Національний університет «Чернігівський колегіум» імені Т.Г.Шевченка

Природничо-математичний факультет

Кафедра інформатики і обчислювальної техніки

Кваліфікаційна робота

освітнього ступеня «бакалавр»

на тему

КЛАСИФІКАЦІЯ ДАНИХ ЗАСОБАМИ СУЧАСНИХ PYTHON-ФРЕЙМВОРКІВ

Виконав:

студент 4 курсу, 44фм групи

спеціальності

122 Комп’ютерні науки

Ісамбаєв Павло Валерійович

Науковий керівник:

д.п.н., проф. Горошко Ю.В.

Чернігів – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

англ. – англійська
БД – база даних
год. – година
КІС – корпоративна інформаційна система
теор. – теоретичний
ІС – інформаційна система
СШІ – система штучного інтелекту
ML - Machine Learning — машинне навчання
DL - Deep learning - глибоке навчання
AI - Artificial Intelligence - штучний інтелект
API – application programming interface
ОПМ – обробка природної мови

ВСТУП

Python швидко став однією з найкращих мов програмування головним чином завдяки своїй простоті, можливості застосування та багатству ресурсів для тих, хто її вивчає. Це мова, яку широко використовують від розробки веб-сайтів до аналізу даних.

Однак розуміння того, як кодувати на Python, полягає не лише у створенні коду. Одним із найпоширеніших інструментів, які використовують програмісти на Python, є фреймворк, термін, який описує інструмент, який допомагає в процесі розробки.

Мета фреймворку полягає в тому, щоб спростити програмування певною мовою або створювати певні типи програм. Таким чином, можемо мати одну структуру, яка полегшує швидку розробку, а інша забезпечує надійну базу веб-додатків.

Фреймворк (англ. framework - каркас, структура) у мовах програмування – це комплекс компонентів, бібліотек та інструментів, які пропонують структуру та готові рішення для роботи над певними завданнями. Або дають вирішення певних проблем у розробці програмного забезпечення. Вони прискорюють розробку, покращують якість коду та стійкість додатків.

Python-фреймворки, такі як scikit-learn, TensorFlow, Keras та PyTorch, пропонують потужні інструменти для машинного навчання та класифікації даних. Вони дозволяють будувати моделі різної складності, від простих лінійних класифікаторів до глибоких нейронних мереж. Крім того, ці фреймворки забезпечують зручні засоби для попередньої обробки даних, візуалізації результатів та оптимізації моделей.

Для даної роботи було вирішено створити модель для машинного навчання про виявлення шахрайства з кредитними картками.

Метою даної кваліфікаційної роботи є дослідження сучасних Python-фреймворків для класифікації даних, аналіз їх можливостей та порівняння ефективності у вирішенні різноманітних задач. У роботі буде розглянуто основні концепції класифікації, алгоритми та методи, що використовуються у

найпопулярніших Python-фреймворках. Також буде проведено експериментальне дослідження з використанням реальних наборів даних для демонстрації переваг та недоліків кожного з фреймворків.

Для навчання та оцінки моделей класифікації у даній роботі вибрано Credit Card Fraud Detection Dataset 2023. Цей набір даних містить інформацію про транзакції з кредитних карток та мітки шахрайських операцій. Використання даного датасету дозволить перевірити здатність різних фреймворків виявляти шахрайські операції, що є важливим завданням у сфері фінансової безпеки.

Потрібно реалізувати такі завдання:

- дослідити доступні фреймворки на Python для машинного навчання;
- датасет та розділення на навчальну та тестову вибірки;
- побудувати та налаштувати моделі класифікації за допомогою фреймворків TensorFlow, XGBoost та PyTorch;
- навчання моделей на тестовій виборці на фреймворках;
- випробувати тестову вибірку та підрахунок метрики для порівняння.

Реалізація цих завдань дозволить оцінити можливості сучасних Python-фреймворків для класифікації даних та їх застосування для виявлення шахрайства з кредитними картками.

РОЗДІЛ 1. ОГЛЯД РІШЕНЬ

1.1. Аналіз методів машинного навчання

Машинне навчання (англ. machine learning) — це підгалузь штучного інтелекту (ШІ), яка фокусується на розробці алгоритмів і моделей, що дозволяють комп'ютерам навчатися та робити прогнози або приймати рішення на основі даних. Машинне навчання дозволяє системам автоматично вдосконалюватися з досвідом без явного програмування для кожного конкретного завдання.

У традиційному програмуванні під час виконання завдання комп’ютер слідує заздалегідь визначеним інструкціям. Однак у машинному навчанні системі дається набір прикладів, за допомогою яких їй необхідно з’ясувати, як розв’язати проблему. Головна мета ML - виключення ручних перевірок, щоб автоматизувати роботу. У процесі розвитку інновації, машини не просто навчаються, а й можуть запам'ятовувати конкретні дії, надаючи правильніші відповіді та варіанти для ухвалення рішень.

Ефективність моделей безпосередньо залежить від кількості та якості навчальних даних.

У XXI столітті складно переоцінити машинне навчання. Воно є двигуном, що приводить світ у рух. Зі збільшенням кількості інформації у світі традиційні методи аналізу стали менш ефективними. Алгоритми ML можуть обробляти і розуміти величезні масиви даних, виявляти в них приховані закономірності і надавати необхідні відомості для прийняття рішень.

Завдяки серії інновацій машинне навчання широко використовується в таких галузях, як опрацювання природної мови та комп’ютерний зір. Успіху сприяли велика кількість тренувальних даних і доступність величезних потужностей для паралельних обчислень за допомогою сучасних процесорів.

Кожен пошуковий запит у Google запускає одразу кілька ML -моделей на кшталт розпізнавання тексту та персоналізації видачі результатів. [1] Так само працює і система виявлення спаму в Gmail, визначаючи шахрайські повідомлення. В онлайн-магазинах технології на базі машинного навчання здатні пе-

редбачити, який продукт користувач захоче купити наступного разу або які треки йому можуть сподобатися на Spotify.

ML -системи знаходять застосування в безлічі галузей. Наприклад, вони допомагають безпілотним автомобілям виявляти пішоходів. Також алгоритми машинного навчання використовують для розпізнавання облич, виявлення пухлин на рентгенівських знімках, опрацювання природної мови чат-ботами і виконання безлічі інших завдань.[7]

Історія машинного навчання є складною та багатогранною, розпочинаючи з основ теорії обчислень і проходячи через численні етапи розвитку алгоритмів, методів та застосувань.

У 1943 році Уолтер Піттс та Уоррен Маккалох у своїй науковій роботі «Логічне обчислення ідей, іманентних нервовій діяльності» представили першу математичну модель нейронних мереж.

У 1949 році Дональд Хебб випустив книгу «Організація поведінки» у якій описав теорію, як поведінка пов'язана з нейронними мережами та активністю мозку. Вона стала одним з монументальних стовпів розвитку машинного навчання.

У 1950 році Алан Тюрінг створив «Тест Тюрінга», щоб визначити, чи має комп'ютер справжній інтелект. Щоб пройти тест, машина мала обдурити людину, змусивши її повірити, що вона також є людиною.

В 1957 Френк Розенблатт розробив перцептрон — перший алгоритм навчання на основі штучної нейронної мережі.

Термін «машинне навчання» з’явився 1959 року. Його ввів піонер у галузі ШІ Артур Самуель.

Фахівець визначив ML як процес, унаслідок якого комп’ютери здатні показати поведінку, що не запрограмована в них від початку.

Раніше Самуель розробив першу самонавчальну систему з гри в шашки. Її продемонстрували публіці 24 лютого 1956 року. Програма аналізувала партії і вивчала ходи, що складають виграшну стратегію. Вона використовувала ці дані в наступних матчах. [24]

У 1962 році майстер із гри в шашки Роберт Нілі змагався із програмою Семуеля. Самонавчальна система виграла тільки першу партію, але стала монументальним технологічним досягненням того часу.

Наразі великої популярності набуває Machine Learning Cloud. Річ у тому, що навчання точної моделі ML вимагає великих обсягів даних, обчислювальних потужностей та інфраструктури. Хмарні обчислення роблять машинне навчання більш доступним, гнучким і економічно ефективним, дозволяючи розробникам швидше створювати алгоритми машинного навчання. Саме ML Cloud допомагає прискорити та керувати всім життєвим циклом ML-проєкту.[20]

Основні концепції машинного навчання

1. Модель:

   Це математичне представлення процесу, який ми намагаємося моделювати. Модель будується на основі даних і використовується для прогнозування або прийняття рішень. Приклади моделей включають лінійну регресію, рішення дерева, нейронні мережі тощо.

2. Алгоритм навчання:

   Це набір правил і методів, за допомогою яких модель навчається з даних. Різні алгоритми використовуються для різних типів задач. Алгоритми можуть бути різними в залежності від типу навчання (контрольоване, неконтрольоване, підкріплювальне) і задачі (класифікація, регресія, кластеризація тощо).

3. Дані:

   Дані є основою для навчання моделей. Це можуть бути числові дані, текст, зображення, відео тощо.

Ключовим елементом успіху у машинному навчанні є якість вхідних даних. Підготовка даних включає в себе декілька критично важливих етапів, які забезпечують точність і надійність отриманих моделей.

Перший крок у підготовці даних — це їх збір. Дані можуть походити з різних джерел, таких як бази даних, файлові системи, зовнішні API або навіть ручний ввід. Важливо забезпечити, щоб дані були релевантні, достатньо об’ємні

та представляли всі можливі сценарії, з якими модель може зіткнутися під час використання.

Після збору даних наступний крок – очищення. Це включає виправлення помилок у даних, видалення дублікатів, опрацювання відсутніх значень і нормалізацію даних для забезпечення їхньої консистенції. Наприклад, можна замінити відсутні значення середніми чи медіанами або використовувати алгоритми для виявлення та виправлення аномалій у даних.

Для ефективного навчання моделей необхідно правильно розділити дані на набори: тренувальний, тестовий і, за потреби, валідаційний. Тренувальний набір використовується для навчання моделі, тестовий — для перевірки її ефективності, а валідаційний (якщо є) допомагає в налаштуванні параметрів моделі та запобіганні перенавчанню. Правильне розділення забезпечує, що модель буде добре працювати не тільки на відомих даних, але й у здатність генералізувати на нових, раніше невідомих даних.[2]

Типи машинного навчання:

1. Контрольоване навчання (Supervised Learning):

- Модель навчається на маркованих даних, тобто на наборах даних, які містять вхідні дані та відповідні мітки або цільові значення. Цей метод навчання застосовується у випадках, коли є великі обсяги даних, припустимо тисячі фотографій домашніх тварин з маркерами (мітками, ярликами): це кішка, а це собака. Необхідно створити алгоритм, за допомогою якого машина могла б по фотографії, яку «не бачила» раніше, визначити, хто на ній зображений: кішка або собака. У ролі «вчителя» в даному випадку виступає людина, яка заздалегідь проставила маркери. Машина сама вибирає ознаки, за якими вона відрізняє кішок від собак. Тому в подальшому знайдений нею алгоритм може бути швидко переналаштований на рішення іншої задачі, наприклад, на розпізнавання курей і качок. Машина сама виконає складну роботу по виділенню ознак, за якими буде розрізняти цих птахів. А нейромережа, яку навчили розпізнавати кішок, можна швидко навчити обробляти результати комп’ютерної томографії.

- Приклади: класифікація (розпізнавання спаму в електронній пошті), регресія (прогнозування цін на нерухомість).

2. Неконтрольоване навчання (Unsupervised Learning):
    - Модель навчається на немаркованих даних, тобто на наборах даних, які містять тільки вхідні дані без відповідних міток. Хоча маркованих, розмічених даних накопичилося вже досить багато, даних без маркерів (міток) все ж набагато більше. Є зображення без підписів, аудіозаписи без коментарів, тексти без анотацій. Завдання машини при неконтрольованому навчанні – знайти зв’язку між окремими даними, виявити закономірності, підібрати шаблони, упорядкувати дані або описати їх структуру, виконати класифікацію даних.
    - Приклади: в рекомендаційних системах, коли в інтернет-магазині на основі аналізу попередніх покупок покупцеві пропонуються товари, які можуть зацікавити його з більшою ймовірністю, ніж інші. Або коли після перегляду відеокліпу відвідувачу пропонують десятки посилань на ролики, чимось схожі на переглянутий.
3. Напівконтрольоване навчання (Semi-Supervised Learning):
    - Поєднання контрольованого та неконтрольованого навчання. Модель навчається на наборі даних, який містить як марковані так і немарковані дані. Решту розмітки виконує сам алгоритм за заданими параметрами. Таке навчання корисне для обробки об’ємних файлів.
4. Підкріплювальне навчання (Reinforcement Learning):
    - Модель навчається шляхом взаємодії з середовищем і отримання зворотного зв'язку у вигляді винагород або покарань. Таке навчання є окремим випадком контрольованого навчання, але вчителем в даному випадку є «середовище». Машина (її в цій ситуації часто називають «агент») не має попередньої інформації про середовище, але має можливість здійснювати в ній будь-які дії. Середовище реагує на ці дії і тим самим надає агенту дані, які дозволяють йому реа-

гувати на них і вчитися. Фактично агент і середовище утворюють систему зі зворотним зв’язком.

- Приклади: управління роботами, які навчаються уникати зіткнень з перешкодами шляхом набуття досвіду, отримуючи зворотній зв’язок при кожному зіткненні. Використовується також в логістиці, при складанні графіків і плануванні завдань, при навчанні машини логічним іграм (покер, нарди, шахи і ін.)

На одній з останніх конференції компанія Gartner спільно з найкращими аналітиками обговорювали основні тенденції технологічного сегмента з використанням машинного навчання, які будуть спрямовані на суттєві економічні та технологічні зміни. Основні з них:

1. "Креативний" штучний інтелект- генеративний ШІ з методиками машинного навчання, що можуть навчати нового та аналізувати будь-яку інформацію. На додаток машини виконуватимуть декомпозицію об'єктів, формуючи нові сутності. Подібна технологія може застосовуватися для програмних кодів, які використовуються при створенні ліків, у маркетингу. Цей інструмент може бути чудовим рішенням для політичної дезінформації в майбутньому.
2. Розподілене підприємство (Distributed Enterprise) - тенденція, яка набрала популярності під час пандемії, коли народилася гібридна модель роботи традиційних офісних компаній з віддаленими робітниками. За очікуваннями Gartner, 75% фірм зможуть на 25% прискорити приплив прибутковості за допомогою розподіленого підприємства, як порівняти з компаніями старого зразка. Багато в чому допомагає освоєнню процесу ШІ і машинне навчання.[6]
3. Автономні системи - платформи програмного типу або ті, які мають фізичне самоврядування з механікою самонавчання. Якщо порівнювати з автоматизованими системами сьогодні, то автономні платформи можуть динамічно адаптувати власні алгоритми до конкретних умов без оновлення ПЗ. Класичні моделі інструментів програмуван-

ня, звичайна автоматизація, зі зростанням компаній у світі, не дають змоги масштабуватися потрібними темпами, яких потребує бізнес. Автономні системи з машинним навчанням повністю вирішують проблему.

4. Гіперавтоматизація - подібна інновація дає швидке зростання для бізнесу, його стійкість. Ці можливості досягаються за допомогою високої швидкості визначення, перевірки, аналітики та автоматизації безлічі складних процесів, які неможливі без машинного навчання та ШІ.
5. Складові додатки - технологічні платформи, що дають змогу швидко адаптувати будь-які бізнес-процеси. Вони дають безпеку, ефективність за будь-якого виду змін ринку. Архітектура складових застосунків розв'язує проблеми, і, за даними Gartner, підприємства, що освоїли Composable Applications, можуть на 80% випередити будь-якого конкурента за швидкістю впровадження нового функціоналу.
6. Мережа кібербезпеки - повноцінна концепція, що дає змогу захищати будь-які цифрові активи, дані компанії, незалежно від їхнього розташування. Вважається, що у 2024 році кількість підприємств, які використовують кібербезпеку, можуть зменшити фінансові втрати від окремих ситуацій на 90%.

Машинне навчання є потужним інструментом, який дозволяє створювати інтелектуальні системи, що можуть автоматично вдосконалюватися з часом. Завдяки широкому спектру методів і застосувань, машинне навчання стає невід'ємною частиною сучасних технологій і бізнес-процесів.

Розвиток технологій і впровадження новаторських підходів у різних галузях стимулює численні дослідження у сфері машинного навчання. Такі дослідження проходять як в сфері торгівлі, так і в освіті. Розвиток сучасних інтелектуальних систем електронної продажів потребує точного та швидкого розв’язку задач прогнозування. Традиційний пошук товару чи послуги в Інтернеті вже не задовольняє розвиток компаній. Тож на заміну приходить

персоналізована система, яка має ряд переваг - збільшення рівня продажів на основі вчасного та правильного визначення потенційних потреб споживача і, як наслідок, збільшення прибутку. Це стає можливим із використанням ефективної моделі побудови прогнозів на основі низки характеристик того чи іншого споживача. Наприклад, проведено моделювання роботи методів машинного навчання для прогнозування суми витрат споживачів роздрібного магазину. [23]

Розглянемо використання фреймворків для машинного навчання в області обробки природної мови (NLP).

Дослідження в області обробки природної мови (ОПМ) фокусуються на розробці алгоритмів та моделей, які дозволяють комп'ютерам розуміти, інтерпретувати та генерувати людську мову. Це міждисциплінарна галузь, що об'єднує лінгвістику, комп'ютерні науки, штучний інтелект та когнітивні науки. Ось кілька основних напрямків досліджень у цій сфері:

1. Морфологічний аналіз:

- Вивчення структури слів та їх частин (морфем), що дозволяє комп'ютеру розпізнавати й інтерпретувати різні форми слова.

2. Синтаксичний аналіз:

- Визначення граматичної структури речень. Це включає розпізнавання частин мови (іменники, дієслова, прикметники тощо) і їх взаємозв'язків у реченні.

3. Семантичний аналіз:

- Визначення значення слів і речень. Ця область досліджень охоплює питання полісемії (множинні значення слів) і синонімії (слова з подібними значеннями).

4. Прагматичний аналіз:

- Дослідження того, як контекст впливає на інтерпретацію мовних виразів. Це включає аналіз значення, що залежить від ситуації, і мовних актів (наприклад, запитів, обіцянок, наказів).

5. Розпізнавання мовлення:

- Технології, що дозволяють комп'ютерам перетворювати усну мову в

текст. Це включає виявлення та обробку акустичних сигналів, фонетичний аналіз і моделювання мовних зразків.

6. Генерація природної мови (NLG):

- Створення тексту, що є зрозумілим і природним для людини, на основі даних або іншої інформації. Цей напрямок включає побудову текстових структур, вибір лексики і стилю.

7. Машинний переклад:

- Автоматичний переклад текстів з однієї мови на іншу з використанням алгоритмів та моделей, що можуть враховувати як лексичні, так і синтаксичні особливості мов.

8. Аналіз тональності (Sentiment Analysis):

- Визначення емоційного забарвлення тексту, наприклад, чи є відгук про продукт позитивним чи негативним.

Дослідження в NLP активно розвиваються завдяки досягненням в галузі машинного навчання, зокрема, використанню нейронних мереж і глибинного навчання, що дозволяють створювати більш точні та ефективні моделі для роботи з природною мовою.

Серед найновіших досягнень варто відзначити кілька ключових напрямків:

1. Синтаксичний аналіз і морфологія: Ці напрямки включають дослідження, спрямовані на аналіз структури речень та морфологічних характеристик слів у різних мовах, що дозволяє покращити розуміння текстів і їх автоматичну обробку.
2. Перенесення знань між мовами: Дослідження в цій області націлені на розробку моделей, здатних переносити знання з однієї мови на іншу, що є особливо корисним для багатомовних додатків і перекладацьких систем.
3. Аудіо та мовні технології: Розробка методів обробки мовлення та аудіо, таких як розпізнавання мови та синтез мовлення, які значно покращують взаємодію людини з машинами.
4. Інтерпретованість та пояснюваність моделей: Цей напрямок досліджень фокусується на розробці методів, які роблять моделі NLP більш прозори-

ми та зрозумілими для користувачів, що є важливим для довіри до систем, які приймають важливі рішення на основі текстових даних.

5. Робастність моделей: Дослідження в цій галузі спрямовані на створення моделей, які зберігають свою ефективність навіть у складних та непередбачуваних умовах, таких як наявність шуму в даних або атак на модель.
6. Використання великих мовних моделей: Такі моделі, як GPT-3 і ChatGPT, показали високу ефективність у виконанні різних завдань NLP, включаючи генерацію тексту, відповідь на запитання та переклад.

Дослідження також показують, що великий внесок у розвиток NLP роблять не лише академічні установи, але й великі технологічні компанії, які вкладають значні ресурси в розробку нових технологій та інфраструктури для обробки мовних даних.

На 9-й Міжнародній науково-технічній конференції, яка відбулася з 17 по 20 листопада 2020 року, була представлена доповідь про сучасні підходи та методи, що використовуються для розв'язання задач обробки природної мови (ОПМ). У доповіді було висвітлено наступні ключові аспекти:

1. Традиційні методи обробки природної мови

- Статистичні методи: Використання статистичних моделей, таких як моделі n-грам, для обробки тексту. Ці методи базуються на частотному аналізі слів та фраз у великих корпусах тексту.
- Машинне навчання: Застосування алгоритмів машинного навчання, таких як баєсівський класифікатор, метод опорних векторів (SVM) та регресія, для задач класифікації та регресії в обробці тексту.

2. Сучасні методи на основі глибинного навчання

- Рекурентні нейронні мережі (RNN): Використання RNN та їх модифікацій, таких як LSTM (довга короткочасна пам'ять) і GRU (гейтована рекурентна одиниця), для роботи з послідовностями тексту, що дозволяє ефективно моделювати контекст і залежності в тексті.

- Конволюційні нейронні мережі (CNN): Застосування CNN для задач обробки тексту, таких як класифікація тексту та аналіз тональності. CNN використовуються для виявлення локальних патернів у тексті.

3. Моделі на основі трансформерів

- BERT (Bidirectional Encoder Representations from Transformers): Опис архітектури BERT, що дозволяє моделювати текст у двох напрямках і забезпечує значне покращення результатів у багатьох задачах ОПМ, таких як питання-відповіді, аналіз тональності та розпізнавання сутностей.
- GPT (Generative Pre-trained Transformer): Огляд моделей GPT, зокрема GPT-2 і GPT-3, що є потужними генеративними моделями для створення тексту, перекладу та інших задач. Ці моделі демонструють високу ефективність завдяки великим масштабам попереднього навчання на величезних корпусах тексту.

4. Інші важливі методи та підходи

- Word Embeddings: Використання векторних представлень слів, таких як Word2Vec, GloVe та FastText, що дозволяють перетворювати слова в багатовимірні вектори, зберігаючи семантичні відносини між ними.
- Тонке налаштування моделей: Підхід до донавчання великих моделей на специфічних задачах або датасетах для досягнення кращих результатів у конкретних застосуваннях.
- Transfer Learning: Використання попередньо навчених моделей та їх адаптація до нових задач шляхом донавчання, що дозволяє економити обчислювальні ресурси і час на навчання.

5. Приклади застосувань ОПМ

- Машинний переклад: Вдосконалення методів машинного перекладу з використанням трансформерних моделей, що дозволяють досягати високої точності перекладу між різними мовами.
- Чат-боти та віртуальні асистенти: Використання ОПМ для створення інтерактивних чат-ботів та віртуальних асистентів, що можуть ефективно взаємодіяти з користувачами на природній мові.

- Аналіз соціальних мереж: Застосування ОПМ для аналізу текстів з соціальних мереж, виявлення настроїв, аналізу тенденцій та моніторингу громадської думки.

Доповідь підсумувала, що сучасні методи обробки природної мови значно покращилися завдяки розвитку глибинного навчання та моделей на основі трансформерів, що відкриває нові можливості для ефективного розв'язання складних задач в різних галузях. [25]

Використання машинного навчання в освіті дозволяє розробляти новаторські методи навчання, підвищувати ефективність адміністративних процесів і забезпечувати персоналізоване навчання для кожного учня.

Завдяки машинному навчанню в освіті тепер можливо створювати персоналізовані шляхи навчання. Якщо освітній заклад використовує освітнє програмне забезпечення на основі машинного навчання, система зможе розуміти унікальні потреби студентів і пропонувати зміни в їхньому навчальному шляху.

Ось три ключові переваги для галузі освіти:

Покращені результати навчання: Завдяки адаптивному навчанню з персоналізованими шляхами, учні можуть краще зрозуміти матеріал і усунути свої слабкі місця.

Індивідуалізовані рекомендації: Використовуючи додатки машинного навчання, заклад може забезпечити індивідуальний підхід до кожного учня без необхідності вручну аналізувати їхню успішність і розробляти плани. Уявіть уроки та завдання, що відповідають цілям, сильним і слабким сторонам кожного студента.

Підтримка різноманітних стилів навчання: Деякі учні навчаються швидше за інших. Завдяки персоналізованим шляхам навчання кожен стиль навчання задовольняється, що підвищує задоволеність студентів та збільшує успішність навчання. Багато освітніх платформ створюють індивідуальне навчання. Наприклад, Інститут фізики (Великобританія та Ірландія) використовує Scoilnet.

Якщо навчальний заклад викладає мови, алгоритми машинного навчання

можуть сприяти такій роботі. Ця технологія використовує обробку природної мови (НЛП) для розуміння мов і семантики, порівнюючи їх із реальними людськими голосами. Наприклад, всесвітньо відомий додаток для вивчення мов Duolingo використовує машинне навчання та НЛП для досягнення цих цілей.

Duolingo, наприклад, просить учнів говорити в мікрофон свого смарт-пристрою. Записаний голос аналізується штучним інтелектом, і учень отримує миттєвий зворотний зв’язок. Крім того, ці технології можуть застосовуватися не лише для мовлення, а й для письмових текстів. Використання автоматизованих інструментів, подібних до цього, у навчальному закладі може суттєво знизити витрати.

Ось три основні переваги використання машинного навчання на мовних платформах Edtech:

1. Технологія розпізнавання мовлення допомагає студентам і викладачам виявляти проблеми зі звуками. Вона автоматично позначає неправильну вимову та пропонує учневі спробувати ще раз, покращуючи навчання.

2. Програмне забезпечення на основі машинного навчання, використовуючи дані студентів, може допомогти створювати особисті профілі та забезпечувати персоналізоване навчання.

3. Зростання доступності НЛП та технології розпізнавання мови робить навчальні платформи більш інклюзивними, сприяючи відчуттю залученості студентів.[4]

В галузі прогнозування захворювань теж проводяться різноманітні дослідження, використовуючи метод машинного навчання. Ось кілька напрямків досліджень:

1. Аналіз медичних даних за допомогою машинного навчання: Дослідження спрямовані на використання алгоритмів машинного навчання для аналізу великих обсягів медичних даних, таких як історії хвороб, результати лабораторних досліджень, зображення з медичних обстежень тощо. Це дозволяє розробляти моделі для прогнозування ризику захворювань, діагностики ранніх стадій хвороб та планування лікування.

2. Генетичні дослідження: Дослідження генетичних факторів, що впливають на схильність до різних захворювань, і розробка моделей прогнозування ризику захворювань на основі генетичних даних. Це включає вивчення генетичних маркерів, пов'язаних з певними захворюваннями, і їх вплив на індивідуальний ризик.

3. Аналіз зображень медичних обстежень: Використання комп'ютерного зору та глибинного навчання для аналізу зображень з медичних обстежень, таких як комп'ютерна томографія (CT), магнітно-резонансна томографія (MRI) та рентгенівські знімки. Дослідження в цій області спрямовані на розробку систем автоматичної діагностики та прогнозування хвороб на основі зображень.

4. Застосування даних з побутових пристроїв для моніторингу стану здоров'я та раннього виявлення захворювань. Ці дані можуть включати інформацію про серцебиття, рухи та інші фізіологічні параметри.

5. Дослідження в області епідеміології: Вивчення поширення захворювань у великих популяціях та розробка моделей для прогнозування тенденцій епідемій і визначення факторів, що впливають на розповсюдження хвороб.

Ці напрямки досліджень спрямовані на розробку і вдосконалення методів та інструментів для прогнозування захворювань, що допомагає в ранньому виявленні та лікуванні хвороб, а також на управління популяційним здоров'ям.

Працюючи над темою «Машинне навчання для класифікації післяопераційного стану пацієнта з використанням стандартизованих медичних даних», дослідники пропонують метод аналізу та системи навчання здоров’ю, який дозволить визначити пріоритет клінічного втручання. Цей метод використовує кластеризацію та візуалізацію часових рядів, а також враховує пріоритетність результатів та статус пацієнта під час госпіталізації.

Було успішно підтверджено медичні процеси, зібравши високоякісні дані щодо статусу пацієнта, медичних цілей та результатів пацієнтів з різних установ. Це було складним завданням зі звичайними електронними медичними записами. Щодо важливих результатів аналізу, система навчання в галузі охорони здоров’я буде використовуватися в цьому проекті для надання зворотного зв’язку кожній

установі протягом певного періоду, а також для аналізу та переоцінки.[8]

Протягом останнього десятиліття методи машинного навчання (ML) активно використовувалися для діагностики різних захворювань, таких як хвороби нирок, рак шкіри, рак молочної залози, хвороби серця, а також для сегментації шару сітківки для діагностики хвороби Альцгеймера та раку простати. Проте, в області прогнозування хронічного загоєння ран було зроблено мало роботи. Звичайні алгоритми машинного навчання, такі як логістична регресія, підтримка векторної машини, дерева рішень і випадкові ліси, значно залежать від представлення ознак.

Тож було запущене дослідження «Синтез часових рядів факторів прогнозу рани з електронних медичних записів за допомогою генеративних змагальних мереж», щоб допомогти клініцистам вирішити, чи використовувати стандартний догляд чи ад'ювантну терапію, а також допомогти їм у плануванні клінічних випробувань. У рамках даного дослідження було розроблено часові ряди медичних генеративних змагальних мереж (GAN), щоб створити синтетичні фактори для прогнозування ран, використовуючи дуже обмежену інформацію, зібрану під час звичайного догляду в спеціалізованому закладі для лікування ран. Згенеровані прогнозні змінні використовуються для розробки прогнозної моделі траєкторії загоєння хронічної рани.

Згенеровані прогнозні змінні використовуються для розробки прогностичної моделі траєкторії загоєння хронічної рани. GAN може створювати як безперервні, так і категоричні характеристики з електронних медичних записів (EMB). До цього використовувалася часова інформація, зібрана під час щотижневого спостереження за пацієнтами, та стратегії умовного навчання для покращення навчання та отримання конфіденційних даних щодо загоєння або незагоєння. Для оцінки здатності запропонованої моделі генерувати реалістичні EMB дані використовувалися такі методи, як тест на синтетичному, навчання на реальному (TSTR), дискримінаційна точність та візуалізація. Згенеровані нашим GAN зразки були використані для навчання моделі прогнозу, що продемонструвало її ефективність у реальних умовах.

Використання згенерованих зразків у навчанні прогностичних моделей призвело до підвищення точності класифікації на 6,66–10,01% у порівнянні з попереднім EMR-GAN. Крім того, запропонований класифікатор прогнозу досягнув площі під кривою (AUC) 0,875, 0,810 і 0,647 під час навчання мережі з використанням даних перших трьох відвідувань, перших двох відвідувань і першого відвідування відповідно. Ці результати свідчать про значне покращення прогнозу загоєння рани в порівнянні з попередніми моделями прогнозу. [19]

У зв'язку з останньою пандемією вчені та лікарі активно шукають нові технології для зупинення або уповільнення пандемії COVID-19. Використання машинного навчання, як важливого аспекту штучного інтелекту, на основі минулих епідемій відкриває нові можливості для боротьби зі спалахом нового коронавірусу. Точний прогноз короткострокового поширення COVID-19 відіграє важливу роль у поліпшенні управління проблемою переповнення лікарень та дозволяє відповідну оптимізацію наявних ресурсів (тобто матеріалів і персоналу).

Було досліджено продуктивність методів глибокого навчання, включаючи гібридні згорткові нейронні мережі – довгострокова пам'ять (LSTM-CNN), гібридні нейронні мережі з рекурентними блоками – згортковими нейронними мережами (GAN-GRU), GAN, CNN, LSTM і обмежену машину Больцмана (RBM), а також базові методи машинного навчання, такі як логістична регресія (LR) і опорна векторна регресія (SVR). Використання гібридних моделей (тобто LSTM-CNN та GAN-GRU) повинно покращити точність прогнозування майбутніх тенденцій COVID-19. Продуктивність досліджених моделей глибокого навчання та машинного навчання була перевірена за допомогою даних часових рядів підтверджених та одужавших від COVID-19 в декількох країнах, що зазнали впливу: Бразилія, Франція, Індія, Мексика, Саудівська Аравія та США. Результати показали, що гібридні моделі глибокого навчання можуть ефективно прогнозувати випадки COVID-19. Крім того, результати підтвердили перевагу моделей глибокого навчання порівняно з двома

розглянутими базовими моделями машинного навчання. Крім того, результати показали, що LSTM-CNN досягла покращених показників з середньою абсолютною відсотковою похибкою 3,718%, серед інших. [18]

Не менш цікавим та корисним є застосування машинного навчання у сейсмології, що відкриває широкі можливості для покращення розуміння сейсмічних явищ, прогнозування землетрусів та вдосконалення заходів безпеки. Ось кілька способів, які використовують машинне навчання у цій галузі:

1. Прогнозування землетрусів: Машинне навчання використовується для аналізу сейсмічних даних та інших геологічних параметрів для прогнозування місця, часу та магнітуди майбутніх землетрусів. Алгоритми класифікації та регресії дозволяють створювати моделі, які можуть попереджати про можливість сейсмічних подій.

2. Виявлення аномалій: Машинне навчання допомагає виявляти аномальні патерни у сейсмічних даних, які можуть передувати землетрусам або іншим сейсмічним подіям. Це дозволяє вчасно реагувати та приймати заходи безпеки.

3. Класифікація сейсмічних подій: Алгоритми класифікації, такі як метод опорних векторів або глибокі нейронні мережі, допомагають ідентифікувати різні типи сейсмічних подій, такі як землетруси, вибухи або природні коливання.

4. Аналіз сейсмічних змін: Машинне навчання дозволяє аналізувати сейсмічні дані для виявлення змін у структурі земної кори, що може вказувати на геодинамічні процеси або підвищену сейсмічну активність.

5. Оптимізація мережі моніторингу: Машинне навчання допомагає оптимізувати розміщення сейсмічних станцій та інших приладів моніторингу для максимального охоплення території та мінімізації часу реакції на сейсмічні події.

Ці застосування машинного навчання в сейсмології допомагають розширити наше розуміння сейсмічних явищ та забезпечити ефективніші стратегії прогнозування та захисту від землетрусів.

Так ряд вчених працюючи над темою «Підхід машинного навчання для оцінки магнітуди землетрусу» пропонували швидкий та надійний метод

отримання кінцевої оцінки магнітуди землетрусу з необроблених сигналів, записаних на окремих сейсмічних станціях. Було розроблено регресор (MagNet), що складається з згорткових та рекурентних нейронних мереж, які не вимагають нормалізації даних, тому форма сигналу під час навчання може використовувати інформацію про амплітуду. Мережа може вивчити відстань-залежні та станція-залежні функції безпосередньо з навчальних даних. Така модель може прогнозувати локальні магнітуди з високою точністю, і середня похибка близька до нуля, а стандартне відхилення становить приблизно 0,2 на основі сигналів з однієї станції без корекції відповіді приладу. Протестовано мережу як для локальних, так і для масштабних тривалостей та показано, що навчання на рівні станцій може бути ефективним підходом до покращення продуктивності. Запропонований підхід має різноманітні потенційні застосування від звичайного моніторингу землетрусів до систем раннього попередження. [16]

Застосування машинного навчання у кібербезпеці є важливим і ефективним засобом для виявлення, запобігання та реагування на кіберзагрози. Ось деякі основні напрямки застосування машинного навчання у кібербезпеці:

1. Виявлення загроз та атак: Машинне навчання може бути використане для аналізу великого обсягу даних з метою виявлення незвичайних або підозрілих активностей в комп'ютерних мережах та системах. Це включає в себе виявлення вторгнень, зловмисних програм, фішингових атак тощо.

2. Прогнозування кіберзагроз: Машинне навчання може допомогти в передбаченні можливих кіберзагроз та виявленні вразливостей у системах кібербезпеки, що дозволяє приймати запобіжні заходи забезпечення безпеки.

3. Аналіз великих обсягів даних: Машинне навчання дозволяє аналізувати великі обсяги даних, зокрема журнали подій, мережевий трафік, системні логи тощо, для виявлення аномальних патернів та потенційних загроз.

4. Автоматизація відповіді на інциденти: Машинне навчання може бути використане для автоматизації процесів відповіді на кіберінциденти, включаючи ідентифікацію, класифікацію та вирішення проблем без участі

людини.

5. Виявлення нових загроз: Машинне навчання може допомогти виявляти нові типи кіберзагроз, навіть ті, які раніше не були відомі, шляхом аналізу та класифікації навчальних даних.

6. Аналіз поведінки користувачів: Машинне навчання може аналізувати поведінку користувачів та виявляти підозрілі активності, такі як незвичайні підключення, несподівані запити на доступ до ресурсів тощо.

Загалом, машинне навчання відіграє ключову роль у підвищенні ефективності та точності систем кібербезпеки, забезпечуючи виявлення та реагування на кіберзагрози в реальному часі.

У світі постійно зростається складність мережевих загроз, що вимагає передових рішень для підвищення безпеки мережі та ефективного виявлення потенційних небезпек. Машинне навчання (ML) стало одним з ключових інструментів у цьому контексті, пропонуючи потенціал для виявлення та ліквідації загроз у реальному часі через аналіз великих обсягів мережевих даних.

Досліджуючи питання «Комплексний огляд ролі машинного навчання в підвищенні безпеки мережі та виявленні загроз» вчені почали з аналізу поточного ландшафту загроз у мережевій безпеці та складнощі традиційних підходів до безпеки. Розглядалися фундаментальні принципи машинного навчання та його застосування для підвищення безпеки мережі. Окреслили різні методи машинного навчання, такі як навчання з учителем, без нагляду та глибоке навчання, і навели їх сильні сторони та обмеження в контексті виявлення загроз.

Після цього розглянули застосування машинного навчання у різних аспектах мережевої безпеки, зокрема виявлення вторгнень, виявлення шкідливих програм, виявлення аномалій та аналіз поведінки. Надали тематичні дослідження та приклади з реального життя для ілюстрації ефективності підходів, заснованих на машинному навчанні, у виявленні та мінімізації загроз безпеці.

Крім того, обговорили проблеми та міркування, пов’язані з розгортанням машинного навчання в середовищах безпеки мережі, такі як конфіденційність даних, можливість інтерпретації моделі та змагальні атаки. Розглянули стратегії вирішення цих проблем та підвищення надійності моделей машинного навчання.

Також окреслили майбутні напрямки досліджень та можливості для використання машинного навчання у підвищенні безпеки мережі.

Використовуючи можливості алгоритмів та методів машинного навчання, організації можуть посилити свій захист від кіберзагроз і краще захистити свої мережі та конфіденційні дані. [17]

Отже, машинне навчання (ML) має значний вплив на різні галузі, надаючи можливості для автоматизації, підвищення ефективності та покращення прийняття рішень. Так ML покращує діагностику захворювань, дозволяючи точніше та швидше виявляти хвороби, такі як рак, діабет та серцево-судинні захворювання. Використання алгоритмів для прогнозування захворювань сприяє ранньому виявленню та своєчасному втручанню, що підвищує шанси на успішне лікування.

ML допомагає у прогнозуванні землетрусів, аналізуючи великі обсяги сейсмічних даних для виявлення патернів, які можуть свідчити про наближення землетрусу. Алгоритми машинного навчання можуть покращити точність та швидкість оцінки параметрів землетрусів, що сприяє своєчасному реагуванню та зменшенню шкоди.

Машинне навчання дозволяє ефективно виявляти кіберзагрози в реальному часі шляхом аналізу мережевого трафіку та поведінки користувачів. Використання ML у кібербезпеці сприяє автоматизації процесів реагування на інциденти, що підвищує здатність організацій захищати свої системи від кіберзагроз.

Машинне навчання допомагає аналізувати генетичні дані для виявлення мутацій та генетичних маркерів, що асоціюються з певними захворюваннями. Сприяє розробці персоналізованої медицини, надаючи індивідуальні

рекомендації щодо лікування на основі генетичних даних пацієнтів.

Використання ML в освіті дозволяє створювати персоналізовані навчальні плани, що відповідають індивідуальним потребам та рівню знань кожного учня. Також ML може автоматизувати адміністративні завдання та надавати індивідуалізовані рекомендації, що покращує загальний навчальний процес.

1.2. Фреймворки у мовах програмування

Популярність Python неухильно зростає та залишається другою за поширеністю мовою у світі, її використовують 11,3 мільйона розробників програмного забезпечення по всьому світу.

Python — це високорівнева мова програмування, яка була створена на початку 1990-х років Гвідо ван Россумом. Вона досить проста у вивченні, а також має широкий спектр застосування: Python використовують для розробки веб-додатків, аналізу даних, штучного інтелекту, наукових обчислень, автоматизації завдань та багато іншого.[5]

Мова підтримує безліч парадигм програмування (правил та стилів написання коду), пропонує велику бібліотеку модулів (готових наборів коду, які можна використовувати у своїй програмі) та інструментів для виконання різних завдань. Крім того, Python має безліч сторонніх бібліотек і фреймворків (наборів інструментів для спрощення написання коду), які розширюють функціональні можливості мови.

Окрім інформатики, мова Python довела свою ефективність у багатьох інших галузях, таких як статистика, аналіз даних, фізика, економіка, електроніка. Python є однією з найшвидше зростаючих і найпотужніших мов із простим у використанні синтаксисом, динамічною семантикою та сотнями надійних бібліотек і фреймворків, яка широко використовується для веб-розробки, а також для великих даних, машинного навчання та штучного інтелекту.[12]

Найбільші світові компанії, такі як Amazon, Spotify, Quora та Google, використовують структуру веб-розробки Python, оскільки вони завжди шукають

інноваційні технології та рішення та хочуть розробляти передові програми, які відповідають багатьом вимогам.

Згідно з останніми звітами, близько 42% розробників заявили, що вони використовують Python як свою основну мову.

За даними TIOBE рейтинг мов програмування у світі такий:

May 2024	May 2023	Change	Programming Language	Ratings	Change
1	1		Python	16.33%	+2.88%
2	2		C	9.98%	-3.37%
3	4	↑	C++	9.53%	-2.43%
4	3	↓	Java	8.69%	-3.53%
5	5		C#	6.49%	-0.94%
6	7	↑	JavaScript	3.01%	+0.57%
7	6	↓	Visual Basic	2.01%	-1.83%
8	12	⇈	Go	1.60%	+0.61%
9	9		SQL	1.44%	-0.03%
10	19	⇈	Fortran	1.24%	+0.46%
11	11		Delphi/Object Pascal	1.24%	+0.23%
12	10	↓	Assembly language	1.07%	-0.13%
13	18	⇈	Ruby	1.06%	+0.26%
14	15	↑	MATLAB	1.06%	+0.18%
15	14	↓	Swift	1.01%	+0.09%
16	8	⇊	PHP	0.97%	-0.62%

Рис. 1.1 Рейтинг мов програмування у світі

Спільнота DOU надала результати опитування українських програмістів стосовно мов програмування, якими вони користуються. На основі 8250 анкет зібрали рейтинги мов за популярністю:

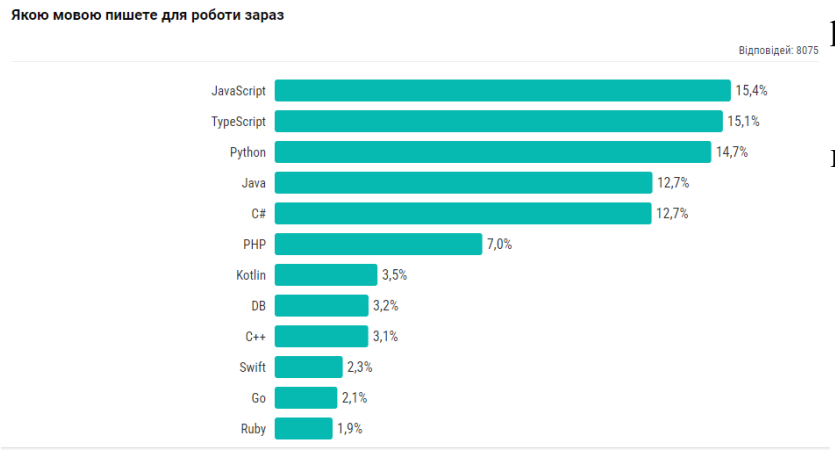

Рис. 1.2 Рейтинг мов програмування в Україні

Одна з ключових переваг Python – наявність великої екосистеми фреймворків, що надають розробникам готові інструменти та рішення для прискорення процесу розробки.

Фреймворк Python — це набір модулів Python, який забезпечує ряд загальних функцій, які можна використовувати як структуру для створення програм будь-якого типу.

Фреймворки створені для спрощення процесу розробки, надаючи загальні вказівки щодо створення програмного забезпечення та абстрагуючи деякі складніші або повторювані завдання. Прикладом повторюваного завдання може бути обробка HTTP-запитів. Оскільки більшості веб-програм потрібно обробляти цей тип запиту, розробники використовують існуючі фреймворки, які полегшують цю функцію, замість того, щоб писати все з нуля або повторно використовувати той самий код у різних проектах. Це дає змогу зосередитися на написанні унікальної та спеціальної логіки для програм.

Наприклад, у фреймворки спочатку можуть бути закладені:

- шаблони дизайну з маршрутизацією URL-адрес;
- валідація введення форм;
- міграція БД;
- ORM;
- механізми шаблонізації виведених форм;
- захист від підробки міжсайтових запитів і різних типів ін'єкцій;
- авторизація, аутентифікація, зберігання та вилучення сеансів;
- конфігурація підключення до баз даних.

Python має різноманітні фреймворки для різних типів розробки.[3] На сьогоднішній день існують десятки фреймворків для Python. За типами їх розподіляють на фуллстекові, мікрофреймворки й асинхронні.

Фуллстек фреймворк Python — це набір інструментів, який надає все, що потрібно розробнику для створення повної веб-програми від початку до кінця.

Це включає в себе спосіб створення інтерфейсу — наприклад, систему шаблонів і підхід до відображення інформації для користувача — і backend, включаючи загальні функції, такі як створення записів бази даних, обробка HTTP-запитів і контроль безпеки програми.

Мікрофреймворк — це мінімалістичний фреймворк, який містить лише основні компоненти, необхідні для створення будь-якої програми.

Він розроблений таким чином, щоб бути легким і легко розширюватися, що робить його хорошим вибором для невеликих проектів або для розробників, які хочуть більше контролювати свій код.

Асинхронний фреймворк призначений для роботи з конкурентністю і паралелізмом, дозволяючи розробникам створювати програми, які можуть виконувати кілька завдань одночасно.

Дуже різноманітне призначення і застосування фреймворків Python, вони широко використовуються в різних галузях розробки програмного забезпечення.[7]

Веб-розробка: фреймворки Python спрощують розробку створення веб-додатків і вебсайтів, надаючи структуру, маршрутизацію URL, управління базами даних, аутентифікацію користувачів та інші типові функції. Популярні фреймворки для веб-розробки – Django, Flask, FastAPI і Pyramid.

Розробка API: фреймворки Python також використовуються для розроблення API (Application Programming Interface), які дають змогу взаємодіяти з іншими додатками та сервісами. API-фреймворки, як-от FastAPI і Flask, пропонують простий спосіб створення та документування API для обміну даними між різними додатками.[9]

Наукові обчислення та аналіз даних: мова програмування Python широко застосовується в галузі наукових обчислень та аналізу даних. Для роботи з даними, виконання математичних операцій і візуалізації результатів використовують фреймворки та бібліотеки такі як NumPy, SciPy, Pandas і Matplotlib.

Автоматизація завдань: фреймворки Python можна використовувати для написання скриптів, обробки даних, керування серверами та інших автоматичних завдань.

Розробка мобільних додатків: за допомогою фреймворків Kivy і BeeWare, розробники можуть створювати кросплатформні мобільні додатки для різних операційних систем.

Розробка ігор: Python широко застосовується в розробці комп'ютерних ігор за допомогою фреймворків, таких як Pygame, які надають інструменти для створення графічних додатків та ігор.

Python активно використовується в галузі штучного інтелекту та машинного навчання. Фреймворки TensorFlow, Keras, XGBoost і PyTorch, надають потужні інструменти для розроблення та навчання нейронних мереж та інших моделей машинного навчання.[14]

Тож можна виділити кілька переваг використання фреймворків Python під час розробки програм. Розглянемо деякі з них:

- Фреймворки полегшують роботу розробника програмного забезпечення, забезпечуючи структуру для коду та набір інструментів і функцій, які оптимізують процес розробки, фреймворки можуть полегшити та зробити ефективнішим для розробників створення програм.
- Фреймворки сприяють організації коду: добре розроблена структура допомагає гарантувати, що код організований і підтримується, полегшуючи розуміння та зміну вихідного коду в майбутньому.
- Фреймворки можуть підвищити продуктивність: надаючи готові компоненти та інструменти та дотримуючись галузевих стандартів,

фреймворки дозволяють розробникам зосередитися на унікальних аспектах своїх програм, а не витрачати час на базові завдання.

1.3. Фреймворк PyTorch

PyTorch — це фреймворк машинного навчання з відкритим вихідним кодом, в основному розроблений дослідницькою лабораторією AI Facebook (FAIR). Він розроблений для глибокого навчання та забезпечує гнучку та ефективну платформу для побудови та розгортання моделей нейронних мереж.[10] PyTorch набув популярності як в академічних дослідженнях, так і в галузевих додатках завдяки простоті використання, динамічному обчислювальному графіку та надійній підтримці прискорення GPU.

Основні характеристики PyTorch

1. Динамічні обчислювальні графіки (визначення за виконанням):

PyTorch використовує динамічні обчислювальні графіки, тобто графік будується на льоту під час виконання операцій. Це забезпечує більшу гнучкість у створенні та модифікації моделей порівняно зі статичними обчислювальними графіками, які використовуються деякими іншими фреймворками.

2. Тензорне обчислення:

Подібно до NumPy, PyTorch забезпечує підтримку багатовимірних масивів (тензорів) і містить повну бібліотеку математичних операцій, які можна виконувати над цими тензорами.

3. Автоматична диференціація (Autograd):

PyTorch має потужну систему автоградації для автоматичної диференціації, яка має вирішальне значення для навчання нейронних мереж за допомогою методів оптимізації на основі градієнта.

4. Прискорення GPU:

PyTorch може використовувати прискорення GPU для прискорення тензорних операцій і навчання нейронної мережі, що робить його придатним для обробки великомасштабних даних і складних моделей.

5. Підтримка динамічних нейронних мереж:

Динамічний характер PyTorch робить його особливо придатним для побудови рекурентних нейронних мереж (RNN) та інших моделей, де розмір або структура вхідних даних можуть змінюватися.

6. Багата екосистема:

PyTorch має зростаючу екосистему бібліотек та інструментів, таких як TorchVision для комп'ютерного зору, TorchText для обробки природної мови та PyTorch Geometric для графових нейронних мереж.

7. Спільнота та документація:

PyTorch має переваги від великої та активної спільноти, великої документації, навчальних посібників і великої кількості ресурсів, доступних для навчання та усунення несправностей.[13]

PyTorch використовується в широкому діапазоні програм, зокрема:

- Аналіз зображень і відео: виявлення об'єктів, класифікація зображень, створення зображень і аналіз відео.
- Обробка природної мови (NLP): класифікація тексту, моделювання мови, машинний переклад і аналіз настроїв.
- Навчання з підкріпленням: навчання агентів для виконання завдань за допомогою навчання на основі винагороди.
- Розпізнавання мовлення: розробка моделей для автоматичного розпізнавання та синтезу мовлення.
- Генеративні моделі: Створення генеративних змагальних мереж (GAN) і варіаційних автокодерів (VAE) для створення реалістичних вибірок даних.

Приклади програм, в яких був використаний фреймворк PyTorch: Facebook, Uber, Tesla, Amazon, Microsoft.

Ось простий приклад використання PyTorch для визначення та навчання нейронної мережі для завдання класифікації:

```
import torch
        import torch.nn as nn
        import torch.optim as optim
```

```
# Define a simple feedforward neural network
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(784, 128)  # Assuming input features are 784 (e.g., 28x28 images)
        self.fc2 = nn.Linear(128, 10)   # Assuming 10 output classes

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Initialize the network, loss function, and optimizer
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Dummy training loop
for epoch in range(10):  # Training for 10 epochs
    optimizer.zero_grad()  # Clear gradients
    outputs = model(inputs)  # Forward pass
    loss = criterion(outputs, labels)  # Compute loss
    loss.backward()  # Backward pass
    optimizer.step()  # Update weights

    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

У цьому прикладі SimpleNN — це базова нейронна мережа з одним прихованим шаром, навчена за допомогою стохастичного градієнтного спуску (SGD), щоб мінімізувати втрату крос-ентропії.

Гнучкість, простота використання та потужні функції фреймворку PyTorch роблять його кращим вибором для багатьох дослідників і практиків у сфері машинного та глибокого навчання. Незалежно від того, розробляються передові дослідницькі моделі чи розгортаються виробничі системи, PyTorch надає інструменти та можливості, які допоможуть досягти успіху.

1.4. Фреймворк TensorFlow

TensorFlow — це платформа машинного навчання з відкритим кодом, розроблена командою Google Brain. Він широко використовується для створення та розгортання моделей машинного навчання, починаючи від простих лінійних регресій і закінчуючи складними архітектурами глибокого навчання. TensorFlow відомий своєю гнучкістю, масштабованістю та надійною екосистемою, що робить його придатним як для дослідницького, так і для виробничого середовища.[21]

Основні характеристики TensorFlow

1. Статичні обчислювальні графіки:

- TensorFlow спочатку використовував статичні обчислювальні графи, також відомі як графи потоку даних. Спочатку користувачі визначають обчислювальний графік, а потім TensorFlow виконує його. Цей підхід забезпечує ефективну оптимізацію та розгортання моделей, але може бути менш гнучким під час розробки.
- У TensorFlow 2.0 за замовчуванням представлено швидке виконання, яке дозволяє виконувати динамічні графіки, подібні до PyTorch, що робить його більш інтуїтивно зрозумілим для розробників.

2. Тензорне обчислення:

- TensorFlow забезпечує підтримку багатовимірних масивів (тензорів) і містить повну бібліотеку математичних операцій для маніпулювання цими тензорами.

3. Автоматична диференціація:

- TensorFlow включає функцію автоматичного диференціювання, яка є важливою для алгоритмів оптимізації на основі градієнта, які використовуються для навчання нейронних мереж.

4. Підтримка GPU і TPU:

- TensorFlow може використовувати GPU і TPU (блоки обробки тензорів) для прискорення обчислень, що робить його ефективним для обробки великомасштабних даних і складних моделей.

5. API високого рівня:

- TensorFlow пропонує API високого рівня, такі як Keras, що спрощує створення та навчання нейронних мереж. Keras забезпечує зручний інтерфейс для швидкого створення та експериментування з моделями.

6. Розгортання моделі:

- TensorFlow Serving — це гнучка, високопродуктивна система обслуговування для розгортання моделей машинного навчання у виробничих середовищах. TensorFlow Lite дозволяє розгортати моделі на мобільних і вбудованих пристроях.

7. TensorFlow Extended (TFX):

- TFX — це наскрізна платформа для розгортання робочих конвеєрів машинного навчання, включаючи перевірку даних, навчання моделі, обслуговування та моніторинг.

8. Надійна екосистема:

- TensorFlow має широку екосистему, яка включає TensorBoard для візуалізації, TensorFlow Hub для обміну попередньо підготовленими моделями та TensorFlow.js для запуску моделей у браузері.

9. Спільнота та документація:

- TensorFlow отримує переваги від великої спільноти, великої документації, численних посібників і ресурсів, які допомагають користувачам вивчати та ефективно застосовувати фреймворк.

Розглянемо деякі з основних функцій та можливостей TensorFlow:

- Підтримка різних типів моделей машинного навчання (включаючи нейронні мережі, лінійну регресію, логістичну регресію тощо).
- Масштабованість (може використовуватись для навчання моделей на великих наборах даних).
- Гнучкість (дозволяє розробникам створювати моделі машинного навчання з урахуванням своїх потреб).

Які завдання можна вирішувати за допомогою TensorFlow?

- Аналіз зображень і відео: класифікація зображень, виявлення об’єктів, сегментація зображень і аналіз відео.
- Обробка природної мови (NLP): для аналізу та розуміння природної мови (визначення тональності тексту, машинний переклад та створення текстових моделей).
- Прогнозування часових рядів: прогноз майбутніх значень часових рядів (корисно у фінансовій аналітиці, метеорології та інших галузях).
- Навчання з підкріпленням: алгоритми навчаються приймати рішення, які допомагають їм отримати максимальну винагороду у певній обстановці чи середовищі, навчання агентів для ігор, робототехніки та інших завдань, які передбачають послідовне прийняття рішень.
- Розпізнавання мовлення: розробка моделей для автоматичного розпізнавання мовлення та синтезу мовлення.
- Генеративні моделі: Створення генеративних змагальних мереж (GAN) і варіаційних автокодерів (VAE) для створення реалістичних вибірок даних.
- Охорона здоров’я: прогнозування захворювань, аналіз медичних зображень і персоналізовані плани лікування.
- Фінанси: виявлення шахрайства, алгоритмічна торгівля та оцінка ризиків.

Крім базових функцій для створення та навчання моделей машинного навчання, бібліотека також надає безліч розширених функцій:

- згорткові нейронні мережі: використовуються для обробки зображень та інших просторових даних;

- рекурентні нейронні мережі: для обробки тимчасових даних, таких як мова та текст;
- глибокі самонавчені моделі: використовують різні методи навчання без вчителя для навчання моделей на даних без міток.

Для прикладу можна навести такі складні проекти, в яких був використаний фреймворк TensorFlow:

AlphaGo: Google DeepMind створив проект AlphaGo — штучний інтелект, який зміг обіграти чемпіона світу з го. Проект використовував глибоке навчання та великі обчислювальні ресурси.

OpenAI GPT (Generative Pre-trained Transformer): це серія моделей, створених OpenAI, для генерації людиноподібного тексту. GPT використовує трансформери та навчається на величезних обсягах текстових даних. TensorFlow був використаний для створення та навчання цих моделей.

Ось простий приклад використання TensorFlow для визначення та навчання нейронної мережі для завдання класифікації:

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define a simple feedforward neural network
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(784,)),  # Assuming input features are 784 (e.g., 28x28 images)
    layers.Dense(10, activation='softmax')  # Assuming 10 output classes
])

# Compile the model
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Dummy data
```

```
import numpy as np
inputs = np.random.random((100, 784))
labels = np.random.randint(10, size=(100,))

# Train the model
model.fit(inputs, labels, epochs=10)

# Evaluate the model
loss, accuracy = model.evaluate(inputs, labels)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

У цьому прикладі проста нейронна мережа визначається за допомогою Keras API у TensorFlow. Мережа компілюється, навчається на фіктивних даних, а потім оцінюється. Комплексний набір інструментів і функцій TensorFlow робить його потужною та універсальною структурою для машинного та глибокого навчання. Його підтримка API високого рівня, масштабованість і надійна екосистема дозволяють користувачам створювати, навчати та розгортати моделі ефективно як у дослідницьких, так і у виробничих умовах. Незалежно від того, розробляєте ви експериментальні моделі чи розгортаєте великомасштабні виробничі системи, TensorFlow надає необхідні інструменти для досягнення успіху.

1.5. Фреймворк XGBoost

XGBoost (eXtreme Gradient Boosting) — це платформа машинного навчання з відкритим вихідним кодом, розроблена для ефективної та масштабованої реалізації алгоритмів посилення градієнта. Він був розроблений Tianqi Chen і широко використовується в наукових змаганнях з даних і машинного навчання завдяки своїй високій продуктивності та точності. XGBoost особливо добре підходить для структурованих (табличних) даних і відомий своєю швидкістю та ефективністю обробки великих наборів даних.[22]

Основні характеристики XGBoost

1. Структура посилення градієнта:

- XGBoost реалізує алгоритм посилення градієнта, який послідовно створює ансамбль дерев рішень. Кожне дерево намагається виправити помилки попередніх дерев, ітеративно покращуючи модель.

2. Регулярізація:

- XGBoost включає L1 (Lasso) і L2 (Ridge) умови регулярізації в цільовій функції, щоб запобігти переобладнанню, що допомагає покращити узагальнення моделі.

3. Паралельні розподілені обчислення:

- XGBoost може використовувати кілька багатоядерних ЦП і підтримувати розподілене обчислення, що робить його ефективним для навчання на великих наборах даних. Він може працювати на різних платформах, включаючи Hadoop, Spark і Dask.

4. Обробка відсутніх значень:

- XGBoost має вбудований механізм обробки відсутніх значень, який може автоматично вивчити найкращий спосіб їх обробки під час навчання.

5. Обрізка дерев

- XGBoost використовує техніку під назвою "max_depth", щоб обмежити глибину дерев і запобігти переобладнанню. Він також використовує техніку скорочення під назвою "max_delta_step" для покращення конвергенції.

6. Блок колонок для паралельного навчання:

- XGBoost реалізує блочну структуру стовпців для оптимізації використання пам’яті та обчислень, підвищуючи ефективність процесу навчання.

7. Перехресна перевірка:

- XGBoost забезпечує інтегровані процедури перехресної перевірки, що дозволяє користувачам легко виконувати перехресну перевірку та налаштовувати гіперпараметри.

8. Масштабованість і гнучкість:

- XGBoost розроблено для високої масштабованості та може використовуватися на одній машині або розподілятися між кількома машинами. Він підтримує різні інтерфейси, включаючи Python, R, Java, Scala та C++.

XGBoost використовується в широкому діапазоні програм, зокрема:

- Класифікація: проблеми бінарної та багатокласової класифікації, такі як виявлення спаму, діагностика захворювань і сегментація клієнтів.
- Регресія: завдання прогнозного моделювання, такі як прогнозування цін на житло, прогнозування продажів і оцінка попиту.
- Рейтинг: Інформаційно-пошукові та рекомендаційні системи.
- Вибір функцій: визначення важливих функцій у наборах даних.
- Виявлення аномалій: виявлення викидів і шахрайських дій.

Приклади програм, в яких був використаний фреймворк XGBoost: Alibaba, JD.com, Tencent, Airbnb.

Ось простий приклад використання XGBoost для завдання класифікації в Python:

```
import xgboost as xgb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create DMatrix for XGBoost
```

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Set XGBoost parameters
params = {
    'objective': 'multi:softprob',  # Specify multiclass classification
    'num_class': 3,  # Number of classes
    'max_depth': 3,
    'eta': 0.1,
    'eval_metric': 'mlogloss'
}

# Train the model
num_rounds = 100
bst = xgb.train(params, dtrain, num_rounds)

# Make predictions
preds = bst.predict(dtest)
best_preds = np.asarray([np.argmax(line) for line in preds])

# Evaluate accuracy
accuracy = accuracy_score(y_test, best_preds)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

У цьому прикладі набір даних Iris використовується для навчання багатокласової моделі класифікації за допомогою XGBoost. Набір даних розбивається на набори для навчання та тестування, і для обох наборів створюється xgb.DMatrix. Модель навчається із заданими параметрами, і оцінюється точність прогнозів на тестовому наборі.

XGBoost — це потужний і гнучкий фреймворк машинного навчання, який чудово працює зі структурованими даними. Його ефективна реалізація посилення градієнта, методів регуляризації та підтримки паралельних і розподілених обчислень робить його кращим вибором для багатьох науковців із обробки даних і практиків машинного навчання.

РОЗДІЛ 2. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1. Пошук набору даних

Важливим фактором для успіху в машинному навчанні є пошук коректного набору даних.

Було обрано готовий набір даних про виявлення шахрайства з кредитними картками за 2023 рік з сайту Kaggle.[15] Цей набір даних містить транзакції за кредитними картками, здійснені власниками європейських карток у 2023 році. Він містить понад 550 000 записів, і дані були анонімізовані для захисту ідентифікаційних даних власників карток. Основна мета цього набору даних - сприяти розробці алгоритмів і моделей виявлення шахрайства для виявлення потенційно шахрайських транзакцій.

Основні характеристики:

- id: Унікальний ідентифікатор для кожної транзакції
- V1-V28: Анонімізовані ознаки, що представляють різні атрибути транзакції (наприклад, час, місцезнаходження тощо)
- Сума: Сума транзакції
- Клас: Двійкова мітка, що вказує на те, чи є транзакція шахрайською (1) чи ні (0)

2.2. Впорядкування навчальної та тестової вибірки

Рішення щодо стратегій поділу даних на навчальні та тестові відіграють важливу роль у життєвому циклі моделі машинного навчання, суттєво впливаючи на її продуктивність у виробничому середовищі. Те, як ми

розділяємо наші дані на навчальні та тестові набори, впливає на здатність моделі навчатися на даних і на її здатність узагальнювати нові, небачені дані. Цей крок має вирішальне значення для розробки надійних моделей, які дають достовірні прогнози в реальних додатках, де ставки на точність прогнозування можуть бути високими.

Готуючи набір даних для проекту машинного навчання, необхідно керуватися кількома важливими аспектами при поділі його на навчальний і тестовий набори. Наступні міркування є важливими для забезпечення ефективного навчання моделі та її здатності добре узагальнювати нові дані.

Розмір набору даних відіграє вирішальну роль у прийнятті рішення про те, як необхідно розділити на навчальну та тестову частини. З великими наборами даних можна виділити менший відсоток для тестування (наприклад, 80/20 або 70/30), оскільки все одно буде достатньо даних для навчання і тестування. Однак з меншими наборами даних потрібно бути більш обережними (наприклад, 90/10), щоб гарантувати, що модель має достатньо даних для навчання.

В даній роботі було використано функцію train_test_split з бібліотеки sklearn для розділення вибірки [11]:

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

Тобто розмір тестової вибірку буде 20%, а навчальної відповідно 80%

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Навчання моделей на тестовій виборці на фреймворках

Для створення класифікації даних засобами сучасних Python фреймворків необхідно створити модель машинного навчання. Основна мета полягає в тому що процес навчання моделі оптимізувати щоб прогнозування було максимально точним. Для реалізації цієї мети розроблено алгоритм навчання (Рис.3.1):

- Початок
- Завантаження даних. В даній роботі було використано бібліотеку Pandas:

  train_dataset_fp = 'creditcard_2023.csv'

  df = pd.read_csv(train_dataset_fp)
- Розділення даних на навчальну та тестову вибірку. В даному випадку використано бібліотеку scikit-learn:

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

- Нормалізація даних. Знову звернемося до бібліотеки scikit-learn:
  scaler = StandardScaler()
  X_train = scaler.fit_transform(X_train)
  X_test = scaler.transform(X_test)
- Побудова та навчання моделі. На цьому етапі починається робота самих фреймворків тому код для кожного буде окремий.
  TensorFlow
  1. Створення моделі:
     model = Sequential([
        Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
     ])
  2. Компіляція моделі:
     model.compile(optimizer=Adam(learning_rate=0.001),
                loss='binary_crossentropy',
                metrics=['accuracy'])
  3. Навчання моделі:
     history = model.fit(X_train, y_train,
                      epochs=10,
                      batch_size=32,
                      validation_split=0.2)

  XGBoost
  1. Створеня моделі:
     Model=xgb.XGBClassifier(use_label_encoder=False,
     eval_metric='logloss')

2. Навчання моделі:

   model.fit(X_train, y_train)

Pytorch

1. Перетворення даних у тензори:

   X_train = torch.tensor(X_train, dtype=torch.float32)

   X_test = torch.tensor(X_test, dtype=torch.float32)

   y_train = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)

   y_test = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)

2. Створення DataLoader:

   train_dataset = TensorDataset(X_train, y_train)

   train_loader=DataLoader(train_dataset, batch_size=32, shuffle=True)

3. Визначення моделі:

```
class CreditCardFraudModel(nn.Module):
    def __init__(self):
        super(CreditCardFraudModel, self).__init__()
        self.layer1 = nn.Linear(X_train.shape[1], 32)
        self.layer2 = nn.Linear(32, 16)
        self.layer3 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = torch.sigmoid(self.layer3(x))
        return x

model = CreditCardFraudModel()
```

4. Визначення функції втрат і оптимізатора:

   criterion = nn.BCELoss()

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

5. Навчання моделі:

```
epochs = 10
for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        X_batch, y_batch = batch

        optimizer.zero_grad()

        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)

        loss.backward()
        optimizer.step()
```

- Оцінювання моделі на тестових даних

TensorFlow

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'\nTest accuracy: {test_acc}')
```

XGBoost

```
y_pred = model.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)
print(f'\nTest accuracy: {test_acc}')
```

Pytorch

```
model.eval()
```

```
with torch.no_grad():
    test_outputs = model(X_test)
    test_loss = criterion(test_outputs, y_test).item()
    test_acc = ((test_outputs.round() == y_test).sum().item()) / y_test.size(0)
```

- Збереження моделі
  `model.save('creditcard_fraud_model.h5')`
- Кінець

3.2. Аналіз випробування тестової вибірки

Для оцінки ефективності різних Python-фреймворків для класифікації даних було використано Credit Card Fraud Detection Dataset 2023.[15] Після побудови та налаштування моделей на основі TensorFlow, XGBoost та PyTorch, було проведено їх навчання та тестування. Нижче наведено результати точності моделей на тестовій вибірці:

- TensorFlow: Точність на тестовій вибірці - 0.9991294741630554
- XGBoost: Точність на тестовій вибірці - 0.9997274150150361
- PyTorch: Точність на тестовій вибірці - 0.9987074195874295

Аналіз результатів

1. TensorFlow
   - Точність: 0.9991
   - TensorFlow продемонстрував високу точність у класифікації транзакцій. Його гнучкість у налаштуванні нейронних мереж та можливості для глибокого навчання роблять його потужним інструментом для задач класифікації.
   - Переваги: Висока точність, гнучкість у побудові моделей, підтримка великих обсягів даних.
   - Недоліки: Може вимагати більше часу на налаштування та оптимізацію моделей.
2. XGBoost

- Точність: 0.9997
- XGBoost показав найвищу точність серед усіх протестованих моделей. Це відображає його ефективність у вирішенні задач класифікації завдяки використанню бустингових дерев.
- Переваги: Висока точність, швидке навчання, можливість обробки нерівномірно розподілених даних.
- Недоліки: Може бути складним у налаштуванні для початківців, потребує ретельного підбору гіперпараметрів.

3. PyTorch
    - Точність: 0.9987
    - PyTorch також продемонстрував високу точність, хоча і трохи нижчу, ніж TensorFlow та XGBoost. PyTorch відомий своєю гнучкістю та легкістю у використанні, що робить його популярним серед дослідників.
    - Переваги: Зручність у використанні, гнучкість у побудові моделей, можливості для дослідження та експериментування.
    - Недоліки: Можливо, трохи нижча точність у порівнянні з TensorFlow та XGBoost, вимагає більше обчислювальних ресурсів для глибоких моделей.
    -

3.3. Складання метрики для порівняння

Метрики для порівняння:

1. Точність (Precision): Відношення правильно передбачених позитивних спостережень до загальної кількості передбачених позитивних спостережень.
2. Повнота (Recall): Відношення правильно передбачених позитивних спостережень до всіх спостережень у дійсному класі.
3. F1-міра: Гармонійне середнє між точністю та повнотою.
4. ROC-AUC: Площа під кривою ROC (Receiver Operating Characteristic), що показує здатність моделі розрізняти класи.

Таблиця 3.1

Порівняння метрик моделей

Модель	Test Accuracy	Test Precision	Test Recall	Test F1 Score	Test ROC-AUC
TensorFlow	0.9989800045723933	0.9980379791184492	0.9999297950014041	0.9989829914080308	0.9999083452094212
XGBoost	0.9997274150150361	0.9994562071324574	1.0	0.9997280296184452	0.9997268722466961
PyTorch	0.9991119005328597	0.9986323227718259	0.9995963212580736	0.9991140894857333	0.9997809378274428

Аналіз результатів

1. Точність (Accuracy):
    - Найвищу точність демонструє модель XGBoost із результатом 0.9997274150150361.
    - Модель PyTorch займає другу позицію із результатом 0.9991119005328597.
    - Модель TensorFlow має точність 0.9989800045723933.
2. Точність (Precision):
    - Модель XGBoost має найвищу точність 0.9994562071324574, що означає найменше хибнопозитивних результатів.
    - PyTorch показує значення 0.9986323227718259, трохи нижче за XGBoost.
    - TensorFlow має точність 0.998037979118492, що також є високим значенням, але найнижчим серед усіх трьох моделей.
3. Повнота (Recall):
    - Модель XGBoost досягає 1.0, що вказує на відсутність хибнонегативних результатів.
    - TensorFlow також демонструє високу повноту 0.9999297950014041.
    - PyTorch має значення 0.9995963212580736.
4. F1-міра (F1 Score):

- Модель XGBoost має найвищу F1-міру 0.999728029618452.
- PyTorch займає другу позицію з F1-мірою 0.9991140894857333.
- TensorFlow демонструє значення 0.9989829914080308.

5. ROC-AUC:
   - TensorFlow має найвищий ROC-AUC із значенням 0.9999083452094212.
   - PyTorch має дуже близьке значення 0.9997809378274428.
   - XGBoost показує значення 0.9997268722466961.

Отже:

- XGBoost демонструє найвищі результати за метриками точності, точності та F1-міри, що робить його найкращою моделлю для даного завдання.
- TensorFlow виділяється найвищим ROC-AUC, що вказує на його здатність ефективно розрізняти класи.
- PyTorch займає проміжне місце, демонструючи високу точність, F1-міру та ROC-AUC, але трохи відстає від XGBoost.

Таким чином, для класифікації даних у задачі виявлення шахрайства з кредитними картками модель XGBoost є найефективнішою за більшістю метрик, тоді як TensorFlow показує найкращий результат у здатності розрізняти класи за ROC-AUC.

ВИСНОВКИ

У даній роботі було проведено дослідження та аналіз ефективності сучасних інструментів для класифікації даних. Було використано три популярні фреймворки: TensorFlow, XGBoost та PyTorch для розв'язання задачі виявлення шахрайства з кредитними картками на основі Credit Card Fraud Detection Dataset 2023.

Результати дослідження показали, що кожен з фреймворків має свої переваги та недоліки, які були оцінені за кількома ключовими метриками: точність (Accuracy), точність (Precision), повнота (Recall), F1-міра (F1 Score) та площа під кривою ROC (ROC-AUC).

Основні висновки:

1. XGBoost:
    - Показав найвищу точність, точність та F1-міру серед усіх трьох моделей, що робить його найефективнішим інструментом для даної задачі.
    - Відзначився відсутністю хибнонегативних результатів, що є критично важливим у задачах виявлення шахрайства.
2. TensorFlow:
    - Виділився найвищим значенням ROC-AUC, що свідчить про його здатність ефективно розрізняти класи.
    - Демонструє стабільно високі показники за всіма метриками, що робить його надійним інструментом для класифікації.
3. PyTorch:
    - Показав високі результати за всіма метриками, займаючи проміжне місце між XGBoost та TensorFlow.
    - Відзначився хорошою точністю та F1-мірою, що підтверджує його ефективність у розв'язанні задач класифікації.

За результатами проведеної роботи можна навести такі рекомендації:

1. Вибір фреймворку залежить від конкретних вимог до задачі. Якщо пріоритетом є максимальна точність та мінімізація хибнонегативних результатів, рекомендується використовувати XGBoost.
2. TensorFlow підходить для задач, де важливо мати високу здатність розрізняти класи, зокрема для задач з нерівномірними класами.
3. PyTorch є відмінним вибором для задач, де важлива гнучкість та простота реалізації складних моделей.

Кваліфікаційна робота демонструє, що сучасні Python-фреймворки забезпечують високу ефективність у розв'язанні задач класифікації даних, зокрема в задачі виявлення шахрайства з кредитними картками, та можуть бути успішно застосовані в практичних завданнях аналізу даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://ai.meta.com/blog
2. https://cloud.itstep.org/blog_3/frameworks-in-programming-languages-what-are-they-for-and-how-to-choose-them
3. https://foxminded.ua/freimvorky-python
4. https://geniusee.com/single-blog/ways-to-benefit-from-machine-learning-in-edtech
5. https://kinsta.com/blog/python-frameworks
6. https://merehead.com/ua/blog/trends-machine-learning-2023
7. https://owlweb.com.ua/news/cikavevidsov/dlya-chogo-potriben-freymvork-i-yak-yogo-vibrati
8. https://pubmed.ncbi.nlm.nih.gov/?term=Analyzing+medical+data+using+machine+learning
9. https://pythonframeworks.com
10. https://pytorch.org
11. https://www.clearbox.ai/blog/2024-02-20-how-to-create-a-train-and-test-dataset
12. https://www.coursera.org
13. https://www.freecodecamp.org/news/pytorch-and-monai-for-healthcare-imaging
14. https://www.intellectsoft.net/blog/python-frameworks-for-web-development
15. https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023
16. https://www.researchgate.net/publication/338184318_A_Machine-Learning_Approach_for_Earthquake_Magnitude_Estimation
17. https://www.researchgate.net/publication/378288472_A_comprehensive_review_of_machine_learning's_role_in_enhancing_network_security_and_threat_detection
18. https://www.sciencedirect.com/science/article/pii/S1532046421001209

19. https://www.sciencedirect.com/science/article/pii/S1532046421003014
20. https://www.temok.com/blog/python-frameworks
21. https://www.tensorflow.org
22. https://xgboost.readthedocs.io/en/stable/index.html
23. Вітинський П. Б., Ткаченко Р. О., Гавриш Б. М.. Дослідження та експериментальний аналіз методів машинного навчання в задачах електронної комерції : Наукові записки / scientific papers 2019 / 1 (58). УДК 004.85, 004.89, 004.9. с 62-70
24. Іан Х. Віттен, Ейбе Френк, Марк А. Холл **Data Mining: Practical Machine Learning Tools and Techniques. 558 с.**
25. Онищенко К. Г. Аналіз методів обробки природної мови / К. Г. Онищенко, Я. Данієль, Р. Каменєв : Інформаційні системи та технології : матеріали 9-ї Міжнар. наук.-техн. конф., 17-20 листопада 2020 р. – Харків : Друкарня Мадрид, 2020. – С. 186–190.
26. Приклад використання згорткової нейронної мережі для розпізнавання номіналу банкнот. УДК 004.93 DOI: https://doi.org/10.53920/ITS-2022-1-3 Науковий журнал «IT SYNERGY», 2022, випуск 1 (2)
27. Проект програмного комплексу для реалізації додатку для розпізнавання лікарських рослин. УДК 004:4’2DOI: https://doi.org/10.53920/ITS-2022-1-1 Науковий журнал «IT SYNERGY», 2022, випуск 1 (2)
28. Розробка програмного комплексу для коригування ваги хворих на цукровий діабет на основі використання нейронної мережі з логістичною регресією. УДК 004:4’2DOI: https://doi.org/10.53920/ITS-2022-1-2 Науковий журнал «IT SYNERGY», 2022, випуск 1 (2)
29. Том М. Мітчелл : **Machine Learning. 421 с.**

ДОДАТОК 1

Код для з використанням TensorFlow

```
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib
from sklearn.metrics import accuracy_score, precision_score, recall_score,
        f1_score, roc_auc_score

train_dataset_fp = 'creditcard_2023.csv'

# Завантаження даних з використанням pandas
df = pd.read_csv(train_dataset_fp)

# Видалення стовпця 'id'
df = df.drop(columns=['id'])

# Відокремлення ознак і міток
features = df.drop(columns=['Class']).values
labels = df['Class'].values

# Поділ даних на навчальну і тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
        random_state=42)
```

```
# Нормалізація даних
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Збереження тестової вибірки у файл
test_data = np.hstack((X_test, y_test.reshape(-1, 1)))
np.savetxt('test_data.csv', test_data, delimiter=',',
           header=','.join(list(df.drop(columns=['Class']).columns) + ['Class']),
           comments='')

# Створення моделі
model = Sequential([
   Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
   Dense(16, activation='relu'),
   Dense(1, activation='sigmoid')
])

# Компіляція моделі
model.compile(optimizer=Adam(learning_rate=0.001),
           loss='binary_crossentropy',
           metrics=['accuracy'])

# Навчання моделі
history = model.fit(X_train, y_train,
                epochs=10,
                batch_size=32,
                validation_split=0.2)

# Оцінювання моделі на тестових даних
```

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'\nTest accuracy: {test_acc}')

# Передбачення на тестових даних
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Обчислення додаткових метрик
test_precision = precision_score(y_test, y_pred)
test_recall = recall_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred)
test_roc_auc = roc_auc_score(y_test, y_pred_prob)

print(f'Test precision: {test_precision}')
print(f'Test recall: {test_recall}')
print(f'Test F1 score: {test_f1}')
print(f'Test ROC-AUC: {test_roc_auc}')

# Збереження моделі
model.save('creditcard_fraud_model.h5')
# Збереження нормалізатора
joblib.dump(scaler, 'scaler.pkl')
```

ДОДАТОК 2

Код з використанням XGBoost

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
        f1_score, roc_auc_score
import joblib

train_dataset_fp = 'creditcard_2023.csv'

# Завантаження даних з використанням pandas
df = pd.read_csv(train_dataset_fp)

# Видалення стовпця 'id'
df = df.drop(columns=['id'])

# Відокремлення ознак і міток
features = df.drop(columns=['Class']).values
labels = df['Class'].values
# Поділ даних на навчальну і тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
        random_state=42)
# Нормализация данных
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Створення моделі XGBoost
model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Навчання моделі
model.fit(X_train, y_train)

# Оцінювання моделі на тестових даних
y_pred = model.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)
test_precision = precision_score(y_test, y_pred)
test_recall = recall_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred)
test_roc_auc = roc_auc_score(y_test, y_pred)

print(f'\nTest accuracy: {test_acc}')
print(f'Test precision: {test_precision}')
print(f'Test recall: {test_recall}')
print(f'Test F1 score: {test_f1}')
print(f'Test ROC-AUC: {test_roc_auc}')

# Збереження моделі
model.save_model('xgboost_creditcard_fraud_model.json')
# Збереження нормалізатора
joblib.dump(scaler, 'scaler.pkl')

# Збереження тестової вибірки у файл
test_data = np.hstack((X_test, y_test.reshape(-1, 1)))
np.savetxt('test_data.csv',test_data,delimiter=',',header=','.join(list(df.drop(colum
ns=['Class']).columns) + ['Class']), comments='')
```

ДОДАТОК 3

Код з використанням Pytorch

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score

train_dataset_fp = 'creditcard_2023.csv'

# Завантаження даних з використанням pandas
df = pd.read_csv(train_dataset_fp)

# Видалення стовпця 'id'
df = df.drop(columns=['id'])

# Відокремлення ознак і міток
features = df.drop(columns=['Class']).values
labels = df['Class'].values

# Поділ даних на навчальну і тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
    random_state=42)

# Нормалізація даних
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Перетворення даних у тензори
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
y_test = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)

# Створення DataLoader
train_dataset = TensorDataset(X_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Визначення моделі
class CreditCardFraudModel(nn.Module):
    def __init__(self):
        super(CreditCardFraudModel, self).__init__()
        self.layer1 = nn.Linear(X_train.shape[1], 32)
        self.layer2 = nn.Linear(32, 16)
        self.layer3 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = torch.sigmoid(self.layer3(x))
        return x

model = CreditCardFraudModel()
```

```
# Визначення функції втрат і оптимізатора
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Навчання моделі
epochs = 10
for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        X_batch, y_batch = batch

        # Обнулення градієнтів
        optimizer.zero_grad()

        # Прямий прохід
        outputs = model(X_batch)

        # Обчислення втрат
        loss = criterion(outputs, y_batch)

        # Зворотний прохід і оптимізація
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Оцінювання моделі на тестових даних
model.eval()
with torch.no_grad():
    test_outputs = model(X_test)
```

```
    test_loss = criterion(test_outputs, y_test).item()
    test_acc = ((test_outputs.round() == y_test).sum().item()) / y_test.size(0)

# Перетворення передбачень на бінарні значення
y_pred = test_outputs.round().numpy()
y_test_np = y_test.numpy()

# Обчислення додаткових метрик
test_precision = precision_score(y_test_np, y_pred)
test_recall = recall_score(y_test_np, y_pred)
test_f1 = f1_score(y_test_np, y_pred)
test_roc_auc = roc_auc_score(y_test_np, test_outputs.numpy())

print(f'\nTest accuracy: {test_acc}')
print(f'Test precision: {test_precision}')
print(f'Test recall: {test_recall}')
print(f'Test F1 score: {test_f1}')
print(f'Test ROC-AUC: {test_roc_auc}')
```