ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «МОДЕЛЬ МАШИННОГО НАВЧАННЯ ДЛЯ ВИРІШЕННЯ ЗАДАЧ БІНАРНОЇ КЛАСИФІКАЦІЇ»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Сергій ХАРЬКОВ*

Виконав:
здобувач вищої освіти
група ІСДМ-62
Сергій ХАРЬКОВ

Керівник:
науковий ступінь, вчене звання
Олег СЕНЬКОВ
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання
_____
Ім’я, ПРІЗВИЩЕ

Київ – 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК
«____» _____20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ СТУДЕНТУ
Харькову Сергію Анатолійовичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Модель машинного навчання для вирішення задач бінарної класифікації»

керівник кваліфікаційної роботи Олег СЕНЬКОВ, к.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «15» жовтня 2024 року № 320.

2. Строк подання кваліфікаційної роботи: 27 грудня 2024 року.

3. Вихідні дані до кваліфікаційної роботи: Датасет Keras;
Значення TN, FP, FN, TP;
N_1 = 60000 зображень на навчанні;
N_2 = 10000 зображень на тренуванні.
Науково-технічна література з питань, пов'язаних з наукою про дані.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Методи аналізу даних та їх застосування у задачах класифікації.

2. Класифікація методами ML/DL.
3. Розробка моделі ML для вирішення задачі бінарної класифікації.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Типи машинного навчання.
   2. Порівняльна характеристика ML та DL.
   3. Датасет.
   4. Класифікація методом SVM.
   5. Реалізація методу SVM.
   6. Фреймворк Keras.
   7. Візуалізація даних.
   8. Модель ML.

6. Дата видачі завдання: 15 жовтня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	20.10 – 24.10.24	
2	Вивчення класифікації методами ML/DL	25.10 – 01.11.24	
3	Дослідження особливостей машинного та глибокого навчання	02.11 – 09.11.24	
4	Дослідження особливостей побудови нейронних мереж	10.11 – 16.11.24	
5	Створення архітектури нейронної мережі	17.11 – 20.11.24	
6	Оцінка якості роботи нейронної мережі	21.11 – 10.12.24	
7	Оформлення роботи: вступ, висновки, реферат	11.12 – 20.12.24	
8	Розробка демонстраційних матеріалів	21.12 – 25.12.24	

Здобувач вищої освіти _____ Сергій ХАРЬКОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник роботи
кваліфікаційної роботи _____ Олег СЕНЬКОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 72 стор., 20 рис., 6 табл., 20 джерел.

Мета роботи - реалізація моделі машинного навчання для вирішення задачі бінарної класифікації.

Об'єкт дослідження – процеси та методи аналізу даних для вирішення задач бінарної класифікації з використанням алгоритмів машинного навчання.

Предмет дослідження – алгоритми, методи та підходи до створення моделі машинного навчання для вирішення задач бінарної класифікації.

Короткий зміст роботи: Досліджені методи та технології машинного навчання, математичні інструменти для аналізу та обробки даних, методи ML/DL для вирішення задач класифікації в мережі, такі як наївний алгоритм Байєса, класифікація методом опорних векторів. Досліджені особливості машинного та глибокого навчання, найпопулярніші фреймворки для роботи з технологіями ML/DL: TensorFlow, Keras, надані рекомендації по використанню зазначених фреймворків. Розроблено модель ML для вирішення задачі бінарної класифікації в інтерактивному середовищі Jupyter Notebook. Здійснено вибір функції втрат для оптимізації нейронної мережі, виконане обчислення метрики якості роботи моделі для оцінки її ефективності та точності у процесі навчання та на тестових даних.

КЛЮЧОВІ СЛОВА: МАШИННЕ НАВЧАННЯ, ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, МОВА ПРОГРАМУВАННЯ PYTHON, ВІЗУАЛІЗАЦІЯ, ТЕХНОЛОГІЯ, АЛГОРИТМ, СОКЕТ, ПРОТОКОЛ, ІНТЕРАКТИВНЕ СЕРЕДОВИЩЕ.

ABSTRACT

Text part of the master`s qualification work: 72 pages, 20 pictures, 6 table, 20 sources.

The purpose of the work is to implement a machine learning model to solve the problem of binary classification.

Object of research is the processes and methods of data analysis for solving binary classification problems using machine learning algorithms.

Subject of research is algorithms, methods and approaches to creating a machine learning model for solving binary classification problems.

Summary of the work: Machine learning methods and technologies, mathematical tools for data analysis and processing, ML/DL methods for solving network classification problems, such as naive Bayes algorithm, support vector classification, were investigated. The features of machine and deep learning, the most popular frameworks for working with ML/DL technologies: TensorFlow, Keras were investigated, and recommendations for the use of these frameworks were provided. An ML model was developed to solve the problem of binary classification in the interactive Jupyter Notebook environment. The selection of the loss function for the optimization of the neural network was carried out, the calculation of the performance quality metric of the model was performed to assess its efficiency and accuracy in the training process and on test data.

KEYWORDS: MACHINE LEARNING, DEEP LEARNING, NEURAL NETWORK, PYTHON PROGRAMMING LANGUAGE, VISUALIZATION, TECHNOLOGY, ALGORITHM, SOCKET, PROTOCOL, INTERACTIVE ENVIRONMENT.

ЗМІСТ

ВСТУП

Актуальність теми. Бінарна класифікація є однією з найпоширеніших задач у галузі машинного навчання, адже дозволяє розв'язувати практичні проблеми, такі як: виявлення шахрайських транзакцій у фінансових системах; класифікація електронних листів на спам та не спам; діагностика захворювань у медицині (наприклад, визначення наявності або відсутності хвороби); прогнозування відтоку клієнтів у маркетингу; розпізнавання об'єктів у комп'ютерному зорі та інші. На сьогоднішній день методи машинного навчання широко застосовуються у різних галузях науки, техніки та бізнесу. З розвитком обчислювальних потужностей та доступністю великих обсягів даних, створення ефективних моделей для задач бінарної класифікації стає критично важливим. Сучасні алгоритми потребують не лише високої точності, але й оптимальної швидкості роботи та здатності до масштабування. Дослідження в рамках цієї роботи дозволяють розробити нові підходи до вибору та налаштування моделей, а також сприяють покращенню їхньої продуктивності. Результати можуть бути використані для вирішення реальних практичних задач у різних сферах, що підкреслює важливість та затребуваність теми.

Метою роботи є реалізація ефективної моделі машинного навчання для вирішення задач бінарної класифікації, яка забезпечує високу точність роботи.

Для досягнення поставленої мети необхідно *виконати наступні завдання*:

- дослідити методи та технології ML;
- дослідити методи ML для задач класифікації;
- дослідити інструменти для аналізу та обробки даних, фреймворки для роботи з технологіями ML;
- розробити модель ML для розпізнавання зображень в інтерактивному середовищі Jupyter Notebook;
- виконати оцінку якості роботи моделі нейронної мережі.

Об'єкт дослідження – процеси та методи аналізу даних для вирішення задач бінарної класифікації з використанням алгоритмів машинного навчання.

Предметом дослідження є алгоритми, методи та підходи до створення моделі машинного навчання для вирішення задач бінарної класифікації.

Наукова новизна отриманих результатів полягає у розробці нейронної мережі для розпізнавання зображень з використанням класу моделі Sequential, з бінарною класифікацією у кожному нейроні.

Апробація результатів магістерської роботи:

1. Харьков С. А. «Порівняльна характеристика технологій машинного та глибокого навчання». Тези доповіді на II Всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». – Київ, 18 листопада 2024 року.

2. Харьков С. А. «Алгоритми сортування в Python». Тези доповіді на II Міжнародній науково-практичній конференції «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії». – Київ, 25-27 грудня 2024 року.

1 МЕТОДИ АНАЛІЗУ ДАНИХ ТА ЇХ ЗАСТОСУВАННЯ У ЗАДАЧАХ КЛАСИФІКАЦІЇ

1.1 Методи машинного навчання для аналізу даних

Методи машинного навчання для аналізу даних допомагають виявляти структури, закономірності та прогнози на основі наявних наборів даних. Вони поділяються на декілька основних типів в залежності від задачі та типу даних (рис. 1.1). Основними методами є: контрольоване навчання (Supervised Learning), неконтрольоване навчання (Unsupervised Learning), напівконтрольоване навчання (Semi-supervised Learning), навчання з підкріпленням (Reinforcement Learning), методи ансамблевого навчання (Ensemble Methods), методи для аналізу часових рядів (Time Series Analysis), глибоке навчання (Deep Learning), методи для обробки тексту (Natural Language Processing, NLP), аналіз аномалій (Anomaly Detection).

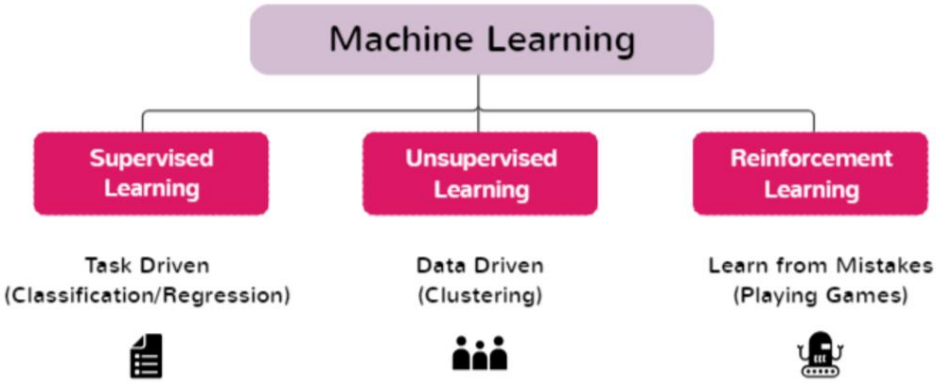


Рис. 1.1 Основні типи машинного навчання

Методи машинного навчання для аналізу даних варіюються від простих підходів, таких як лінійна регресія та кластеризація, до складних моделей глибокого навчання. Вибір методу залежить від типу даних та специфіки задачі: чи це прогнозування, класифікація, кластеризація або виявлення аномалій.

Контрольоване навчання використовується для задач, де є розмічені дані (відомі відповіді). Модель навчається передбачати результати на основі цих даних. Задачами, які відносяться до контрольованого навчання є ***задачі***

класифікації та ***регресії***. Класифікація - аналіз даних для віднесення об'єктів до певних категорій або класів. Приклади: *логістична регресія, метод опорних векторів (SVM), дерева рішень, Random Forest, k-ближчих сусідів (k-NN).* Регресія - прогнозування неперервних значень на основі наявних змінних. Приклади: *лінійна регресія, ридж-регресія (Ridge Regression), градієнтний бустинг (XGBoost, LightGBM).*

Неконтрольоване навчання використовується для роботи з нерозміченими даними, де потрібно виявити приховані структури або патерни в даних. Задачами, які відносяться до контрольованого навчання є задачі ***кластеризації*** та ***зменшення розмірності, асоціативне правило***. Кластеризація – це групування даних у кластери на основі їхньої схожості. Популярним алгоритмом для кластеризації на основі середніх значень є алгоритм *k-середніх (k-means);* групування на основі щільності, що виявляє шуми - *DBSCAN (Density-Based Spatial Clustering of Applications with Noise);* створення деревоподібної структури кластерів - *ієрархічна кластеризація*. Зменшення розмірності використовується для візуалізації та зменшення кількості ознак у великих наборах даних. *Principal Component Analysis (PCA)* - зменшення розмірності шляхом знаходження головних компонент; *t-SNE (t-Distributed Stochastic Neighbor Embedding)* Використовується для візуалізації складних даних у 2D/3D просторі. Асоціативне правило - виявлення правил взаємозв'язку між елементами у великих наборах даних. Алгоритмами для вирішення задач цього типу є *алгоритм Apriori; FP-Growth.*

Напівконтрольоване навчання поєднує розмічені та нерозмічені дані для підвищення ефективності моделі. Використовується, коли розмічені дані доступні в обмеженій кількості. У напівконтролшьованому навчанні використовуються ***графові алгоритми*** для задач класифікації, коли є невеликий набір розмічених даних, наприклад, *Label Propagation* - алгоритм, що розширює мітки з розмічених на нерозмічених даних.

У навчанні з підкріпленням агент вчиться виконувати дії в середовищі, отримуючи винагороди або покарання за свої дії, з метою максимізації загальної

винагороди. Алгоритми, які застосовуються у задач цього типу: *Q-learning; Deep Q-Networks (DQN); політикове градієнтне навчання (Policy Gradient)*.

Методи ансамблевого навчання використовують поєднання кількох моделей для поліпшення точності прогнозів та зменшення варіативності. ***Bagging (Bootstrap Aggregating)*** комбінує кілька моделей, навчених на різних підмножинах даних, щоб зменшити варіативність. Використовує алгоритм *Random Forest.* ***Boosting*** покращує модель, поступово навчаючи нові моделі на помилках попередніх. Використовує алгоритми *Gradient Boosting Machines (GBM); XGBoost; AdaBoost.* ***Stacking*** поєднує виходи різних моделей, використовуючи метамодель для об'єднання прогнозів.

Методи для аналізу часових рядів призначені для роботи з послідовностями даних, що змінюються з часом. *ARIMA (AutoRegressive Integrated Moving Average)* використовується для моделювання часових рядів та прогнозування. *Exponential Smoothing (ETS)* використовується для прогнозування часових рядів шляхом експоненціального згладжування. *LSTM (Long Short-Term Memory)* - тип рекурентної нейронної мережі, що добре працює з послідовностями даних.

Глибоке навчання використовує багатошарові нейронні мережі для аналізу великих та складних наборів даних. ***Convolutional Neural Networks (CNN)*** використовуються для аналізу зображень та даних, що мають просторову структуру. ***Recurrent Neural Networks (RNN)*** використовуються для аналізу послідовних даних, наприклад, часових рядів або тексту. ***Autoencoders*** використовуються для задач зниження вимірності та пошуку аномалій.

Методи для обробки тексту використовуються для аналізу та моделювання текстових даних. Ці методи використовують наступні алгоритми: *TF-IDF (Term Frequency-Inverse Document Frequency)* - визначає важливість слів у документі; *Word2Vec ma GloVe* - алгоритми для створення векторних представлень слів; *Transformers (BERT, GPT)* використовуються для складних задач обробки природної мови, таких як переклад, генерація тексту або відповіді на запитання.

Аналіз аномалій - методи для виявлення аномальних даних або поведінки в наборах даних. *Isolation Forest* - алгоритм, що будує "ліс" із випадкових дерев і

використовує його для виявлення аномалій. *One-Class SVM* - модель, що вивчає "нормальні" дані та виявляє аномалії на основі відхилень.

Задачі з учителем (Supervised Learning) є важливими у машинному навчанні з наступних причин: чіткість навчання, висока точність прогнозів, простота та зрозумілість, широкий спектр застосувань, контрольованість і модифікація, ефективне використання великих обсягів даних, можливість інтерпретації результатів, зворотній зв’язок для покращення моделей. Задачі такого типу є важливими, оскільки вони дозволяють створювати високоточні, інтерпретовані моделі, які можна ефективно використовувати для вирішення реальних завдань на основі великих обсягів даних.

У задачах з учителем модель отримує доступ до розмічених даних, де для кожного прикладу вхідних даних (вектор ознак) відомо правильну відповідь (мітку або результат). Це дозволяє моделі вчитися зі зворотним зв’язком і безпосередньо мінімізувати помилки, порівнюючи прогнозовані значення з реальними. Завдяки цьому процес навчання стає більш контрольованим і ефективним. Оскільки модель навчається на основі реальних відповідей, задачі з учителем часто забезпечують високу точність прогнозів. Це особливо важливо для завдань, де потрібна точність, наприклад, у медичних діагнозах, фінансових прогнозах або системах рекомендацій.

Методи з учителем часто простіші у впровадженні та зрозуміліші в порівнянні з методами без учителя. Оскільки є чітка мета (передбачити мітку), такі моделі легше аналізувати та оцінювати. Наприклад, у задачі класифікації можна легко перевірити точність, повноту або специфічність моделі.

Задачі з учителем мають широкий спектр застосувань:

- *класифікація* - визначення категорії для нового об'єкту (наприклад, спам або не спам);
- *регресія* - передбачення безперервних величин (наприклад, прогнозування ціни акцій або вартості будинку);
- *розпізнавання образів* - моделі можуть використовуватися для ідентифікації об'єктів на зображеннях або відео;

- *прогнозування* - використовується в системах прогнозування погоди, фінансових ринків тощо.

Оскільки модель навчається на відомих даних, її можна легко модифікувати, якщо потрібно поліпшити результати або адаптувати до нових умов. Можна змінювати алгоритми, додавати нові ознаки або більше даних, що дозволяє гнучко реагувати на потреби системи. У сучасних умовах у багатьох сферах накопичуються величезні обсяги даних, і якщо ці дані правильно розмічені, методи з учителем можуть ефективно навчати моделі для вирішення складних завдань. Наприклад, в області обробки природної мови (NLP) або комп'ютерного зору великі розмічені набори даних дозволяють створювати дуже потужні моделі.

У багатьох випадках результати моделей з учителем можуть бути інтерпретовані та пояснені. Наприклад, у моделях регресії можна аналізувати важливість різних змінних (факторів), що полегшує прийняття рішень на основі моделі. Моделі з учителем можуть бути постійно вдосконалені завдяки зворотньому зв’язку від розмічених даних. Якщо модель робить помилки, ці помилки можуть використовуватися для корекції алгоритму, що з часом дозволяє отримувати більш точні результати.

Задачі класифікації є важливими у машинному навчанні, оскільки вони часто виникають у реальних сценаріях, де важливо розподілити об'єкти у різні категорії або класи. Задачі класифікації мають широкий спектр застосувань. Застосовуються у різних галузях, оскільки дуже багато практичних проблем можна звести до класифікації. Наприклад, у сфері медицини можна здійснювати класифікацію пацієнтів на основі симптомів (діагностика хвороби); в області фінансів можна здійснювати класифікацію транзакцій як легітимних або шахрайських; розпізнавання мови і тексту забезпечує класифікацію повідомлень на спам або не спам; в області біометрії можна здійснювати класифікацію зображень для розпізнавання облич або відбитків пальців.

Класифікаційні моделі часто використовуються для прийняття рішень у реальному часі. Наприклад, системи рекомендацій можуть класифікувати продукти або контент для користувача на основі його попередніх дій. Якщо

виконувати аналіз ризиків можна побачити, що моделі класифікуватимуть клієнтів за рівнем кредитного ризику. У системах автономного керування транспортом будуть класифікуватися об'єкти на дорозі, такі як пішоходи або автомобілі.

Задачі класифікації дозволяють автоматизувати рутинні процеси, які раніше виконувались вручну. Це можна дослідити в автоматичному розпізнаванні образів, наприклад, коли в медицині моделі допомагають аналізувати рентгенівські знімки або МРТ для класифікації станів пацієнтів. Також це можна проглянути при обробці документів, де можна здійснити класифікацію юридичних документів за типом, що прискорить роботу юристів і аналітиків.

Машинні алгоритми класифікації дозволяють швидко та точно обробляти великі обсяги даних. Це має велике значення для систем, які повинні оперативно обробляти дані, наприклад, у реальному часі. Сюди вдносять системи кібербезпеки, де потрібно класифікувати мережні активності, визначаючи потенційні загрози та фільтрація контенту: класифікують контент на платформах соціальних медіа для забезпечення відповідності політикам і запобігання поширенню шкідливого контенту.

Для вирішення задач класифікації існують визначені методи та алгоритми. Класифікація є універсальною задачею, яка може бути вирішена різними методами машинного навчання. Це дозволяє вибирати найкращий підхід залежно від складності даних і вимог. Такими підходами можуть бути:

1) *Логістична регресія* для простих задач класифікації.

2) *Класифікаційні дерева рішень (Decision Trees)* для візуалізації та пояснення рішень.

3) *Методи опорних векторів (SVM)* для високоточних класифікацій.

4) *Глибокі нейронні мережі* для складних багатокласових задач, таких як розпізнавання зображень або обробка природної мови (рис. 1.2).

Кожен з алгоритмів для вирішення задач класифікації має свої переваги залежно від типу даних, задачі та вимог до точності, швидкості або інтерпретованості моделі. Вибір алгоритму класифікації залежить від багатьох

факторів, таких як розмір та складність даних, кількість класів, вимоги до точності, інтерпретованість та час виконання. Часто кращий підхід — це спробувати кілька алгоритмів і вибрати той, який показує найкращі результати на конкретному наборі даних.

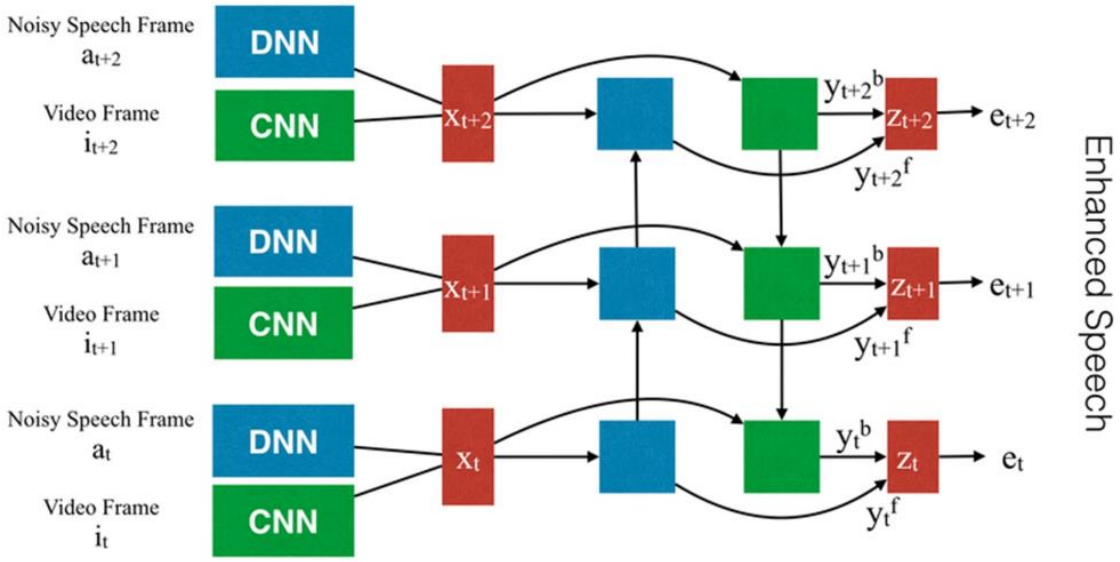


Рис. 1.2 Архітектура мультимодальної гібридної глибокої нейронної мережі

У складних системах, таких як системи діагностики, управління ризиками або рекомендаційні системи, класифікація відіграє ключову роль у прийнятті рішень. Моделі можуть не лише автоматизувати процес, але й надавати рекомендації для прийняття правильних рішень людиною.

Класифікація дозволяє вирішувати як бінарні задачі (два класи), так і багатокласові задачі, де об'єкти можуть належати до кількох категорій. Наприклад: аналіз зображень - розпізнавання різних об'єктів на фотографіях; обробка природної мови - класифікація емоцій у тексті на позитивні, негативні та нейтральні.

Класифікаційні моделі можуть бути досить інтерпретованими. Наприклад, дерева рішень та логістична регресія надають зрозумілі правила або коефіцієнти, що пояснюють, чому даний об'єкт було віднесено до певної категорії. Це важливо у багатьох сферах, де важливий прозорий процес прийняття рішень. Класифікація

є основою для багатьох інших складніших задач машинного навчання. Наприклад, розпізнавання об'єктів у комп'ютерному зорі базується на класифікації зображень; аналіз тональності текстів у завданнях NLP зводиться до класифікації текстів за емоційними категоріями. Таким чином, задачі класифікації є надзвичайно важливими, оскільки вони вирішують широкий спектр проблем, дозволяючи автоматизувати процеси, покращити точність рішень і підтримати складні системи у реальному світі.

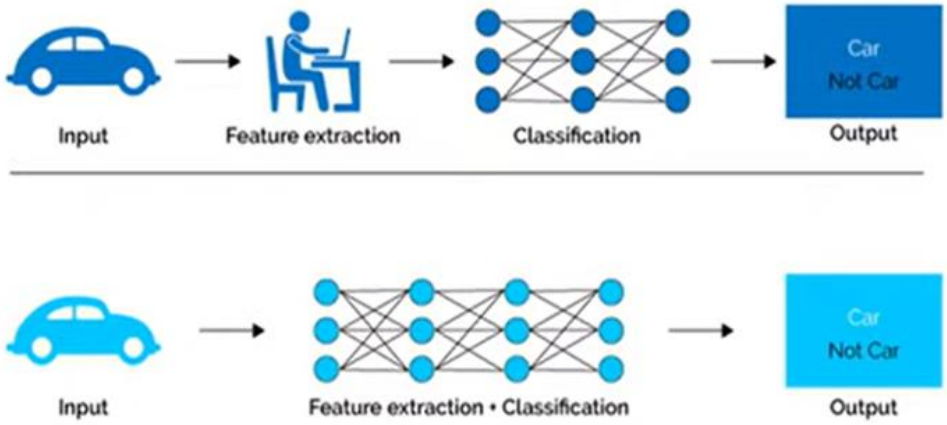


Рис. 1.3 Machine Learning/Deep Learning

Глибоке навчання (Deep Learning) і машинне навчання (Machine Learning) є підрозділами штучного інтелекту (AI), але мають відмінності у підходах, складності та сферах застосування.

Машинне навчання охоплює всі алгоритми, що дозволяють комп'ютерам навчатися на даних без явного програмування.

Глибоке навчання - підрозділ машинного навчання, що фокусується на використанні штучних нейронних мереж із багатьма шарами (глибиною), щоб моделювати складні залежності.

1.2 Використання датасетів у задачах класифікації

Коли ми говоримо про регресію – це коли у нас є якісь точки і ми хочемо писати якусь функцію, щоб поставити кожній точці у відповідність якусь іншу. Це задача регресії. Ми стараємося підібрати якусь функцію або ввести якісь правила. X переходить в Y, де X та Y – це якісь числа.

Задача класифікації – це коли у нас є якісь класи. У нас є X (вектор) і ми хочемо дати відповідь, який це саме клас:

$$X \rightarrow C$$

Є відомий датасет Titanic (те, що ми можемо предиктити) – вижили пасажири чи ні (табл. 1.1).

Таблиця 1.1

Датасет Titanic

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Значення Survive може бути тільки 0 або 1:

titanic	*surive*
	0
	1
	0
	1
	1

Відповідно у нас є дані про пасажирів і відповідь буде: вижила людина чи не вижила. Тобто у нас є клас: виживший або не виживший.

Ще однією популярною задачею є датасет Iris (табл. 1.2). Там є інформація про квітки. Є квітка, є інформація про пелюстку: яка в неї довжина, яка в неї ширина і є така сама інформація про листок (його висота і ширина).

Таблиця 1.2

Датасет Iris

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

Маючи ці 4 числа, ми маємо дати відповідь, а який саме це вид iris. Тобто там є 3 види iris (різні сорти) і за цими чотирма числами *x1, x2, x3, x4* ми маємо поставити у відповідь, який же це клас: iris_1 або iris_2. Або з якими ймовірностями ці iris:

$$x1, x2, x3, x4 \rightarrow \begin{matrix} i_1 \\ i_2 \end{matrix}$$

Це задача класифікації. Для вирішення цієї задачі можна спробувати використати алгоритм *логістичної регресії*. Логістична регресія вирішує задачу: з якою ймовірністю у нас об'єкт належить до певного класу. Ця ймовірність обчислюється за допомогою такої формули 1.1:

$$\acute{p} = h_\Theta(x) = \sigma(x^T\Theta), \tag{1.1}$$

де x^T – у нас є значення *x1, x2, x3, x4* і є вектор Θ - це коефіцієнти біля *x;*

σ - логістична функція, яка має вигляд:

$$\sigma(t) = \frac{1}{1+e^{-t}}. \tag{1.2}$$

Така функція називається сигмоїдальною. Тобто це сигмоїда або називається логістичною функцією. Сигмоїда має вигляд, приведений на рис. 1.4.

Функція наближається до 1, але ніколи її не перетне і так само 0, тобто це дві асимптоми. Ця функція в нулі має значення 0.5. Якщо підставимо t=0, отримаємо 1 поділене на 1 + e^0, тобто результат буде ½. Якщо візьмемо t дуже велику –

відповідно результат ми отримаємо дуже малим і отримаємо значення дуже наближене до 1. Відповідно чим більше *t*, тим більше це наближається до 1. І якщо t буде дуже малим (від'ємним), тоді *e* буде дуже великим числом і в результаті нам потрібно буде 1 поділити на дуже велике число.

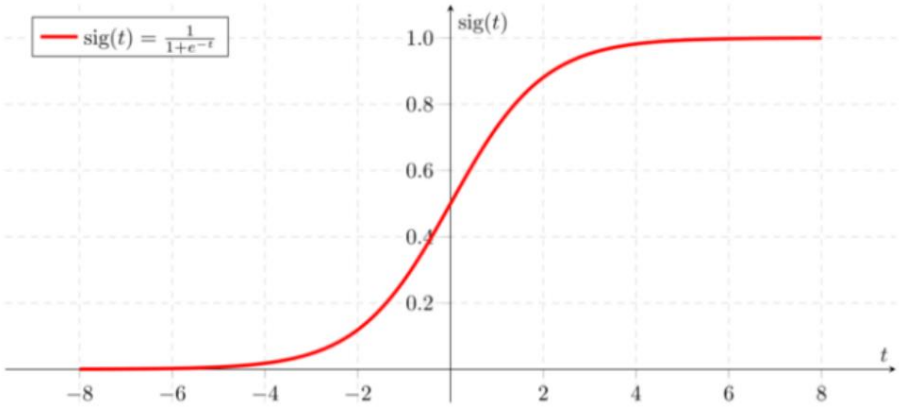


Рис.1.4 Графік сигмоїдальної функції

Нам треба підібрати такі Θ, щоб наша функція виглядала таким чином і у нас буде вироблятися ось такий прогноз:

$$\acute{y} = \begin{cases} 0, & \text{якщо } \acute{p} < 0.5 \\ 1, & \text{якщо } \acute{p} > 0.5 \end{cases},$$

де $\acute{y}$ – прогнозований y.

Потрібно підібрати такі Θ (таку функцію), щоб червоні точки у нас були по одну сторону кривої сигмоїди (справа зверху) від значення 0.5, а чорні - по іншу сторону (зліва знизу) від значення 0.5. При цьому можуть бути викиди. Але нам потрібно підібрати такі параметри Θ, щоб ми могли найточніше розділити наші 2 класи через таку функцію.

Тут аналогічно тому, як у нас була функція похибки, яку нам потрібно було мінімізувати. У регресії це була MSE (середньоквадратична відстань). Тут у нас буде називатися логарифмічна відстань. Наша функція виглядає так:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(\acute{p}^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - \acute{p}^{(i)}\right)\right), \qquad (1.3)$$

де m - кількість зразків (точок).

Ця функція має такий вигляд, так як потрібно все мінімізувати – знайти мінімум цієї функції. Функцію логарифма має вигляд, представлений на рис. 1.5.

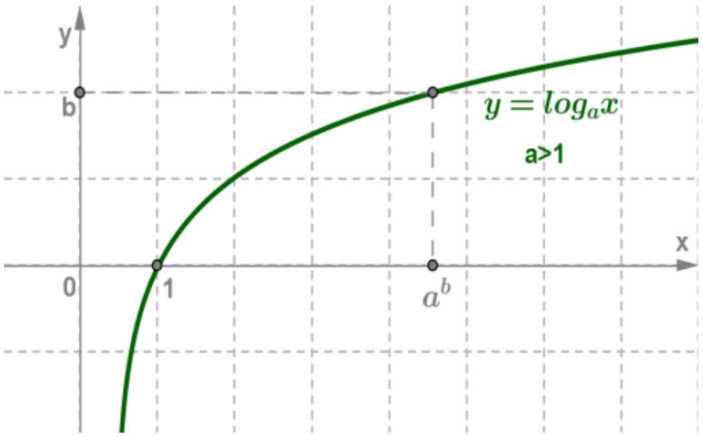


Рис. 1.5 Логарифмічна функція

ṕ – ймовірність, яку ми отримуємо, коли наші параметри Θ множимо на вхідні ознаки x. Ми отримуємо деяке значення p. Ми підібрали параметри Θ, дивимося, яке значення отримали. y=1 – це наша відповідь, тобто той клас, який ми хочемо отримати. Якщо у нас p - велике число, наприклад, *p(i) = 0.9,* відповідно правильна відповідь - це 1, *p* дає нам *0.9*. Якщо *p(i) = 0.9* - ми отримуємо достатньо низьке значення (все добре). Якщо *$(1 – y^{(i)})$* - ця частина стає нулем (ця частина у нас якраз розрахована на ті випадки, коли наша відповідь класу = 0). Тобто, якщо у нас y = 1 – ця половинка поки що не рахується. Тут все добре. У нас значення достатньо низьке. Тобто наше значення буде приблизно: *1* log0.9* (дуже маленьке значення).

Якщо ймовірність *p(i) = 0.1,* отримаємо дуже велике від'ємне число. Із знаком «-» при множенні на 1 у нас буде велика похибка. Якщо у нас клас 0, то навпаки. Якщо у нас 0 і нам дало високу ймовірність і ми знову таки отримаємо високу похибку. Якщо у нас правильна відповідь 0 і наша *p=0*, тоді у нас все добре і функція похибки низька.

Якщо у нас ймовірність велика і відповідь 1, то тоді все добре і у нас мала функція похибки. Якщо у нас ймовірність мала, а відповідь має бути 1, то тоді похибка велика. Якщо у нас правильна відповідь 0 і ймовірність велика, тобто p >

0.5, то тоді також похибка велика. Якщо у нас ймовірність *p* низька і відповідь правильна 0, то тоді похибка маленька і все добре.

І ми стараємося підібрати такі параметри Θ, щоб цю мінімізувати функцію (1.3).

На жаль у нас немає аналітичного розв'язку цього рівняння, тобто ми не можемо знайти мінімум цієї функції і взяти похідну, прирівняти до нуля і вирішити. Але у нас є частинні похідні і по частинним похідним через градієнт це все вирішується. Це принцип роботи звичайної логістичної регресії**.** Але справа в тому, що така регресія працює тільки для двох класів, коли у нас є один по основі і коли у нас є клас 1. Для роботи з багатьма класами потрібно використовувати алгоритм багатозмінної (багатокласової) логістичної регресії.

1.3 Алгоритм логістичної регресії

Якщо ми знаємо як навчити нашу модель, щоб вона змогла класифікувати або це відноситься до класу 0, або це відноситься до класу 1, то ми можемо вирішити таку задачу. Наприклад, у нас є три класи: A, B і C. Тоді ми можемо написати три моделі і підібрати параметри Θ: $A\ \ \Theta_A$

$$B\ \ \Theta_B$$

$$C\ \ \Theta_C$$

У нас будуть всі три різні параметри для кожного класу. І ці параметри будуть вирішувати задачу чи це відноситься до класу A, чи відноситься до класу не A. Цей класифікатор буде знаходити такі параметри, щоб добре розділити B і не B, C і не C: $A\ \ \Theta_A\ \ A,\ \ \neg A$

$B\ \ \Theta_B\ B,\ \ \neg B$

$C\ \ \Theta_C\ \ C,\ \ \neg C$

Тобто ми перетворили багатокласову логістичну регресію на звичайну логістичну регресію, тільки тепер у нас будуть вирішуватися три моделі.

Наприклад, у нас є якісь вхідні ознаки *x* та *y*. У нас є *x1, x2, x3* і у нас може бути клас 1 або клас 2.

X	y
$x1^1x2^1x3^1$	A
$x1^2x2^2x3^2$	B
$x1^3x2^3x3^3$	A
$x1^4x2^4x3^4$	C

То ми можемо це переробити в таку задачу. Для того, щоб нам знайти параметри ось ці (Θ_A по першому класу):

$$A\ \Theta_A\ A,\ \urcorner A$$
$$B\ \Theta_B\ B,\ \urcorner B$$
$$C\ \Theta_C\ C,\ \urcorner C$$

Щоб знайти ці параметри, наш вектор перетвориться в ось такий:

X	y	y_A	y_B	y_C
$x1^1x2^1x3^1$	A	1	0	0
$x1^2x2^2x3^2$	B	0	1	0
$x1^3x2^3x3^3$	A	1	0	0
$x1^4x2^4x3^4$	C	0	0	1

І так абсолютно по всім рядкам ми зможемо знайти всі параметри Θ_A, Θ_B, Θ_C. Зазвичай всі ці вектори Θ_A, Θ_B, Θ_C складають одну велику Θ і це називається матрицею параметрів:

$$A\ \Theta_A\ A,\ \urcorner A$$
$$B\ \Theta_B\ B,\ \urcorner B$$
$$C\ \Theta_C\ C,\ \urcorner C$$

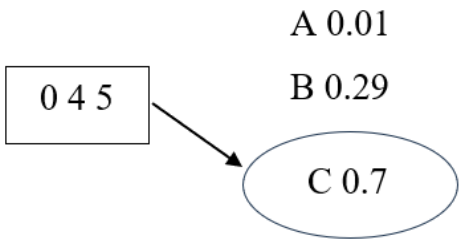
Θ - матриця параметрів.

Після того, як ми отримаємо всі три вектори Θ, нам потрібно буде зробити прогноз. Цей прогноз робиться за допомогою багатовимірної логістичної функції і

пишеться це так: ймовірність того, що даний зразок x належить до класу k, пишеться:

$$\acute{P}_k = \acute{G}(s(x))_k = \frac{e^{S_k(x)}}{\sum_{j=1}^{k} e^{S_j(x)}}. \quad (1.4)$$

Відповідно, коли ми це порахували, то нашим результатом буде той клас, який набрав найбільшу кількість цих балів. Тобто, ось ми натренували три звичайних логістичних регресії (вони попали у багатокласову логістичну регресію і зберігаються у матриці параметрів Θ). Потім, коли нам попадають якісь нові значення x, наприклад, 0, 4, 5 і нам потрібно сказати, який це буде клас: A, B або C, то у нас будуть такі відповіді: що клас A з ймовірністю 0.01, клас B з ймовірністю 0.29, клас C з ймовірністю 0.7. І у нас вибереться максимальний клас і скаже, що наш новий рядок 0 4 5 належить до класу C із ймовірністю 0.7:



Можемо переробити задачу:

$$A \mid B \qquad A, \neg A$$

У задачі регресії наші дані були не обмежені. Тут маємо задачу класифікації: нам потрібно віднести до двох класів (задача бінарної класифікації – клас A і клас B). Замість того, що вирішувати задачу клас A, клас B, ми переробимо її в аналогічну задачу: у нас буде клас A і клас не A (клас A і B не може бути одночасно). Коли до нас попадає якийсь новий зразок x, нам треба дати відповідь число: 1, якщо ми говоримо, що він належить до класу A і 0, якщо не належить до класу A. Ми придумали функцію (рис. 1.1) справа від значення 0.5 якої червоні точки, зліва – чорні, але є деякі викиди.

Вона обмежена від 0 до 1 і записується за формулою (1.2).

Тепер замість t ми хочемо вставити $x^T\Theta$. Тобто у нас є вхідні наші x - вони будуть множитися на якийсь вектор Θ, який нам треба знайти.

Логістична регресія:

$$\acute{p} = h_0(x) = g(x^T\theta), \tag{1.5}$$

$$g(t) = \frac{1}{1+e^{-t}}.$$

І ми хочемо підібрати ось такий вектор Θ, щоб як можна більше червоних точок (ті, що у нас клас А) було справа і вище від 0.5 і щоб як можна більше чорних точок (ті, що у нас не А, тобто В) було зліва і нижче від 0.5.

Таким чином, ми зможемо виміряти їх ймовірності і наш класифікатор буде максимально точним.

Тобто ці Θ будуть знаходитися за допомогою мінімізації функції (1.3):

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(\acute{p}^{(i)}) + (1-y^{(i)})\log(1-\acute{p}^{(i)})).$$

Ця функція дає маленьке значення, якщо у нас класифікатор правильно працює і велике значення, якщо він помиляється. Ми можемо у формулу J(Θ) в $y^{(i)}$ попідставляти 0 і 1, а у $log(\acute{p}^{(i)})$ попідставляти значення 0.9 і 0.1. Якщо у нас значення 0.9, то ми говоримо, що це має бути 1, якщо у нас 0.1 – значить, що це буде 0, тому що ймовірність менша за 0.5. Можемо спробувати попідставляти. Якщо буде співпадати значення 0.9 і 1 – тоді значення буде мале. Якщо будуть помилки типу 0.1 і 1 або 0 і 0.9 – тоді значення похибки буде велике. Алгоритм сатарається знайти мінімум цієї функції.

Багатокласова логістична регресія – це коли у нас є багато класів A, B, C, D і т.д., і тепер ми не можемо розділити це все однією функцією, то тоді ми просто робимо багато маленьких задач і для кожного класу вирішуємо цю проблему, типу всі значення А стають 1, всі ¬А стають значеннями 0. Потім робимо так для колонки В, потім робимо так для колонки С. Знаходимо матрицю всіх параметрів. І потім наш результат буде той, який дав нам найбільшу ймовірність.

2 КЛАСИФІКАЦІЯ МЕТОДАМИ ML/DL

2.1 Використання метрик TN, FP, FN, TP для оцінки якості роботи моделі

Якщо регресію ми знали в принципі, як розуміти її помилку, що можна виміряти середньоквадратичну відстань, то для класифікації не все так просто. У нас може бути 4 випадки. Є спрогнозований клас і фактичний клас. Побудуємо табл. 2.1.

Таблиця 2.1

Спрогнозований і фактичний класи

		Спрогнозований клас	
Фактичний клас		0	1
	0		
	1		

Тобто є у нас справжній (фактичний клас): 0 та 1. А це наш прогнозатор (спрогнозований клас): він нам говорить 0 та 1. Якщо у нас є задача, наприклад, з цими вижившими/не вижившими. У нас є якась сукупність людей: 1 2 3 4 5 6 і є їхні відповіді: чи вижили вони, чи не вижили:

1	0
2	0
3	0
4	1
5	1
6	1

Тобто – це реально фактичні дані, які були: люди під номерами 1, 2, 3 – не вижили, люди під номерами 4, 5, 6 – вижили відповідно. І тепер ми будуємо модель – прогнозатор. Ми даємо їй дані. І тепер цей прогнозатор говорить нам чи вижили вони, чи не вижили насправді. І наш прогнозатор дає такі результати:

1	0	1
2	0	0
3	0	0
4	1	0
5	1	1
6	1	1

Тобто спрогнозований клас 0 і фактичний клас 0 – 2 і 3 сюди попадають. Тепер: фактичний клас 0, а спрогнозований 1 – маємо 1. Тепер: фактичний клас 1, спрогнозований клас 0 – це 4. Фактичний клас 1, спрогнозований клас 1 – це 5, 6. Бачимо фактичні класи і спрогнозовані (табл. 2.2).

Таблиця 2.2

Значення спрогнозованого і фактичного класів

Фактичний клас	Спрогнозований клас	
	0	1
0	2, 3	1
1	4	5, 6

TN – true negative;

FP – false positive;

FN – false negative;

TP – true positive

Таблиця 2.3

Оцінка моделі

Фактичний клас	Спрогнозований клас	
	0	1
0	TN [2, 3]	FP [1]
1	FN [4]	TP [5, 6]

FN означає, що модель помилилася і спрогнозувала негативну оцінку.

FP – це коли фактичний клас 0, а ми помилилися позитивно (помилилися, давши позитивну оцінку - 1).

TN – коли був негативний клас і ми його вгадали.

TP – коли був позитивний клас і наш класифікатор знайшов залежність.

Коли ми робимо класифікатор, нам дуже важливо дивитися на ці 4 числа.

Точність моделі визначається за формулою:

$$\text{точність} = \frac{TP}{TP + FP} \tag{2.1}$$

У нашому випадку точність складає 0.66 (так підібрані числа, що буде 2/3).

Повнота моделі визначається за формулою:

$$\text{повнота} = \frac{TP}{TP + FN} \tag{2.2}$$

У нас вона також буде 0.66.

FN – це коли у нас фактичний клас один, але ми говоримо, що відповідь негативна, а TP – це коли ми вгадали. Повнота – це який відсоток із одиничок всього ми вгадали правильно. Точність – це як точно наша відповідь працює. Коли ми говоримо, що це клас 1, то наскільки точно, що це клас 1 – який відсоток із одиничок, який ми запредиктили, є реально одиничками фактичними. А повнота – це як багато з них ми захватили.

В залежності від задачі, коли ми будемо робити класифікацію – нам потрібно буде дивитися на цю табличку. І залежить вже від нашої бізнес-задачі, що саме для нас важливо, наприклад, умовно ми видаємо якісь кредити у банку (задача банку – який кредит можна погодити). І от нам не страшно відмовляти клієнтам. Нехай 1 – це людина поверне кредит. Нам потрібно як найбільше вгадати ось цих одиничок. І якщо у банка мало грошей, то нам не страшно, щоб ми відмовили комусь у кредиті. Коли фактичний клас 1, а ми запредиктили 0 (FN) – це не дуже страшно (типу ми відмовили людині, але нам гроші дуже важливі). А FP – це дуже страшна помилка, тому що ми будемо давати кредити тим, хто його не

поверне. Тому нам потрібно мати з вами дуже високу точність (ми маємо спиратися на метрику «точність»), а за повноту можемо поки що за неї не переживати. Це основні нюанси класифікації.

Але взагалі, якщо нам потрібно зробити хороший класифікатор, то є міра, яка називається F1:

$$F_{1\text{ міра}} = \frac{2}{\frac{1}{\text{точність}} + \frac{1}{\text{повнота}}} = \frac{TP}{TP + \frac{FN + FP}{2}}. \quad (2.3)$$

Це вважається хорошою мірою, по якій можна скомбінувати і точність, і повноту, і подивитися як класифікатор працює, які його результати.

2.2 Формальна і ймовірнісна постановка задачі класифікації

Методи класифікації, які найчастіше використовуються разом з нейронними мережами для вирішення завдань класифікації - це метод наївного Байєса і метод опорних векторів. На сьогоднішній день метод наївного Байєса є одним з провідних методів багатокласової класифікації через свою простоту використання. В основі алгоритму даного методу лежить теорема Байєса.

Підходів до класифікації існує багато в різних науках і відповідно є велика кількість різноманітних формулювань цього процесу і іноді вони протирічать один одному. Наприклад, класифікація у біології. Ми розбиваємо всіх тварин на царства, потім на види, потім на класи, на родини, види, підвиди і іноді ділимо на популяцію. У математичній статистиці класифікація – це нормалізована задача і вона полягає в тому, щоб ми всю множину наших об’єктів розділили по певним класам за певними ознаками. В основі класифікації з точки зору математичної статистики лежить дискретний аналіз. Дискретний аналіз – це поняття досить широке, тому розглянемо окремі аспекти, які потрібно знати для того, щоб розуміти що таке класифікація об’єктів.

Існують дві постановки задачі класифікації – це *постановка формальна* і *постановка ймовірнісна*. Формальна постановка передбачає: оскільки у об’єкта x_1, x_2, x_3 – певні показники, x_4 знаходиться у визначених межах, то він у нас належить

до даного конкретного класу. Ймовірнісна постановка: оскільки у цього об'єкта вибірки певні характеристики, то ймовірно він відноситься до даного класу.

Формальна постановка. Нехай X – множина опису об'єктів, Y – скінченна множина номерів (імен або міток) класів. Існує невідома цільова залежність – відображення $y^*: X \to Y$, значення якої відомі лише на об'єктах навчальної вибірки $X^m = \{(x_1, y_1)\ldots,(x_m, y_m)\}$. Необхідно побудувати алгоритм $a: X \to Y$, що здатен класифікувати довільний об'єкт x із вибірки X.

Ймовірнісна постановка. Припустимо, що множина пар «об'єкт, клас» $X : Y$ є ймовірнісним простором із невідомою мірою ймовірності p. Є кінцева навчальна вибірка спостережень $X^m = \{(x_1, y_1)\ldots,(x_m, y_m)\}$, згенерована згідно міри ймовірності p. Необхідно побудувати алгоритм $a: X \to Y$, що здатен класифікувати довільний об'єкт x із вибірки X.

У першому випадку у нас на приналежність об'єкту до певного класу впливає характеристика його провідних факторів. У другому випадку на першому місці впливу знаходиться показник ймовірності приналежності до одної або іншої підмножини.

Класифікація методом логістичної регресії відноситься до методу ймовірнісної класифікації, в той час як дерево ухвалення рішень використовує формальний підхід до класифікації. Дерево ухвалення рішень багатокласово реалізує підхід, а логістична регресія реалізує бінарний підхід з точки зору тієї або іншої статистичної гіпотези.

Дослідимо 2 методи, один з який реалізує бінарну класифікацію, інший – багатофакторну. Той метод, який реалізує бінарну класифікацію, реалізує формальний підхід. Метод, який реалізує багатокласову класифікацію, реалізує ймовірнісний підхід.

Перший алгоритм, дослідження якого виконаємо – це наївний алгоритм Байєса.

Наївний алгоритм Байєса – це алгоритм класифікації, оснований на теоремі Байєса з припущенням про незалежність ознак. Іншими словами, НБА передбачає, що наявність будь-якої ознаки у класі не пов'язана з наявністю будь-якої іншої ознаки. Наприклад, фрукт може вважатися яблуком, якщо він червоний, круглий

та його діаметр становить близько 8 сантиметрів. Навіть якщо ці ознаки залежать одна від одної або від інших ознак, у будь-якому випадку вони роблять незалежний внесок у ймовірність того, що цей фрукт є яблуком.

Наївний Байєсовський класифікатор припускає, що наявність функції у класі не пов’язана з якоюсь іншою функцією. Навіть якщо ці ознаки залежать одна від одної або від наявості інших ознак, всі ці властивості незалежно сприяють ймовірності того, що конкретний фрукт є яблуком, апельсином або бананом і саме тому називається «наївний».

Даний алгоритм передбачає, що всі ознаки, які ми використовуємо для класифікації якогось певного об’єкту у множині, між собою не взаємопов’язані, тобто у них немає якогось певного взаємозв’язку. Якщо так, то ми можемо сподіватися з певною долею ймовірності, що якщо у об’єкта колір жовтий або червоний, або зеленуватий, якщо він має округлу форму і якщо він у діаметрі від 10 до 15 см, то це скоріше за все – яблуко. А якщо у нас об’єкт жовтий, у діаметрі 3 см, а форма у нього продовгувата, то це, скоріше за все, банан. При цьому ми думаємо, що у нас форма ніяким чином не залежить від кольору, а колір ніяким чином не залежить від діаметру. І діаметр (а це головна проблема Баєсовського алгоритму) ніяким чином не впливає на форму, хоча ми знаємо, що іноді геометричні показники якраз безпосередньо впливають на форму. Але Баєсовський алгоритм цього не враховує. Ось тому цей алгоритм називається «наївним». Тобто відбувається класифікація об’єктів, але робиться припущення, яке не завжди є коректним.

Як же нам бути, якщо провідні показники мають якість приховані зв’язки один з одним. Для цього спочатку потрібно ці зв’язки виявити. Для того, щоб виявити ці зв’язки, ми можемо використовувати як варіант коефіцієнти кореляції, коефіцієнт парної кореляції Пірсона, якщо у нас цей показник або в один, або в інший бік більший або дорівнює 0.65, то ми з певною долею впевненості можемо сказати, що між цими двома факторами існує певний зв’язок. І коли ми цей зв’язок встановили, нам потрібно ці фактори прибрати з наших розрахунків, з тренувальної вибірки. Якщо ми цього не зробимо, у нас будуть тоді певні

проблеми з точністю, тому що наївний Баєсовський алгоритм цього не враховує, а значить, оскільки приховані зв'язки не враховуються, але вони є, у нас можуть бути якісь не такі значення.

Теорема Байєса дозволяє розрахувати апостеріорну ймовірність P(C|X) на основі P(C), P(X) та P(X|C). Вона показує, як часто відбувається подія C при настанні події X, позначається як P(C|X) і має другу назву апостеріорна (наївна) ймовірність.

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{2.4}$$

де P(C|X) – ймовірність гіпотези C при настанні події X, тобто ймовірність настання C, коли X вже відбулася; P(X|C) – ймовірність настання події, якщо гіпотеза C вірна; P(C) – ймовірність того, що гіпотеза C вірна; P(X) – ймовірність настання події X.

Теорема Баєса теж займається припущеннями ймовірності того чи справджується у нас гіпотеза C при справедливості гіпотези X, а от чи справджується у нас подія 1 при настанні події номер 2. Коли ми це розраховуємо, ми використовуємо таку формулу із вказаними показниками, коли у нас відомі кожен із показників ймовірності і незалежного настання події C, незалежного настання події B і ймовірності настання події, якщо гіпотеза (подія) C у нас настане, то ми можемо розрахувати і основний показник *P*(C|X). Але при цьому у нас ця формула працює, якщо подія C і подія X не залежать одна від одної.

Існують декілька підходів до реалізації алгоритму Байєса на Python. Оскільки алгоритм дуже популярний, він реалізований у багатьох бібліотеках. Є він у бібліотеці Scikit-learn, є у деяких фреймворках, призначених для побудови нейронних мереж. Також ми можемо реалізувати його за допомогою Python коду без використання якихось спеціальних бібліотек.

Для реалізації даного алгоритму ми можемо як варіант імпортувати відомі бібліотеки. З бібліотеки Scikit-learn ми імпортуємо датасет Iris.

2.3 Приклад реалізації наївного алгоритму Байєса в Python

Приклад реалізації наївного алгоритму Байєса в Python приведений на рис. 2.1.

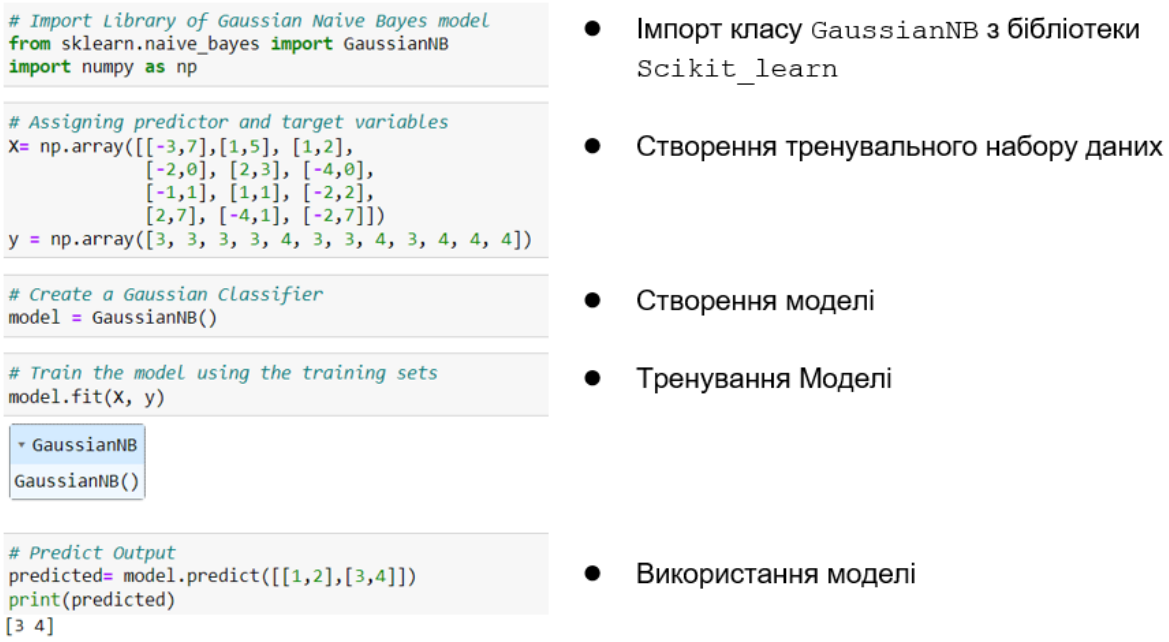


Рис. 2.1 Реалізація наївного алгоритму Байєса в Python

Для обчислення метрик можна використовувати функції із бібліотеки Metrics або ML Metrics.

Позитивними сторонами наївного алгоритму Байєса є те, що класифікація в ньому виконується легко і швидко. Він є одним із самих швидких алгоритмів багатокласової класифікації.

Коли у нас справджується припущення про незалежність наших ознак, наївний алгоритм Байєса показує дуже високі показники точності. Тобто, якщо ми попередньо підготували дані, поприбирали звідти взаємопов’язані ознаки, по-перше, у нас буде хороша метрика, по-друге, порівняно з іншими алгоритмами класифікації для тренування наївного алгоритму Байєса потрібний менший об’єм вибірки. Найголовнішим є те, що даний алгоритм добре працює з категоризованими значеннями. Якщо інші алгоритми класифікації не дуже люблять категоризовані значення і нам потрібно з деяких алгоритмів ці категорії прибирати, то наївний алгоритм Байєса працює з категоріями без проблем.

Є й певні недоліки. Якщо у нас хоча б одна провідна ознака є залежною, у нас різко падає метрика. Якщо у нас у тестовому наборі присутні якісь дані, які відносяться до класів, відсутніх у тренувальному наборі, то ми можемо отримати нульову частоту, коли модель його не прокласифікує. Це теж потрібно враховувати.

Тобто, якщо ми працюємо з цим алгоритмом, по-перше, вибірка може бути невелика, але представники класів її повинні бути всі, які ми будемо використовувати. Якщо з'являється щось нове, то потрібно перенавчати нашу модель.

Даний алгоритм застосовується там, де потрібна швидка класифікація. Алгоритм працює в тих системах, де здійснюється багатокласова класифікація. Це самий популярний алгоритм багатокласової класифікації цифрових даних. Використовується у підсистемах класифікації текстів, фільтрації спаму, аналізі тональності тексту. Даний алгоритм одним із перших був застосований у деяких системах, які аналізували переписку через e-mail, а потім аналізували озвучені слова при підслуховуванні переговорів через операторів мобільного зв'язку у деяких шпигунських системах. Є припущення, що у знаменитій американській системі Echelon використовується цей алгоритм. Даний алгоритм можна використовувати у рекомендаційних системах. Він у поєднанні з моделлю колаборативних фільтрацій дозволяє реалізовувати хороші, досить прості, не вибагливі до ресурсів рекомендаційні системи. Поки ще не існує альтернативи по популярності даному алгоритму.

Застосування наївного алгоритму Байєса:

1) Класифікація реального часу. НБА дуже швидко навчається, тому його можна використовувати для обробки даних у режимі реального часу.

2) Багатокласова класифікація. НБА забезпечує можливість багатокласової класифікації. Це дозволяє прогнозувати ймовірності для багатьох значень цільової змінної.

3) Класифікація текстів, фільтрація спаму, аналіз тональності тексту. При вирішенні завдань, пов'язаних із класифікацією текстів, НБА перевершує багато

інших алгоритмів. Завдяки цьому даний алгоритм знаходить широке застосування в області фільтрації спаму (ідентифікація спаму в електронних листах) та аналізу тональності тексту (аналіз соціальних медіа, ідентифікація позитивних і негативних думок клієнтів).

4) Рекомендаційні системи. Наївний байєсівський класифікатор у поєднанні з колаборативною фільтрацією (collaborative filtering) дозволяє реалізувати рекомендаційну систему. В рамках такої системи за допомогою методів машинного навчання та інтелектуального аналізу даних нова для користувача інформація відфільтровується на підставі спрогнозованої думки користувача про неї.

Таблиця 2.4

Переваги і недоліки наївного алгоритму Байєса

Переваги	**Недоліки**
Класифікація, у тому числі багатокласова, виконується легко та швидко	Якщо у тестовому наборі даних є деяке значення категорійної ознаки, яке не зустрічалося у навчальному наборі даних, тоді модель надасть нульову ймовірність цьому значенню і не зможе зробити прогноз. Це відомо під назвою «нульова частота» (zero frequency). Цю проблему можна вирішити за допомогою згладжування. Одним із найпростіших методів є згладжування за Лапласом (Laplace smoothing)
Коли припущення про незалежність виконується, НБА перевершує інші алгоритми, такі як логістична регресія (logistic regression) і потребує меншого обсягу навчальних даних	Хоча НБА є хорошим класифікатором, значення спрогнозованих ймовірностей не завжди досить точні. Тому не слід покладатися на результати, повернені методом predict_proba.
НБА краще працює з категорійними ознаками, ніж безперервними. Для безперервних ознак передбачається нормальний розподіл, що є досить сильним припущенням	Ще одним обмеженням НБА є припущення про незалежність ознак. Насправді набори повністю незалежних ознак зустрічаються досить рідко

2.4 Класифікація методом опорних векторів

Проведемо дослідження іншого методу класифікації – *методу опорних векторів* (SVM, Support Vector Machine). Метод опорних векторів реалізує

бінарну багатофакторну класифікацію. Він класифікує множину по багатьом факторам, але класифікацію зводить до двох класів як логістична регресія. На відміну від логістичної регресії, цей алгоритм не ймовірнісний, а формальний. Суть роботи методу опорних векторів полягає в тому, щоб знайти у гіперпросторі ознак компонентів нашої множини так звані гіперплощини. Ці гіперплощини розділяють якусь певну вибірку за певними ознаками. Для кожної ознаки робиться своя гіперплощина, яка скаладається з двох площин: класу 1 і класу 2 (рис. 2.2). Це перший етап класифікації.

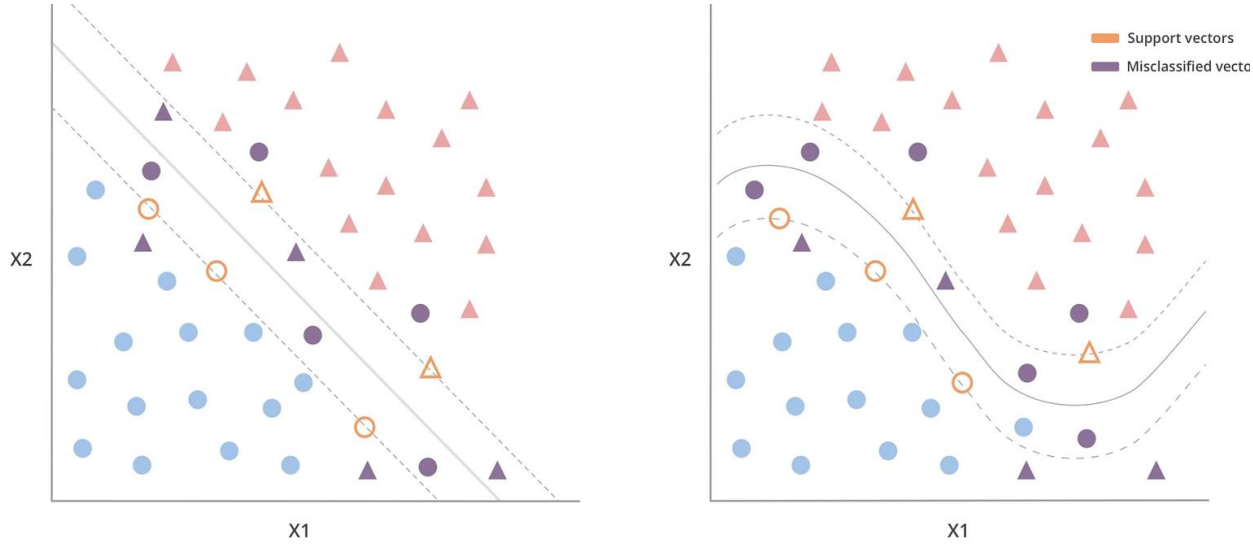


Рис. 2.2 Класифікація методом опорних векторів

Перевага даного методу полягає в тому, що він може визначати як лінійний, так і нелінійний кордон за допомогою функцій ядра, тому він підходить для реальних завдань, де дані не завжди повністю розділяються прямою лінією. SVM робить два важливі припущення:

- дані лінійно розділяються. Навіть якщо лінійний кордон знаходиться у розширеному просторі;
- модель представлена з використанням внутрішніх результатів, що припускає застосування ядер.

Мета SVM - визначити гіперплощину, яка поділяє точки на два класи (рис. 2.3).

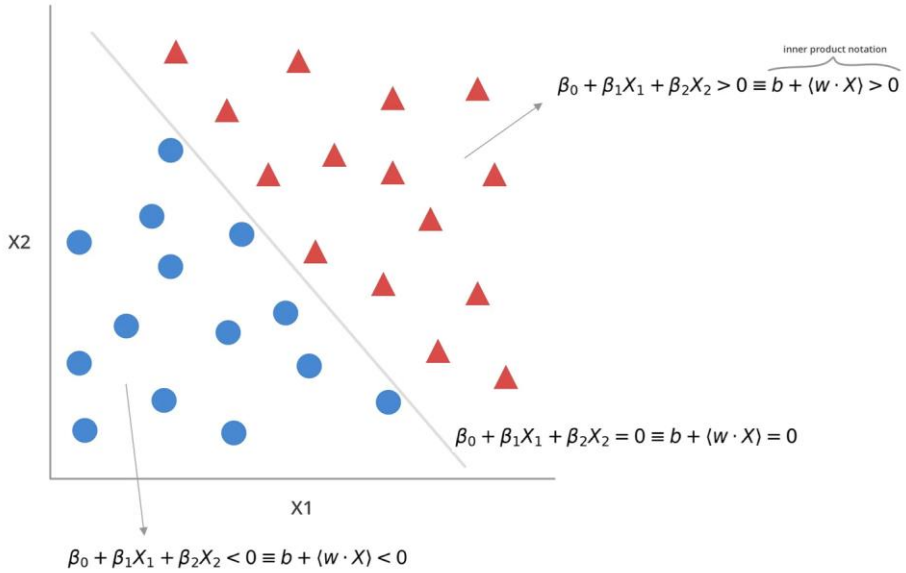


Рис. 2.3 Розділяючі гіперплощини

Кількість таких гіперплощин завжди буде на одиницю менша, ніж кількість факторів, по яких ми проводимо нашу класифікацію.

Крім гіперплощини, даний алгоритм передбачає наявність зазорів (рис. 2.4). Зазори – це певний простір справа і зліва від гіперплощини (або зверху і знизу – в залежності від того, як лягає гіперплощина). Елементи, які потрапляють у цей зазор, вважаються спірними і не класифікуються. Це нам дає те, що даний алгоритм досить точно буде класифікувати більшість елементів нашої вибірки. Тобто він розкидає їх на клас 1 і на клас 2, і ми можемо бути впевненими, що ці елементи дійсно кардинально відрізняються один від одного.

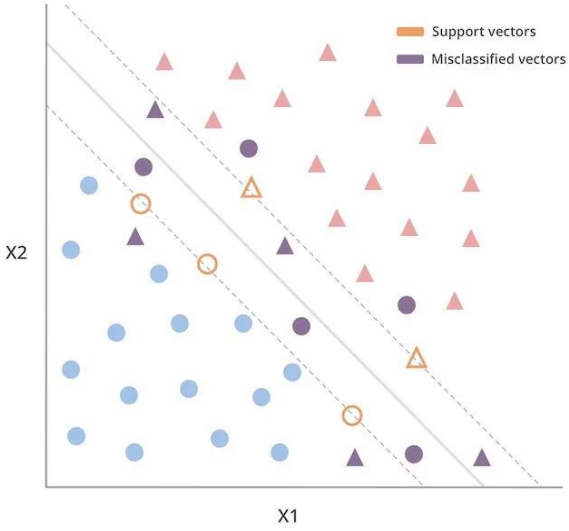


Рис. 2.4 Простір для класифікації

Але, в той же час при використанні даного алгоритму завжди можлива певна кількість втрат наших елементів, які не будуть класифіковані, а які будуть просто потрапляти у ці зазори (справа і зліва від лінії гіперплощини) і ми їх не класифікуємо.

Існують 2 підходи до побудови гіперплощин з зазорами. Це підхід до побудови за допомогою квадратичної кривої (рис. 2.5), тобто це не пряма лінія, а крива. Або ж лінійний зазор (рис. 2.6), коли ми розділяємо нашу множину прямою лінією.

Імпорт бібліотеки, створення і тренування моделі.

```
from sklearn import svm
model = svm.SVC(kernel='poly', degree=2)
model.fit(x_train, y_train)
```

```
SVC
SVC(degree=2, kernel='poly')
```

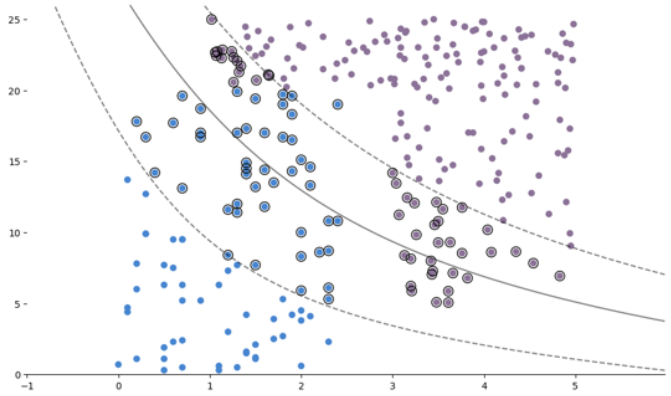


Рис. 2.5 Реалізація методу опорних векторів в Python (зазор квадратичною кривою)

Імпорт бібліотеки, створення і тренування моделі з лінійним ядром.

```
model = svm.SVC(kernel='linear')
model.fit(x_train, y_train)
```

```
SVC
SVC(kernel='linear')
```

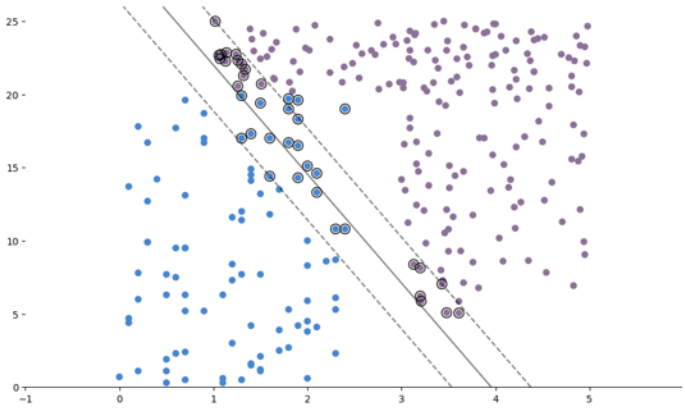


Рис. 2.6 Реалізація методу опорних векторів в Python (лінійний зазор)

В залежності від того з якими даними ми працюємо, у нас найкращі метрики може показати або один, або другий підходи.

SVM добре працює навіть з обмеженими наборами даних, оскільки він використовує лише ключові точки (опорні вектори) для побудови гіперплощини, що розділяє класи. Завдяки максимізації відстані між гіперплощиною розділення та найближчими точками даних, SVM зазвичай забезпечує хорошу узагальнюючу здатність, що знижує ризик перенавчання. SVM підтримує різні типи ядер, що дозволяє адаптувати алгоритм до даних, які важко розділити у початковому просторі. Це робить його придатним для нелінійно роздільних задач.

SVM добре справляється із задачами, де кількість ознак значно перевищує кількість зразків (наприклад, у текстовій класифікації). Підхід SVM базується на чітко визначеній задачі квадратичного програмування, яка має єдине глобальне розв'язання, що забезпечує стабільність результатів. Завдяки використанню ядрових функцій і відбору лише ключових опорних векторів, SVM менш чутливий до зайвих ознак у даних.

Хоча початково SVM є бінарним класифікатором, існують техніки для розширення його на мультикласову класифікацію, такі як методи "один проти всіх" (one-vs-all) або "один проти одного" (one-vs-one).

SVM може ефективно працювати в умовах обмежених зразків даних, що робить його корисним у ситуаціях, де збір великого обсягу даних складний або дорогий.

Метод опорних векторів можна використовувати для вирішення різноманітних задач класифікації, від обробки тексту до розпізнавання образів. Але варто пам'ятати про його обмеження, такі як чутливість до вибору параметрів та обчислювальна складність при роботі з великими наборами даних.

3 РОЗРОБКА МОДЕЛІ ML ДЛЯ ВИРІШЕННЯ ЗАДАЧІ БІНАРНОЇ КЛАСИФІКАЦІЇ

3.1 Робота з даними: масштабування, фільтрація даних, переведення цільового значення у категорію

Виконаємо розробку моделі ML для вирішення задачі бінарної класифікації – повноцінний проєкт для класифікації зображень. Для цього візьмемо чорно-білі картинки 28 на 28 пікселів. Для вирішення завдання скористаємося датасетом MNIST. Його можна взяти з бібліотеки Keras з модуля Datasets. Для цього потрібно завантажити цей датасет командою: *mnist.load_data()* в середовище Jupyter Notebook.

Keras - це високорівневий фреймворк для розробки моделей машинного і глибокого навчання, написаний мовою програмування Python (рис. 3.1). Його основна мета - спростити створення, навчання та тестування моделей нейронних мереж. Keras надає простий і зрозумілий API, який дозволяє легко створювати як стандартні, так і складні архітектури нейронних мереж; дозволяє виконувати низькорівневі налаштування моделей при необхідності; працює як на процесорах (CPU), так і на графічних процесорах (GPU), підтримуючи як локальні обчислення, так і хмарні обчислювальні сервіси.

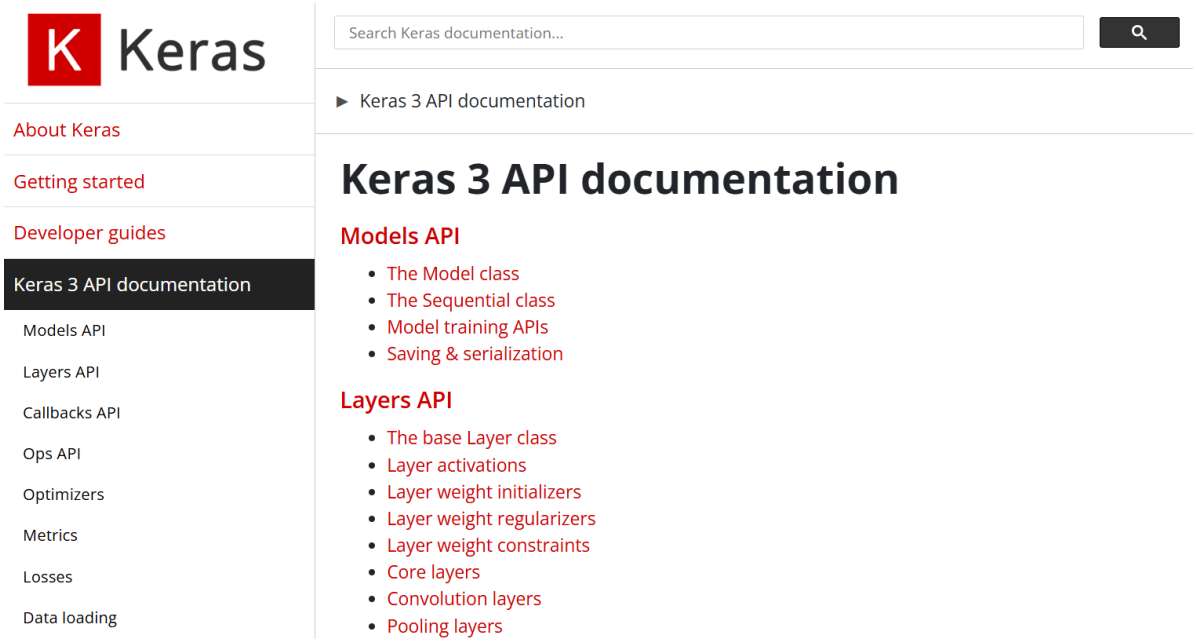


Рис. 3.1 Фреймворк Keras

Даний фреймворк розроблений як зручний і гнучкий інструмент, що дозволяє швидко розробляти прототипи моделей та експериментувати з архітектурами нейронних мереж.

Keras використовується для побудови нейронних мереж для задач комп'ютерного зору, обробки природної мови, генеративних моделей; швидкого створення прототипів моделей машинного навчання; використання готових передтренованих моделей; оптимізації моделей для мобільних і вбудованих пристроїв.

Для розробки моделі ML вибране середовище Jupyter Notebook завдяки перевагам, які забезпечують зручність, гнучкість і ефективність роботи (рис. 3.2). Код у Jupyter Notebook виконується по комірках, що дозволяє поступово розробляти модель, тестувати її окремі компоненти та швидко виявляти помилки. Результати виконання (числові значення, графіки) відображаються одразу під коміркою, що спрощує аналіз даних. Підтримує всі популярні бібліотеки для роботи з даними та машинного навчання.

Jupyter Notebook працює у браузері та підтримується на більшості операційних систем (Windows, macOS, Linux). Хоча Jupyter Notebook здебільшого асоціюється з Python, він підтримує й інші мови програмування, такі як R, Julia, Scala, що розширює його застосування. Jupyter Notebook дозволяє експериментувати з моделями, змінювати параметри й одразу спостерігати за результатами. Це робить його ідеальним для навчання, досліджень і прототипування.

Рис. 3.2 Jupyter Notebook

Jupyter Notebook поєднує інтерактивність, гнучкість та інструменти візуалізації, що робить його ідеальним середовищем для розробки, тестування та документування моделей машинного навчання. Його зручність особливо цінується при навчанні та дослідницькій роботі.

Датасет MNIST підвантажується окремо зразу на навчання і на тестування: (X_train, y_train); (X_test, y_test). MNIST - це набір зображень рукописних цифр розміром 28×28 пікселів у градаціях сірого, що робить його простим для обробки. Легко завантажується й інтегрується у більшість бібліотек машинного навчання (наприклад, Keras, TensorFlow, PyTorch). Відомі метрики точності моделей на цьому датасеті дозволяють порівнювати результати різних підходів і архітектур.

Даний датасет підходить для новачків, оскільки не потребує складної передобробки даних; дозволяє швидко протестувати алгоритми й зрозуміти принципи роботи моделей; зображення мають низьку роздільну здатність (28×28), що робить обчислення швидкими навіть на комп'ютерах без потужних GPU.

Датасет MNIST доцільно використовувати під час розробки моделі машинного навчання, оскільки він є простим, стандартним і добре підтримуваним набором даних. Він дозволяє швидко тестувати алгоритми, оцінювати їх продуктивність і розвивати навички у машинному навчанні.

Потрібно виконати перевірку розмірностей:

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train.shape, X_test.shape
```

```
((60000, 28, 28), (10000, 28, 28))
```

Результат перевірки: на навчанні 60000 зображень розмірністю 28 x 28 пікселів. На тестуванні 10000 зображень 28 x 28 пікселів. Отримано зображення 28 x 28 пікселів:

```
X_train[0].shape
```

```
(28, 28)
```

Для того, щоб подивитися на зображення, їх потрібно відрисувати. Відрисуємо чорно-білими 5 перших зображень:

```
import matplotlib.pyplot as plt

fig,ax = plt.subplots(1, 5, figsize = (15, 10))

for i in range(5):
    ax[i].imshow(X_train[i], cmap='gray')
    ax[i].axis('off')
```

Matplotlib - потужна бібліотека для створення візуалізацій у Python. Вона дозволяє створювати широкий спектр графічних зображень, від простих графіків до складних візуалізацій даних (рис. 3.3). Підтримує інтерактивні графіки у Jupyter Notebook, можливість динамічно змінювати виведення під час виконання програм.

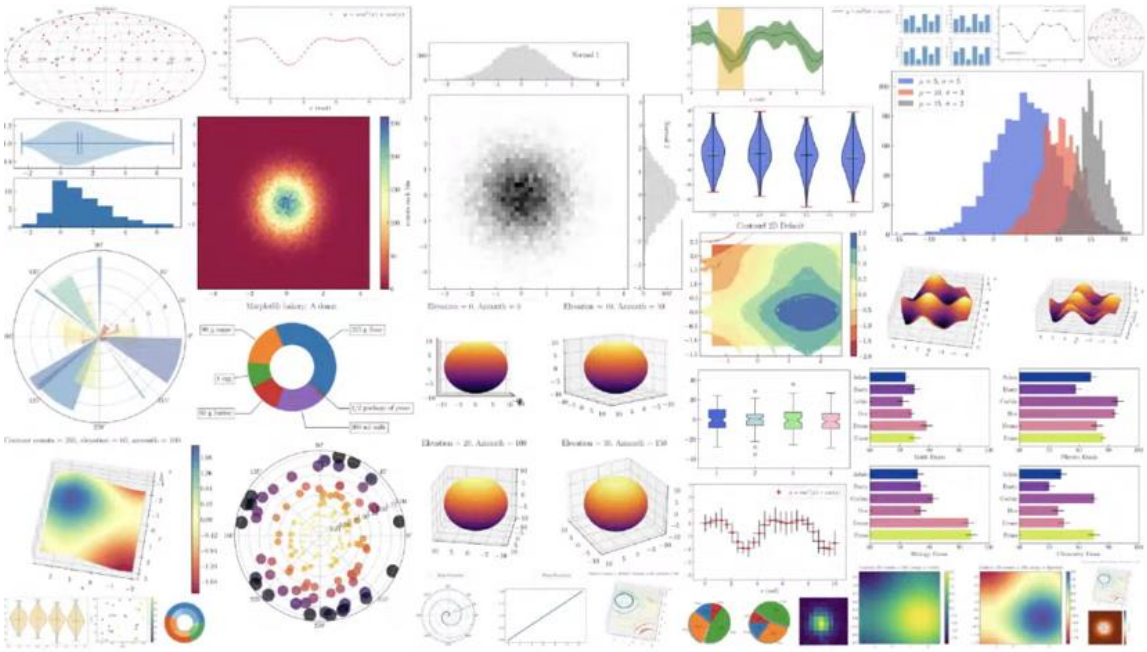


Рис. 3.3 Візуалізація даних

Matplotlib підтримує виведення графіків у різних форматах, таких як PNG, SVG, PDF, EPS, JPG, легко інтегрується з іншими бібліотеками, такими як Seaborn для покращеної візуалізації або Pandas для побудови графіків із DataFrame. Дану бібліотеку оптимізовано для обробки великих обсягів даних, може ефективно працювати з великими наборами точок або великими матрицями даних.

Бібліотеку Matplotlib можна використовувати як у локальних середовищах, так і в онлайн-платформах, таких як Google Colab, Jupyter Notebook, що робить її універсальною для наукових досліджень і аналізу даних. В цілому, це дуже гнучка і потужна бібліотека для візуалізації даних у Python, що пропонує великий набір функцій для побудови графіків і діаграм. Вона є основною бібліотекою для візуалізації даних у науці, техніці та машинному навчанні, завдяки своїй простоті використання, можливості детального налаштування та інтеграції з іншими інструментами Python.

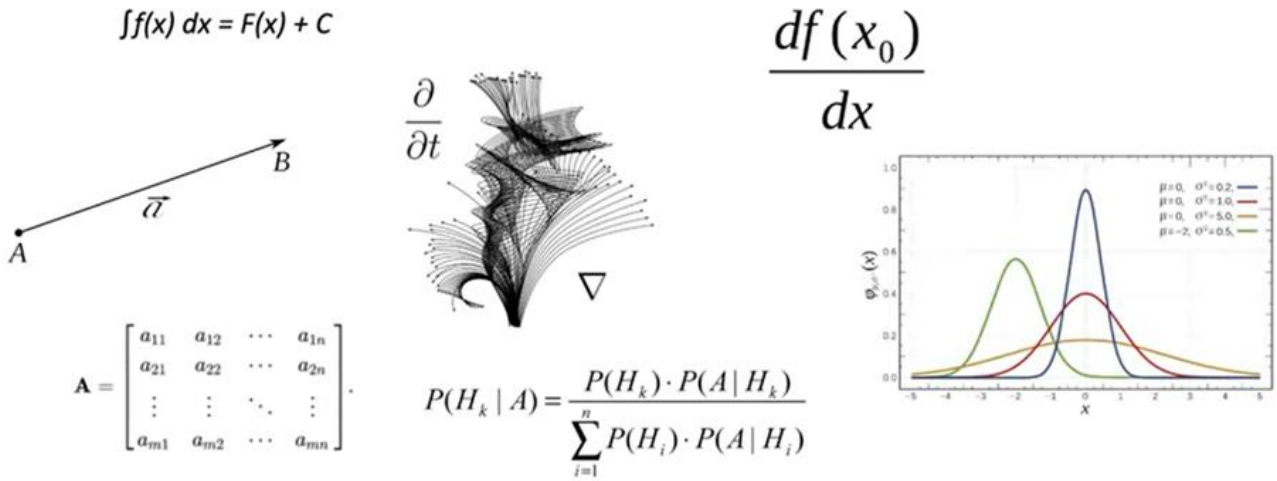


Рис. 3.4 Математичний апарат для обробки даних

Отримали чорно-білі зображення 28 x 28 пікселів чисел: 5, 0, 4, 1, 9. Визначимо розмітку:

```
y_train[:5]
```

```
array([5, 0, 4, 1, 9], dtype=uint8)
```

Використаємо для навчання тільки два класи для того, щоб зробити задачу бінарної класифікації. Візьмемо індекси, що відповідають цим цільовим значенням. Візьмемо цільові значення тільки по цим індексам.

```
idxs = np.where((y_train == 0) | (y_train == 1))
y_train = y_train[idxs]
```

Візьмемо ознаки для навчання по цим індексам:

```
X_train = X_train[idxs]
```

Розмірності:

```
X_train.shape, y_train.shape
```

```
((12665, 28, 28), (12665,))
```

У результаті відбулося стискання датасету. Датасету з 60000 зображень до 12000 і цільових значень відповідно також 12000. Зображення не загубилися і не добавилися нові. Це зроблено на навчанні. На тесті все ще є різні 10 цифр. Відповідно для тестування залишимо тільки нулі і одиниці.

```
idxs = np.where((y_test == 0) | (y_test == 1))
y_test = y_test[idxs]

X_test = X_test[idxs]

X_test.shape, y_test.shape
```

```
((2115, 28, 28), (2115,))
```

Необхідно виконати візуалізацію даних, щоб впевнитися, що отримано нулі і одиниці:

```
fig,ax = plt.subplots(1, 5, figsize = (15, 10))

for i in range(5):
    ax[i].imshow(X_train[i], cmap='gray')
    ax[i].axis('off')
```

Цільові значення - це значення, які ми намагаємося передбачити або класифікувати у задачах машинного навчання. Вони є результатом або виходом, що залежить від вхідних даних і використовуються для навчання моделей.

Цільові значення нулі і одиниці:

```
y_train[:5]
```

```
array([0, 1, 1, 1, 1], dtype=uint8)
```

Піксель можна представляти двома основними способами: від 0 до 255 або від 0 до 1. Для нейронних мереж краще від 0 до 1. Подивимося, в яких масштабах початково представлені пікселі, а потім зробимо нормування.

Нормування даних - це процес трансформації числових даних у певний діапазон або масштаб з метою покращення їхніх властивостей для подальшого аналізу чи обробки. У машинному навчанні нормування є важливим етапом передобробки даних, оскільки воно може значно покращити стабільність і швидкість навчання моделей.

Метою нормування даних є зменшення впливу різних масштабів ознак, прискорення збіжності моделей, покращення точності моделей. Дані з різними одиницями вимірювання або діапазонами значень можуть негативно вплинути на алгоритми, чутливі до масштабу. У випадку градієнтного спуску нормування допомагає уникнути ситуацій, коли великі значення одних ознак домінують над малими значеннями інших. Нормовані дані можуть допомогти моделі краще розрізняти залежності між ознаками.

Масштабування означає зміну величини або діапазону значень даних. Це може бути зроблено шляхом перетворення даних в новий масштаб або діапазон, наприклад, перехід з одного числового діапазону в інший. Масштабування необхідне для покращення роботи моделей машинного навчання, оскільки різні масштаби вхідних характеристик (або змінних) можуть впливати на ефективність навчання та точність моделі. Масштабування може допомогти прискорити процес навчання моделі, оскільки алгоритми оптимізації можуть працювати ефективніше з даними, які мають подібні масштаби.

```
print(X_train.min(), X_train.max())

X_train = X_train / 255.0
X_test = X_test / 255.0

print(X_train.min(), X_train.max())
```

```
0 255
0.0 1.0
```

Бачимо, що у нас мінімум 0, максимум 255. Це дуже погано для нейронної мережі. Тому що різні масштаби і ваги будуть змінюватися погано. Нормуємо дані, не будемо використовувати MinMaxScaler з sklearn, а скористаємося діленням на 255, так як зображення представлені пікселями в діапазоні від 0 до 255, а для нейронної мережі комфортніше навчатися на діапазоні від 0 до 1.

Зробимо нормування діленням на 255. Отримуємо мінімальне значення в нулі, максимальне - в одиниці.

Ньюансом є цільові значення. Нейронна мережа буде видавати ймовірність приналежності до нуля або ймовірність приналежності до одиниці. Тобто це не просто бінарна класифікація, а це у кожному нейроні буде бінарна класифікація. Щоб така мережа навчалася, потрібно з міток 0 та 1 зробити перетворення у бінарний вигляд, зробити деяку категорію. Потрібно змінити вид мітки класу, зараз це 0 або 1, потрібно перетворити у бінарний вигляд.

Отримуємо 2 стовпчики, де перший стовпчик – це мітка чи є зображення 0 класом, другий стовпчик – чи є зображення 1 класом.

```
from keras.utils import to_categorical

y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

print(y_train[:5])
```

```
[0 1 1 1 1]
```

Цільові значення: 0, 1, 1, 1, 1. Дивимося, що у нас вийшло після переведення у бінарну складову. Візьмемо ці 5 об’єктів:

```
y_train_cat[:5]
```

```
array([[1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

У такому форматі у буде відбуватися навчання. Є колонка для того, щоб бути нулем і є колонка для того, щоб бути одиницею. Так мережа буде навчатися краще. Так що із цільових значень із міток 0, 1 ми переводимо у бінарну складову в деяку категорію: бути нулем, бути одиницею.

Необхідно ще спростити вирішення завдання. Зображення мають розмір 28 х 28 пікселів. Це свідчить про те, що нейронна мережа буде навчатися довго і це не означає, що мережа добре навчиться. Потрібно зображення зробити меншими. Для цього необхідно скористатися бібліотекою TensorFlow і її функцією *resize*. Туди необхідно передати нашу вибірку для отримання розмірності 6 х 6. Але при цьому, щоб *resize* відбувся, необхідно змінити розмірність, добавити канали.

Відбувається обробка з чорно-білих зображень і каналу для кольорів немає. Але користувачі звикли бачити картинки у трьохканальному представленні. Це RGB (червоний, зелений, блакитний). Поки зображення чорно-білі, кольорів немає. Тому необхідно створити фіктивну канальність, щоб бібліотека TensorFlow нормально працювала.

TensorFlow - це потужна бібліотека з відкритим кодом для обчислень і машинного навчання, розроблена компанією Google. Вона широко використовується для побудови та тренування моделей машинного навчання, глибокого навчання (рис. 3.5).

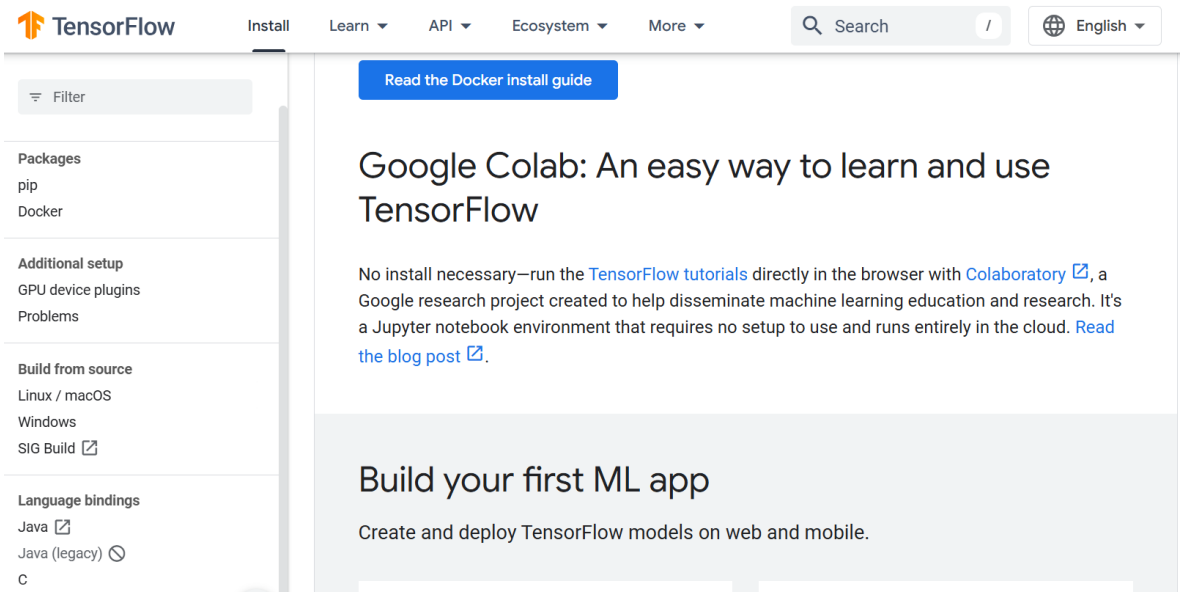


Рис. 3.5 Фреймворк TensorFlow

TensorFlow підтримує виконання на CPU і GPU, а також на TPU (Tensor Processing Unit), що дозволяє значно прискорити тренування моделей, особливо у глибокому навчанні. TensorFlow можна використовувати на різних від тренування на одній машині до розподілених обчислень на багатьох серверах або у хмарних середовищах. TensorFlow має вбудовані функції для створення і тренування нейронних мереж будь-якої складності, включаючи конволюційні нейронні мережі (CNN), рекурентні нейронні мережі (RNN), глибокі автокодери та інші типи мереж.

Бібліотека дозволяє будувати як прості, так і складні архітектури для вирішення різноманітних задач, від класифікації зображень до генерації тексту або прогнозування часових рядів.

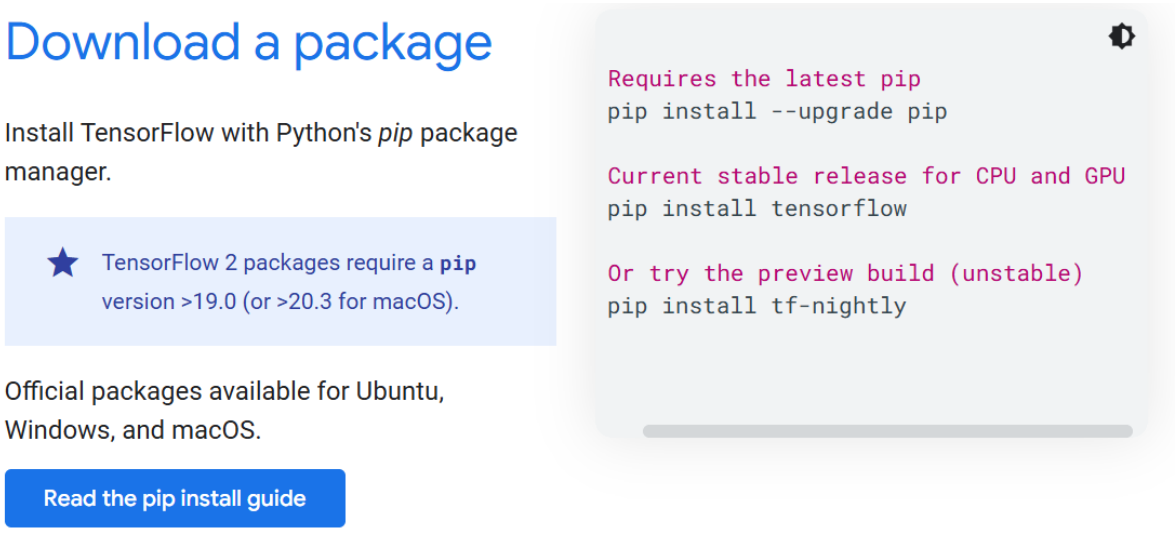


Рис. 3.6 Варіанти встановлення фреймворку TensorFlow

Тензори - це основний тип даних у TensorFlow. TensorFlow автоматично обчислює градієнти через технологію автоматичного диференціювання. Це означає, що для тренування нейронних мереж бібліотека може сама вираховувати похідні функцій та оптимізувати параметри моделі за допомогою алгоритмів, таких як метод градієнтного спуску.

TensorFlow дозволяє тренувати моделі на розподілених системах за допомогою різних стратегій, таких як паралельне тренування на кількох GPU або використання декількох машин. Для розподіленого навчання є можливість використовувати TensorFlow Distributed, що забезпечує ефективне тренування на великих обсягах даних і з великими моделями.

Хоча основна реалізація TensorFlow виконана на Python, бібліотека підтримує й інші мови програмування, такі як C++, JavaScript (TensorFlow.js), Java і Swift. Це дозволяє розгортати моделі у різних середовищах.

Бібліотека TensorFlow дозволяє ефективно тренувати моделі на великих обсягах даних, працювати з нейронними мережами будь-якої складності,

оптимізувати моделі для різних платформ (мобільні, веб, сервери) та інтегрувати з іншими популярними інструментами для обробки та візуалізації даних.

У результаті, щоб простіше навчати нейронну мережу необхідно змінити масштаб зображень з 28 х 28 пікселів зробити масштаб меншим.

```
X_train[..., np.newaxis].shape
```

```
(12665, 28, 28, 1)
```

Потрібно створити фіктивну розмірність, перевести зображення і прибрати фіктивну розмірність.

```
import matplotlib.pyplot as plt

X_train_resized = tf.image.resize(X_train[..., np.newaxis], (6, 6)) [..., 0]
X_test_resized = tf.image.resize(X_test[..., np.newaxis], (6, 6)) [..., 0]

fig, ax = plt.subplots(1, 5, figsize=(15, 10))

for i in range(5):
    ax[i].imshow(X_train_resized[i], cmap='gray')
    ax[i].axis('off')
```

Після виконання вищевказаних дій, розмірність буде вихідною – без каналу, а просто 28 х 28 пікселів:

```
X_train[..., np.newaxis][..., 0].shape
```

```
(12665, 28, 28)
```

Виконаємо візуалізацію. Отримуємо картинки, вони все ще схожі на нулі та одиниці, але значно меншого масштабу. Вони тепер не 28 х 28, а 6 х 6.

Відбулася підготовка даних: по-перше, здійснено масштабування даних (процес перетворення значень змінних до єдиного масштабу, що є важливим кроком попередньої обробки даних, який допомагає покращити продуктивність моделей, особливо чутливих до масштабів вхідних даних), по-друге, здійснено фільтрацію даних, щоб навчатися тільки на нулях та одиницях і по-третє, виконане переведення цільового значення у категорію, щоб було 2 нейрони для передбачення нулів або передбачення одиниць.

3.2 Створення архітектури нейронної мережі

Для того, щоб навчити нейронну мережу для будь-якої задачі потрібно визначитися з архітектурою мережі, розуміти, що потрібно оптимізувати в мережі і яким чином її навчити. На вхід нашої нейронної мережі надходить зображення 6 x 6, яке потрібно перетворити, так як наша мережа поки не вміє працювати з двовимірним входом. Для цього необхідно застосувати шар з Keras flatten, який витягує зображення в один вектор. Зображення 6 x 6 необхідно перетворити у вектор розмірністю 36.

На зараз зображення є двовимірною величиною. Матриця:

```
X_train_resized[0].numpy()
```

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.03594765, 0.9888889 , 0.42941174,
        0.        ],
       [0.        , 0.00228757, 0.7885626 , 0.        , 0.9901961 ,
        0.        ],
       [0.        , 0.65588236, 0.        , 0.        , 0.88235307,
        0.        ],
       [0.        , 0.66078436, 0.5088234 , 0.7235297 , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        ]], dtype=float32)
```

Матричне представлення даних використовує двовимірну структуру, де кожен рядок матриці представляє одне спостереження (об'єкт), а кожен стовпець — одну ознаку або характеристику, яка описує це спостереження. Матричне представлення є ефективним для обробки великої кількості даних з багатьма ознаками. Воно дозволяє виконувати операції на множині даних за допомогою векторизованих обчислень.

Матричне представлення використовується для роботи з великими наборами даних, де кожне спостереження має багато ознак. Це дозволяє одночасно обробляти багато спостережень. Матричні операції, такі як множення матриць, використовуються у багатьох алгоритмах машинного навчання, особливо у методах лінійної алгебри (наприклад, при обчисленні коефіцієнтів у лінійних моделях, тренуванні нейронних мереж).

Матричне представлення даних використовується, коли потрібно обробляти великі набори даних з багатьма спостереженнями і ознаками, наприклад, у

задачах класифікації з великими наборами ознак або у нейронних мережах, де потрібні швидкі матричні обчислення. Матричне представлення дозволяє ефективно працювати з великими наборами даних, де кожен об'єкт або спостереження є окремим рядком, а ознаки — стовпцями.

Розмірність:

```
X_train_resized[0].numpy().shape
```

```
(6, 6)
```

Поки що працюємо тільки з повнозв’язним шаром. Повнозв’язний шар приймає на вхід вектор, а не матрицю. Тому нам потрібно зробити вектор. Це можна зробити через виклик методу flatten:

```
X_train_resized[0].numpy().flatten()
```

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.03594765, 0.9888889 ,
       0.42941174, 0.        , 0.        , 0.00228757, 0.7885626 ,
       0.        , 0.9901961 , 0.        , 0.        , 0.65588236,
       0.        , 0.        , 0.88235307, 0.        , 0.        ,
       0.66078436, 0.5088234 , 0.7235297 , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        ], dtype=float32)
```

Отримуємо вектор. Розмірність:

```
X_train_resized[0].numpy().flatten().shape
```

```
(36,)
```

Тепер цей вектор можна передавати до нейронної мережі. Векторне представлення використовується, коли кожен об'єкт або спостереження у наборі даних представляється одновимірним масивом або вектором значень. Кожен елемент цього вектора зазвичай відповідає певній ознаці (фічі) спостереження.

Вектори є основною математичною структурою, яка підтримує більшість алгоритмів машинного навчання. Вектори є основною формою, у якій дані подаються на вхід більшості моделей машинного навчання. Вектори є невід'ємною частиною математичного представлення даних у машинному навчанні.

Кількість елементів у векторі відповідає кількості ознак, які використовуються для опису одного об'єкта. Наприклад, для спостереження з трьох характеристик

(вік, вага, зріст) вектор буде мати три елементи. Кожне значення у векторі зазвичай має чітке значення, що допомагає при аналізі даних.

Векторне представлення даних корисне для задач з малими обсягами даних або для задач, де потрібно працювати з окремими об'єктами (наприклад, для прогнозування на основі одиничного вхідного вектора). Векторне представлення підходить для опису окремих об'єктів або спостережень, де кожен об'єкт має декілька ознак.

На виході мають бути дві ймовірності: бути або не бути визначеним класом, тому на виході мають бути два нейрони, кожний з яких повинен відповідати за певний клас.

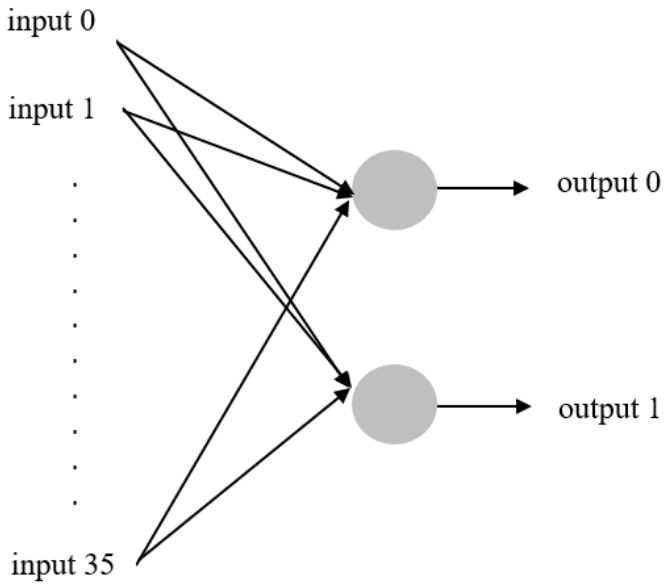


Рис. 3.7 Архітектура нейронної мережі

36 входів – це наші пікселі. Пропускаємо їх через 2 нейрони. Для кожного нейрону на вхід надходить 36 входів. Є байєс. Для другого нейрону – теж саме: 36 входів і байєс. Кожний нейрон на виході видає ймовірності: один нейрон видає ймовірність бути нульовим класом і другий нейрон видає ймовірність бути першим класом. Дана мережа має 1 вихідний шар і 2 вихідних нейрони.

На вихідному шарі потрібно використовувати функцію активації sigmoid, так як вона дозволяє вирішувати задачу бінарної класифікації дуже добре.

Реалізуємо нейронну мережу на Keras:

```
from keras.layers import Flatten
tf.random.set_seed(9)

model = Sequential([
    Flatten(input_shape=(6, 6)),
    Dense(2, activation='sigmoid')
])

model.summary()
```

Щоб постійно зображення не розтягувати, щоб це не вносити в етап передобробки, можемо це внести у саму нейронну мережу. Є спеціальний шар для витягування нашої матриці, нашого зображення у вектор – шар Flatten. На вхід надходить картинка 6 х 6, а на виході буде 36. А далі повнозв'язний шар на 2 нейрони. Так як у нас задача класифікації, то можемо взяти функцію активації сігмоїдою. Активація сигмоїдою використовується для класифікації.

Функції активації є важливим компонентом у нейронних мережах, оскільки вони визначають, як вихідні значення нейронів впливають на подальший процес навчання та прийняття рішень у моделі. Вони вводять непередбачуваність (нелінійність) у модель, що дозволяє нейронним мережам вивчати складні патерни та взаємозв'язки у даних.

Функції активації дозволяють нейронним мережам апроксимувати нелінійні функції. Без функції активації мережа буде просто лінійною комбінацією входів, що обмежує здатність мережі вирішувати складні задачі, такі як класифікація, регресія або розпізнавання зображень. Завдяки функціям активації нейронні мережі можуть розпізнавати складні та високоабстрактні ознаки у даних. Це особливо важливо для задач, пов'язаних з образами, текстами, відео, де є багатошарові патерни.

У багатьох реальних задачах дані мають складні нелінійні залежності і без функцій активації мережа не могла б їх навчитися. Наприклад, у задачах класифікації, де класи можуть бути нелінійно розділені, функції активації дозволяють створити поверхні, які не є лінійними.

Flatten: приймає зображення 6 х 6. На виході – 36. Потім пропускаємо через повнозв'язний шар і на виході 2 нейрони, 2 сигнали. Рахуємо кількість вагів. Вагів – 74, так як для одного нейрону - 36 входів + 1 bias. Для другого нейрону так само. В результаті: (36+1) * 2 = 74 настроюваних вагів.

Отримали модель:

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 36)                0

 dense_5 (Dense)             (None, 2)                 74

=================================================================
Total params: 74 (296.00 Byte)
Trainable params: 74 (296.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Це архітектура, що будемо навчати. Етап створення архітектури нейронної мережі є критично важливим, оскільки саме від цього залежить здатність моделі ефективно навчатися, узагальнювати знання та вирішувати поставлені задачі. Добре продумана архітектура допомагає уникнути перенавчання (overfitting) і дозволяє моделі узагальнювати знання для нових даних. Невдала архітектура може вимагати занадто багато обчислювальних ресурсів або часу для навчання та виконання, що ускладнює роботу з великими наборами даних. Наприклад, занадто велика кількість параметрів може значно збільшити час тренування без помітного поліпшення якості.

Дані можуть бути структурованими (табличними), неструктурованими (зображення, текст, аудіо) або часовими рядами. Кожен тип даних вимагає специфічної архітектури. Ефективна архітектура часто відкриває нові можливості в аналізі даних і покращенні продуктивності. Вибір правильної архітектури визначає здатність нейронної мережі навчатися з даних, бути обчислювально ефективною.

3.3 Вибір функції втрат для оптимізації нейронної мережі

Так як необхідно вирішувати задачу класифікації, для класифікації взяти функцію втрат MSE не можна. Потрібно брати щось інше. І це *бінарна крос-ентропія* (Binary Cross-Entropy, BCE) - функція втрат, яка широко

використовується у задачах класифікації, де є лише два класи. Бінарна, тому що маємо бінарну задача (0 або 1). Крос-ентропія – стандартна функція втрат для класифікації. Вона вимірює різницю між передбаченими значеннями (ймовірностями) і реальними мітками (0 або 1), допомагаючи моделі навчатися максимально точно і передбачати ймовірність належності об'єкта до одного з класів.

Формула бінарної крос-ентропії:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right] \quad (3.1)$$

де N - кількість прикладів у наборі даних (batch size);

y_i - реальна мітка класу (0 або 1);

$\acute{y}_i$ - передбачена ймовірність класу 1 (значення на виході моделі після функції активації, наприклад, sigmoid);

log - натуральний логарифм.

Якщо $y_{i=1}$: враховується тільки перший доданок – $\log(\acute{y}_i)$. Модель отримує штраф, якщо передбачена ймовірність $\acute{y}_i$ далека від 1.

Якщо $y_{i=0}$: враховується другий доданок - $\log(1 - \acute{y}_i)$. Модель отримує штраф, якщо передбачена ймовірність $\acute{y}_i$ далека від 0.

Це забезпечує асиметричну обробку залежно від реальної мітки.

Передбачувані значення $\acute{y}_i$ мають бути у діапазоні від 0 до 1. Тому перед застосуванням бінарної крос-ентропії зазвичай використовується функція активації sigmoid, яка нормалізує вихідні значення моделі до інтервалу [0,1]. Якщо передбачення дуже далеке від істинного значення (наприклад, модель передбачає $\acute{y}_i = 0.01$, коли $y_i = 1$), штраф (значення втрати) буде значно більшим.

Бінарна крос-ентропія використовується в задачах, таких як: двокласова класифікація; багатоміткова класифікація з незалежними мітками (multi-label classification). Ця функція втрат є однією з найбільш ефективних для задач, де метою є передбачення ймовірностей для бінарних результатів.

Нейронну мережу будемо оптимізувати за допомогою стохастичного градієнтного спуску:

```
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics='accuracy')
```

Стохастичний градієнтний спуск (SGD, Stochastic Gradient Descent) — це метод оптимізації, який широко використовується у машинному та глибокому навчанні для мінімізації функцій втрат (наприклад, у задачах регресії або класифікації). Основна ідея цього методу полягає в тому, щоб на кожному кроці виконувати оновлення параметрів моделі (наприклад, ваги нейронної мережі) на основі лише одного випадково вибраного зразка або маленької підмножини навчальних даних. Це контрастує з класичним градієнтним спуском, де для кожного кроку використовуються усі навчальні дані для обчислення середнього градієнта.

Переваги стохастичного градієнтного спуску:

- часто працює швидше на великих наборах даних, оскільки для кожного кроку потрібно лише один зразок або невеликий батч;
- може уникати потрапляння в локальні мінімумі функції втрат завдяки випадковості в оновленнях.

Існують також варіанти стохастичного градієнтного спуску, такі як міні-батчевий градієнтний спуск (mini-batch SGD) і адаптивні методи, такі як Adam, які намагаються усунути деякі недоліки класичного SGD.

Запускаємо навчання:

```
%%time
model.fit(X_train_resized, y_train_cat, epochs=5)
```

Навчаємося на бінарній крос-ентропії. Будемо рахувати метрику accuracy.

Навчаємося на наших нормованих зображеннях, розмір яких змінено (масштабовано), передбачати цільове значення у вигляді категорії. Модель навчається на 5 епохах.

Епохи навчання (або просто епохи) - це кількість повних ітерацій через весь навчальний набір даних під час тренування моделі машинного навчання. Іншими словами, одна епоха означає, що модель "бачить" всі тренувальні дані один раз і

виконує оновлення своїх параметрів (наприклад, ваг у нейронній мережі) для кожного зразка або батчу у процесі навчання.

У кожній епосі модель використовує всі тренувальні зразки для обчислення градієнтів та оновлення параметрів. Після кожної ітерації (або кожного батчу даних) модель коригує свої параметри, щоб мінімізувати функцію втрат (наприклад, зменшити помилку прогнозу). Якщо епох більше однієї, модель проходить цей процес кілька разів.

Більше епох не завжди означає кращі результати. Якщо навчання триває занадто довго, модель може перенавчитися (overfitting), тобто вона стане занадто пристосованою до тренувальних даних і погано працюватиме на нових, невідомих даних. Мала кількості епох може означати, що модель не встигає достатньо навчитися на даних і це може призвести до недонавчання (underfitting). Для кожної епохи зазвичай оцінюється функція втрат або точність, щоб визначити, як модель навчається з часом.

При тренуванні нейронних мереж або інших моделей машинного навчання вибирається певна кількість епох, що є компромісом між достатньою кількістю навчання та униканням перенавчання.

```
Epoch 1/5
396/396 [==============================] - 1s 964us/step - loss: 0.6530 - accuracy: 0.6408
Epoch 2/5
396/396 [==============================] - 0s 978us/step - loss: 0.4952 - accuracy: 0.9389
Epoch 3/5
396/396 [==============================] - 0s 957us/step - loss: 0.4034 - accuracy: 0.9623
Epoch 4/5
396/396 [==============================] - 0s 924us/step - loss: 0.3432 - accuracy: 0.9678
Epoch 5/5
396/396 [==============================] - 0s 976us/step - loss: 0.3011 - accuracy: 0.9700
CPU times: total: 3.22 s
Wall time: 2.21 s

<keras.src.callbacks.History at 0x22c8fdec5e0>
```

Навчання пройшло. Loss був більшим, а став меншим. Accuracy більш показова: була 0.6408, стала 0.9700, тобто прагне до одиниці, а це є ідеальним. Подивимося по передбаченням, що у нас виходить. Мережа навчається, помилка зменшується, метрика стає кращою.

Потрібно перевірити, як модель працює на нових даних. На виході модель надає 2 ймовірності: бути нульовим класом або бути першим класом. Для

вибраного об’єкту ймовірність бути першим класом значно вища, ніж ймовірність бути нульовим класом.

Теорія ймовірності є основою багатьох методів та підходів у машинному навчанні, оскільки вона дозволяє ефективно працювати з невизначеністю та варіацією даних. У реальних задачах дані часто мають шуми, пропущені значення або можуть бути неповними та неточними. Теорія ймовірності дозволяє моделювати та оцінювати невизначеність у цих даних і враховувати її у процесі навчання.

Байєсівські класифікатори або використовують ймовірності для прийняття рішень. У багатьох методах машинного навчання, таких як марковські процеси, гнучкі моделі, латентні простори або нейронні мережі з байєсівським підходом, передбачення або класифікація здійснюються через оцінку ймовірностей. Наприклад, логістична регресія оцінює ймовірність приналежності до певного класу.

Теорія ймовірності дозволяє визначити ймовірність помилок моделі, тобто ймовірність того, що модель дасть невірний прогноз. Це важливо для створення моделей, які можуть надавати ймовірнісні оцінки своїх прогнозів, а не лише "жорсткі" класи або значення. Машинне навчання часто полягає в пошуку оптимальних параметрів або моделей, що найкраще підходять для даних. Використання ймовірнісних методів дає можливість оцінити, яка модель має найвищу ймовірність бути правильною, враховуючи різноманітні варіанти даних.

Генеративні моделі ґрунтуються на ймовірнісних підходах, де вони моделюють ймовірності для кожного класу або вихідного значення і використовують ці ймовірності для генерації нових даних. У багатьох випадках машинне навчання використовується для прийняття рішень в умовах невизначеності, де потрібно врахувати ймовірнісні результати для максимізації вигоди або мінімізації ризику. Теорія ймовірності допомагає оцінити ймовірність успіху або невдачі у таких сценаріях і є необхідною для розуміння та реалізації багатьох методів машинного навчання, оскільки вона допомагає моделювати, аналізувати та обробляти

невизначеність, а також приймати обґрунтовані рішення в умовах обмежених даних.

Робимо передбачення на нашому об’єкті:

```
print("Передбачення нейронної мережі: ")
pred = model.predict(X_test_resized[:1])
pred
```

```
Передбачення нейронної мережі:
1/1 [==============================] - 0s 35ms/step

array([[0.22781794, 0.7562316 ]], dtype=float32)
```

Для одного об’єкту виходить 2 передбачених значення. Ймовірність бути нульовим класом і ймовірність бути першим класом. І по самій великій ймовірності перемагає перший клас. Так що наш тестовий об’єкт ймовірніше всього буде першим класом на думку нейронної мережі. Для цього ми можемо взяти максимальну ймовірність.

Для того, щоб видати фінальну мітку класу, можемо взяти клас, де максимальна передбачена ймовірність. Беремо індекс нашої максимальної ймовірності і це 1.

```
pred_cls = pred.argmax()
pred_cls
```

```
1
```

Подивимося наочно, що це в нас за об’єкти. Перевіримо передбачення візуально (рис. 3.8).

```
idx = 0
plt.imshow(X_test_resized[idx])
plt.title(f'pred{pred_cls}, true {y_test[idx]}');
```

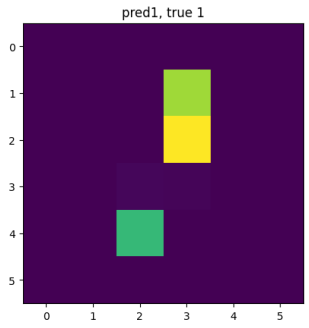


Рис. 3.8 Передбачення моделі

Дійсно одиниця.

Це дуже маленька картинка 6 х 6. І це у нас 1.

Подивимося на вихідне зображення до масштабування. Це у нас також 1. Такий вигляд мало зображення до зміни розміру (рис. 3.9).

```
plt.imshow(X_test[idx])
plt.title(f'pred{pred_cls}, true {y_test[idx]}');
```

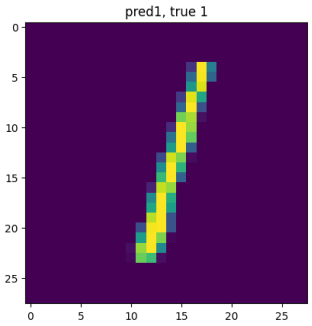


Рис. 3.9 Вигляд зображення до масштабування

Але і на маленькій картинці наша мережа справилася. Так що на одному об’єкті тесту все успішно. Можемо виконати перевірку на всіх даних для навчання. Робимо передбачення на всіх тестових об’єктах.

```
preds = model.predict(X_test_resized)
preds
```

```
67/67 [==============================] - 0s 2ms/step
```

Отримуємо передбачення:

```
array([[0.22781794, 0.7562316 ],
       [0.89985806, 0.14126487],
       [0.19201227, 0.7834291 ],
       ...,
       [0.18991402, 0.74486876],
       [0.81710756, 0.14917615],
       [0.17823091, 0.8084226 ]], dtype=float32)
```

Тут також 2 стовпчики. Один стовпчик для передбачення бути нульовим класом і другий стовпчик для передбачення бути першим класом. Знаходимо найбільшу ймовірність відносно одного об’єкту. Бачимо, що у нас перший об’єкт має більшу ймовірність бути першим класом, тому він перший клас.

Беремо мітку класу, де максимальна ймовірність.

```
preds_cls = preds.argmax(axis=1)
preds_cls
```

```
array([1, 0, 1, ..., 1, 0, 1], dtype=int64)
```

У другому рядку більша ймовірність для нульового класу, тому тут нульовий клас:

```
preds_cls = preds.argmax(axis=1)
preds_cls
```

```
array([1, 0, 1, ..., 1, 0, 1], dtype=int64)
```

У третьому рядку перший клас має більшу ймовірність, тому передбачена мітка – одиничка:

```
preds_cls = preds.argmax(axis=1)
preds_cls
```

```
array([1, 0, 1, ..., 1, 0, 1], dtype=int64)
```

3.4 Обчислення метрики якості роботи нейронної мережі

Метрики якості роботи нейронної мережі - це показники, які використовуються для оцінки її ефективності та точності у процесі навчання та на тестових даних. Вони дозволяють визначити, наскільки добре модель вирішує конкретну задачу, чи мінімізує вона функцію втрат і наскільки точні її прогнози. Метрики можуть варіюватися залежно від типу задачі.

Точність - це одна з найпоширеніших метрик для задач класифікації. Вона вимірює частку правильно класифікованих зразків серед усіх зразків. Визначається за формулою:

$$\text{Точність} = \frac{\text{Кількість правильних прогнозів}}{\text{Кількість всіх прогнозів}} \tag{3.2}$$

Точність особливо корисна, коли класи мають збалансоване представлення у даних.

Функція втрат (loss function) є критично важливою для навчання нейронної мережі. Вона оцінює, наскільки помилковими є прогнози моделі порівняно з реальними значеннями. Типи функцій втрат залежать від типу задачі:

- для класифікації часто використовують крос-ентропійні втрати (cross-entropy loss);
- для регресії використовують середньоквадратичну помилку (MSE, mean squared error).

$$\text{Втрати} = \frac{1}{n}\sum_{i=1}^{n} \text{Функція втрат}(y_i, \hat{y}_i) \quad , \tag{3.3}$$

де y_i — реальні значення;

$\acute{y}_i$ – прогнозовані значення.

Метрики точності, повноти і F1-міри використовуються у задачах класифікації та особливо корисні при незбалансованих даних.

Точність (Precision) - це частка правильних позитивних прогнозів серед усіх прогнозів, що були класифіковані як позитивні:

$$\text{Точність} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.4}$$

Повнота (Recall) - це частка правильних позитивних прогнозів серед усіх реальних позитивних прикладів:

$$\text{Повнота} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.5}$$

F1-міра — це середнє гармонійне між точністю і повнотою:

$$\text{F1-міра} = 2 \times \frac{\text{Точність} \times \text{Повнота}}{\text{Точність} + \text{Повнота}} \tag{3.6}$$

F1-міра корисна, коли потрібно зберігати баланс між точністю та повнотою, особливо в задачах з незбалансованими класами.

AUC-ROC - це метрика, що використовується для оцінки порогової чутливості моделі, за допомогою кривої прийому характеристик (ROC). Вона вимірює здатність моделі правильно класифікувати позитивні та негативні зразки при різних порогових значеннях:

- AUC (площа під кривою) показує, наскільки добре модель відрізняє позитивний клас від негативного;
- чим вище AUC, тим краще модель здатна класифікувати дані.

MAE вимірює середню величину помилки між передбаченими і реальними значеннями у задачах регресії. Визначається як середнє значення абсолютних відмінностей:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.7}$$

MAE дає уявлення про те, на скільки в середньому прогнози відрізняються від реальних значень.

MSE вимірює середній квадрат різниці між передбаченими і реальними значеннями:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.8}$$

MSE має тенденцію підвищувати значення великих помилок, оскільки помилки підносяться до квадрату.

Коефіцієнт детермінації R^2 використовується для оцінки того, наскільки добре модель відтворює реальні значення у задачах регресії. Він вимірює, яку частину варіації у даних може пояснити модель:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad , \tag{3.9}$$

де $\bar{y}$ — середнє значення реальних даних.

Перехресна перевірка (Cross-Validation) використовується для оцінки якості моделі, шляхом її тестування на різних підмножинах даних. Це дає змогу оцінити стабільність і надійність моделі.

Вибір метрики залежить від типу задачі і особливостей даних:

- для задач класифікації доцільно використовуються точність, точність/повнота, AUC-ROC;

- для задач регресії доцільно використовувати втрати, MAE, MSE, R^2;

- для задач з незбалансованими класами важливими є точність, повнота, F1-міра та AUC.

Правильний вибір метрики дозволяє більш точно оцінити якість моделі і допомагає у подальшому поліпшенні її продуктивність.

Виконаємо обчислення метрики якості *accuracy* для нашої задачі:

```
from sklearn.metrics import accuracy_score
print(f'test acc: {accuracy_score(y_test, preds_cls)*100:.2f}% ({(y_test == preds_cls).sum()} out of{y_test.shape})')
```

```
test acc: 97.68% (2066 out of(2115,))
```

Виконане порівняння наших справжніх значень з передбаченими класами. І обчислена метрика. Точність моделі становить 97.68%. Таким чином ми навчили нейронну мережу для задачі бінарної класифікації. Визначено правильно 2066 об’єктів з 2115. Модель спрацювала успішно.

Метрика accuracy легко інтерпретується. У випадку збалансованих класів (коли кількість прикладів кожного класу приблизно однакова), accuracy є дуже гарною метрикою. Вона дає точну оцінку того, скільки загальних помилок було зроблено. У таких випадках точність добре відображає загальну ефективність моделі.

Коли ми лише починаємо тренувати модель, accuracy може бути корисною для першої оцінки її ефективності. Якщо точність є досить високою, це може свідчити про те, що модель налаштовується належним чином. Іноді у задачах класифікації потрібно просто зрозуміти загальний рівень точності і якщо втрата або помилка кожного класу (позитивного або негативного) є приблизно однаковою у контексті задачі, accuracy є хорошою метрикою. Accuracy є універсальною метрикою, яка підходить для багатьох типів моделей і може бути використана як стандартна метрика у задачах класифікації, незалежно від алгоритму (наприклад, логістична регресія, SVM, дерева рішень, нейронні мережі).

Для простих задач, де класи сильно не перекриваються (наприклад, у задачах з чітко визначеними позитивними та негативними прикладами), accuracy може бути достатньою метрикою для оцінки продуктивності.

ВИСНОВКИ

У результаті виконання магістерської роботи було розроблено модель машинного навчання на фреймворку Keras для вирішення задач бінарної класифікації. Розроблено нейронну мережу. Узагальнюючи алгоритм, порядок дій, який був виконаний, щоб нейронна мережа навчалася, по-перше, потрібно виділити дані, так як необхідно отримати вибірку для навчання. Опційно масштабування даних – це нормалізація, стандартизація або звичайне ділення на 255. Далі можна масштабувати дані, але це більш актуально для картинок і якщо у нас класифікація, то потрібно перевести мітки класів у бінарне представлення під деякі категорії. Перший етап у навчанні нейронної мережі пов’язаний з роботою з даними. Отже, на першому етапі розробки і навчання нейронної мережі необхідно отримати вибірку для навчання: масштабування даних (MinMaxScaler, StandardScaler, /255.0); Resize даних за необхідності; для класифікації потрібно перевести мітки класів у бінарне представлення.

Другий етап – це створення архітектури нейронної мережі: необхідно вибрати кількість входів; вибрати кількість шарів; вибрати кількість вихідних нейронів; вибрати функцію активації. На цьому етапі у нас вже є мережа.

На третьому етапі потрібно визначити, оптимізацію чого необхідно провести. Потрібно вибрати функцію втрат. Із стандартних і які часто зустрічаються – це MSE для регресії, а для класифікації – бінарна крос-ентропія на два класи (binary_crossentropy) або ж категорійна крос-ентропія (categorical_crossentropy), якщо класів більше, ніж два.

На четвертому етапі потрібно визначити, яким чином оптимізувати модель нейронної мережі. Потрібно вибрати оптимізатор. Із стандартних та звичних: sgd, adam.

Потім модель потрібно підготовлювати до навчання через компіляцію: .compile(). У компіляцію потрібно закласти оптимізатор, функцію втрат і опційно метрики.

На наступному етапі модель потрібно навчати: .fit().

На виході потрібно подивитися, що виходить по результатам, порівняти з істинною і якщо нас все влаштовує, наша модель працює добре.

Точність моделі, розробленої у роботі, становить 97.68%. Для навчання нейронних мереж необхідно багато практики.

Задача бінарної класифікації є однією з базових і ключових у машинному навчанні через її широке застосування та фундаментальний характер, знаходить застосування у багатьох галузях. У таких задачах, як пошукові системи або рекомендаційні системи, бінарна класифікація може допомогти визначити релевантність документу або продукту. Бінарну класифікацію можна адаптувати для розв’язання багатьох задач, навіть якщо початкові дані не очевидно підходять до цієї категорії. Наприклад, перетворення регресійної задачі в задачу бінарної класифікації через встановлення порогів.

ПЕРЕЛІК ПОСИЛАНЬ

1. https://pytorch.org/
2. https://www.tensorflow.org/
3. http://realpython.com/python3-object-oriented-programming/
4. http://leetcode.com/problemset/all/
5. http://perso.limsi.fr/pointal/
6. http://media/python:cours:mementopython3-english.pdf
7. https://numpy.org/
8. https://matplotlib.org/
9. Yang, Guanyu et al. (2020). “Weakly-supervised convolutional neural networks ofrenal tumor segmentation in abdominal CTA images”. In: BMC Medical Imaging 20.1, p. 37. ISSN: 1471-2342. DOI: 10.1186/s12880- 020- 00435- w. URL: https://doi.org/10.1186/s12880-020-00435-w.
10. Heller, Nicholas et al. (2020). The KiTS19 Challenge Data: 300 Kidney Tumor Cases with Clinical Context, CT Semantic Segmentations, and Surgical Outcomes. arXiv: 1904. 00445 [q-bio.QM].
11. Hollo, Kaspar (2019). Exploring the Value of Weakly-Supervised Deep Learning Approaches for Artefact Segmentation in Brightfield Microscopy Images. URL: https://comserv. cs . ut . ee / home / files / hollo _ softwareengineering _ 2021.
12. Wang, Haofan et al. (2020). Score-CAM: Score-Weighted Visual Explanations for Convo-lutional Neural Networks. arXiv: 1910.01279 [cs.CV].
13. Лі, Кай-Фу Штучний інтелект: 10 передбачень для майбутнього / Кай-Фу Лі, Чень Цюфань; пер. з англ. А. Райчук. – Київ: Форс Україна, 2022. – 464 с.
14. Selvaraju, Ramprasaath R. et al. (2019). “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: International Journal of ComputerVision 128.2, pp. 336–359. DOI: 10 . 1007 / s11263 - 019 - 01228 - 7. URL: https://doi.org/10.1007\%2Fs11263-019-01228-7.
15. Jake VanderPlas Python Data Science Handbook: Essential Tools for Working with Data, 2022, 551p.

16. Грас Д. Data Science. Наука про дані з нуля: Пер. з англ. – 2-е видання., перероб. та доп. – 2021. – 416 с.

17. Лосєв М.Ю. Програмування мовою Python: навчальний посібник / М. Ю. Лосєв, В. М. Федорченко. – Харків, - Львів: Видавництво ПП «Новий Світ - 2000», 2023. – 178 с.

18. Золотухіна О.А., Негоденко О.В., Резник С.Ю., Разіна С.Я. Якість та тестування інформаційних систем. Навчальний посібник підготовлено до друку для самостійної роботи студентів вищих навчальних закладів. Київ: ННІІТ ДУТ, 2020. – 128 с.

19. Зінченко О.В., Фесенко М.А., Березівський М.Ю., Кисіль Т.М. Технології смарт-систем. – Навчальний посібник. – К.: ДУІКТ, 2023. – 162 с.

20. Нікольський Ю. В., Пасічник В. В., Щербина Ю. М. Системи штучного інтелекту: навчальний посібник. – Львів: «Магнолія-2006», 2024. – 279 с.

21. Hua, M., Wang, Y., Li, C., Huang, Y., Yang, L. UAV-aided mobile edge computing systems with one by one access scheme. IEEE Trans. Green Commun. Netw., 3(3), 2019. — 664-678 с.

22. Multiple Access Schemes for Cellular Systems. Electronic Notes, Oct. 2020. — [Електронний ресурс]. Режим доступу: https://www.electronics-notes.com/articles/connectivity/cellular-mobile-phone/multiple-access-schemes-technology-techniques.php

23. What’s the Difference Between FDD and TDD. Electronic Design, Oct. 2020. — [Електронний ресурс]. Режим доступу: https://www.electronicdesign.com/technologies/communications/article/21801257/whats-the-difference-between-fdd-and-tdd

24. Yao, H., Mai, T., Jiang, C., Kuang, L., Guo, S. AI Routers & Network Mind: A Hybrid Machine Learning Paradigm for Packet Routing. IEEE Computational Intelligence Magazine, 14(4), 2019. — 21-30 с.

25. Li, L., Wen, X., Lu, Z., Pan, Q., Hu, W. J. A. Z. Energy-efficient UAV-enabled MEC system: Bits allocation optimization and trajectory design. Sensors, 19(20), 2019. — 4521 с.

26. Wu, G., Miao, Y., Zhang, Y., Barnawi, A. Energy efficient for UAV-enabled mobile edge computing networks: Intelligent task prediction and offloading. Comput. Commun., 150, Jan. 2020. — 556-562 c.

27. Mozaffari, M., Saad, W., Bennis, M., Nam, Y.-H., Debbah, M. A tutorial on UAVs for wireless networks: Applications challenges and open problems. IEEE Commun. Surveys Tuts., 21(3), 2019. — 2334-2360 c.

28. Wang, Y., Ru, Z.-Y., Wang, K., Huang, P.-Q. Joint deployment and task scheduling optimization for large-scale mobile users in multi-UAV-enabled mobile edge computing. IEEE Trans. Cybern., 50(9), 2020. — 3984-3997 c.

29. Zhang, J., Zhou, L., Tang, Q., Ngai, E. C.-H., Hu, X., Zhao, H., et al. Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing. IEEE Internet Things J., 6(2), 2019. — 3688-3699 c.

30. Nguyen, V., Khanh, T. T., Nam, P. V., Thu, N. T., Hong, C. S., Huh, E.-N. Towards flying mobile edge computing. Proc. Int. Conf. Inf. Netw. (ICOIN), Jan. 2020. — 723-725 c.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)