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

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Використання машинного навчання для створення алгоритмів передбачення споживчого попиту і оптимізації запасів в роздрібних мережах»

Виконала:
студентка IV курсу, групи КІ-02
Ткаченко Дарина Володимирівна _____

Керівник:
доцент кафедри штучного інтелекту,
к.т.н., доцент
Жиров Олександр Леонідович _____

Консультант з економічного розділу:
професор кафедри економічної кібернетики ФММ,
д.е.н., професор
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри штучного інтелекту,
Кравець Павло Володимирович _____

Рецензент:
директор НН ФТІ, д.т.н,
професор
Новиков Олексій Миколайович
_____

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

Студентка _____

Київ – 2024 року

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

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ткаченко Дарині Володимирівні

1. Тема роботи «Використання машинного навчання для створення алгоритмів передбачення споживчого попиту і оптимізації запасів в роздрібних мережах», керівник роботи Жиров Олександр Леонідович, доцент кафедри ШІ, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи «10» червня 2024 року.

3. Вихідні дані до роботи: змодельовані дані споживчого попиту.

4. Зміст роботи: аналіз предметної області, проектування системи, розробка і тестування системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): діаграми, скріншоти виводу результатів та інтерфейсу програми, графіки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук Олена Анатоліївна, професор, д. е. н.		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми дипломної роботи (ДР)	15.04.2024 - 21.04.2024	Виконано
2	Ознайомлення зі структурою ДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ»	22.04.2024 - 28.04.2024	Виконано
3	Ознайомлення з ДСТУ 3008:2015 та стандартами ЄСПД	29.04.2024 - 03.05.2024	Виконано
4	Проведення дослідження за темою ДР	04.05.2024 - 12.05.2024	Виконано
5	Проведення роботи над теоретичною частиною ДР	12.05.2024 - 22.05.2024	Виконано
6	Проведення роботи над програмним продуктом	22.05.2024 - 03.06.2024	Виконано
7	Оформлення ДР та аналіз отриманих результатів	04.06.2024 – 10.06.2024	Виконано

Студент Дарина ТКАЧЕНКО

Керівник Олександр ЖИРОВ

РЕФЕРАТ

Дипломна робота: 81 с., 11 рис., 6 табл., 40 посилань, 1 додаток.

ШТУЧНИЙ ІНТЕЛЕКТ, НЕЙРОННА МЕРЕЖА, ЕЛЕКТРОННА КОМЕРЦІЯ, МОДЕЛЬ, СПОЖИВЧИЙ ПОПИТ, РОЗРОБКА, ПРОГНОЗУВАННЯ.

Об'єктом дослідження є процеси прогнозування споживчого попиту на основі аналізу великих даних за допомогою нейронних мереж.

Предметом дослідження є архітектури нейронних мереж та методи їх навчання, які застосовуються для аналізу та прогнозування споживчого попиту.

Мета роботи полягає у розробці та дослідженні нейронної мережі для точного та надійного прогнозування споживчого попиту, що сприятиме покращенню аналітичних можливостей у сфері електронної комерції.

В роботі розроблено та досліджено нейронну мережу для прогнозування споживчого попиту. Встановлено, що нейронна мережа має високу точність прогнозування, є надійним інструментом в електронній комерції для прийняття обґрунтованих рішень щодо ціноутворення. Проведено аналіз різних архітектур нейронних мереж, який показав глибокі нейронні мережі з конволюційними та рекурентними шарами, які найбільш ефективні для завдань прогнозування у динамічно змінюваних умовах ринку.

ABSTRACT

Master's thesis: 81 p., 11 figures, 6 tables, 40 references, 1 appendix.

ARTIFICIAL INTELLIGENCE, NEURAL NETWORK, E-COMMERCE, MODEL, CONSUMER DEMAND, DEVELOPMENT, FORECASTING.

The object of the study is the process of forecasting consumer demand based on big data analysis using neural networks.

The subject of research is the architecture of neural networks and the methods of their training used to analyze and forecast consumer demand.

The purpose of the work is to develop and study a neural network for accurate and reliable forecasting of consumer demand, which will improve analytical capabilities in the field of e-commerce.

The work developed and tested a neural network for forecasting consumer demand. It has been established that the neural network has high accuracy in forecasting and is a reliable tool in electronic commerce for making sound decisions on price formation. An analysis of various neural network architectures was conducted, which showed that deep neural networks with convolutional and recurrent layers are the most effective for predicting tasks in dynamically changing market conditions.

ЗМІСТ

ВСТУП

Сучасний ринок наповнений незліченною кількістю різноманітних продуктів. Кожна велика компанія користується послугами спеціалістів з логістики, які планують маршрути та час доставки продукції на місця її видачі. Для цього потрібно приблизно розуміти обсяги збуту, щоб правильно регулювати об’єм товару по всій мережі крамниць. З цією задачею передбачення споживчого попиту гарно справляються рекурентні нейронні мережі, ця робота детально розглядає саме цей спосіб їх використання.

Актуальність даного дослідження обумовлена зростаючим попитом на точні та надійні прогностичні моделі, які стають частиною інструментарію для прийняття логістичних рішень, зокрема надаючи інформацію про приблизний споживчий попит. Нейронні мережі пропонують інноваційні підходи до аналізу великих масивів даних, що дозволяє виявляти приховані закономірності та тенденції.

Практичне значення роботи полягає у розробці ефективної нейронної мережі, яка може бути використана для прогнозування споживчого попиту. Це дозволить підприємствам більш точно планувати свої стратегії, оптимізувати запаси та покращувати обслуговування клієнтів, що, в свою чергу, сприятиме зростанню їх конкурентоспроможності на ринку.

Дана робота важлива для подальшого розвитку методів прогнозування у сфері електронної комерції, а також для вдосконалення аналітичних інструментів на основі штучного інтелекту.

РОЗДІЛ 1 АНАЛІЗ ОСНОВНИХ ТЕОРЕТИЧНИХ ПИТАНЬ

1.1 Поняття нейронних мереж

Нейронні мережі є однією з основних технологій у сфері штучного інтелекту, які здобули значну популярність завдяки їх здатності вирішувати складні задачі, такі як розпізнавання образів, обробка природної мови і прогнозування. Розглянемо історію розвитку нейронних мереж, різні парадигми навчання, основні види нейронних мереж, алгоритми їх роботи та практичне використання.

Історія нейронних мереж починається з середини XX століття, коли вчені почали досліджувати можливості створення обчислювальних систем, що імітують роботу людського мозку.

У 1943 році Уоррен Маккаллох і Волтер Піттс запропонували першу математичну модель нейрона. Вони показали, що за допомогою мережі таких нейронів можна реалізувати будь-яку булеву функцію. Ця робота стала основою для подальших досліджень у галузі нейронних мереж.

У 1958 році Френк Розенблатт запропонував модель перцептрона, яка стала першим практичним застосуванням нейронних мереж. Перцептрон був здатний навчатися за допомогою алгоритму зворотного розповсюдження помилки, що дозволяло йому вирішувати задачі класифікації. Однак, перцептрон мав обмеження, зокрема не міг вирішувати задачі, які не є лінійно роздільними.

У 1980-х роках дослідники почали розробляти багатошарові нейронні мережі, які складалися з декількох шарів нейронів. Це дозволило вирішувати більш складні задачі, такі як розпізнавання образів і мова. Однією з ключових подій стало відкриття алгоритму зворотного поширення помилки (backpropagation) у 1986 році, який дозволив ефективно навчати багатошарові мережі.

У 2000-х роках розвиток обчислювальних потужностей і доступ до великих обсягів даних привели до нового сплеску інтересу до нейронних мереж. Глибокі нейронні мережі (deep neural networks) стали використовуватися для розв'язання задач, які раніше вважалися недосяжними, таких як розпізнавання мови і зображень на рівні, що перевищує людські можливості. Революція в глибокому навчанні почалася з роботи Джеффрі Гінтона та його колег, які показали ефективність глибоких нейронних мереж на практичних задачах.

Існує кілька основних парадигм навчання нейронних мереж: контрольоване навчання, неконтрольоване навчання і навчання з підкріпленням.

Контрольоване навчання є найпоширенішою парадигмою навчання нейронних мереж. У цьому підході мережа навчається на основі навчальних даних, які містять вхідні дані та відповідні мітки. Мета полягає в тому, щоб навчити мережу правильно передбачати мітки для нових, невідомих даних.

Прикладом контрольованого навчання є розпізнавання рукописних цифр. Вхідні дані – зображення цифр, а мітки – відповідні цифри. Мережа навчається розпізнавати цифри на основі цих даних.

Неконтрольоване навчання не вимагає наявності міток у навчальних даних. Мета його полягає в тому, щоб виявити приховані структури або патерни в даних. Один з поширених методів неконтрольованого навчання – кластеризація, де мережа навчається групувати схожі дані разом.

Прикладом неконтрольованого навчання є аналіз клієнтських даних для виявлення груп клієнтів з подібною поведінкою. Мережа аналізує дані про покупки і групує клієнтів за схожістю.

Навчання з підкріпленням (reinforcement learning) – це підхід, у якому агент навчається взаємодіяти зі середовищем з метою максимізації нагороди. Агент виконує дії і отримує зворотній зв'язок у вигляді нагороди або штрафу, на основі чого він коригує свою поведінку.

Прикладом навчання з підкріпленням є навчання робота ходити. Робот отримує позитивні нагороди за правильні кроки і штрафи за падіння. З часом робот вчиться ходити стабільно.

Існує багато різних видів нейронних мереж, кожна з яких має свої особливості і застосування.

1. Перцептрон складається з одного шару нейронів, де кожен нейрон приймає кілька вхідних сигналів і виробляє вихідний сигнал на основі лінійної комбінації входів і функції активації. Цей вид нейронної мережі може вирішувати тільки лінійно роздільні задачі.
2. Багатошаровий перцептрон (MLP) складається з кількох шарів нейронів і вихідного шару. MLP може вирішувати нелінійні задачі завдяки використанню нелінійних функцій активації, таких як сигмоїдна або ReLU (Rectified Linear Unit).
3. Рекурентні нейронні мережі (RNN) найчастіше використовуються для роботи з текстом або часовими рядами. Особливістю RNN є те, що вони мають петлі зворотнього зв'язку, які дозволяють враховувати попередні стани для передбачення наступних. Це робить RNN ефективними для задач, де контекст є важливим, таких як переклад текстів або прогнозування рядів даних.
4. LSTM є особливим видом RNN, який вирішує проблему зникнення градієнта під час тривалого навчання. LSTM використовують спеціальні комірки пам'яті, які можуть зберігати інформацію протягом тривалих періодів часу, що робить їх ефективними для задач, де важливий довготривалий контекст.
5. Конволюційні нейронні мережі (CNN) широко використовуються для обробки зображень і відео. Вони складаються з шарів конволюції, які виділяють локальні особливості зображення, і шарів підсемплювання, які зменшують розміри зображення. CNN

ефективно витягують просторові ієрархічні особливості, що робить їх ідеальними для задач розпізнавання образів.

6. Автокодувальники (Autoencoders) – це нейронні мережі, які навчаються копіювати вхідні дані на вихід, при цьому стискаючи інформацію у прихованих шарах. Вони використовуються для задач зниження вимірності, виявлення аномалій і генерації нових даних. Автокодувальники складаються з двох основних частин: енкодера, який стискає вхідні дані, і декодера, який відновлює їх з прихованого представлення.
7. Генеративно-змагальні мережі (GAN) складаються з двох мереж: генератора і дискримінатора. Генератор створює нові дані, схожі на вхідні, а дискримінатор оцінює, наскільки ці дані є реальними. GAN використовуються для створення різних видів даних, і вони показали вражаючі результати у створенні високоякісних зображень.

Навчання нейронних мереж є складним процесом, який включає налаштування вагових коефіцієнтів на основі вхідних даних і функції втрат. Основні алгоритми навчання нейронних мереж включають:

Градієнтний спуск є основним алгоритмом оптимізації, який використовується для навчання нейронних мереж. Ідея полягає в тому, щоб мінімізувати функцію втрат шляхом оновлення вагових коефіцієнтів у напрямку негативного градієнта. Основна формула для оновлення ваг виглядає так:

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i}, \tag{1.1}$$

де w_i – ваговий коефіцієнт;

α – швидкість навчання;

а $\frac{\partial L}{\partial w_i}$ – градієнт функції втрат щодо ваги.

Backpropagation є алгоритмом, який використовується для обчислення градієнтів у багатошарових нейронних мережах. Він заснований на методі ланцюгового правила і дозволяє ефективно оновлювати ваги у всіх шарах мережі. Алгоритм складається з двох етапів: прямого проходу, під час якого обчислюється вихід мережі, і зворотного проходу, під час якого обчислюються градієнти і оновлюються ваги.

Стохастичний градієнтний спуск (SGD) є варіантом градієнтного спуску, який використовує випадкові підвибірки даних для оновлення ваг. Це дозволяє пришвидшити процес навчання і зменшити обчислювальні витрати. Основна формула для оновлення ваг у SGD виглядає так:

$$w_i = w_i - \alpha \frac{\partial L(x_i, y_i)}{\partial w_i}, \tag{1.2}$$

де $L(x_i,\ y_i)$ – функція втрат для і-тої випадкової вибірки.

Існує кілька адаптивних алгоритмів навчання, які автоматично налаштовують швидкість навчання на основі величини градієнта.

1. Adagrad: алгоритм, який використовує змінні швидкості навчання нейронної мережі на основі попередніх градієнтів.
2. RMSprop: алгоритм, який використовує експоненційне згладжування для обчислення середньоквадратичного градієнта.
3. Adam: комбінація Adagrad і RMSprop, яка використовує змінні швидкості навчання і моментум для прискорення процесу навчання.

Нейронні мережі мають широке застосування в різних галузях, включаючи комп'ютерне бачення, обробку природної мови, медицину, фінанси та інші.

1. Нейронні мережі, зокрема CNN, широко використовуються в комп'ютерному баченні для задач розпізнавання об'єктів, класифікації зображень, сегментації і детекції об'єктів.

Наприклад, CNN використовуються в системах розпізнавання облич, автоматичному водінні автомобілів і діагностиці медичних зображень.

2. Нейронні мережі, такі як RNN і трансформери, використовуються для обробки текстів і мови. Вони застосовуються для задач перекладу текстів, розпізнавання мови, аналізу настроїв, створення чат-ботів і генерації текстів. Наприклад, модель GPT-3, розроблена OpenAI, використовує трансформери для генерації високоякісних текстів на основі вхідних даних.

Нейронні мережі знаходять застосування в медицині. Наприклад, CNN використовуються для автоматичного виявлення ракових утворень на рентгенівських знімках, а RNN – для аналізу медичних записів і прогнозування розвитку захворювань.

У фінансовій галузі нейронні мережі використовуються для прогнозування ринкових тенденцій, аналізу ризиків, виявлення шахрайства і автоматичної торгівлі. Наприклад, RNN і LSTM використовуються для аналізу часових рядів фінансових даних і прогнозування цін акцій, а автокодувальники – для виявлення аномалій у транзакціях.

Нейронні мережі використовуються в індустрії розваг для створення реалістичних персонажів, генерації музики, автоматичного перекладу субтитрів і рекомендацій контенту. Наприклад, GAN використовуються для створення реалістичних зображень і відео, а RNN – для генерації музичних композицій і текстів пісень.

Нейронні мережі є потужним інструментом, від комп'ютерного бачення і обробки природної мови до медицини і фінансових технологій, вони продовжують змінювати світ і відкривати нові можливості для досліджень і інновацій.

Історія розвитку нейронних мереж демонструє, як технології можуть еволюціонувати і вдосконалюватися з часом, завдяки постійним дослідженням і досягненням у галузі обчислювальної техніки. З подальшим

розвитком обчислювальних потужностей і доступом до великих обсягів даних, нейронні мережі будуть продовжувати грати ключову роль у розвитку штучного інтелекту і впливати на наше повсякденне життя.

1.2 Прогнозування числових рядів

У вузькому сенсі це спеціальне наукове дослідження певних перспектив подальшого розвитку будь-якого процесу.

Необхідність прогнозування обумовлена бажанням знати майбутні події, які в принципі достовірно неможливі, виходячи з принципів статистики (похибка поточних оцінок), схоластики (багатовимірні результати), емпіричного (методологічна похибка моделі) і філософського (межа поточних знань).

Точність прогнозу визначається наступним чином:

- «достовірний» обсяг вихідних даних (перевірених з відомими помилками) і тривалість їх збору;
- обсяг неперевірених вихідних даних і тривалість їх збору;
- система взаємодії властивостей прогнозованого об'єкта з прогнозованим об'єктом передбачуваності;
- методи і моделі прогнозування.

Зі зростанням набору факторів, що впливають на точність прогнозу, він фактично замінюється рутинними розрахунками з певними помилками.

Прогноз розділений (умовний) поділяється:

- за термінами: короткострокові, середньострокові, довгострокові, наддовгострокові;
- за масштабом: приватні, місцеві, регіональні, промислові, національні, глобальні (global);

- за відповідальністю (автора): на індивідуальному, корпоративному (organization) рівні, на рівні державної установи.

В даний час широко використовується комплексний метод прогнозування, який називається Форсайт (вивчення майбутнього), особливо в Європі. Форсайт – це експертна оцінка стратегічного напрямку соціально-економічного та інноваційного розвитку, система методів виявлення нових технологій, які вплинуть на світ в середньостроковій і довгостроковій перспективі. За допомогою цього методу створюються різні можливі картини майбутнього світу, включаючи їх індивідуальні, найбільш важливі або цікаві значущі аспекти, сформовані в результаті різних сценаріїв розвитку ситуації і планованого сценарію власних дій [3].

Фахівці використовують прогнозування в економіко-соціальній, науково-технічній, територіальній та інших сферах для різних термінів планування. Форсайт же використовується для стратегічного планування розвитку країн або при конструюванні великих систем державного регулювання та управління.

Статистичні методи прогнозування – це наукова і освітня область, основними завданнями якої є розробка, вивчення і застосування сучасних математичних і статистичних методів прогнозування, заснованих на об'єктивних даних. Розвиток теорії і практики стохастичного і статистичного моделювання методів прогнозування фахівців.

Методи прогнозування в поєднанні з методами прогнозування умов ризику, які спільно використовують економічні, математичні та економетричні (як математико-статистичні, так і професійні) моделі. Науковою основою методів статистичного прогнозування є прикладна статистика і теорія прийняття рішень.

Простий метод відновлення залежностей для прогнозування базується на часових рядах, які визначаються відомою кількістю точок на осі часу. В аналізі таких даних використовуються імовірнісні моделі, де до часових параметрів додаються інші змінні, такі як обсяги грошової маси. Основним

завданням є аналіз багатовимірності, інтерполяції та екстраполяції. Метод найменших квадратів, який часто застосовується для визначення цих змінних, був розроблений з 1794 по 1795 рік і часто використовує прості лінійні функції з коефіцієнтом 1. Варто зауважити, що трансформації змінних, як-от логарифмування, можуть бути корисні. Хоча інші методи екстраполяції, такі як сплайни, застосовуються рідше, вони зазвичай надають кращі статистичні характеристики.

Оцінка точності прогнозу (особливо з використанням довірчих даних) є необхідною частиною процедури прогнозування. Імовірнісні статистичні моделі залежного відновлення часто використовуються, наприклад, для побудови найкращих прогнозів за допомогою методу максимальної ймовірності. Були розроблені параметричні (зазвичай засновані на моделі нормальної похибки) та непараметричні оцінки точності прогнозування та довірчих меж (засновані на центральній граничній теоремі теорії ймовірностей). Також, використовуються евристики, не засновані на теорії ймовірнісної статистики, такі як метод ковзного середнього. Для статистичного прогнозування найчастіше використовують багатовимірну регресію.

Не потрібно використовувати нереалістичні припущення щодо нормальності помилок вимірювань та відхилень від лінії регресії (поверхні), але щоб відмовитися від припущень про нормальність, необхідно покладатися на інший математичний інструмент, заснований на теорії ймовірностей, методах лінеаризації та багатовимірній центральній граничній теоремі про спадковість конвергенції [4]. Це дозволяє оцінювати точки та інтервали параметрів, перевіряти значимість відмінностей від 0 у непараметричних налаштуваннях та будувати довірчі межі для прогнозів.

Проблема вибору та перевірки факторів, які впливають на модель, є критичною в статистичному моделюванні. Зазвичай початковий список потенційних факторів великий, але важливо звузити його до набору найбільш значущих характеристик. Багато сучасних досліджень фокусуються на

методах вибору оптимальних функцій, хоча ця проблема все ще не має остаточного рішення, іноді виявляються несподівані аномалії. Щодо статистичних методів, розвиток непараметричних підходів для оцінки щільності розподілу ймовірностей виявляється перспективним, особливо у застосуванні до відновлення регресійних залежностей. Ці методи дозволяють ефективно аналізувати нечислові дані, що розширює можливості прогнозування і інтерпретації статистичних моделей.

Сучасні методи статистичного прогнозування також включають моделі експоненціального згладжування, авторегресії зі ковзними середніми та системи економетричних рівнянь, засновані як на параметричному, так і на непараметричному підходах.

Комп'ютерні статистичні методи можуть допомогти визначити чи можна застосовувати асимптотичні результати до кінцевих (так званих «малих») розмірів вибірки. Також, можна створювати різні імітаційні моделі. Потрібно звернути увагу на корисність методу доставки даних (метод завантаження). Комп'ютеризована система прогнозування об'єднує різні методи прогнозування на робочому місці одного автоматизованого оцінювача.

Основними етапами обробки прогнозів для фахівців з прогнозування є перевірки узгодженості, кластерний аналіз і пошук групових думок. Узгодженість думок експертів, представлених в ході оцінки, була перевірена з використанням рангових коефіцієнтів кореляції Кендалла і Спірмена і рангових коефіцієнтів збігу Кендалла і Бабінгтонсміта. Використовуються параметричні моделі порівняння пар-Терстоун, Бредлі-Террі-Льюїс-і непараметричні моделі теорії Люціана [1, 3]. Корисна процедура коригування оцінки та класифікації шляхом встановлення відповідних біноміальних співвідношень. Якщо немає узгодженості, використовується метод найближчого сусіда або інші методи кластерного аналізу (автоматична класифікація, неконтрольоване розпізнавання зображень), щоб розділити

думки експертів на подібні групи. Класифікація Люціана базується на імовірнісній статистичній моделі.

Для складання остаточного висновку експертної комісії використовуються різні методи. Метод середнього арифметичного і медіанного рангу відрізняється своєю простотою. Комп'ютерне моделювання [3] дозволило встановити багато характеристик медіани Кемені, які рекомендується використовувати як остаточний (узагальнений, середній) висновок експертної комісії.

Існує чимало ситуацій, пов’язаних з соціальними, технічними, економічними та екологічними ризиками, де необхідно здійснити оцінку. У таких випадках важливо провести кваліфіковану оцінку. Використовуються різні види критеріїв, що застосовуються при невизначеності (ризику) для прийняття рішень. Через непослідовність рішень, прийнятих на основі різних критеріїв, виникає потреба у використанні експертних оцінок [2].

У конкретних завданнях прогнозування можна класифікувати ризики, налаштовувати завдання для оцінки конкретних ризиків і виконувати конфігурацію ризиків, зокрема, дерево причин (тобто, дерево невдач) і дерево наслідків (тобто, центральним завданням є, наприклад, створення груп і узагальнених показників, показники конкурентоспроможності та якості).

Ризики повинні враховуватися при прогнозуванні економічного впливу прийнятих рішень, споживчої поведінки і конкурентного середовища, економічного і макроекономічного розвитку зарубіжних країн, екологічного стану навколишнього середовища, технологічної безпеки, екологічної небезпеки промислових та інших об'єктів.

Сучасні методи комп'ютерного прогнозування включають бази даних економетричних даних, моделювання (в тому числі засноване на використанні методів статистичного тестування) і використання експертів, математики, статистики і моделей.

Комп'ютерні програми часто застосовують для прогнозування часових рядів та дозволяють автоматизувати більшість процесів у створенні прогнозів і мінімізувати помилки, пов'язані з введенням даних і побудовою моделей. Такі програми можуть функціонувати як на локальному комп'ютері, так і мати інтернет-додатки (наприклад, веб-сайти), вони часто використовують R, SPSS, Statistica та інші технології.

Прогнозуюча здатність нейронних мереж безпосередньо пов'язана з їх здатністю узагальнювати та ідентифікувати приховані залежності між вхідними та вихідними даними. Навчена мережа може прогнозувати майбутні значення для даного набору на основі деяких попередніх значень та (або) деяких факторів, які існують на даний момент, але прогноз можливий лише в тому випадку, якщо попередні зміни насправді певною мірою визначають майбутнє.

Можна розглянути основні проблеми прогнозування за допомогою нейронних мереж, це такі як:

- вибір правильного алгоритму обробки даних;
- створення правильного набору даних для навчання та тестування.

Проблема вибору правильного алгоритму обробки даних полягає в наявності великої кількості різних алгоритмів штучного інтелекту, які виконують аналіз по-своєму. Щоб вибрати правильний алгоритм, необхідно провести детальний аналіз інформації про роботу алгоритмів штучного інтелекту для обробки даних.

Навчання нейронних мереж вимагає багато даних, навчання займає багато часу і вимагає великої кількості навчальних зразків, тому створення правильного набору даних є простим завданням збору даних.

1.3 Аналіз існуючих рішень

R – це мова програмування, яку використовують для статистичних обчислень та графіків, що підтримується основною командою r та Фондом статистичних обчислень R [7], який широко використовується статистиками та розробниками даних для аналізу даних.

Офіційне програмне середовище R – це пакет GNU. Він написаний переважно на C, Fortran та R (частково розміщений окремо). Він доступний безкоштовно під загальною публічною ліцензією GNU, що дозволяє компілювати виконувані файли для різних операційних систем. Ця мова забезпечує інтерфейс командного рядка, але також існують різні зовнішні графічні інтерфейси, такі як RStudio та Jupyter, які інтегрують робочі середовища і підтримують візуалізацію даних.

R і його бібліотеки містять безліч статистичних засобів: лінійне і нелінійне моделювання, тести, просторовий і часовий аналіз, класифікацію, кластеризацію тощо. Стандартні функції r написані в самому R, тому користувачі можуть легко слідувати обраному алгоритму.

Для цілей, що вимагають великих обчислень, код на C, C++ та Fortran може бути пов'язаний та викликаний під час виконання. Досвідчені користувачі можуть безпосередньо керувати об'єктами R, написавши код на C, C++, Java, Net або Python.

R має високий ступінь розширення завдяки використанню користувацьких пакетів для певних функцій та конкретних областей досліджень. Завдяки своїй спадщині, R володіє більш потужними інструментами об'єктно-орієнтованого програмування, ніж більшість мов статистичних обчислень. Його розширенню також сприяють правила лексичного визначення предметної області.

Statistica – це набір програмних продуктів і аналітичних рішень, спочатку розроблений компанією StatSoft і придбаний Dell в березні 2014

році. Програмне забезпечення доступне завдяки інтеграції з вільним середовищем програмування R з відкритим вихідним кодом [2]. Функціонал для аналізу даних, управління даними, процедури візуалізації та різноманітні інструменти для аналізу, прогнозування, кластеризації, класифікації – все це доступно в 6 лінійках продуктів [3 – 5].

Програмне забезпечення Statistica включає інструменти для створення аналітичних і дослідницьких графіків, як доповнення до стандартних 2D та 3D графіків. Воно надає можливості для інтерактивного маркування, анотування, та маніпулювання даними для більш глибокого аналізу.

Операції з програмним забезпеченням зазвичай включають завантаження таблиць даних і застосування статистичних функцій з меню, що випадає або з стрічки (у версії 9.0 і пізніших версіях). Потім, у Меню вам буде запропоновано вказати необхідні змінні та тип аналізу. Вам не потрібно вводити дані з командного рядка. Кожен аналіз може містити графічні або табличні результати і зберігається в окремій робочій книзі.

1.4 Постановка задачі

Основна задача даної роботи полягає в використанні машинного навчання, щоб передбачувати споживчий попит. Метою є розробка моделі, яка приймає історичні дані як вхідну інформацію та на їх основі робить передбачення. Дослідження передбачає аналіз вже існуючих методів та способів прогнозування, на основі цих даних буде створений відповідний програмний продукт. Результатом роботи буде аналіз роботи вибраного алгоритму для даної задачі.

Висновки до розділу 1

У розділі 1 було проведено детальний аналіз основних теоретичних аспектів, пов'язаних з нейронними мережами, прогнозуванням числових рядів та аналізом існуючих рішень у цій галузі. Спочатку розглянуто поняття нейронних мереж, їхню структуру та принципи функціонування. Було визначено, що нейронні мережі складаються з нейронів, організованих у шари, де кожен нейрон здійснює обчислення на основі вхідних сигналів та вагових коефіцієнтів. Під час аналізу різних типів нейронних мереж, таких як багатошарові персептрони, згорткові та рекурентні нейронні мережі, виявлено, що вони можуть ефективно вирішувати широкий спектр завдань у різних галузях, таких як обробка зображень, розпізнавання мови та прогнозування числових даних.

Далі розглянуто питання прогнозування числових рядів. Це завдання часто зустрічається в економіці, фінансах, метеорології та інших сферах, де необхідно передбачити майбутні значення на основі історичних даних. Було проаналізовано основні методи прогнозування, включаючи лінійну регресію, методи автокореляції та сучасні методи, такі як рекурентні нейронні мережі та довготривалу короткочасну пам'ять (LSTM). Виявлено, що останні методи мають високу точність та здатні враховувати складні нелінійні залежності у даних.

Нарешті, проведено аналіз існуючих рішень у галузі нейронних мереж та прогнозування числових рядів. Було розглянуто декілька комерційних та наукових проектів, що використовують ці технології. Виявлено, що сучасні нейронні мережі можуть бути успішно застосовані для вирішення реальних проблем, що підтверджується їхньою ефективністю та точністю в різних практичних застосуваннях. Таким чином, теоретичні основи, розглянуті у цьому розділі, закладають фундамент для подальшого практичного дослідження та впровадження нейронних мереж у різні галузі.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ПРОГНОЗУВАННЯ

2.1 Життєвий цикл розробки

Модель водоспаду – це давня методологія у сфері управління проектами та розробки програмного забезпечення, відома своїм лінійним та послідовним підходом. Представлена доктором Вінстоном Ройсом у 1970 році, ця модель швидко набула популярності завдяки своїй чіткій структурі та методичному проходженню через окремі етапи. Незважаючи на свій вік, модель водоспаду продовжує залишатися актуальною, особливо в проектах, де вимоги добре зрозумілі з самого початку і навряд чи зміняться.

Модель водоспаду поділяється на шість основних етапів: Аналіз вимог, проектування системи, реалізація, інтеграція та тестування, розгортання та обслуговування. Кожна фаза повинна бути завершена до початку наступної, що забезпечує систематичне просування через життєвий цикл проекту.

Аналіз вимог. Цей початковий етап має вирішальне значення, оскільки він закладає основу для всього проекту. Під час аналізу вимог менеджери та аналітики проекту збирають детальну інформацію від зацікавлених сторін, щоб зрозуміти їхні потреби та очікування. Ця інформація ретельно документується в документі специфікації вимог. Мета полягає в тому, щоб зафіксувати всі функціональні та нефункціональні вимоги, не залишаючи жодних двозначностей. Цей документ слугує відправною точкою впродовж усього проекту, керуючи всіма наступними етапами.

Проектування системи. На етапі проектування системи зібрані вимоги перетворюються на план системи. Цей етап зазвичай поділяється на два підетапи: високорівневе проектування (HLD) і низькорівневе проектування (LLD). HLD фокусується на загальній архітектурі системи, визначаючи основні компоненти та їх взаємодію. LLD заглиблюється в деталі кожного компонента, визначаючи алгоритми, структури даних та інтерфейси.

Результатом цього етапу є проектна документація, якою керується команда розробників під час фази впровадження.

Реалізація. На етапі реалізації відбувається розробка коду програмного продукту. Розробники працюють над окремими компонентами відповідно до проектної документації. Ця фаза часто є найбільш трудомісткою та ресурсоємною. Кожен компонент розробляється окремо і ретельно тестується, щоб переконатися, що він відповідає заданим вимогам. Основна увага приділяється написанню чистого, ефективного та безпомилкового коду.

Інтеграція та тестування. Після того, як всі компоненти розроблені, їх інтегрують, щоб сформувати цілісну систему. Ця інтегрована система проходить різні рівні тестування: модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування. Основна мета - виявити і виправити будь-які дефекти, гарантуючи, що система функціонує правильно і відповідає всім вимогам. Тестування на цьому етапі є критично важливим, оскільки це перша можливість побачити, як вся система працює разом.

Розгортання. Після успішного тестування система готова до розгортання. Цей етап передбачає встановлення системи у виробничому середовищі, налаштування її для використання, а також надання необхідного навчання та документації кінцевим користувачам. Розгортання може бути складним процесом, особливо для великих систем, і може проводитися поетапно, щоб мінімізувати ризики.

Обслуговування. Заключним етапом моделі водоспаду є технічне обслуговування, яке починається після запуску системи в експлуатацію. Обслуговування передбачає вирішення будь-яких проблем, що виникають, впровадження оновлень і забезпечення того, щоб система продовжувала відповідати потребам користувачів. Цей етап може тривати роками і охоплювати такі види діяльності, як виправлення помилок, підвищення продуктивності та додавання нових функцій.

Однією з головних переваг моделі водоспаду є її простота і легкість у використанні. Лінійний і послідовний характер моделі робить її простою для розуміння і управління. Кожна фаза має конкретні результати і процес перевірки, що забезпечує чітку структуру і організацію. Ця чіткість гарантує, що всі зацікавлені сторони, включаючи розробників, менеджерів і клієнтів, мають спільне розуміння прогресу і цілей проекту.

Ще однією значною перевагою є чітко визначені етапи, які допомагають відстежувати прогрес і зберігати фокус. Кожна фаза має чіткі цілі та результати, що полегшує відстеження та управління проектом. Така структура особливо корисна для великих проектів зі складними вимогами, де детальне планування і документація мають важливе значення.

Вичерпна документація – ще одна сильна сторона моделі «Водоспад». На кожному етапі створюється велика кількість документації, що створює чіткий слід розвитку проекту. Ця документація є безцінною для подальшого використання, сприяє кращій комунікації між зацікавленими сторонами і забезпечує безперервність, навіть якщо члени команди змінюються.

Незважаючи на свої переваги, модель водоспаду має суттєві недоліки, насамперед негнучкість. Модель передбачає, що всі вимоги можуть бути зібрані заздалегідь і залишатимуться стабільними протягом усього проекту. Насправді ж вимоги часто змінюються, і така жорсткість може бути проблематичною, якщо після завершення фази необхідно внести зміни. Перегляд і модифікація попередніх фаз може бути дорогим і трудомістким процесом.

Ще одним суттєвим недоліком є затримка фази тестування. Тестування починається лише після фази впровадження, а це означає, що будь-які дефекти або проблеми не виявляються до кінця процесу розробки. Це може призвести до значних переробок і збільшення витрат, якщо будуть виявлені серйозні проблеми. На противагу цьому, ітеративні моделі, такі як Agile, включають безперервне тестування, що дозволяє виявляти і вирішувати проблеми на більш ранніх стадіях.

Модель водоспаду також має тенденцію бути менш ефективною для проектів, де вимоги не є чітко визначеними з самого початку. Припущення про стабільність вимог робить її менш придатною для проектів у динамічних і швидкозмінних середовищах. Лінійний підхід також може призвести до збільшення тривалості проекту, оскільки кожна фаза повинна бути завершена до початку наступної, що призводить до затримок, якщо на будь-якій фазі виникають проблеми.

Модель водоспаду з її структурованим і методичним підходом залишається наріжним каменем в управлінні проектами та розробці програмного забезпечення. Її чіткі фази та вичерпна документація забезпечують надійну основу для управління складними проектами. Однак її негнучкість і припущення про стабільні вимоги роблять її менш придатною для проектів у динамічних середовищах. Хоча новіші методології, такі як Agile, усунули багато обмежень моделі водоспаду, простота і ясність моделі водоспаду продовжують робити її цінним інструментом у правильних контекстах. Розуміння її сильних і слабких сторін дозволяє менеджерам проектів вибрати найкращий підхід для своїх конкретних потреб, що гарантує успіх проекту.

Монолітна архітектура програмного забезпечення – це традиційний підхід до проектування програмного забезпечення, коли всі компоненти та функції програми переплітаються в єдине ціле. Цей архітектурний стиль широко використовується в індустрії програмного забезпечення завдяки простоті процесів розробки та розгортання. У монолітній архітектурі всі модулі тісно пов'язані між собою, а це означає, що додаток розробляється, тестується і розгортається як єдине ціле.

Монолітна архітектура визначається єдиною, уніфікованою кодовою базою, в якій знаходяться всі функціональні можливості програми. Цей архітектурний стиль характеризується наступними особливостями:

- **єдиний модуль розгортання**: У монолітній архітектурі весь додаток пакується і розгортається як єдине ціле. Це означає, що

всі компоненти, включаючи користувацькі інтерфейси, бізнес-логіку та рівні доступу до даних, об'єднані разом.

- **тісний зв'язок**: Компоненти в монолітному додатку тісно пов'язані між собою, що означає, що зміни в одній частині системи можуть вплинути на інші частини. Така тісна інтеграція полегшує прямий зв'язок між компонентами, але також може призвести до ускладнень в обслуговуванні та масштабуванні.
- **єдина кодова база**: Додаток розробляється на єдиній кодовій базі, що полегшує управління та версіювання. Всі розробники працюють над однією і тією ж кодовою базою, що спрощує співпрацю та обмін кодом.
- **уніфікована база даних**: Зазвичай монолітний додаток використовує єдину базу даних для зберігання всіх своїх даних. Такий уніфікований підхід спрощує управління даними, але може створювати вузькі місця в продуктивності по мірі зростання додатку.

Монолітна архітектура має кілька переваг, особливо для малих і середніх додатків і команд.

Простота і легкість розробки. Монолітна архітектура відносно проста у розробці та розгортанні. Уніфікована база коду означає, що розробники можуть швидко створювати та ітерації над додатком без необхідності керувати складним міжсервісним зв'язком.

Продуктивність. Монолітні додатки часто працюють добре, тому що всі компоненти працюють в рамках одного процесу. Це зменшує накладні витрати, пов'язані з міжпроцесною взаємодією, які можуть бути значними в архітектурах мікросервісів.

Простота тестування та налагодження. Оскільки додаток є єдиним цілим, тестування та налагодження є більш простими. Розробники можуть запускати весь додаток локально, що полегшує виявлення та виправлення проблем.

Єдиний конвеєр розгортання. Розгортання монолітного додатку передбачає розгортання одного модуля, що спрощує процес розгортання. Конвеєри безперервної інтеграції та розгортання (CI/CD) легше налаштовувати та керувати монолітними додатками порівняно з тими, що включають декілька мікросервісів.

Ефективність використання ресурсів. Монолітні додатки можуть бути більш ресурсоефективними, оскільки вони не потребують декількох екземплярів середовища виконання або накладних витрат, пов'язаних з управлінням декількома сервісами.

Незважаючи на свої переваги, монолітна архітектура має кілька суттєвих недоліків, особливо в міру того, як додатки збільшуються в розмірі та складності.

Проблеми з масштабуванням. Масштабування монолітного додатку може бути складним завданням. Оскільки всі компоненти тісно пов'язані між собою, важко масштабувати окремі частини програми незалежно. Це може призвести до неефективності та збільшення витрат, оскільки весь додаток необхідно масштабувати, щоб впоратися зі збільшенням навантаження на будь-який окремий компонент.

Обмежена гнучкість. Тісний зв'язок між компонентами означає, що зміни в одній частині програми можуть призвести до необхідності внесення змін в інші. Відсутність гнучкості може сповільнити розробку і ускладнити впровадження нових функцій або технологій.

Складність у великих додатках. У міру зростання додатку, база коду може стати громіздкою і складною в управлінні. Це може призвести до збільшення часу збірки та розгортання, підвищення ризику внесення помилок і більш крутої кривої навчання для нових розробників.

Ризики розгортання. Оскільки весь додаток розгортається як єдине ціле, навіть невеликі зміни вимагають перерозгортання всього додатка. Це збільшує ризик простою і ускладнює впровадження розгортання з нульовим часом простою.

Технічний борг. З часом монолітні додатки можуть накопичувати технічний борг, оскільки компоненти стають тісно переплетеними. Рефакторинг кодової бази для усунення цієї заборгованості може бути масштабним заходом, який часто вимагає значних ресурсів і часу.

Монолітна архітектура програмного забезпечення залишається життєздатним та ефективним підходом для багатьох додатків, особливо для тих, що мають малий та середній розмір або відносно стабільні та добре зрозумілі вимоги. Простота, легкість розробки та переваги продуктивності роблять її привабливим вибором для багатьох команд розробників. Однак, зі зростанням складності та масштабу додатків, обмеження монолітної архітектури стають більш помітними. Проблеми з масштабуванням, обмежена гнучкість та ризик накопичення технічного боргу є суттєвими недоліками, які можуть стати на заваді довгостроковому успіху проекту.

В останні роки багато організацій перейшли до архітектури мікросервісів, щоб подолати ці обмеження, забезпечуючи більшу масштабованість, гнучкість і стійкість. Проте, розуміння сильних і слабких сторін монолітної архітектури дозволяє архітекторам і розробникам програмного забезпечення приймати обґрунтовані рішення, обираючи архітектурний стиль, який найкраще відповідає потребам і цілям їхнього проекту. Ретельно зважуючи компроміси, команди можуть використовувати переваги монолітної архітектури, пом'якшуючи при цьому її недоліки, забезпечуючи успішну реалізацію надійних і зручних для обслуговування програмних рішень.

2.2 Аналіз варіантів використання

Дослідження варіантів використання в інформаційних системах є критично важливим етапом, який забезпечує глибоке розуміння їх

призначення та функціональної діяльності. Особливо це актуально для систем машинного навчання, призначених для передбачення споживчого попиту. Такий підхід дозволяє ретельно аналізувати, як користувачі взаємодіють із системою, які сценарії використання можливі, та які типи користувачів будуть залучені.

Системи машинного навчання для передбачення споживчого попиту знаходять застосування у різних сферах бізнесу та промисловості. Використання діаграм використання в таких системах дозволяє краще зрозуміти та оптимізувати процеси, зокрема:

Роздрібна торгівля. У сфері роздрібної торгівлі передбачення споживчого попиту є ключовим для управління запасами та планування закупівель. Діаграми використання допомагають зрозуміти, як продавці та менеджери використовують систему для аналізу попиту, планування акцій та оптимізації запасів.

Логістика і ланцюги постачання. Для компаній, що займаються логістикою, передбачення попиту дозволяє оптимізувати маршрути доставки та мінімізувати витрати. Діаграми використання можуть ілюструвати взаємодію між логістичними операторами, системою прогнозування та постачальниками.

Виробництво. Виробничі підприємства використовують системи прогнозування попиту для планування виробничих потужностей та забезпечення безперебійного процесу виробництва. Діаграми використання можуть відображати взаємодію між виробничими менеджерами, аналітиками попиту та системою машинного навчання.

Фінансові послуги. У фінансовому секторі системи передбачення попиту можуть використовуватися для аналізу ринкових трендів та планування продуктів. Діаграми використання допомагають зрозуміти, як аналітики та фінансові консультанти використовують систему для прийняття стратегічних рішень.

Найпростішою та ефективною формою візуалізації взаємодії між користувачем та системою є діаграма використання. Вона ілюструє взаємодію користувача з різними сценаріями використання системи, дозволяючи ідентифікувати різні типи користувачів та моделі використання. Ці діаграми можуть використовуватися разом з іншими видами діаграм для створення повної картини функціонування системи.

Діаграми використання, завдяки їх простоті, можуть стати ефективним інструментом комунікації зі зацікавленими сторонами, допомагаючи їм краще зрозуміти, як система буде розроблятися та працювати. У відомому вислові «план використання – це суть вашої системи» закладена істина: ці плани дозволяють зобразити основні функціональні аспекти системи на високому рівні, роблячи їх зрозумілими навіть для неспеціалістів.

Однією з головних переваг використання діаграм використання є їх здатність надавати спрощене графічне представлення системи, що полегшує комунікацію між різними учасниками проекту. Зацікавлені сторони можуть легко зрозуміти динамічні аспекти системи, її можливості та обмеження. Дослідження показують, що використання діаграм допомагає передати наміри системи більш зрозуміло, ніж інші види діаграм, такі як діаграми класів.

Для створення діаграм використання використовуються кілька основних компонентів:

- **межа системи**: представляє собою прямокутник з ім'ям системи. ця межа може бути опущена, якщо не несе корисної інформації.
- **актор**: стилізована роль особи або іншої сутності, яка взаємодіє з системою. актори не пов'язані один з одним.
- **прецедент**: еліпс з підписом, що вказує на системну операцію. прецеденти представляють різні сценарії використання системи і описують, що система виконує, а не як це робиться.

Використання діаграм використання у системах машинного навчання для передбачення споживчого попиту дозволяє ретельно дослідити

взаємодію користувачів із системою, оптимізувати процеси та полегшити комунікацію зі зацікавленими сторонами. Завдяки своїй простоті та наочності, ці діаграми надають ефективний спосіб передачі складної інформації, сприяючи успішній реалізації проектів і досягненню стратегічних цілей організацій.

2.3 Проектування внутрішньої будови

Уніфікована мова моделювання (UML) є напівформальною мовою, яка використовується для моделювання різних типів систем. Створена Грейді Бучем, Джеймсом Рамбо та Іваром Якобсоном, UML зараз розробляється Object Management Group (OMG). UML застосовується для моделювання як існуючих реалій, наприклад, роботи відділу в компанії, так і для моделювання систем, які ще не створені, переважно для IT-систем.

UML використовується з графічним представленням, де елементи системи зображаються символами, що пов’язані між собою на діаграмах. На діаграмах варіантів використання можна побачити можливі дії користувачів у системі. UML офіційно визначено OMG у Метамоделі UML – Meta-Object Facility (MOF). Як і інші специфікації, засновані на MOF, метамодель UML і моделі UML можуть бути серіалізовані мовою XML Metadata Interchange (XMI), яка базується на стандарті XML. Хоча UML було розроблено для визначення, візуалізації, конструювання та документування програмно-інтенсивних систем, його застосування не обмежується програмним моделюванням. UML також використовується для моделювання бізнес-процесів, системної інженерії та представлення організаційних структур. Мова моделювання систем (SysML) є варіантом UML, створеним для моделювання системної інженерії, і визначена як профіль UML 2.0.

UML також може використовувати мову обмежень об’єктів (OCL) для розробки формальних обмежень.

UML не є методом сам по собі, але він був розроблений для сумісності з провідними методами розробки об’єктно-орієнтованого програмного забезпечення, такими як OMT, Booch, Objectory. У процесі розвитку UML деякі з цих методів було оновлено для використання нової нотації. Нові методи, як-от Rational Unified Process (RUP), також були створені на основі UML. Сьогодні існує багато інших методів на основі UML, призначених для надання більш конкретних рішень або досягнення певних цілей.

Для UML версії 2.2 існує 14 основних діаграм і 3 абстрактні діаграми, що охоплюють структуру, поведінку та взаємодії. Наприклад, діаграма класів є статичною структурною діаграмою, яка показує структуру системи в об’єктних моделях через ілюстрацію класів і зв'язків між ними. Вона відображає класи (типи) об’єктів у програмі, тоді як діаграма об’єктів показує екземпляри об’єктів і їхні залежності у певний момент часу. Діаграми класів дозволяють моделювати статичні аспекти перспективи проектування, формалізувати дані та специфікації методів, а також використовуватися як графічний засіб для відображення деталей реалізації класу. На рис. 2.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

Рисунок 2.1 – Діаграма варіантів використання

Клас у UML-моделі об’єктно-орієнтованої програми представлений прямокутником із назвою класу всередині нього. Відокремлена частина прямокутника під назвою класу може містити атрибути класу, такі як методи,

властивості або поля. Кожен атрибут відображається як ім’я, а також може мати тип, значення та інші характеристики. Методи класу можуть бути розміщені в окремій частині прямокутника. Атрибути і методи можуть бути позначені видимістю: "+" для public, "#" для protected, "−" для private, "~" для доступу на рівні пакета.

У UML є кілька видів зв’язків між класами. Залежність є найслабшим зв’язком і вказує на використання одного класу іншим. Асоціація представляє сильніший зв’язок між об’єктами класів, наприклад, коли компанія наймає працівників. Узагальнення або успадкування описує відносини між класами та підкласами, представляючи загальні та специфічні класи. Агрегація відображає відношення частина-ціле, де елементи можуть існувати окремо від цілого. Композиція, в свою чергу, є більш сильною формою агрегації, в якій частини належать лише одному цілому і знищуються разом з ним. Наприклад, двигун автомобіля може існувати окремо (агрегація) або бути знищеним разом з автомобілем (композиція).

Отже, UML є потужним інструментом для моделювання різних аспектів програмних і бізнес-систем, дозволяючи створювати точні та зрозумілі моделі, що сприяють ефективному плануванню, розробці та документуванню складних систем. На рис. 2.2 представлено внутрішню будову системи.

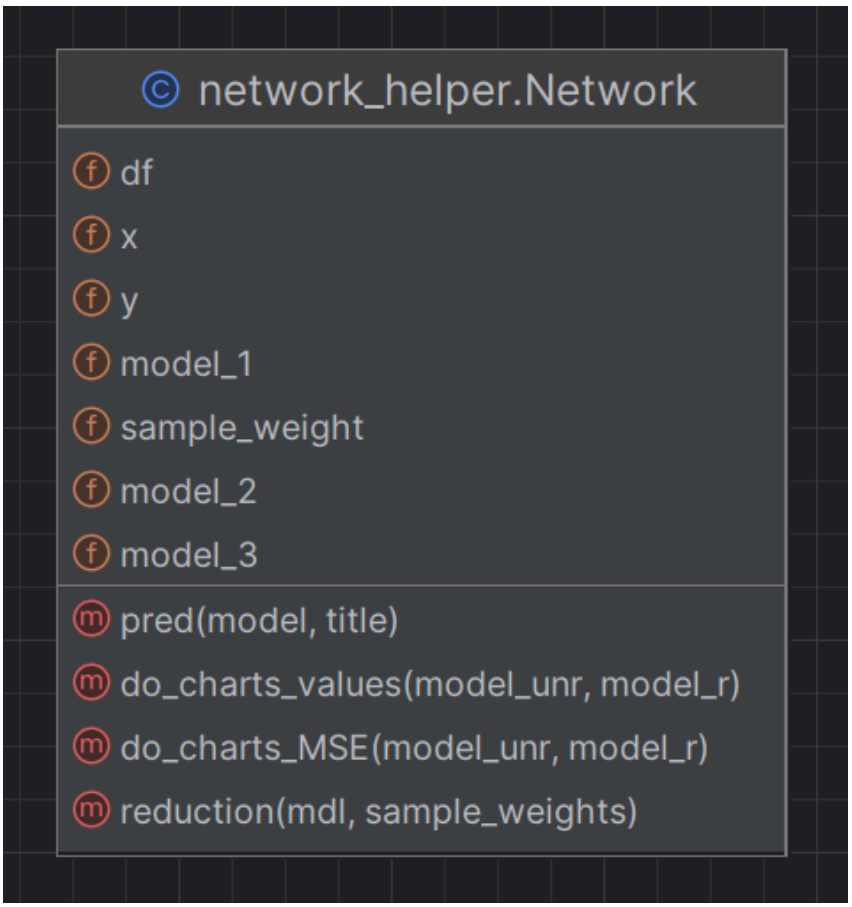

Рисунок 2.2 – Діаграма класів системи

Висновки до розділу 2

У розділі 2 було здійснено проектування системи прогнозування, розглянуто життєвий цикл розробки, проведено аналіз варіантів використання та виконано проектування внутрішньої будови системи. Спершу, досліджено життєвий цикл розробки програмного забезпечення для системи прогнозування, починаючи від етапу планування і завершуючи етапом підтримки. На етапі планування визначено основні вимоги до системи, встановлено цілі та завдання. Наступні етапи розробки включали аналіз, проектування, реалізацію, тестування та впровадження, що забезпечує послідовний підхід до створення надійної та ефективної системи.

Аналіз варіантів використання допоміг визначити ключові сценарії, в яких система прогнозування буде застосовуватися. Було розглянуто різні випадки використання. Цей аналіз дозволив краще зрозуміти потреби користувачів та адаптувати систему до різних умов експлуатації, забезпечуючи її гнучкість та універсальність.

Проектування внутрішньої будови системи зосереджувалося на розробці архітектури та основних компонентів.

Таким чином, у розділі 2 закладено основи для створення ефективної та функціональної системи прогнозування, що відповідає сучасним вимогам та потребам користувачів.

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ ПРОГНОЗУВАННЯ

3.1 Вибір інструментальних засобів розробки

Python – це універсальна мова програмування високого рівня, її використовують для розробки програмного забезпечення з машинним навчанням або аналізом даних. Синтаксис мови допомагає розробникам писати послідовний логічний код як у малих, так і у великих проектах.

Python підтримує динамічне збирання сміття, розробники можуть використовувати кілька парадигм програмування, наприклад структурне, об'єктно-орієнтоване та функціональне програмування. Розширена стандартна бібліотека Python часто називається «мовою з батарейками», але вона має величезну перевагу.

Гвідо ван Россум почав розробляти Python наприкінці 1980 років, а в 1991 році випустив першу версію під назвою Python 0.9.0. У 2000 році було представлено Python 2.0, що включав нові можливості, такі як розширені можливості управління списками та удосконалена система збору сміття на основі посилань. Python 2 був офіційно виведений з ужитку у 2020 році з випуском версії 2.7.18. Запущений у 2008 році, Python 3.0 став основною версією мови, не повністю сумісною з попередніми версіями, але затвердженою як стандарт.

Python завжди був однією з найпопулярніших мов програмування. Гвідо ван Россум розпочав розробку у Нідерландах на початку 1980-х років і розробив його як мову, що мала наступити за мову програмування ABC, розроблену SETC, що дозволила йому створювати винятки та взаємодіяти з операційною системою Amoeba. Перша версія Python була випущена в 1991 році, а Гвідо ван Россум керував розробкою проекту до 2018 року, коли він оголосив про те, що більше не займатиметься ним і отримав неформальний титул "благодійного диктатора за життя" для Python. Ван Россум завжди був відомий своєю відданістю спільноті Python та своїм довгостроковим внеском

у проект. Він продовжує відігравати ключову роль, беручи участь у роботі Комітету з аудиту Python разом з іншими розробниками. У 2019 році активні розробники Python core самі обрали Бретта Кеннона, Ніка Коглана, Баррі Варшаву, Керол Віллінг та Гвідо ван Россума до Ради директорів з 5 осіб. З тих пір Гвідо ван Россум зняв свою кандидатуру на посаду в раді директорів у 2020 році.

16 жовтня 2000 року був випущений Python 2.0 і приніс з собою значний набір нових функцій.

З релізом Python 2008, зокрема версіями 12/3/3.0, була запроваджена значуща редакція мови, яка не була повністю сумісна зі старішими версіями. Низка ключових функцій була поступово введена в Python 2.6. Версії 1.x та 2.7.x, а також інструмент 2to3 допомагають автоматично адаптувати ваш код з Python2 до Python3, що полегшує процес міграції.

Початково припинення підтримки Python 2.7 було заплановано на 2015 рік, але термін було відтерміновано до 2020 року через значний обсяг наявного коду, який ускладнював перехід на Python 3. Після закінчення періоду підтримки Python2.7 більше не отримуватиме оновлення безпеки чи інші виправлення, а отримуватиме лише підтримувані версії Python3.6.x або пізніших версій.

Зокрема, Python розроблений для підтримки функціонального програмування з широким набором функцій, таких як фільтрація, вибірка, агрегування, та ітерації.

Стандартна бібліотека включає модулі такі як itertools та functools, які забезпечують повний спектр функціональних інструментів, розроблених на зразок мов Haskell та Standard ML.

Python має вражаючу розширюваність завдяки модульному підходу, але не всі функції вбудовані в ядро мови. Ця модульність дозволяє легко додавати програмований інтерфейс до існуючого додатку і користується популярністю серед розробників завдяки розширенню його функціональності.

Гвідо ван Россум визначив основні принципи проектування на Python, зосередившись на читабельності коду, змістовному та зрозумілому синтаксисі та простоті вибору методів кодування. На відміну від філософії Perl, що «існує кілька способів зробити це», Python – це «більше ніж один спосіб зробити це» під девізом «вам потрібен лише 1, бажано 1 очевидний спосіб зробити це».

Розробники Python прагнуть запобігти передчасній оптимізації і не вносять змін у несуттєві частини CPython, які можуть підвищити швидкість за рахунок складності. Коли швидкість дуже важлива, програмісти можуть перетворити важливі фрагменти коду в Розширення, написані на C, використовувати інші інтерпретатори, наприклад PyPy. Існують, також такі варіанти, як Cython, які можуть перетворювати код Python в C і надавати прямий доступ до API рівня C інтерпретатора Python.

Однією з ключових цілей Python є забезпечення задоволення від програмування. На це вказує назва мови, отримана на честь британської комедійної групи Monty Python. Python також може підкреслити гумор у підручниках та довідкових матеріалах, наприклад, використовуючи відомі приклади, такі як «спам» та «яйце», як стандартні назви.

У порівнянні з іншими мовами, він не виділяє блоки коду за допомогою фігурних дужок або роздільників слів, а фокусується на відступах, що мають смислове значення.

Рекомендований розмір відступу – 4 пробіли.

Оператори Python включають (серед іншого):

- символ « = » як оператор присвоєння;
- «if» використовується як початок блоку умовного виконання коду, також використовуються «else» та «elif»;
- використовувати оператор «for», щоб відобразити ітеративний об'єкт і записати кожен елемент у локальну змінну, щоб використовувати його як вкладений блок;

- поки умова істинна, використовувати оператор «while» для виконання блоку коду;
- оператор «try» дозволяє перехоплювати та обробляти винятки, що містяться у блоці коду, гарантуючи, що код буде виконано без помилок;
- необхідно використовувати оператор «raise» для створення певного винятку або для повторного виклику перехопленого винятку;
- оператор «class» використовується в об'єктно-орієнтованому програмуванні для створення класів;
- оператор «def» визначає функцію або метод;
- випущений у 2006-2009 роках, оператор Python2.5 дозволяє керувати блоками коду в контекстному менеджері. Це дозволяє ініціалізувати та звільнити ресурс, замінюючи часто використовувану ідіому «try / finally», подібну до шаблону ініціалізації та вилучення ресурсів (RAII);
- оператор «break» завершує виконання циклу;
- оператор «продовжити» переходить до наступної ітерації та пропускає поточну ітерацію;
- оператор «del» видаляє змінну, видаляє всі посилання на значення та видає помилку при спробі використовувати змінну. Віддалені змінні можуть бути створені заново;
- оператор «Ppass» використовується для виконання функції NOP (без операції) і створення порожнього блоку коду;
- оператор «assert» використовується для налагодження;
- оператор «yield» використовується в функції генераторі для повернення значень;
- оператор «return» використовується для повернення змінної з функції;

- оператор import використовується для приєднання модулів, що містять функції або змінні, які можуть бути використані в поточному додатку. Для імпорту необхідно використовувати import<ім’я модуля>[як<псевдонім>],<ім’я модуля>import* або вкажіть<ім’я модуля>import<визначення 1>[Як<псевдонім 1>],<визначення 2>[Як<псевдонім 2>],....;

У Python імена змінних не обмежуються типами даних, а фіксовані типи не присвоюються. Однак, у певний момент часу змінна посилається на певний об'єкт із певним типом. Це називається динамічним набором тексту і є протилежністю статичному набору тексту.

Python не підтримує оптимізацію кінцевих викликів чи першокласні розширення, і згідно з висловлюваннями Гвідо ван Россума, такі можливості навряд чи будуть підтримуватися в майбутньому. Однак, у версії Python 2.5 покращено підтримку функцій, таких як генератори з обмеженнями, і інформація може передаватися між генераторами в одному напрямку. В Python 2.5 інформація може повертатися назад до функції генератора, тоді як у Python 3.3 інформація може транслюватися через кілька рівнів стека.

Враховуючи широке застосування та розширені можливості Python, зазначимо, що ця мова програмування вважається універсальною. Саме тому її було обрано для виконання даної роботи.

PyCharm – це інтегроване середовище розробки (IDE), створене спеціально для роботи з Python. Це середовище було розроблене компанією JetBrains, яка включає широкий спектр функцій, таких як аналіз коду, графічні вказівники, модулі інтегрованого тестування і інтеграції з системами контролю версій (VCS). PyCharm також підтримує Django та платформу Anaconda для аналізу даних.

PyCharm є кросплатформним, адже доступне для користувачів Windows, macOS та Linux. Від платної версії PyCharm Professional Edition з відкритим кодом до безкоштовної версії PyCharm Community Edition, ця професійна версія пропонує додаткові можливості та має власну ліцензію.

PyCharm регулярно оновлюється, і користувачі можуть отримувати оновлення функцій. Наприклад, бета-версія була запущена в період з 2010 по 2017 рік, тоді як версія 1.0 вийшла на ринок трохи пізніше. Ранні версії 2.0, 3.0 та 4.0 були випущені в різні роки.

PyCharm Community Edition, безкоштовна версія популярного інтегрованого середовища розробки Python (IDE) з відкритим кодом, була представлена 22 жовтня 2013 року. Це інструмент, який підтримує роботу розробників на Python і пропонує безліч можливостей для підвищення продуктивності та якості коду.

PyCharm Professional Edition, платна версія відомого інтегрованого середовища розробки Python (IDE), була розроблена компанією JetBrains.

Ця версія включає всі функції Community Edition і додає багато додаткових інструментів та можливостей, які спрямовані на поліпшення продуктивності та ефективності роботи професійних розробників. PyCharm Professional підтримує розробку веб-додатків з Django, Flask та іншими фреймворками, роботу з базами даних.

Scikit-images, також називається scikits.image – це бібліотека для обробки зображень з відкритим кодом для Python. Бібліотека містить алгоритми сегментації, геометричного перетворення, маніпулювання колірним простором, аналізу, фільтрації, морфології, виявлення об'єктів і багато іншого. Він був розроблений для тісної співпраці з NumPy та бібліотекою SciPy Python. Спочатку Проєкт був розроблений Стефаном ван дер Уолтом і розповсюджувався як незалежне розширення SciPy «SciKit» (SciPy toolkit). Після цього основна кодова база була істотно переписана іншими розробниками.

Keras – це бібліотека з відкритим кодом, що надає високорівневий інтерфейс для Python для роботи з нейронними мережами. Вона слугує інтерфейсом для бібліотеки TensorFlow. У версіях, що передували 2.3, Keras сприяв безпеці серверних компонентів, використовуючи TensorFlow, Microsoft Cognitive Toolkit, Theano, та PlaidML. З початку версії 2.4, Keras

зосереджується на швидких експериментах та акцентується на легкості використання, модульності, та масштабованості. Головним розробником та підтримувачем Keras є Франсуа Шолле, інженер з Google, який займається розробкою глибоких нейронних мереж, включаючи модель Xception.

Слід зазначити, що Keras, інтерфейс до бібліотеки TensorFlow, містить різні компоненти нейронної мережі, включаючи шари, цільові об'єкти, функції активації, оптимізатори та інструменти для роботи з зображеннями і текстовими даними. Ці компоненти полегшують написання коду для глибоких нейронних мереж.

Keras дає можливість розробляти згорткові та рекурентні нейронні мережі, а також загальні корисні рівні, такі як відключення пакетів та нормалізація, проекти розміщені на GitHub, а підтримка спільноти здійснюється через сторінку проблем на GitHub та канал Slack.

Keras може використовуватись для розробки програмного продукту, який працює на смартфонах (iOS та Android), в інтернеті або на віртуальних машинах Java. Він також підтримує розподілене навчання моделей на кластерах GPU та TPU.

Розроблена командою Google Brain, TensorFlow – це всім доступна бібліотека з відкритим кодом для машинного навчання та штучного інтелекту, орієнтована на вивчення та виявлення глибоких нейронних мереж. Випущений у 2015 році, TensorFlow доступний на різних мовах програмування, Python в тому числі.

При цьому, однією з ключових особливостей TensorFlow є самовизначення, яке дозволяє автоматично обчислювати градієнти параметрів моделі, важливі для алгоритмів оптимізації, таких як зворотне поширення. TensorFlow відстежує порядок операцій, що виконуються над вхідними тензорами в моделі, і обчислює градієнти відповідних параметрів.

При постійному виконанні TensorFlow включає режим «своєчасного виконання». Це означає, що операція обчислюється негайно, а не додається до графіка обчислень для подальшого виконання дані вводяться в кожному

рядку коду, а не в графіку обчислень, тому можна поступово бачити, який код виконується за допомогою налагоджувача, завдяки своїй покроковій прозорості ця парадигма виконання вважається простою у налагодженні [34].

TensorFlow надає API для ефективного розподілу обчислень серед різних пристроїв із варіативними стратегіями розподілу, часто застосовуючи це для навчання і оцінювання моделей у тензорних потоках, що стало стандартною практикою в ШІ [35, 36].

У TensorFlow також присутній різноманітний набір функцій для втрат, включаючи ті, що добре відомі, як функції втрат для навчання та оцінки моделей. До прикладів відносяться середньоквадратична помилка (MSE) та бінарна перехресна ентропія [37].

Ці функції втрат дозволяють квантифікувати "помилку" між вихідними даними моделі та очікуваними вихідними даними, сприяючи двома різними підходами. Різні набори даних та моделі використовують різні типи втрат для визначення пріоритетності конкретних аспектів продуктивності.

TensorFlow забезпечує API-доступ до різних метрик, що вимірюють продуктивність моделей машинного навчання. Можна використовувати різні показники точності, включаючи двійкові, категоріальні та поєднані, такі як точність, відкидання та об'єднання перетину через об'єднання (IoU) [38].

Tensorflow.nn – це модуль для виконання примітивних нейромережевих операцій над моделлю деякі з цих операцій включають в себе зміну згортки (1/2/3D, плоску, в глибину), функції активації (Softmax, RELU, GELU, сигмовидну тощо). Інші тензорні операції (максимальне об'єднання, зміщення-додавання та ін.) і їх варіації, а також інші тензорні операції (максимальне об'єднання, зміщення-додавання тощо) [39].

Щодо оптимізатора, то TensorFlow містить в собі клас оптимізаторів, таких як adam, ADAGRAD та стохастичний градієнтний спуск (SGD) [40]. Під час навчання моделі різні оптимізатори забезпечують різні режими налаштування параметрів, які часто впливають на продуктивність моделі.

Tensorflow.Keras розроблений як абстрактний інтерфейс високого рівня для інших подібних бібліотек низького рівня і призначений для швидкого створення прототипів TensorFlow, Microsoft Cognitive Toolkit (CNTK) та глибоких нейронних мереж бібліотеки Theano як серверних бібліотек, забезпечуючи простоту використання, модульність та масштабованість. Він орієнтований на масштабованість.

Keras розроблений в рамках дослідницького проекту ONEIROS [2], його автором є Франсуа Шолле з Google [3].

У 2017 році команда TensorFlow вирішила офіційно підтримати keras [4]. Шолле заявив, що Keras розроблений не як окрема бібліотека, а як інтерфейс.

Він пропонує набір модулів, які дозволяють розробляти глибокі нейронні мережі, використовуючи загальну інтуїтивно зрозумілу мову [4]. Microsoft додала бекенд до CNTK з версії CNTK 2.0. [5, 6].

3.2 Розробка механізмів прогнозування

Програмний продукт складається з багатьох елементів. В даному підрозділі переглянуто їх всі. Основними функціями застосунку є створення датасетів для подальшого їх використання, навчання моделей машинного навчання з різними параметрами для того, щоб порівняти їх, прогнозування і підрахування середньої квадратичної похибки, редукція та графічне представлення результатів.

Перш за все необхідно отримати дані для їх обробки. Це зроблено за допомогою бібліотеки pandas, яка використовується для створення типу даних DataFrame. Цей тип даних дуже зручний для аналізу інформації, оптимізований для препроцесингу. Далі вибираються тільки ті стовпці, які будуть використані для тренування моделі та її тестування.

Наступна частина програмного продукту тренує три моделі машинного навчання. Їх кілька, щоб продемонструвати ефективність застосованих практик. Перша модель це лінійна регресія без додавання ваг, друга з їх додаванням, третя з використанням регресії.

Створено окрему функцію для прогнозування використовуючи натреновані моделі. Вона буде використана для їх аналізу. Також ця функція вираховує середню квадратичну похибку. Для програмного продукту обрана саме вона, адже прогнозування споживчого попиту це задача регресії, а не класифікації. Ця похибка буде використана для порівняння цих моделей.

Реалізовано функцію для редукції ваг моделі. Це потужний інструмент, адже вона переглядає кожен зв’язок нейронної мережі та видаляє ті, які погіршують результат. Отже, редукція значно впливає на результати нейронної мережі.

Написано дві функції для візуалізації результатів роботи програмного продукту. Одна з них графічно порівнює результати прогнозів різних моделей машинного навчання. Цей графік демонструє те як моделі можуть бути практично використані та наскільки точні їх прогнози. Другий графік демонструє середні квадратичні похибки всіх моделей.

3.3 Розробка інтерфейсу системи

Консольний інтерфейс, або командний рядок, є одним із найосновніших елементів комп'ютерних систем. Він надає користувачам можливість взаємодії з комп'ютером через введення текстових команд, на відміну від графічного інтерфейсу користувача (GUI). Такий спосіб взаємодії особливо популярний серед системних адміністраторів, розробників програмного забезпечення та IT-професіоналів завдяки високому рівню контролю і гнучкості, який він забезпечує.

Термінал – це ключовий елемент сучасних операційних систем, які забезпечують користувачам потужність і гнучкість в управлінні комп'ютером. Він є своєрідним мостом між користувачем і операційною системою, дозволяючи виконувати різноманітні завдання та отримувати доступ до різних ресурсів.

Командний рядок, часто відомий як консоль або термінал, є текстовим інтерфейсом, який дозволяє користувачам взаємодіяти з операційною системою за допомогою команд. Це середовище, де можна виконувати програми, переглядати файли, керувати системою та багато іншого. Відмінною особливістю командного рядка є його потужність і гнучкість, оскільки він дозволяє виконувати завдання, які можуть бути складними або навіть неможливими в інших середовищах.

Крім того, термінал відкриває двері до світу автоматизації та скриптів, дозволяючи користувачам створювати скрипти, які виконують послідовність команд автоматично. Це не лише економить час, але й робить рутинні завдання більш ефективними та простими. Відправляючи команди через термінал, користувач може швидко отримати необхідну інформацію чи виконати дії, використовуючи мінімальну кількість ресурсів.

Крім технічної складності, термінал також навчає користувачів системній архітектурі та принципам операційних систем. Він дозволяє краще розуміти те, як комп'ютер взаємодіє з користувачем та іншими програмами. Це важливо не лише для інженерів та програмістів, але й для звичайних користувачів, які хочуть краще розуміти свої комп'ютери та взаємодіяти з ними більш ефективно.

Таким чином, термінал і командний рядок відіграють важливу роль у сучасних операційних системах, надаючи користувачам потужний інструмент для взаємодії з комп'ютером. Вони дозволяють виконувати різноманітні завдання, навчають основам комп'ютерної архітектури та дозволяють автоматизувати рутинні дії. Таким чином, вони є не лише інструментом, але й ключовою складовою сучасної комп'ютерної культури.

Розроблений інтерфейс представлено на рис. 3.1.

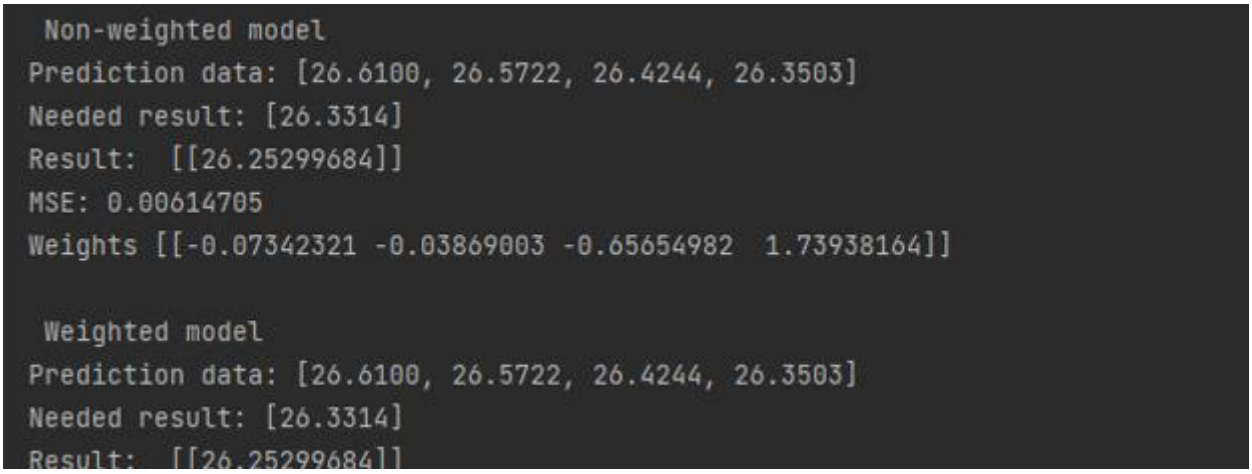

Рисунок 3.1 – Розроблений інтерфейс системи

3.4 Аналіз результатів

Програмний продукт розроблено так, щоб можна було побачити та проаналізувати проміжні результати роботи програмного продукту. Для цього реалізовано їх вивід в термінал.

Програмний продукт виводить в консоль результати прогнозів для кожної з трьох описаних вище моделей і середньоквадратичну помилку для них (рис. 3.2)

Для більш детального розуміння виконаної роботи та її результатів були розроблені функції побудови графіків порівняння. Вони дозволяють наочно оцінити різницю в прогнозах і середньоквадратичних помилках між звичайною нейронною мережею та мережею, що пройшла процедуру редукції.

Графік порівняння прогнозів зображено на рис. 3.3.

```
 Non-weighted model
Prediction data: [26.6100, 26.5722, 26.4244, 26.3503]
Needed result: [26.3314]
Result:  [[26.25299684]]
MSE: 0.00614705
Weights [[-0.07342321 -0.03869003 -0.65654982  1.73938164]]

 Weighted model
Prediction data: [26.6100, 26.5722, 26.4244, 26.3503]
Needed result: [26.3314]
Result:  [[26.25299684]]
MSE: 0.00614705
Weights [[-0.07342321 -0.03869003 -0.65654982  1.73938164]]

 Reduced model
Prediction data: [26.6100, 26.5722, 26.4244, 26.3503]
Needed result: [26.3314]
Result:  [[26.32484591]]
MSE: 0.00004296
Weights [[-0.01913052 -0.02511482 -0.4843013   1.49165757]]
```

Рисунок 3.2 – Вивід результатів роботи на консоль

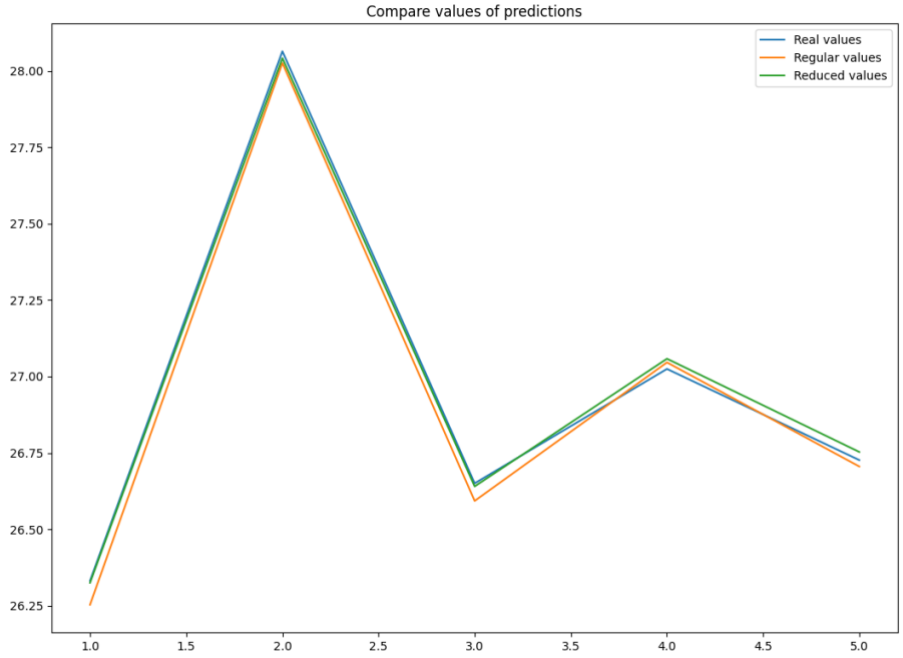

Рисунок 3.3 – Графік прогнозів моделей

Графік порівняння середньоквадратичних помилок зображено на рис. 3.4.

Як показано на графіках, редукція не завжди створює мережу, яка перевершує стандартну у всіх випадках. Проте зменшення середньоквадратичної помилки на більшості інтервалів можна вважати позитивним результатом. Це відбувається тому, що загальна середня похибка для всього датасету значно зменшується завдяки її зниженню в більшості ситуацій.

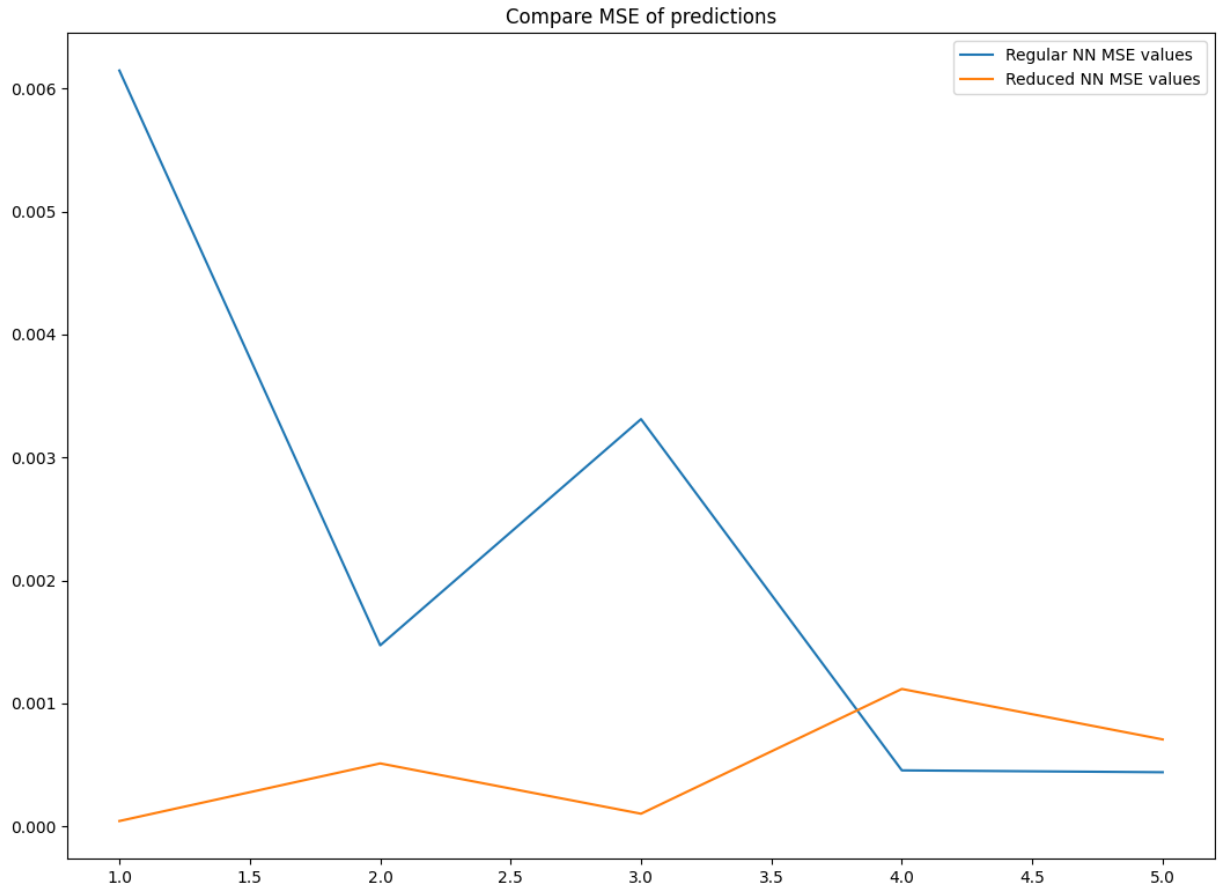

Рисунок 3.4 – Графік середньоквадратичних помилок

Висновки до розділу 3

У третьому розділі дипломної роботи детально описується процес розробки системи штучного інтелекту, починаючи з вибору інструментальних засобів і закінчуючи тестуванням системи. Вибір Python і бібліотеки scikit-learn для розробки підкреслює зосередження на гнучкості та ефективності. Розробка та навчання моделі штучного інтелекту відіграють ключову роль, забезпечуючи основу для функціональності системи. Паралельно, важливу роль відіграє створення інтуїтивно зрозумілого інтерфейсу, що забезпечує взаємодію користувача з системою.

Завершальний етап тестування дозволяє виявити та виправити можливі помилки та недоліки, гарантуючи надійність та ефективність розробленої системи. Загалом, цей розділ відображає комплексний та систематизований підхід до розробки системи штучного інтелекту, від підбору інструментів до фінального тестування.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

В даному розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на передбаченні споживчого попиту та оптимізації запасів в роздрібних мережах.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі. Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

Метод ФВА застосовується для проведення техніко-економічного аналізу розробки моделей для прогнозування ринку цінних паперів.

Фактичний аналіз це аналіз функцій програмного продукту призначеного для того, щоб точно та надійно прогнозувати споживчий попит та покращити аналітичні можливості у сфері електронної комерції.

Вимоги до ПП (програмного продукту):

- можливість запустити програму на стандартних персональних комп'ютерах;
- зручність та зрозумілість для користувача;
- помірна швидкість тренування моделей;
- зручний для аналізу вивід результатів.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту призначеного для того, щоб точно та надійно прогнозувати споживчий попит. Це основна функція, також можна виділити такі:

- F_1 – вибір мови програмування;
- F_2 – вибір фреймворку для машинного навчання;
- F_3 – вибір середовища розробки.

В кожної з цих функцій є декілька варіантів реалізації.

Функція F_1:

- Python;
- C++.

Функція F_2:

– scikit-learn;

– Keras.

Функція F_3:

– Visual Studio Code;

– PyCharm.

Морфологічна карта системи з варіантами реалізації основних функцій (рис. 4.1).

В морфологічній карті відображається множина всіх можливих варіантів основних функцій.

Побудуємо позитивно-негативну матрицю варіантів основних функцій (табл. 4.1). Нам варто відкинути деякі варіанти, адже вони не задовольняють потреби програмного продукту. Ці варіанти відзначені у морфологічній карті.

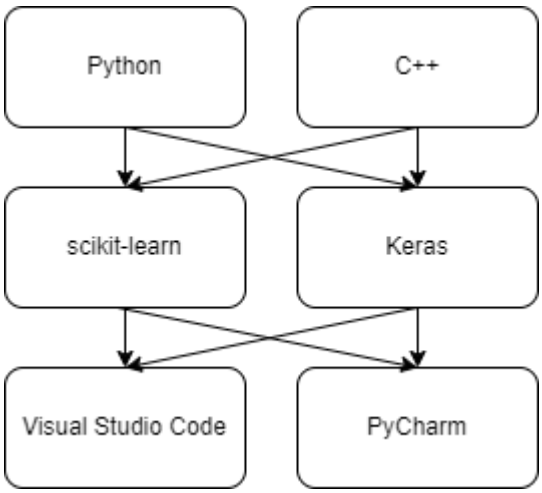

Рисунок 4.1 – Морфологічна карта

Функція F_1: доцільним вибором буде більша кількість бібліотек та інструментів, які вже оптимізовані розробниками. Варіант Б відкинутий.

Функція F_2: обидва варіанти підходять для розробки, тому можна використати варіант А чи Б.

Функція F_3: віддаємо перевагу варіанту Б, щоб використовувати зручні інструменти для розробки.

Таким чином, будемо розглядати такий варіант реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{б},$$
$$F_1\text{а} - F_2\text{б} - F_3\text{б}.$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	*А*	Легкий синтаксис, великий вибір бібліотек	Гірша швидкість роботи
	Б	Швидка мова програмування	Більше часу і зусиль на написання програми
F_2	*А*	Широкий вибір інструментів для машинного навчання	Гірше підходить для глибоких нейронних мереж
	Б	Легка у використанні	Менш гнучка для складних моделей
F_3	*А*	Багато корисних розширень, які допомагають в розробці	Обмежені обчислювальні ресурси для тренування моделі
	Б	Оптимізоване середовище розробки з великою кількістю інструментів	Висока ресурсоємність

4.3 Обґрунтування системи параметрів ПП

Для розрахунку коефіцієнта технічного рівня потрібно визначити основні параметри, ґрунтуючись на інформації, розглянутій вище.

Обрані параметри для характеристики програмного продукту:

– *X*1 – швидкодія мови програмування;
– *X*2 – об’єм пам’яті для обчислень та збереження даних;
– *X*3 – тривалість роботи програми;
– *X*4 – приблизний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	*Умовні позначення*	*Одиниці виміру*	*Значення параметра*		
			гірші	середні	кращі
Швидкодія мови програмування	*X*1	оп/мс	9000	13000	16000
Об’єм пам’яті	*X*2	Мб	3	2	1,5
Тривалість роботи програми	*X*3	мс	6000	4500	3500
Приблизний об’єм програмного коду	*X*4	кількість рядків коду	170	125	100

За даними табл. 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

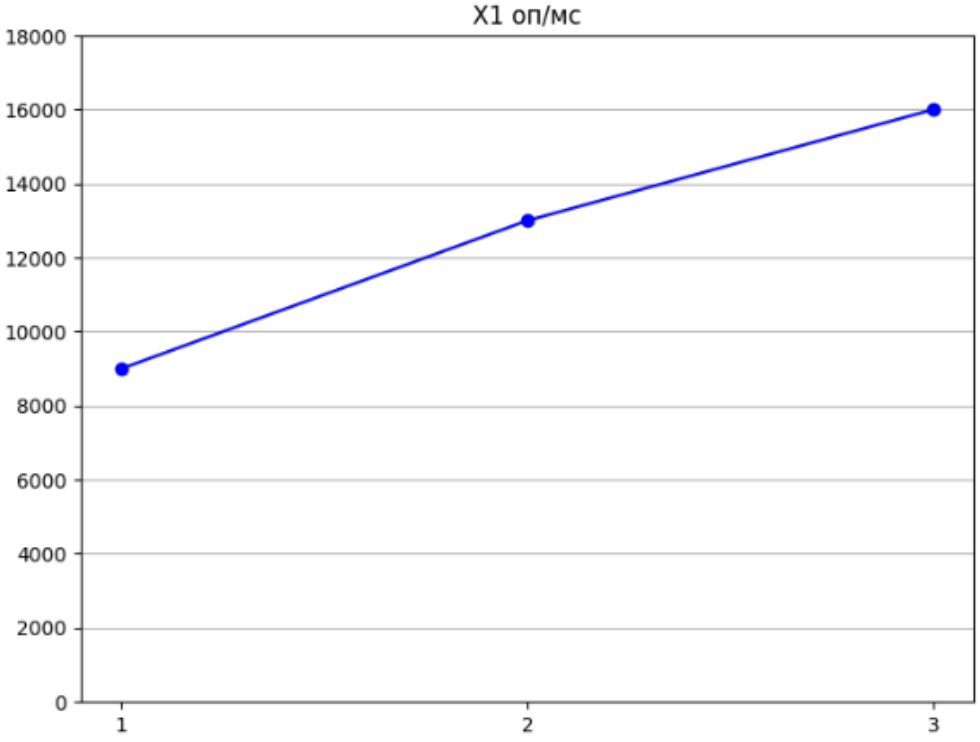

Рисунок 4.2 – *X*1, швидкодія мови програмування

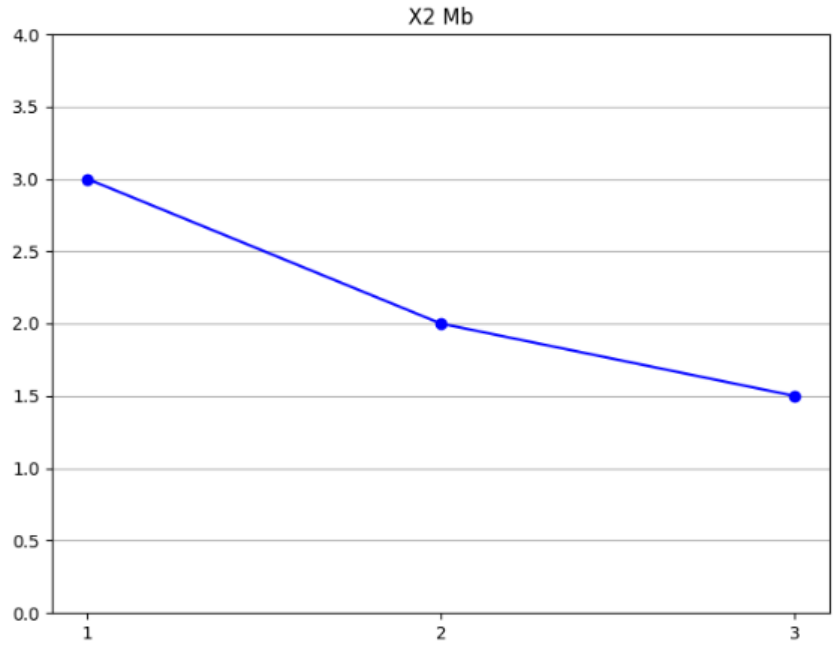

Рисунок 4.3 – *X*2, об’єм пам’яті

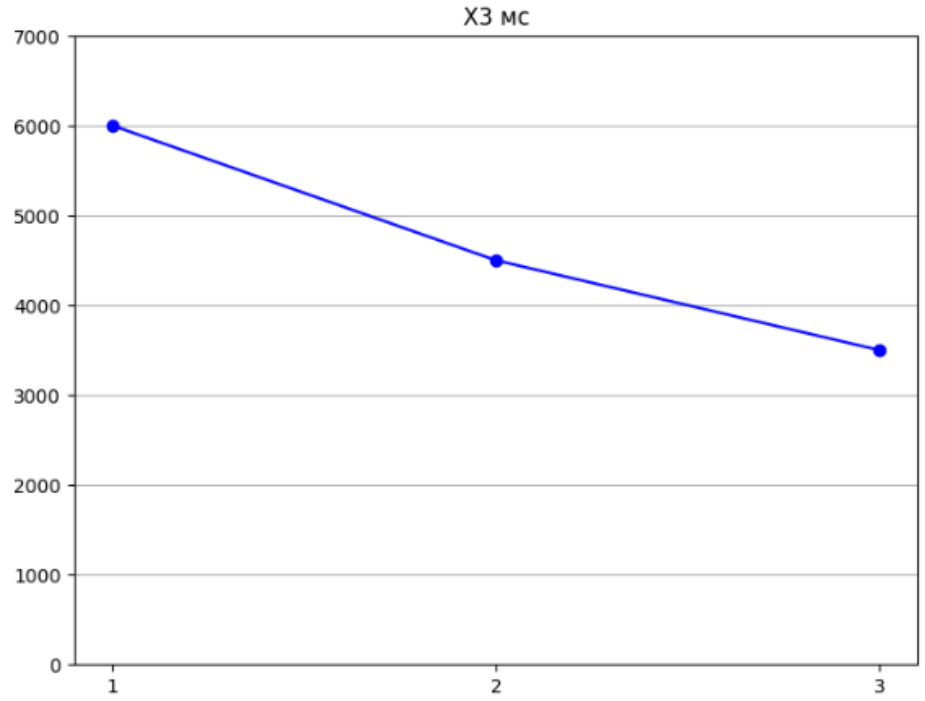

Рисунок 4.4 – *X*3, тривалість навчання моделі

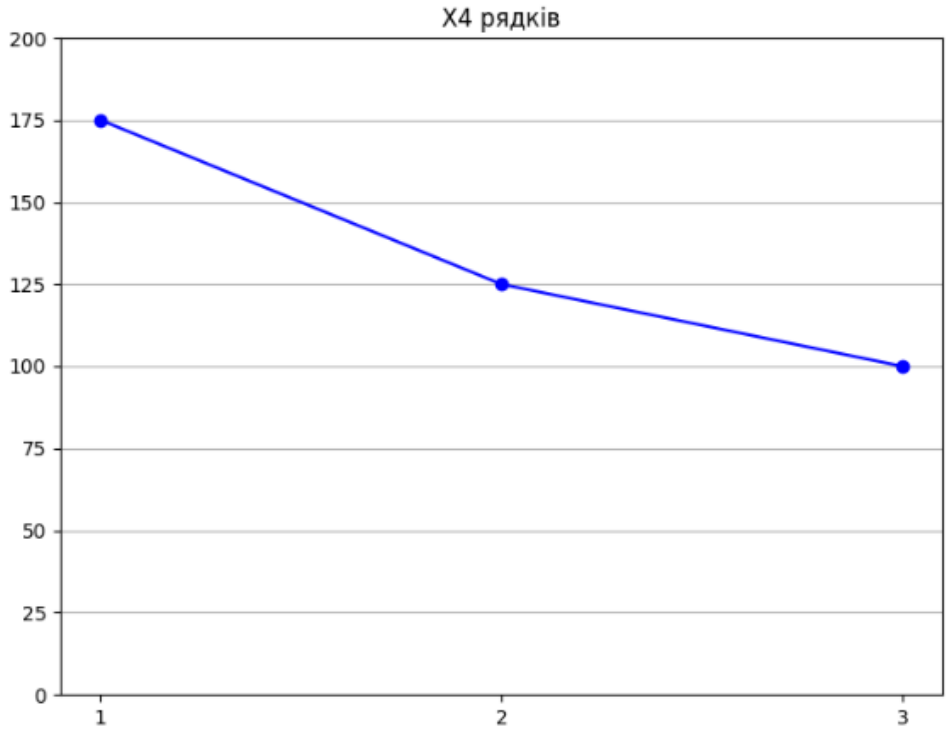

Рисунок 4.5 – *X*4, приблизний об’єм програмного коду

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у табл. 4.3.

Для перевірки степені достовірності експертних оцінок, визначимо параметри.

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів,

n – кількість параметрів.

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 195.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 195}{7^2(4^3 - 4)} = 0{,}796 > W_k = 0{,}67.$$

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	*Назва параметра*	*Одиниці виміру*	*Ранг параметра за оцінкою експерта*							*Сума рангів* R_i	*Відхилення* Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	оп/мс	3	4	3	3	3	3	2	21	3,5	12,25
X2	Об’єм пам’яті	Мб	2	1	2	2	1	2	3	13	-4,5	20,25
X3	Тривалість роботи програми	мс	4	3	4	4	4	4	4	27	9,5	90,25
X4	Приблизний об’єм програмного коду	Кількість рядків коду	1	2	1	1	2	1	1	9	-8,5	72,25
	Разом		10	10	10	10	10	10	10	70	0	195

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у табл. 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Пара-метри	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	2	3	4	5	6	7		
*X*1 і *X*2	>	>	>	>	>	>	<	>	1,5
*X*1 і *X*3	<	<	<	<	<	>	<	<	0,5
*X*1 і *X*4	>	>	>	>	>	>	>	>	1,5
*X*2 і *X*3	<	<	<	<	<	<	<	<	0,5
*X*2 і *X*4	>	>	>	<	<	>	>	>	1,5
*X*3 і *X*4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

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

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

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{в}i} = \frac{b_i}{\sum_{i=1}^{n} b_i},$$

$$b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{в}i} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з табл. 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	*Параметри* x_j				*Перша ітер.*		*Друга ітер.*		*Третя ітер.*	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
*X*1	1	1,5	0,5	1,5	4,5	0,28	16,25	0,28	59,125	0,27
*X*2	0,5	1	0,5	1,5	3,5	0,22	12,25	0,21	44,875	0,21
*X*3	1,5	1,5	1	1,5	5,5	0,34	21,25	0,36	77,875	0,36
*X*4	0,5	0,5	0,5	1	2,5	0,16	9,25	0,15	34,125	0,16
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (об'єм пам'яті), *X3* (тривалість тренування моделі) та *X4* (приблизний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (табл. 4.6):

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

де n – кількість параметрів,

$K_{вi}$– коефіцієнт вагомості i–го параметра,

B_i – оцінка i–го параметра в балах.

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

Основні функції	*Варіант реалізації функції*	*Пара-метри*	*Абсолютне значення параметра*	*Бальна оцінка пара-метра*	*Коефіцієнт вагомості параметра*	*Коефі-цієнт рівня якості*
*F*1	А	*X*1	15000	4	0,27	1,08
*F*2	А	*X*4	126	5	0,16	0,8
	Б	*X*4	150	3	0,16	0,48
*F*3	Б	*X*2	2	5	0,21	1,05

За даними з таблиці 4.6 за формулою:

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

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}08 + 0{,}8 + 1{,}05 = 2{,}93.$$

$$K_{K2} = 1{,}08 + 0{,}48 + 1{,}05 = 2{,}61.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Усі варіанти охоплюють два окремих завдання.

1. Розробка проекту програмного продукту.

2. Розробка програмної оболонки.

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

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

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

Загальна трудомісткість обчислюється як:

$$T_О = T_Р \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М},$$

де $T_Р$ – трудомісткість розробки ПП,

$K_П$ – поправочний коефіцієнт,

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

$K_М$ – коефіцієнт рівня мови програмування,

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

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

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ = 56 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання:

$К_П$ = 1,7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0,9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 60 \cdot 1,7 \cdot 0,9 = 85,68 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ = 28 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$T_2 = 30 \cdot 0.9 \cdot 0,8 = 20,16 \text{ людино-днів.}$$

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

$$T_I = (85,68 + 20,16 + 4,8 + 20,16) \cdot 8 = 1046,4 \text{ людино-годин.}$$

$$T_{II} = (85,68 + 20,16 + 6,91 + 20,16) \cdot 8 = 1063,28 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

У розробці беруть участь два програмісти з окладом 22000 грн, один аналітик в області даних з окладом 24000. Визначимо середню зарплату за годину за формулою:

$$C_Ч = \frac{M}{T_m \cdot t} \text{грн,}$$

де M – місячний оклад працівників,

T_m – кількість робочих днів в місяць,

t – кількість робочих годин в день.

$$C_{\text{ч}} = \frac{22000 + 22000 + 24000}{3 \cdot 21 \cdot 8} = 134{,}92 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{ЗП}} = C_{\text{ч}} \cdot T_i \cdot K_{\text{Д}},$$

де $C_{\text{ч}}$ – величина погодинної оплати праці програміста,

T_i – трудомісткість відповідного завдання,

$K_{\text{Д}}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $C_{\text{ЗП}} = 134{,}92 \cdot 1046{,}4 \cdot 1{,}2 = 169417{,}14$ грн,

II. $C_{\text{ЗП}} = 134{,}92 \cdot 1063{,}28 \cdot 1{,}2 = 172150{,}1$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0{,}22 = 169417{,}14 \cdot 0{,}22 = 37271{,}77$ грн,

II. $C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0{,}22 = 172150{,}1 \cdot 0{,}22 = 37873{,}02$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($C_{\text{М}}$)

Так як одна ЕОМ обслуговує одного програміста з окладом 22000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_{\text{Г}} = 12 \cdot \text{М} \cdot \text{К}_{\text{З}} = 12 \cdot 22000 \cdot 0{,}2 = 52800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г \cdot (1 + К_З) = 48000 \cdot (1 + 0,2) = 63360 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{ЗП} \cdot 0,22 = 57600 \cdot 0,22 = 13939,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 22000 грн:

$$C_А = K_{ТМ} \cdot K_А \cdot Ц_{ПР} = 1,1 \cdot 0,25 \cdot 22000 = 6050 \text{ грн,}$$

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

$K_А$ – річна норма амортизації,

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_Р = K_{ТМ} \cdot Ц_{ПР} \cdot K_Р = 1,1 \cdot 22000 \cdot 0,08 = 1936 \text{ грн,}$$

де $K_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot K_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0,8 = 1491,2 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році,

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

$Д_Р$ – кількість днів планових ремонтів устаткування,

t –кількість робочих годин в день,

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

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1491,2 \cdot 0,2 \cdot 0,2 \cdot 5,22 = 311,36 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу,

$К_З$– коефіцієнтом зайнятості приладу,

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

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0,67 = 22000 \cdot 0,67 = 14740 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

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

$$С_{ЕКС} = 63360 + 13939,2 + 6050 + 1936 + 311,36 + 14740 = 100336,56 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 91262,82 \ / \ 1584,4 = 67,29 \text{ грн/год.}$$

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

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T,$$

I. $C_{\text{М}} = 67{,}29 \cdot 1046{,}4 = 70407{,}85$ грн,

II. $C_{\text{М}} = 67{,}29 \cdot 1063{,}28 = 71543{,}63$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{\text{Н}} = C_{\text{ЗП}} \cdot 0{,}67,$$

I. $C_{\text{Н}} = 169417{,}14 \cdot 0{,}67 = 113509{,}49$ грн,

II. $C_{\text{Н}} = 172150{,}1 \cdot 0{,}67 = 115340{,}56$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}},$$

I. $C_{\text{ПП}} = 169417{,}14 + 37271{,}77 + 70407{,}85 + 113509{,}49 = 390606{,}26$ грн,

II. $C_{\text{ПП}} = 172150{,}1 + 37873{,}02 + 71543{,}63 + 115340{,}56 = 396907{,}31$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{Кj}} / C_{\text{Фj}},$$

$$К_{ТЕР1} = 5,1 / 390606,26 = 7,5012 \cdot 10^{-6},$$

$$К_{ТЕР2} = 4,62 / 396907,31 = 6,5758 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 7,5012 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 7,5012 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;
– використаний фреймворк – scikit-learn;
– платформа для виконання коду – PyCharm.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

У цьому розділі проведено повний функціонально-вартісний аналіз програмного продукту. Також знайдено оцінку основних функцій програмного продукту. У результаті виконання функціонально-вартісного аналізу програмного комплексу, що розроблюється, визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

Робота досліджує використання моделей машинного навчання для прогнозування споживчого попиту. Використання машинного навчання для цього завдання показало високу ефективність, редукція не вплинула суттєво на результат.

Проведено огляд літератури в якій йдеться про існуючі методи передбачення часових рядів. Опрацьовано джерела, які допомогли дослідити рішення задачі реалізовані іншими дослідниками. Сформовано постановку задачі, досліджено методи її вирішення.

Розглянуто найпопулярніші методи проектування системи розробки програмного продукту. В результаті їх дослідження обраний спосіб планування та реалізації застосунку, який вирішить поставлену задачу.

Досліджено популярні інструменти розробки програмного продукту. Обрано мову програмування, бібліотеку та середовище виконання. Розроблено програмний продукт, який вирішує поставлену задачу, демонструє результати, дозволяє проаналізувати різні методи рішення.

Проведено функціонально-вартісний аналіз, за допомогою оцінки параметрів різних варіантів програмного продукту та економічного аналізу, обрано інструменти його розробки.

Ця робота підтверджує потенціал використання методів машинного навчання для прогнозування часових рядів, в тому числі споживчого попиту. Реалізований програмний продукт може бути використаний в системі прийняття рішень аналітиками, виробниками та спеціалістами з логістики.

ПЕРЕЛІК ПОСИЛАНЬ

1. Хардесті Л. Пояснення: нейронні мережі. *Офіс новин MIT*. 2017.
2. Ян З. Комплексна біомедична фізика. *Каролінський інститут.* Швеція: Elsevier, 2014. С. 1.
3. Єпископ С. М. Розпізнавання образів і машинне навчання. Нью-Йорк: Springer, 2006.
4. Вапник В. Н., Вапник В. Н. (1998). Природа статистичної теорії навчання. *Берлін Гейдельберг.* Нью-Йорк: Springer, 1998. Вип. 2.
5. Гудфеллоу Я., Бенгіо Й., Курвіль А. Глибоке навчання. 16 квітня 2016 р. *MIT Press*.
6. Феррі К., Кайзер С. Нейронні мережі для дітей. *Джерельні книги*. 2019.
7. Менсфілд М. Список робіт, що стосуються методу найменших квадратів.
8. Шмідхубер Дж. Анотована історія сучасного ШІ та глибокого навчання. 2022.
9. Шмідхубер Дж. Формальна теорія творчості, веселощів і внутрішньої мотивації (1990-2010). *Транзакції IEEE щодо автономного психічного розвитку*. 2010. Вип. 2 (3). С. 230-247. DOI: 10.1109/TAMD.2010.2056368. S2CID 234198.
10. Шмідхубер Дж. Генеративні суперницькі мережі є особливими випадками штучної цікавості (1990), а також тісно пов’язані з мінімізацією передбачуваності (1991). *Нейронні мережі*. 2020. Вип. 127. С. 58-66.
11. Шмідхубер Дж. Глибоке навчання в нейронних мережах: огляд. *Нейронні мережі*. 2015. Вип. 61. С. 85-117. DOI:10.1016/j.neunet.2014.09.003.
12. Як біоінспіроване глибоке навчання продовжує перемагати на змаганнях. URL: www.kurzweilai.net .
13. Грейвс А., Шмідхубер Дж. (2009). Офлайн-розпізнавання рукописного тексту за допомогою багатовимірних рекурентних нейронних мереж / ред.

Koller D., Schuurmans D., Bengio Y., Bottou L. (eds.). *Фонд нейронних систем обробки інформації (NIPS)*. С. 545-552.

14. Грейвс А., Лівіцкі М., Фернандес С., Бертоламі Р., Бунке Х., Шмідхубер Дж. Нова коннекціоністська система для необмеженого розпізнавання рукописного тексту. *Транзакції IEEE щодо аналізу шаблонів і машинного інтелекту*. 2009. Вип. 31 (5). С. 855-868. DOI: 10.1109/tpami.2008.137. ISSN 0162-8828.
15. Ciresan D., Meier U., Schmidhuber J. Глибокі нейронні мережі з кількома стовпцями для класифікації зображень. *Конференція IEEE 2012 з комп'ютерного зору та розпізнавання образів*. 2012. С. 3642-3649. DOI:10.1109/cvpr.2012.6248110.
16. Billings S. A. Ідентифікація нелінійної системи: методи NARMAX у часовій, частотній та просторово-часовій областях. 2013.
17. Гудфеллоу І., Пуже-Абаді Дж., Мірза М., Сю Б., Уорд-Фарлі Д., Озаір С. та ін.. Нейронні систем обробки інформації. *Generative Adversarial Networks*. 2014. С. 2672-2680.
18. Готуйся, не панікуй: синтетичні медіа та дипфейки. URL: свідок.org. (дата звернення: 25.11.2020).
19. GAN 2.0: гіперреалістичний генератор обличчя NVIDIA. URL: SyncedReview.com.
20. Karras T., Aila T., Laine S., Lehtinen J. Поступове зростання GAN для покращення якості, стабільності та варіації. 2017.
21. Шрівастава Р. К., Грефф К., Шмідхубер Дж. Магістральні мережі. 2015.
22. Срівастава Р. К., Грефф К., Шмідхубер Дж. (2015). Навчання дуже глибоких мереж. *Досягнення в нейронних системах обробки інформації*. 2015. Вип. 28. С. 2377-2385.
23. He K., Zhang X., Ren S., Sun J. Глибоке залишкове навчання для розпізнавання зображень. *Conference on Computer Vision and Pattern Recognition (CVPR)*. США: IEEE, 2016. С. 770-778. DOI:10.1109/CVPR.2016.90.

24. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N. та ін. Увага - це все, що вам потрібно». 2017.
25. Wolf T., Debut L., Sanh V., Chaumond J., Delangue C., Moi A. та ін. Трансформери: найсучасніша обробка природної мови. Матеріали конференції 2020 року з емпіричних методів обробки природної мови: системні демонстрації. 2020. С. 38-45. DOI:10.18653/v1/2020.emnlp-demos.
26. Не С. Трансформер в CV. *Трансформер в CV*. *Назустріч Data Science*. 2021.
27. Рамезанпур А., Бім А.Л., Чен Дж. Х., Машагі А. Статистична фізика для медичної діагностики: алгоритми навчання, висновків та оптимізації. *Діагностика*. 2020. Вип. 10. С. 972.
28. Miljanovic M. Порівняльний аналіз нейронних мереж із рекурентною та кінцевою імпульсною відповіддю в прогнозуванні часових рядів. *Індійський журнал комп'ютерів та техніки*. 2012. Вип. 3 (1).
29. Lau S. Покрокове керівництво згорткової нейронної мережі - налаштування гіперпараметрів. *Середній*. 2023.
30. Kelleher J. D., Mac N. B., D'Arcy A. Основи машинного навчання для прогнозної аналітики даних: алгоритми, робочі приклади та приклади (2-е видання). Массачусетс: The MIT Press. 2020.
31. Wei J. Забудьте про швидкість навчання, втрати затухання. 2019.
32. Li Y., Fu Y., Li H., Zhang S. W. Покращений алгоритм навчання зворотного поширення нейронної мережі з самоадаптивною швидкістю навчання. *Міжнародна конференція з обчислювального інтелекту та природних обчислень*. 2009. Т. 1. С. 73-76. DOI:10.1109/CINC.2009.111.
33. Huang G. B., Zhu Q. Y., Siew C. K. Екстремальна навчальна машина: теорія та застосування. *Нейрокомп'ютинг*. 2006. Вип. 70 (1). С. 489-501. DOI: 10.1016/j.neucom.2005.12.
34. Widrow B. та ін. Алгоритм без пропозицій: новий алгоритм навчання для багатошарових нейронних мереж. *Нейронні мережі*. 2013. Вип. 37. С. 182-188. DOI:10.1016/j.neunet.

35. Ollivier Y., Charpiat G. Навчання рекурентних мереж без зворотного треку. 2015.
36. Хінтон G. E. Практичний посібник із навчання обмежених машин Больцмана. 2021.
37. Бернард Е. Введення в машинне навчання. Шампань: Wolfram Media. 2021. С. 9.
38. Бернард Е. Введення в машинне навчання. Шампань: Wolfram Media. 2021. С. 12.
39. Бернард Е. Вступ до машинного навчання. Wolfram Media Inc. 2021. С. 9.
40. Ojha V. K., Abraham A. , Snášel V. Метаевристичний дизайн прямої нейронної мережі: огляд двох десятиліть досліджень. *Інженерні програми штучного інтелекту*. 2017. Вип. 60. С. 97 – 116. DOI:10.1016/j.engappai.2017.01.013.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt


def pred(model, title):
    Xp = [26.6100, 26.5722, 26.4244, 26.3503]
    y_pred = [26.3314]
    Xp = np.reshape(Xp, (1, -1))
    predictions = model.predict(Xp)
    print('\n', title)
    print('Prediction data: [26.6100, 26.5722, 26.4244,
26.3503]')
    print('Needed result: [26.3314]')
    print('Result: ', predictions)
    print('MSE:  %.8f'  %  mean_squared_error(y_pred,
predictions))
    print('Weights', model.coef_)


def do_charts_values(model_unr, model_r):
    Xp = [[26.6100, 26.5722, 26.4244, 26.3503],
          [27.9783, 28.0087, 28.0096, 28.0576],
          [26.6931, 26.6457, 26.6752, 26.6455],
```

```
                [26.9290, 26.8601, 26.8928, 27.0029],
                [26.9298, 26.8102, 26.7317, 26.7452]
                ]
        y_pred = [26.3314, 28.0642, 26.6504, 27.0247, 26.7264]
        predicts_unr = []
        predicts_r = []
        i = 0
        for el in Xp:
            el = np.reshape(el, (1, -1))

predicts_unr.append(model_unr.predict(el).tolist()[0][0])
        for elem in Xp:
            elem = np.reshape(elem, (1, -1))

predicts_r.append(model_r.predict(elem).tolist()[0][0])

        fig = plt.figure(figsize=(10, 10))
        ax1 = plt.subplot()
        ax1.plot((1,  2,  3,  4,  5),  (y_pred),  label='Real
values')
        ax1.legend()
        ax2 = plt.subplot()
        ax2.plot((1,   2,   3,   4,   5),   (predicts_unr),
label='Regular values')
        ax2.legend()
        ax3 = plt.subplot()
        ax3.plot((1, 2, 3, 4, 5), (predicts_r), label='Reduced
values')
        ax3.legend()
```

```
        plt.title('Compare values of predictions')
        plt.show()


    def do_charts_MSE(model_unr, model_r):
        Xp = [[26.6100, 26.5722, 26.4244, 26.3503],
              [27.9783, 28.0087, 28.0096, 28.0576],
              [26.6931, 26.6457, 26.6752, 26.6455],
              [26.9290, 26.8601, 26.8928, 27.0029],
              [26.9298, 26.8102, 26.7317, 26.7452]
              ]
        y_pred = [26.3314, 28.0642, 26.6504, 27.0247, 26.7264]
        predicts_unr = []
        predicts_r = []
        i = 0
        j = 0

        for el in Xp:
            el = np.reshape(el, (1, -1))

predicts_unr.append(mean_squared_error([y_pred[i]],
model_unr.predict(el)))
            i += 1
        for elem in Xp:
            elem = np.reshape(elem, (1, -1))
            predicts_r.append(mean_squared_error([y_pred[j]],
model_r.predict(elem)))
            j += 1
```

```
        print(predicts_unr)
        print(predicts_r)
        fig = plt.figure(figsize=(10, 10))
        ax2 = plt.subplot()
        ax2.plot((1,   2,   3,   4,   5),   (predicts_unr),
label='Regular NN MSE values')
        ax2.legend()
        ax3 = plt.subplot()
        ax3.plot((1, 2, 3, 4, 5), (predicts_r), label='Reduced
NN MSE values')
        ax3.legend()
        plt.title('Compare MSE of predictions')
        plt.show()


    def reduction(mdl, sample_weights):
        model = mdl
        Xp = [26.6100, 26.5722, 26.4244, 26.3503]
        y_pred = [26.3314]
        Xp = np.reshape(Xp, (1, -1))
        predictions = model.predict(Xp)
        etalon_MSE = mean_squared_error(y_pred, predictions)
        current_mse = mean_squared_error(y_pred, predictions)
        for i in range(36):
            temp = sample_weights[i]
            sample_weights[i] = 0
            model.fit(x.values, y.values, sample_weight)
            prediction_test = model.predict(Xp)
```

```
            temp_mse = mean_squared_error(y_pred,
prediction_test)
            if temp_mse > current_mse:
                sample_weights[i] = temp
            else:
                current_mse = temp_mse
        return model


    df = pd.read_csv("prices_month_ds.csv")
    x = df.iloc[:, :4]
    y = df.iloc[:, 4:5]

    model_1 = LinearRegression()
    model_1.fit(x.values, y.values)
    pred(model_1, 'Non-weighted model')

    sample_weight = np.ones(36) * 20
    model_2 = LinearRegression()
    model_2.fit(x.values, y.values, sample_weight)
    pred(model_2, 'Weighted model')

    model_3 = reduction(model_2, sample_weight)
    pred(model_3, 'Reduced model')

    do_charts_values(model_1, model_3)
    do_charts_MSE(model_1, model_3)
```