Київський національний університет імені Тараса
ШевченкаФакультет комп’ютерних наук та кібернетики
Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
на тему:

АНАЛІЗ ДЕЯКИХ АЛГОРИТМІВ БІРЖЕВОЇ ТОРГІВЛІ

студента 2 курсу
Астахова Максима Вячеславовича

Науковий керівник:
доктор фізико–математичних наук
Самойленко І.В.

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол № 8 від 03 травня 2023 р.

Завідувач кафедри ДО проф. Іксанов О.М.

Київ — 2023

ЗМІСТ

ВСТУП

В сучасному світі біржова торгівля є однією з ключових складових фінансової системи. Для інвесторів та трейдерів, що займаються біржовою торгівлею, важливою є точність прогнозування змін цін на фондовому ринку, щоб забезпечити оптимальні торгівельні рішення та мінімізувати ризики. Для цього активно використовуються методи прогнозування часових рядів.

Магістерська кваліфікаційна робота на тему "Аналіз деяких алгоритмів біржової торгівлі" має на меті дослідити та порівняти різні методи прогнозування часових рядів. Метою цього дослідження є визначення найефективнішої моделі для прогнозування цін на наступну годину.

Для дослідження були зібрані дані про ціну криптовалюти Біткоїн(Bitcoin) з 01.01.2022 по 06.01.2023, а також використані усі “твіти” з хештегом #BTC у соціальній мережі Twitter за то й же проміжок часу.

1 БІРЖА ТА КРИПТОВАЛЮТИ

Криптовалюта (cryptocurrency) – це різновид незалежної цифрової валюти, у якій децентралізовано управління такими процесами платіжної системи (емісія монет, підтвердження транзакцій, зберігання даних, аудит облікової системи та прийняття рішень щодо оновлень). Платіжна система діє повністю в автоматичному режимі, без можливості внутрішнього та зовнішнього адміністрування. Стати учасником системи може будь-хто охочий. Вартість монет не зазнає впливу з боку їх творців, проведення транзакцій та володіння криптовалютою відбувається на основі рівних прав усіх користувачів.Децентралізація є характерною особливістю криптовалюти. Саме вона забезпечує таку стійкість та надійність системи, завдяки чому криптовалюти дають можливість вільно здійснювати платежі, уникаючи бюрократичного складника та не створюючи додаткових обмежень. Слід зазначити, що така система стійка до інтернет-цензури та різного роду атак, оскільки відкрита для всіх учасників (всі користувачі системи можуть здійснювати аудит транзакцій, створювати та підтверджувати їх без необхідності проходити ідентифікацію/верифікацію особи).

Криптовалюти характеризуються тим, що інформація системи зберігається в блокчейні. Блокчейн (blockchain) - база даних, що містить інформацію про транзакції у вигляді ланцюга блоків (записів). Кожен блок містить інформацію, що підтверджує цілісність попереднього блоку (а саме часову позначку, хеш попереднього блоку та дані транзакцій, подані як Дерево Меркла (хеш-дерево)). Кожен наступний блок підтверджує цілісність попереднього аж до першого блоку (так званий genesis block, першим створений у ланцюжку, за яким учасники можуть створювати наступні блоки). Така система забезпечується односторонній

зв'язок всіх блоків, тобто що кожен блок створений після появи попереднього. ВІдповідно внесення змін в один з блоків стає майже неможливим, оскільки потребує відповідних змін в усіх блоках після нього.

Однією з найпоширеніших криптовалют є Bitcoin - це протокол, що реалізує незалежну валюту та платіжну систему, яка керує цією валютою. Вперше домен bitcoin.org було зареєстровано в 2008 році людиною чи групою під псевдонімом Сатоші Накамото. Пізніше у статті «Bitcoin: A Peerto-Peer Electronic Cash System» він згадав два проекти, ідеї яких йому вдалось поєднати: HashCash Адама Бека (підхід proof-of-work) і B-Money Вей Дая (модель мережі розподіленого зберігання даних про транзакції). Наразі біткоїн має величезну популярність, активно використовується компаніями та навіть державами як законний засіб платежу.

Ціна на криптовалюти формується на біржі. Біржа - це інституція, яка забезпечує можливість торгівлі цінними паперами, товарами, валютами та іншими активами між різними учасниками. У контексті фінансової теорії, біржа може розглядатися як відкрите ринкове місце з високою ліквідністю та прозорістю, на якому зустрічаються попит та пропозиції на купівлю/продаж активів.

Біржа є важливим інструментом для формування цін на активи та їх оцінки. Це досягається шляхом встановлення біржових курсів та торговельних режимів, що регулюються спеціальними правилами та нормативними актами. Крім того, біржі забезпечують учасникам ринку широкий спектр інформації про активи та про умови їх торгівлі.Біржі можуть бути універсальними, де торгуються різноманітні активи, або спеціалізованими, де торгуються певні види активів. Біржі грають важливу роль в економіці, оскільки вони допомагають створювати ринкові ціни на основі принципу попиту та пропозиції. Іншими словами, вони сприяють ефективному розподілу ресурсів і капіталу в економіці. Крім того,

важливим елементом біржі є її роль у регулюванні торгівлі. Біржі встановлюють правила та процедури, які мають на меті захист учасників від шахрайства та інших видів зловживань. Вони також надають механізми для розв'язання спорів між учасниками. Слід також зазначити, що, біржі сприяють ліквідності, оскільки вони забезпечують велику кількість потенційних покупців та продавців. Це дозволяє учасникам швидко купувати та продавати активи без значного впливу на ціну.Біржі відіграють важливу роль у глобалізації фінансових ринків, оскільки вони дозволяють учасникам з різних країн проводити торговельні операції.

Криптовалютна біржа, або цифрова валютна біржа (DCE), це бізнес, що дозволяє клієнтам торгувати криптовалютами або цифровими валютами за інші активи, такі як традиційні фіатні гроші або інші цифрові валюти. Вони можуть бути ринковими майданчиками для криптовалют або можуть бути майданчиками для їхнього перетворення на традиційні валюти.

Криптовалютні біржі використовують системи замовлень для створення мережі покупців і продавців. Використовуючи ці системи, покупці та продавці криптовалют можуть вказувати кількість, яку вони хочуть купити або продати, і ціну, за яку вони хочуть торгувати.Ці біржі зазвичай надають захист від шахрайства, оскільки вони вимагають від учасників проходити процес перевірки особи (KYC - "Know Your Customer") перед торгівлею. Однак, залежно від юрисдикції, вони можуть бути менш регульовані, ніж традиційні фінансові установи.

Криптовалютні біржі також відрізняються від традиційних фінансових бірж тим, що вони зазвичай працюють цілодобово, сім днів на тиждень. Це відображає глобальну природу криптовалютного ринку, а також факт, що криптовалютні транзакції можуть бути виконані в будь-який час.Крім того, криптовалютні біржі також можуть надавати послуги зберігання, дозволяючи користувачам тримати свою криптовалюту на біржі. Однак це також означає, що користувачі повинні

довіряти біржі свої приватні ключі, що є основою криптографічної безпеки в блокчейні. Це може створювати ризик, якщо біржа стане жертвою хакерської атаки, і ці приватні ключі будуть вкрадені.

Деякі криптовалютні біржі також пропонують послуги маржинальної торгівлі. Маржинальна торгівля дозволяє трейдерам використовувати позики для торгівлі більшою кількістю криптовалюти, ніж вони фактично володіють. Це може збільшити потенційний прибуток, але також збільшує ризик збитків.Ще однією особливістю деяких криптовалютних бірж є наявність токенів біржі. Ці токени можуть бути використані для отримання знижок на комісії за торгівлю, участі в голосуванні за нові функції біржі або навіть отримання частини прибутків біржі.

Важливо розуміти, що криптовалютні біржі - це високоризиковані платформи, які потребують досконалого розуміння технічних аспектів торгівлі та безпеки. При виборі біржі для торгівлі криптовалютами, трейдерам слід уважно вивчити її репутацію, безпеку, комісійні збори та доступність торговельних пар.

2 ЗБІР ТА ОБРОБКА ДАННИХ

2.1 Збір цін біткоїну

Інформацію про ціни було взято з сайту біржі Binance, оскільки на даний момент, це найбільша крипто-біржа. За допомогою їх відкритого API, було зібрані погодинні свічки, що містять таку інформацію:

a) Дата (час початку години, за яку було забрано інформацію)
b) Ціна відкриття (ціна на момент початку години)
c) Найвища ціна (найвища ціна за дану годину)
d) Найнижча ціна (найнижча ціна за дану годину)
e) Ціна закриття (ціна на момент кінця години)
f) Об’єм торгів (Сума, яка була залучена в торгах пари BTC\USD за дану годину)

	date	open	high	low	close	volume
0	2022-01-01 00:00:00	46202.9	46720.7	46202.8	46634.5	3148.967762
1	2022-01-01 01:00:00	46634.5	46946.6	46571.3	46774.9	2287.477036
2	2022-01-01 02:00:00	46774.8	46904.8	46698.5	46790.5	1105.248448
3	2022-01-01 03:00:00	46790.5	46877.6	46752.8	46795.2	901.866279
4	2022-01-01 04:00:00	46795.3	46849.6	46600.0	46687.3	1814.473982
...	...	...	...	...	...	...
8876	2023-01-05 20:00:00	16852.8	16854.0	16836.1	16845.7	494.838564
8877	2023-01-05 21:00:00	16845.7	16847.9	16826.2	16836.5	551.708257
8878	2023-01-05 22:00:00	16836.5	16843.7	16823.3	16823.4	435.089907
8879	2023-01-05 23:00:00	16823.3	16830.3	16809.5	16816.5	946.626674
8880	2023-01-06 00:00:00	16816.6	16857.6	16816.5	16846.5	714.812390

8881 rows × 7 columns

Рисунок 1.1. Зібрані дані у вигляді таблиці

Таким чином було зібрано 8881 рядків з даними (код програми у Додатку).

2.2 Нормалізація цін

Для коректної роботи моделей, які будуть використані, дані було нормалізовано (приведено до проміжку [0, 1]). Було розглянуто декілька варіантів нормалізації, наприклад перевести данні так, щоб мат. очікування та дисперсія дорівнювали 0.5. Проте найкращі результати при перших спробах навчити моделі були отримані за при “Мін-Макс Нормалізації”. Її суть полягає в тому, що з вибірки даних обирається найменше (далі min) та найбільше (далі max) значення, після чого до всіх елементів застосується наступна функція $f(x) = \frac{x-min}{max}$. Даний метод було застосовано для кожної колонки даних. Результат можна побачити на рисунках 1.2 та 1.3.

	open	high	low	close
0	0.945254	0.954426	0.950147	0.958596
1	0.958594	0.961385	0.961528	0.962936
2	0.962931	0.960098	0.965457	0.963418
3	0.963416	0.959260	0.967134	0.963563
4	0.963564	0.958397	0.962415	0.960228
...	...	...	...	...
8876	0.038068	0.034298	0.043120	0.037822
8877	0.037848	0.034110	0.042814	0.037537
8878	0.037564	0.033981	0.042725	0.037132
8879	0.037156	0.033568	0.042299	0.036919
8880	0.036949	0.034409	0.042515	0.037846

8881 rows × 5 columns

Рисунок 1.2. Дані після нормалізації

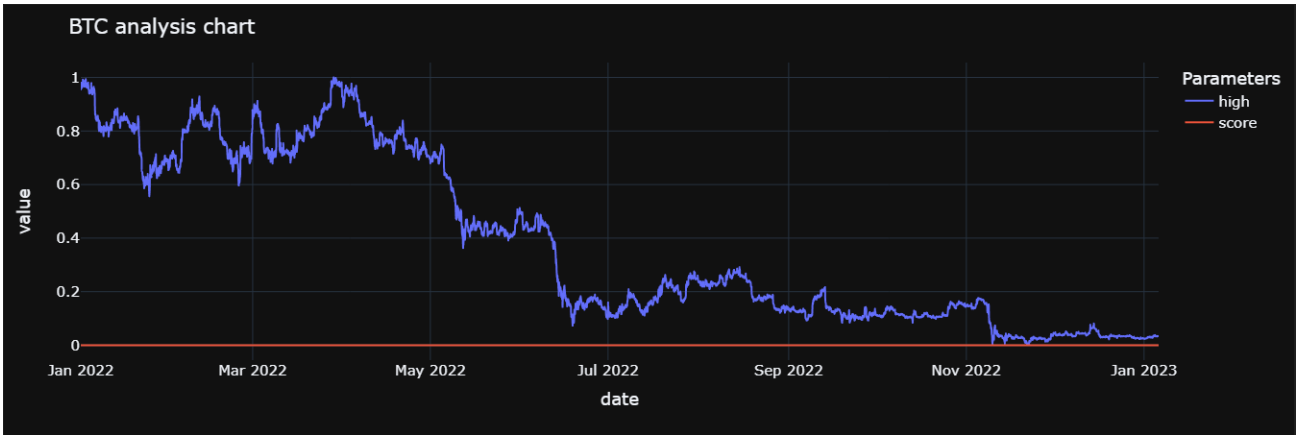

Рисунок 1.3. Графік ціни після нормалізації

3 ARIMA МОДЕЛЬ

3.1 AR модель

Модель авторегресії (AR) прогнозує майбутню поведінку на основі даних про минулу поведінку, визначаючи лінійну залежність між значеннями часового ряду та попередніми значеннями цього ж ряду. Цей тип аналізу використовується, коли існує відповідна кореляція між значеннями часових рядів. Авторегресійне моделювання виконується на основі даних із поточного ряду на основі одного або кількох минулих значень того самого ряду.

При використанні авторегресійної моделі припускають, що минулі значення часових рядів впливають на поточні значення. Саме тому статистична техніка широко використовується для аналізу природних явищ, економічних процесів та інших процесів, які змінюються з часом. Багато регресійних моделей використовують лінійні комбінації вхідних даних для прогнозування результату. На противагу їм, моделі авторегресії використовують минулі значення вихідних даних для визначення майбутніх прогнозів.

Оскільки авторегресійні моделі прогнозують ціни лише на основі минулої інформації, передбачається, що основи залишаються незмінними. У результаті передбачення можуть бути неточними або несподіваними, якщо основні рушійні сили системи змінюються, особливо коли галузь швидко трансформується в технологічному плані.

Основними перевагами AR-моделі є можливість визначати рекурентні закономірності в даних та здійснення передбачень за меншою кількістю вхідних даних завдяки self-variable series (серії змінних самої себе; замість використання додаткових вхідних даних, модель використовує лише попередні значення

змінної для прогнозування майбутніх значень; слід зауважити, що такий підхід має значні обмеження при можливості нових зовнішніх впливів на змінну).

Обмеження авторегресійної моделі. Для нормальної роботи моделі коефіцієнт автокореляції повинен бути не менше 0,5. Іншими слова, ступінь кореляції між минулими і майбутніми даними має бути досить значною, така модель добре підходить для прогнозування різноманітних динамік в сфері економіки, адже тут присутня значна залежність від минулих історичних подій та соціальних факторів.

Авторегресивна модель полягає в тому, що кожне наступне значення часового має лінійну залежність від минулих. Авторегресійний процес порядку p $(AR(p))$ виглядає наступним чином

$$X_t = a_0 + \sum_{i=1}^{p} a_i X_{t-i} + e_t,$$

де $a_0, a_1, \dots, a_p$ – коефіцієнти авторегресії, X_t – значення ряду в момент часу t, e_t – білий шум (послідовність незалежних та однаково розподілених, як правило нормально, величин)

3.2 MA модель

Модель ковзного середнього (MA-модель) є поширеним підходом для моделювання, аналізу та прогнозування часових рядів, пошуку тенденції в даних. MA-модель базується на припущенні, що поточне значення часового ряду залежить від попередніх значень помилок з відставанням. Модель ковзного середнього фіксує властивості стійкості та згладжування даних з урахуванням впливу попередніх значень помилок на поточні значення часового ряду.

Важливо зазначити, що моделі ковзного середнього не обмежуються одним терміном помилки з відставанням. Подібно до моделей AR, вони можуть включати кілька термінів помилки з відставанням, позначених як MA(q), де q представляє порядок моделі ковзного середнього. Кожен лагований член помилки (термін помилки з відставанням) множиться на відповідний коефіцієнт, щоб визначити його вплив на поточне значення. Для підбору оптимального порядку моделі (q) здійснюється за допомогою порівняння кількох моделей з різними порядками за критеріями, наприклад AIC (Критерій Акаіке) або BIC (Байєсівський інформаційний критерій).

Основна перевага MA-моделі - здатність враховувати короткострокові коливання та шум у часовому ряді, оскільки терміни помилки з відставанням згладжують шуми та зміни ряду.

При використанні MA-моделі слід враховувати декілька важливих аспектів, зокрема те, що модель базується на припущенні, що випадкова помилка має нульове середнє значення та стаціонарну коваріаційну структуру. Також слід враховувати, що для довгострокових прогнозів модель матиме порівняно меншу ефективність. Крім того, при аналізі і моделюванні за допомогою MA-моделі слід досліджувати автокореляційну функцію помилок (ACF). Її значення поза статистичною межею довіри можуть свідчити про присутність суттєвої кореляції у помилках, що, в свою чергу, може вказувати на неадекватність моделі MA.

Модель ковзної середньої (MA(q)) – це модель типу

$$X_t = \sum_{j=1}^{q} b_j \, e_{t-j},$$

де $b_1, \dots, b_q$– коефіцієнти авторегресії, e_{t-j} – це різниця між X_{t-j} та середнім значення ряду X_t.

3.3 ARIMA модель

У статистиці та економетриці, зокрема в аналізі часових рядів, модель авторегресійної інтегрованої ковзної середньої (ARIMA) є узагальненням моделі авторегресійної ковзної середньої (ARMA). Щоб краще зрозуміти дані або спрогнозувати майбутні точки ряду, обидві ці моделі адаптуються до даних часових рядів. Моделі ARIMA застосовуються в деяких випадках, коли дані показують докази нестаціонарності в сенсі середнього (але не дисперсії/автоковаріації), де початковий крок розрізнення (що відповідає «інтегрованій» частині моделі) може бути застосований один або більше разів, щоб усунути нестаціонарність середньої функції. Коли сезонність відображається в часовому ряді, сезонна різниця може бути застосована для усунення сезонної складової. Оскільки модель ARMA, згідно з теоремою декомпозиції Уолда, теоретично достатня для опису регулярного (він же суто недетермінованого) стаціонарного часового ряду широкого сенсу, ми мотивовані зробити стаціонарним а нестаціонарні часові ряди, наприклад, за допомогою розрізнення, перш ніж ми зможемо використовувати модель ARMA. Зауважте, що якщо часовий ряд містить передбачуваний підпроцес (він же чистий синус або комплексно-значний експоненціальний процес), передбачуваний компонент розглядається як ненульовий середній, але періодичний (тобто сезонний) компонент в ARIMA таким чином, щоб він був усунутий сезонною різницею.

Частина AR ARIMA вказує на те, що змінна, яка цікавить, регресує на її власні відсталі (тобто попередні) значення. Частина MA вказує на те, що помилка регресії насправді є лінійною комбінацією членів помилки, значення яких мали місце одночасно та в різний час у минулому. І вказує на те, що значення даних було замінено на різницю між їхніми значеннями та попередніми значеннями (і

цей процес розрізнення міг виконуватися більше одного разу). Мета кожної з цих функцій полягає в тому, щоб модель якомога краще відповідала даним.

Модель ARIMA(p, d, q) має вигляд

$$\Delta^d X_t = a_0 + \sum_{i=1}^{p} a_i \, \Delta^d X_{t-i} + \sum_{j=1}^{q} b_j \, e_{t-j} + e_t,$$

де $a_0, a_1, \dots, a_p, b_1, \dots, b_q$ – коефіцієнти авторегресії, Δ^d – оператор різниці часового ряду порядку d, e_{t-j} – це різниця між X_{t-j} та середнім значення ряду X_t та e_t – білий шум.

Програмна реалізація моделі для отримання результатів навчалась досить тривалий проміжок часу(більше декількох годин), що робить не практичним її використання у задачі прогнозування ціни на наступну годину.

4 XGBOOST МОДЕЛЬ

4.1 Дерева рішень

Дерево рішень (дерево класифікації, регресійне дерево) — це ієрархічна модель підтримки прийняття рішень, яка використовує деревоподібну модель рішень та їхніх можливих наслідків, включаючи результати випадкових подій, витрати на ресурси та користь. Це один із способів відображення алгоритму, який містить лише умовні оператори керування. Дерева рішень зазвичай використовуються в дослідженні операцій і управлінні операціями, зокрема в інтелектуальному аналізі рішень, для визначення стратегій, які найімовірніше досягають мети; є популярним інструментом у машинному навчанні.

Дерево рішень — це структура, подібна до блок-схеми, у якій кожен внутрішній вузол представляє перевірку атрибута, кожна гілка представляє результат перевірки, а кожен листковий вузол представляє мітка класу. Кожен листок представляє значення цільової змінної, яка змінюється в процесі руху від кореня до листка. Кожен внутрішній вузол відповідає одній з вхідних змінних. Дерево також може бути "вивчено" шляхом розбиття вихідних наборів змінних на підмножини, засновані на тестуванні значень атрибутів. Рішення приймається після обчислення всіх атрибутів. Шляхи від кореня до листа представляють правила класифікації. Приклади дерев рішень зображені на рисунку 4.1.

В аналізі рішень, дерево рішень та тісно пов’язана з ним діаграма впливу використовуються як візуальний і аналітичний інструмент підтримки прийняття рішень, в якому обчислюються очікувані значення (або очікувана користь) конкуруючих варіантів. Дерево рішень складається з трьох типів вузлів:

- Вузли прийняття рішень
- Шансові вузли

- Кінцеві вузли

Якщо на практиці рішення повинні прийматися в режимі реального часу, без повторного відкликання в умовах неповного знання, дерево рішень має паралельно використовувати модель ймовірності як модель найкращого вибору (або алгоритм моделі онлайн-вибору. Інше використання дерев рішень - в ролі описового засобу для обчислення умовних ймовірностей.

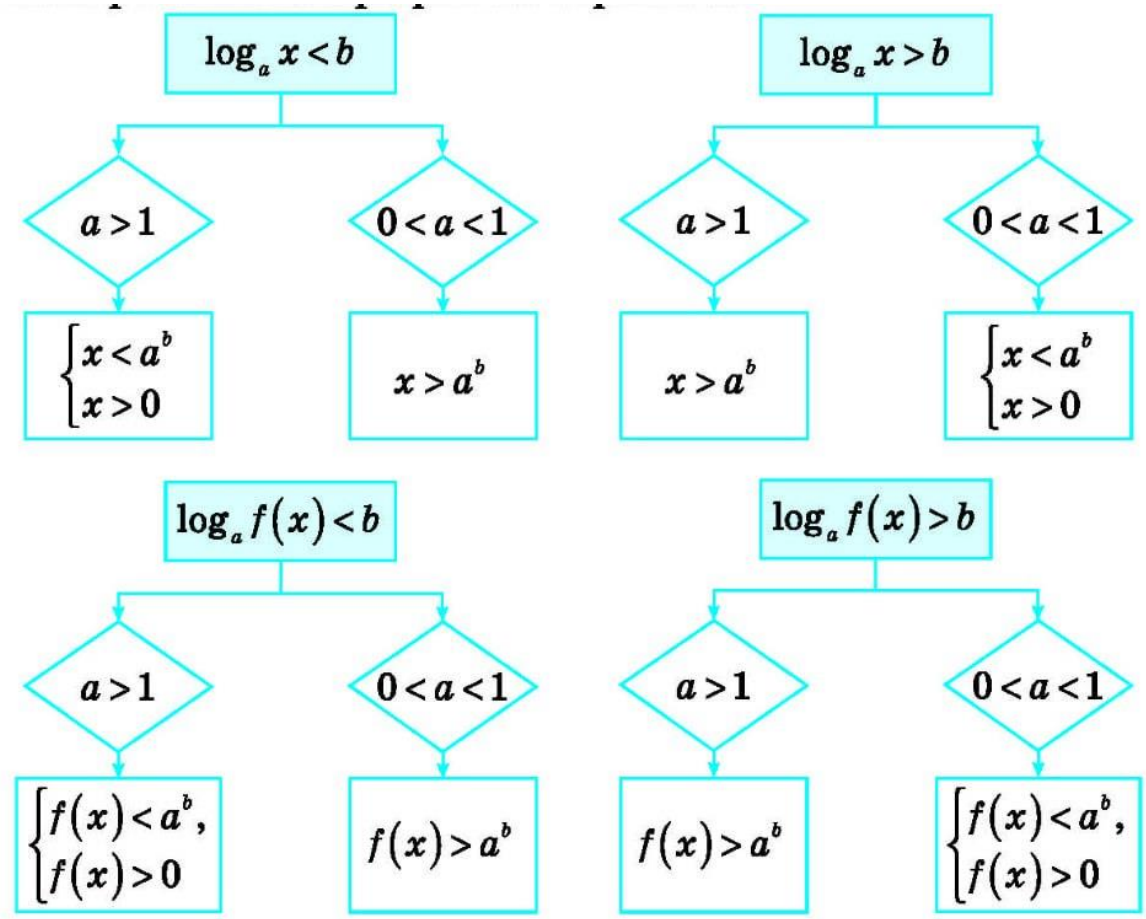

Рисунок 4.1 Приклади дерев рішень.

4.2 Boosting моделі

У машинному навчанні boosting (підсилювання) — це “ансамблевий” мета-алгоритм машинного навчання для зменшення зсуву, а також дисперсії у навчанні з учителем. Крім того, boosting визначається як сімейство алгоритмів машинного навчання, які перетворюють слабкі алгоритми навчання в сильні.

Слабкий алгоритм навчання визначається як класифікатор, який слабо корелює з правильною класифікацією (може позначати приклади краще, ніж випадкове вгадування). На відміну від слабкого алгоритму, сильний алгоритм навчання є класифікатором, добре корельованим з правильною класифікацією. Гіпотеза “бустигу” пов'язана з процесом налаштування “слабкого” алгоритму навчання для “строгого навчання”. Неформально запитують, чи випливає існування ефективного алгоритму навчання, виходом якого є гіпотеза, ефективність якої лише трохи краща за ефективність випадкового вибору, існування ефективного алгоритму, який дає гіпотезу довільної точності. Алгоритми, які отримують таку гіпотезу, швидко стають відомі просто як «бустинг».

4.3 XGBoost моделі

XGBoost (eXtreme Gradient Boosting) — це бібліотека програмного забезпечення з відкритим вихідним кодом (одна з найпопулярніших і ефективніших реалізацій алгоритму градієнтного підсилення на деревах), яка надає регуляризаційну структуру підвищення градієнта для C++, Java, Python та низки іншим мов. Він базується на концепції градієнтного підсилення, яка полягає у послідовному навчанні слабких моделей, кожна з яких коригує попередню модель у напрямку антиградієнту “функції втрати” (loss function).

В основі XGBoost лежить алгоритм градієнтного бустингу дерев рішень. Градієнтний бустинг - це техніка машинного навчання для завдань класифікації та регресії, яка будує модель передбачення у формі ансамблю слабких моделей передбачень, зазвичай дерев рішень. Навчання ансамблю проводиться послідовно на відміну, наприклад, від беггинга. На кожній ітерації обчислюються відхилення передбачень уже навченого ансамблю на навчальній вибірці. Наступна модель, яка буде додана в ансамбль, буде передбачати ці відхилення. Таким чином, додавши передбачення нового дерева до передбачень навченого

ансамблю, ми можемо зменшити середнє відхилення моделі, яке є таргетом оптимізаційної задачі. Нові дерева додаються в ансамбль доти, доки помилка зменшується, або доки виконується одне з правил "ранньої зупинки" .

XGBoost використовує методи оптимізації градієнтного спуску, такі як регуляризація та стохастичний градієнтний спуск, для покращення швидкості та точності навчання моделей. Він використовує дерева рішень як базові моделі, оскільки вони можуть моделювати складні нелінійні залежності між вхідними змінними.Алгоритм XGBoost використовує різні техніки для уникнення перенавчання, такі як обмеження глибини дерев, мінімізація функції втрати з використанням регуляризації та розподілу Хі-квадрат для обрізки дерев. Це дозволяє моделі XGBoost піддаватися ефективному навчанню з великими наборами даних, уникати перенавчання та досягати високої точності прогнозування.Одна з ключових особливостей XGBoost полягає в його здатності до автоматичного визначення ваги ознак, що дозволяє використовувати його для важливості ознак та відбору ознак. Він також підтримує паралельну обробку, що робить його швидким та масштабованим для обробки великих обсягів даних.

XGBoost також має вбудовану підтримку категоріальних ознак, оптимізацію розподілу пам'яті для ефективної роботи з великими наборами даних, а також можливість використовувати розподілене обчислення для розпаралелювання навчання моделі та прискорення процесу. Завдяки своїм потужним можливостям та ефективності, XGBoost широко використовується в різних сферах, включаючи розпізнавання образів, рекомендаційні системи, аналітику даних, фінансове прогнозування, біоінформатику та багато інших областей, де важлива точність та масштабованість моделей машинного навчання.

4.2 Програмна реалізація

Модель була реалізована за допомогою бібліотеки XGBoost на мові програмування python версії 3.10 (код програми можна знайти в додатку). Модель зупиняла навчання після того, як побудувала тисячу дерев. Максимальна довжина кожного з побудованих дерев була 3 вузли. Learning rate (Це коефіцієнт швидкості навчання, який контролює внесок кожного дерева в кінцеву модель) був взятий за одну соту.

На етапі тренування на вхід модель отримувала одну свічку і на виході від неї очікувалось побачити найвище значеня ціни на наступну годину. На тестових даних модель показала похибку RMSE приблизно рівну 0.03641.

Тепер дана похибка буде братись як найбільша прийнятна похибка, при оцінюванні наступних моделей.

4 LSTM МОДЕЛЬ

4.1 Нейронні мережі

Нейронні мережі — це комплексні моделі обчислювальних систем, що базуються на ідеї та принципах функціонування біологічних нейронних мереж. Такі системи складаються з штучних структурних одиниць - нейронів, що зв'язані між собою та є аналогією біологічним нейронам.

Нейронні мережі вирішують задачі з допомогою обробки вхідних даних мережею нейронів, де сигнал передається між зв'язаними вузлами. Біологічна аналогія цього процесу - передача сигналів у нервовій системі від одного нейрона до іншого через синапси. Нейронні мережі навчаються виявляючи закономірності і залежності в навчальних прикладах і потім застосовують їх для вирішення завдань. Навчання відбувається на великих об'ємах даних, що попередньо обробляються та нормалізуються. Після цього настає етап розробки архітектури, що є оптимальною для вирішення поставленої задачі. Ваги мережі ініціалізуються випадковим чином, далі вхідні дані обробляються мережею, обраховуються вихідні значення. За допомогою функції втрат обчислюється помилка мережі (різниця між вихідними значеннями, отриманими від мережі, та очікуваними значеннями), завдяки якій здійснюється коригування вагів методом зворотного поширення помилки. Застосування алгоритмів оптимізації, таких як градієнтний спуск, дозволяє скорегувати синаптичні ваги і підвищити точність моделі. Цей процес повторюється, оновлюючи параметри та покращуючи точність моделі. Кінцевим етапом є оцінка та валідація моделі з використанням окремих наборів даних (тестовий набір даних). Оптимізація та постійне навчання нейронної мережі також є необхідним етапом для покращення ефективності моделі в реальних умовах.

Нейронні мережі широко використовуються в багатьох сферах, зокрема комп'ютерне зорове сприйняття (розпізнавання об'єктів, виявлення образів і аналізу відео-змісту), обробка природної мови, діагностики та прогнозування в медицині, рекомендаційній сфері, прогнозування фінансових ринків, кібербезпеці та наукових дослідженнях різного роду. Нейронні мережі є потужним інструментом для обробки і аналізу даних у великому масштабі, точного прогнозування та виявлення складних закономірностей, що знаходить застосування в різних галузях життя.

4.2 Рекурентні нейронні мережі

Рекурентні нейронні мережі є классом нейронних мереж, що призначені для роботи з послідовними даними та дозволяють використовувати інформацію з попередніх кроків (виходів) як вхідні дані до наступних кроків, маючи приховані стани та зворотні зв'язки. Завдяки цьому RNN можуть використовувати контекст для вирішення задач, що дозволяє моделювати залежності в часових послідовностях.

Моделі RNN в основному використовуються в області обробки природної мови та розпізнавання мовлення (розпізнавання контекстно-чутливих мов, покращення машинного перекладу та багатомовної обробки мов), аналізу часових рядів тощо.

Структура RNN основана на ідеї застосування однієї й тієї ж самої структури на кожному кроці та забезпечення зворотного зв'язку, завдяки якому інформація з попередніх кроків передається у поточний. Основний компонент RNN - це рекурентний шар, який складається з нейронів, відомих як "рекурентні нейрони", кожен з яких здійснює обробку даних на основі вхідних даних з поточного кроку вхідної послідовності та попереднього стану.

Існує кілька типів архітектури RNN, найпоширенішими є прості RNN, LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit). LSTM та GRU стали вирішенням проблеми зникнення градієнту, що полягає в тому, що в деяких випадках,під час зворотного поширення помилки, градієнт стає занадто малим андто швидко, що унеможливлює відповідні зміни значень ваг. Це може призвести до того, що модель не зможе продовжувати тренуватись. LSTM і GRU мають більш складнішу структуру, що дозволяє керувати доступом до попередньої інформації.

4.3 Опис LSTM моделі

LSTM вирішує основну проблему RNN, а саме "зникаючий градієнт" (vanishing gradient), який унеможливлює збереження довгострокових залежностей в даних. Архітектура LSTM дозволяє зберігати необхідну інформацію та відкридати ту, що не несе цінності для вирішення проблеми. Ця структура забезпечує ефективне визначення довгострокових залежностей в послідовних даних. Крім того, LSTM однаково ефективно обробляє послідовності різної довжини, що недоступно для простих RNN, де вхідні послідовності мають фіксовану довжину.

Фактично, усі задачі, що потребують пошуку та аналізу довгострокових залежностей, ефективно вирішуються за допомогою LSTM, проте не під силу RNN. При обробці довгих послідовностей, LSTM краще фільтруюють дані, враховують та передають лише необхідні фрагменти, що дозволяє контролювати потік інформації у моделі. Крім того, вони здатні вловлювати складні ієрархічні структури та враховувати різні рівні залежностей для прийняття рішень. На противагу цьому, прості RNN не мають механізмів для зменшення впливу шуму та виявлення незначущих даних, що можуть призвести до зниження точності моделювання, а також оперують даними лише на одному рівні.

LSTM (Long Short-Term Memory) є одним з типів рекурентних нейронних мереж (RNN), які здатні зберігати попередній стан та використовувати його для вирішення завдань, що потребують обробки послідовностей даних, завдяки будові нейронів.

Основний принцип роботи LSTM полягає в тому, що кожен LSTM-нейрон має внутрішній стан, який він може зберігати та модифікувати з часом. Крім того, LSTM має три внутрішні вентилі, що відповідають за контроль потоку інформації в мережі.

Перший вентиль - "вентиль забуття" - вирішує, яка інформація має бути забута, а яка збережена у вигляді внутрішнього стану нейрону. Він використовується для відфільтровування непотрібної інформації, що може завадити мережі.

Другий вентиль - "вентиль входу" - визначає, яка нова інформація має бути додана до внутрішнього стану нейрону. Він вирішує, які значення вхідних даних мають бути враховані в мережі.

Третій вентиль - "вентиль виходу" - визначає, який внутрішній стан мережі повинен бути використаний для вирішення поточної задачі. Він відповідає за те, яка частина внутрішнього стану має бути відправлена на вихід мережі.

LSTM-нейрони дозволяють мережі зберігати та використовувати довготривалу інформацію з попередніх етапів обробки даних. Це забезпечує більш ефективну обробку послідовностей даних та використання інформації, що дуже корисно в багатьох задач. Схема нейрону зображена на рисунку 4.1.

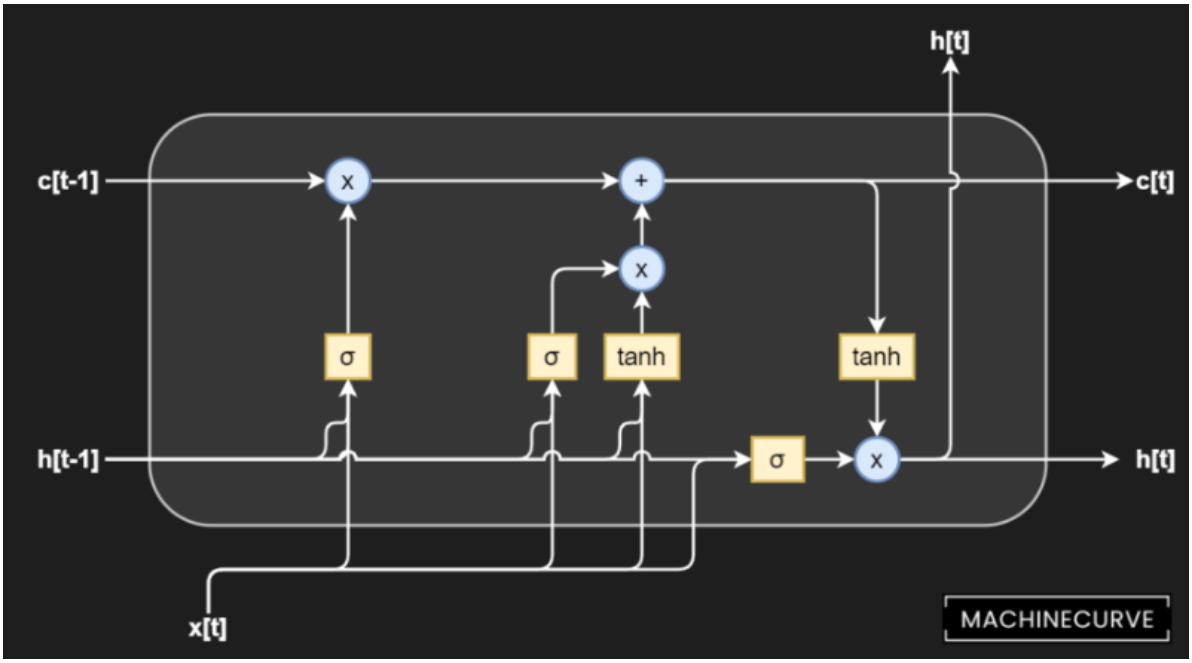

Рисунок 4.1 Схема нейрону в моделі LSTM

Обрахунки, що відбуваються в середині одного нейрону можна описати так:

$$f_t = \sigma_g\left(W_f x_t + U_t h_{t-1} + b_f\right),$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i),$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o),$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_g(W_c x_t + U_c h_{t-1} + b_c),$$

$$h_t = o_t \circ \sigma_h(c_t),$$

де $\circ$ - поелементний добуток;

h_{t-1} – вихід з нейрону у момент часу $t-1$;

x_t – вхідні дані у момент часу t;

c_{t-1} – стан(“те що в пам’яті”) нейрону у момент часу $t-1$;

i_t – вентель входу;

f_t – вентель забуття;

o_t – вентель виходу;

W,U,b – параметри, знайдені при навчані моделі;

$\sigma_g = \frac{e^x}{e^x+1}$ - сигмоїдна функція;

$\sigma_h = \frac{e^x-e^{-x}}{e^x+e^{-x}}$ - гіперболічний тангенс.

Тренування моделі відбувається за допомогою методу зворотного поширення похибки (backpropagation), тобто ваги кожного нейрону змінюються пропорційно тому, наскільки сильно кожен з них повпливав на похибку.

4.4 Додаткова підготовка даних

Дані цін на біткоїн, були поділені на 3 частини

1) Перші 60% - дані для тренування
2) Наступні 20% - дані для валідаії (ті за якими буде обраний оптимальний момент зупинки навчання)
3) Наступні 20% - тестові данні, ті які не приймали участі в навчанні моделі, на них буде обраховану остаточну похибку моделі

Після цього дані було згруповано наступним чином:

— Перший елемент вибірки містив в собі всі дані про ціну починаючи з 1-ї(за порядком) години, до 72-ї

— Другий елемент містив дані з 2-ї години, до 73-ї

— …

— N-й елемент містив дані з n-ї, до (n+71)-ї години

— …

Саме в такому вигляді дані будуть передаватись до моделі(програмна реалізація в додатку).

4.5 Програмна реалізація

Модель було написано на мові python версії 3.10 за допомогою бібліотеки PyTorch. Функцію помилок при навчанні було обрано RMSE (Корінь з суми квадратів). При навчанні на вхід до моделі давались данні про ціну за останні 72 години і на виході очікувалась найбільша ціна за наступну годину.

При написанні моделі було перевірено різні архітектури побудови (кількість слоїв нейронів, різні значення коефіцієнта навчання (learning rate) (відповідає за швидкість навчання; шлях впливу на те, наскільки сильно змінюються ваги після кожної ітерації), та кількість нейронів у слоях, додаткові слої не LSTM моделі, різні довжини послідовності годин, яка передається на вхід).

Найкращий результат на тестових даних показала модель LSTM з двома слоями по 720 та 72 нейрони в кожному, коефіцієнт навчання 0.001, навчання відбувалось 100 епох. Похибка RMSE на нормалізованих даних в середньому складає 0.00231(Результати для інших архітектур будуть в додатку)

5 LSTM З УРАХУВАННЯМ ТВІТІВ

5.1 Збір та обробка данних з Twetter

Датасет з твітами з хештегом #BTC було взято з сайту Kaggle(посилання в додатку). Очистка даних почалась з виключення рядків, які не мали тексту (видалялися записи, що складалися лише з хештегів). Далі були видалені твіти, зроблені людьми, у яких немає підписників, зазвичай це заблоковані на даний момент акаунти. Час створення твіту переведено до Unix формату, та округлено (вниз) до години, щоб поєднати з даними про ціну(код програми в додатку).

5.2 BERT та finBert

Bidirectional Encoder Representations from Transformers (BERT) — це сімейство моделей для обробки природної мови, представлене в 2018 році дослідниками Google. Опитування літератури 2020 року дійшло висновку, що «трохи більше ніж за рік BERT став повсюдною базою в експериментах з обробки природної мови (NLP), налічуючи понад 150 дослідницьких публікацій, які аналізують і вдосконалюють модель.

Ключовою технічною інновацією BERT є застосування двонаправленого навчання Transformer до мовного моделювання. Це відрізняється від попередніх спроб, які розглядали послідовність тексту зліва направо або поєднували навчання зліва направо та справа наліво. Результати статті показують, що мовна модель, яка навчається двонаправлено, може мати глибше відчуття мовного контексту, ніж однонаправлені мовні моделі. У статті дослідники описують нову техніку під назвою Masked LM (MLM), яка дозволяє двонаправлене навчання на моделях, у яких раніше це було неможливо.

BERT використовує Transformer, механізм уваги, який вивчає контекстні зв’язки між словами в тексті. Transformer містить два окремих механізми — кодер, який зчитує введений текст, і декодер, який створює прогноз для завдання. Оскільки метою BERT є створення мовної моделі, необхідний лише механізм кодування.

На відміну від спрямованих моделей, які зчитують введений текст послідовно (зліва направо або справа наліво), кодувальник Transformer зчитує всю послідовність слів одночасно. Тому його вважають двонаправленим, хоча точніше було б сказати, що він ненаправлений. Ця характеристика дозволяє моделі вивчати контекст слова на основі всього його оточення.

BERT, безсумнівно, є проривом у використанні машинного навчання для обробки природної мови. Той факт, що він доступний і дозволяє швидко тонко налаштувати, ймовірно, дасть змогу використовувати його для широкого спектру практичних застосувань у майбутньому.

Новітнім розширенням сімейства моделей BERT для обробки природної мови є finBERT - модель, спеціально розроблена для фінансових даних та текстів. Враховуючи специфічні особливості фінансової галузі, finBERT покликаний покращити розуміння фінансової мови та забезпечити точніші результати в фінансових завданнях.

Основою finBERT є архітектура BERT, з його двонаправленим навчанням та механізмом уваги Transformer. Однак, finBERT проходить додаткове попереднє навчання на спеціалізованих фінансових даних, які можуть включати новини, фінансові звіти, ринкові дані та інші відомості з фінансового сектору. Це дозволяє моделі стати більш чутливою до фінансових термінів, взаємозв'язків та нюансів, що специфічні для цієї галузі.

Застосування finBERT у фінансовому секторі може мати велике значення для аналізу ринків, прогнозування тенденцій, оцінки ризиків та прийняття рішень. Модель може допомогти зрозуміти настрої та емоційне забарвлення(далі настрій) фінансових новин, класифікувати фінансові документи, виявляти ключові фінансові події та багато іншого. finBERT поєднує силу BERT зі спеціалізованим фінансовим контекстом, що робить його потужним інструментом для роботи з фінансовими даними та текстами.

5.3 Обрахування настрою твітів за годину

Оскільки finBERT був навчений на новинах, написаних в офіційному стилі, його було дотреновано на вибірці з твітів. Дотреновуваня відбувалось 4 епохи, результати на тестових данних на рисунку 5.1.

Всі твіти було оцінено за допомогою finBERT. Далі всі твіти було згруповано за часов створення, на кожну годину, за яку було зібрано ціни на актив. Значення “настрою” людей щодо біткоїну у певну годину було оцінено за формулою

$$S = \sum_{k=1}^{n} f(t_k) * \frac{followers_k}{c}$$

де S – настрій за годину, n – кількість твітів, зібраних за цю годину, t_k- твіт під номером k за цю годину, f – модель finBERT, $followers_k$ – кількість підписників людини, яка зробила k-й твіт, c – константа, що означає середню кількість підписників у зібраних даних, та дорівнює приблизно 5000.

Таким чином, до даних з цінами будо додано колонку з настроєм.

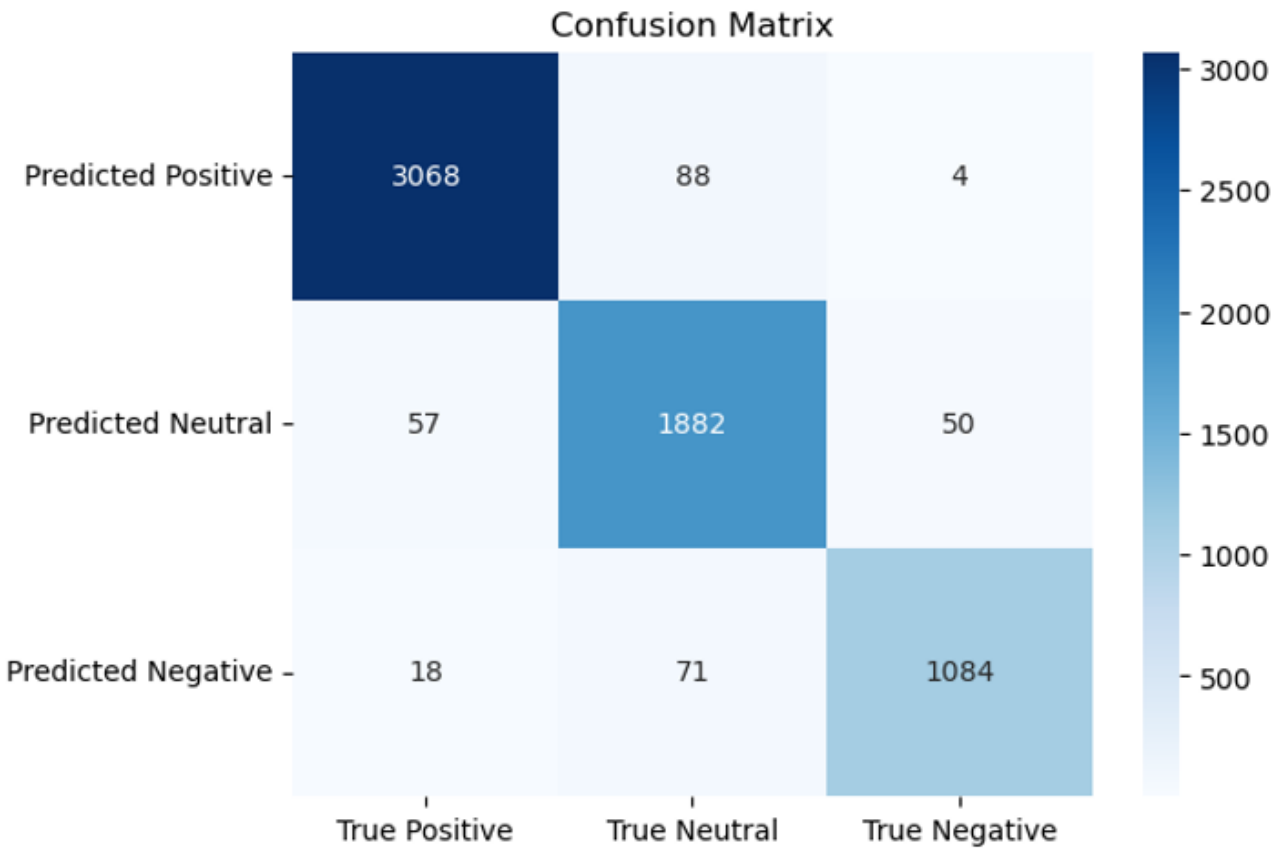

Рисунок 5.1. Результати довчання моделі finBERT

5.4 Програмна реалізація

Як і попередня модель, ця також була реалізована за допомогою бібліотеки PyTorch. Дані які даються на вхід моделі складаються з цін за останні 72 години, але тепер до них додається настрій вирахуваний по твітам за кожну годину.

У процесі розробки цієї моделі було випробовано багато різних архітектур, і виявилось, що вибір архітектури може значно вплинути на значення похибки, змінюючи її на порядок. Найкращі результати були показані моделлю з одного шару звичайних тензорів з 72 нейронами та двома LSTM шарами, в яких перший мав 720 нейронів, а другий - 72 нейрони. За результатами оцінки моделі, значення

квадратного кореня середньоквадратичної похибки (RMSE) склало 0.00263. (Результати інших архітектур цієї моделі в додатку)

ВИСНОВКИ

Було розглянуто та побудовано декілька моделей прогнозування часових рядів. Для кожної з моделей було підібрано найкращу архітектуру серед розглянутих, для прогнозування ціни біткоіну на наступну годину. XGBoots модель показала результати гірші, за інші моделі, проте її тренування займає в рази менше часу та не потребує великих обчислювальних потужностей. Найкращий результат показала LSTM модель проте підбір архітектури для найкращого результату зайняв великий проміжок часу.

Модель LSTM що враховувала емоційне забарвлення постів з соціальної мережі Twitter, на тему біткоіна, показала результати гірші, ніж модель що їх не враховувала. Це може бути пов’язано з тим, що повідомлення з цієї мережі не впливають(або впливають за мало) на зміну ціни криптовалюти. Проте даний підхід може гарно себе проявити в подібних задачах прогнозування часових рядів, де на значення що прогнозується сильний вплив мають соціальні чинники.

Крім врахування емоційного забарвлення, існує також можливість включення інших соціальних чинників у модель прогнозування. Наприклад, можна аналізувати обсяги торгівлі біткоіном на різних біржах, активність груп та спільнот, пов'язаних з криптовалютою, а також новини та події, пов'язані з регулюванням криптовалютного ринку.

Урахування цих соціальних чинників може забезпечити більш повний контекст для прогнозування ціни біткоїна. Наприклад, великий обсяг торгівлі може вказувати на значний інтерес до криптовалюти, що може впливати на її ціну. Активність груп та спільнот може відображати сентимент споживачів та трейдерів щодо біткоїна. Аналіз новин та регуляторних рішень може допомогти

прогнозувати можливі зміни у законодавстві або ринковій ситуації, що впливають на ціну криптовалюти.

У подальших дослідженнях можна розглядати комбінацію різних чинників, включаючи технічні показники, фундаментальні дані та соціальні чинники, для побудови комплексних моделей прогнозування. Це може забезпечити більш точні та надійні прогнози ціни біткоїна та інших криптовалют.

В цілому, розвиток моделей прогнозування ціни біткоїна та інших криптовалют є активною галуззю досліджень, і подальші вдосконалення та інтеграція різних чинників можуть покращити ефективність таких моделей.

СПИСОК ДЖЕРЕЛ ПОСИЛАНЬ

1. Jenkins G. M., Reinsel G. C., Box G. E. Time Series Analysis: Forecasting & Control. Pearson Education Asia Limited, 2005.
2. Kelleher J. D. Deep Learning. MIT Press, 2019. 296 p.
3. Chollet F. Deep Learning with Python. Manning Publications, 2017. 384 p.
4. Shmueli G. Practical Time Series Forecasting: A Hands-On Guide. CreateSpace Independent Publishing Platform, 2011. 84 p.
5. Horev R. BERT Explained: State of the art language model for NLP. *Medium*. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 (date of access: 16.05.2023).
6. Shinde R. Financial News Sentiment Analysis using FinBERT. *Medium*. URL: https://medium.com/@ravirajshinde2000/financial-news-sentiment-analysis-using-finbert-25afcc95e65f (date of access: 16.05.2023).
7. XGBoost Documentation – xgboost 1.7.5 documentation. *XGBoost Documentation – xgboost 1.7.5 documentation*. URL: https://xgboost.readthedocs.io/en/stable/ (date of access: 16.05.2023).
8. PyTorch documentation – PyTorch 2.0 documentation. *PyTorch*. URL: https://pytorch.org/docs/stable/index.html (date of access: 16.05.2023).
9. *arXiv.org e-Print archive*. URL: https://arxiv.org/ftp/arxiv/papers/2303/2303.09397.pdf (дата звернення: 16.05.2023).
10. Kahn F. Ultimate Beginners Guide to Cryptocurrencies: An Introduction to Cryptocurrencies and the Technologies That Powers Them. BookBaby, 2019.
11. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Incorporated, 2022.

ДОДАТОК

Увесь код з додатку, написаний на мові програмування Python, версії 3.10. Перед самим кодом вказані бібліотеки, використані для його написання(за умови, що вони не входять до стандартного набору бібліотек)

Функції, за допомогою яких було зібрано данні про ціну Bitcoin(з використанням бібліотеки requests)

```python
def get_binance_data(symbol: str, interval: str, start_time: dt.datetime | None = None, end_time: dt.datetime | None = None) -> list:
    url = f"https://api.binance.com/api/v3/klines?symbol={symbol}&interval={interval}"
    if start_time is not None:
        url += f"&startTime={int(start_time.timestamp() * 1000)}"
    if end_time is not None:
        url += f"&endTime={int(end_time.timestamp() * 1000)}"
    url += "&limit=1000"

    parsed_data = []
    while True:
        response = requests.get(url)
        if response.status_code != 200:
            raise Exception("Failed to retrieve data from Binance API")
        new_data = response.json()
        if not new_data:
            break
        data = []
        for d in new_data:
            data.append({
                "unix": d[0],
                "date": dt.datetime.fromtimestamp(d[0]/1000).astimezone(pytz.utc).strftime("%Y-%m-%d %H:%M:%S"),
                "symbol": symbol,
                "open": float(d[1]),
                "high": float(d[2]),
                "low": float(d[3]),
                "close": float(d[4]),
                "Volume BTC": float(d[5]),
                "Volume USDT": float(d[7]),
                "tradecount": int(d[8])
            })
        parsed_data += data
        start_time = dt.datetime.fromtimestamp(int(data[-1]["unix"]) / 1000) + dt.timedelta(seconds=1)
        url = f"https://api.binance.com/api/v3/klines?symbol={symbol}&interval={interval}"
        url += f"&startTime={int(start_time.timestamp() * 1000)}"
        if end_time is not None and start_time >= end_time:
            break
        time.sleep(1)

    return parsed_data
```

```python
def save_data(data: list[dict], filename: str) -> None:

    if not filename.endswith(".csv"):
        filename += ".csv"

    with open(filename, "w", newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter=",")
        writer.writerow(["unix", "date", "symbol", "open", "high", "low",
                         "close", "Volume BTC", "Volume USDT", "tradecount"])

        for row in data:
            writer.writerow([row["unix"], row["date"], row["symbol"], row["open"], row["high"], row["low"],
                             row["close"], row["Volume BTC"], row["Volume USDT"], row["tradecount"]])
```

Код, який запускає функції для збору даних

```python
if __name__ == "__main__":

    start_date = dt.datetime(year=2022,
                             month=11,
                             day=9,
                             hour=1,
                             minute=0,
                             second=0)
    end_date = dt.datetime(year=2023,
                           month=4,
                           day=15,
                           hour=19,
                           minute=0,
                           second=0)
    binance_data = get_binance_data("BTCUSDT", "1h", start_date, end_date)

    print(binance_data[0])
    for value in binance_data[0]:
        print(type(value))

    save_data(binance_data, "BTCUSDT_1m_2022-11-09_2023-04-15")
```

Візуалізація зібраних даних(за допомогою бібліотеки matplotlib)

```
column_names = cycle(['Open Price', 'Close Price', 'High Price', 'Low Price', 'Score'])

fig = px.line(data, x=data.date, y=[data['open'],
                                    data['close'],
                                    data['high'],
                                    data['low'],
            ],
            labels={'Date' : 'Date', 'value' : 'Stock value'})
fig.update_layout(title_text='BTC analysis chart',
                  font_size=15,
                  legend_title_text='Parameters')
fig.for_each_trace(lambda t: t.update(name=next(column_names)))
fig.show()
```

Python

Навчання XGBoost моделі(з використанням бібліотеки xgboost)

```
from xgboost import XGBRegressor
my_model = XGBRegressor(n_estimators=1000)
my_model.fit(X_train, y_train, verbose=False)
```

Обробка данних для LSTM моделі

```python
def lookback_split(dataframe, target_column, lookback=5):
    x = []
    y = []
    for i in range(lookback, len(dataframe) - lookback - 1):
        x.append(dataframe[i - lookback:i])
        y.append(dataframe[target_column].iloc[i])
    return np.array(x), np.array(y)
```

```python
def split(dataframe, target_column, train_size=0.6,
          valid_size=0.2, lookback=5, verbose=False):

    train_ind = int(train_size * len(dataframe))
    val_ind = train_ind + int(valid_size * len(dataframe))

    train = dataframe[:train_ind]
    val = dataframe[train_ind:val_ind]
    test = dataframe[val_ind:]

    x_train, y_train = lookback_split(train, target_column, lookback)
    x_val, y_val = lookback_split(val, target_column, lookback)
    x_test, y_test = lookback_split(test, target_column, lookback)

    if verbose:
        print("Shapes: ", "\n",
              "x_train", x_train.shape, "y_train", y_train.shape, "\n",
              "x_test", x_test.shape, "y_test", y_test.shape, "\n",
              "x_val", x_val.shape, "y_val", y_val.shape)

    return x_train, y_train, x_test, y_test, x_val, y_val
```

```python
lookback_size = 72
X_train, y_train, X_test, y_test, X_val, y_val = split(data,
                                                       'high',
                                                       lookback=lookback_size,
                                                       verbose=True)
```

```
Shapes:
 x_train (5183, 72, 5) y_train (5183,)
 x_test (1632, 72, 5) y_test (1632,)
 x_val (1631, 72, 5) y_val (1631,)
```

RMSE похибки різних архітектор RMSE

Parameters	Result RMSE
LSTM(72, 300)	
MinMax, lookback=72, lr=0.001, epochs=100, batch_size=1024	0.007609109
LSTM(72, 100)	
MinMax, lookback=72, lr=0.01, epochs=35, batch_size=32	0.002898689
MinMax, lookback=72. lr=0.01, epochs=100, batch_size=64	0.003295243
LSTM(72, Drop(0.1), 72, Drop(0.1))	
MinMax, lookback=72, lr=0.01, epochs=40, batch_size=32	0.048177189
MinMax, lookback=72, lr=0.02, epochs=40, batch_size=32	0.005654401
MinMax, lookback=72, lr=0.02, epochs=50, batch_size=32	0.004150682
MinMax, lookback=72, lr=0.001, epochs=35, batch_size=32	0.003676703
MinMax, lookback=72, lr=0.002, epochs=50, batch_size=32	0.003475409
MinMax, lookback=72, lr=0.0025, epochs=50, batch_size=32	0.025049408
LSTM(72, Drop(0.1), 144, Drop(0.1), 72, Drop(0.1))	
MinMax, lookback=72, lr=0.001, epochs=100, batch_size=32	0.013764886
MinMax, lookback=72, lr=0.002, epochs=50, batch_size=32	0.037114596
MinMax, lookback=72, lr=0.0001, epochs=100, batch_size=32	0.005796443
MinMax, lookback=72, lr=0.0001, epochs=100, batch_size=64	0.006327856
LSTM(720, 72)	
MinMax, lookback=72, lr=0.001, epochs=50, batch_size=64	0.041896869
MinMax, lookback=72, lr=0.001, epochs=70, batch_size=32	0.003271056
MinMax, lookback=72, lr=0.001, epochs=100, batch_size=32	0.002312905

Посилання на датасет з “твітами”:

https://www.kaggle.com/datasets/kaushiksuresh147/bitcoin-tweets

Очищення датасету з твітів

```python
def remove_empty_texts(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    return dataframe.dropna(subset=['text'])


def is_number(string: str) -> bool:
    """..."""
    regex = r'^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$'
    return bool(re.match(regex, string))


def remove_unfollowed_users(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    # Remove rows where 'user_followers' column contains string values
    dataframe = dataframe[dataframe['user_followers'].apply(lambda x: is_number(str(x)))]
    dataframe_copy = dataframe.copy()
    dataframe_copy['user_followers'] = dataframe['user_followers'].astype(float)

    # Filter out users with less than 10 followers
    return dataframe_copy[dataframe_copy['user_followers'] >= 10]
```

```python
def remove_only_hashtags(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    # create a regular expression to match hashtags
    hashtag_regex = re.compile(r'#\w+')

    # create a boolean mask to filter rows where 'text' contains only hashtags
    mask = dataframe['text'].apply(
        lambda x: len([word for word in x.split() if not bool(hashtag_regex.match(word))]) == 0
    )

    # remove rows where 'text' contains only hashtags
    dataframe = dataframe[~mask]

    return dataframe


def remove_irrelevant_tweets(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    # Fill missing values
    dataframe['hashtags'] = dataframe['hashtags'].fillna('[]')
    dataframe = dataframe[dataframe['hashtags'].astype(str).str.startswith('[')]
    # Convert string to list of hashtags
    dataframe_copy = dataframe.copy()
    dataframe_copy['hashtags'] = dataframe['hashtags'].apply(lambda x: ast.literal_eval(x))

    # create a list of irrelevant hashtags
    irrelevant_hashtags = ['giveaway', 'contest', 'sweepstakes',
                           'free', 'win', 'openfollow',
                           '500aday', '1000aday', 'autofollowback',
                           'followme', 'followgain', 'instantfollowback',
                           'mustfollow', 'f4f', 'retweet',
                           'teamhitfollow', 'follow', 'like4like',
                           'retweettrain', 'l4l', 'THF']

    # create a boolean mask to filter rows with irrelevant hashtags
    mask = dataframe_copy['hashtags'].apply(
        lambda x: any(hashtag.lower() in irrelevant_hashtags for hashtag in x)
    )

    # remove rows with irrelevant hashtags
    return dataframe_copy[~mask]
```

```python
def round_date(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    dataframe['date'] = pd.to_datetime(dataframe['date'], format='%Y-%m-%d %H:%M:%S')
    dataframe['date'] = dataframe['date'].dt.round('H')
    return dataframe


def convert_user_verified(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    dataframe['user_verified'] = dataframe['user_verified'].apply(
        lambda x: 1 if x is True or x == 'True' else 0
    )
    return dataframe


def convert_user_followers(dataframe: pd.DataFrame) -> pd.DataFrame:
    """..."""
    dataframe['user_followers'] = pd.to_numeric(dataframe['user_followers'], errors='coerce')

    # drop rows with NaN values (i.e. non-float values)
    return dataframe.dropna(subset=['user_followers'])
```

```python
def hard_preprocess(sentence: str,
                    stop_words: list = None,
                    stemming: bool = True,
                    stemmer: any = None) -> str:
    """..."""
    if stop_words is None:
        stop_words = []
    # Remove replies (with "@")
    sentence = re.sub(r'@\w+|', '', sentence)
    if stemming:
        sentence = stemmer.stem(sentence)
        tokens = gensim.utils.simple_preprocess(sentence)
        tokens = [token for token in tokens if token not in stop_words
                  and token != " "
                  and token.strip() not in punctuation]
    else:
        tokens = gensim.utils.simple_preprocess(sentence)
        tokens = [token for token in tokens if (token not in
                                                gensim.parsing.preprocessing.STOPWORDS and
                                                token not in stop_words)]

    output = " ".join(tokens)
    return output


def collapse_dots(sentence: str) -> str:
    """..."""
    # Collapse sequential dots
    sentence = re.sub("\.+", ".", sentence)
    # Collapse dots separated by whitespaces
    all_collapsed = False
    output = sentence
    while not all_collapsed:
        output = re.sub(r"\.(( )*)\.", ".", sentence)
        all_collapsed = sentence == output
        sentence = output
    return output
```

```python
    # Remove rows with empty text
    dataframe = remove_empty_texts(dataframe)

    # Remove rows with only hashtags
    dataframe = remove_only_hashtags(dataframe)

    # Remove rows where 'user_followers' is less than 10
    dataframe = remove_unfollowed_users(dataframe)

    # Remove rows with irrelevant hashtags
    dataframe = remove_irrelevant_tweets(dataframe)

    # Convert date strings to Unix format
    dataframe = round_date(dataframe)

    # Convert 'user_verified' to int
    dataframe = convert_user_verified(dataframe)

    # Convert 'user_followers' to numerical
    dataframe = convert_user_followers(dataframe)

    # Drop useless columns
    dataframe.drop(['is_retweet', 'source', 'user_favourites',
                    'user_friends', 'user_name', 'user_location',
                    'user_description', 'user_created',
                    'hashtags', 'tweet_id'], axis=1, inplace=True, errors='ignore')

    stop_words = stopwords.words('english')
    stemmer = stem.PorterStemmer()
    # Hard text preprocessing
    dataframe['hard_cleaned_text'] = dataframe['text'].apply(lambda x: hard_preprocess(sentence=str(x),
                                                                                       stop_words=stop_words,
                                                                                       stemming=True,
                                                                                       stemmer=stemmer))

    # Light text preprocessing
    dataframe['soft_cleaned_text'] = dataframe['text'].apply(lambda x: soft_preprocess(str(x)))

    return dataframe.reset_index(drop=True)
```

Модель finBERT

```python
# https://huggingface.co/yiyanghkust/finbert-pretrain
model = BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-pretrain', num_labels=3)
tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-pretrain')
```

Python

Позхибки для різних архітектур LSTM моделі з урахуванням настрою

Parameters	Result RMSE
LSTM(720, 72)	
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.001, epochs=150, patience=120, batch_size=32, scheduler=None	0.00289495
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.001, epochs=200, patience=0, batch_size=32, scheduler=None	0.00383549
MinMax, lookback=72, dropout=0, lr=0.002, w_decay=0.001, epochs=100, patience=80, batch_size=32, scheduler=None	0.003571366
MinMax, lookback=72, dropout=0, lr=0.002, w_decay=0.001, epochs=200, patience=0, batch_size=32, scheduler=None	0.003665757
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0, epochs=200, patience=0, batch_size=32, scheduler=None	0.008023121
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.01, epochs=200, patience=0, batch_size=32, scheduler=None	0.009634713
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.0005, epochs=200, patience=0, batch_size=32, scheduler=None	0.002623461
MinMax, lookback=72, dropout=0, lr=0.002, w_decay=0.0002, epochs=200, patience=0, batch_size=32, scheduler=None	0.006780065
MinMax, lookback=72, dropout=0, lr=0.0001, w_decay=0.0005, epochs=300, patience=0, batch_size=64, scheduler=None	0.006306589
MinMax, lookback=72, dropout=0, lr=0.0001, w_decay=0.00005, epochs=300, patience=0, batch_size=64, scheduler=None	0.003905662
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.0001, epochs=300, patience=0, batch_size=32, scheduler=None	0.003123118
LSTM(720, 360, 72)	
MinMax, lookback=72, dropout=0, lr=0.0001, w_decay=0.0001, epochs=200, patience=0, batch_size=32, scheduler=None	0.010256641
MinMax, lookback=72, dropout=0, lr=0.0001, w_decay=0.001, epochs=200, patience=0, batch_size=32, scheduler=None	0.003285824
MinMax, lookback=72, dropout=0, lr=0.0001, w_decay=0.0001, epochs=200, patience=0, batch_size=32, scheduler=None	0.008965584
MinMax, lookback=72, dropout=0, lr=0.0002, w_decay=0.0001, epochs=200, patience=0, batch_size=32, scheduler=None	0.00631117
LSTM(72, 72, 72)	
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.001, epochs=200, patience=0, batch_size=32, scheduler=None	0.009801847
MinMax, lookback=72, dropout=0, lr=0.0005, w_decay=0.001, epochs=200, patience=0, batch_size=32, scheduler=None	0.010414687
LSTM(Dense(72), LSTM(64, drop=0.2), LSTM(32, drop=0.2), Dense(16))	
MinMax, lookback=72, lr=0.001, w_decay=0.0001, epochs=300, patience=0, batch_size=64, scheduler=None	0.0300782
MinMax, lookback=72, lr=0.005, w_decay=0.001, epochs=300, patience=0, batch_size=32, scheduler=None	0.019007018
MinMax, lookback=72, lr=0.0001, w_decay=0.0001, epochs=300, patience=0, batch_size=32, scheduler=None	0.006692299
MinMax, lookback=72, lr=0.0001, w_decay=0.00005, epochs=300, patience=0, batch_size=32, scheduler=None	0.025442555
MinMax, lookback=72, lr=0.0005, w_decay=0, epochs=100, patience=0, batch_size=32, scheduler=None	0.059104139
LSTM(Dense(72), LSTM(64), LSTM(32), Dense(16))	
MinMax, lookback=72, dropout=0, lr=0.0005, w_decay=0, epochs=100, patience=0, batch_size=32, scheduler=None (- outliers)	0.011870031
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0, epochs=100, patience=0, batch_size=32, scheduler=None (- outliers)	0.0162512
LSTM(Dense(72), LSTM(720), LSTM(72))	
MinMax, lookback=72, dropout=0, lr=0.001, w_decay=0.0001, epochs=100, patience=0, batch_size=32, scheduler=None (- outliers)	0.019000185
MinMax, lookback=72, dropout=0, lr=0.0005, w_decay=0.0001, epochs=200, patience=0, batch_size=32, scheduler=None (- outliers)	0.007064831