Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
магістра
(освітньо-кваліфікаційний рівень)
на тему

Аналіз великих даних для прогнозування ринкових трендів за допомогою Python

Виконав: студент 6 курсу, групи 602-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Овчаренко О. М.
(прізвище та ініціали)
Керівник Скакаліна О. В.. В.
(прізвище та ініціали)

Полтава – 2024 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
спеціальність 122 «Комп'ютерні науки»

на тему

«Аналіз великих даних для прогнозування ринкових трендів за допомогою Python»

Студента групи 602-ТН Овчаренка Олександра Миколайовича

Керівник роботи
кандидат технічних наук,
доцент Скакаліна О. в.

Завідувач кафедри
кандидат технічних наук,
доцент Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 79 с., 42 рисунки, 1 додаток, 24 джерела.

Об’єкт дослідження: програмне забезпечення, що реалізує методи та алгоритми машинного навчання, застосовані для аналізу та прогнозування ринкових трендів.

Мета роботи: оцінка ефективності різних алгоритмів машинного навчання у контексті прогнозування фінансових ринків і може слугувати основою для подальших досліджень та розробок у цій галузі; розробка програмного забезпечення, що реалізує методи та алгоритми машинного навчання, застосовані для аналізу та прогнозування ринкових трендів.

Методи: аналіз існуючих рішень, аналіз даних, глибинне навчання, штучний інтелект, обробка даних, моніторинг системи.

Ключові слова: PYTHON, ШІ, ГЛИБОКЕ НАВЧАННЯ, ДІАГРАМА, ГРАФІК, АНАЛІЗ ДАНИХ.

ANNOTATION

Qualification work of master's degree: 79 p., 42 figures, 1 application, 24 sources.

Object of study: software that implements machine learning methods and algorithms used to analyze and predict market trends.

The goal of the work: to assess the effectiveness of various machine learning algorithms in the context of forecasting financial markets and can serve as a basis for further research and development in this area; developing software that implements machine learning methods and algorithms used to analyze and forecast market trends.

Methods: analysis of existing solutions, web programming, image processing, data encryption, system monitoring.

Keywords: PYTHON, AI, DEEP LEARNING, DIAGRAM, GRAPH, DATA ANALYSIS.

ЗМІСТ

ВСТУП

Великі обсяги даних є невід'ємною частиною процесу прийняття рішень у різних галузях. Аналіз великих даних (Big Data) набуває все більшої актуальності в контексті прогнозування ринкових трендів, оскільки дозволяє виявити приховані закономірності та тенденції, які можуть суттєво вплинути на фінансові ринки.

Дипломна робота присвячена розробці проекту на мові Python, в якому застосовано машинне навчання для прогнозування часових рядів на основі великих обсягів фінансових даних. Зокрема, розглядаються методи машинного навчання, які дозволяють обробляти та аналізувати дані для точнішого прогнозування ринкових трендів.

У теоретичній частині роботи розглядаються основи машинного навчання, зокрема:

- основи машинного навчання: основні концепції та методи, що лежать в основі машинного навчання;
- керовані та некеровані алгоритми: види алгоритмів, їх відмінності та застосування в практиці;
- ефективність алгоритмів навчання: оцінка ефективності різних алгоритмів на основі їх точності та швидкості;
- узагальнення алгоритмів машинного навчання: як алгоритми можуть бути адаптовані до різних типів задач;
- гіперпараметри та набори перевірки: роль гіперпараметрів у налаштуванні моделей та важливість наборів перевірки для оцінки їх якості.
- основні поняття фондового ринку та його прогнозування: огляд ключових аспектів фондового ринку та методів його прогнозування;
- методи машинного навчання для прогнозування: різні підходи та техніки машинного навчання, що використовуються для прогнозування фінансових ринків.

Проект, реалізований у рамках цієї роботи, демонструє застосування машинного навчання для прогнозування ринкових трендів на основі аналізу великих обсягів даних. Використання Python дозволяє ефективно реалізувати алгоритми машинного навчання та обробляти часові ряди, що є критично важливим для точного прогнозування ринкових змін.

Робота спрямована на оцінку ефективності різних алгоритмів машинного навчання у контексті прогнозування фінансових ринків і може слугувати основою для подальших досліджень та розробок у цій галузі.

Об'єктом дослідження є фінансові ринки, зокрема акції компанії BBCA, та великі обсяги історичних даних про їх торгові операції. Це включає дані про відкриття, закриття, максимальні та мінімальні ціни акцій.

Суб'єктом дослідження є методи та алгоритми машинного навчання, застосовані для аналізу та прогнозування ринкових трендів. Це охоплює розробку та впровадження моделей LSTM для прогнозування часових рядів, масштабування даних, а також оцінку ефективності різних підходів до прогнозування в умовах великого обсягу даних.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ РОБОТИ

1.1. Основи машинного навчання

Алгоритм машинного навчання – це алгоритм, який здатний навчатися на основі даних. Але що ми маємо на увазі під навчанням? Том Мітчелл дає таке визначення: «Кажуть, що комп'ютерна програма навчається на досвіді E щодо деякого класу завдань T і показника ефективності P, якщо її продуктивність при виконанні завдань з класу T, що вимірюється P, покращується з досвідом E». Можна уявити собі дуже широке розмаїття досвіду E, завдань T і показників ефективності P, і в цій книзі ми не робимо жодних спроб дати формальне визначення того, що може бути використано для кожної з цих сутностей.

Натомість у наступних розділах наведено інтуїтивно зрозумілі описи та приклади різних типів завдань, показників ефективності та досвіду, які можна використовувати для побудови алгоритмів машинного навчання.

Машинне навчання дозволяє нам вирішувати завдання, які занадто складно вирішити за допомогою фіксованих програм, написаних і спроектованих людиною. З наукової та філософської точки зору машинне навчання цікаве тим, що розвиток нашого розуміння машинного навчання тягне за собою розвиток нашого розуміння принципів, які лежать в основі інтелекту.

У цьому відносно формальному визначенні слова «завдання» сам процес навчання не є завданням. Навчання – це наш засіб досягнення здатності виконувати завдання. Наприклад, якщо ми хочемо, щоб робот міг ходити, то ходити – це завдання. Ми можемо запрограмувати робота, щоб він навчився ходити, або ж спробувати безпосередньо написати програму, яка визначає, як ходити вручну. Завдання машинного навчання зазвичай описуються в термінах того, як система машинного навчання повинна обробити приклад.

Приклад – це набір характеристик, які були кількісно виміряні в якомусь об'єкті або події, які ми хочемо, щоб система машинного навчання обробила. Зазвичай ми представляємо приклад у вигляді вектора x ∈ Rn, де кожен елемент xi вектора є іншою характеристикою. Наприклад, характеристиками зображення зазвичай є значення пікселів на зображенні [1].

За допомогою машинного навчання можна розв'язувати багато типів задач. Деякі з найпоширеніших завдань машинного навчання включають наступні:

- класифікація. У цьому типі завдань комп'ютерну програму просять вказати, до якої з k категорій належить деяка вхідна інформація. Для розв'язання цієї задачі алгоритм навчання зазвичай просять обчислити функцію f : Rn → {1, . . . , k}.

Коли y = f (x), модель відносить вхідні дані, описані вектором x, до категорії, ідентифікованої числовим кодом y. Існують й інші варіанти задачі класифікації, наприклад, коли f виводить розподіл ймовірностей по класах.

Прикладом задачі класифікації є розпізнавання об'єктів, де на вхід подається зображення (зазвичай описане як набір значень яскравості пікселів), а на виході – числовий код, що ідентифікує об'єкт на зображенні. Наприклад, робот Willow Garage PR2 може виконувати функції офіціанта, розпізнаючи різні види напоїв і подаючи їх людям по команді.

Сучасне розпізнавання об'єктів найкраще досягається за допомогою глибокого навчання. Розпізнавання об'єктів – це та сама базова технологія, яка дозволяє комп'ютерам розпізнавати обличчя (Taigman та ін., 2014), що може бути використана для автоматичного позначення людей у фотоколекціях і дозволить комп'ютерам більш природно взаємодіяти зі своїми користувачами;

- класифікація з відсутніми вхідними даними. Класифікація стає більш складною, якщо комп'ютерній програмі не гарантовано, що кожне вимірювання у вхідному векторі завжди буде надано. Для того, щоб вирішити завдання класифікації, алгоритм навчання повинен визначити лише одну функцію, яка відображає вхідний вектор у категоріальний вихід. Коли деякі

вхідні дані можуть бути відсутніми, замість однієї функції класифікації, алгоритм навчання повинен вивчити набір функцій. Кожна функція відповідає класифікації x з різною підмножиною відсутніх вхідних даних. Подібні ситуації часто виникають у медичній діагностиці, оскільки багато видів медичних тестів є дорогими або інвазивними.

Один із способів ефективно визначити такий великий набір функцій – це вивчити розподіл ймовірностей за всіма відповідними змінними, а потім розв'язати задачу класифікації, відкинувши пропущені змінні. Маючи n вхідних змінних, ми можемо отримати всі 2n різних функцій класифікації, необхідних для кожного можливого набору відсутніх вхідних даних, але нам потрібно вивчити лише одну функцію, що описує спільний розподіл ймовірностей.

Багато інших задач також можна узагальнити для роботи з відсутніми вхідними даними; класифікація з відсутніми вхідними даними – лише один з прикладів того, що може зробити машинне навчання;

- регресія. У цьому типі завдань комп'ютерну програму просять передбачити числове значення за деякими вхідними даними. Для вирішення цієї задачі алгоритм навчання просять вивести функцію $f : Rn \rightarrow R$.

Цей тип завдань схожий на класифікацію, за винятком того, що формат виводу відрізняється. Прикладом регресійної задачі є передбачення очікуваної суми страхового відшкодування, яку може отримати застрахована особа (використовується для встановлення страхових премій), або передбачення майбутніх цін на цінні папери. Такі прогнози також використовуються в алгоритмічній торгівлі;

- транскрипція. У цьому типі завдань систему машинного навчання просять спостерігати за відносно неструктурованим представленням певних даних і транскрибувати їх у дискретну, текстову форму. Наприклад, в оптичному розпізнаванні символів комп'ютерній програмі показують фотографію, що містить зображення тексту, і просять повернути цей текст у вигляді послідовності символів (наприклад, у форматі ASCII або Unicode).

Google Street View використовує глибоке навчання для обробки адресних номерів у такий спосіб.

Іншим прикладом є розпізнавання мови, коли комп'ютерній програмі надається звуковий сигнал, і вона видає послідовність символів або ідентифікаційних кодів слів, що описують слова, які були вимовлені в аудіозаписі. Глибоке навчання є важливим компонентом сучасних систем розпізнавання мовлення, що використовуються у великих компаніях, включаючи Microsoft, IBM і Google;

- машинний переклад. У задачі машинного перекладу вхідні дані вже складаються з послідовності символів певною мовою, і комп'ютерна програма повинна перетворити їх на послідовність символів іншою мовою. Це зазвичай застосовується до природних мов, наприклад, перекладу з англійської на французьку. Глибоке навчання нещодавно почало мати важливий вплив на вирішення такого роду завдань;
- структурований вивід. До завдань структурованого виводу належать будь-які завдання, де вихідними даними є вектор (або інша структура даних, що містить кілька значень) з важливими взаємозв'язками між різними елементами.

Це широка категорія, яка охоплює описані вище завдання транскрипції та перекладу, а також багато інших завдань. Одним із прикладів є синтаксичний аналіз – відображення речення природної мови в дерево, яке описує його граматичну структуру, і позначення вузлів дерева як дієслів, іменників, прислівників тощо.

Іншим прикладом є попіксельна сегментація зображень, коли комп'ютерна програма відносить кожен піксель зображення до певної категорії. Наприклад, глибоке навчання можна використовувати для анотування розташування доріг на аерофотознімках (Mnih and Hinton, 2010).

Вихідні дані не обов'язково повинні так точно відображати структуру вхідних даних, як у цих завданнях на зразок анотацій. Наприклад, при

створенні підписів до зображень комп'ютерна програма спостерігає за зображенням і виводить речення природною мовою, що описує зображення.

Ці завдання називаються завданнями структурованого виводу, оскільки програма повинна виводити кілька значень, які тісно пов'язані між собою. Наприклад, слова, виведені програмою для створення підписів до зображень, повинні утворювати правильне речення;

- виявлення аномалій. У цьому типі завдань комп'ютерна програма просіює набір подій або об'єктів і позначає деякі з них як незвичні або нетипові.

Прикладом завдання виявлення аномалій є виявлення шахрайства з кредитними картками. Моделюючи ваші купівельні звички, компанія, що випускає кредитні картки, може виявити зловживання вашими картками. Якщо злодій викраде вашу кредитну картку або інформацію про неї, його покупки часто матимуть інший розподіл ймовірності за типами покупок, ніж у вас.

Компанія, що випускає кредитні картки, може запобігти шахрайству, заблокувавши рахунок, як тільки ця картка була використана для нехарактерної покупки;

- синтез і вибірка. У цьому типі завдань алгоритм машинного навчання просять згенерувати нові приклади, схожі на ті, що містяться в навчальних даних. Синтез і вибірка за допомогою машинного навчання можуть бути корисними для медіа-додатків, де художнику може бути дорого або нудно генерувати великі обсяги контенту вручну.

Наприклад, відеоігри можуть автоматично генерувати текстури для великих об'єктів або ландшафтів, замість того, щоб вимагати від художника ручного маркування кожного пікселя. У деяких випадках ми хочемо, щоб процедура дискретизації або синтезу генерувала певний тип результату на основі вхідних даних.

Наприклад, у задачі синтезу мовлення ми вводимо письмове речення і просимо програму видати звуковий сигнал, що містить розмовну версію цього

речення. Це свого роду завдання структурованого виводу, але з тим застереженням, що не існує єдиного правильного виводу для кожного входу, і ми явно прагнемо до великої кількості варіацій у виводі, щоб вивід виглядав більш природним і реалістичним;

- знаходження (імпутація) відсутніх значень. У цьому типі завдань алгоритму машинного навчання надається новий приклад x ∈ Rn, але з деякими входами xi з x, яких бракує. Алгоритм повинен передбачити значення відсутніх елементів;
- знешкодження. У цьому типі завдань алгоритм машинного навчання отримує на вході зіпсований приклад x˜ ∈ Rn, отриманий невідомим процесом пошкодження з чистого прикладу x ∈ Rn.

Модель-учень повинен передбачити чистий приклад x за його зіпсованою версією x˜, або в більш загальному випадку передбачити умовний розподіл ймовірностей p(x | x˜) ;

- оцінка щільності або оцінка масової функції ймовірності. У задачі оцінки щільності алгоритму машинного навчання потрібно вивчити функцію pmodel : Rn → R, де pmodel(x) можна інтерпретувати як функцію щільності ймовірності (якщо x неперервна) або масову функцію ймовірності (якщо x дискретна) на просторі, з якого були взяті приклади.

Щоб добре виконати таке завдання (ми уточнимо, що саме це означає, коли будемо обговорювати показники ефективності P), алгоритм повинен вивчити структуру даних, які він бачив. Він повинен знати, де приклади щільно кластеризуються, а де вони малоймовірні.

Більшість завдань, описаних вище, вимагають, щоб алгоритм навчання хоча б неявно фіксував структуру розподілу ймовірностей. Оцінка щільності дозволяє нам явно зафіксувати цей розподіл. В принципі, ми можемо виконувати обчислення над цим розподілом для вирішення інших завдань. Наприклад, якщо ми виконали оцінку щільності для отримання розподілу ймовірностей p(x), ми можемо використати цей розподіл для розв'язання задачі інтерполяції пропущених значень. Якщо значення xi відсутнє, а всі інші

значення, позначені через x-i, відомі, то ми знаємо, що розподіл для нього має вигляд p(xi | x -i).

На практиці оцінка щільності не завжди дозволяє розв'язати всі ці пов'язані задачі, тому що в багатьох випадках необхідні операції над p(x) є обчислювально нерозв'язними.

Звісно, можливі й інші завдання та типи завдань. Типи завдань які ми перерахували тут, призначені лише для того, щоб навести приклади того, що може робити машинне навчання, а не для того, щоб машинне навчання і не для визначення жорсткої таксономії завдань [2].

1.2. Керовані та некеровані алгоритми

Алгоритми машинного навчання можна умовно поділити на некеровані та керовані, залежно від того, який досвід їм дозволено отримати в процесі навчання. Більшість алгоритмів навчання можна зрозуміти як такі, що мають можливість працювати з цілим набором даних. Набір даних – це сукупність багатьох прикладів.

Одним з найстаріших наборів даних, які вивчають статистики та дослідники машинного навчання, є набір даних Iris (Fisher, 1936). Це колекція вимірювань різних частин 150 рослин ірису. Кожна окрема рослина відповідає одному прикладу. Ознаками кожного прикладу є виміри кожної з частин рослини: довжина чашолистка, довжина чашечки, висота чашечки, довжина рослини: довжина чашолистка, ширина чашолистка, довжина пелюстки та ширина пелюстки. Набір даних також записує, до якого виду належить кожна рослина. У наборі даних представлено три різні види представлено три різних види.

Алгоритми навчання без вчителя працюють з набором даних, що містить багато ознак, а потім вивчають корисні властивості структури цього набору даних. В контексті глибокого навчання ми зазвичай хочемо вивчити весь

розподіл ймовірностей, який згенерував набір даних, чи то явно, як в оцінці щільності, чи то неявно для таких завдань, як синтез або згладжування.

Деякі інші алгоритми неконтрольованого навчання виконують інші функції, наприклад, кластеризацію, яка полягає в поділі набору даних на кластери схожих прикладів.

Алгоритми керованого навчання опрацьовують набір даних, що містить ознаки, але кожен приклад також пов'язаний з міткою або ціллю.

Наприклад, набір даних Iris анотований з видами кожної рослини ірису. Алгоритм керованого навчання алгоритм може вивчити набір даних Iris і навчитися класифікувати рослини ірису на три різних видів на основі їхніх вимірів. Грубо кажучи, неконтрольоване навчання передбачає спостереження за кількома прикладами випадкового вектора x і намагається явно або неявно вивчити розподіл ймовірностей p(x) або деякі цікаві властивості цього розподілу, тоді як контрольоване навчання передбачає спостереження за кількома прикладами випадкового вектора x та пов'язаного з ним значення або вектора y, і навчання передбачати y від x, як правило, шляхом оцінюючи p(y|x).

Термін контрольоване навчання походить від уявлення про те, що що цільовий показник y надається інструктором або вчителем, який показує системі машинного показує системі навчання, що робити. У неконтрольованому навчанні немає ні інструктора, ні або вчителя немає, і алгоритм повинен навчитися осмислювати дані без цього керівництва.

Навчання без інструктора і навчання з інструктором не є формально визначеними термінами. Межі між ними часто розмиті. Багато технологій машинного навчання можна можна використовувати для виконання обох завдань.

Наприклад, ланцюгове правило ймовірності стверджує що для вектора x ∈ Rn спільний розподіл можна розкласти у вигляді формули:

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(\mathrm{x}_i \mid \mathrm{x}_1, \ldots, \mathrm{x}_{i-1})$$

Така декомпозиція означає, що ми можемо розв'язати нібито некеровану задачу моделювання p(x), розбивши її на n керованих задач навчання [3].

Альтернативно, ми можемо розв'язати задачу навчання p(y | x) під наглядом, використовуючи традиційні технології неконтрольованого навчання для вивчення спільного розподілу p(x, y) і виведення у вигляді формули:

$$p(y \mid \mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')}.$$

Хоча неконтрольоване навчання та контрольоване навчання не є повністю формальними або поняттями, вони допомагають приблизно класифікувати деякі речі, які ми робимо з алгоритмами машинного навчання.

Традиційно люди називають регресію, класифікацію та завдання структурованого виводу керованим навчанням. Оцінювання щільності в оцінки щільності для підтримки інших завдань зазвичай вважається неконтрольованим навчанням. Можливі й інші варіанти парадигми навчання.

Наприклад, у напівкерованому навчанні деякі приклади включають ціль контролю, а інші – ні. не включають. У навчанні з декількома прикладами вся колекція прикладів позначається як така, що як така, що містить або не містить приклад класу, але окремі члени колекції не позначаються. Деякі алгоритми машинного навчання працюють не лише з фіксованим набором даних.

Наприклад, алгоритми навчання з підкріпленням взаємодіють з навколишнім середовищем, тому між системою навчання та її досвідом існує зворотний зв'язок. Такі алгоритми виходять за рамки цієї книги. Більшість алгоритмів машинного навчання просто досліджують набір даних. Набір даних можна можна описати різними способами.

У всіх випадках набір даних – це набір прикладів, які, в свою чергу, є наборами ознак. Одним з поширених способів опису набору даних є матриця дизайну.

Матриця дизайну – це матриця, що містить різні приклади в кожному рядку. Кожен стовпчик матриці відповідає окремій ознаці. Наприклад, набір

даних Iris містить 150 прикладів з чотирма характеристиками для кожного прикладу. Це означає, що ми можемо представити набір даних матрицею дизайну $X \in R^{150\times4}$, де Xi,1 – довжина чашолистка рослини i, Xi,2 – ширина чашолистка рослини i т.д.

Звичайно, щоб описати набір даних як навчальну матрицю, необхідно мати можливість описати кожен приклад як вектор, і кожен з цих векторів повинен бути однакового розміру. Це не завжди можливо. Наприклад, якщо у вас є колекція фотографій з різною шириною і висотою, то різні фотографії будуть містити різну кількість пікселів, тому не всі фотографії можна описати вектором однакової довжиною вектора.

У таких випадках замість того, щоб описувати набір даних як матрицю з m рядків, ми будемо описувати його як множину, що містить m елементів: {x(1), x(2), . . . , x(m) }.

Цей запис не означає, що будь-які два вектори прикладів x(i) і x(j) мають однаковий розмір. У випадку керованого навчання приклад містить мітку або ціль, а також набір ознак. Наприклад, якщо ми хочемо використати алгоритм навчання для розпізнавання об'єктів на фотографіях, нам потрібно вказати, який об'єкт зображено на кожній з них.

Це можна зробити за допомогою цифрового коду, де 0 означає людину, 1 – автомобіль, 2 – кішку і т.д.

Часто при роботі з набором даних, що містить дизайн-матрицю спостережень ознак X, ми також надаємо вектор міток y, де yi – це мітка для прикладу i. Звичайно, іноді мітка може бути більше, ніж просто одне число.

Наприклад, якщо ми хочемо навчити систему розпізнавання мови транскрибувати цілі речення, то міткою для кожного прикладу речення буде послідовність слів. Так само, як не існує формального визначення контрольованого і неконтрольованого навчання, не існує жорсткої таксономії наборів даних або досвіду. Описані тут структури охоплюють більшість випадків, але завжди можна розробити нові для нових застосувань [4].

1.3. Ефективність алгоритмів навчання

Для того, щоб оцінити можливості алгоритму машинного навчання, ми повинні розробити кількісну міру його продуктивності. Зазвичай цей показник P є специфічним для завдання T, яке виконує система.

Для таких завдань, як класифікація, класифікація з відсутніми вхідними даними та транскрипція, ми часто вимірюємо точність моделі. Точність – це просто частка прикладів, для яких модель дає правильні результати. Ми також можемо отримати еквівалентну інформацію, вимірявши частоту помилок, тобто частку прикладів, для яких модель видає неправильний результат.

Ми часто називаємо частоту помилок очікуваною втратою 0-1. Втрата 0-1 для конкретного прикладу дорівнює 0, якщо його правильно класифіковано, і 1, якщо ні. Для таких завдань, як оцінка щільності, немає сенсу вимірювати точність, частоту помилок або будь-які інші види втрат 0-1.

Замість цього ми повинні використовувати іншу метрику продуктивності, яка дає моделі безперервну оцінку для кожного прикладу. Найпоширенішим підходом є звіт про середню лог-вірогідність, яку модель присвоює деяким прикладам. Зазвичай нас цікавить, наскільки добре алгоритм машинного навчання працює на даних, яких він раніше не бачив, оскільки це визначає, наскільки добре він буде працювати в реальному світі. Тому ми оцінюємо ці показники ефективності за допомогою тестового набору даних, який відрізняється від даних, що використовуються для навчання системи машинного навчання. Вибір показника ефективності може здатися простим і об'єктивним, але часто буває важко вибрати показник ефективності, який добре відповідає бажаній поведінці системи.

У деяких випадках це пов'язано з тим, що важко вирішити, що саме слід вимірювати. Наприклад, при виконанні завдання транскрибування, чи слід вимірювати точність системи при транскрибуванні цілих послідовностей, або ж використовувати більш дрібнозернисту міру продуктивності, яка дає часткову оцінку за правильність деяких елементів послідовності? При

виконанні завдання регресії, чи слід більше карати систему, якщо вона часто робить помилки середнього розміру, чи якщо вона рідко робить дуже великі помилки?

Такі рішення залежать від конкретного застосування. В інших випадках ми знаємо, яку величину в ідеалі хотіли б виміряти, але виміряти її непрактично. Наприклад, це часто виникає в контексті оцінки щільності.

Багато з найкращих імовірнісних моделей представляють розподіли ймовірностей лише неявно. Обчислення фактичного значення ймовірності, приписаного конкретній точці простору, в багатьох таких моделях є нерозв'язною задачею. У цих випадках необхідно розробити альтернативний критерій, який все ще відповідає цілям проектування, або розробити хороше наближення до бажаного критерію [5].

1.4. Узагальнення алгоритмів машинного навчання

Основна проблема машинного навчання полягає в тому, що ми повинні добре працювати з новими, раніше не баченими вхідними даними, а не тільки з тими, на яких наша модель була навчена.

Здатність добре працювати на раніше неспостережуваних вхідних даних називається узагальненням.

Зазвичай при навчанні моделі машинного навчання ми маємо доступ до навчальної вибірки, можемо обчислити деяку міру помилки на навчальній вибірці, яка називається помилкою навчання, і ми зменшуємо цю помилку навчання. Досі те, що ми описали, було просто проблемою оптимізації.

Відмінність машинного навчання від оптимізації полягає в тому, що ми хочемо, щоб помилка узагальнення, яку також називають помилкою тестування, також була низькою.

Помилка узагальнення визначається як очікуване значення помилки на нових вхідних даних. Очікуване значення береться для різних можливих

вхідних даних, виходячи з розподілу вхідних даних, з якими, як ми очікуємо, система може зіткнутися на практиці.

Зазвичай ми оцінюємо помилку узагальнення моделі машинного навчання, вимірюючи її продуктивність на тестовому наборі прикладів, які були зібрані окремо від навчального набору.

У прикладі лінійної регресії навчають модель, мінімізуючи помилку навчання, що можна описати за допомогою формули:

$$\frac{1}{m^{(\text{train})}}||\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}||_2^2$$

Як ми можемо вплинути на продуктивність на тестовому наборі, коли ми можемо спостерігати лише за навчальним набором? Теорія статистичного навчання дає деякі відповіді.

Якщо навчальна та тестова вибірки зібрані довільно, ми дійсно мало що можемо зробити. Якщо ми можемо зробити деякі припущення про те, як збираються навчальна і тестова вибірки, то ми можемо досягти певного прогресу. Навчальні та тестові дані генеруються розподілом ймовірностей між наборами даних, що називається процесом генерації даних. Зазвичай ми робимо набір припущень, відомих під загальною назвою припущення i.i.d.

Ці припущення полягають у тому, що приклади в кожному наборі даних є незалежними один від одного, і що навчальна та тестова вибірки є однаково розподіленими, отриманими з одного й того ж розподілу ймовірностей. Це припущення дозволяє нам описати процес генерації даних за допомогою розподілу ймовірностей для одного прикладу. Той самий розподіл потім використовується для генерації кожного навчального прикладу і кожного тестового прикладу.

Ми називаємо цей спільний базовий розподіл розподілом, що генерує дані, і позначаємо його pdata. Ця ймовірнісна структура та припущення про неперервність дозволяють нам математично дослідити зв'язок між помилкою навчання та помилкою тесту. Один безпосередній зв'язок між помилкою навчання і помилкою тестування полягає в тому, що очікувана помилка

навчання випадково обраної моделі дорівнює очікуваній помилці тестування цієї моделі. Припустимо, що ми маємо розподіл ймовірностей p(x, y) і робимо з нього вибірки, щоб сформувати навчальну та тестову вибірки.

Для деякого фіксованого значення w очікувана помилка навчальної вибірки точно така ж, як і очікувана помилка тестової вибірки, тому що обидва очікування формуються за допомогою одного і того ж процесу вибірки набору даних. Єдина різниця між цими двома умовами полягає в назві, яку ми присвоюємо набору даних, з якого робимо вибірку. Звичайно, коли ми використовуємо алгоритм машинного навчання, ми не фіксуємо параметри заздалегідь, а вибираємо обидва набори даних. Ми виділяємо навчальний набір, потім використовуємо його для вибору параметрів, щоб зменшити похибку навчального набору, а потім виділяємо тестовий набір.

При такому процесі очікувана помилка тесту більша або дорівнює очікуваному значенню помилки навчання.

Фактори, що визначають, наскільки добре буде працювати алгоритм машинного навчання і його можливості включають в себе:

1. Зробити помилку навчання малою.
2. Зробити розрив між помилкою навчання і помилкою тесту невеликим.

Ці два фактори відповідають двом основним проблемам машинного навчання: недостатній та надмірній пристосованості. Недостатня пристосованість виникає, коли модель не в змозі отримати достатньо низьке значення помилки на навчальній вибірці. Перенавчання відбувається, коли розрив між помилкою навчання і помилкою тесту занадто великий [6].

Ми можемо контролювати, чи є модель більш схильною до надмірного або недостатнього використання, змінюючи її місткість. Неформально, пропускна здатність моделі – це її здатність виконувати широкий спектр функцій. Моделі з низькою здатністю можуть не впоратися з навчальною вибіркою. Моделі з високою здатністю можуть надмірно пристосовуватися,

запам'ятовуючи властивості навчальної вибірки, які не приносять їм користі на тестовій вибірці.

Одним із способів контролювати здатність алгоритму навчання є вибір його простір гіпотез, тобто набір функцій, які алгоритм навчання може алгоритму навчання дозволено обирати як рішення. Наприклад, алгоритм лінійної регресії має множину всіх лінійних функцій вхідних даних як простір гіпотез. Ми можемо узагальнити лінійну регресію, включивши в простір гіпотез не лише лінійні функції, але й поліноми. до простору гіпотез. Це збільшує пропускну здатність моделі.

Поліном першого степеня дає нам модель лінійної регресії, з якою ми вже знайомі, з прогнозуванням у вигляді виразу:

$$\hat{y} = b + wx.$$

Ввівши x^2 як ще одну характеристику в лінійну регресійну модель, ми можемо отримати модель, яка є квадратичною функцією від x:

$$\hat{y} = b + w_1 x + w_2 x^2$$

Хоча ця модель реалізує квадратичну функцію свого входу, вихід є все ще є лінійною функцією параметрів, тому ми можемо використовувати звичайні рівняння для навчання моделі у закритій формі.

Ми можемо продовжувати додавати нові степені x як додаткові функції, наприклад, щоб отримати поліном 9-го степеня:

$$\hat{y} = b + \sum_{i=1}^{9} w_i x^i.$$

Алгоритми машинного навчання зазвичай працюють найкраще, коли їхня потужність відповідає реальній складності завдання, яке вони повинні виконати, і кількості навчальних даних, які їм надаються.

Моделі з недостатньою потужністю не здатні вирішувати складні завдання.

Моделі з високою потужністю можуть вирішувати складні завдання, але коли їхня потужність вища, ніж потрібно для вирішення поточного завдання, вони можуть перевантажитися.

Рисунок 1.1 показує цей принцип у дії. Ми порівнюємо лінійний, квадратичний і предиктор 9-го степеня, які намагаються вирішити задачу, де справжня функція, що лежить в основі, є квадратичною. Лінійна функція не здатна врахувати кривизну істинної базової задачі, тому вона не підходить.

Предиктор 9-го степеня здатен представити правильну функцію, але він також здатен представити нескінченно багато інших функцій, які точно проходять через навчальні точки, оскільки ми маємо більше параметрів, ніж навчальних прикладів. У нас мало шансів вибрати рішення, яке добре узагальнює, коли існує так багато дико різних рішень.

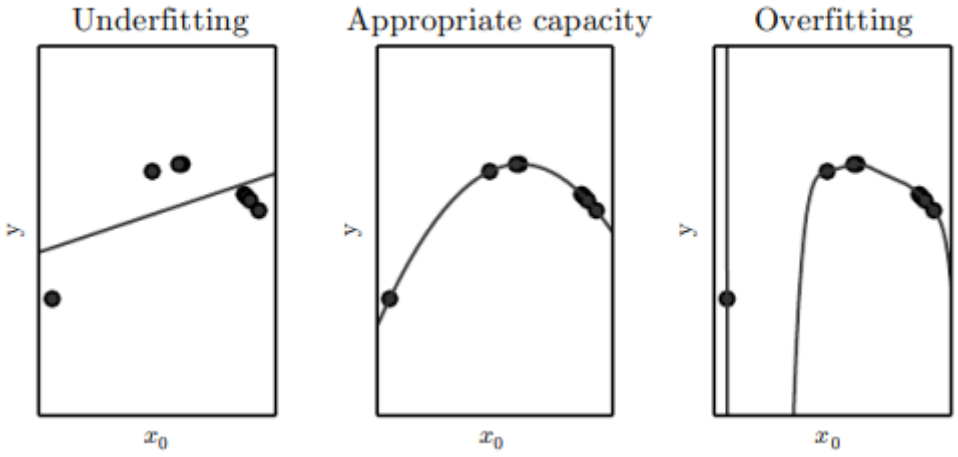

Рисунок 1.1 – Порівняння предикторів для вирішення задачі

У цьому прикладі квадратична модель ідеально відповідає справжній структурі задачі, тому вона добре узагальнює нові дані.

Ліворуч лінійна функція, підібрана до даних, страждає від недостатнього припасування – вона не може вловити кривизну, яка присутня в даних. По центру квадратична функція, підібрана до даних, добре узагальнює невидимі точки. Вона не страждає від значної кількості надмірної або недостатньої підгонки. Праворуч поліном 9-го степеня, підібраний до даних, страждає від

надмірної підгонки. Тут ми використали псевдообернену функцію Мура-Пенроуза для розв'язання недостатньо визначених нормальних рівнянь.

Розв'язок точно проходить через усі навчальні точки, але нам не пощастило виділити правильну структуру. Тепер він має глибоку долину між двома навчальними точками, яка не з'являється в істинній базовій функції. Вона також різко зростає зліва від даних, тоді як істинна функція в цій області спадає.

Вище ми розглянули лише один спосіб змінити потужність моделі: змінити кількість вхідних ознак, які вона має, і одночасно додати нові параметри, пов'язані з цими ознаками.

Насправді існує багато способів змінити потужність моделі. Пропускна здатність не визначається лише вибором моделі. Модель визначає, з якого сімейства функцій алгоритм навчання може вибирати при зміні параметрів, щоб зменшити мету навчання. Це називається репрезентативністю моделі.

У багатьох випадках пошук найкращої функції в межах цього сімейства є дуже складною оптимізаційною задачею. На практиці алгоритм навчання насправді не знаходить найкращу функцію, а лише ту, яка значно зменшує помилку навчання.

Ці додаткові обмеження, такі як недосконаліть алгоритму оптимізації, означають, що ефективна здатність алгоритму навчання може бути меншою, ніж репрезентативна здатність модельного сімейства. Наші сучасні уявлення про покращення узагальнення моделей машинного навчання – це вдосконалення думок філософів, що сягають корінням принаймні ще Птолемея. Багато ранніх вчених посилалися на принцип ощадливості, який зараз найбільш відомий як «бритва Оккама». Цей принцип стверджує, що серед конкуруючих гіпотез, які однаково добре пояснюють відомі спостереження, слід вибирати «найпростішу». Ця ідея була формалізована та уточнена у 20-му столітті засновниками теорії статистичного навчання.

Статистична теорія навчання пропонує різні способи кількісної оцінки здатності моделі. Серед них найбільш відомим є розмірність Вапніка-

Червоненкіса (VC). Розмірність VC вимірює потужність бінарного класифікатора. Розмірність VC визначається як найбільше можливе значення m, для якого існує навчальна вибірка з m різних точок x, які класифікатор може довільно позначити. Кількісне визначення ємності моделі дозволяє теорії статистичного навчання робити кількісні прогнози.

Найважливіші результати в теорії статистичного навчання показують, що розбіжність між помилкою навчання і помилкою узагальнення обмежена зверху величиною, яка зростає зі збільшенням ємності моделі, але зменшується зі збільшенням кількості навчальних прикладів. Ці межі забезпечують інтелектуальне обґрунтування того, що алгоритми машинного навчання можуть працювати, але вони рідко використовуються на практиці при роботі з алгоритмами глибокого навчання. Частково це пов'язано з тим, що межі часто є досить розмитими, а частково з тим, що визначити потужність алгоритмів глибокого навчання може бути досить складно.

Проблема визначення потужності моделі глибокого навчання є особливо складною, оскільки ефективна потужність обмежена можливостями алгоритму оптимізації, а у нас мало теоретичного розуміння дуже загальних проблем неопуклої оптимізації, пов'язаних з глибоким навчанням. Ми повинні пам'ятати, що хоча простіші функції більш схильні до узагальнення (мають невеликий розрив між помилкою навчання і помилкою тесту), ми все одно повинні вибирати досить складну гіпотезу, щоб досягти низької помилки навчання. Зазвичай помилка навчання зменшується до асимптотики до мінімально можливого значення помилки зі збільшенням потужності моделі (припускаючи, що міра помилки має мінімальне значення).

Як правило, помилка узагальнення алгоритмів машинного навчання має U-подібну криву як функцію потужності моделі. Це проілюстровано за допомогою Рис 1.2.

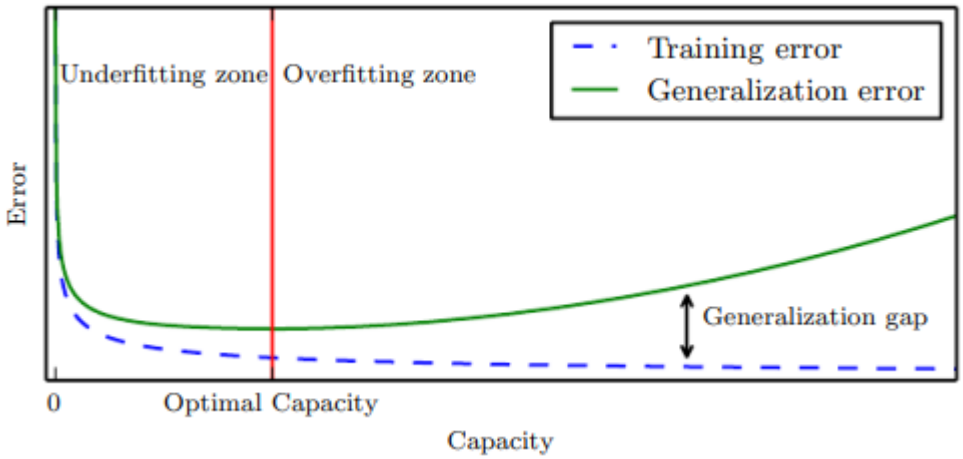

Рисунок 1.2 – Помилка узагальнення алгоритмів машинного навчання

Наведений вище рисунок демонструє типовий взаємозв'язок між пропускною здатністю та помилкою. Помилка навчання і помилка тестування поводяться по-різному. На лівому кінці графіка помилка навчання і помилка узагальнення обидві високі. Це режим недостатнього навчання. Коли ми збільшуємо потужність, помилка навчання зменшується, але розрив між помилкою навчання і помилкою узагальнення збільшується. Зрештою розмір цього розриву переважує зменшення помилки навчання, і ми входимо в режим перенавчання, де ємність є занадто великою, вищою за оптимальну.

Іноді непараметричні моделі є лише теоретичними абстракціями (наприклад, алгоритм, який перебирає всі можливі розподіли ймовірностей), які неможливо реалізувати на практиці. Однак ми також можемо створювати практичні непараметричні моделі, роблячи їхню складність функцією розміру навчальної вибірки.

Одним із прикладів такого алгоритму є регресія найближчих сусідів. На відміну від лінійної регресії, яка має вектор ваг фіксованої довжини, модель регресії найближчого сусіда просто зберігає X та y з навчальної вибірки. Коли потрібно класифікувати тестову точку x, модель шукає найближчий запис у навчальній множині і повертає пов'язану з ним цільову функцію регресії.

Алгоритм також можна узагальнити на метрики відстані, відмінні від норми L2, такі як вивчені метрики відстані. Якщо алгоритму дозволено

розривати зв'язки шляхом усереднення значень уі для всіх Хі,: які є найближчими, то цей алгоритм здатен досягти мінімально можливої помилки навчання (яка може бути більшою за нуль, якщо два однакові входи пов'язані з різними виходами) на будь-якому наборі регресійних даних.

Також можна створити непараметричний алгоритм навчання, обернувши параметричний алгоритм навчання всередині іншого алгоритму, який збільшує кількість параметрів за потреби. Наприклад, ми можемо уявити собі зовнішній цикл навчання, який змінює ступінь полінома, отриманого за допомогою лінійної регресії, поверх поліноміального розкладання вхідних даних.

Ідеальна модель – це оракул, який просто знає справжній розподіл ймовірностей, що генерує дані. Навіть така модель все одно матиме певну похибку на багатьох задачах, тому що в розподілі все ще може бути певний шум. У випадку керованого навчання відображення від x до y може бути за своєю природою стохастичним, або y може бути детермінованою функцією, яка включає інші змінні, окрім тих, що входять до x. Помилка, якої зазнає оракул, роблячи прогнози на основі істинного розподілу p(x, y), називається помилкою Байєса.

Похибка навчання і похибка узагальнення змінюються залежно від розміру навчальної вибірки. Очікувана помилка узагальнення ніколи не може зростати зі збільшенням кількості навчальних прикладів. Для непараметричних моделей більша кількість даних дає краще узагальнення, поки не буде досягнуто найкращої можливої помилки. Будь-яка фіксована параметрична модель з меншою, ніж оптимальна, потужністю буде асимптотично наближатися до значення помилки, що перевищує помилку Байєса, що показано за допомогою ілюстрації, наведеної на Рис 1.3.

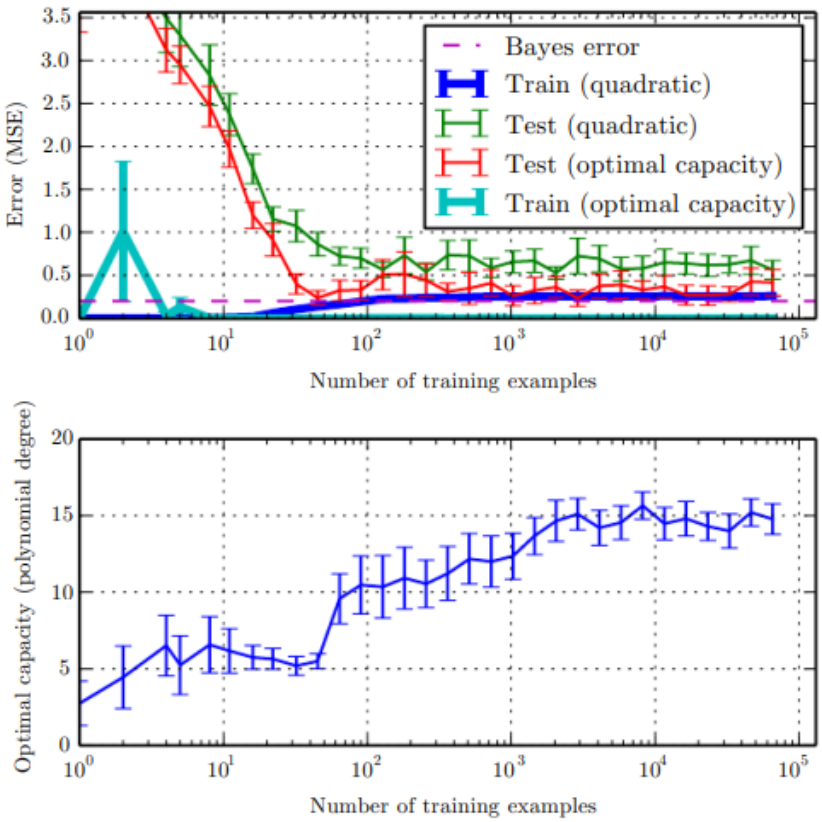

Рисунок 1.3 – Демонстрація помилки Байєса у вигляді графіків

Зауважимо, що модель може мати оптимальну потужність, але все одно мати великий розрив між помилкою навчання та помилкою узагальнення. У такій ситуації ми можемо зменшити цей розрив, зібравши більше навчальних прикладів [7].

1.5. Гіперпараметри та набори перевірки

Більшість алгоритмів машинного навчання мають кілька налаштувань, за допомогою яких ми можемо керувати поведінкою алгоритму навчання. Ці параметри називаються гіперпараметрами. Значення гіперпараметрів не адаптуються самим алгоритмом навчання (хоча ми можемо розробити процедуру вкладеного навчання, де один алгоритм навчання вивчає найкращі гіперпараметри для іншого алгоритму навчання).

У прикладі поліноміальної регресії, який був розглянутий вище за допомогою графіків, є один гіперпараметр: степінь полінома, який діє як гіперпараметр потужності.

Значення λ, яке використовується для керування силою спаду ваги, є ще одним прикладом гіперпараметру. Іноді в якості гіперпараметру обирається параметр, який алгоритм навчання не вивчає, оскільки його важко оптимізувати. Частіше налаштування має бути гіперпараметром, тому що вивчення цього гіперпараметру на навчальній вибірці не є доцільним. Це стосується всіх гіперпараметрів, які керують пропускною здатністю моделі.

Якщо такі гіперпараметри вивчати на навчальній вибірці, вони завжди обиратимуть максимально можливу пропускну здатність моделі, що призведе до надмірної підгонки, що було продемонстровано за допомогою рисунка вище.

Наприклад, ми завжди можемо краще підігнати навчальну множину за допомогою полінома вищого степеня і параметра спадання ваги $\lambda = 0$, ніж за допомогою полінома нижчого степеня і позитивного параметра спадання ваги.

Щоб вирішити цю проблему, нам потрібен валідаційний набір прикладів, яких алгоритм навчання не помічає. Раніше ми обговорювали, як можна використовувати утриманий тестовий набір, що складається з прикладів, які походять з того ж розподілу, що і навчальний набір, для оцінки помилки узагальнення учня після завершення процесу навчання. Важливо, що тестові приклади жодним чином не використовуються для вибору моделі, в тому числі її гіперпараметрів. З цієї причини жоден приклад з тестового набору не може бути використаний у валідаційному наборі. Тому ми завжди будуємо валідаційну вибірку з навчальних даних.

Зокрема, ми розбиваємо навчальні дані на дві непересічні підмножини. Одна з цих підмножин використовується для навчання параметрів. Інша підмножина – це наша валідаційна множина, яка використовується для оцінки помилки узагальнення під час або після навчання, що дозволяє відповідним чином оновлювати гіперпараметри. Підмножину даних, що використовується

для навчання параметрів, все ще зазвичай називають навчальною вибіркою, хоча її можна сплутати з більшим масивом даних, що використовується для всього процесу навчання.

Підмножина даних, яка використовується для вибору гіперпараметрів, називається валідаційною множиною. Зазвичай для навчання використовується близько 80% навчальних даних, а для перевірки – 20%.

Оскільки валідаційний набір використовується для «тренування» гіперпараметрів, помилка валідаційного набору буде занижувати помилку узагальнення, хоча, як правило, на меншу величину, ніж помилка навчання. Після завершення оптимізації всіх гіперпараметрів похибку узагальнення можна оцінити за допомогою тестової вибірки. На практиці, коли той самий тестовий набір неодноразово використовувався для оцінки продуктивності різних алгоритмів протягом багатьох років, і особливо якщо врахувати всі спроби наукової спільноти перевершити заявлену найсучаснішу продуктивність на цьому тестовому наборі, ми в кінцевому підсумку отримаємо оптимістичні оцінки і для цього тестового набору.

Таким чином, бенчмарки (benchmarks) можуть стати застарілими і не відображати справжню продуктивність навченої системи в реальних умовах. На щастя, спільнота має тенденцію переходити до нових (і, як правило, більш амбітних і великих) наборів тестових даних [8].

1.6. Основні поняття фондового ринку та його прогнозування

Фондовий ринок – це місце, де відбуваються регулярні операції з купівлі-продажу акцій публічних акціонерних товариств. Місце, де відбувається така фінансова діяльність, називається називається фондовою біржею.

Терміни «фондовий ринок» і «фондова біржа» використовуються взаємозамінні. В Індії фондовий ринок контролюється регуляторним органом, відомим як Рада з цінних паперів та бірж Індії (SEBI). Деякі з провідних бірж

у всьому світі Нью-Йоркська фондова біржа, Американська фондова біржа NASDAQ, Лондонська фондова біржа у Великій Британії тощо. Як первинний ринку компанія може випустити цінні папери для широкої громадськості у формі первинної публічної пропозиції (IPO).

Таким чином компанія може залучити більше коштів від громадськості для своєї майбутньої діяльності, таких як розширення, зменшення боргу тощо. Після того, як акції розподілені серед громадськості (тобто інвесторам або акціонерам), компанія реєструє ці цінні папери на фондовій біржі.

Після того, як акції котируються на фондовій біржі, інвестори можуть купувати або продавати акції на біржі. біржі, інвестори можуть купувати або продавати акції. Тут фондовий ринок виступає як вторинний ринок, де акціями можна торгувати між інвесторами за визначеною ціною. Біржі діють як кліринговий центр для кожної транзакції усуваючи ризик невиконання угоди. Компанія, акції якої котируються на фондовій біржі, також може пропонувати нові або додаткові акції шляхом подальшого розміщення на більш пізній стадії, що називається офертою на продаж або через емісію прав. Вони можуть навіть викупити акції у акціонерів або припинити з торгів.

Компанії зазвичай оголошують дивіденди акціонерам, виходячи з прибутку, отриманого в якості винагороди. винагороду. Купівля та продаж акцій передбачає приріст капіталу, якщо інвестор продає акцію за вищою ціною, ніж ціна купівлі. Аналогічно, існує ймовірність втрати капіталу, якщо ціна акції ще більше впаде, і інвестор вирішить продати її, щоб уникнути подальших втрат. Всі акціонери компанії мають право голосу при обранні членів правління, які будуть відповідають за нагляд за основними рішеннями, прийнятими керівництвом компанії.

Існує велика кількість гіпотез щодо фондових ринків. Деякі з них:

- гіпотеза ефективного ринку: є важливою теорією середини 1960-х років. Це тип гіпотези, згідно з якою ціна акцій відображає всю доступну інформацію на даний момент. Це означає, що ціна акцій є точно оціненою до тих пір, поки не відбудуться будь-які подальші зміни в оцінці. Багато

інвесторів, таких як Уоррен Баффет, випередили ринок, передбачаючи ціни в межах загального ринку;

- принцип п'ятдесяти відсотків. Цей принцип стверджує, що будь-яка акція, яка перебуває у висхідному тренді, матиме корекцію ціни, перш ніж продовжить свій висхідний тренд. Наприклад, якщо акція зросла приблизно на 30 відсотків за кілька днів, то перед тим, як вона знову почне зростати, відбудеться тимчасова корекція приблизно на 15 відсотків. Якщо акція зросла на значну суму, то деякі трейдери захочуть перетворити паперові прибутки на реальні, що спричинить тимчасову корекцію. Зниження ціни акцій викличе паніку серед інших інвесторів, що призведе до подальшого падіння ціни. Низька ціна привабить нових інвесторів, які почнуть вкладати гроші в ці акції, і ціна продовжить зростати далі;
- теорія більшого дурня. У цьому типі стратегії інвестор купує певні акції, думаючи, що на більш пізніх етапах він зможе продати їх іншій особі за вищою ціною. Це означає, що інвестор може заробити гроші, якщо знайдеться хтось інший, хто купить за вищою ціною. Тут інвестиції здійснюються не тому, що ціна акцій є справедливою або цінною, а тому, що інвестор вірить, що зможе продати їх комусь іншому за вищою ціною;
- теорія непарних партій. Цей тип теорії базується на продажі непарних партій акцій, якими володіють окремі інвестори, і які будуть використовуватися як індикатор для купівлі цих акцій. Ця теорія базується на припущенні, що дрібні інвестори зазвичай помиляються;
- теорія перспектив. Цю теорію також називають теорією уникнення втрат. Якщо люди мають вибір між двома різними стратегіями, то вони оберуть ту стратегію, яка має менше шансів закінчитись втратою. Якщо людині надається вибір, який є ризикованим, але може призвести до виграшу, то вона обиратиме меншу очікувану корисність, але з високою ймовірністю. Якщо людина має вибір, який є ризикованим і може призвести до втрат, то вона обиратиме нижчу очікувану корисність, оскільки вона може уникнути втрат;

- теорія раціональних очікувань. Цей тип теорії стверджує, що людина буде інвестувати або витрачати відповідно до того, що, на її думку, станеться в майбутньому. Наприклад, трейдер або інвестор вважатиме, що ціна акцій зростатиме. Вважаючи так, він купує ці акції, що в кінцевому підсумку призведе до зростання ціни на ці акції. В інших випадках інвестори вважають, що акції недооцінені. Інші інвестори-однодумці також вважають, що акції недооцінені, що призводить до зростання ціни на них;
- теорія коротких відсотків. Ця теорія припускає, що високий рівень короткого інтересу до певної акції може спричинити зростання ціни на неї. Оскільки велика кількість трейдерів починає відкривати короткі позиції по певній акції. Ці продавці коротких позицій повинні покрити свої позиції до кінця торгового дня, оскільки їм потрібно придбати акції, щоб повернути їх покупцям за рахунок підвищення ціни акцій у найближчому майбутньому. Наприклад, багато трейдерів починають короткий продаж акцій на початку, думаючи, що вони придбають акції за найнижчою можливою ціною.

Оскільки ці трейдери поспішають придбати акції за найнижчою можливою ціною, виникає раптове збільшення попиту, що призводить до зростання ціни на акції.

Прогнозування цін на акції є дуже складним завданням. Багато трейдерів та інвесторів на фондовому ринку завжди шукають методику, яка б гарантувала прибуток шляхом прогнозування руху цін на акції та мінімізувала ризик інвестування. Для прогнозування ціни акцій використовується багато методів, таких як фундаментальний аналіз, аналіз часових рядів, технічний аналіз тощо;

- фундаментальний аналіз: метод вимірювання внутрішньої вартості компанії шляхом перевірки економічних та фінансових факторів. Це дослідження інфраструктури компанії, історії компанії, продуктового портфеля, фінансових результатів компанії, наявності у компанії боргів, конкурентів та перспектив компанії;

- аналіз часових рядів. Традиційно часові ряди використовуються для прогнозування майбутніх значень ціни акцій на основі попередніх історичних значень;
- статистичний аналіз. Статистичний аналіз намагається передбачити майбутній рух цін, надаючи трейдерам або інвесторам інформацію, необхідну для отримання прибутку. Технічний аналіз використовує графіки, які містять такі дані, як 50/100/200 денні ковзаючі середні. Високі/низькі ціни, обсяги торгів тощо. Ця інформація може бути використана для прогнозування руху ціни акцій. Основне припущення технічного аналізу полягає в тому, що ринок обробив всю доступну інформацію, і вона відображена на ціновому графіку;
- розпізнавання моделей: техніка в основному зосереджена на виявленні закономірностей або тенденцій у даних. Прогнозування здійснюється за допомогою графіків, побудованих з плином часу. Закономірності допомагають інвесторам визначити майбутню еволюцію акцій;
- машинне навчання. Завдання машинного навчання класифікуються як контрольоване і неконтрольоване навчання. Метою машинного навчання є навчання алгоритму для автоматичного зіставлення вхідних даних і прогнозування вихідних даних. Існує кілька алгоритмів машинного навчання для прогнозування даних фондового ринку, таких як дерево рішень, випадковий ліс, SVM, LSTM тощо. Аналіз настроїв: У цьому процесі прогнозування фондового ринку може бути зроблено на основі новинних стрічок або твітів публічної компанії;
- гібридний. У цьому підході для прогнозування цін на акції об'єднуються різні підходи, наприклад, статистичний аналіз та розпізнавання образів, аналіз настроїв та машинне навчання тощо. Перевагою цього підходу є покращена продуктивність та підвищена точність прогнозування [9].

Схему класифікації методів для прогнозування цін активів фондових ринків показано за допомогою Рис. 1.4.

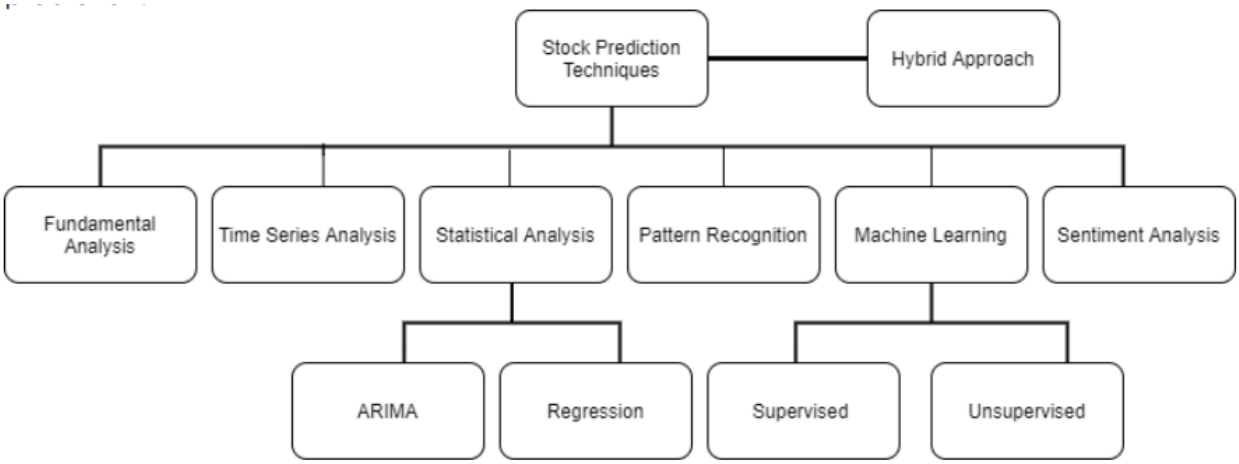

Рисунок 1.4 – Схема класифікації методів прогнозування

1.7. Методи машинного навчання для прогнозування

Прогнозування цін, особливо у фінансовій сфері, вже давно є складним завданням через притаманну ринкам складність і невизначеність. Традиційні методи, такі як статистичні моделі, часто борються з нелінійною та динамічною природою фінансових даних. Останніми роками глибоке й машинне навчання стали потужним інструментом для прогнозування цін, надаючи нові способи аналізу та прогнозування складних закономірностей у даних.

Серед найбільш відомих та ефективних методів машинного навчання для прогнозування часових рядів, окремим випадком яких є прогнозування ринкових трендів, є такі моделі:

1. LSTM – LSTM.
2. LSTM Bidirectional – Двонаправлена LSTM.
3. LSTM 2-Path – Двоканальна LSTM.
4. GRU – GRU.
5. GRU Bidirectional – Двонаправлена GRU.
6. GRU 2-Path – Двоканальна GRU.
7. Vanilla – Проста RNN.
8. Vanilla Bidirectional – Двонаправлена RNN.
9. Vanilla 2-Path – Двоканальна RNN.

10. LSTM Seq2seq – LSTM Seq2seq.
11. LSTM Bidirectional Seq2seq – Двонаправлена LSTM Seq2seq.
12. LSTM Seq2seq VAE – LSTM Seq2seq VAE.
13. GRU Seq2seq – GRU Seq2seq.
14. GRU Bidirectional Seq2seq – Двонаправлена GRU Seq2seq.
15. GRU Seq2seq VAE – GRU Seq2seq VAE.
16. Attention-is-all-you-Need – "Увага – це все, що вам потрібно".
17. CNN-Seq2seq – CNN-Seq2seq.
18. Dilated-CNN-Seq2seq – Розширена CNN-Seq2seq.

Розглянемо найбільш ефективні моделі, одну з яких згодом використаємо в програмному забезпеченні для прогнозування ринкових трендів.

- Довга короткочасна пам'ять (Long Short-Term Memory, LSTM) – це тип архітектури штучної рекурентної нейронної мережі (RNN), що використовується в галузі глибокого навчання. На відміну від стандартних нейронних мереж прямого поширення, LSTM мають зворотні зв'язки, що дозволяє їм використовувати часові залежності між послідовностями даних. LSTM розроблені для вирішення проблеми зникнення або вибуху градієнтів, які можуть виникати при навчанні традиційних ШНМ на послідовностях даних. Це робить їх добре придатними для задач, що включають послідовні дані, такі як обробка природної мови (NLP), розпізнавання мови та прогнозування часових рядів.

Архітектура моделі LSTM показана за допомогою схеми на Рис. 1.5.

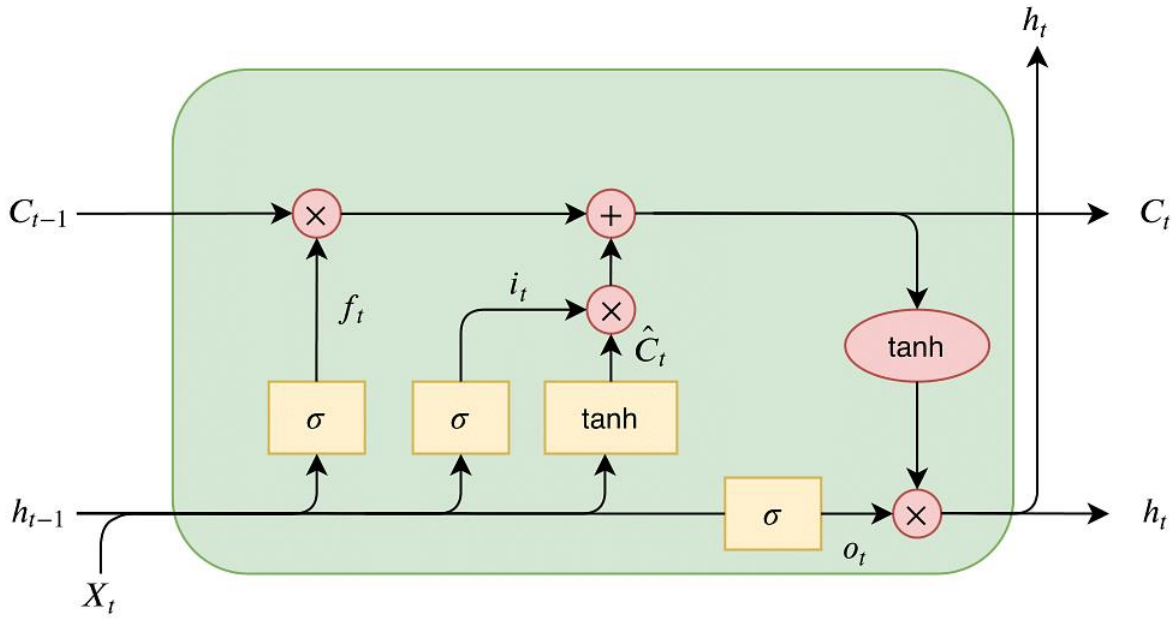

Рисунок 1.5 – Архітектура моделі LSTM

Мережі LSTM вводять комірки пам'яті, які мають здатність зберігати інформацію в довгих послідовностях. Кожна комірка пам'яті має три основні компоненти: вхідний вентиль, вентиль забування та вихідний вентиль. Ці вентилі допомагають регулювати потік інформації в комірку пам'яті та з неї.

Вхідний вентиль (продемонстровано на Рис. 1.6) визначає, яку частину нового входу слід зберігати в комірці пам'яті. Він приймає поточний вхід і попередній прихований стан як вхідні дані і видає значення від 0 до 1 для кожного елемента комірки пам'яті.

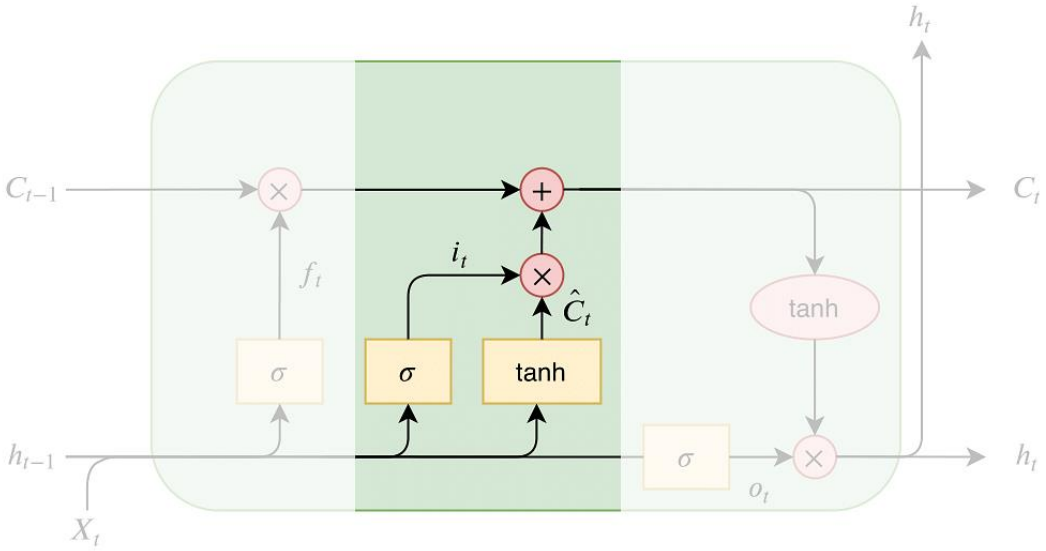

Рисунок 1.6 – Етап вхідного вентиля

Вентиль забування (продемонстровано на Рис. 1.7) вирішує, яку інформацію видалити з комірки пам'яті. Він приймає поточний вхід і попередній прихований стан як входи, і виводить значення від 0 до 1 для кожного елемента комірки пам'яті. Значення 0 означає, що інформація ігнорується, тоді як значення 1 означає, що вона зберігається.

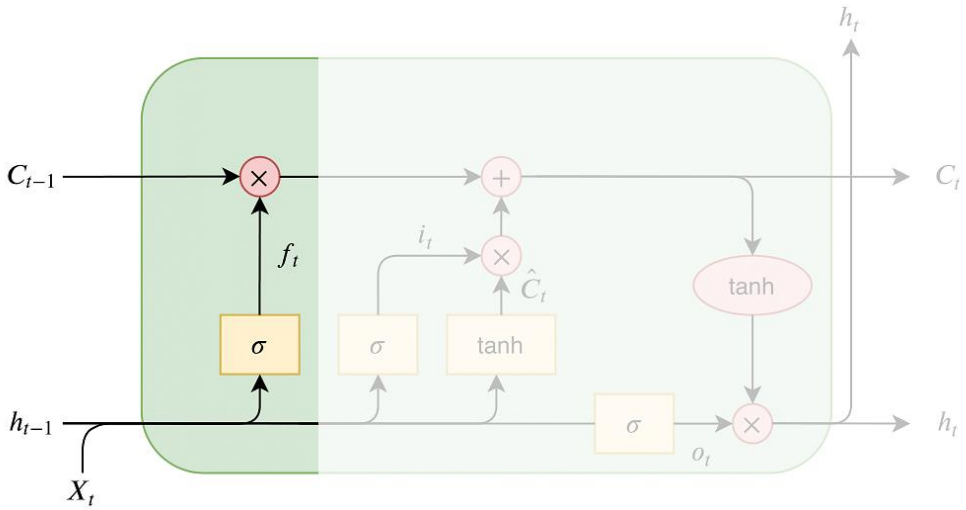

Рисунок 1.7– Етап вентиля забування

Вихідний вентиль контролює, яку частину вмісту комірки пам'яті слід використовувати для обчислення прихованого стану. Він приймає поточний вхід і попередній прихований стан як входи, і виводить значення від 0 до 1 для кожного елемента комірки пам'яті (див. Рис. 1.8).

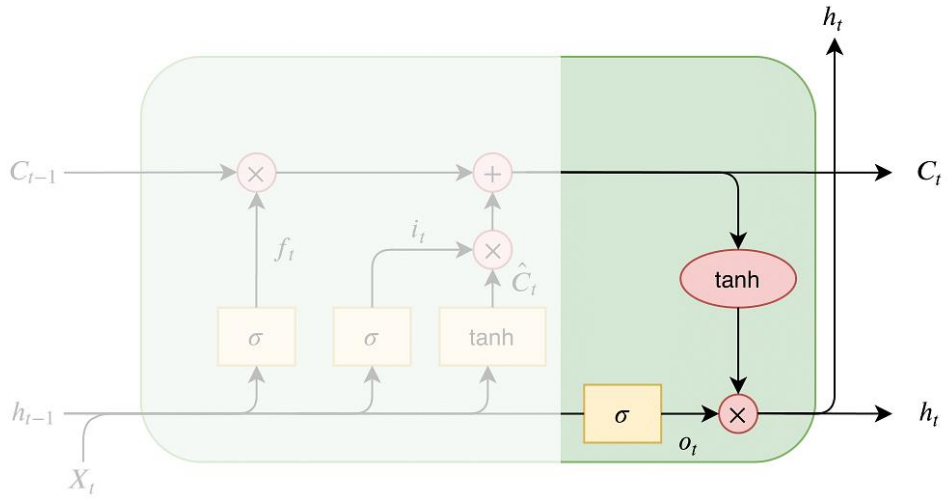

Рисунок 1.8 – Етап вихідного вентиля

Використовуючи ці вентилі, LSTM-мережі можуть вибірково зберігати, оновлювати та отримувати інформацію з довгих послідовностей. Це робить їх особливо ефективними для задач, що вимагають моделювання довготривалих залежностей, таких як розпізнавання мови, переклад та аналіз настроїв.

У наведеній вище архітектурі вихідний вентиль є останнім кроком у комірці LSTM, і це лише одна з частин повного процесу. Перш ніж LSTM-мережа зможе робити бажані прогнози, потрібно врахувати ще кілька моментів.

Наприклад, якщо ви намагаєтеся передбачити ціну акцій на наступний день на основі даних про ціни за попередні 30 днів, то кроки в комірці LSTM будуть повторюватися 30 разів. Це означає, що модель LSTM ітеративно створила б 30 прихованих станів для прогнозування ціни акцій на наступний день.

Потік інформації в LSTM відбувається повторюваним чином, утворюючи ланцюгоподібну структуру, як продемонстровано на Рис. 1.9.

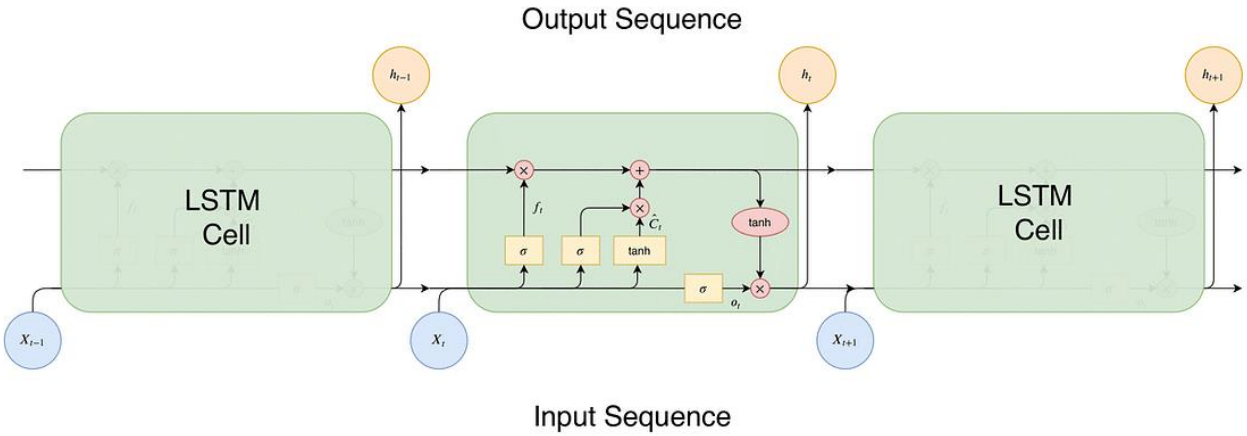

Рисунок 1.9 – Процес роботи моделі LSTM

Потік останнього виходу комірки до кінцевого стану додатково контролюється вихідним вентилем. Однак вихід комірки LSTM все ще залишається прихованим станом, і він не має прямого відношення до ціни акцій, яку ми намагаємося передбачити. Щоб перетворити прихований стан у бажаний вихід, застосовується лінійний шар як останній крок у процесі LSTM.

Цей крок лінійного шару відбувається тільки один раз, в самому кінці, і він не включений в діаграми комірки LSTM, тому що він виконується після повторних кроків комірки LSTM [10];

- GRU – Gated Recurrent Unit – Рекурентний блок із закритим доступом. Це тип архітектури рекурентної нейронної мережі (RNN), схожий на LSTM. Як і LSTM, GRU призначений для моделювання послідовних даних, дозволяючи інформації вибірково запам'ятовуватися або забуватися з часом. Однак GRU має простішу архітектуру, ніж LSTM, з меншою кількістю параметрів, що може зробити його простішим у навчанні та ефективнішим в обчислювальному плані.

Основна відмінність між GRU та LSTM полягає в тому, як вони обробляють стан комірки пам'яті. В LSTM стан комірки пам'яті зберігається окремо від прихованого стану і оновлюється за допомогою трьох вентилів: вхідного, вихідного і вентилю забуття. У GRU стан комірки пам'яті замінюється «вектором активації кандидата», який оновлюється за допомогою двох вентилів: вентилю скидання та вентилю оновлення.

Ворота скидання визначають, яку частину попереднього прихованого стану слід забути, а ворота оновлення визначають, яку частину вектора-кандидата на активацію включити в новий прихований стан.

Для вирішення проблеми зникаючого градієнта стандартного RNN, GRU використовує, так звані, update gate і reset gate. По суті, це два вектори, які вирішують, яку інформацію передавати на вихід. Особливістю цих векторів є те, що їх можна навчити зберігати інформацію з давніх часів, не розмиваючи її в часі, або видаляти інформацію, яка не має відношення до прогнозу.

Щоб пояснити математику, яка стоїть за цим процесом, ми розглянемо один блок з наступної рекурентної нейронної мережі, який продемонстрований на Рис. 1.10.

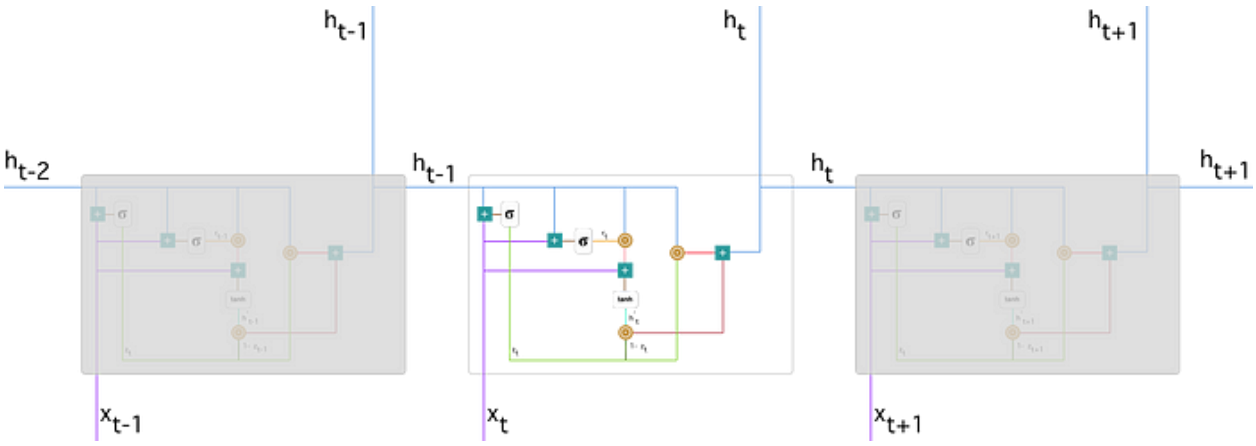

Рисунок 1.10 – Рекурентна нейронна мережа із закритим рекурентним блоком

Більш детальна версія цього єдиного блока показана на Рис. 1.11.

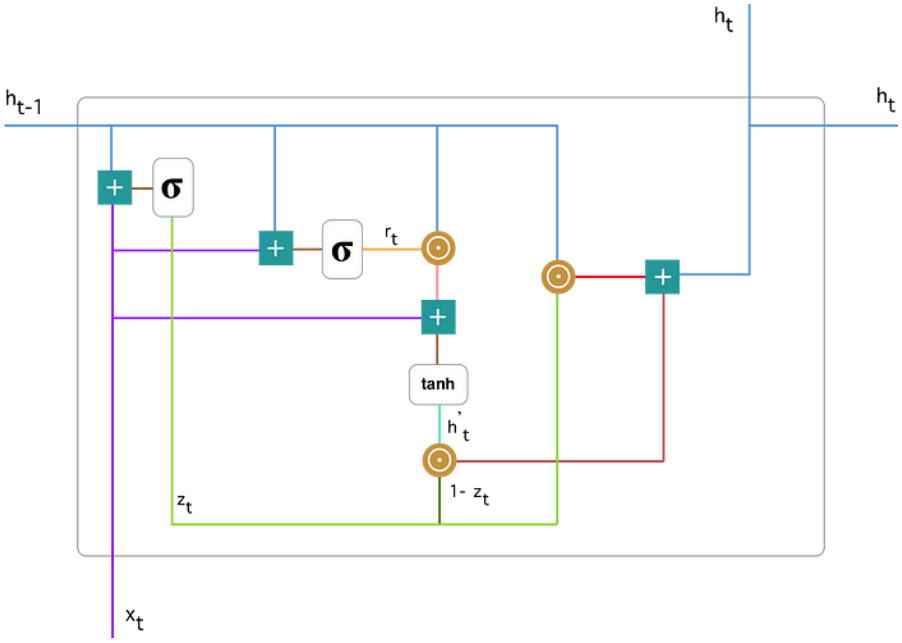

Рисунок 1.11 – Більш детальна версія єдиного блока GRU

Позначення операторів, використаних у моделі, показані за допомогою Рис. 1.12.

Рисунок 1.12 – Позначення операторів, використаних у моделі

1. Update gate

Ми починаємо з обчислення стробу оновлення z_t для кроку часу t за формулою:

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Коли x_t підключено до блоку мережі, він множиться на власну вагу W(z). Те саме відбувається з h_(t-1), який містить інформацію для попередніх t-1 блоків і множиться на власну вагу U(z). Обидва результати додаються разом і застосовується сигмоїдна функція активації, щоб розподілити результат між 0 і 1. За наведеною вище схемою, ми маємо стан, продемонстрований на Рис. 1.13.

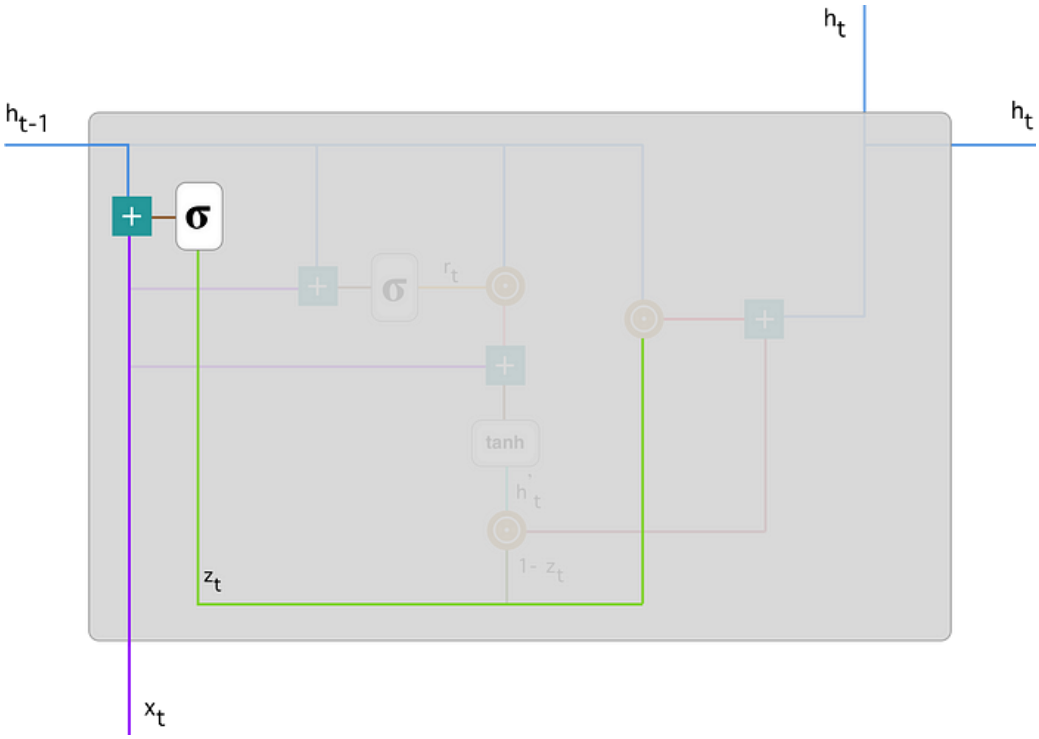

Рисунок 1.13 – Позначення стану Update gate

Ворота оновлення (Update gate) допомагають моделі визначити, скільки інформації з минулого (з попередніх часових кроків) потрібно передати в майбутнє. Це дуже важливо, тому що модель може вирішити скопіювати всю інформацію з минулого і усунути ризик зникнення проблеми градієнта. Ми розглянемо використання воріт оновлення пізніше. А поки що запам'ятайте формулу для z_t.

2. Reset gate

По суті, цей гейт використовується з моделі, щоб вирішити, яку частину минулої інформації забути. Для його розрахунку ми використовуємо:

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Ця формула така ж сама, як і для воріт оновлення. Різниця полягає у вагах та використанні воріт, які ми побачимо трохи згодом. На схемі нижче показано, де знаходиться ворота скидання (Рис. 1.14).

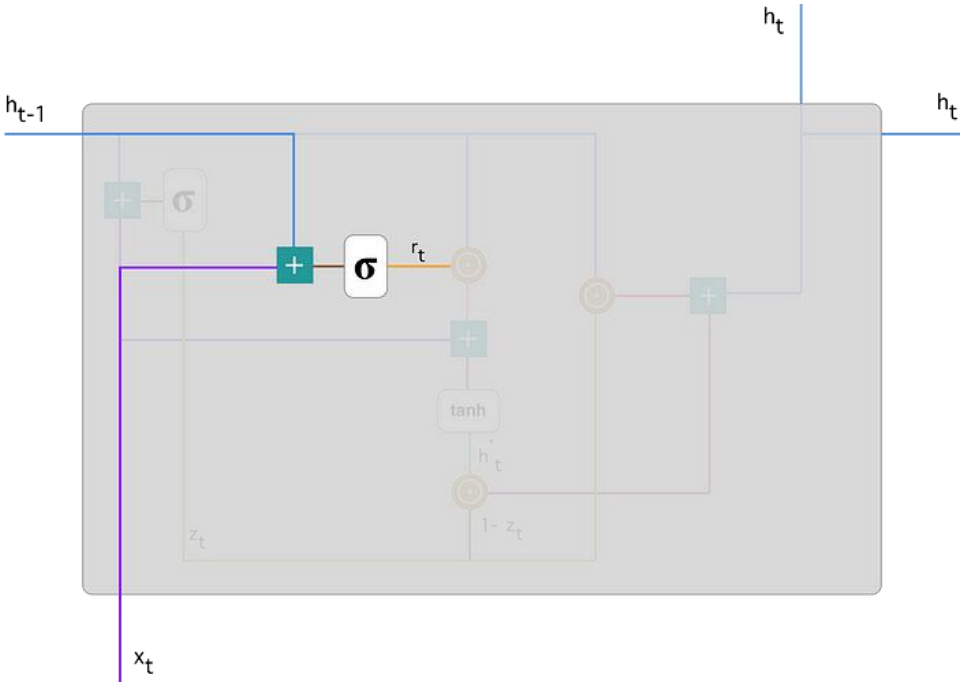

Рисунок 1.14 – Позначення стану Reset gate

Як і раніше, ми підключаємо h_(t-1) – синю лінію та x_t – фіолетову лінію, множимо їх на відповідні ваги, підсумовуємо результати та застосовуємо сигмоїдну функцію.

3. Current memory content

Вентилі впливають на кінцевий результат таким чином: спочатку ми почнемо з використання воріт скидання. Ми введемо новий вміст пам'яті, який буде використовувати вентиль скидання для зберігання відповідної інформації з минулого. Він обчислюється наступним чином:

$$h_t^{'} = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

Ми виконуємо поелементне множення h_(t-1) – синя лінія та r_t – помаранчева лінія, а потім підсумовуємо результат – рожева лінія з вхідними даними x_t – фіолетова лінія. Нарешті, tanh використовується для отримання h'_t – яскраво-зелена лінія. Це продемонстровано на Рис. 1.15.

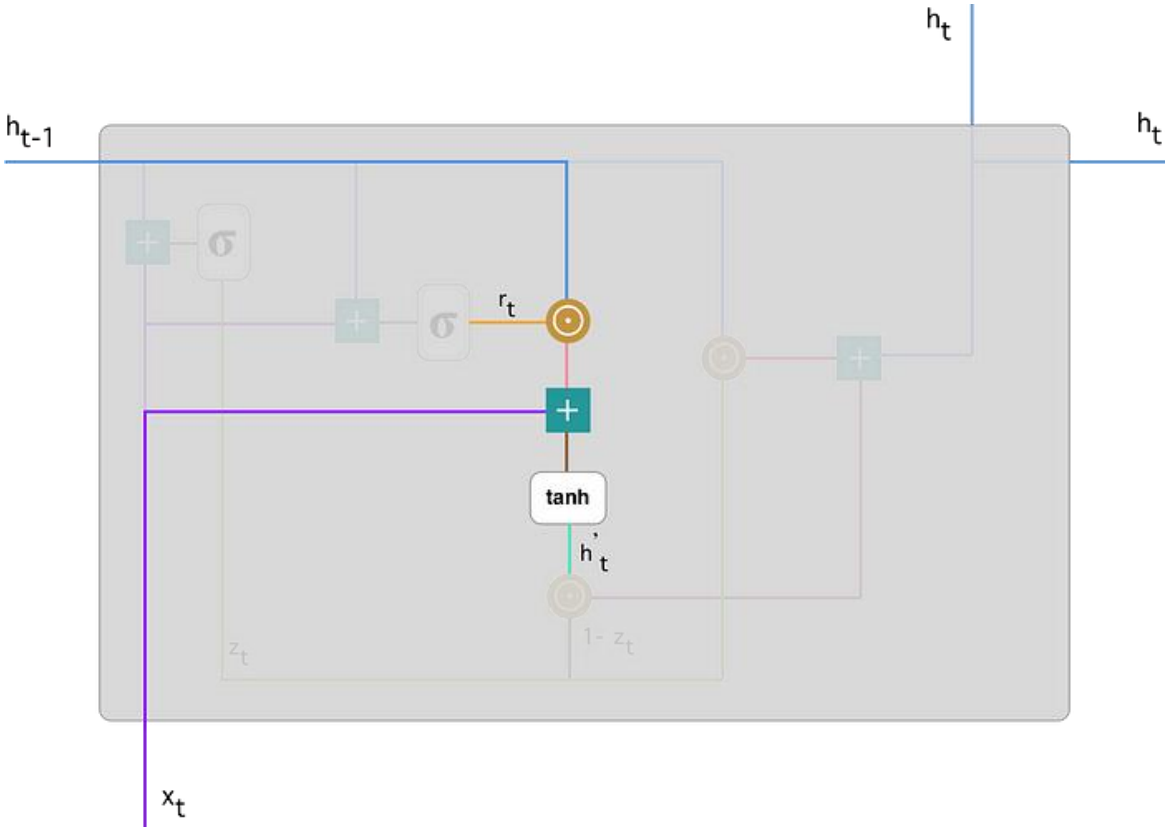

Рисунок 1.15 – Позначення стану Current memory content

4. Final memory at current time step

На останньому кроці мережа повинна обчислити h_t – вектор, який містить інформацію для поточного блоку і передати її мережі. Для цього потрібен вентиль оновлення. Він визначає, що взяти з поточного вмісту пам'яті – h'_t, а що з попередніх кроків – h_(t-1). Це робиться наступним чином:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'}$$

Модель може навчитися встановлювати вектор z_t близьким до 1 і зберігати більшу частину попередньої інформації. Оскільки z_t буде близьким до 1 на цьому кроці, 1-z_t буде близьким до 0, що призведе до ігнорування

значної частини поточного контенту (в даному випадку останньої частини рецензії, яка пояснює сюжет книги), яка не є релевантною для нашого прогнозу. Графічно цей етап показано на Рис. 1.16.

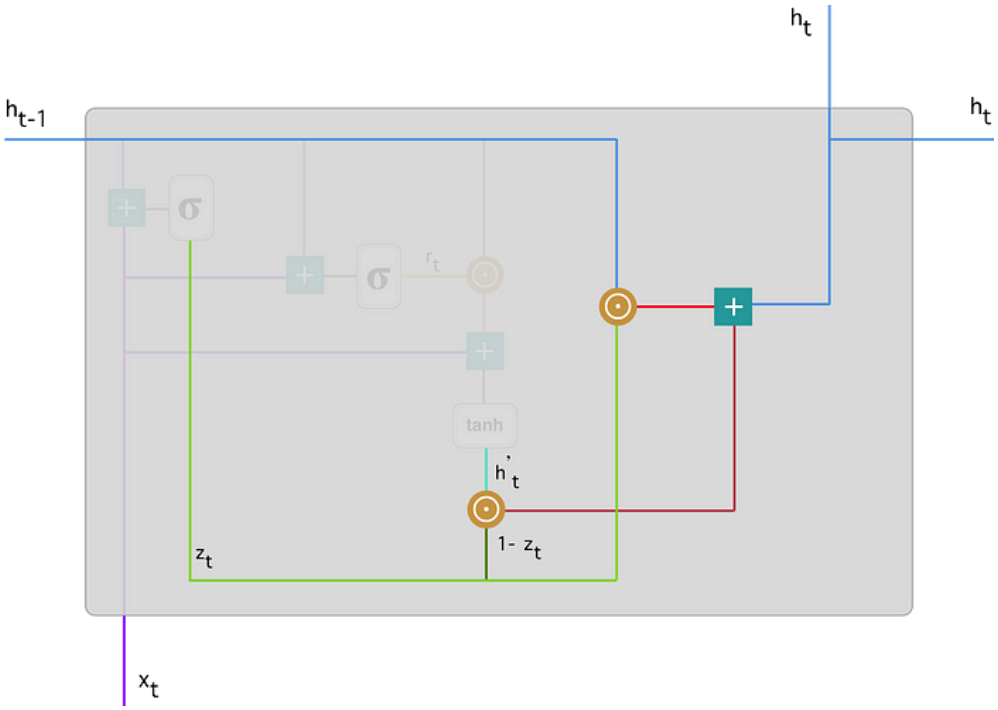

Рисунок 1.16 – Позначення стану Final memory at current time step

Можна побачити, як z_t – зелена лінія використовується для обчислення 1-z_t, що у поєднанні з h'_t – яскраво-зеленою лінією дає результат у темно-червоній лінії. z_t також використовується з h_(t-1) – синьою лінією для поелементного множення. Нарешті, h_t – синя лінія є результатом підсумовування виходів, що відповідають яскраво-червоній та темно-червоній лініям [11];

- RNN – Recurrent Neural Network – Рекурентна нейронна мережа.

Фундаментальна архітектура RNN складається з наступних компонентів:

1. Вхід. Вхідний шар відповідає за отримання послідовних даних. Наприклад, у задачах обробки природної мови кожен елемент послідовності може відповідати слову або символу.

2. Прихований шар. Прихований шар є основним компонентом ШНМ. Він підтримує внутрішній стан або прихований стан, який змінюється, коли

мережа обробляє кожен елемент послідовності. Прихований стан фіксує інформацію з попередніх часових кроків і, отже, відповідає за збереження контексту та моделювання залежностей всередині послідовності.

3. Рекурентний зв'язок. Це найважливіша особливість ШНМ. Це набір вагових коефіцієнтів і зв'язків, які зациклюються назад до прихованого шару від самого себе на попередньому часовому кроці. Цей цикл дозволяє ШНМ підтримувати та оновлювати свій прихований стан під час обробки кожного елемента послідовності. Повторюване з'єднання дозволяє мережі пам'ятати і використовувати інформацію з минулого.

4. Вихідний шар. Вихідний шар відповідає за створення прогнозів або виходів на основі інформації в прихованому стані. Кількість нейронів у цьому шарі залежить від конкретного завдання. Наприклад, у мовній моделі це може бути шар softmax для передбачення наступного слова в послідовності.

Графічно схему RNN архітектури показано за допомогою Рис. 1.17.

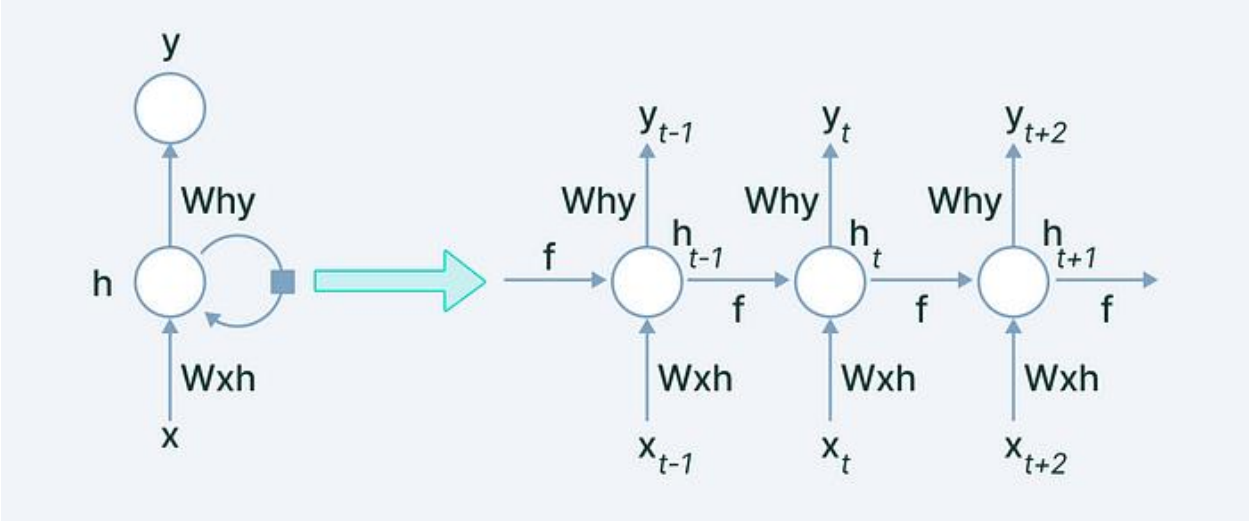

Рисунок 1.17 – Схема RNN архітектури

Схему-порівняння трьох описаних вище архітектур методів машинного навчання для прогнозування часових рядів, зокрема фондових ринків, показано за допомогою Рис. 1.18.

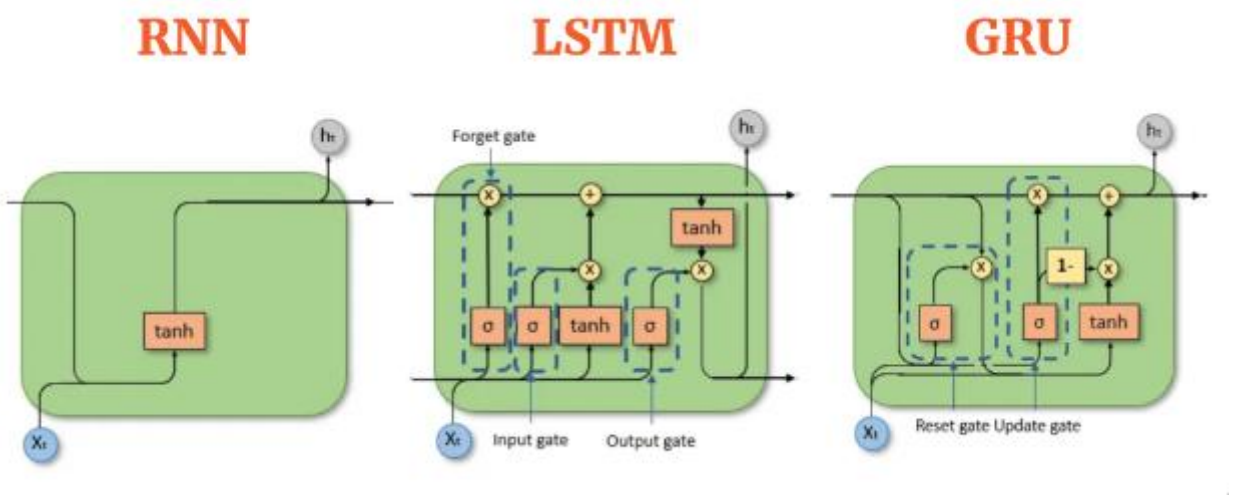

Рисунок 1.18 – Схема-порівняння методів машинного навчання для прогнозування

Як показано вище, в той час як RNN, LSTM і GRU працюють за принципом рекурентності та послідовної обробки даних. Кожна модель має свої сильні сторони та ідеальні сфери застосування, і ви можете вибрати модель залежно від конкретного завдання, даних та доступних ресурсів [12].

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір середовища розробки

Jupyter Notebook (Юпітер-ноутбук) – це програма-блокнот для запису, передавання та запуску коду (див. Рис. 2.1). Нею можна користуватися як своєрідним середовищем розробки. Вона існує як веб-сервіс, тобто доступна через інтернет і дозволяє передавати код іншим розробникам.

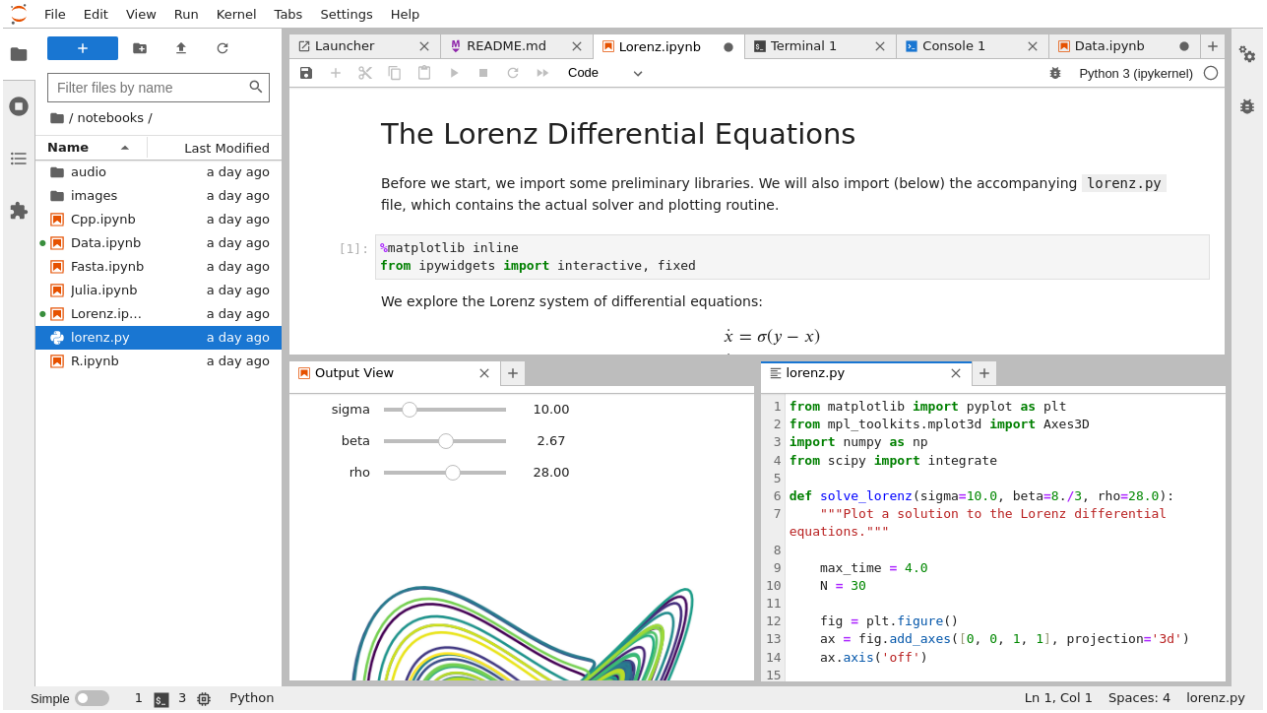

Рисунок 2.1 – Вигляд платформи Jupyter Notebook

Найчастіше середовище використовують для Python, але воно існує і для інших мов програмування. Jupyter Notebook підтримує мови Ruby, Perl, R, MATLAB, Julia та інші. Часто це спеціалізовані мови для завдань, які передбачають швидке написання і виконання маленької програми.

Відмінність Юпітер-ноутбука від традиційних середовищ розробки – в його інтерактивності. Програма дає змогу запускати окремі ділянки та блоки коду, виконувати їх у будь-якому порядку. А результати роботи одразу можна вивести в те саме вікно поруч із кодом.

Юпітер-ноутбуками у множині називають документи з кодом, створені в середовищі. Найчастіше з юпітер-ноутбуками працюють програмісти на Python. Так склалося історично: проєкт виріс з IPython (див. Рис. 2.2), особливого розширення для «Пітона», хоча зараз у платформи є підтримка і для інших мов.

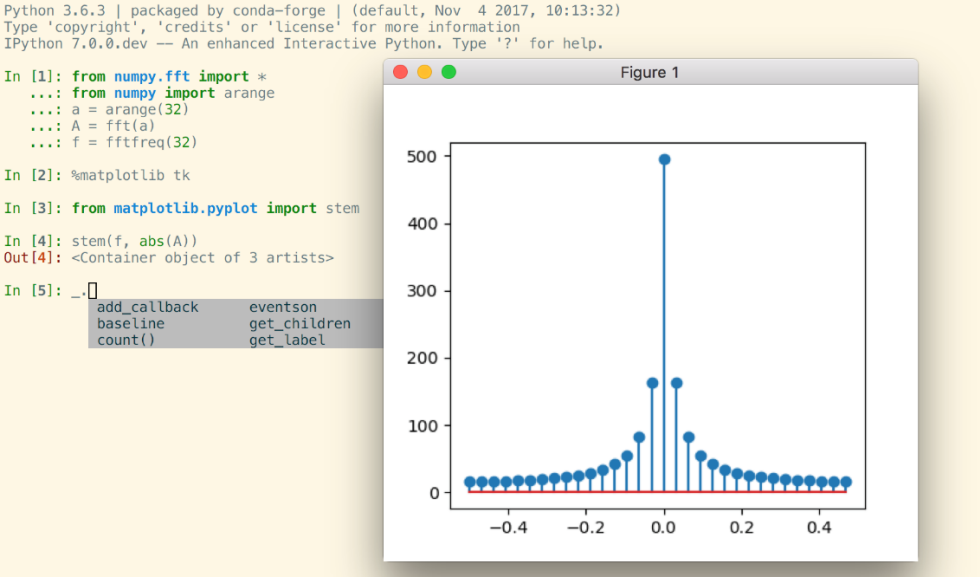

Рисунок 2.2 – Вигляд платформи IPython

Основні сфери використання середовища – big data і data science, машинне навчання, математична статистика та аналітика. У цих напрямках стала в пригоді здатність Jupyter Notebook виводити дані туди ж, де написаний код. Виходить ніби сторінка блокнота: в одному місці зібрані ділянки коду, результати їхнього виконання, таблиці, ділянки та графіки.

Але спробувати Jupyter Notebook для своїх проектів можна і поза цією сферою. Якщо ваша галузь передбачає часту роботу з документами і графіками, можливо, вам сподобається це середовище. Jupyter-ноутбук дає ті самі можливості, що і стандартна IDE, але водночас він гнучкіший і, як кажуть його творці, документоцентричний. Тобто все написане має вигляд документа і зібрано в одному місці. Це включає в себе:

- писати код у спеціальному середовищі з підсвічуванням синтаксису, виправленням помилок та іншими можливостями IDE;

- запускати різні ділянки коду в довільній послідовності або написану програму цілком;
- завантажувати якісь дані, обробляти і перетворювати їх, не зачіпаючи при цьому інші ділянки програми;
- вставляти і виводити результати, включно з візуалізацією, просто посеред коду;
- ділитися кодом з іншими розробниками і давати їм загальний доступ до проекту;
- організовувати командну роботу, коли в кожного програміста – своє завдання, пов'язане з іншими;
- писати супровідний текст і оформляти «документ» так, щоб він мав гарний і зрозумілий вигляд.

Jupyter Notebook існує у двох версіях: хмарна і для комп'ютера. Обидві безкоштовні, з відкритим вихідним кодом. Хмарною версією можна користуватися прямо через браузер. Потрібно тільки підключення до інтернету: принцип роботи в неї як, наприклад, у Google Документів.

Локальна версія дає можливість працювати з рідкісними бібліотеками, яких може не бути в хмарі. До того ж на окремому комп'ютері середовище здатне працювати швидше, а в розробника більше контролю.

Для завантаження локальної версії знадобиться Python і встановлений у нього пакет Jupyter. Його можна завантажити через консоль за допомогою вбудованої в Python утиліти pip. А ще Юпітер-ноутбук входить до складу розширення Anaconda (див. Рис. 2.3).

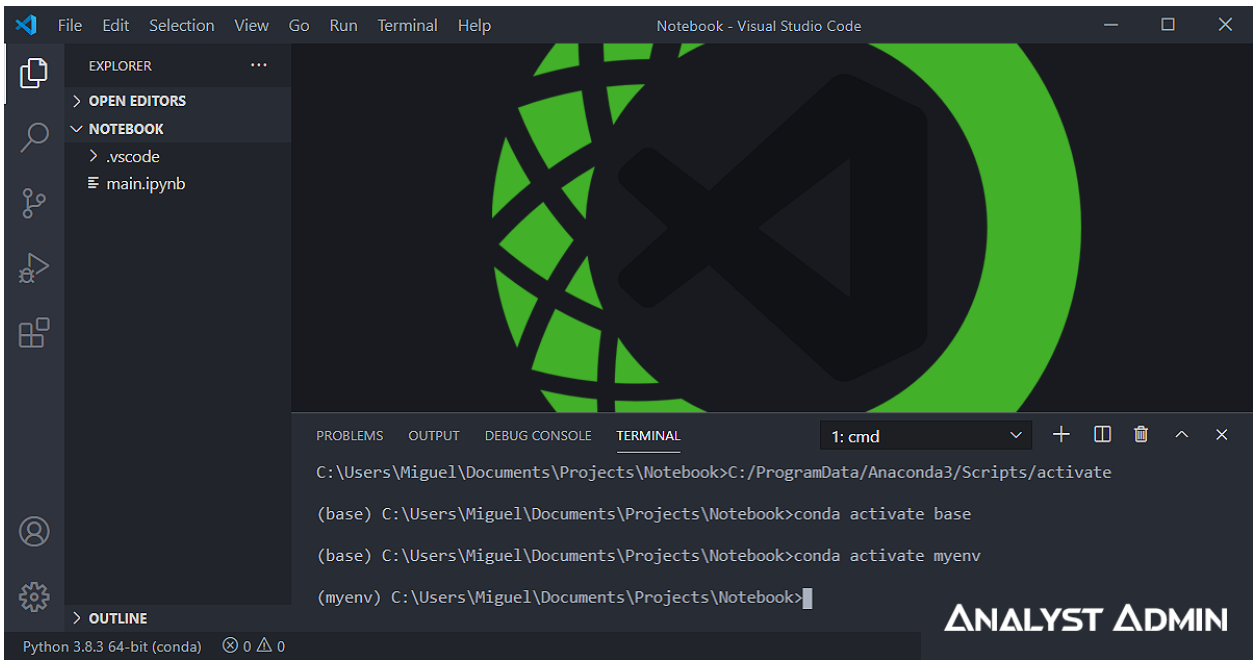

Рисунок 2.3 – Вигляд розширення Anaconda у ПЗ Visual Studio Code

Завантажені пакети запустять локальний Jupyter-сервер, і інтерфейс середовища відкриється в браузері. Ним можна користуватися на свій розсуд, але доступу до хмари в локального сервера немає. Інтерфейс програми схожий одночасно на текстовий редактор і середовище розробки. Можна уявити Юпітер-ноутбук як документ з можливістю писати і виконувати код. Там є інструменти і для форматування самого документа, і для запуску або налагодження програм.

Якщо розробник пише Python, він може просто створити файл і почати програмувати, як у звичайному середовищі. Для підключення інших мов знадобиться скористатися спеціальними командами – їх називають магічними. Магічна команда перемикає середовище на зазначену мову. Так можна підключити навіть bash, консольну мову команд для операційних систем.

Код у Юпітер-ноутбуці розділений на окремі ділянки – блоки. Кожен блок можна запускати окремо від інших. Між блоками може перебувати текст, графіка, результати виконання коду тощо. Виходить своєрідний інтерактивний документ – одночасно програма і текстовий файл.

Крім написання коду, в Юпітер-ноутбуці можна писати текст. Це не просто коментарі: у середовищі є можливості для створення повноцінної

інтерактивної статті з кодом. Jupyter Notebook підтримує розмітку Markdown, яка дає змогу створювати заголовки та списки, додавати інтерактивні посилання та робити багато іншого.

Так можна створювати аналітичні звіти, статті та інтерактивні параграфи для підручників. Тому Jupyter Notebook люблять в аналітиці даних і статистиці, де часто бувають потрібні точні звіти та візуалізації з результатами.

Картинки, графіки та візуалізацію можна додавати як у текстовий документ. Крім того, Юпітер-ноутбук підтримує математичні формули. Графіку можна виводити інтерактивно: запускати блоки коду, щоб вони намалювали потрібну візуалізацію. Результат останнього запуску не зітреться після виходу: він залишиться в документі, його зможуть переглянути інші люди або сам власник, коли наступного разу відкриє ноутбук.

Для візуалізації Jupyter Notebook використовує бібліотеки мов. Наприклад, графіки для Python найчастіше малюються через Matplotlib (див. Рис. 2.4). Юпітер-ноутбук підтримує й інші розширення для візуалізації, зокрема в інтерактивному режимі. Щоправда, деякі з них платні або повільні.

Рисунок 2.4 – Демонстрація графічних можливостей бібліотеки Matplotlib

У Jupyter Notebook, як і в будь-якій IDE, можна запустити код. Відмінність у тому, що результати показуються відразу і відображаються в тому ж документі. Для запуску є спеціальна панель із кнопками: запустити, зупинити, налагодити тощо.

Якщо код працює не так, можна переписати його і запустити знову. Виведений у документ результат при цьому зміниться. Коли все буде готово, можна зберігати файл і виходити: під час наступного входу всі результати будуть на місці і код не знадобиться запускати заново.

У випадку з локальною версією файли зберігаються на пристрої. До хмарної знадобиться підключити сховище, в якому лежатимуть створені ноутбуки. Найчастіше користуються Google Colab – відгалуженням Google Drive для розробки та досліджень. Можливості Colab дають змогу запускати код у браузері, зберігати документи та ділитися ними.

Для документа в хмарі можна налаштувати загальний доступ, щоб його могли побачити інші фахівці. Це зручно, наприклад, при створенні звітів або при командній розробці [13].

Jupyter Notebook має ряд переваг у порівнянні з іншими інструментами для розробки та аналізу даних. Порівняння з іншими інструментами для розробки показано за допомогою Табл. 2.1.

Таблиця 2.1 – Порівняння Jupyter Notebook з іншими інструментами

Характеристика	**Jupyter Notebook**	**IDE (Integrated Development Environment)**	**JupyterLab**
Інтерактивність	Дозволяє покроково виконувати та відображати результати в осередках.	Зазвичай, вимагає окремого запуску програми.	Деяка інтерактивність, але не така багата, як Jupyter Notebook.
Візуалізація даних	Зручний візуалізації даних	Вимагає додаткових бібліотек чи	Підтримує візуалізацію даних і має

	вбудованими засобами.	окремих інструментів.	великі можливості.
Навчання та презентації	Прекрасно підходить для навчання та створення інтерактивних презентацій.	Орієнтований на розробку та налагодження коду.	Має потенціал для використання у навчанні та презентаціях.
Багатозадачність	Обмежена можливість працювати з кількома файлами одночасно.	Підтримує відкриття декількох файлів та проектів в одному середовищі.	Підтримує відкриття декількох ноутбуків та файлів у різних вкладках.
Розширюваність	Обмежена підтримка плагінів та розширень.	Має безліч плагінів та розширень для додаткової функціональності.	Має високий рівень розширюваності за допомогою плагінів і розширень.

Саме тому було обрано Jupyter Notebook в якості середовища розробки для розробки, дослідження й аналізу великих даних в цілях прогнозування ринкових трендів [14].

2.2 Вибір мови програмування

Python – це інтерпретована, об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Високорівневі вбудовані структури даних у поєднанні з динамічною типізацією та динамічним зв'язуванням роблять її дуже привабливою для швидкої розробки додатків, а також для використання в якості мови сценаріїв або мови-клею для з'єднання існуючих компонентів. Простий, легкий для вивчення синтаксис Python робить акцент на читабельності, а отже, знижує витрати на підтримку програм. Python підтримує модулі та пакети, що заохочує модульність програм та повторне використання коду. Інтерпретатор Python та велика стандартна бібліотека

доступні у вигляді вихідних кодів або двійкових файлів безкоштовно для всіх основних платформ і можуть вільно розповсюджуватися.

Часто програмісти закохуються в Python через підвищену продуктивність, яку вона забезпечує. Оскільки немає етапу компіляції, цикл редагування-тестування-налагодження відбувається неймовірно швидко. Налагоджувати програми на Python легко: помилка або неправильний ввід ніколи не призведе до помилки сегментації. Натомість, коли інтерпретатор виявляє помилку, він згенерує виключення. Якщо програма не перехоплює виняток, інтерпретатор виводить трасування стеку. Налагоджувач на рівні коду дозволяє перевіряти локальні та глобальні змінні, обчислювати довільні вирази, встановлювати точки зупинки, переглядати код по одному рядку за раз тощо.

Налагоджувач написаний самою мовою Python, що свідчить про інтроспективну силу Python. З іншого боку, часто найшвидший спосіб налагодити програму – це додати кілька операторів виводу у вихідний код: швидкий цикл редагування-тестування-налагодження робить цей простий підхід дуже ефективним.

Python широко використовується для розробки веб-сайтів і програмного забезпечення, автоматизації завдань, аналізу даних і візуалізації даних. Оскільки її відносно легко вивчити, Python використовується багатьма непрограмістами, такими як бухгалтери та науковці, для вирішення різноманітних повсякденних завдань, таких як організація фінансів.

«Написання програм – це дуже творча та корисна діяльність, – каже викладач Мічиганського університету та Coursera Чарльз Р. Северанс у своїй книзі «Python for Everybody» («Пітон для всіх»). «Ви можете писати програми з багатьох причин, починаючи від заробітку до вирішення складної проблеми аналізу даних, щоб розважитися і допомогти комусь вирішити проблему».

Що можна робити за допомогою python? Деякі речі включають в себе:

- аналіз даних і машинне навчання;
- веб-розробка;

- автоматизація або написання сценаріїв;
- тестування та створення прототипів програмного забезпечення;
- повсякденні завдання.

Python став основним інструментом у науці про дані, дозволяючи аналітикам даних та іншим фахівцям використовувати мову для проведення складних статистичних розрахунків, створення візуалізацій даних, побудови алгоритмів машинного навчання, маніпулювання та аналізу даних, а також для виконання інших завдань, пов'язаних з даними.

За допомогою Python можна створювати широкий спектр різноманітних візуалізацій даних, таких як лінійні та гістограми, кругові діаграми, гістограми та 3D-графіки. Python також має низку бібліотек, які дозволяють програмістам швидше та ефективніше писати програми для аналізу даних та машинного навчання, наприклад, TensorFlow та Keras.

Python часто використовується для розробки бек-енду веб-сайту або додатку – частин, які користувач не бачить. Роль Python у веб-розробці може включати надсилання даних на сервери та з серверів, обробку даних та зв'язок з базами даних, маршрутизацію URL-адрес та забезпечення безпеки. Python пропонує кілька фреймворків для веб-розробки. Найпоширеніші з них – Django та Flask (див. Рис. 2.5).

Рисунок 2.5 – Демонстрація логотипів фреймворків для веб-розробки Django та Flask

Python використовують бекенд-інженери, full stack-інженери, Python-розробники, інженери програмного забезпечення та DevOps-інженери, які займаються веб-розробкою.

Якщо вам доводиться повторювати одну і ту ж задачу, ви можете працювати ефективніше, автоматизувавши її за допомогою Python. Написання коду, який використовується для побудови таких автоматизованих процесів, називається написанням сценаріїв. У світі кодування автоматизація може використовуватися для перевірки помилок у декількох файлах, перетворення файлів, виконання простих математичних розрахунків і видалення дублікатів даних.

Python може використовуватися навіть відносними новачками для автоматизації простих завдань на комп'ютері, таких як перейменування файлів, пошук і завантаження онлайн-контенту або надсилання електронних листів чи текстів через певні проміжки часу.

У розробці програмного забезпечення Python може допомогти у виконанні таких завдань, як контроль збірки, відстеження помилок та тестування. За допомогою Python розробники програмного забезпечення можуть автоматизувати тестування нових продуктів або функцій. Деякі інструменти Python, що використовуються для тестування програмного забезпечення, включають Green та Requestium (див. Рис. 2.6).

Рисунок 2.6 – Демонстрація логотипів інструментів Green та Requestium

Python – це мова не лише для програмістів та аналітиків даних. Вивчення Python може відкрити нові можливості для тих, хто менше працює з даними, наприклад, для журналістів, власників малого бізнесу або маркетологів у

соціальних мережах. Python також може дозволити непрограмістам спростити певні завдання в їхньому житті. Ось лише кілька завдань, які можна автоматизувати за допомогою Python:

- відстеження цін на фондовому ринку або криптовалюти;
- оновлення списку покупок продуктів;
- перейменування великих пакетів файлів;
- перетворення текстових файлів у електронні таблиці;
- випадковий розподіл обов'язків між членами сім'ї;
- автоматичне заповнення онлайн-форм.

Саме тому обрана мова програмування Python ідеально підходить для розробки для розробки, дослідження й аналізу великих даних в цілях прогнозування ринкових трендів [15].

2.3 Збір та підготовка даних

Дані для розробки, дослідження й аналізу великих даних в цілях прогнозування ринкових трендів за допомогою машинного навчання були отримані на порталі www.kaggle.com (див. Рис. 2.7).

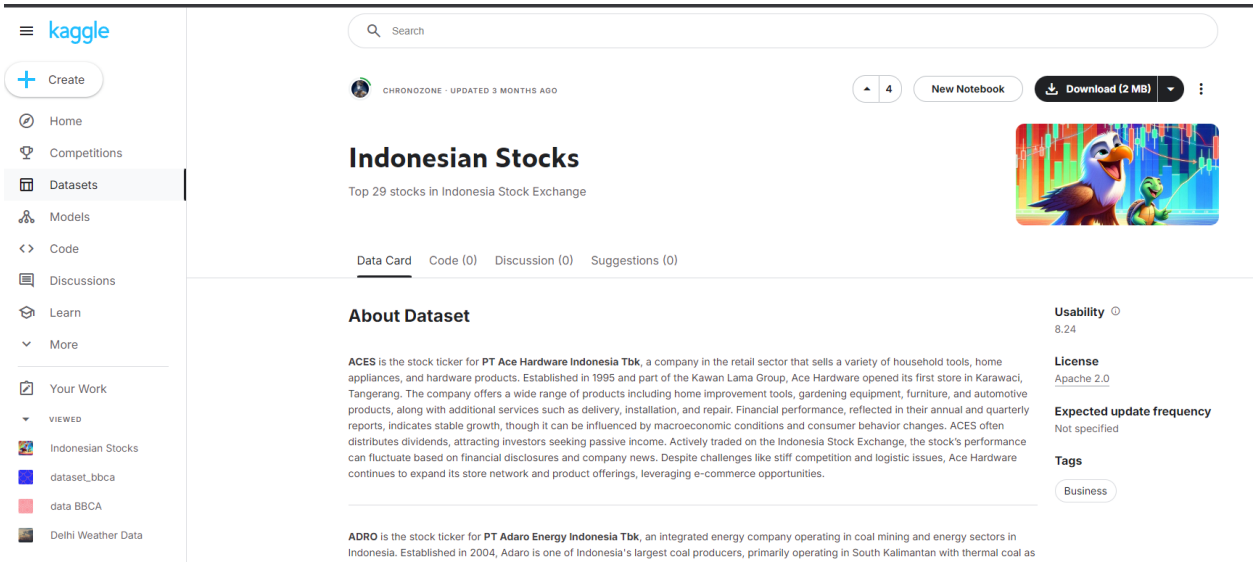

Рисунок 2.7 – Демонстрація порталу www.kaggle.com

Kaggle – це платформа для змагань з науки про дані, де науковці та інженери з машинного навчання можуть змагатися один з одним у створенні найкращих моделей для вирішення конкретних проблем або аналізу певних наборів даних. Платформа також надає спільноту, де користувачі можуть співпрацювати над проектами, обмінюватися кодом і наборами даних, а також вчитися на роботі один одного. Заснована у 2010 році, компанія Google придбала Kaggle у 2017 році, і тепер платформа є частиною Google Cloud.

На Kaggle проводяться різноманітні конкурси, спонсоровані організаціями, від прогнозування медичних результатів до класифікації зображень або виявлення шахрайських транзакцій. Учасники можуть подавати свої моделі і бачити, як вони працюють у публічній таблиці лідерів, а також отримувати відгуки від інших конкурентів і спільноти.

Окрім змагань, Kaggle також пропонує публічні набори даних, блокноти для машинного навчання та навчальні посібники, які допомагають користувачам вивчати та практикувати свої навички в галузі науки про дані та машинного навчання. Він став популярною платформою як для початківців, так і для досвідчених науковців, які вдосконалюють свої навички, створюють свої портфоліо та спілкуються з іншими представниками індустрії.

Kaggle в першу чергу використовується для проведення змагань з науки про дані, де учасники можуть змагатися один з одним у створенні найкращих моделей для вирішення конкретних проблем. Організації з усього світу спонсорують ці змагання, і вони охоплюють широкий спектр тем, таких як класифікація зображень, обробка природної мови та прогнозне моделювання.

Kaggle також використовується для:

- навчання: kaggle надає такі ресурси, як публічні набори даних, навчальні посібники з машинного навчання та блокноти з кодом, які дозволяють користувачам вивчати та практикувати навички з науки про дані;
- співпраці: kaggle дозволяє користувачам формувати команди та співпрацювати над поданням заявок, обмінюватися кодом і наборами даних, а також надавати один одному зворотній зв'язок;

- створення спільноти: kaggle має велику спільноту науковців з даних, інженерів машинного навчання та ентузіастів даних, що надає користувачам платформу для спілкування, обміну ідеями та спільної роботи над проектами;
- Дослідження: набори даних та конкурси kaggle є важливими для дослідницьких цілей, що робить його платформою для тестування та вдосконалення алгоритмів машинного навчання.

Загалом, Kaggle – це універсальна платформа, яка пропонує широкий спектр можливостей для дослідників даних та інженерів машинного навчання, від навчання та співпраці до досліджень. Змагання Kaggle – це виклики, в яких аналітики даних та інженери машинного навчання змагаються за створення найкращих моделей для вирішення конкретних проблем або аналізу певних наборів даних. Спонсорами цих змагань виступають різні організації, від бізнесу до академічних установ, а учасники з усього світу мають право змагатися.

Змагання, як правило, включають набір даних і проблему, а учасники повинні розробити і представити модель, яка вирішує проблему або прогнозує цільову змінну з найвищою точністю. Конкурси мають різну структуру, наприклад, класифікацію, регресію або комп'ютерний зір, залежно від характеру набору даних і проблеми, що вирішується.

Учасники співпрацюють і обмінюються ідеями протягом усього процесу, а деякі змагання навіть пропонують призи командам з найкращими результатами. Учасники також можуть брати участь в обговореннях і форумах, пов'язаних зі змаганнями, де вони можуть ставити запитання, ділитися своїм прогресом і отримувати відгуки від інших учасників [16].

Для проекту був отриманий набір даних Indonesian Stocks (Top 29 stocks in Indonesia Stock Exchange). Його опис містить таку інформацію:

BBCA – це тікер для акцій PT Bank Central Asia Tbk, одного з найбільших банків Індонезії (див. Рис. 2.8). Заснований у 1957 році, Банк Центральна Азія став провідним банком, відомим своїми інноваційними та якісними

банківськими послугами. Його основні сегменти бізнесу включають роздрібний, корпоративний та комерційний банкінг, пропонуючи широкий спектр банківських продуктів та послуг, таких як заощадження, кредити, інвестиції та транзакційні послуги.

Рисунок 2.8 – Демонстрація логотипу BBCA

BBCA також фокусується на технологіях та інноваціях, розвиваючи цифрові банківські послуги для задоволення потреб своїх клієнтів. Фінансові показники банку, відображені в його річних та квартальних фінансових звітах, демонструють стабільне зростання, що підтримується ефективним управлінням ризиками та високими темпами економічного зростання в Індонезії. BBCA регулярно виплачує дивіденди, що робить його привабливим вибором для інвесторів, які шукають пасивний дохід.

Акції компанії активно торгуються на Індонезійській фондовій біржі, а їхні цінові коливання часто стають головною подією на ринку. Незважаючи на такі виклики, як інтенсивна конкуренція в банківській галузі, регуляторні зміни та кредитні ризики, BBCA вбачає значні можливості для зростання в розширенні своєї філіальної мережі, збільшенні проникнення на ринок та подальшому розвитку своїх цифрових послуг [17, 18].

2.4 Архітектура проекту

Архітектура проекту для розробки, дослідження й аналізу великих даних в цілях прогнозування ринкових трендів має такі базові спрощені кроки для реалізації:

1. Імпорт необхідних бібліотек: Використовуються бібліотеки для роботи з даними (pandas, numpy), візуалізації (seaborn, matplotlib), масштабування даних (MinMaxScaler), та побудови LSTM моделі (keras).
2. Завантаження та попередня обробка даних:

- дані завантажуються з CSV файлу;
- перетворення типів даних для колонки Date і Volume;
- установка Date як індексу;
- перевірка наявності та коректності колонки 'Adj Close'.

3. Масштабування даних:

- дані масштабуються до діапазону від 0 до 1 для покращення роботи LSTM моделі [19].

4. Створення тренувальних і тестових наборів даних:

- тренувальний набір включає 95% даних;
- тестовий набір включає останні 5%.

5. Побудова та тренування LSTM моделі:

- модель складається з двох LSTM шарів та двох Dense шарів.
- модель тренується на тренувальних даних.

6. Прогнозування:

- отримуються прогнозовані значення на основі тестових даних;
- результати прогнозування виводяться у вигляді графіку.

7. Візуалізація:

- графіки показують фактичні та прогнозовані значення для оцінки якості моделі.

8. Виведення результату точності прогнозування у вигляді коренів середньоквадратичної помилки [20].

2.5 Програмна реалізація

Програмна реалізація складається з таких основних блоків програми:

1. Імпорт бібліотек. Програма починається з імпорту необхідних бібліотек, таких як `pandas`, `numpy`, `seaborn`, `matplotlib`, а також бібліотек для роботи з моделлю LSTM, як-от `keras` і `yfinance`. Ці бібліотеки забезпечують роботу з даними, візуалізацію та створення моделі. Етап продемонстровано за допомогою Рис. 2.9.

```
# Імпортуємо необхідні бібліотеки
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
import datetime
from IPython.display import Markdown
from datetime import datetime
import textwrap

# Імпортуємо бібліотеки для масштабування даних і виявлення аномалій
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest

# Імпортуємо бібліотеки для створення та навчання LSTM моделі
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Імпортуємо yfinance для завантаження даних
import yfinance as yf
```

Рисунок 2.9 – Один з блоків програми

2. Інформація про джерело даних. За допомогою `Markdown` виводиться інформація про джерело даних, які використовуються для аналізу. У цьому випадку використовуються дані з `yfinance` для акцій BBCA за період з 01/01/2004 по 31/12/2024. Етап продемонстровано за допомогою Рис. 2.10.

```
# Виводимо інформацію про джерело даних
Markdown("""
Джерело даних: yfinance\n
- BBCA (01/01/2004 - 31/12/2024)\n
""")
```

Рисунок 2.10 – Один з блоків програми

3. Зчитування та підготовка даних. Програма зчитує дані про акції з локальних CSV-файлів і перетворює їх у формат `DataFrame`. Дата зберігається в колонці `Date`, обсяг торгів – у колонці `Volume`. Далі всі дані об'єднуються в один `DataFrame` для подальшого аналізу. Етап продемонстровано за допомогою Рис. 2.11.

```
# Читаємо дані з локального каталогу
for x, y, z in zip(stock_list, stock_code, stock_ma):
    globals()[x] = pd.read_csv(f'data//{y}_15_01_2023.csv', encoding='utf-8')
    globals()[x]['Date'] = pd.to_datetime(globals()[x]['Date'])
    globals()[x]['Volume'] = globals()[x]['Volume'].astype(int)
    globals()[z] = globals()[x]
    globals()[z] =  globals()[z].set_index('Date')
    com_list.append(globals()[z])
```

Рисунок 2.11 – Один з блоків програми

4. Аналіз даних та візуалізація. Цей блок програми будує різні графіки для візуалізації історичних даних про акції:

- графік ціни відкриття: показує зміни в ціні відкриття акцій за період;
- графік ціни закриття: відображає ціни закриття акцій;
- графік об'єму торгів: демонструє зміни в обсязі торгів протягом періоду. Етап продемонстровано за допомогою Рис. 2.12 [21].

```
# Побудова лінійного графіка на основі значень 'Open' ціни
for x, y in zip(stock_list, stock_code):
    ax.plot('Date', 'Open', data=globals()[x], label=y)

ax.set_ylabel('Відкриття Ціни')
ax.tick_params(axis='x', rotation=45)
ax.grid(False)
ax.legend()

plt.show()

# Побудова графіка для ціни закриття
chart = f'Ціна Закриття {stock_code[0]}, 2004-2024'

Markdown(f"""
### {chart}
""")

fig, ax = plt.subplots(figsize=(14, 8))
fig.suptitle(f'{chart}', fontweight='bold')

for x, y in zip(stock_list, stock_code):
    ax.plot('Date', 'Close', data=globals()[x], label=y)

ax.set_ylabel('Ціна Закриття')
ax.tick_params(axis='x', rotation=45)
ax.grid(False)
ax.legend()

plt.show()
```

Рисунок 2.12 – Один з блоків програми

5. Підготовка даних для прогнозування. Вибирається колонка `Open` для прогнозування, дані масштабуються з використанням `MinMaxScaler`. Далі дані розділяються на тренувальні та тестові набори, з яких формується вхідний масив для LSTM-моделі. Етап продемонстровано за допомогою Рис. 2.13.

```
# Масштабування даних
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
```

Рисунок 2.13 – Один з блоків програми

6. Створення та навчання LSTM моделі. Програма створює нейронну мережу з використанням LSTM шарів. Модель налаштовується і тренується на тренувальних даних. Етап продемонстровано за допомогою Рис. 2.14.

```
# Побудова LSTM моделі
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

Рисунок 2.14 – Один з блоків програми

7. Прогнозування та оцінка точності. Після навчання моделі проводиться прогнозування цін на тестових даних. Прогнозовані значення порівнюються з фактичними значеннями, і обчислюється корінь середньоквадратичної помилки (RMSE) для оцінки точності моделі. Етап продемонстровано за допомогою Рис. 2.15.

```
# Прогнозування цін на основі тестових даних
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

Рисунок 2.5 – Один з блоків програми

8. Візуалізація результатів прогнозування. Результати прогнозування візуалізуються на графіку, де показані фактичні дані, тренувальні дані та передбачені значення. Етап продемонстровано за допомогою Рис. 2.16.

```
# Візуалізація результатів прогнозування
train = data[:training_data_len]
valid = data[training_data_len:]
valid = valid.assign(Predictions=predictions)
```

Рисунок 2.16 – Один з блоків програми

9. Аналіз результатів. Виводиться остаточний аналіз результатів, включаючи RMSE для тренувальних і тестових даних, а також корені середньоквадратичної помилки. Це дозволяє оцінити, наскільки добре модель

змогла передбачити майбутні ціни акцій та виявити можливе перенавчання. Етап продемонстровано за допомогою Рис. 2.17 [22].

```
# Оцінка точності прогнозування для тренувальних даних
# Використовуємо лише ті значення, які можна порівняти за розміром

train_predict = model.predict(x_train)
train_predict = scaler.inverse_transform(train_predict)
```

Рисунок 2.17 – Один з блоків програми

Повний код програмної реалізації проекту наведено в Додатку А.

2.6 Робота програмного забезпечення

Модель використовує машинне навчання для аналізу числових рядів на великих об'ємах даних, що включають близько 4800 рядків. Це дозволяє їй враховувати широкий спектр історичних даних, що є ключовим для точного прогнозування майбутніх трендів. Враховуючи великий обсяг даних і складність обчислень, модель потребує близько 5 хвилин для повної реалізації, що забезпечує збалансованість між швидкістю роботи та точністю прогнозів.

На виході програма надає такі вихідні дані:

- перші кілька рядків вхідних даних, щоб упевнитись в правильності набору числових рядів;
- графік відкриття ціни (open price) для тікера акцій BBCA за 2004-2024 роки;
- графік закриття ціни (close price) для тікера акцій BBCA за 2004-2024 роки;
- об’єм торгів для тікера акцій BBCA за 2004-2024 роки;
- дані про тренування моделі на наборі даних;
- результати прогнозування у вигляді графіка;
- результати оцінки точності моделі (Корінь середньоквадратичної помилки (RMSE) для тренувальних даних і корінь

середньоквадратичної помилки (RMSE) для тестових даних. Для демонстрації роботи програми наведено Рис. 2.18-2.24.

```
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    4803 non-null   datetime64[ns, UTC+07:00]
 1   Open    4803 non-null   int64
 2   High    4803 non-null   int64
 3   Low     4803 non-null   int64
 4   Close   4803 non-null   int64
 5   Volume  4803 non-null   int64
dtypes: datetime64[ns, UTC+07:00](1), int64(5)
memory usage: 225.3 KB
```

Рисунок 2.18 – Вивід програмою перших рядків вхідних даних

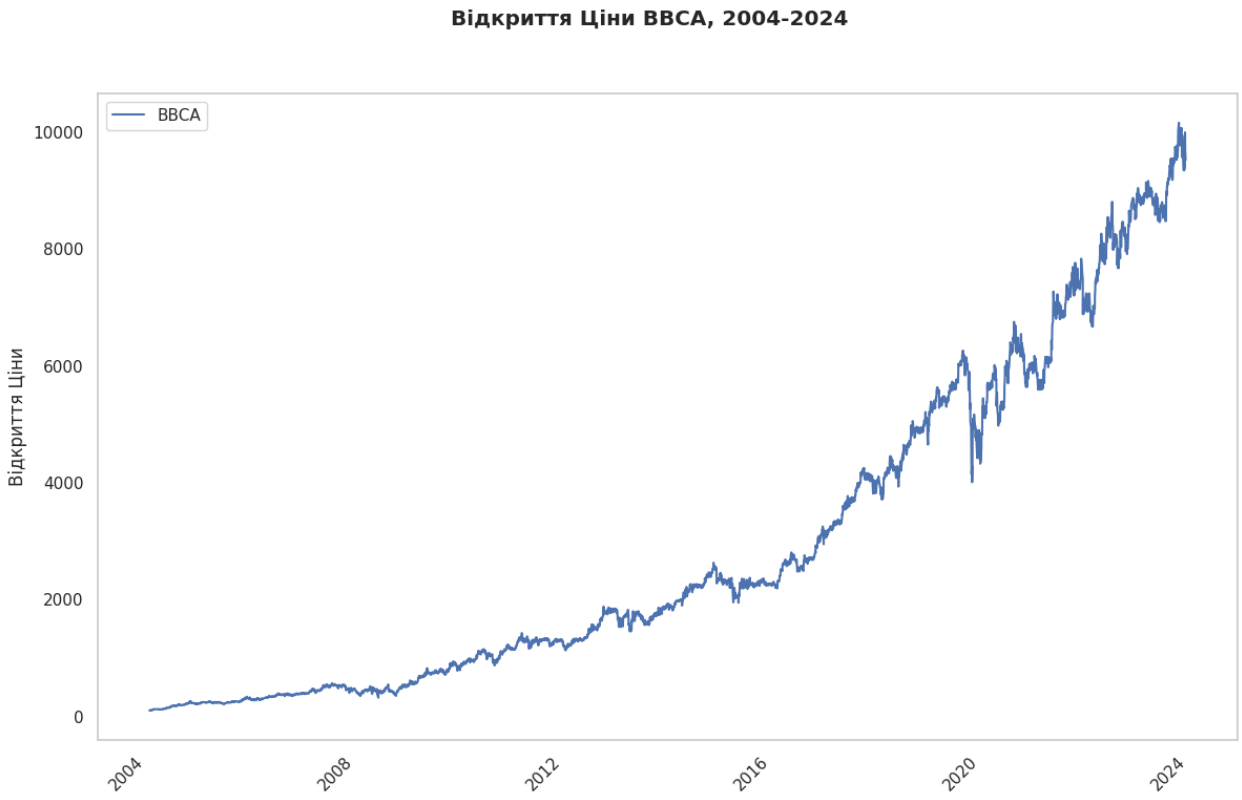

Рисунок 2.19 – Вивід програмою графіка відкриття ціни (open price) для тікера акцій BBCA

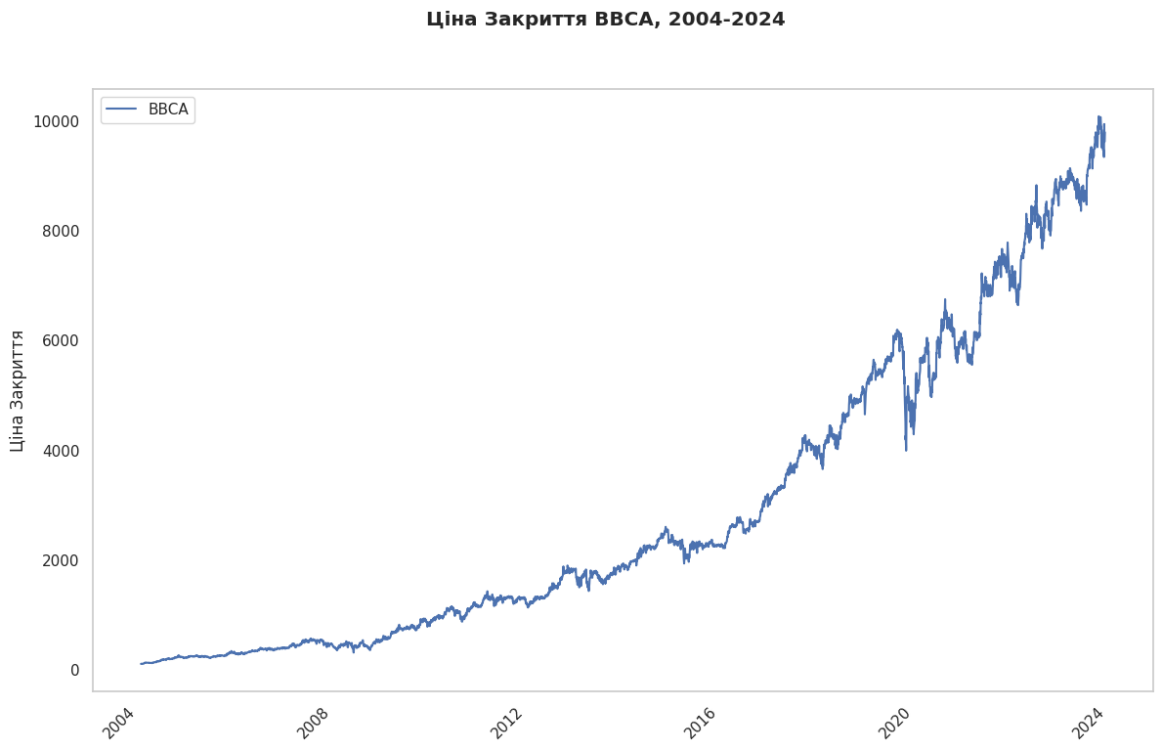

Рисунок 2.20 – Вивід програмою графіка закриття ціни (close price) для тікера акцій BBCA

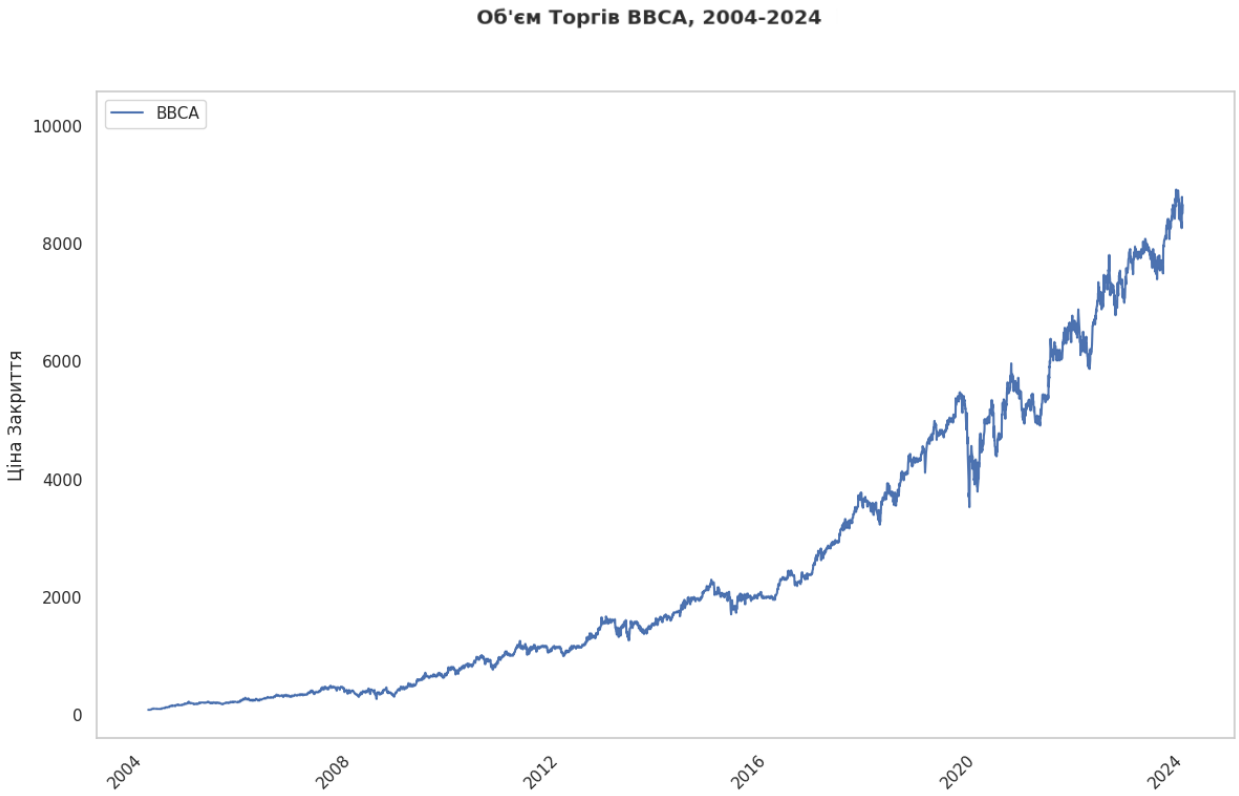

Рисунок 2.21 – Вивід програмою графіка об’єма торгів для тікера акцій BBCA

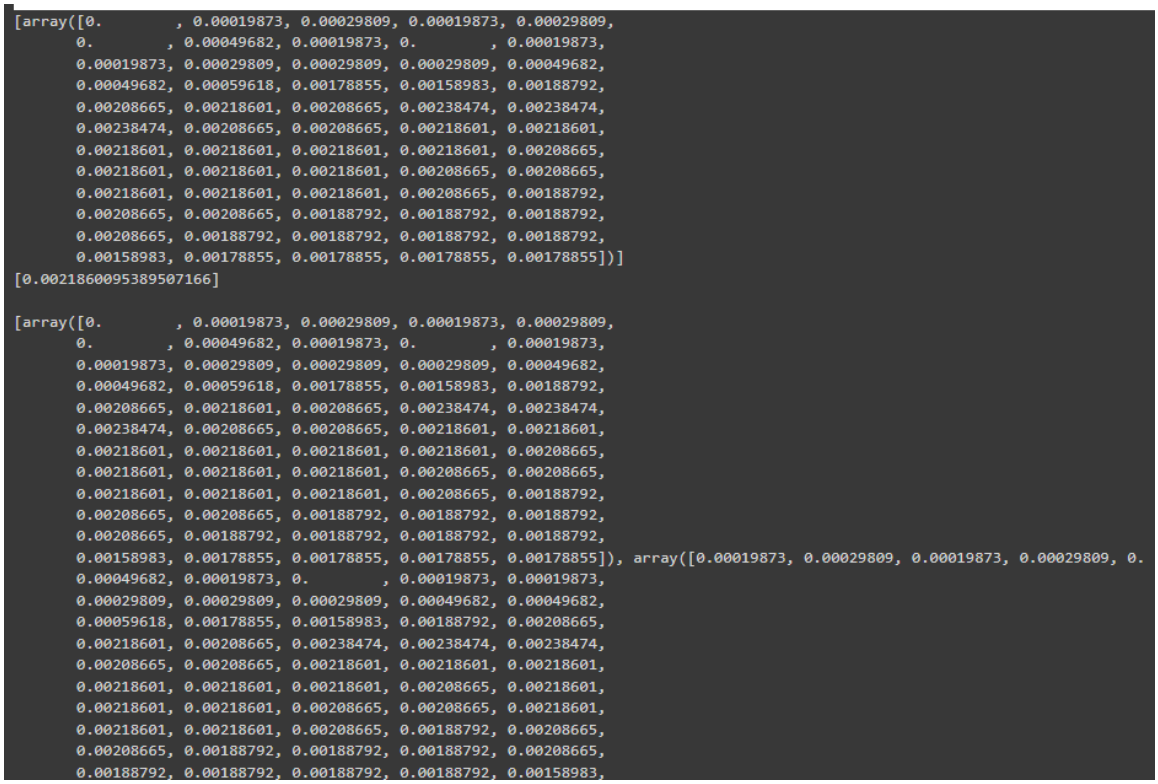

Рисунок 2.22 – Вивід програмою інформації про тренування моделі на наборі даних

Рисунок 2.23 – Вивід програмою графіка прогнозуваня ціни для тікера акцій BBCA

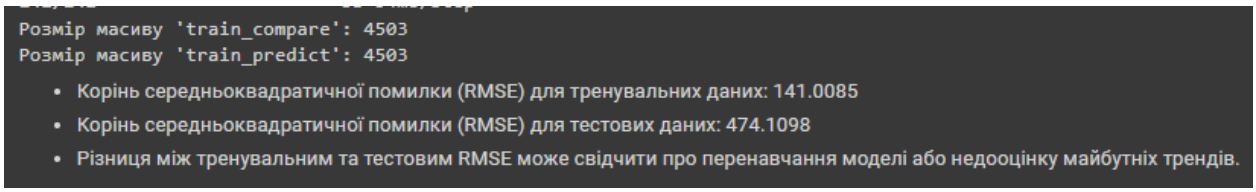

Рисунок 2.24 – Вивід програмою результатів оцінки точності моделі

На виконання програмі було потрібно 5 хвилин 20 секунд і 223 кілобайти пам’яті.

ВИСНОВКИ

Підсумовуючи, під час виконання даної роботи було реалізовано такі основні завдання:

1. Досліджено теоретичну частин, необхідну для виконання даної наукової роботи. Це включило в себе: основи машинного навчання, керовані та некеровані алгоритми, ефективність алгоритмів навчання, узагальнення алгоритмів машинного навчання, гіперпараметри та набори перевірки, основні поняття фондового ринку та його прогнозування й методи машинного навчання для прогнозування
2. У практичній частині було розглянуто й виконано такі моменти:
   - вибір середовища розробки;
   - вибір мови програмування;
   - збір та підготовка даних;
   - архітектура проекту;
   - програмна реалізація;
   - демонстрація роботи програмного забезпечення.

У результаті було отримано модель, що використовує машинне навчання для аналізу числових рядів на великих об'ємах даних, що включають близько 4800 рядків. Це дозволяє їй враховувати широкий спектр історичних даних, що є ключовим для точного прогнозування майбутніх трендів.

Ураховуючи великий обсяг даних і складність обчислень, модель потребує близько 5 хвилин для повної реалізації, що забезпечує збалансованість між швидкістю роботи та точністю прогнозів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. Jung,“Machine Learning: The Basics,” Springer, Singapore, 2022

2. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2023

3. Tishan, Yasiru. (2023). Understanding the Difference Between Supervised and Unsupervised Learning Techniques. 10.13140/RG.2.2.36176.48641.

4. Supervised and Unsupervised Learning for Data Science, Michael W. Berry Azlinah Mohamed Bee Wah Yap, ISBN 978-3-030-22474-5, 2020

5. Beyond Accuracy: Understanding and Measuring Efficiency in Machine Learning Models [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/7-beyond-accuracy-understanding-measuring-efficiency-machine-pushp/

6. What Is Generalization In Machine Learning? [Електронний ресурс] – Режим доступу до ресурсу: https://magnimindacademy.com/blog/what-is-generalization-in-machine-learning/

7. Franz, Arthur, Experiments on the Generalization of Machine Learning Algorithms. 10.1007/978-3-030-93758-4_9, 2022.

8. Tunability: Importance of Hyperparameters of Machine Learning Algorithms, Philipp Probst, Anne-Laure Boulesteix, Bernd Bischl.

9. STOCK MARKET FORECASTING: A REVIEW OF LITERATURE, Srivatsa Maddodi, K. G. Nandha Kumar, eISSN 2598-246X

10. The Ultimate Guide to Building Your Own LSTM Models [Електронний ресурс] – Режим доступу до ресурсу: https://www.projectpro.io/article/lstm-model/832

11. Understanding GRU Networks [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

12. Compare the different Sequence models (RNN, LSTM, GRU, and Transformers) [Електронний ресурс] – Режим доступу до ресурсу: https://aiml.com/compare-the-different-sequence-models-rnn-lstm-gru-and-transformers/

13. Jupyter Notebook [Електронний ресурс] – Режим доступу до ресурсу: https://blog.skillfactory.ru/glossary/jupyter-notebook/

14. Для чого використовують Jupyter Notebook? [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/ru/jupyter-notebook/

15. What Is Python Used For? A Beginner’s Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

16. What Is Kaggle and What Is It Used For? [Електронний ресурс] – Режим доступу до ресурсу: https://www.coursera.org/articles/kaggle

17. Indonesian Stocks [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/datasets/chronozone/indonesian-stocks?select=BBCA.csv

18. Downey, Allen B. (May 2012). Think Python: How to Think Like a Computer Scientist (version 1.6.6 ed.). Cambridge University Press. ISBN 978-0-521-72596-5.

19. Hamilton, Naomi (5 August 2008). "The A-Z of Programming Languages: Python". Computerworld. Archived from the original on 29 December 2008. Retrieved 31 March 2010.

20. Lutz, Mark (2013). Learning Python (5th ed.). O'Reilly Media. ISBN 978-0-596-15806-4.

21. Summerfield, Mark (2009). Programming in Python 3 (2nd ed.). Addison-Wesley Professional. ISBN 978-0-321-68056-3.

22. Ramalho, Luciano (May 2022). Fluent Python. O'Reilly Media. ISBN 978-1-4920-5632-4.

ДОДАТОК А
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

Файл project.py

```
from IPython.display import Markdown

# Виводимо інформацію про джерело даних
Markdown("""
Джерело даних: yfinance\n
- BBCA (01/01/2004 – 31/12/2024)\n
""")

# Імпортуємо необхідні бібліотеки
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
import datetime
from IPython.display import Markdown
from datetime import datetime
import textwrap

# Імпортуємо бібліотеки для масштабування даних і виявлення аномалій
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest

# Імпортуємо бібліотеки для створення та навчання LSTM моделі
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Імпортуємо yfinance для завантаження даних
import yfinance as yf

# Встановлюємо тему для графіків
sns.set_theme(style="whitegrid")

# Функція для автоматичного переносу міток на графіку
def wrap_labels(ax, width, break_long_words=False):
    labels = []
    for label in ax.get_xticklabels():
        text = label.get_text()
        labels.append(textwrap.fill(text, width=width,
break_long_words=break_long_words))
    ax.set_xticklabels(labels, rotation=0)

# Створюємо списки акцій, їх кодів та ковзних середніх
```

```
stock_list = ['bbca']
stock_code = ['BBCA']
stock_ma = ['bbca_ma']
com_list = []

# Читаємо дані з локального каталогу
for x, y, z in zip(stock_list, stock_code, stock_ma):
    globals()[x] = pd.read_csv(f'data//{y}_15_01_2023.csv', encoding='utf-
8')
    globals()[x]['Date'] = pd.to_datetime(globals()[x]['Date'])
    globals()[x]['Volume'] = globals()[x]['Volume'].astype(int)
    globals()[z] = globals()[x]
    globals()[z] =  globals()[z].set_index('Date')
    com_list.append(globals()[z])

# Об'єднуємо дані в один DataFrame
df = pd.concat(com_list, axis=0)

# Виводимо перші рядки даних
bbca.head()

# Виводимо інформацію про DataFrame
bbca.info()

# Побудова графіків для аналізу даних
chart = f'Відкриття Ціни {stock_code[0]}, 2004-2024'

Markdown(f"""
### {chart}
""")

fig, ax = plt.subplots(figsize=(14, 8))
fig.suptitle(f'{chart}', fontweight='bold')

# Побудова лінійного графіка на основі значень 'Open' ціни
for x, y in zip(stock_list, stock_code):
    ax.plot('Date', 'Open', data=globals()[x], label=y)

ax.set_ylabel('Відкриття Ціни')
ax.tick_params(axis='x', rotation=45)
ax.grid(False)
ax.legend()

plt.show()

# Побудова графіка для ціни закриття
chart = f'Ціна Закриття {stock_code[0]}, 2004-2024'

Markdown(f"""
### {chart}
""")
```

```
fig, ax = plt.subplots(figsize=(14, 8))
fig.suptitle(f'{chart}', fontweight='bold')

for x, y in zip(stock_list, stock_code):
    ax.plot('Date', 'Close', data=globals()[x], label=y)

ax.set_ylabel('Ціна Закриття')
ax.tick_params(axis='x', rotation=45)
ax.grid(False)
ax.legend()

plt.show()

# Побудова графіка для об'єму торгів
chart = f'Об\'єм Торгів {stock_code[0]}, 2004-2024'

Markdown(f"""
### {chart}
""")

fig, ax = plt.subplots(figsize=(14, 8))
fig.suptitle(f'{chart}', fontweight='bold')

for x, y in zip(stock_list, stock_code):
    ax.bar(globals()[x]['Date'], globals()[x]['Volume'], label=y)

ax.set_ylabel('Об\'єм Торгів')
ax.tick_params(axis='x', rotation=45)
ax.grid(False)
ax.legend()

plt.show()

# Побудова прогнозування цін за допомогою LSTM
chart = f'{stock_code[0]} Прогнозування'

Markdown(f"""
### {chart}
""")

# Створення нового DataFrame з колонкою 'Open'
data = bbca_ma.filter(['Open'])

# Перетворення DataFrame у numpy масив
dataset = data.values

# Розділення даних на тренувальні та тестові
training_data_len = int(np.ceil(len(dataset) * .95))

# Масштабування даних
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)

# Створення тренувального набору даних
train_data = scaled_data[0:int(training_data_len), :]
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i <= 61:
        print(x_train)
        print(y_train)
        print()

# Перетворення даних у numpy масиви та зміна їх форми
x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

# Побудова LSTM моделі
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(x_train.shape[1],
1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Компіляція та навчання моделі
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size=1, epochs=1)

# Створення тестового набору даних
test_data = scaled_data[training_data_len - 60:, :]
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

# Прогнозування цін на основі тестових даних
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Обчислення кореня середньоквадратичної помилки (RMSE)
rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
print(f'RMSE: {rmse}')

# Візуалізація результатів прогнозування
```

```
train = data[:training_data_len]
valid = data[training_data_len:]
valid = valid.assign(Predictions=predictions)

chart = f'{stock_code[0]} Результати Прогнозування'

Markdown(f"""
### {chart}
""")

plt.figure(figsize=(10, 4))
plt.suptitle(f'{chart}', fontweight='bold')

plt.plot(train[['Open']])
plt.plot(valid[['Open', 'Predictions']], linewidth=2)
plt.legend(['Train', 'Val', 'Прогноз'], loc='lower right')

plt.show()

# Аналіз результатів: оцінка точності прогнозування
Markdown("""
### Аналіз результатів: оцінка точності прогнозування
""")


# Оцінка точності прогнозування для тренувальних даних
# Використовуємо лише ті значення, які можна порівняти за розміром

train_predict = model.predict(x_train)
train_predict = scaler.inverse_transform(train_predict)

# Скорочуємо розмір 'train' до розміру передбачених значень
train_compare = train['Open'][-len(train_predict):]

# Перевірка розмірів масивів перед обчисленням RMSE
print(f"Розмір масиву 'train_compare': {len(train_compare)}")
print(f"Розмір масиву 'train_predict': {len(train_predict)}")

train_rmse = np.sqrt(np.mean((train_compare.values -
train_predict.flatten()) ** 2))
test_rmse = rmse

Markdown(f"""
- Корінь середньоквадратичної помилки (RMSE) для тренувальних даних:
{train_rmse:.4f}\n
- Корінь середньоквадратичної помилки (RMSE) для тестових даних:
{test_rmse:.4f}\n
- Різниця між тренувальним та тестовим RMSE може свідчити про перенавчання
моделі або недооцінку майбутніх трендів.\n
""")
```