in columns.items():
            true_count = self.df[column].sum()
            false_count = len(self.df) - true_count
            total_count = len(self.df)
            true_percentage = (true_count / total_count) * 100
            false_percentage = (false_count / total_count) * 100
            results[column_name] = {
                'True': round(true_percentage, 2),
                'False': round(false_percentage, 2)
            }
        formatted_result = f"maker_open {results['maker_open']['True']}% maker close
{results['maker_close']['True']}%"
        return formatted_result
    def longest_drawdown_period(self):
        df = self.days_convert()
```

```
        df['cumulative_sum_pl'] = df['pl_usd'].cumsum()
        df['drawdown'] = df['cumulative_sum_pl'] - df['cumulative_sum_pl'].cummax()
        df['in_drawdown'] = df['drawdown'] < 0
        longest_drawdown_period = 0
        current_drawdown_start = None
        for index, row in df.iterrows():
            if row['in_drawdown']:
                if current_drawdown_start is None:
                    current_drawdown_start = row['time_open']
            else:
                if current_drawdown_start is not None:
                    drawdown_period = row['time_open'] - current_drawdown_start
                    longest_drawdown_period = max(longest_drawdown_period, drawdown_period.days)
                    current_drawdown_start = None
        if current_drawdown_start is not None:
            drawdown_period = df['time_open'].iloc[-1] - current_drawdown_start
            longest_drawdown_period = max(longest_drawdown_period, drawdown_period.days)
        return longest_drawdown_period
    def average_deal_time(self):
        self.df['deal_time'] = (self.df['time_close'] - self.df['time_open']).dt.total_seconds() / 60
        return round(self.df['deal_time'].mean(), 3)
    def average_deals_per_day(self):
        first_date = self.df['time_open'].min()
        last_date  = self.df['time_open'].max()
        total_days = (last_date.normalize() - first_date.normalize()).days + 1
        total_deals = self.df.shape[0]
        return round(total_deals / total_days, 3)
    def average_deals_per_month(self):
        first_date = self.df['time_open'].min()
        last_date  = self.df['time_open'].max()
        total_months = (last_date.year - first_date.year) * 12 + (last_date.month - first_date.month) + 1
        total_deals = self.df.shape[0]
        return round(total_deals / total_months, 3)
    def start_end_backtest(self):
        start = self.df['time_open'].iloc[0]
        end = self.df['time_open'].iloc[-1]
        duration = end - start
        return f"{start}\nBacktest end: {end}\nBacktest duration: {duration}"
    def stats(self):
        print('Max profit:', f'{self.profit_sum()}%')
        print('Max drawdown:', f'{self.max_drawdown()}%')
        print('Max daily drawdown:', f'{self.max_daily_drawdown()}%')
        print('Average deal profit:', f'{self.average_profit()}%')
        print('Average deal loss:', f'{self.average_loss()}%')
        print('Average daily profit:', f'{self.average_profit(self.days_convert())}%')
        print('Average daily loss:', f'{self.average_loss(self.days_convert())}%')
        print('Deals win rate:', f'{self.win_rate_deals()}%')
        print('Sharpe ratio:', self.sharpe_ratio())
        print('Longest drawdown period:', f'{self.longest_drawdown_period()} days')
        print('Backtest start:', self.start_end_backtest())
        print('Average Deals per day:', self.average_deals_per_day())
        print('Average Deals per month:', self.average_deals_per_month())
analysis = PortfolioAnalysis(trades)
analysis.stats()
temp = trades.sort_values('time_open')
temp['cum_pl_usd'] = temp['pl_usd'].cumsum()
plt.figure(figsize=(10, 6))
```

```
plt.plot(temp['time_open'], temp['cum_pl_usd'])
plt.xlabel('Time Open')
plt.ylabel('Cumulative P&L (USD)')
plt.title('Cumulative P&L over Time')
plt.tight_layout()
plt.show()
def economic_analysis(model_name, min_probability, close_time):
    model1 = load(model_name)
    test = Backtest(model1, btc_df_test, 1000)
    test.predict(min_probability=min_probability)
    test.calculate_trades(target_close=close_time)
    trades = test.trades.copy()
    trades = trades.rename(columns={
        'side':       'side',
        'open_time':   'time_open',
        'open_price':  'price_open',
        'close_time':  'time_close',
        'close_price': 'price_close'
    })
    trades['symbol']          = 'BTC'
    trades['order_weight']    = 1
    trades['is_maker_open']   = False
    trades['is_maker_close']  = True
    trades['amount']          = 0.0005
    trades['volume_usd']      = trades['amount'] * trades['price_open']
    trades['pl_pct']          = (trades.price_close - trades.price_open) / trades.price_open
    trades['pl_usd'] = trades['pl_pct'] * trades['volume_usd'] * np.where(
        trades['side'].str.lower() == 'buy',
        1,
        -1
    )
    analysis = PortfolioAnalysis(trades)
    analysis.stats()
    temp = trades.sort_values('time_open')
    temp['cum_pl_usd'] = temp['pl_usd'].cumsum()
    plt.figure(figsize=(10, 6))
    plt.plot(temp['time_open'], temp['cum_pl_usd'])
    plt.xlabel('Time Open')
    plt.ylabel('Cumulative P&L %')
    plt.title('Cumulative P&L over Time')
    plt.tight_layout()
    plt.show()
    display(trades)
economic_analysis(model_name='models/parabolicSAR/random_forest/model_btc_test.joblib',
min_probability=0.7, close_time=2)
economic_analysis(model_name='models/parabolicSAR/xgboost/btc_xgb_test2.joblib',
min_probability=0.9, close_time=2)
economic_analysis(model_name='models/parabolicSAR/svc/btc_svc_scaled_test.pkl',
min_probability=0.75, close_time=2)
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ