МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Факультет менеджменту
Кафедра економічної кібернетики, комп'ютерних наук та інформаційних технологій

ШТУЧНИЙ ІНТЕЛЕКТ ТА МАШИННЕ НАВЧАННЯ

Конспект лекцій
для здобувачів першого (бакалаврського) рівня вищої освіти
ОПП «Комп'ютерні науки» спеціальності F3(122) «Комп'ютерні науки» денної форми здобуття вищої освіти

МИКОЛАЇВ
2025

УДК 004.8
Ш94

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету від 27 березня 2025 року, протокол № 7.

Укладачі:

О.В.Шебаніна — д-р екон. наук, професор, декан факультету менеджменту Миколаївського національного аграрного університету;

С. І. Тищенко — к.п.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. Ю. Пархоменко — к.ф.-м.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О.О. Жебко — асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій, Миколаївського національного аграрного університету;

А.М.Коломієць — асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій, Миколаївського національного аграрного університету;

Рецензенти:

О. С. Садовий - канд. техн. наук, доцент, завідувач кафедри агроінженерії Миколаївського національного аграрного університету

Ю. В. Грицук - канд. техн. наук, доцент кафедри загальної інженерної підготовки Донбаської національної академії будівництва і архітектури

Ш94 **Штучний** інтелект та машинне навчання : конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності F3(122) «Комп'ютерні науки» денної форми здобуття вищої освіти / уклад. О. В. Шебаніна, С. І. Тищенко, О. Ю. Пархоменко, О.О. Жебко, А. М. Коломієць. Миколаїв : МНАУ, 2025. 192 с.

УДК 004.8

ЗМІСТ

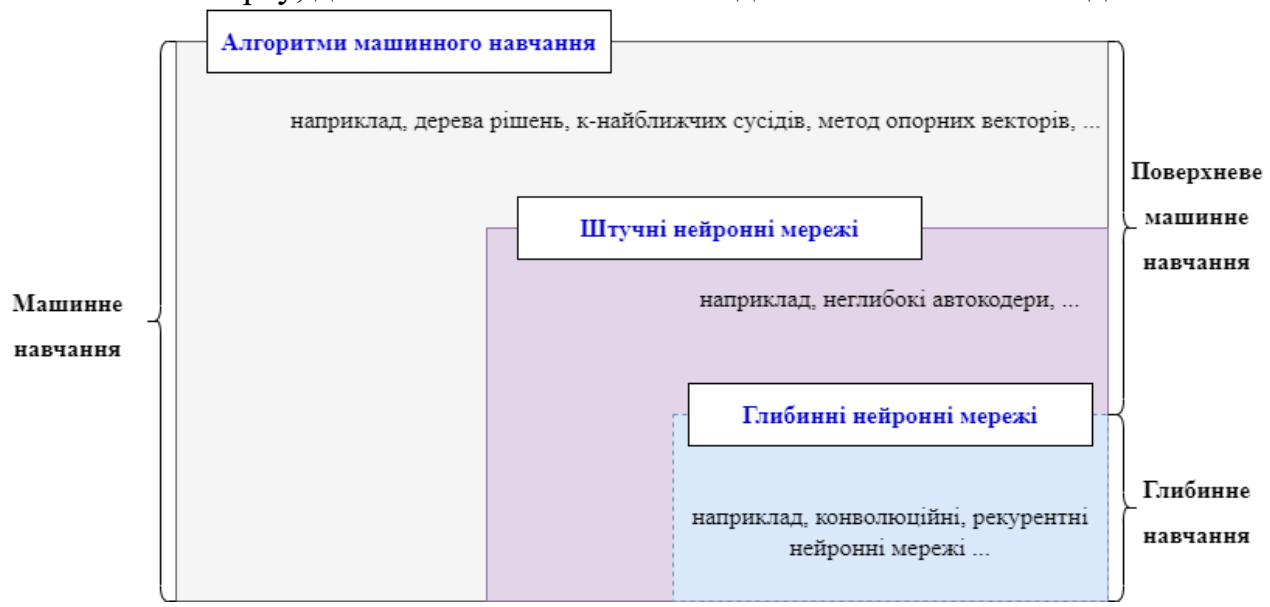
Змістовний модуль 1. Класичне машинне навчання 4
ТЕМА 1. Статистичний аналіз. R. Використання базових типів даних та конструкцій 4
ТЕМА 2. Обробка наборів даних. Введення в статистичний аналіз в R 15
ТЕМА 3. Вступ до штучного інтелекту 28
ТЕМА 4. Машинне навчання 40
ТЕМА 5. Класичне машинне навчання 53
ТЕМА 6. Навчання без вчителя та навчання з підкріпленням 81
Змістовний модуль 2. Глибинне навчання. Data Minig 98
ТЕМА 7. Ансамблеві методи. Глибинне навчання 98
ТЕМА 8. Проєктування нейронних мереж 116
ТЕМА 9. Дифузійні нейронні мережі 125
ТЕМА 10. Stable Diffusion. LoRa 132
ТЕМА 11. Дані та шкали попередня обробка даних 142
ТЕМА 12. Випадкові велечини 149
ТЕМА 13. Методи та стадії Data Mining. Закони розподілу 158
ТЕМА 14. Кластерний аналіз 165
ТЕМА 15. Алгоритми пошуку асоціативних правил 175
ТЕМА 16. Генетичні алгоритми як інструмент data mining 183
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 189

Змістовний модуль 1. Класичне машинне навчання

ТЕМА 1. Статистичний аналіз. R. Використання базових типів даних та конструкцій

Машинне навчання, разом з Data mining, є складовою штучного інтелекту.

В свою чергу, до машинного навчання відноситься багато складових.



Одним із прикладів машинного навчання є комп'ютерний зір. Це потужна технологія, що дозволяє розпізнавати зображення та робити висновки на основі отриманих даних. Машинний зір використовується в системі Google об'єктив, що дозволяє робити/завантажувати зображення та шукати певні його елементи за допомогою реверсивного пошуку. Але це дуже складна система, що постійно розвивається впродовж тривалого часу. Якщо спростити, то машинний зір можна представити у вигляді застосунку, що розпізнає зображення, а потім поділяє їх на дві категорії.

Для прикладу уявімо програму, що приймає на вході зображення, а на виході дає відповідь: чи морква це, чи ні. Задача дуже проста і не має місця в реальному застосуванні, але дозволяє зрозуміти принцип роботи машинного зору. Так само ми можемо представити іншу задачу: розпізнавати номерні знаки на авто, поки не знайдеться заданий номерний знак. Така задача дуже часто використовується у СППР, якими користується поліція, оскільки зменшує навантаження на людину і дає змогу повністю (залежно від умов до 97%) автоматизувати процес виявлення вкрадених авто за допомогою дорожніх камер.

Але повернемося до моркви. Якщо така прогрма існує, то вона має працювати, надаючи три можливих варіанта відповіді: морква, НЕ морква, не вдалось розпізнати. Останній варіант, це той, через який неможливо повністю автоматизувати процес розпізнавання, оскільки існує велика кількість

візуальних перешкод, на які людина не може сильно вплинути під час розробки програми.

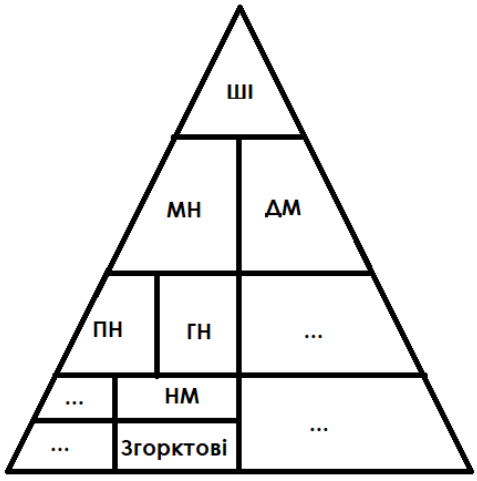
Інші два варіанти повністю зрозумілі: даємо зображення моркви – має відповісти «морква», даємо зображення картоплі – має відповісти «НЕ морква». Здавалося б, дуже проста задача, але навіть вона має свої труднощі.

Поглянемо на складові такої програми в розрізі усього життєвого циклу. Знадобиться, власне, програма, що прийматиме зображення та моделі; модель, що натренована на певному наборі даних; сам набір даних, що має бути розділений на дві (іноді три) частини; зображення для роботи програми. З зображеннями вже розібралися, програма є дуже простою, оскільки, по суті, це просто робота з файлами та трохи алгоритму зв'язку моделі і вхідних даних. Тепер про модель. Модель – набір ваг (якщо ми говоримо про реалізацію за допомогою згорткових нейронних мереж), що були отримані в результаті тренування на наборі даних.

Принципи створення та роботи моделей розглянемо в наступних заняттях. На зараз лишаємо набір даних. Що ж тут такого складного, зробити купу фотографій моркви з різних ракурсів, під різними кутами і піде? Ні, насправді все набагато важче. Перш за все, дані мають бути одного типу. Тобто, ми говоримо не про формат (.png .jpg і т.д.), хоча це також може впливати на роботу, а про орієнтацію пристрою під час зйомки, який відсоток зображення займає морква, наявність кольору на зображенні і т.д. Велика кількість факторів, які впливатимуть на якість навчання моделі і все це ще на етапі підготовки даних.

Уявімо, що все буде робитися однотипно, без великих помилок під час зйомки. Тепер маємо подбати також про розмірність зображення. Що це таке і на що впливає. Скажімо, є зображення 100 на 100 пікселів. Для комп'ютера це 10000 значень, а зображень потрібно багато. Тож перш за все постає питання пам'яті. Навчання моделі також витрачає ресурс (або CPU, або GPU), залежно від технології та задачі. Більш докладно про це буде описано в наступних темах, але на зараз залишимо оті 10000 значень для одного запису.

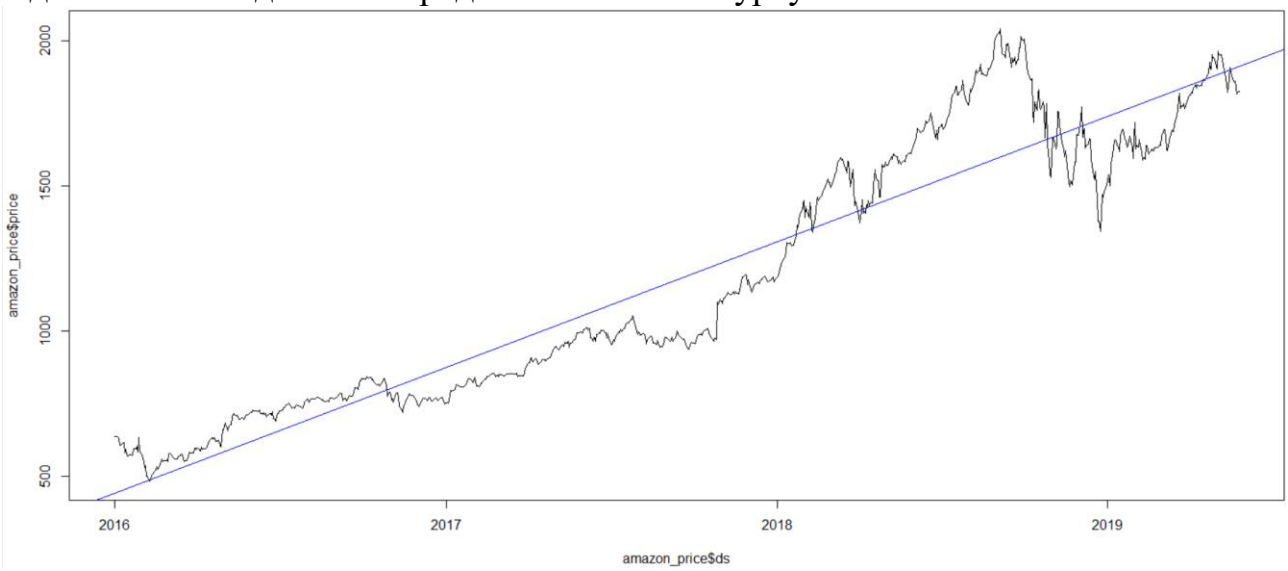
Якщо поглянути на описане вище послідовно, то можна отримати таку асоціацію – трикутник.



Спочатку ми розділили штучний інтелект на складові, потім машинне навчання на поверхневе та глибинне. Далі пішли по нейронним мережам одразу

до згорткових (а їх набагато більше), розклали роботу з таким типом мереж на складові і тепер розглядаємо набори даних. Таким чином ми шукаємо початкову точку, з якої починається штучний інтелект в цілому. Але шукати не треба, адже відповідь вже давно є – це дані. Звідси робимо простий висновок – обробка даних є однім з ключових моментів, адже навіть бедоганна модель, при правильному підборі способу розв'язання задачі не дасть нормального результату без обробленого набору даних.

10000 значень для одного запису – важко для розуміння і опрацювання людині, тому розглянемо інший приклад використання машинного навчання. Для цього звернемося до часових рядів. Часові ряди в найпростішому своєму вигляді можуть бути представлені двома змінними з безліччю записів. Дуже відомий всім вид часових рядів – коливання курсу валют.



Такий графік (майже такий, тут заповнені пропуски) будується зі звичайної таблиці.

	ds	price
57	2086-02-27	555.23
58	2086-02-28	NA
59	2086-03-01	NA
60	2086-03-02	552.52
61	2086-03-03	579.04
62	2086-03-04	580.21
63	2086-03-05	577.49
64	2086-03-06	575.14

З таблиці видно, що представлена вона двома змінними: час та вартість і має безліч записів.

Але до чого тут машинне навчання? Річ у тім, що складовою машинного навчання є прогнозування за допомогою прогнозних моделей, а курс валют, що представлений часовим рядом як раз підходить під задачу прогнозування.

Задача прогнозування не обмежується простим курсом валют. Є і інші, більш складні приклади. Розглянемо прогнозування температури. Ну і чим же

температура від валют відрізняється? Температура може бути не тільки навколишнього середовища, а й атомного ректору. Якщо вбудувати прогноз в СППР, ми, потенційно, отримуємо ситуацію, в якій температура реактора все ще в нормі, але вже отримуємо повідомлення про помилку.

Як це відбувається і для чого це? Під час певних процесів температура може підвищуватися, або знижуватися. Все логічно. Але, якщо ми говоримо про довготривалі процеси, де кожна дія має накопичувальний характер. Отримуємо ситуацію, коли незначна помилка під час якогось процесу теоретично може призвести до різкого підвищення температури всередині реактора. Через те, що прогнозування в цьому прикладі є частиною СППР, ми можемо сподіватися, що автоматичне керування попередить про можливу небезпеку завчасно, щоб в оператора був час на реакцію. Тобто в результаті ми отримуємо час.

І цей час може бути витрачений на діагностику обладнання, або на припинення процесу, або на внесення корективів у сам процес і т.д. Або, за рахунок прогнозованості процесу, оператор може приймати рішення про додаткові засоби безпеки. Час є важливим ресурсом.

Зв'язавши все логічним чином переходимо до наступних питань: чого в даних є пропуски і як обробляти їх та інші проблемні моменти в даних? Тут на допомогу прийдуть методи статистики, адже часовий ряд як раз і є статистичними даними. Статистичних методів існує безліч, велика кількість з яких вивчається в рамках інших дисциплін, або навіть у школі. Простим прикладом є середнє значення вибірки.

Враховуючи той факт, що даних потрібно багато, а збираються вони людьми – згадуємо про людський фактор, тобто помилки. Статистичний аналіз дозволяє виявляти, коригувати їх, та використовувати різні методи, що дозволяють розуміти структуру вибірки. Таким чином, спускаючись по вищезгаданому трикутнику, ми доходимо до його основи – даних та методів статистичного аналізу, що дозволяють працювати з великими наборами даних одночасно. Існує велика кількість автоматизованих прикладних програм, що використовуються для статистичного аналізу. Тепер розглянемо поверхнево сам аналіз та одну з програм, що буде корисною як для нього, так і для машинного навчання.

Статистичний аналіз

Статистичний аналіз – це процес збору та аналізу даних з метою виявлення закономірностей і тенденцій. Це компонент аналітики даних. Це метод для усунення упередженості в оцінюванні даних за допомогою чисельного аналізу. Ця техніка корисна для збору інтерпретацій досліджень, розробки статистичних моделей і планування опитувань і досліджень. Статистичний аналіз можна використовувати в таких ситуаціях, як збір інтерпретацій досліджень, статистичне моделювання або планування опитувань і досліджень. Це також може бути корисним для організацій бізнес-аналітики, яким доводиться працювати з великими обсягами даних.

Статистичний аналіз – це науковий інструмент, який допомагає збирати та аналізувати великі обсяги даних, щоб визначити загальні закономірності,

тенденції та перетворити їх на значущу інформацію. Простими словами, статистичний аналіз – це інструмент аналізу даних, який допомагає зробити важливі висновки з необроблених і неструктурованих даних.

Висновки зроблені за допомогою статистичного аналізу, який полегшує прийняття рішень і допомагає підприємствам робити прогнози на майбутнє на основі минулих тенденцій. Його можна визначити як науку про збір і аналіз даних для виявлення тенденцій і закономірностей і їх представлення.

Статистичний аналіз передбачає роботу з числами та використовується підприємствами та іншими установами для використання даних для отримання суттєвої інформації.

Тип	**Опис**
Описовий аналіз	Описовий статистичний аналіз передбачає збір, інтерпретацію, аналіз та узагальнення даних для представлення їх у формі діаграм, графіків і таблиць. Замість того, щоб робити висновки, він просто робить складні дані легкими для читання та розуміння.
Інференційний аналіз	Інференційний статистичний аналіз спрямований на отримання значущих висновків на основі проаналізованих даних. Він вивчає зв'язок між різними змінними або робить прогнози для всієї сукупності.
Прогнозний аналіз	Прогнозний статистичний аналіз – це тип статистичного аналізу, який аналізує дані для визначення минулих тенденцій і прогнозування майбутніх подій на їх основі. Він використовує алгоритми машинного навчання, аналіз даних, моделювання даних і штучний інтелект для проведення статистичного аналізу даних.
Наказовий аналіз	Наказовий аналіз проводить аналіз даних і призначає найкращий курс дій на основі результатів. Це тип статистичного аналізу, який допомагає прийняти обґрунтоване рішення.
Дослідницький аналіз даних	Дослідницький аналіз схожий на інференційний аналіз, але відмінність полягає в тому, що він включає дослідження невідомих асоціацій даних. Він аналізує потенційні зв'язки в даних.
Причинно-наслідковий аналіз	Причинно-наслідковий статистичний аналіз зосереджується на визначенні причинно-наслідкового зв'язку між різними змінними в необроблених даних. Простими словами, він визначає, чому щось відбувається, і його вплив на інші змінні. Ця методологія може бути використана компаніями для визначення причини невдачі.

Статистичний аналіз можна назвати благом для людства, і він має багато переваг як для окремих осіб, так і для організацій. Нижче наведено деякі з причин, чому вам варто розглянути можливість інвестування в статистичний аналіз:

а) Це може допомогти вам визначити місячні, квартальні, річні показники прибутку від продажів і витрат, що полегшує прийняття рішень;

б) Це може допомогти вам прийняти обґрунтовані та правильні рішення;

в) Це може допомогти вам визначити проблему або причину збою та внести виправлення. Наприклад, він може визначити причину збільшення загальних витрат і допомогти вам скоротити марнотратні витрати;

г) Це може допомогти вам провести аналіз ринку та скласти ефективну стратегію маркетингу та продажів;

ґ) Це допомагає підвищити ефективність різних процесів.

R

Для вирішення поставленої задачі було обрано середовище розробки RGui. RGui базується на R, де R – мова програмування і програмне середовище для статистичних обчислень, аналізу та зображення даних в графічному вигляді. Розробка R відбувалась під істотним впливом двох наявних мов програмування: мови програмування S з семантикою успадкованою від Scheme. R названа за першою літерою імен її засновників Роса Іхаки (Ross Ihaka) та Роберта Джентлмена (Robert Gentleman) працівників Оклендського Університету в Новій Зеландії.

Якщо коротко, то:

а) R – це «мова та середовище для статистичних обчислень і графіки»; ви можете думати про це як про комбінацію пакету статистики та мови програмування;

б) R повністю безкоштовне; вам не потрібно платити за це, і ви можете вносити в нього будь-які зміни;

в) R працює на Windows, MacOS, Linux і багатьох варіантах Unix;

г) R не підтримується жодним комерційним підприємством, але він має дуже активну спільноту розробників. Посібники повні, і про це середовище існує багато підручників;

ґ) У R є величезна кількість вбудованих стандартних і найсучасніших статистичних функцій, широкий вибір (безкоштовних) додаткових пакетів, які збільшують функціонал, і ви можете ще розширити їх. Кожен стандартний статистичний аналіз можна виконати в R;

д) R здебільшого керується командним рядком (хоча були розроблені різні графічні інтерфейси); це ускладнює використання, але забезпечує гнучкість, документування та повторення аналізів.

R має значні можливості для здійснення статистичних аналізів, включаючи лінійну і нелінійну регресію, класичні статистичні тести, аналіз часових рядів (серій), кластерний аналіз і багато іншого.

R легко розбудовується завдяки використанню додаткових функцій і пакетів доступних на сайті Comprehensive R Archive Network.

Багато з вищевказаних особливостей середовища розробки/мови програмування можна приписати й іншим представникам тієї чи іншої сфери, тому необхідним є вказати саме ті переваги, які відрізняють R від усіх своїх конкурентів. Перш за все – R це мова створена саме для статистичного аналізу, тому в ній вже спрощено роботу з усіма видами числової інформації.

Наприклад, в ній прибрано необхідність опрацьовувати базові завдання з масивами через цикли, замість цього є можливість безпосередньої роботи як із стовбцями, так і з рядками даних.

По-друге, R – сучасна мова та середовище з великою спільнотою. В історії вже зустрічалися мови, що спеціалізуються саме на математичній та статистичній обробці даних, але вони або застарілі, або не мають можливості використовувати сучасні засоби обробки типів наборів даних.

По-третє, в R реалізована потужна графічна база, що дозволяє ефективно візуалізувати дані та їх окремі компоненти. Якщо потужності замало, то на допомогу приходять безкоштовні бібліотеки, що на додачу до потужності надають і зручність візуалізації, за необхідності.

Особливості синтаксису в R

Звичайні типи даних (рядок, символ, числа чи операції над ними) можна виводити одразу в консоль без створення змінних. Рядкові змінні мають бути загорнуті в лапки.

Функція print() також існує, але її частіше використовують у функціях, що загорнуті в {}. Без print() виводитись число не буде.

```
> u
Error: object 'u' not found
> "u"
[1] "u"
> 5
[1] 5
> 5 + 5
[1] 10
```

```
> for (x in 1:10) {
+   print(x)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Коментарі виводяться за допомогою решітки # перед кожним рядком коментаря (нема багаторядкового коментування). У R немає команди для оголошення змінної. Змінна створюється, коли ви вперше присвоюєте їй значення. Щоб присвоїти значення змінній, використовуйте знак <-. Щоб вивести (або надрукувати) значення змінної, просто введіть її назву.

```
> name <- "John"
> age <- 40
> name
[1] "John"
> age
[1] 40
```

Ви також можете об'єднати два або більше елементів за допомогою функції paste(). Для об'єднання тексту і змінної у R використовується кома (,).

```
> text <- "awesome"
>
> paste("R is", text)
[1] "R is awesome"
> text1 <- "R is"
> text2 <- "awesome"
>
> paste(text1, text2)
[1] "R is awesome"
> num1 <- 5
> num2 <- 10
>
> num1 + num2
[1] 15
```

Змінна може мати коротку назву (наприклад, x та y) або більш описову назву (age, carname, total_volume). Правила для змінних R такі:

- Ім'я змінної повинно починатися з літери і може бути комбінацією літер, цифр, крапки(.)
- та підкреслення(_). Якщо ім'я починається з крапки(.), після неї не може йти цифра.
- Ім'я змінної не може починатися з цифри або символу підкреслення (_)
- Назви змінних чутливі до регістру (вік, вік і AGE - це три різні змінні)
- Зарезервовані слова не можна використовувати як змінні (TRUE, FALSE, NULL, if...).

У програмуванні тип даних є важливим поняттям.

Змінні можуть зберігати дані різних типів, а різні типи можуть виконувати різні функції.

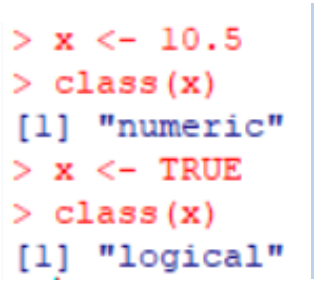
У R змінні не обов'язково оголошувати з певним типом, вони навіть можуть змінювати тип після того, як їх було задано.

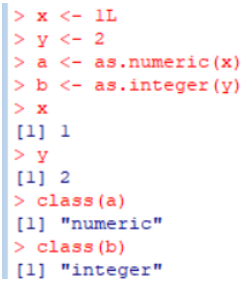
Основні типи даних у R можна розділити на наступні типи:

- числові - (10.5, 55, 787)
- цілі - (1L, 55L, 100L, де літера "L" позначає, що це ціле число)
- комплексний - (9 + 3i, де "i" - уявна частина)
- символьний (так званий рядок) - ("k", "R is exciting", "FALSE", "11.5")
- логічний (так званий булевий) - (TRUE або FALSE)

Ми можемо використовувати функцію class() для перевірки типу даних змінної. Ви можете перетворити один тип в інший за допомогою наступних функцій:

as.numeric()
as.integer()
as.complex()





R має багато вбудованих математичних функцій, які дозволяють виконувати математичні задачі над числами.

Наприклад, функції min() і max() можна використовувати для знаходження найменшого або найбільшого числа у множині.

Функція sqrt() повертає квадратний корінь з числа.

Функція abs() повертає абсолютне (додатне) значення числа.

Функція ceil() округлює число в більшу сторону до найближчого цілого, а функція floor() округлює число в меншу сторону до найближчого цілого і повертає результат.

Ви можете присвоїти багаторядковий рядок змінній. Однак зауважте, що R додаватиме "\n" у кінці кожного розриву рядка. Це називається символ переходу на новий рядок, а символ n вказує на новий рядок.

Якщо ви хочете, щоб розриви рядків було вставлено у тій самій позиції, що і у коді, скористайтеся функцією cat().

У мові R існує багато корисних функцій для роботи з рядками.

Наприклад, щоб знайти кількість символів у рядку, скористайтеся функцією nchar().

Для перевірки наявності символу або послідовності символів у рядку використовуйте функцію grepl().

Оператор	**Опис**	**Приклад**
:	Створює ряд чисел у послідовності	x <- 1:10
%in%	З'ясувати, чи належить елемент вектору	x %in% y
%*%	Матричне множення	x <- Matrix1 %*% Matrix2

Оператор if

Оператор if записується з ключовим словом if і використовується для вказівки блоку коду, який виконується, якщо умова має значення TRUE.

Ключове слово else if - це спосіб R сказати "якщо попередні умови не були істинними, то спробувати цю умову". Ключове слово else перехоплює все, що не перехоплено попередніми умовами.

```
> a <- 200
> b <- 33
>
> if (b > a) {
+   print("b is greater than a")
+ } else if (a == b) {
+   print("a and b are equal")
+ } else {
+   print("a is greater than b")
+ }
[1] "a is greater than b"
```

Цикли While

За допомогою оператора break ми можемо зупинити цикл, навіть якщо умова while має значення TRUE

```
> i <- 1
> while (i < 6) {
+   print(i)
+   i <- i + 1
+   if (i == 4) {
+     break
+   }
+ }
[1] 1
[1] 2
[1] 3
```

За допомогою оператора next ми можемо пропустити ітерацію, не завершуючи цикл.

Цикли For

Цикл for використовується для перебору послідовності.

Це менше схоже на ключове слово for в інших мовах програмування, і працює більше як метод ітератора, як в інших об'єктно-орієнтованих мовах програмування.

За допомогою циклу for ми можемо виконати набір операторів, по одному разу для кожного елемента у векторі, масиві, списку тощо.

```
> fruits <- list("apple", "banana", "cherry")
>
> for (x in fruits) {
+   print(x)
+ }
[1] "apple"
[1] "banana"
[1] "cherry"
```

R Функції

Щоб створити функцію, використовуйте ключове слово function().Щоб викликати функцію, використовуйте ім'я функції з круглими дужками, наприклад, my_function():

```
> my_function <- function() {
+ print("Hello World!")
+ }
> my_function()
[1] "Hello World!"
```

У функції можна передавати інформацію як аргументи.

Аргументи вказуються після імені функції в круглих дужках. Ви можете додавати скільки завгодно аргументів, просто розділяйте їх комою.

За замовчуванням функція повинна викликатися з правильною кількістю аргументів. Це означає, що якщо ваша функція очікує 2 аргументи, ви повинні викликати функцію з 2 аргументами, не більше і не менше:

Значення параметра за замовчуванням

У наступному прикладі показано, як використовувати значення параметра за замовчуванням.

Якщо ми викликаємо функцію без аргументу, вона використовує значення за замовчуванням:

```
> my_function <- function(country = "Norway") {
+   paste("I am from", country)
+ }
>
> my_function("Sweden")
[1] "I am from Sweden"
> my_function()
[1] "I am from Norway"
```

Щоб дозволити функції повертати результат, використовуйте функцію return().

```
> my_function <- function(x) {
+   return (5 * x)
+ }
>
> print(my_function(3))
[1] 15
```

Зазвичай, коли ви створюєте змінну всередині функції, ця змінна є локальною і може бути використана тільки всередині цієї функції.

Щоб створити глобальну змінну всередині функції, можна скористатися оператором глобального присвоювання <<-.

Також використовуйте оператор глобального присвоювання, якщо ви хочете змінити глобальну змінну всередині функції.

ТЕМА 2. Обробка наборів даних. Введення в статистичний аналіз в R

Структури даних

Вектор - це просто список елементів одного типу.

Щоб об'єднати список елементів у вектор, використовуйте функцію c() і розділяйте елементи комою.

У прикладі нижче ми створюємо векторну змінну з назвою fruits, яка об'єднує рядки:

```
> fruits <- c("banana", "apple", "orange")
> fruits
[1] "banana" "apple"  "orange"
```

У цьому прикладі ми створюємо вектор, який поєднує числові значення:

```
> numbers <- c(1, 2, 3)
>
> numbers
[1] 1 2 3
```

Щоб створити вектор з числовими значеннями в послідовності, використовуйте оператор ":"

```
> numbers <- 1:10
>
> numbers
 [1]  1  2  3  4  5  6  7  8  9 10
```

Щоб дізнатися, скільки елементів має вектор, використовуйте функцію length().

Щоб відсортувати елементи вектора за алфавітом або числом, використовуйте функцію sort().

Ви можете отримати доступ до елементів вектора, звертаючись до його індексу в дужках []. Перший елемент має індекс 1, другий елемент має індекс 2 і так далі.

Ви також можете отримати доступ до декількох елементів, посилаючись на різні позиції індексів за допомогою функції c():

```
> fruits <- c("banana", "apple", "orange", "mango", "lemon")
> fruits[c(1, 3)]
[1] "banana" "orange"
```

Ви також можете використовувати від'ємні індекси для доступу до всіх елементів, окрім вказаних. Щоб змінити значення конкретної позиції, зверніться до її індексного номера:

```
> fruits[1] <- "pear"
> fruits
[1] "pear"   "apple"  "orange" "mango"  "lemon"
```

Щоб повторити вектори, використовуйте функцію rep():

```
> repeat_indepent <- rep(c(1,2,3), times = c(5,2,1))
> repeat_indepent
[1] 1 1 1 1 1 2 2 3
> repeat_each <- rep(c(1,2,3), each = 3)
> repeat_each
[1] 1 1 1 2 2 2 3 3 3
```

Щоб зробити більші або менші кроки в послідовності, використовуйте функцію seq(). Функція seq() має три параметри: від - місце, де починається послідовність, до - місце, де закінчується послідовність, і по - інтервал послідовності.

Список у R може містити багато різних типів даних. Список – це колекція даних, яка є впорядкованою та змінною.

Щоб створити список, використовуйте функцію list():

```
> thislist <- list("apple", "banana", "cherry")
> thislist
[[1]]
[1] "apple"

[[2]]
[1] "banana"

[[3]]
[1] "cherry"
```

Ви можете отримати доступ до елементів списку, посилаючись на його індекс, вказаний у дужках. Перший пункт має індекс 1, другий - 2 і так далі. Щоб змінити значення певного елемента, зверніться до його індексу. Щоб дізнатися, скільки елементів містить список, використовуйте функцію length().

Щоб дізнатися, чи присутній заданий елемент у списку, використовуйте оператор %in%. Щоб додати елемент в кінець списку, використовуйте функцію append(). Щоб додати елемент праворуч від заданого індексу, додайте у функції append() вираз "after=номер індексу":

```
> thislist <- list("apple", "banana", "cherry")
>
> append(thislist, "orange", after = 2)
[[1]]
[1] "apple"

[[2]]
[1] "banana"

[[3]]
[1] "orange"

[[4]]
[1] "cherry"
```

Ви також можете видаляти елементи списку. У наступному прикладі створено новий, оновлений список без елемента "яблуко":

```
> thislist <- list("apple", "banana", "cherry")
>
> newlist <- thislist[-1]
> newlist
[[1]]
[1] "banana"

[[2]]
[1] "cherry"
```

Ви можете вказати діапазон індексів, вказавши, де починається і де закінчується діапазон, за допомогою оператора “:”

```
> thislist <- list("apple", "banana", "cherry", "orange", "kiwi", "melon", "man$
>
> (thislist)[2:5]
[[1]]
[1] "banana"

[[2]]
[1] "cherry"

[[3]]
[1] "orange"

[[4]]
[1] "kiwi"
```

Існує декілька способів об'єднання двох або більше списків у R.

Найпоширенішим способом є використання функції c(), яка об'єднує два елементи разом:

```
> list1 <- list("a", "b", "c")
> list2 <- list(1,2,3)
> list3 <- c(list1,list2)
```

Матриця – це двовимірний набір даних зі стовпчиками та рядками.

Стовпчик - це вертикальне представлення даних, а рядок - горизонтальне представлення даних.

Матрицю можна створити за допомогою функції matrix(). Вкажіть параметри nrow і ncol, щоб отримати кількість рядків і стовпців:

```
> thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)
> thismatrix
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Ви також можете створити матрицю з рядків. Ви можете отримати доступ до елементів за допомогою дужок [ ]. Перша цифра "1" у дужці вказує на позицію рядка, а друга цифра "2" - на позицію стовпця. Доступ до всього рядка можна отримати, якщо вказати кому після числа в дужці. Доступ до всього стовпця можна отримати, якщо вказати кому перед числом у дужці. Доступ до декількох рядків можна отримати за допомогою функції c():

```
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fig"), nrow = 3, ncol = 3)
>
> thismatrix[c(1,2),]
     [,1]     [,2]     [,3]
[1,] "apple"  "orange" "pear"
[2,] "banana" "grape"  "melon"
```

Можна отримати доступ до більш ніж одного стовпця аналогічно. Використовуйте функцію cbind() для додавання додаткових стовпців (функція rbind() для додавання додаткових рядків) до матриці:

```
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fig"), nrow = 3, ncol = 3)
> newmatrix <- rbind(thismatrix, c("strawberry", "blueberry", "raspberry"))
> newmatrix
     [,1]         [,2]         [,3]
[1,] "apple"      "orange"     "pear"
[2,] "banana"     "grape"      "melon"
[3,] "cherry"     "pineapple" "fig"
[4,] "strawberry" "blueberry" "raspberry"
```

Використовуйте функцію c() для видалення рядків і стовпців у матриці:

```
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange", "mango", "pineapple"), nrow = 3, ncol =2)
> thismatrix <- thismatrix[-c(1), -c(1)]
> thismatrix
[1] "mango"     "pineapple"
```

Щоб дізнатися, чи присутній заданий елемент у матриці, використовуйте оператор %in%. Використовуйте функцію dim(), щоб знайти кількість рядків і стовпців у матриці. Для знаходження розмірності матриці використовується

функція length(). Загальна кількість комірок у матриці дорівнює кількості рядків, помноженій на кількість стовпців. Ви можете циклічно переглядати матрицю за допомогою циклу for. Цикл почнеться з першого рядка, рухаючись праворуч:

```
> thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
>
> for (rows in 1:nrow(thismatrix)) {
+   for (columns in 1:ncol(thismatrix)) {
+     print(thismatrix[rows, columns])
+   }
+ }
[1] "apple"
[1] "cherry"
[1] "banana"
[1] "orange"
```

Знову ж таки, ви можете використовувати функцію rbind() або cbind() для об'єднання двох або більше матриць разом. На відміну від матриць, масиви можуть мати більше двох вимірів.

Ми можемо використовувати функцію array() для створення масиву, а параметр dim - для вказівки розмірів.

```
> thisarray <- c(1:24)
> thisarray
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
> multiarray <- array(thisarray, dim = c(4, 3, 2))
> multiarray
, , 1

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

Ви можете отримати доступ до елементів масиву, посилаючись на індексну позицію. Ви можете використовувати дужки [] для доступу до потрібних елементів масиву. Синтаксис наступний: array[позиція рядка, позиція стовпця, рівень матриці]. Ви також можете отримати доступ до цілого рядка або стовпця з матриці у масиві за допомогою функції c():

```
> multiarray[c(1),,1]
[1] 1 5 9
```

Фрейм даних - це дані, що відображаються у форматі таблиці.

Таблиці даних можуть містити різні типи даних. Якщо перший стовпець може бути символьним, то другий і третій можуть бути числовими або логічними. Однак кожен стовпець повинен мати один і той самий тип даних.

Для створення фрейму даних використовуйте функцію data.frame():

```
> Data_Frame <- data.frame (
+   Training = c("Strength", "Stamina", "Other"),
+   Pulse = c(100, 150, 120),
+   Duration = c(60, 30, 45)
+ )
> Data_Frame
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
```

Використовуйте функцію summary() для підсумовування даних з фрейму даних.

Ми можемо використовувати одинарні дужки [ ], подвійні дужки [[ ]] або $ для доступу до стовпців з фрейму даних:

```
> summary(Data_Frame)
   Training             Pulse          Duration
 Length:3           Min.   :100.0   Min.   :30.0
 Class :character   1st Qu.:110.0   1st Qu.:37.5
 Mode  :character   Median :120.0   Median :45.0
                    Mean   :123.3   Mean   :45.0
                    3rd Qu.:135.0   3rd Qu.:52.5
                    Max.   :150.0   Max.   :60.0
```

```
> Data_Frame[1]
  Training
1 Strength
2  Stamina
3    Other
>
> Data_Frame[["Training"]]
[1] "Strength" "Stamina"  "Other"
>
> Data_Frame$Training
[1] "Strength" "Stamina"  "Other"
```

Використовуйте функцію rbind() для додавання нових рядків до фрейму даних. Використовуйте функцію cbind() для додавання нових стовпців до фрейму даних. Використовуйте функцію c() для видалення рядків і стовпців у фреймі даних. Використовуйте функцію dim() для знаходження кількості рядків і стовпців у фреймі даних. Ви також можете використовувати функцію ncol() для знаходження кількості стовпців і nrow() для знаходження кількості рядків. Використовуйте функцію length(), щоб знайти кількість стовпців у фреймі даних (подібно до ncol()). Використовуйте функцію rbind(), щоб об'єднати два або більше фреймів даних у R по вертикалі. А функція cbind() - для об'єднання двох або більше фреймів даних у R по горизонталі.

Фактори використовуються для категоризації даних. Прикладами факторів є

Демографія: Чоловіки/жінки

Музика: Рок, поп, класика, джаз

Тренування: Сила, витривалість

Щоб створити фактор, використовуйте функцію factor() і додайте вектор як аргумент:

```
> music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))
> music_genre
[1] Jazz    Rock    Classic Classic Pop     Jazz    Rock    Jazz
Levels: Classic Jazz Pop Rock
```

З наведеного вище прикладу видно, що фактор має чотири рівні (категорії): Класика, Джаз, Поп та Рок. Щоб вивести лише рівні, скористайтеся функцією levels(). Ви також можете задати рівні, додавши аргумент levels у функцію factor(). Використовуйте функцію length(), щоб дізнатися, скільки елементів у факторі. Щоб отримати доступ до елементів у факторі, зверніться до індексного номера, використовуючи дужки []. Щоб змінити значення

конкретного елемента, зверніться до його індексного номера. Зверніть увагу, що ви не можете змінити значення певного елемента, якщо він ще не вказаний у факторі. Однак, якщо ви вже вказали його всередині аргументу levels, це спрацює.

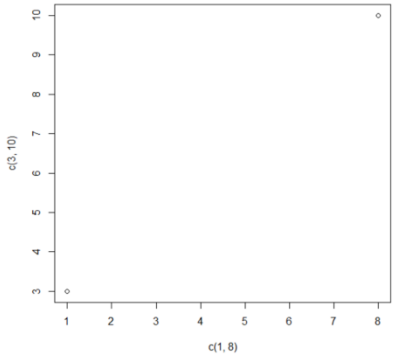
Графіки

Функція plot() використовується для малювання точок (маркерів) на діаграмі. Функція отримує параметри для задання точок на діаграмі. Параметр 1 задає точки на осі x. Параметр 2 задає точки на осі y.

У найпростішому випадку ви можете використовувати функцію plot() для побудови графіка залежності двох чисел одне від одного. Побудуйте по одній точці на діаграмі в позиції (1) і позиції (3): plot(1, 3).

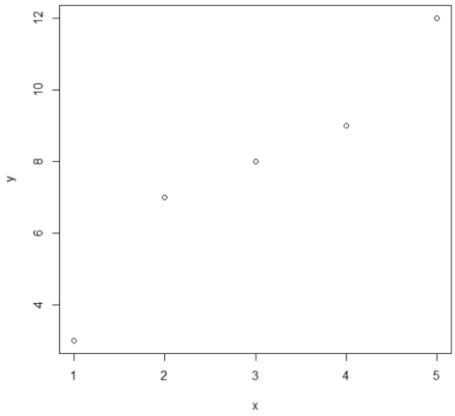
Щоб намалювати більше точок, використовуйте вектори:

```
> plot(c(1, 8), c(3, 10))
```



Ви можете побудувати скільки завгодно точок, просто переконайтеся, що у вас є однакова кількість точок на обох осях. Для кращої організації, коли у вас багато значень, краще використовувати змінні:

```
> x <- c(1, 2, 3, 4, 5)
> y <- c(3, 7, 8, 9, 12)
>
> plot(x, y)
```

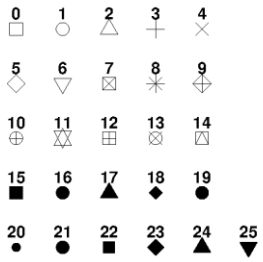


Якщо ви хочете намалювати точки послідовно, як на осі x, так і на осі y, використовуйте оператор :. Функція plot() також приймає параметр типу зі значенням l, щоб намалювати лінію, яка з'єднає всі точки на діаграмі. Функція plot() також приймає інші параметри, такі як main, xlab і ylab, якщо ви хочете налаштувати діаграму з головним заголовком і різними підписами для осей x і y. Існує багато інших параметрів, за допомогою яких можна змінити вигляд точок.

Використовуйте col="color", щоб додати колір до точок.

Використовуйте cex=number для зміни розміру точок (1 - за замовчуванням, 0.5 означає на 50% менше, а 2 - на 100% більше).

Використовуйте pch зі значенням від 0 до 25 для зміни формату форми точки. Значення параметра pch коливається від 0 до 25, що означає, що ми можемо вибрати до 26 різних типів форм точок:



Лінійна діаграма має лінію, яка з'єднує всі точки діаграми.

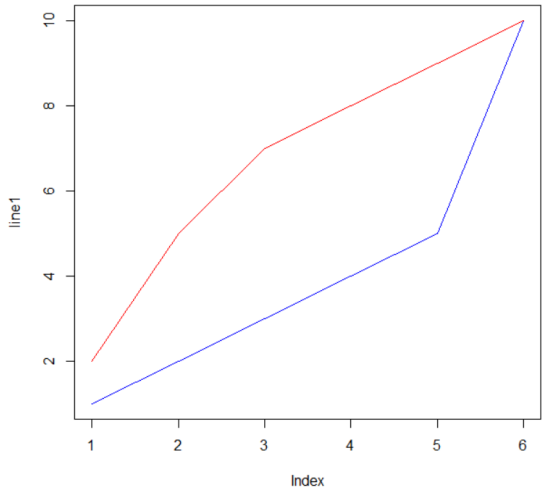
Щоб створити лінію, скористайтеся функцією plot() і додайте параметр type зі значенням "l". Колір лінії за замовчуванням чорний. Щоб змінити колір, використовуйте параметр col. о Щоб змінити ширину лінії, використовуйте параметр lwd (1 - за замовчуванням, 0.5 означає на 50% менше, а 2 - на 100% більше). За замовчуванням лінія суцільна. Використовуйте параметр lty зі значенням від 0 до 6, щоб вказати формат рядка.

Наприклад, lty=3 відобразить пунктирну лінію замість суцільної. Доступні значення параметра lty:

- 0 вилучає лінію;
- 1 відображає суцільну лінію;
- 2 відображає пунктирну лінію;
- 3 відображає пунктирну лінію;
- 4 виводить "пунктирну" лінію;
- 5 відображає "довгу пунктирну" лінію;
- 6 відображає "дві пунктирні" лінії.

Щоб відобразити більше однієї лінії на графіку, використовуйте функцію plot() разом з функцією lines():

```
> line1 <- c(1,2,3,4,5,10)
> line2 <- c(2,5,7,8,9,10)
>
> plot(line1, type = "l", col = "blue")
> lines(line2, type="l", col = "red")
```

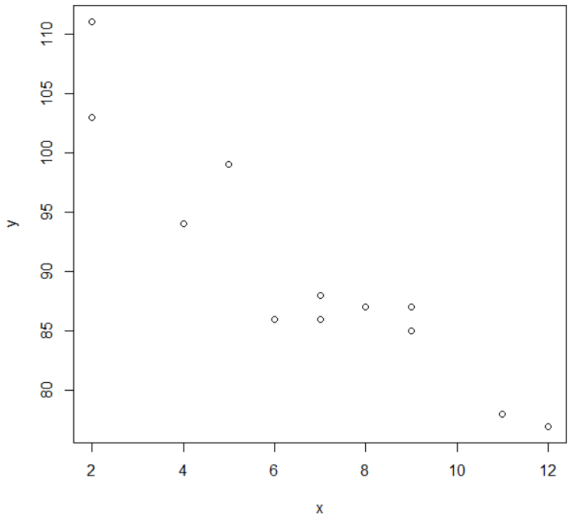


З розділу "Графік" ви дізналися, що функція plot() використовується для побудови графіків чисел відносно один одного.

"Точкова діаграма (діаграма розсіювання)" - це тип діаграми, який використовується для відображення зв'язку між двома числовими змінними і відображає по одній точці для кожного спостереження.

Вона потребує двох векторів однакової довжини, один для осі x (горизонтальний) і один для осі y (вертикальний):

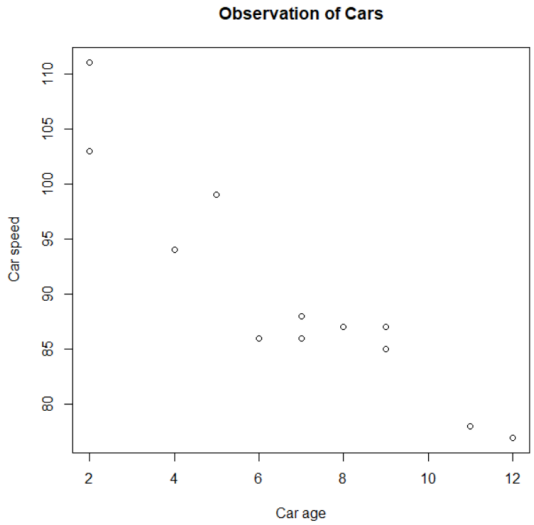
```
> x <- c(5,7,8,7,2,2,9,4,11,12,9,6)
> y <- c(99,86,87,88,111,103,87,94,78,77,85,86)
>
> plot(x, y)
```



Спостереження у наведеному вище прикладі повинно показати результат 12 автомобілів, що проїжджають повз.

Це може бути незрозуміло для тих, хто бачить графік вперше, тому додамо заголовок і різні підписи, щоб краще описати діаграму розсіювання:

```
> x <- c(5,7,8,7,2,2,9,4,11,12,9,6)
> y <- c(99,86,87,88,111,103,87,94,78,77,85,86)
>
> plot(x, y, main="Observation of Cars", xlab="Car age", ylab="Car speed")
```



Нагадаємо, що спостереження у прикладі вище є результатом спостереження за 12 автомобілями, що проїжджають повз.

Вісь x показує вік автомобіля. Вісь y показує швидкість автомобіля, коли він проїжджав повз.

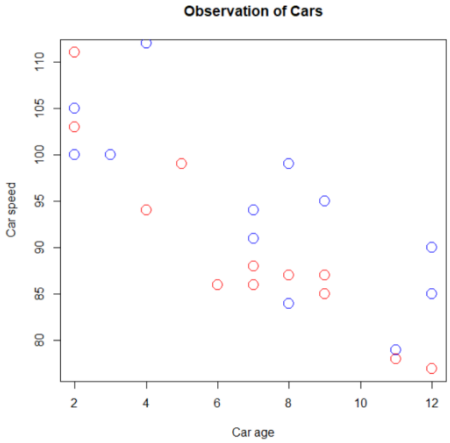
Чи існує взаємозв'язок між цими спостереженнями?

Здається, що чим новіший автомобіль, тим швидше він їздить, але це може бути збігом, адже ми зареєстрували лише 12 автомобілів.

У наведеному вище прикладі, здається, існує зв'язок між швидкістю автомобіля та віком, але що, якщо ми побудуємо графік спостережень за інший день? Чи покаже нам діаграма розсіювання щось інше?

Щоб порівняти графік з іншим графіком, скористайтеся функцією points():

```
> x1 <- c(5,7,8,7,2,2,9,4,11,12,9,6)
> y1 <- c(99,86,87,88,111,103,87,94,78,77,85,86)
> x2 <- c(2,2,8,1,15,8,12,9,7,3,11,4,7,14,12)
> y2 <- c(100,105,84,105,90,99,90,95,94,100,79,112,91,80,85)
> plot(x1, y1, main="Observation of Cars", xlab="Car age", ylab="Car speed", col="red", cex=2)
> points(x2, y2, col="blue", cex=2)
```

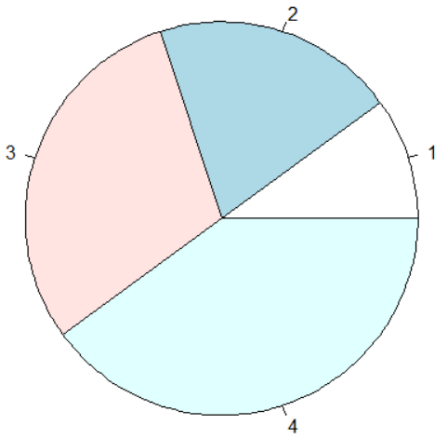


Щоб мати змогу побачити різницю порівняння, ви повинні присвоїти графікам різні кольори (за допомогою параметра col). Червоним кольором позначено значення дня 1, а синім - дня 2. Зверніть увагу, що ми також додали параметр cex для збільшення розміру точок.

Висновок спостереження: Порівнюючи два графіки, я думаю, що можна з упевненістю сказати, що вони обидва дають нам той самий висновок: чим новіший автомобіль, тим швидше він їздить.

Кругова діаграма - це кругове графічне представлення даних. Використовуйте функцію pie() для побудови секторних діаграм:

```
x <- c(10,20,30,40)
pie(x)
```



Як ви можете бачити, кругова діаграма малює один круг для кожного значення вектора (в даному випадку 10, 20, 30, 40). За замовчуванням, побудова першого круга починається з осі x і рухається проти годинникової стрілки.

Розмір кожного пирога визначається шляхом порівняння значення з усіма іншими значеннями, за допомогою цієї формули:

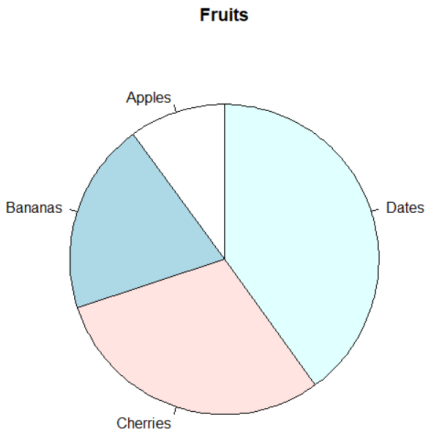
Значення, поділене на суму всіх значень: x/sum(x).

Ви можете змінити початковий кут кругової діаграми за допомогою параметра init.angle.

Значення init.angle задається кутом у градусах, де за замовчуванням кут дорівнює 0 (pie(x, init.angle = 90)).

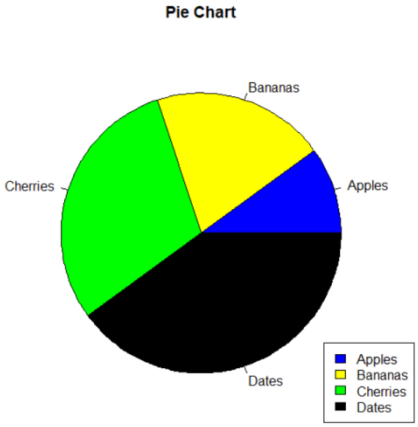
Використовуйте параметр label для додавання підписів до секторної діаграми, а параметр main - для додавання заголовка:

```
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")
pie(x, label = mylabel, main = "Fruits", init.angle = 90)
```



Ви можете додати колір до кожного пирога за допомогою параметра col. Щоб додати список пояснень до кожного пирога, використовуйте функцію legend():

```
colors <- c("blue", "yellow", "green", "black")
pie(x, label = mylabel, main = "Pie Chart", col = colors)
legend("bottomright", mylabel, fill = colors)
```



Легенда може бути розташована як завгодно: bottomright, bottom, bottomleft, left, topleft, top, topright, right, center.

Гістограма використовує прямокутні стовпчики для візуалізації даних. Стовпчасті діаграми можуть відображатися горизонтально або вертикально. Висота або довжина стовпчиків пропорційна до значень, які вони представляють.

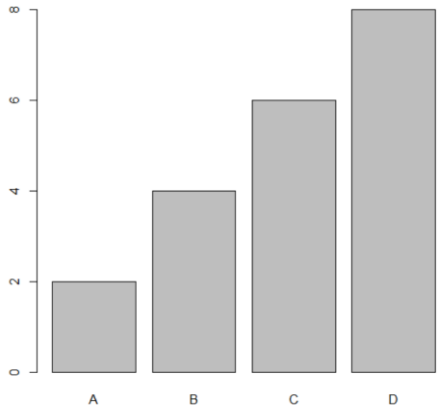
Використовуйте функцію barplot() для побудови вертикальної гістограми (рис.32). Змінна x представляє значення на осі x (A,B,C,D).

Змінна y представляє значення на осі y (2,4,6,8).

Потім ми використовуємо функцію barplot() для створення гістограми значень.

Змінна names.arg визначає назви кожного спостереження на осі x.

```
x <- c("A", "B", "C", "D")
y <- c(2, 4, 6, 8)
barplot(y, names.arg = x)
```



Використовуйте параметр col для зміни кольору смуг. Для зміни текстури стовпчиків використовуйте параметр density: barplot(y, names.arg = x, density = 10). Використовуйте параметр width для зміни ширини смуг: barplot(y, names.arg = x, width = c(1,2,3,4)). Якщо ви хочете, щоб стовпчики відображалися горизонтально, а не вертикально, використовуйте horiz=TRUE.

Статистика

Статистика - це наука про аналіз, огляд і висновки даних. Деякі основні статистичні числа включають:

- Середнє значення, медіана і мода
- Мінімальне та максимальне значення
- Процентилі
- дисперсія і стандартне відхилення
- Коваріація та кореляція
- Розподіл ймовірностей

Мова R була розроблена двома статистиками. Вона має багато вбудованих функцій, на додаток до бібліотек, призначених саме для статистичного аналізу.

Набір даних - це сукупність даних, часто представлених у вигляді таблиці.

Існує популярний вбудований набір даних у R під назвою "mtcars" (Motor Trend Car Road Tests), який отримано з журналу Motor Trend US за 1974 рік.

У наведених нижче прикладах (і в наступних розділах) ми будемо використовувати набір даних mtcars для статистичних цілей:

```
> mtcars
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
```

Використовуйте функцію dim(), щоб знайти розмірність набору даних, і функцію names(), щоб переглянути назви змінних. Використовуйте функцію rownames(), щоб отримати назву кожного рядка в першому стовпчику, тобто назву кожного автомобіля. Тепер, коли ми маємо деяку інформацію про набір даних, ми можемо почати аналізувати його за допомогою статистичних чисел.

Наприклад, ми можемо використати функцію summary() для отримання статистичного зведення даних:

```
> summary(mtcars)
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

Ми можемо використовувати функції which.max() і which.min(), щоб знайти індексну позицію максимального і мінімального значення в таблиці:

```
> Data_Cars <- mtcars
>
> which.max(Data_Cars$hp)
[1] 31
> which.min(Data_Cars$hp)
[1] 19
```

А ще краще об'єднати which.max() і which.min() з функцією rownames(), щоб отримати назву автомобіля з найбільшою і найменшою потужністю:

```
> rownames(Data_Cars)[which.max(Data_Cars$hp)]
[1] "Maserati Bora"
> rownames(Data_Cars)[which.min(Data_Cars$hp)]
[1] "Honda Civic"
```

У статистиці нас часто цікавлять три значення, які нас цікавлять:

- Mean - середнє значення
- Median - центральне значення
- Mode - найпоширеніше значення

Щоб обчислити середнє значення змінної з набору даних mtcars, знайдіть суму всіх значень і розділіть цю суму на кількість значень. На наше щастя, функція mean() в R може зробити це за вас. Медіанне значення - це значення посередині, після того, як ви відсортували всі значення. Якщо ми подивимося

на значення змінної wt (з набору даних mtcars), то побачимо, що посередині є два числа.

Якщо є два числа посередині, вам потрібно розділити суму цих чисел на два, щоб знайти медіану. На щастя, у R є функція, яка робить все це за вас: Просто використовуйте функцію median() для знаходження середнього значення. Модальне значення - це значення, яке з'являється найбільшу кількість разів. R не має функції для обчислення моди. Однак ми можемо створити власну функцію для її обчислення. Замість того, щоб обчислювати її самостійно, ми можемо використати наступний код для знаходження моди. R не має функції для обчислення моди. Однак ми можемо створити власну функцію для її знаходження.

```
> mean(Data_Cars$wt)
[1] 3.21725
> median(Data_Cars$wt)
[1] 3.325
> names(sort(-table(Data_Cars$wt)))[1]
[1] "3.44"
```

Перцентилі використовуються в статистиці, щоб дати вам число, яке описує значення, нижче якого знаходиться певний відсоток значень. Що таке 75-й процентиль ваги автомобілів? Відповідь: 3,61 або 3 610 фунтів, що означає, що 75% автомобілів важать 3 610 фунтів або менше:

```
> quantile(Data_Cars$wt, c(0.75))
 75%
3.61
```

Якщо запустити функцію quantile() без вказівки параметра c(), ви отримаєте процентилі 0, 25, 50, 75 і 100.

ТЕМА 3. Вступ до штучного інтелекту

Означення та історія виникнення ШІ

Штучний інтелект (ШІ, англ. Artificial intelligence) — наука та технологія створення інтелектуальних машин, в особливості інтелектуальних комп'ютерних програм. ШІ пов'язаний з завданням використання комп'ютерів для розуміння людського інтелекту, але не обов'язково обмежується біологічно правдоподібними методами (Джон Маккарті, 1956 р., конференція у Дартмутському університеті). В подальшому було зроблено чимало спроб дати формальне визначення інтелекту взагалі і інтелекту штучного зокрема. Найбільш відомим, очевидно, є визначення предмету теорії штучного інтелекту, що було дане видатним дослідником у галузі штучного інтелекту М. Мінські і яке у більш або менш видозміненому вигляді потрапило до словників та енциклопедій: "штучний інтелект є дисципліна, що вивчає можливість створення програм для вирішення задач, які при вирішенні їх людиною потребують певних інтелектуальних зусиль". Це визначення зустріло критику, яка полягала в тому, що під нього можна підвести що завгодно, наприклад, виконання простих арифметичних операцій. Відтак до цього визначення додається поправка: "сюди не входять задачі, для яких відома процедура їх вирішення". Таке визначення також важко вважати задовільним. Рассел та Норвіг наводять класифікацію означень ШІ у табличній формі

Системи, які мислять подібно до людини	Системи, які мислять раціонально
Системи, які діють подібно до людини	Системи, які діють раціонально

Єдиної відповіді на питання чим займається ШІ, не існує. Майже кожен автор дає своє визначення. Зазвичай ці визначення зводяться до наступних:

- штучний інтелект вивчає методи розв'язання задач, які потребують людського розуміння. Тут мова іде про те, щоб навчити ШІ розв'язувати тести інтелекту. Це передбачає розвиток способів розв'язання задач за аналогією, методів дедукції та індукції, накопичення базових знань і вміння їх використовувати.
- штучний інтелект вивчає методи розв'язання задач, для яких не існує способів розв'язання або вони не коректні (через обмеження в часі, пам'яті тощо).

Завдяки такому визначенню інтелектуальні алгоритми часто використовуються для розв'язання NP-повних задач, наприклад, задачі комівояжера.

- штучний інтелект займається моделюванням людської вищої нервової діяльності.
- штучний інтелект — це системи, які можуть оперувати з знаннями, а найголовніше — навчатися. В першу чергу мова ведеться про те, щоби визнати клас експертних систем (назва походить від того, що вони спроможні замінити «на посту» людей-експертів) інтелектуальними системами.

Останнє визначення, що з'явилося у 1990-х рр., засноване на так званому агентноорієнтованому підході. Цей підхід акцентує увагу на тих методах і

алгоритмах, які допоможуть інтелектуальному агенту виживати в оточуючому середовищі під час виконання свого завдання. Тому тут значно краще вивчаються алгоритми пошуку і прийняття рішення.

Джек Коупленд у праці "Що таке ШІ" відзначає, що незважаючи на наявність численних підходів та визначень як до розуміння ШІ, так і до створення інтелектуальних інформаційних систем, можна виділити два основні підходи щодо розроблення систем ШІ:

1. низхідний (Top-Down AI, семіотичний, символьний) — створення експертних систем, баз знань та систем логічного виведення, що моделюють та імітують високорівневі психічні процеси: мислення, міркування, мова, емоції, творчість і т.п.;
2. висхідний (Bottom-Up AI, біологічний, конекціоністський) — вивчення нейронних мереж і еволюційних обчислень, які моделюють інтелектуальну поведінку на основі біологічних елементів, а також створення відповідних обчислювальних систем, таких як нейрокомп'ютери.

Другий підхід, власне кажучи не відноситься до визначення ШІ, яке дав Дж. Маккарті. Їх поєднує тільки кінцева мета.

Відсутність чіткого визначення ШІ не заважає оцінювати інтелектуальність на інтуїтивному рівні. Можна навести як мінімум два методи такої оцінки:

метод експертних оцінок. Рішення про ступінь інтелектуальності приймає досить велика група експертів (незалежно або у взаємодії між собою);

метод тестування. Існує значна кількість так званих інтелектуальних тестів, апробованих практикою, і ці тести широко використовуються для оцінки рівня розумових здібностей людини, а також у психології та психіатрії. І метод експертних оцінок, і метод тестування мають свої недоліки. Головний з них полягає у тому, що оцінка дається виходячи лише з власних уявлень експертів, авторів тестів і т.п. про те, як має бути. Тому ці способи не дуже придатні для оцінки будь-якого інтелекту, крім людського.

Серед психіатрів можна почути вислів: "він міркує логічно вірно, але неправильно". Наприклад, дається тестове завдання: "серед слів соловей, чапля, перепілка, стілець, шпак виділити зайве".

Більшість людей, не задумуючись, дає відповідь стілець, тому що всі інші слова — це назви птахів. І раптом хтось дає відповідь шпак, пояснюючи це тим, що це єдине слово, в якому відсутня літера "л".

Обидві класифікації є логічно вірними і формально рівноправними. Але чому ж перевага віддається одній з них? Тому, що так міркує більшість людей. А чому так міркує більшість людей? Очевидно, тому, що перша класифікація вважається більш важливою для людської практики. Це положення приймається на аксіоматичному рівні, без доведення. Але те, що є більш важливим для людської практики, зовсім не обов'язково буде так само важливим для розумного робота або для інопланетянина.

Необхідно підкреслити, що поняття “штучний інтелект” не можна зводити лише до створення пристроїв, які імітують людину в усій повноті її діяльності.

Насправді ж, спеціалісти які працюють в цій області вирішують іншу задачу: виявити механізми, які лежать в основі діяльності людини, щоб застосувати їх при вирішенні конкретних науково-технічних задач. І це лише одна з можливих проблем.

Приклади інтелектуальних задач

До переліку інтелектуальних задач можна віднести:

- розпізнавання образів;
- логічне мислення;
- аналіз ситуації;
- розуміння нової інформації;
- навчання і самонавчання;
- планування цілеспрямованих дій.

Більш детально зупинимося на перших двох задачах.

На інтуїтивному рівні можна сформулювати декілька типових задач розпізнавання:

- ідентифікувати об’єкт, що спостерігається людиною, тобто вирізнити його серед інших (наприклад, побачивши іншу людину, впізнати у ній свою дружину);
- здійснити розпізнавання у класичній постановці, тобто віднести об’єкт, що спостерігається людиною, до одного з заздалегідь відомих класів об’єктів (наприклад, відрізнити легковий автомобіль від вантажного);
- провести кластеризацію (розбиття множини об’єктів на класи).

Людина робить класифікацію просто. Чоловік, повернувшись додому, відразу ж пізнає свою дитину, собаку і т.д. Але він рідко може пояснити, як він це робить. Якби це можна було зробити, алгоритми розпізнавання можна було б легко програмувати і широко застосовувати.

Теорія розпізнавання, яка інтенсивно розвивається, необхідна для того, щоб навчити вирішувати задачі розпізнавання і штучні інтелектуальні системи. Зокрема, сформульовано такий ключовий принцип: будь-який об’єкт у природі – унікальний; унікальні об`єкти – типізовані.

У відповідності до цього принципу, розпізнавання здійснюється на основі аналізу певних характерних ознак. Вважається, що в природі не існує двох об’єктів, для яких співпадають абсолютно всі ознаки, і це теоретично дозволяє здійснювати ідентифікацію. Якщо ж для деяких об’єктів співпадають деякі ознаки, ці об’єкти теоретично можна об’єднувати в групи, або класи, за цими співпадаючими ознаками. "Близькість" значень ознак дає змогу проводити розбиття множини на кластери.

Проблема полягає у тому, що різноманітних ознак дуже багато. Незважаючи на легкість, з якою людина проводить розпізнавання, вона дуже

рідко в змозі виділити ознаки, суттєві для цього. До того ж, об'єкти, як правило, змінюються з часом.

Розпізнавання об'єктів і ситуацій має виняткове значення для орієнтації людини в навколишньому світі і для прийняття правильних рішень. Розпізнавання, як правило, здійснюється людиною на інтуїтивному, підсвідомому рівні, людина навчилася цьому за мільйони років еволюції. На цьому фоні спроби навчити розпізнаванню складних об'єктів штучні інтелектуальні системи, навіть шляхом автоматизованого виділення характерних ознак, мають досить слабкі досягнення.

Логічне мислення — перехід від початкових положень до їх наслідків за формалізованими законами логіки. І тут пересічна людина рідко в змозі пояснити, за якими алгоритмами вона здійснює логічні побудови. Але методики, за якими можна автоматизувати логічне мислення, досить відомі.

Перш за все, це формальна логіка Аристотеля на основі конструкцій, які отримали назву силогізмів. Класичний приклад.

Перше положення. Усі люди смертні. Друге положення. Сократ — людина.

Висновок: Сократ смертний.

Якщо перше та друге положення у силогізмі істинні та задовольняють певним загальним формальним вимогам, тоді і висновок буде істинним незалежно від змісту тверджень, що входять до силогізму.

Аристотелем було запропоновано декілька формальних конструкцій силогізмів, які він вважав достатньо універсальними. У XIX столітті почала розвиватися сучасна математична логіка, яка розглядає аристотелевські силогізми як один із часткових випадків.

Основою більшості сучасних систем, призначених для автоматизації логічних побудов, є метод резолюцій Робінсона. Але практична автоматизація логічного мислення зіткнулася з двома серйозними проблемами.

Перша з них — феномен, який Р. Беллман називав прокляттям розмірності. Зовнішній світ являє собою винятково складне переплетіння різноманітних об'єктів та зв'язків між ними. Для того, щоб тільки ввести всю цю інформацію до пам'яті інтелектуального комп'ютера, може знадобитися не одна тисяча років.

Інша проблема — алгоритмічна нерозв'язність. Відомо, що в рамках будь-якої досить складної формальної теорії існують положення, які є істинними, але які не можна ні довести, ні спростувати (теорема Геделя про нерозв'язність формальної арифметики.

Реальні програми, які здійснюють логічне виведення (вони часто називаються експертними системами) мають досить обмежене застосування. Вони мають обмежений набір фактів та правил з певної, більш-менш чітко окресленої предметної галузі і можуть використовуватися лише у цій галузі.

Що ж стосується людини, то якість її логічного мислення часто буває далеким від бездоганного. Люди рідко проводять логічні побудови до кінця , часто роблять логічні помилки, а інколи взагалі керуються принципами, невірними з точки зору нормальної логіки, а рішення приймається на

підсвідомому, інтуїтивному рівні. Зрозуміло, що таке рішення може бути помилковим. Але, якби це було не так, людина була б практично не здатною ні до якої діяльності — ні до фізичної, ні до продуктивної розумової.

Для задач, які розглядалися вище, характерною була спільна риса:

їх погана формалізованість, відсутність або невизначеність чітких алгоритмів вирішення. Саме такі задачі і являють собою основний предмет розгляду в теорії штучного інтелекту.

До зовсім іншого класу відносяться обчислювальні задачі. Важко відповісти на запитання, як саме людина здійснює ті чи інші обчислення. Добре відомими є і низька швидкість, і невисока надійність цього виду людської діяльності. Але були запропоновані ефективні принципи комп'ютерних обчислень.

Тест Тьюринга

Праця А. Тьюринга "Обчислювальні машини та інтелект" з'явилася у 1950 р. і була однією з перших публікацій з даної тематики. Тюрінг розглянув питання про те, чи можна примусити машину думати по-справжньому. Відзначивши, що існує певна невизначеність у самому питанні (що таке "машина" і що значить "думати"), яка виключає можливість раціональної відповіді, Тюрінг запропонував замінити питання про інтелект більш чітким емпіричним тестом.

Тест Тьюринга порівнює здібності гадано "розумної" машини із здібностями людини — єдиним і найкращим стандартом розумної поведінки. В цьому тесті машину і її людського супротивника (слідчого, експерта) поміщають у окремі кімнати. Ще у одній кімнаті знаходиться "людина". Експерт не може бачити машину чи людину і може спілкуватися з ними лише за допомогою текстового пристрою, наприклад комп'ютерного термінала.

Слідчий повинен відрізнити комп'ютер від людини виключно за допомогою їх відповідей на запитання, які він їм задає. Якщо слідчий не може відрізнити комп'ютер від людини, тоді, як стверджує Тьюринг, машину потрібно визнати розумною.

Особливості тесту Тьюринга:

1. Тест дає об'єктивне уявлення про інтелект, тобто реакцію розумної людини на певний набір запитань.

2. Виключаються якісь особливі вимоги про внутрішню будову комп'ютера.

3. Виключає можливість упередженого ставлення на користь живих істот.

Методики, засновані на тесті Т. стали незамінними при розробці та перевірці сучасних експертних систем.

Багато спеціалістів вважали, що тест Тьюринга є цілком задовільним для визначення рівня інтелектуальності комп'ютерної системи. Але виявилось, що це не зовсім так.

В основі тесту Тьюринга лежить неявне припущення про те, що необхідною умовою ведення діалогу є розуміння співрозмовника. Але у кінці 60-х рр. американський кібернетик Дж. Вейценбаум створив дві програми — ЕЛІЗА і ЛІКАР. Вони були створені для використання в психіатрії, але

виявилось, що за їх допомогою можна обманути експерта в умовах тесту Тьюринга. В основі лежить ідея фатичного діалогу.

Фатичним діалогом називається діалог без розуміння співрозмовника, на основі формального перефразування почутого.

Так, наприклад, на висловлювання пацієнта "Ви не відверті зі мною" програма може відповісти "Чому Ви думаєте, що я не відверта з Вами?" У ряді випадків програма може видати заздалегідь заготовлену фразу. Так, на слова пацієнта "Мій батько мене не любив" вона може відповісти "Розкажіть мені про Вашу сім'ю".

Звичайно, це не має ніякого відношення ні до розуміння, ні до інтелекту, але багато експертів, які проводили з цими програмами тест Тьюринга, вирішили, що мають справу з людиною.

Фатичний діалог може бути і менш примітивним, але системи, які здатні тільки на підтримку такого діалогу, не можна вважати інтелектуальними. Розуміння співрозмовника є абсолютно необхідним для нормального діалогу, а для такого розуміння необхідно мати певну суму знань про світ.

Поява систем, що реалізують фатичний діалог, завдала сильного удару по визначенню Тьюринга. Безумовно, воно зберігає своє методологічне значення, але вже не може претендувати на універсальність (так само, як і будь-яке інше з відомих визначень).

Класичний метод написання програм, що реалізують фатичний діалог — застосування співставлення зі зразком. В основі методу лежить співставлення речень, які вводяться людиною, зі зразками, що зберігаються програмою. В залежності від результату співставлення реалізується та чи інша заздалегідь запрограмована операція. І самих зразків, і операцій, що їм відповідають, як правило, порівняно небагато.

Що таке машинне навчання

Машинне навчання - це підрозділ ШІ, що включає методи побудови алгоритмів, здатних навчатися.

Машинне навчання - це підрозділ ШІ, математична дисципліна, що використовує розділи математичної статистики, чисельні методи оптимізації, виділяє знання з даних.

Машинне навчання вивчає методи побудови алгоритмів, які можуть навчатися з даних і робити прогноз на даних.

Машинне навчання - процес, у результаті якого машина здатна показувати поведінку, яку в неї не було явно закладено.

Машинне навчання (ML) прагне автоматично вивчати значущі зв'язки та шаблони на прикладах і спостереженнях. Досягнення в ML сприяли нещодавньому зростанню інтелектуальних систем з людськими когнітивними можливостями, які проникають у наше ділове та особисте життя та формують мережеву взаємодію на електронних ринках усіма мислимими способами, завдяки чому компанії покращують процес прийняття рішень для продуктивності, залученості та утримання співробітників, системи помічників, які можна навчити, адаптуються до індивідуальних уподобань користувачів, а торгові агенти похитують традиційні ринки фінансової торгівлі.

Крім розкрученого вигляду, науковцям, як і професіоналам, потрібне глибоке розуміння базових концепцій, процесів, а також викликів для впровадження такої технології. На цьому фоні ідея полягає в тому, щоб передати розуміння машинного навчання та глибокого навчання (DL) у контексті електронних ринків. Таким чином, спільнота може отримати вигоду з цих технологічних досягнень — чи то для вивчення великих і високорозмірних активів даних, зібраних у цифрових екосистемах, чи для розробки нових інтелектуальних систем для електронних ринків. Щоб забезпечити розуміння галузі, необхідно розрізняти кілька відповідних термінів і понять один від одного.

У широкому розумінні штучний інтелект включає в себе будь-яку техніку, яка дозволяє комп'ютерам імітувати людську поведінку та відтворювати або перевершувати прийняття людських рішень для вирішення складних завдань самостійно або з мінімальним людським втручанням. Як таке, воно пов'язане з різними центральними проблемами, включаючи представлення знань, міркування, навчання, планування, сприйняття та комунікацію, і стосується різноманітних інструментів і методів (наприклад, міркування на основі випадків, системи на основі правил, генетичні алгоритми, нечіткі моделі, мультиагентні системи). Ранні дослідження штучного інтелекту зосереджувались насамперед на жорстко закодованих висловлюваннях у формальних мовах, які потім комп'ютер може автоматично вираховувати на основі правил логічного висновку. Ця подія також відома як підхід бази знань.

Однак парадигма стикається з кількома обмеженнями, оскільки людям, як правило, важко пояснити всі свої неявні знання, необхідні для виконання складних завдань.

Машинне навчання долає такі обмеження. Загалом, ML означає, що продуктивність комп'ютерної програми покращується з досвідом щодо певного класу завдань і показників ефективності. Таким чином, він спрямований на автоматизацію завдання побудови аналітичної моделі для виконання когнітивних завдань, таких як виявлення об'єктів або переклад природною мовою. Це досягається шляхом застосування алгоритмів, які ітеративно вивчають дані навчання, що стосуються конкретної проблеми, що дозволяє комп'ютерам знаходити приховані ідеї та складні шаблони без явного програмування.

Особливо в задачах, пов'язаних з великомірними даними, такими як класифікація, регресія та кластеризація, ML демонструє хорошу застосовність. Вивчаючи попередні обчислення та витягуючи закономірності з масивних баз даних, це може допомогти отримати надійні та повторювані рішення. З цієї причини алгоритми ML були успішно застосовані в багатьох сферах, таких як виявлення шахрайства, оцінка кредитоспроможності, аналіз наступної найкращої пропозиції, розпізнавання мови та зображень або обробка природної мови (NLP).

Виходячи з поставленої проблеми та наявних даних, ми можемо виділити три типи ML: навчання з вчителем, навчання без вчителя та навчання з підкріпленням.

Навчання з вчителем

Навчання з вчителем вимагає навчального набору даних, який охоплює приклади для вхідних даних, а також відповіді з мітками або цільові значення для вихідних даних. Прикладом може бути передбачення кількості активних користувачів, підписаних на ринкову платформу протягом місяця, як результат (вважається цільовою змінною або змінною y) на основі різних вхідних характеристик, таких як кількість проданих продуктів або позитивні відгуки користувачів (часто згадуються як вхідні функції або змінні x). Потім пари вхідних і вихідних даних у навчальному наборі використовуються для калібрування відкритих параметрів моделі ML. Коли модель успішно навчена, її можна використовувати для прогнозування цільової змінної y з урахуванням нових або невидимих точок даних вхідних функцій x.

Що стосується типу навчання з вчителем, ми можемо додатково розрізнити проблеми регресії, де прогнозується числове значення (наприклад, кількість користувачів), і проблеми класифікації, де результатом передбачення є категорична класова приналежність, наприклад «глядачі» або «покупці».

Навчання без вчителя

Навчання без вчителя має місце, коли система навчання має виявляти шаблони без будь-яких попередніх позначок чи специфікацій. Таким чином, навчальні дані складаються лише зі змінних x з метою пошуку цікавої структурної інформації, такої як групи елементів, які мають спільні властивості (відомі як кластеризація),

або представлення даних, які проектуються з простору великої розмірності в нижчий (відоме як зменшення розмірності). Яскравим прикладом неконтрольованого навчання на електронних ринках є застосування методів кластеризації для групування клієнтів або ринків у сегменти з метою більш конкретного спілкування з цільовою групою.

Навчання з підкріпленням

У системі навчання з підкріпленням замість того, щоб надавати пари вхідних і вихідних даних, ми описуємо поточний стан системи, визначаємо мету, надаємо перелік дозволених дій та їх обмежень середовища для їх результатів, і дозволяємо моделі ML випробувати процес самостійне досягнення мети за принципом проб і помилок для максимізації винагороди.

Моделі навчання з підкріпленням з великим успіхом застосовуються в середовищах закритого світу, таких як ігри, але вони також актуальні для багатоагентних систем, таких як електронні ринки.

До 1950 - Статистичні методи

1950-ті рр. - Початок (шашки Самуеля, персептрони, логічний висновок, ...)

1960-ті рр. - Байєсівські методи

1970-ті рр. - "Зима" ШІ

1980-ті рр. - Backpropagation, згорткові мережі та ін. - "відлига"

1990-ті рр. - Машина опорних векторів та ін. (зміщення від дедуктивного навчання до індуктивного)

2000-ті рр. - Ансамблі дерев рішень, "ядерні" методи

2010-ті рр. - Глибоке навчання (третя весна "ШІ")

Data Mining vs Machine Learning

Отже, і ML, і DM витягують закономірності ("знання") з даних, але (трохи) з різними цілями:

- ML - щоб навчити машину;
- DM - щоб навчити людину.

Тому насамперед:

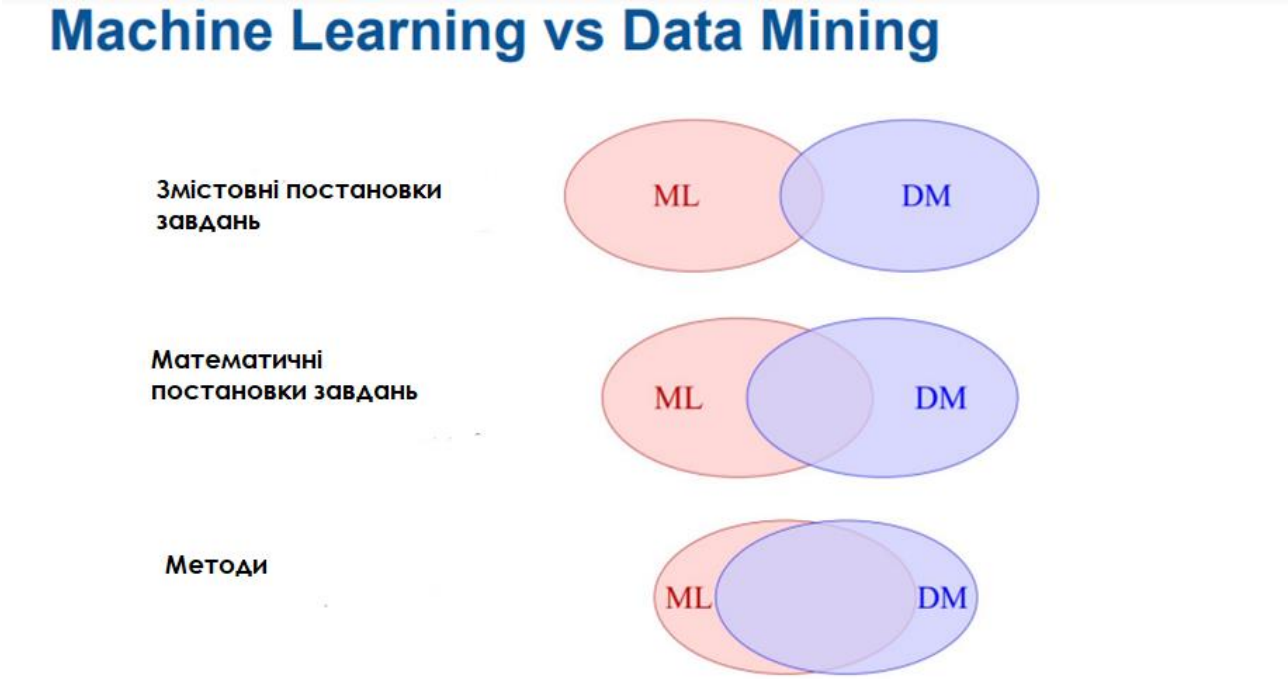
- у ML мінімізують помилку;
- у DM важлива інтерпретованість результату.

Є трохи інша точка зору на питання, чим ML відрізняється від DM:

DM має справу зі змістовними завданнями, а ML - з математичною теорією, звідси трохи дивні терміни, наприклад, "алгоритми машинного навчання в аналізі даних"

Науки про знання (Data Science)

- Бази даних
- Великі дані (Big Data)
- Data Mining
- Machine Learning
- …



Розділи математики, що використовуються в машинному навчанні

- Лінійна алгебра
- Теорія ймовірностей та математична статистика
- Методи оптимізації
- Чисельні методи
- Математичний аналіз
- Дискретна математика
- ін.

Основи Data Mining

Для знаходження прихованих знань потрібно застосовувати спеціальні методи автоматичного аналізу, за допомогою яких доводиться практично добувати знання з "завалів" інформації: методи та алгоритми Data Mining.

Термін Data Mining дістав назву від двох понять: дані – data і переробка сирого матеріалу (гірської руди) – mining. Однозначного перекладу на українську мову не має, використовуються наступні переклади:

— видобуток даних
— розвідка даних
— глибинний аналіз даних
— просіювання інформації
— вилучення даних
— знаходження знань в базах даних
— інтелектуальний аналіз даних

Деякі дослідники вважають невдалими більшість варіантів перекладу і оперують англомовними термінами. Ми будемо дотримуватися термінів Інтелектуальний аналіз даних та Data Mining.

ІАД – Data Mining: застосовується для дослідження даних, обробки зразків даних, очищення і збору даних, опису здобуття знань у базах даних. ІАД є процесом виявлення кореляції, тенденцій, шаблонів, зв'язків і категорій.

За визначенням SAS Institute, Data Mining – це процес виділення, дослідження і моделювання великих обсягів даних для виявлення невідомих до цього структур (patterns) з метою досягнення переваг у бізнесі.

За визначенням Gartner Group, Data Mining – це процес, мета якого – виявляти нові кореляції, зразки і тенденції у результаті просіювання великого обсягу даних з використанням методик розпізнавання зразків і статистичних та математичних методів.

Найбільш повне та точне, класичне означення Data Mining було дано Григорієм П'ятецьким-Шапіро, якого вважають одним із засновників та ідеологів таких напрямів, як Data Mining та Knowledge Discovery in Data (KDD - знаходження знань у базах даних):

Інтелектуальний аналіз даних (ІАД, Data Mining) – процес виявлення у необроблених даних раніше невідомих нетривіальних, практично корисних і доступних інтерпретацій знань, необхідних для прийняття рішень у різних сферах діяльності.

П'ятецький-Шапіро працював над багатьма цікавими проектами: прогнозування відтоку клієнтів мобільного оператора, аналіз даних в області охорони здоров’я, аналіз роботи програмного забезпечення, прогнозування ризиків захворювання Альцгеймера, виявлення банківського шахрайства, виявлення підробки ювелірних виробів, виявлення проблемних місць медицинського страхування телефонної компанії для зменшення фінансування на покриття страхових виплат.

Стосовно нових течій та напрямів ІАД П'ятецький-Шапіро виділяє обробку потокових даних та використання геоданних для пошуку просторово-часових закономірностей, збереження анонімності особи при аналізі великих даних (зокрема, у охороні здоров’я).

Розглянемо властивості виявлених знань, згідно визначення:

— Знання повинні бути новими, раніше невідомими. Затрати на отримання відомих знань не окупаються.

— Знання повинні бути нетривіальні. Результати аналізу повинні відображати неочевидні, несподівані закономірності в даних, що містять сховані знання. Результати, які могли б бути отримані більше простими способами (наприклад, візуальним переглядом), не виправдують залучення потужних методів Data Mining.

— Знання повинні бути практично корисними. Знайдені знання повинні бути застосовні, у тому числі й на нових даних, з досить високим ступенем вірогідності. Корисність полягає в тім, щоб ці знання могли принести певну вигоду при їхньому застосуванні.

— Знання повинні бути доступні для розуміння людині. Знайдені закономірності повинні бути логічно з'ясовні, у противному випадку існує ймовірність, що вони є випадковими. Крім того, виявлені знання повинні бути представлені в зрозумілій для людини формі.

В цілому в інтелектуальному аналізі даних прийнято вважати, що знайдене знання повинно мати наступні властивості:

— знання відображає результат дослідження системи (пізнання об'єктивної реальності);

— знання виражене певним, зрозумілим людині чином (використовує загальноприйняті символи, поняття, природну мову);

— знання повинне бути компактним (за формою, опису), що робить його доступним до розуміння, інтерпретації і подальшого використання.

В Data Mining для подання отриманих знань служать моделі. Види моделей залежать від методів їх створення.

Виникнення і розвиток Data Mining зумовлені такими основними факторами:

— вдосконалення програмно-апаратного забезпечення;

— вдосконалення технологій зберігання і запису даних;

— накопичення великої кількості ретроспективних даних;

— вдосконалення алгоритмів обробки інформації.

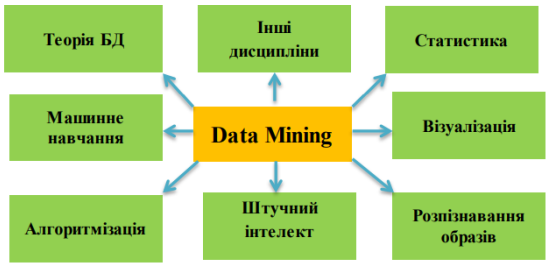
Сутність і мета Data Mining: це технологія, призначена для пошуку у великих інформаційних масивах даних неочевидних, об'єктивних, корисних на практиці закономірностей. ІАД здійснюється за допомогою використання технологій розпізнавання шаблонів, а також статистичних і математичних методів.

В основу технології Data Mining покладено концепцію шаблонів (patterns), що є закономірностями, які властиві вибіркам даних і можуть бути подані у формі, зрозумілій людині. При розвідці даних багаторазово виконуються операції і перетворення над "сирими" даними (відбір ознак, стратифікація (розшарування даних), кластеризація, візуалізація, регресія), що призначені для знаходження:

- структур, які інтуїтивно зрозумілі для людей і краще розкривають суть процесів, що лежать в основі їх протікання;

- моделей, які можуть передбачити результат або значення певних ситуацій, використовуючи історичні або суб'єктивні дані.

Технологія ІАД є міждисциплінарною областю дослідження. Предметна область Data Mining виникла і розвивається на базі: прикладної статистики, розпізнавання образів, штучного інтелекту, теорії баз даних, інших дисциплін.



Технологія ІАД, використовуючи методи таких дисциплін, як теорія інформації, системи штучного інтелекту, теорія ймовірностей, математична статистика, машинне навчання та інших має велику кількість методів і алгоритмів, реалізованих у різних діючих системах Data Mining. Багато з таких систем інтегрують в собі відразу кілька підходів. Проте, як правило, в кожній системі є якась ключова компонента, на яку робиться головна ставка. При цьому основна увага приділяється обчислювальній ефективності використовуваних алгоритмів при обробці великих обсягів даних.

ТЕМА 4. Машинне навчання

Чому машинне навчання важливе?

Штучний інтелект визначатиме наше майбутнє сильніше, ніж будь-яка інша інновація цього століття. Той, хто цього не розуміє, незабаром відчує себе покинутим, прокинувшись у світі, сповненому технологій, який все більше нагадує магію.

Швидкість прискорення вже вражає. Після кількох зим штучного інтелекту та періодів марних надій за останні чотири десятиліття, швидкий прогрес у сфері зберігання даних та комп'ютерних обчислювальних потужностей кардинально змінив правила гри в останні роки.

У 2015 році компанія Google навчила розмовного агента (ШІ), який міг не лише переконливо взаємодіяти з людьми як служба технічної підтримки, а й обговорювати питання моралі, висловлювати думки та відповідати на загальні запитання, що базуються на фактах.

Того ж року DeepMind розробив агента, який перевершив людський рівень у 49 іграх Atari, отримуючи на вході лише пікселі та оцінку гри. Невдовзі, у 2016 році, DeepMind застаріли власне досягнення, випустивши новий надсучасний ігровий метод під назвою A3C.

Тим часом AlphaGo перемогла одного з найкращих гравців у Го - надзвичайне досягнення у грі, в якій люди домінували протягом двох десятиліть після того, як машини вперше підкорили шахи. Багато майстрів не могли збагнути, як машина може осягнути всі нюанси і складність цієї стародавньої китайської військово-стратегічної гри з 10^{170} можливих позицій на дошці (у Всесвіті існує лише 10^{80} атомів).

У березні 2017 року OpenAI створив агентів, які винайшли власну мову, щоб співпрацювати та ефективніше досягати своєї мети. Незабаром після цього Facebook, як повідомлялося, успішно навчив агентів вести переговори і навіть брехати.

Лише кілька днів тому (на момент написання цієї статті), 11 серпня 2017 року, OpenAI досягнув ще однієї неймовірної віхи, перемігши найкращих світових професіоналів у матчах 1v1 в онлайн-грі Dota 2.

Значна частина наших повсякденних технологій працює завдяки штучному інтелекту. Наведіть камеру на меню під час наступної подорожі до Тайваню, і вибір ресторану чарівним чином з'явиться англійською мовою через додаток Google Translate.

Сьогодні ШІ використовується для розробки науково обґрунтованих планів лікування онкохворих, миттєвого аналізу результатів медичних аналізів для негайного перенаправлення до відповідного фахівця, а також для проведення наукових досліджень з метою розробки ліків.

Одна порада: важливо розглядати знання як своєрідне семантичне дерево - переконайтеся, що ви розумієте фундаментальні принципи, тобто стовбур і великі гілки, перш ніж переходити до листя/деталей, інакше їм не буде за що зачепитися. - Ілон Маск, Reddit AMA

Штучний інтелект - це дослідження агентів, які сприймають навколишній світ, формують плани та приймають рішення для досягнення своїх цілей. Його основи включають математику, логіку, філософію, ймовірність, лінгвістику, неврологію та теорію прийняття рішень. Багато галузей підпадають під сферу ШІ, таких як комп'ютерний зір, робототехніка, машинне навчання та обробка природної мови.

Машинне навчання - це підгалузь штучного інтелекту. Його мета - дати можливість комп'ютерам навчатися самостійно. Алгоритм навчання машини дозволяє їй виявляти закономірності у спостережуваних даних, будувати моделі, що пояснюють світ, і прогнозувати події без чітких попередньо запрограмованих правил і моделей.

Ефект штучного інтелекту: що насправді є "штучним інтелектом"?

Точний стандарт для технологій, які кваліфікуються як "штучний інтелект", є дещо розмитим, а інтерпретації змінюються з часом. Термін "штучний інтелект", як правило, описує машини, що виконують завдання, які традиційно належать до компетенції людини. Цікаво, що коли комп'ютери з'ясовують, як виконати одне з цих завдань, люди схильні говорити, що це не зовсім інтелект. Це відомо як ефект штучного інтелекту.

Наприклад, коли Deep Blue від IBM переміг чемпіона світу з шахів Гаррі Каспарова в 1997 році, люди скаржилися, що він використовував методи "грубої сили" і це зовсім не був "справжній" інтелект. Як писала Памела МакКордук: "Частиною історії галузі штучного інтелекту є те, що кожного разу, коли хтось з'ясовував, як змусити комп'ютер щось робити - добре грати в шашки, розв'язувати прості, але відносно неформальні задачі, - з'являвся хор критиків, які казали: "Це не мислення"" (McCorduck, 2004).

Можливо, існує певне "je ne sais quoi", притаманне тому, що люди надійно сприймають як "штучний інтелект":

"ШІ - це те, що ще не було зроблено". - Дуглас Хофстедер

Тож чи вважається калькулятор штучним інтелектом? Можливо, в певному розумінні. А як щодо безпілотного автомобіля? Сьогодні - так. У майбутньому, можливо, ні. Ваш новий крутий стартап-чат-бот, який автоматизує блок-схему? Звичайно... чому б і ні.

Потужний ШІ назавжди змінить наш світ; щоб зрозуміти, як саме, варто почати з вивчення машинного навчання

Технології, про які йшлося вище, є прикладами штучного вузького інтелекту (ШІ), який може ефективно виконувати вузько визначені завдання.

Тим часом ми продовжуємо робити фундаментальні кроки на шляху до штучного загального інтелекту людського рівня (ШЗІ), також відомого як сильний ШІ. Визначення ШІ - це штучний інтелект, який може успішно виконувати будь-яке інтелектуальне завдання, яке може виконувати людина, включаючи навчання, планування та прийняття рішень в умовах невизначеності, спілкування природною мовою, жарти, маніпулювання людьми, торгівлю акціями або... перепрограмування самого себе.

І це останнє дуже важливо. Як тільки ми створимо ШІ, здатний самовдосконалюватися, він розблокує цикл рекурсивного самовдосконалення,

що може призвести до інтелектуального вибуху протягом невідомого періоду часу, від багатьох десятиліть до одного дня.

У нещодавньому звіті Інституту майбутнього людства (Future of Humanity Institute) було проведено опитування групи дослідників ШІ щодо часових рамок для AGI, і було виявлено, що "дослідники вважають, що існує 50% ймовірність того, що ШІ перевершить людину у виконанні всіх завдань через 45 років" (Grace et al, 2017). Ми особисто спілкувалися з низкою розсудливих і розумних практиків у галузі ШІ, які прогнозують набагато довші терміни (верхня межа - "ніколи"), а також з іншими, чиї прогнози є тривожно короткими - всього кілька років.

Поява штучного суперінтелекту, вищого за людський, може стати однією з найкращих або найгірших подій, які коли-небудь траплялися з нашим видом. Це несе з собою величезну проблему визначення того, чого ШІ буде хотіти у дружній до людини спосіб.

Хоча неможливо сказати, що чекає на нас у майбутньому, одне можна сказати напевно: 2017 рік - вдалий час для того, щоб почати розуміти, як мислять машини. Щоб вийти за межі абстракцій філософа в кріслі і розумно формувати наші дорожні карти і політику щодо ШІ, ми повинні розібратися в деталях того, як машини бачать світ - чого вони "хочуть", їхні потенційні упередження і режими невдач, їхні темпераментні примхи - так само, як ми вивчаємо психологію і нейронауки, щоб зрозуміти, як люди навчаються, приймають рішення, діють і відчувають.

Чому ми хочемо, щоб машини вчилися?

Уявімо Біллі. Біллі хоче купити машину. Він намагається підрахувати, скільки йому потрібно відкладати щомісяця для цього. Він переглянув десятки оголошень в інтернеті і дізнався, що нові автомобілі коштують близько $20 000, вживані однорічні - $19 000, дворічні - $18 000 і так далі.

Біллі, наш геніальний аналітик, починає помічати закономірність: так, ціна автомобіля залежить від його віку і падає на $1,000 щороку, але не опускається нижче $10,000.

Говорячи мовою машинного навчання, Біллі винайшов регресію - він передбачив значення (ціну) на основі відомих історичних даних. Люди роблять це постійно, коли намагаються оцінити розумну вартість вживаного iPhone на eBay або з'ясувати, скільки реберець купити для вечірки барбекю. 200 грамів на людину? 500?

Так, було б непогано мати просту формулу для всіх проблем у світі. Особливо, для барбекю-вечірки. На жаль, це неможливо.

Повернімося до автомобілів. Проблема в тому, що вони мають різні дати виробництва, десятки опцій, технічний стан, сезонні сплески попиту і бозна-скільки ще прихованих факторів. Пересічний Біллі не може тримати всі ці дані в голові, коли розраховує ціну.

Люди помиляються – нам потрібні роботи, які зроблять математику за них. Отже, давайте підемо обчислювальним шляхом. Надамо машині деякі дані і попросимо її знайти всі приховані закономірності, пов'язані з ціною.

І вона працює. Найцікавіше те, що машина справляється з цим завданням набагато краще, ніж реальна людина, яка ретельно аналізує всі залежності в своїй голові.

Так народилося машинне навчання.

Три складові машинного навчання

Якщо не брати до уваги всі ці штучні інтелекти, єдиною метою машинного навчання є прогнозування результатів на основі вхідних даних. І все. Всі завдання ML можна представити таким чином, або це не завдання ML з самого початку.

Чим більша різноманітність у вибірках, тим легше знайти відповідні закономірності та передбачити результат. Отже, нам потрібні три компоненти для навчання машини:

Дані. Хочете виявити спам? Отримайте зразки спам-повідомлень. Хочете прогнозувати акції? Знайдіть історію цін. Хочете дізнатися вподобання користувачів? Проаналізуйте їхню активність у Facebook. Чим різноманітніші дані, тим кращий результат. Десятки тисяч рядків - це мінімум для найвідчайдушніших.

Існує два основні способи отримання даних - ручний та автоматичний. Дані, зібрані вручну, містять набагато менше помилок, але на їх збір йде більше часу, що робить їх дорожчими в цілому.

Автоматичний підхід дешевший - ви збираєте все, що можете знайти, і сподіваєтеся на краще.

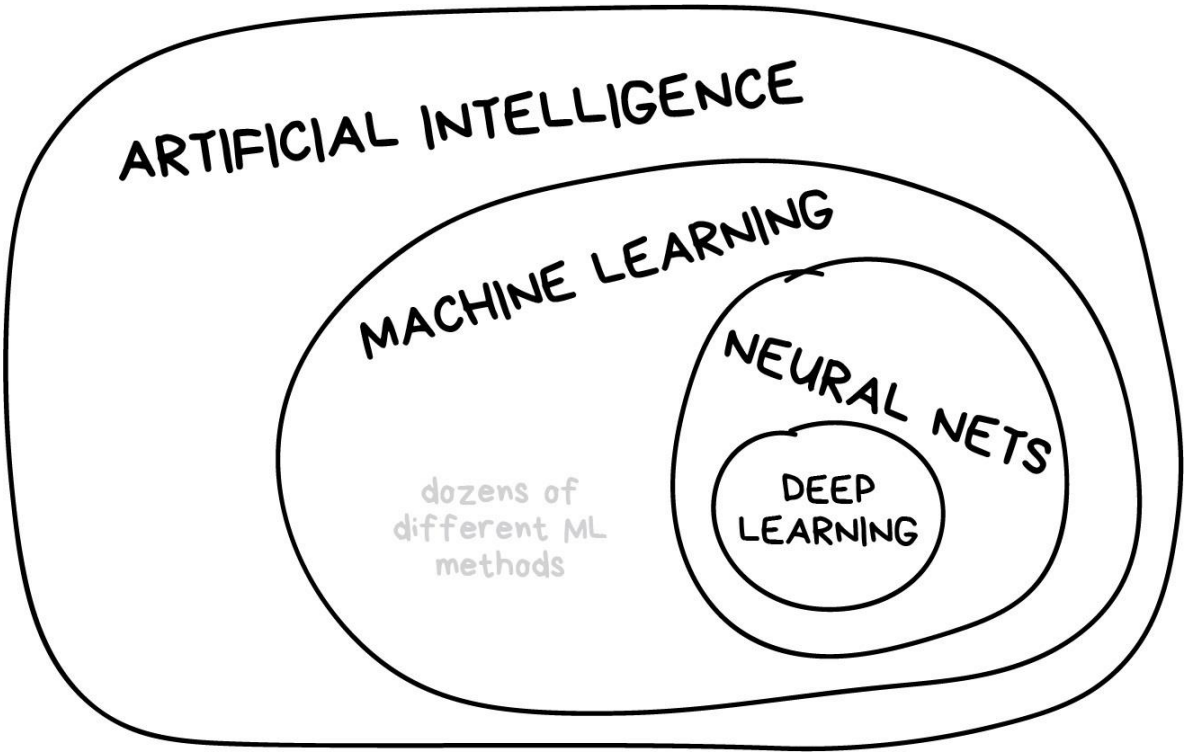
Деякі розумники, такі як Google, використовують власних клієнтів для маркування даних для них безкоштовно. Пам'ятаєте ReCaptcha, яка змушує вас "Вибрати всі вуличні знаки"? Це саме те, що вони роблять. Безкоштовна робоча сила! Чудово. Зібрати хорошу колекцію даних (зазвичай їх називають датасетом) надзвичайно складно. Вони настільки важливі, що компанії можуть навіть розкривати свої алгоритми, але рідко - набори даних.

Характеристики. Також відомі як параметри або змінні. Це може бути пробіг автомобіля, стать користувача, ціна акцій, частота слів у тексті. Іншими словами, це фактори, на які звертає увагу машина.

Коли дані зберігаються в таблицях, все просто - характеристики - це назви стовпців. Але що це таке, якщо у вас є 100 Гб фотографій котів? Ми не можемо вважати кожен піксель характеристикою. Ось чому вибір правильних ознак зазвичай займає набагато більше часу, ніж всі інші частини ML. Це також основне джерело помилок. Люди завжди суб'єктивні. Вони вибирають тільки ті характеристики, які їм подобаються або здаються "більш важливими".

Алгоритми. Найочевидніша частина. Будь-яку проблему можна вирішити по-різному. Метод, який ви оберете, впливає на точність, продуктивність і розмір кінцевої моделі. Але є один важливий нюанс: якщо дані погані, навіть найкращий алгоритм не допоможе. Іноді це називають "сміття на вході - сміття на виході". Тому не звертайте надто багато уваги на відсоток точності, спробуйте спочатку зібрати більше даних.

Навчання vs інтелект



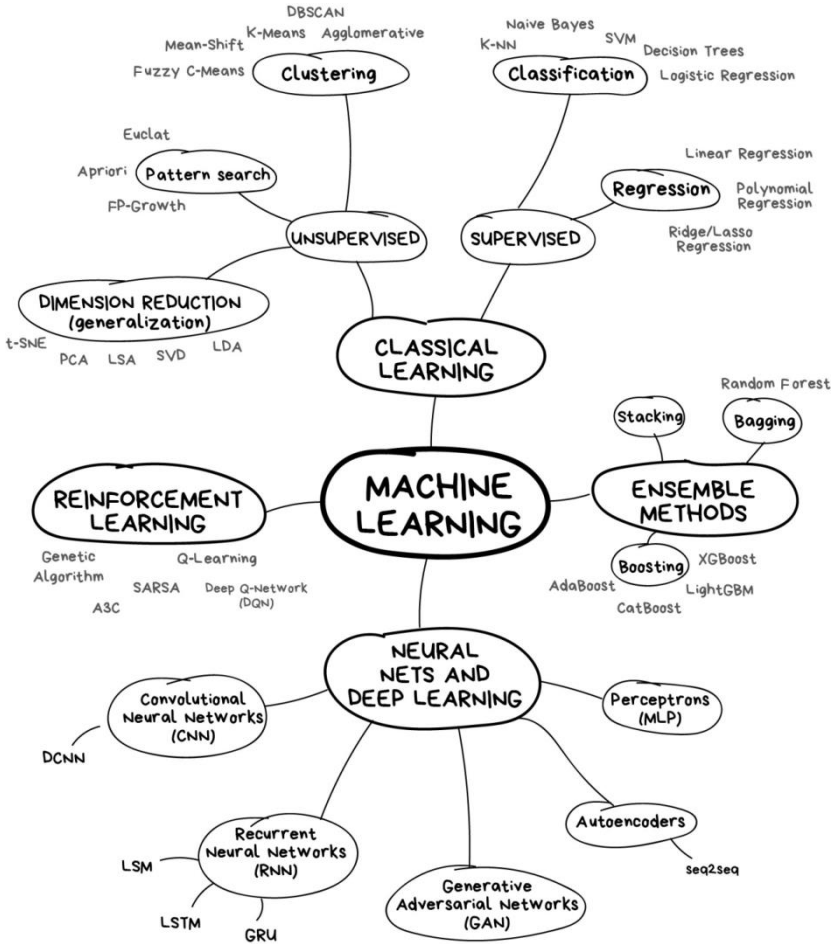
Штучний інтелект - це назва цілої галузі знань, подібно до біології чи хімії.

Машинне навчання - це частина штучного інтелекту. Важлива частина, але не єдина.

Нейронні мережі - один з видів машинного навчання. Популярний, але є й інші хороші хлопці в класі.

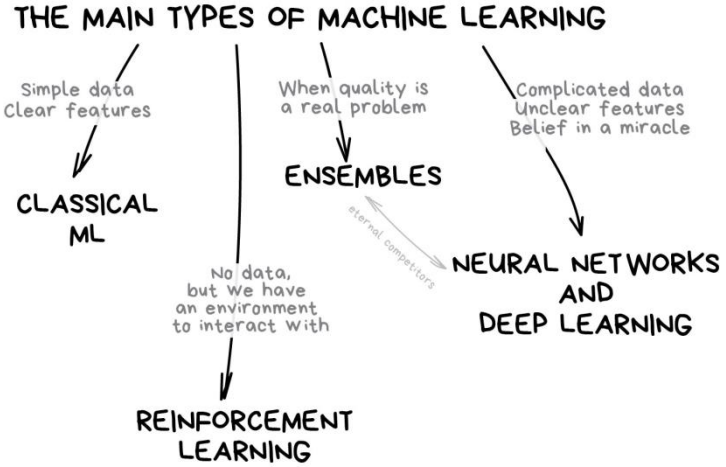
Глибоке навчання - це сучасний метод побудови, навчання та використання нейронних мереж. По суті, це нова архітектура. Зараз на практиці ніхто не відокремлює глибоке навчання від "звичайних мереж". Ми навіть використовуємо для них одні й ті ж бібліотеки. Щоб не виглядати дурнем, краще просто назвати тип мережі і уникати модних слів.

Machine can	Machine cannot
Forecast	Create something new
Memorize	Get smart really fast
Reproduce	Go beyond their task
Choose best item	Kill all humans



Завжди важливо пам'ятати - у світі машинного навчання ніколи не існує єдиного способу вирішити проблему. Завжди є кілька алгоритмів, які підходять, і ви повинні вибрати, який з них підходить краще. Звісно, все можна вирішити за допомогою нейромережі, але хто буде платити за всі ці GeForces?

Почнемо з базового огляду. На сьогоднішній день існує чотири основні напрямки машинного навчання.



Основні алгоритми моделей машинного навчання

1. **Дерево прийняття рішень**

Це метод підтримки ухвалення рішень, що ґрунтується на використанні деревоподібного графа: моделі ухвалення рішень, яка враховує їхні потенційні наслідки (з розрахунком імовірності настання тієї чи іншої події), ефективність, ресурсовитратність.

Для бізнес-процесів це дерево складається з мінімальної кількості запитань, які передбачають однозначну відповідь - "так" або "ні". Послідовно давши відповіді на всі ці запитання, ми приходимо до правильного вибору. Методологічні переваги дерева ухвалення рішень - у тому, що воно структурує і систематизує проблему, а підсумкове рішення ухвалюється на основі логічних висновків.

2. **Наївна байєсівська класифікація**

Наївні байєсівські класифікатори відносяться до сімейства простих імовірнісних класифікаторів і беруть початок з теореми Байєса, яка стосовно даного випадку розглядає функції як незалежні (це називається суворим, або наївним, припущенням). На практиці використовується в таких галузях машинного навчання:

- визначення спаму, що приходить на електронну пошту;
- автоматична прив'язка новинних статей до тематичних рубрик;
- виявлення емоційного забарвлення тексту;
- розпізнавання облич та інших патернів на зображеннях.

3. **Метод найменших квадратів**

Усім, хто хоч трохи вивчав статистику, знайоме поняття лінійної регресії. До варіантів її реалізації належать і найменші квадрати. Зазвичай за допомогою лінійної регресії розв'язують задачі з підгонки прямої, яка проходить через безліч точок. Ось як це робиться за допомогою методу найменших квадратів: провести пряму, виміряти відстань від неї до кожної з точок (точки і лінію з'єднують вертикальними відрізками), отриману суму перенести наверх. У результаті та крива, в якій сума відстаней буде найменшою, і є шукана (ця лінія пройде через точки з нормально розподіленим відхиленням від істинного значення).

Лінійну функцію зазвичай використовують під час підбору даних для машинного навчання, а метод найменших квадратів - для зведення до мінімуму похибок шляхом створення метрики помилок.

4. **Логістична регресія**

Логістична регресія - це спосіб визначення залежності між змінними, одна з яких категоріально залежна, а інші незалежні. Для цього застосовується логістична функція (акумулятивний логістичний розподіл). Практичне значення логістичної регресії полягає в тому, що вона є потужним статистичним методом передбачення подій, який містить у собі одну або кілька незалежних змінних. Це затребуване в таких ситуаціях:

- кредитний скоринг;
- заміри успішності проведених рекламних кампаній;
- прогноз прибутку з певного товару;
- оцінка ймовірності землетрусу в конкретну дату.

5. **Метод опорних векторів (SVM)**

Це цілий набір алгоритмів, необхідних для розв'язання задач на класифікацію та регресійний аналіз. Виходячи з того, що об'єкт, який перебуває в N-вимірному просторі, належить до одного з двох класів, метод опорних

векторів будує гіперплощину з мірністю (N - 1), щоб усі об'єкти опинилися в одній із двох груп. На папері це можна зобразити так: є точки двох різних видів, і їх можна лінійно розділити. Крім сепарації точок, цей метод генерує гіперплощину таким чином, щоб вона була максимально віддалена від найближчої точки кожної групи.

SVM і його модифікації допомагають розв'язувати такі складні завдання машинного навчання, як сплайсинг ДНК, визначення статі людини за фотографією, виведення рекламних банерів на сайти.

SVM

Історія машин на основі опорних векторів (SVM) бере свій початок наприкінці 20-го століття, і на її розвиток вплинули дослідники в галузі машинного навчання, оптимізації та теорії статистичного навчання. Ось коротка хронологія ключових подій в історії машин на основі опорних векторів:

1. **1960-ті - 1970-ті роки**: Розробка лінійних класифікаторів:

- Основи SVM можна простежити в роботах статистиків та інформатиків, які розробляли лінійні класифікатори. У цей період досліджувалася концепція пошуку гіперплощини для розділення різних класів у просторі ознак.

2. **1989**: Вапнік та Червоненкіс представляють SVM:

- Володимир Вапнік та Олексій Червоненкіс представили концепцію машин опорних векторів у статті під назвою "Природа теорії статистичного навчання". Вони описали SVM як новий підхід до розпізнавання образів, зосереджений на пошуку оптимальної гіперплощини, яка максимально розділяє точки даних різних класів.

3. **1992**: Впровадження SVM з м'якою межею (Soft Margin SVM):

- У статті під назвою "Алгоритм навчання для класифікаторів з оптимальною межею" Бернхард Бозер, Ізабель Гійон та Володимир Вапнік розширили формулювання SVM для обробки нелінійно роздільних даних. Вони ввели ідею "м'якої межі", яка допускає деяку неправильну класифікацію в навчальних даних.

4. **1995**: Алгоритм послідовної мінімальної оптимізації (SMO):

- Джон Платт представив алгоритм послідовної мінімальної оптимізації, ефективну техніку оптимізації для навчання ШНМ. SMO значно покращив масштабованість навчання ШНМ і сприяв практичному застосуванню методу.

5. **1996**: Нелінійні ядра SVM:

- Корінна Кортес та Вапнік розширили можливості SVM для обробки нелінійних границь рішень, ввівши поняття функцій ядра. Це дозволило SVM відображати вхідні дані у вимірні простори, що дало змогу знаходити нелінійні границі рішень.

6. **2000і**: Популяризація та застосування:

- SVM набули популярності як в академічних колах, так і в промисловості завдяки своїй ефективності в різних додатках, включаючи розпізнавання зображень, класифікацію текстів і біоінформатику.

7. **2000 – 2010і**: Подальший розвиток і розширення:

- Дослідники продовжували вивчати варіації SVM, такі як багатокласові SVM, регресія опорних векторів (SVR) і використання різних функцій ядра. Концепція "хитрощів ядра" стала широко визнаною і використовуваною.

8. **Теперішній час**: Поточні дослідження та застосування:

- ШНМ продовжують залишатися активною сферою досліджень, де тривають зусилля з підвищення їхньої ефективності, масштабованості та застосовності до великих наборів даних. Незважаючи на появу нових алгоритмів машинного навчання, SVM залишаються цінним інструментом у різних сферах.

Машини опорних векторів довели свою універсальність та ефективність у вирішенні завдань класифікації та регресії, і їх продовжують застосовувати в різних галузях машинного навчання та аналізу даних.

Основні алгоритми моделей машинного навчання

6. **Метод ансамблів**

Він базується на алгоритмах машинного навчання, що генерують безліч класифікаторів і розділяють усі об'єкти з даних, які знову надходять, на основі їхнього усереднення або підсумків голосування. Спочатку метод ансамблів був окремим випадком байєсівського усереднення, але потім ускладнився й обріс додатковими алгоритмами:

- бустінг (boosting) - перетворює слабкі моделі на сильні за допомогою формування ансамблю класифікаторів (з математичного погляду це є поліпшувальним перетином);
- беггінг (bagging) - збирає ускладнені класифікатори, при цьому паралельно навчаючи базові (поліпшувальне об'єднання);
- коригування помилок вихідного кодування.

Метод ансамблів - потужніший інструмент порівняно з окремими моделями прогнозування, оскільки:

- він зводить до мінімуму вплив випадковостей, усереднюючи помилки кожного базового класифікатора;
- зменшує дисперсію, оскільки кілька різних моделей, що виходять із різних гіпотез, мають більше шансів дійти до правильного результату, ніж одна окремо взята;
- унеможливлює вихід за межі множини: якщо агрегована гіпотеза опиняється поза множиною базових гіпотез, то на етапі формування комбінованої гіпотези її розширюють за допомогою того чи того способу, і гіпотеза вже входить до неї.

7. **Алгоритми кластеризації**

Кластеризація полягає в розподілі множини об'єктів за категоріями так, щоб у кожній категорії - кластері - опинилися найбільш схожі між собою елементи.

Кластеризувати об'єкти можна за різними алгоритмами. Найчастіше використовують такі:

- на основі центру ваги трикутника;
- на базі підключення;
- скорочення розмірності;

- щільності (засновані на просторовій кластеризації);
- імовірнісні;
- машинне навчання, зокрема нейронні мережі.

Алгоритми кластеризації використовують у біології (дослідження взаємодії генів у геномі, що налічує до кількох тисяч елементів), соціології (опрацювання результатів соціологічних досліджень методом Ворда, який на виході дає кластери з мінімальною дисперсією і приблизно однакового розміру) та інформаційних технологіях.

8. **Метод головних компонент (PCA)**

Метод головних компонент, або PCA, являє собою статистичну операцію з ортогонального перетворення, що має на меті переведення спостережень за змінними, що можуть бути якось взаємопов'язані між собою, у набір головних компонент - значень, які лінійно не корельовані.

Практичні завдання, в яких застосовується PCA, - візуалізація та більшість процедур стиснення, спрощення, мінімізації даних для того, щоб полегшити процес навчання. Однак метод головних компонент не годиться для ситуацій, коли вихідні дані слабо впорядковані (тобто всі компоненти методу характеризуються високою дисперсією). Тож його застосовність визначається тим, наскільки добре вивчено й описано предметну область.

9. **Сингулярне розкладання**

У лінійній алгебрі сингулярне розкладання, або SVD, визначається як розкладання прямокутної матриці, що складається з комплексних або дійсних чисел. Так, матрицю M розмірністю [m*n] можна розкласти таким чином, що M = UΣV, де U і V будуть унітарними матрицями, а Σ - діагональною.

Одним із окремих випадків сингулярного розкладання є метод головних компонент. Найперші технології комп'ютерного зору розробляли на основі SVD і PCA, вони працювали так: спочатку обличчя (або інші патерни, які належало знайти) представляли у вигляді суми базисних компонент, потім зменшували їхню розмірність, після чого здійснювали їхнє зіставлення із зображеннями з вибірки. Сучасні алгоритми сингулярного розкладання в машинному навчанні, звісно, значно складніші та витонченіші за своїх попередників, але суть їхня загалом не змінилася.

10. **Аналіз незалежних компонент (ICA)**

Це один зі статистичних методів, який виявляє приховані чинники, що впливають на випадкові величини, сигнали тощо. ICA формує породжувальну модель для баз багатофакторних даних. Змінні в моделі містять деякі приховані змінні, причому немає жодної інформації про правила їхнього змішування. Ці приховані змінні є незалежними компонентами вибірки і вважаються негаусівськими сигналами.

На відміну від аналізу головних компонент, пов'язаного з даним методом, аналіз незалежних компонент є більш ефективним, особливо в тих випадках, коли класичні підходи виявляються безсилими. Він виявляє приховані причини явищ і завдяки цьому знайшов широке застосування в найрізноманітніших галузях - від астрономії та медицини до розпізнавання мови, автоматичного тестування та аналізу динаміки фінансових показників.

Приклади застосування в реальному житті

Приклад 1. **Діагностика захворювань**

Пацієнти в даному випадку є об'єктами, а ознаками - всі спостережувані в них симптоми, анамнез, результати аналізів, уже вжиті лікувальні заходи (фактично вся історія хвороби, формалізована і розбита на окремі критерії). Деякі ознаки - стать, наявність або відсутність головного болю, кашлю, висипу та інші - розглядаються як бінарні. Оцінка тяжкості стану (вкрай важкий, середньої тяжкості та ін.) є порядковою ознакою, а багато інших - кількісними: об'єм лікарського препарату, рівень гемоглобіну в крові, показники артеріального тиску і пульсу, вік, вага. Зібравши інформацію про стан пацієнта, що містить багато таких ознак, можна завантажити її в комп'ютер і за допомогою програми, здатної до машинного навчання, вирішити такі завдання:

- провести диференціальну діагностику (визначення виду захворювання);
- обрати найоптимальнішу стратегію лікування;
- спрогнозувати розвиток хвороби, її тривалість і результат;
- прорахувати ризик можливих ускладнень;
- виявити синдроми - набори симптомів, супутні даному захворюванню або порушенню.

Жоден лікар не здатний опрацювати весь масив інформації по кожному пацієнту миттєво, узагальнити велику кількість інших подібних історій хвороби й одразу ж видати чіткий результат. Тому машинне навчання стає для лікарів незамінним помічником.

Приклад 2. **Пошук місць залягання корисних копалин**

У ролі ознак тут виступають відомості, здобуті за допомогою геологічної розвідки: наявність на території місцевості якихось порід (і це буде ознакою бінарного типу), їхні фізичні та хімічні властивості (які розкладаються на низку кількісних і якісних ознак).

Для навчальної вибірки беруть 2 види прецедентів: райони, де точно присутні родовища корисних копалин, і райони зі схожими характеристиками, де ці копалини не були виявлені. Але видобуток рідкісних корисних копалин має свою специфіку: у багатьох випадках кількість ознак значно перевищує кількість об'єктів, і методи традиційної статистики погано підходять для таких ситуацій. Тому під час машинного навчання акцент робиться на виявлення закономірностей у вже зібраному масиві даних.

Для цього визначають невеликі та найбільш інформативні сукупності ознак, які максимально показові для відповіді на питання дослідження - є в зазначеній місцевості та чи інша копалина чи ні. Можна провести аналогію з медициною: у родовищ теж можна виявити свої синдроми. Цінність застосування машинного навчання в цій царині полягає в тому, що отримані результати не тільки мають практичний характер, а й становлять серйозний науковий інтерес для геологів і геофізиків.

Приклад 3. **Оцінка надійності та платоспроможності кандидатів на отримання кредитів**

Із цим завданням щодня стикаються всі банки, що займаються видачею кредитів. Необхідність в автоматизації цього процесу назріла давно, ще в 1960-1970-ті роки, коли в США та інших країнах почався бум кредитних карт.

Особи, які запитують у банку позику, - це об'єкти, а ось ознаки відрізнятимуться залежно від того, фізична це особа чи юридична. Ознаковий опис приватної особи, яка претендує на кредит, формується на основі даних анкети, яку вона заповнює. Потім анкета доповнюється деякими іншими відомостями про потенційного клієнта, які банк отримує за своїми каналами.

Частина з них відносяться до бінарних ознак (стать, наявність телефонного номера), інші - до порядкових (освіта, посада), більшість же є кількісними (величина позики, загальна сума заборгованостей за іншими банками, вік, кількість членів сім'ї, дохід, трудовий стаж) або номінальними (ім'я, назва фірми-роботодавця, професія, адреса).

Для машинного навчання складається вибірка, до якої входять кредитоотримувачі, чия кредитна історія відома. Усі позичальники діляться на класи, у найпростішому випадку їх 2 - "хороші" позичальники і "погані", і позитивне рішення про видачу кредиту ухвалюють тільки на користь "хороших".

Складніший алгоритм машинного навчання, званий кредитним скорингом, передбачає нарахування кожному позичальнику умовних балів за кожну ознаку, і рішення про надання кредиту залежатиме від суми набраних балів. Під час машинного навчання системи кредитного скорингу спочатку призначають певну кількість балів за кожною ознакою, а потім визначають умови видачі позики (термін, відсоткову ставку та інші параметри, які відображаються в кредитному договорі). Але існує також і інший алгоритм навчання системи - на основі прецедентів.

Етапи вирішення завдань машинного навчання

Розв'язання задач машинного навчання, як правило, складається з декількох етапів, причому конкретні етапи можуть відрізнятися залежно від характеру задачі та залучених даних. Ось загальний огляд типових етапів розв'язання задач машинного навчання:

1. **Визначте проблему**:
   - Чітко сформулюйте проблему, яку ви хочете вирішити.
   - Зрозумійте цілі та завдання рішення для машинного навчання.
   - Визначте критерії успіху і метрики для оцінки.
2. **Зберіть і підготуйте дані**:
   - Зберіть відповідні дані для навчання, перевірки та тестування.
   - Очистіть і попередньо обробіть дані для обробки відсутніх значень, викидів і невідповідностей.
   - Розділіть дані на навчальні та тестові набори.
3. **Досліджуйте та аналізуйте дані**:
   - Проведіть попередній аналіз даних (EDA), щоб отримати уявлення про дані.
   - Візуалізуйте розподіл даних, кореляції та закономірності.
   - Визначте потенційні особливості та взаємозв'язки.

4. **Інженерія особливостей**:

- Виберіть або створіть релевантні ознаки, які сприятимуть створенню прогнозної моделі.
- Перетворення та попередня обробка ознак, щоб зробити їх придатними для обраних алгоритмів.

5. **Вибір моделі**:

- Виберіть відповідний алгоритм машинного навчання на основі типу задачі (класифікація, регресія, кластеризація тощо) та характеристик даних.
- Враховуйте такі фактори, як складність моделі, інтерпретованість і масштабованість.

6. **Навчіть модель**:

- Використовуйте навчальні дані для навчання обраної моделі.
- Налаштуйте гіперпараметри для оптимізації роботи моделі.
- Відстежуйте та запобігайте надмірному пристосуванню, перевіряючи модель на окремому валідаційному наборі даних.

7. **Оцініть модель**:

- Оцініть продуктивність моделі на тестовому наборі, використовуючи відповідні метрики оцінки.
- Порівняйте результати моделі з базовими моделями або еталонами.
- Ітеративно вдосконалюйте модель або експериментуйте з різними алгоритмами, якщо це необхідно.

8. **Налаштування та оптимізація**:

- Налаштуйте гіперпараметри для покращення продуктивності моделі.
- Розгляньте такі методи, як регуляризація, щоб запобігти надмірному пристосуванню.
- Вивчіть ансамблеві методи або інші передові методи для підвищення продуктивності.

9. **Розгортання моделі**:

- Інтегруйте навчену модель у виробниче середовище.
- Впровадити моніторинг та реєстрацію даних для оцінки продуктивності моделі в реальних умовах.

10. **Моніторинг та підтримка**:

- Регулярно відстежуйте продуктивність моделі у виробничому середовищі.
- Оновлюйте модель за необхідності, щоб адаптуватися до мінливих шаблонів даних або вимог.
- Вирішуйте потенційні проблеми, такі як дрейф концепції або викривлення даних.

11. **Документуйте рішення**:

- Задокументуйте весь процес, включаючи попередню обробку даних, вибір моделі та деталі навчання.
- Надайте чітку документацію про те, як використовувати модель, інтерпретувати результати та вирішувати проблеми.

ТЕМА 5. Класичне машинне навчання

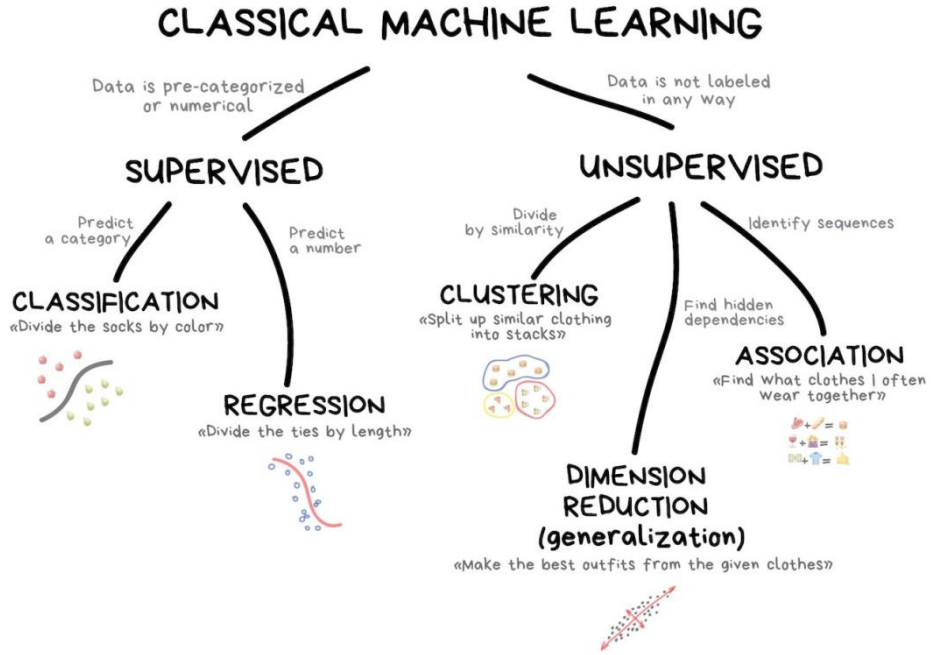
Класичне машинне навчання

Перші методи прийшли з чистої статистики в 50-х роках. Вони вирішували формальні математичні задачі - пошук закономірностей у числах, оцінка близькості точок даних, обчислення напрямків векторів.

Сьогодні над цими алгоритмами працює половина інтернету. Коли ви бачите список статей, які потрібно "прочитати далі", або ваш банк блокує вашу картку на випадковій заправці в глушині, швидше за все, це робота одного з цих маленьких хлопців.

Великі технологічні компанії є великими шанувальниками нейронних мереж. Очевидно, що так. Для них 2% точності - це додаткові 2 мільярди доларів доходу. Але коли ти маленький, це не має сенсу. Я чув історії про команди, які витратили рік на новий алгоритм рекомендацій для свого веб-сайту електронної комерції, перш ніж виявили, що 99% трафіку надходило з пошукових систем. Їхні алгоритми були марними. Більшість користувачів навіть не відкривали головну сторінку.

Незважаючи на популярність, класичні підходи настільки природні, що ви могли б легко пояснити їх малюкові. Вони схожі на елементарну арифметику - ми використовуємо її щодня, навіть не замислюючись.



Навчання під наглядом

Класичне машинне навчання часто поділяють на дві категорії - контрольоване та неконтрольоване навчання.

У першому випадку у машини є "наглядач" або "вчитель", який дає машині всі відповіді, наприклад, чи це кішка на картинці, чи собака. Вчитель вже розділив (позначив) дані на котів і собак, і машина використовує ці приклади для навчання. Один за одним. Собака за котом.

Навчання без нагляду означає, що машина залишається наодинці з купою фотографій тварин і завданням з'ясувати, хто є хто. Дані не позначені, вчителя

немає, машина намагається знайти будь-які закономірності самостійно. Про ці методи ми поговоримо нижче.

Зрозуміло, що з учителем машина навчатиметься швидше, тому її частіше використовують у реальних завданнях. Існує два типи таких завдань: класифікація - передбачення категорії об'єкта, і регресія - передбачення конкретної точки на числовій осі.

Типи задач машинного навчання

Усі завдання, розв'язувані за допомогою ML, належать до однієї з таких категорій.

1) Завдання регресії - прогноз на основі вибірки об'єктів з різними ознаками. На виході має вийти дійсне число (2, 35, 76.454 та ін.), наприклад ціна квартири, вартість цінного паперу через півроку, очікуваний дохід магазину на наступний місяць, якість вина під час сліпого тестування.

2) Завдання класифікації - отримання категоріальної відповіді на основі набору ознак. Має кінцеву кількість відповідей (як правило, у форматі "так" або "ні"): чи є на фотографії кіт, чи є зображення людським обличчям, чи хворий пацієнт на рак.

3) Завдання кластеризації - розподіл даних на групи: поділ усіх клієнтів мобільного оператора за рівнем платоспроможності, віднесення космічних об'єктів до тієї чи іншої категорії (планета, зірка, чорна діра тощо).

4) Завдання зменшення розмірності - зведення великої кількості ознак до меншої (зазвичай 2-3) для зручності їх подальшої візуалізації (наприклад, стиснення даних).

5) Завдання виявлення аномалій - відокремлення аномалій від стандартних випадків. На перший погляд воно збігається із завданням класифікації, але є одна суттєва відмінність: аномалії - явище рідкісне, і навчальних прикладів, на яких можна натаскати машинно навчальну модель на виявлення таких об'єктів, або дуже мало, або просто немає, тому методи класифікації тут не працюють. На практиці таким завданням є, наприклад, виявлення шахрайських дій з банківськими картами.

Класифікація

"Розділяє об'єкти на основі одного з атрибутів, відомих заздалегідь. Розділяє шкарпетки за кольором, документи за мовою, музику за жанром"

Сьогодні використовується для:

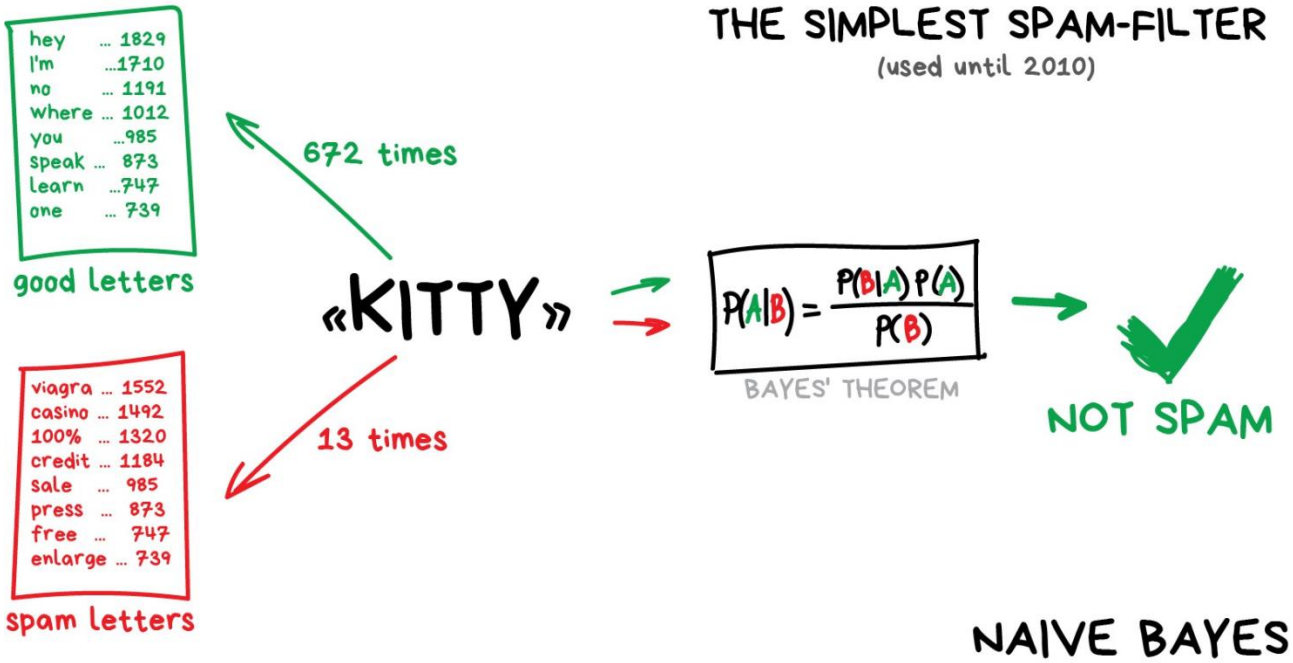
- Фільтрація спаму
- Визначення мови
- Пошук схожих документів
- Аналіз настрою
- Розпізнавання рукописних символів і цифр
- Виявлення шахрайства

Популярні алгоритми: Наївний Байєс, Дерево рішень, Логістична регресія, K-найближчих сусідів, Машина опорних векторів

Машинне навчання - це здебільшого класифікація речей. Машина тут схожа на дитину, яка вчиться сортувати іграшки: ось робот, ось машинка, ось робо-машинка... О, зачекайте. Помилка! Помилка!

У класифікації завжди потрібен вчитель. Дані повинні бути позначені ознаками, щоб машина могла призначати класи на їх основі. Класифікувати можна все: користувачів за інтересами (як це роблять алгоритмічні стрічки), статті за мовою та темою (що важливо для пошукових систем), музику за жанром (плейлисти Spotify) і навіть ваші електронні листи.

У фільтрації спаму широко використовується наївний алгоритм Байєса. Машина підраховує кількість згадок "віагри" в спамі та нормальній пошті, потім перемножує обидві ймовірності за допомогою рівняння Байєса, підсумовує результати і - ура, ми маємо машинне навчання.



Пізніше спамери навчилися обходити байєсівські фільтри, додаючи в кінці листа багато "хороших" слів. За іронією долі, цей метод отримав назву "отруєння Байєса". Наївний Байєс увійшов в історію як найелегантніший і перший практично корисний, але зараз для фільтрації спаму використовують інші алгоритми.

Ось ще один практичний приклад класифікації. Припустимо, вам потрібні гроші в кредит. Як банк дізнається, чи повернете ви їх чи ні? Немає способу дізнатися напевно. Але у банку є багато профайлів людей, які брали гроші раніше. У них є дані про вік, освіту, професію, зарплату і - найголовніше - факт повернення грошей. Або ні.

Використовуючи ці дані, ми можемо навчити машину знаходити закономірності і отримувати відповідь. З отриманням відповіді немає жодних проблем. Проблема в тому, що банк не може сліпо довіряти відповіді машини. Що, якщо станеться збій системи, хакерська атака або швидке виправлення від п'яного старшого.

Щоб впоратися з цим, у нас є Дерева рішень. Всі дані автоматично розбиваються на питання з відповідями "так" і "ні". З точки зору людини вони можуть звучати трохи дивно, наприклад, чи заробляє кредитор більше $128.12? Але машина придумує такі питання, щоб найкраще розділити дані на кожному кроці.

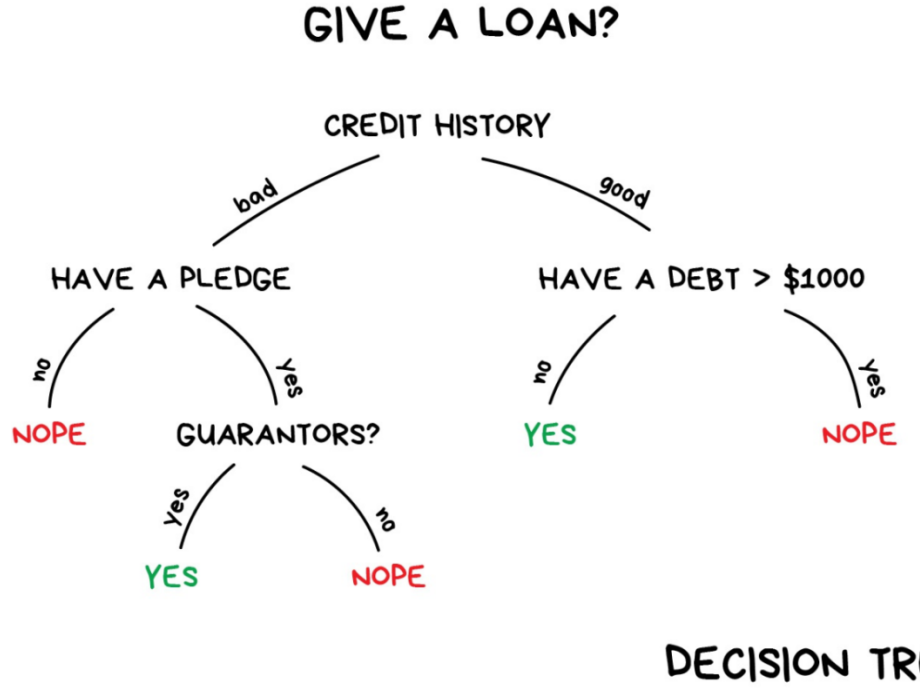
Так будується дерево. Чим вище гілка - тим ширше питання. Будь-який аналітик може взяти його і пояснити потім. Він може не зрозуміти, але пояснити легко! (типовий аналітик)

Дерева рішень широко використовуються в сферах високої відповідальності: діагностиці, медицині, фінансах.

Два найпопулярніші алгоритми формування дерев - CART і C4.5.

У чистому вигляді дерева рішень сьогодні використовуються рідко. Однак вони часто стають основою великих систем, а їхні ансамблі навіть працюють краще, ніж нейронні мережі. Про це ми поговоримо пізніше.

Коли ви щось шукаєте в Google, це саме та купка німих дерев, які шукають для вас різні варіанти відповідей. Пошукові системи люблять їх за те, що вони швидкі.



DECISION TREE

Машини опорних векторів (SVM) по праву є найпопулярнішим методом класичної класифікації. Його використовували для класифікації всього, що існує: рослин за зовнішнім виглядом на фотографіях, документів за категоріями тощо.

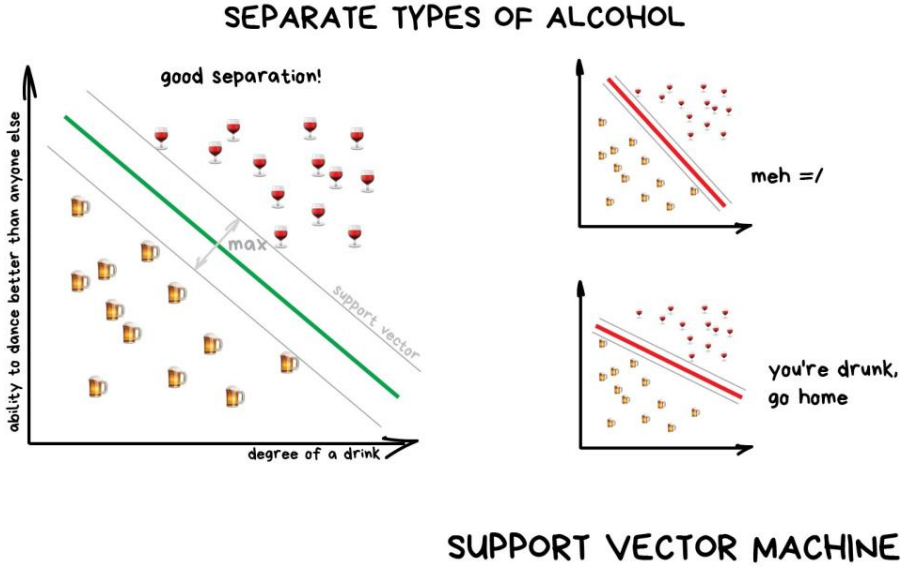
Ідея SVM проста - він намагається провести дві лінії між точками даних з найбільшою відстанню між ними. Є ще одна дуже корисна сторона класифікації - виявлення аномалій. Коли ознака не відповідає жодному з класів, ми її виділяємо. Зараз це використовується в медицині - на МРТ комп'ютери виділяють всі підозрілі ділянки або відхилення від норми. На фондових ринках це використовують для виявлення аномальної поведінки трейдерів, щоб знайти інсайдерів. Навчаючи комп'ютер правильним речам, ми автоматично вчимо його, що робити неправильно.

Сьогодні для класифікації все частіше використовують нейронні мережі. Що ж, це те, для чого вони були створені.

Емпіричне правило: чим складніші дані, тим складніший алгоритм. Для тексту, чисел і таблиць я б обрав класичний підхід. Там моделі менші, вони швидше навчаються і працюють чіткіше. Для зображень, відео та всіх інших

складних речей з великими даними я б однозначно звернув увагу на нейронні мережі.

Ще п'ять років тому можна було знайти класифікатор облич, побудований на SVM. Сьогодні простіше вибрати з сотень попередньо навчених мереж. А от для спам-фільтрів нічого не змінилося. Вони все ще пишуться за допомогою SVM. І немає жодної вагомої причини переходити з нього на інший.



Регресія

«Намалюй лінію через ці точки. Так, це і є машинне навчання»

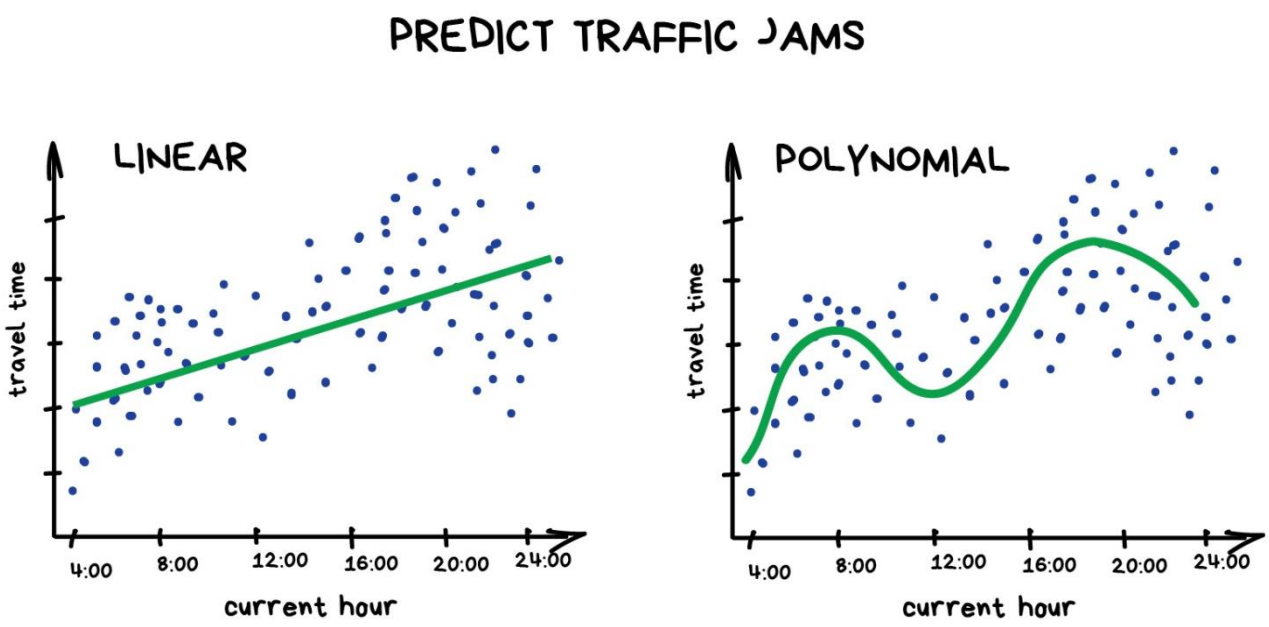
Сьогодні це використовується для:

- Прогнозування цін на акції
- Аналіз попиту та обсягів продажів
- Медична діагностика
- Будь-яких числово-часових кореляцій

Популярними алгоритмами є лінійна та поліноміальна регресії.

Регресія - це, по суті, класифікація, де ми прогнозуємо число, а не категорію. Прикладами є ціна автомобіля за його пробігом, трафік за часом доби, обсяг попиту за зростанням компанії тощо. Регресія ідеальна, коли щось залежить від часу.

Всі, хто працює з фінансами та аналізом, люблять регресію. Вона навіть вбудована в Excel. І вона дуже плавна всередині - машина просто намагається намалювати лінію, яка вказує на середню кореляцію. Хоча, на відміну від людини з ручкою і дошкою, машина робить це з математичною точністю, вираховуючи середній інтервал до кожної точки.



REGRESSION

Коли лінія пряма - це лінійна регресія, коли крива - поліноміальна. Це два основних типи регресії. Інші більш екзотичні. Логістична регресія - це біла ворона в отарі. Не дайте їй себе обдурити, оскільки це метод класифікації, а не регресії.

Однак, змішувати регресію і класифікацію – це нормально. Багато класифікаторів перетворюються на регресію після певного налаштування. Ми можемо не тільки визначити клас об'єкта, але й запам'ятати, наскільки він близький до нього. Ось вам і регресія.

Альтернативний погляд на навчання з вчителем

Скільки грошей ми заробимо, витративши більше доларів на цифрову рекламу? Чи поверне цей позичальник кредит чи ні? Що станеться з фондовим ринком завтра?

У задачах керованого навчання ми починаємо з набору даних, що містить навчальні приклади з відповідними правильними мітками. Наприклад, при навчанні класифікації рукописних цифр алгоритм навчання під контролем бере тисячі зображень рукописних цифр разом з мітками, що містять правильне число, яке представляє кожне зображення. Потім алгоритм вивчає взаємозв'язок між зображеннями та відповідними цифрами і застосовує вивчений зв'язок для класифікації абсолютно нових зображень (без міток), які машина ніколи раніше не бачила. Ось як ви можете поповнити рахунок, сфотографувавши його телефоном!

Щоб проілюструвати, як працює контрольоване навчання, розглянемо проблему прогнозування річного доходу на основі кількості років вищої освіти, яку людина здобула. Висловлюючись більш формально, ми хотіли б побудувати модель, яка апроксимує зв'язок f між кількістю років вищої освіти X та відповідним річним доходом Y.

$$Y = f(X) + \epsilon$$

X (вхід) = роки навчання у вищому навчальному закладі

Y (вихід) = річний дохід

f = функція, що описує зв'язок між X та Y

є (епсилон) = випадковий член похибки (додатний або від'ємний) із середнім значенням нуль

Щодо епсилон:

(1) є представляє незвідну похибку в моделі, яка є теоретичною межею продуктивності вашого алгоритму через шум, притаманний явищам, які ви намагаєтеся пояснити. Наприклад, уявіть, що ви будуєте модель для передбачення результату підкидання монети.

Одним із методів прогнозування доходу може бути створення жорсткої моделі, що базується на певних правилах, для визначення взаємозв'язку між доходом та освітою. Наприклад: "Я припускаю, що за кожен додатковий рік вищої освіти річний дохід збільшується на $5,000".

дохід = ($5,000 * роки_навчання) + базовий_дохід

Цей підхід є прикладом конструювання рішення (а не вивчення рішення, як у випадку з методом лінійної регресії, описаним нижче).

Ви можете розробити більш складну модель, включивши деякі правила щодо типу диплому, років досвіду роботи, рівнів освіти тощо. Наприклад: "Якщо вони отримали ступінь бакалавра або вищу освіту, дайте оцінку доходу з коефіцієнтом 1,5".

Але таке програмування на основі чітких правил погано працює зі складними даними. Уявіть, що ви намагаєтеся розробити алгоритм класифікації зображень, який складається з операторів if-then, що описують комбінації яскравості пікселів, які повинні бути позначені як "кішка" або "не кішка".

Кероване машинне навчання вирішує цю проблему, змушуючи комп'ютер виконувати роботу за вас. Виявляючи закономірності в даних, машина здатна формувати евристики. Основна відмінність між цим і людським навчанням полягає в тому, що машинне навчання працює на комп'ютерному обладнанні і найкраще розуміється через призму комп'ютерних наук і статистики, тоді як людський пошук закономірностей відбувається в біологічному мозку (при цьому досягаючи тих самих цілей).

У керованому навчанні машина намагається вивчити зв'язок між доходом і освітою з нуля, проганяючи марковані навчальні дані через алгоритм навчання. Ця вивчена функція може бути використана для оцінки доходу людей, чий дохід Y невідомий, якщо у нас є роки освіти X в якості вхідних даних. Іншими словами, ми можемо застосувати нашу модель до немаркованих тестових даних, щоб оцінити Y.

Мета керованого навчання полягає в тому, щоб якомога точніше передбачити Y на нових прикладах, де X відоме, а Y невідоме. Далі ми розглянемо кілька найпоширеніших підходів до цього.

Два завдання керованого навчання: регресія та класифікація

Регресія: передбачити безперервне числове значення. За скільки продадуть цей будинок?

Класифікація: присвоїти мітку. Це зображення кота чи собаки?

Регресія прогнозує безперервну цільову змінну Y. Вона дозволяє оцінити значення, наприклад, ціни на житло або тривалість життя людини, на основі вхідних даних X.

Тут цільова змінна означає невідому змінну, яку ми хочемо передбачити, а неперервна означає, що немає розривів (розривів) у значенні, якого може набувати Y. Вага та зріст людини є неперервними величинами. Дискретні змінні, з іншого боку, можуть набувати лише скінченної кількості значень - наприклад, кількість дітей у когось є дискретною змінною.

Прогнозування доходу є класичною регресійною задачею. Ваші вхідні дані X включають всю відповідну інформацію про осіб з набору даних, яка може бути використана для прогнозування доходу, наприклад, роки освіти, роки досвіду роботи, назва посади або поштовий індекс. Ці атрибути називаються ознаками, які можуть бути числовими (наприклад, стаж роботи) або категоріальними (наприклад, посада або галузь знань).

Вам потрібно якомога більше навчальних спостережень, що пов'язують ці ознаки з цільовим результатом Y, щоб ваша модель могла вивчити зв'язок f між X та Y.

Дані поділяються на навчальну та тестову частини. Навчальний набір містить мітки, тому ваша модель може навчатися на цих мічених прикладах. Тестовий набір не має міток, тобто ви ще не знаєте значення, яке намагаєтеся передбачити. Важливо, щоб ваша модель могла узагальнювати ситуації, з якими вона раніше не стикалася, щоб вона могла добре працювати на тестових даних.

Регресія

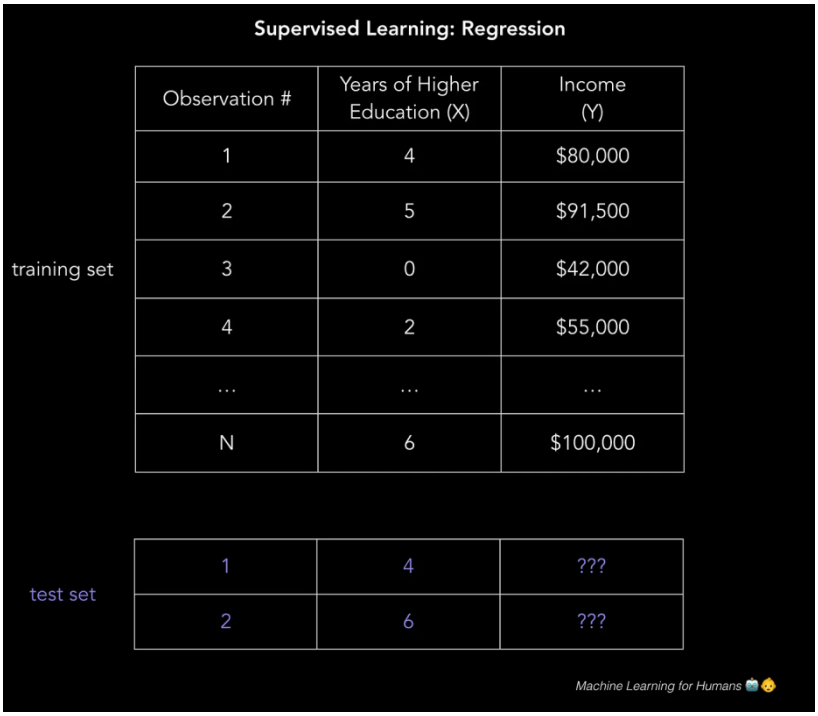
$Y = f(X) + \epsilon$, де $X = (x1, x2...xn)$

Навчання: машина дізнається f з маркованих навчальних даних

Тест: машина пророкує Y з немаркованих тестових даних

Зауважте, що X може бути тензором з будь-якою кількістю вимірів. Одномірний тензор - це вектор (1 рядок, багато стовпців), двовимірний тензор - це матриця (багато рядків, багато стовпців), а потім ви можете мати тензори з 3, 4, 5 і більше вимірами (наприклад, 3D-тензор з рядками, стовпцями і глибиною).

У нашому тривіально простому двовимірному прикладі це може бути файл .csv, де кожен рядок містить рівень освіти та дохід людини. Додайте більше стовпців з додатковими характеристиками, і ви отримаєте більш складну, але, можливо, більш точну модель.



Тож як ми вирішуємо ці проблеми?

Як ми будуємо моделі, які роблять точні, корисні прогнози в реальному світі? Ми робимо це за допомогою алгоритмів керованого навчання.

Тепер давайте перейдемо до найцікавішої частини: знайомства з алгоритмами. Ми розглянемо деякі способи підходу до регресії та класифікації, а також проілюструємо ключові концепції машинного навчання.

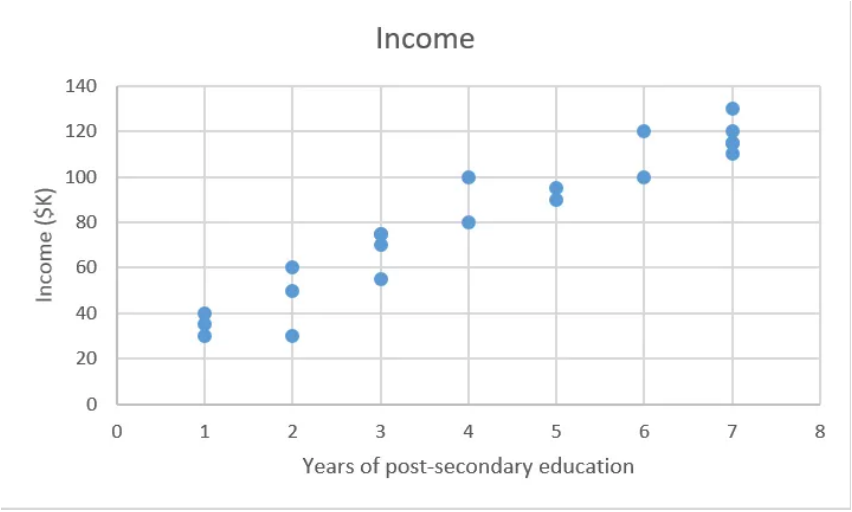
Лінійна регресія (звичайний метод найменших квадратів)

Спочатку ми зосередимося на розв'язанні задачі прогнозування доходу за допомогою лінійної регресії, оскільки лінійні моделі не дуже добре працюють із задачами розпізнавання зображень (це сфера глибокого навчання, яку ми розглянемо пізніше).

У нас є набір даних X і відповідні цільові значення Y. Метою звичайної регресії методом найменших квадратів (МНК) є вивчення лінійної моделі, яку ми можемо використовувати для прогнозування нового значення Y за раніше невідомим значенням x з якомога меншою похибкою. Ми хочемо вгадати, скільки людина заробляє на основі того, скільки років освіти вона здобула.

X_train = [4, 5, 0, 2, ..., 6] # роки післяшкільної освіти

Y_train = [80, 91.5, 42, 55, ..., 100] # відповідний річний дохід, у тисячах доларів



Лінійна регресія - це параметричний метод, тобто він робить припущення про форму функції, що зв'язує X та Y (приклади непараметричних методів ми розглянемо пізніше). Нашою моделлю буде функція, яка прогнозує ŷ при певному значенні x:

$$\hat{y} = \beta_0 + \beta_1 * x + \epsilon$$

У цьому випадку ми робимо явне припущення, що між X та Y існує лінійна залежність - тобто на кожне збільшення X на одиницю ми бачимо постійне збільшення (або зменшення) Y.

β0 - це y-інтервал, а β1 - нахил нашої лінії, тобто на скільки відсотків збільшується (або зменшується) дохід з кожним наступним роком навчання.

Наша мета полягає в тому, щоб дізнатися параметри моделі (в даному випадку β0 і β1), які мінімізують похибку в прогнозах моделі.

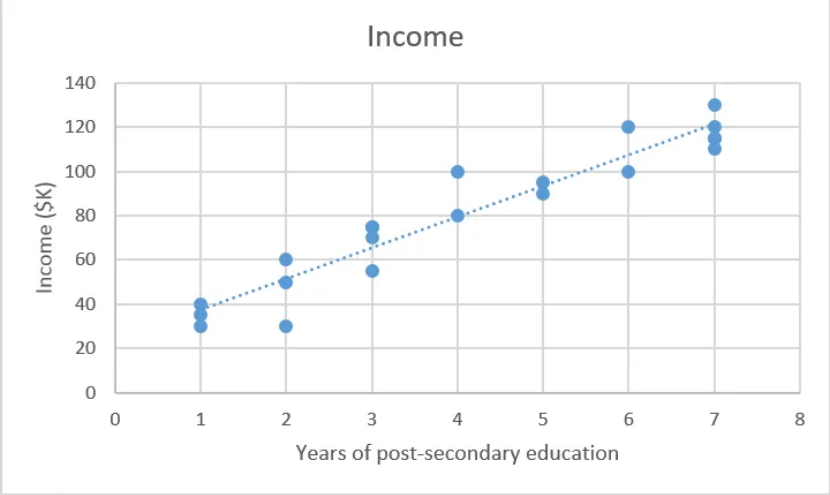
Знайти найкращі параметри:

1. Визначити функцію витрат, або функцію втрат, яка вимірює, наскільки неточними є прогнози нашої моделі.

2. Знайти параметри, які мінімізують втрати, тобто роблять нашу модель максимально точною.

Графічно, у двовимірному вимірі, це призводить до лінії найкращої пристосованості. У трьох вимірах ми б намалювали площину, і так далі з гіперплощинами вищих вимірів.

Примітка про розмірність: наш приклад двовимірний для простоти, але зазвичай у вашій моделі буде більше функцій (x) і коефіцієнтів (бета), наприклад, якщо ви додасте більше релевантних змінних, щоб підвищити точність прогнозів вашої моделі. Ці ж принципи узагальнюються для вищих вимірів, хоча візуалізувати речі за межами трьох вимірів стає набагато складніше.



Математично, ми розглядаємо різницю між кожною реальною точкою даних (y) і прогнозом нашої моделі (ŷ). Ми підносимо ці різниці до квадрату, щоб уникнути від'ємних чисел і покарати більші різниці, а потім складаємо їх і беремо середнє значення. Це міра того, наскільки добре наші дані відповідають лінії.

$$Cost = \frac{\sum_1^n ((\beta_1 x_i + \beta_0) - y_i))^2}{2 * n}$$

n = кількість спостережень. Використання 2*n замість n робить математику більш чистою, коли ми беремо похідну, щоб мінімізувати втрати, хоча деякі статистики вважають це блюзнірством.

Для такої простої задачі, як ця, ми можемо обчислити закритий розв'язок за допомогою калькулятора, щоб знайти оптимальні бета-параметри, які мінімізують нашу функцію втрат. Але зі зростанням складності функції витрат знаходження замкненого розв'язку за допомогою обчислень стає неможливим. Це є мотивацією для ітераційного підходу, який називається градієнтний спуск, що дозволяє нам мінімізувати складну функцію втрат.

Градієнтний спуск: вивчаємо параметри

"Зав'яжіть очі, зробіть крок униз. Ви знайшли дно, коли вам більше нікуди йти, окрім як вгору".

Градієнтний спуск буде з'являтися знову і знову, особливо в нейронних мережах. Бібліотеки машинного навчання, такі як scikit-learn і TensorFlow, повсюдно використовують його у фоновому режимі, тому варто розібратися в деталях.

Мета градієнтного спуску - знайти мінімум функції втрат нашої моделі, ітеративно отримуючи все краще і краще її наближення.

Уявіть, що ви йдете долиною із зав'язаними очима. Ваша мета - знайти дно долини. Як би ви це зробили?

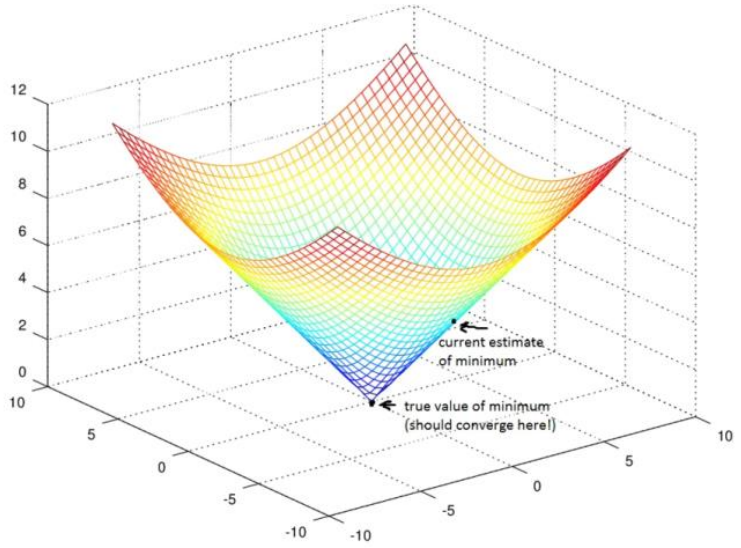
Розумним підходом було б торкнутися землі навколо себе і рухатися в тому напрямку, де земля має найкрутіший нахил донизу. Зробіть крок і повторюйте той самий процес безперервно, поки земля не стане рівною. Тоді ви знатимете, що досягли дна долини; якщо рухатися в будь-якому напрямку від того місця, де ви знаходитесь, ви опинитеся на тій самій висоті або підніметеся ще вище.

Повертаючись до математики, земля стає нашою функцією втрат, а висота на дні долини є мінімумом цієї функції.

Погляньмо на функцію втрат, яку ми бачили в регресії:

$$Cost = \frac{\sum_1^n ((\beta_1 x_i + \beta_0) - y_i))^2}{2 * n}$$

Ми бачимо, що це дійсно функція двох змінних: β0 і β1. Всі інші змінні визначені, оскільки X, Y та n задаються під час навчання. Ми хочемо спробувати мінімізувати цю функцію.



Функція має вигляд f(β0,β1)=z. Щоб почати градієнтний спуск, ви робите деяке припущення про параметри β0 і β1, які мінімізують функцію.

Далі ви знаходите частинні похідні функції втрат по кожному параметру бета: [dz/dβ0, dz/dβ1]. Часткова похідна показує, наскільки збільшиться або зменшиться загальний збиток, якщо ви збільшите β0 або β1 на дуже малу величину.

Іншими словами, наскільки збільшення вашої оцінки річного доходу за умови нульової вищої освіти (β0) збільшить втрати (тобто неточність) вашої моделі? Ви хочете рухатися в протилежному напрямку, щоб в кінцевому підсумку спуститися вниз і мінімізувати втрати.

Аналогічно, якщо ви збільшите свою оцінку того, наскільки кожен наступний рік освіти впливає на дохід (β1), наскільки це збільшить втрати (z)? Якщо часткова похідна dz/β1 є від'ємним числом, то збільшення β1 - це добре, тому що це зменшить загальні втрати. Якщо це додатне число, то потрібно зменшити β1. Якщо це число дорівнює нулю, не змінюйте β1, тому що це означає, що ви досягли оптимуму.

Продовжуйте робити це до тих пір, поки не досягнете дна, тобто алгоритм не зійдеться і втрати не будуть мінімізовані. Існує багато хитрощів і виняткових випадків, які виходять за рамки цієї серії, але загалом саме так ви знаходите оптимальні параметри для вашої параметричної моделі.

Перенавчання і недонавчання

У машинному навчанні перенавчання (overfitting) і недонавчання (underfitting) є двома поширеними проблемами, які можуть вплинути на продуктивність моделі. Ці проблеми пов'язані з тим, наскільки добре модель узагальнює нові, небачені дані.

1. **Перенавчання** (Overfitting):

- **Визначення**: Перенавчання відбувається, коли модель занадто добре засвоює навчальні дані, фіксуючи шум або випадкові коливання в даних замість основних закономірностей.

- **Ознаки**:

- Модель працює дуже добре на навчальних даних, але погано на нових, небачених даних (тестових даних).

- Модель може мати надто складні взаємозв'язки або ознаки, які є специфічними для навчальної вибірки, але погано узагальнюються.

- **Причини**:

- Занадто складна модель (висока складність моделі або занадто багато параметрів).

- Навчання на занадто великій кількості епох.

- Недостатньо даних, коли модель запам'ятовує навчальні приклади замість того, щоб вивчати загальні закономірності.

- **Пом'якшення наслідків**:

- Використовуйте методи регуляризації для покарання складних моделей.

- Зменшіть складність моделі.

- Збільште кількість навчальних даних.

- Використовуйте такі методи, як перехресна перевірка, для оцінки продуктивності моделі.

2. **Недотренованість** (Underfitting):

- **Визначення**: Недонавчання відбувається, коли модель є занадто простою і не здатна вловити основні закономірності в навчальних даних.

- **Ознаки**:

- Модель погано працює як на навчальних даних, так і на нових, ще не бачених даних.

- Вона не може вловити суттєві взаємозв'язки або особливості в даних.

- **Причини**:

- Занадто проста модель (низька складність моделі або занадто мало параметрів).

- Недостатнє навчання або нездатність навчатися протягом достатньої кількості епох.

- **Пом'якшення наслідків**:

- Підвищення складності моделі.

- Додайте до моделі більше релевантних функцій.

- Тренуйте для більшої кількості епох.

- Забезпечити наявність достатньої кількості даних для навчання.

Щоб досягти балансу між перенавчанням і недонавчанням, важливо знайти оптимальний рівень складності моделі. Це часто передбачає

експерименти та налаштування гіперпараметрів. Такі методи, як перехресна перевірка, регуляризація та моніторинг продуктивності як на навчальних, так і на валідаційних наборах даних, можуть допомогти діагностувати та вирішити ці проблеми.

Перенавчання

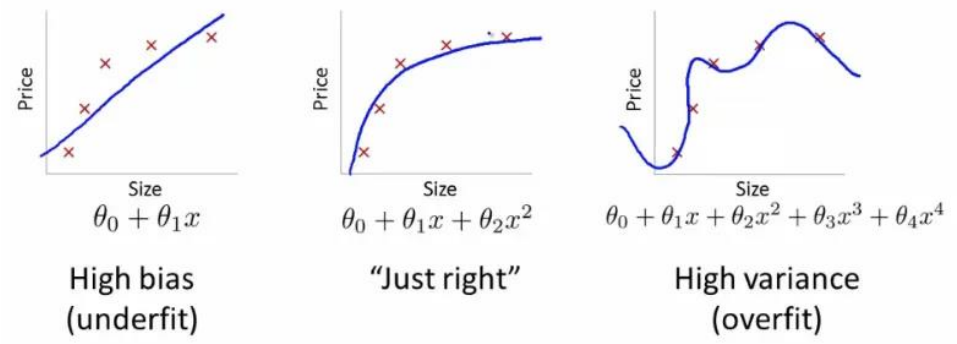
Поширеною проблемою в машинному навчанні є надмірне пристосування: навчання функції, яка чудово пояснює навчальні дані, на яких навчалася модель, але погано узагальнює невидимі тестові дані. Надмірне пристосування відбувається, коли модель перенавчається на навчальних даних до такої міри, що починає вловлювати особливості, які не є репрезентативними для моделей реального світу. Це стає особливо проблематичним, коли ви робите свою модель все більш складною. Недостатня пристосованість - це пов'язана з цим проблема, коли ваша модель недостатньо складна, щоб вловити основну тенденцію в даних.

Компроміс між похибкою (зсувом) та дисперсією

Зсув - це кількість помилок, що виникають при апроксимації реальних явищ спрощеною моделлю.

Дисперсія - це те, наскільки змінюється тестова помилка вашої моделі залежно від зміни навчальних даних. Вона відображає чутливість моделі до особливостей набору даних, на якому вона навчалася.

Зі збільшенням складності моделі та її гнучкості, зміщення зменшується (вона добре пояснює навчальні дані), але збільшується дисперсія (вона не так добре узагальнює). Зрештою, для того, щоб мати хорошу модель, вам потрібна модель з низьким зміщенням і низькою дисперсією.



Пам'ятайте, що єдине, що нас цікавить, це те, як модель працює на тестових даних. Ви хочете передбачити, які електронні листи буде позначено як спам, перш ніж їх буде позначено, а не просто побудувати модель, яка на 100% точна для перекласифікації електронних листів, які використовувалися для створення. Огляд заднім числом 20/20 — реальне питання полягає в тому, чи допоможуть отримані уроки в майбутньому.

Модель праворуч має нульові втрати для навчальних даних, оскільки вона ідеально відповідає кожній точці даних. Але урок не є узагальненим. Це зробило б жахливу роботу, пояснюючи нову точку даних, якої ще немає на лінії.

Два способи боротьби з перенавчанням:

1. Використовуйте більше навчальних даних. Чим більше у вас даних, тим важче їх перевантажити, навчившись занадто багато на одному навчальному прикладі.

2. Використовуйте регуляризацію. Додайте штраф у функцію втрат за побудову моделі, яка надає занадто велику пояснювальну силу будь-якій одній ознаці або дозволяє враховувати занадто багато ознак.

$$Cost = \frac{\sum_1^n ((\beta_1 x_i + \beta_0) - y_i)^2}{2 * n} + \lambda \sum_{i=0}^{1} \beta_i^2$$

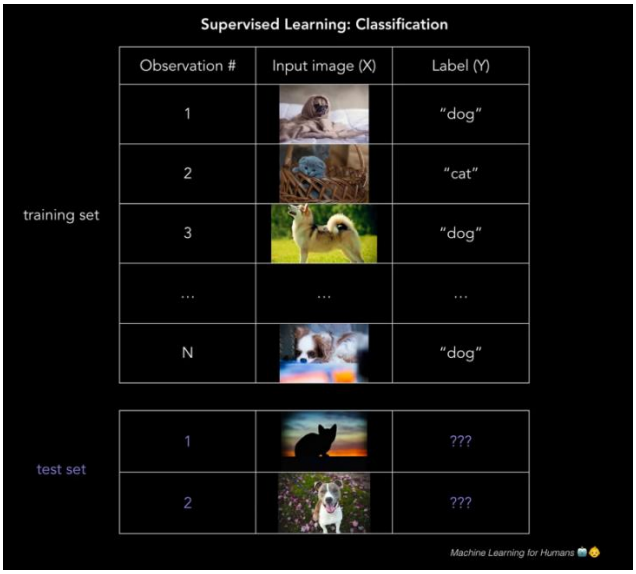
Перша частина суми вище - це наша нормальна функція витрат. Друга частина - це член регуляризації, який додає штраф за великі бета-коефіцієнти, які надають занадто велику пояснювальну силу будь-якій конкретній ознаці. З цими двома елементами функція витрат тепер балансує між двома пріоритетами: поясненням навчальних даних і запобіганням тому, щоб це пояснення не стало надто специфічним.

Лямбда-коефіцієнт члена регуляризації у функції вартості є гіперпараметром: загальним параметром вашої моделі, який можна збільшувати або зменшувати (тобто налаштовувати) для покращення продуктивності. Вище значення лямбда буде більш суворо карати великі бета-коефіцієнти, що може призвести до потенційного надмірного припасування. Щоб визначити найкраще значення лямбда, ви використовуєте метод, який називається перехресною перевіркою, що передбачає вилучення частини навчальних даних під час навчання, а потім перевірку того, наскільки добре ваша модель пояснює вилучену частину даних.

Це спам чи ні? Чи збирається цей позичальник повертати кредит? Чи будуть ці користувачі натискати на рекламу чи ні? Хто ця людина на вашому фото у Facebook?

Класифікація передбачає дискретну цільову мітку Y. Класифікація - це проблема віднесення нових спостережень до класу, до якого вони, найімовірніше, належать, на основі моделі класифікації, побудованої на основі маркованих навчальних даних.

Точність ваших класифікацій залежатиме від ефективності обраного вами алгоритму, того, як ви його застосовуєте, і від того, скільки корисних навчальних даних у вас є.



Логістична регресія: 0 чи 1?

Логістична регресія - це метод класифікації: модель виводить ймовірність того, що категоріальна цільова змінна Y належить до певного класу.

Хорошим прикладом класифікації є визначення того, чи є кредитна заявка шахрайською.

Зрештою, кредитор хоче знати, чи варто надавати позичальнику кредит, і він має певну толерантність до ризику того, що заявка насправді є шахрайською. У цьому випадку метою логістичної регресії є розрахунок ймовірності (від 0% до 100%) того, що заявка є шахрайською. Маючи ці ймовірності, ми можемо встановити певний поріг, вище якого ми готові кредитувати позичальника, а нижче якого ми відхиляємо його заявку на кредит або відкладаємо її на подальший розгляд.

Хоча логістична регресія часто використовується для бінарної класифікації, де є два класи, майте на увазі, що класифікація може бути виконана з будь-якою кількістю категорій (наприклад, при присвоєнні рукописним цифрам мітки від 0 до 9, або при використанні розпізнавання облич для визначення друзів на фотографіях у Facebook).

Чи можна просто використовувати звичайний метод найменших квадратів?

Ні, не можна. Якщо ви навчили модель лінійної регресії на купі прикладів, де Y = 0 або 1, ви можете в кінцевому підсумку передбачити деякі ймовірності, менші за 0 або більші за 1, що не має сенсу. Замість цього ми використаємо модель логістичної регресії (або логіт-модель), яка була розроблена для визначення ймовірності від 0% до 100% того, що Y належить до певного класу.

Як працює математика?

Логіт-модель - це модифікація лінійної регресії, яка гарантує виведення ймовірності між 0 і 1 шляхом застосування сигмоїдної функції, яка на графіку виглядає як характерна S-подібна крива, яку ви побачите трохи пізніше.

$$S(x) = \frac{1}{1 + e^{-x}}.$$

Сигмоїдна функція, яка стискає значення між 0 та 1.

Згадайте початкову форму нашої простої лінійної регресійної моделі, яку тепер називатимемо g(x), оскільки ми будемо використовувати її в складній функції:

$$g(X) = \beta_0 + \beta_1 x + \epsilon$$

Тепер, щоб вирішити проблему отримання результатів моделі, менших за 0 або більших за 1, ми визначимо нову функцію F(g(x)), яка перетворить g(x), роздавлюючи результат лінійної регресії до значення в діапазоні [0,1]. Чи можете ви придумати функцію, яка робить це?

Ви думаєте про сигмоїдну функцію? Ви маєте рацію.

Отже, ми підставляємо g(x) у сигмоїдну функцію вище, в результаті чого отримуємо функцію від нашої початкової функції (так, все стає мета), яка виводить ймовірність між 0 і 1:

$$P(Y = 1) = F(g(x)) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x)}}$$

Іншими словами, ми обчислюємо ймовірність того, що навчальний приклад належить до певного класу: P(Y=1).

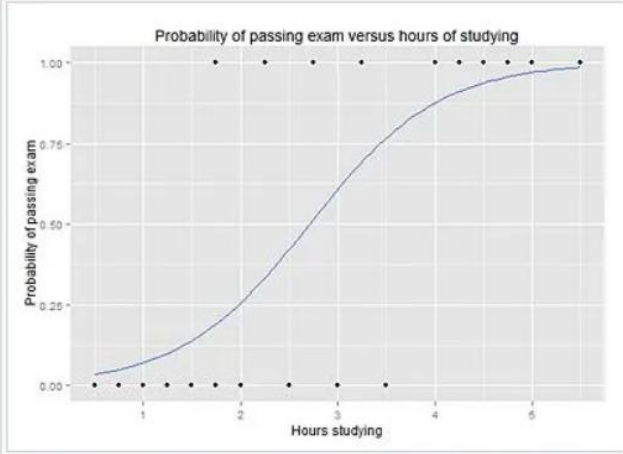
Тут ми виділили p, ймовірність того, що Y=1, у лівій частині рівняння. Якщо ми хочемо отримати чистий вигляд β0 + β1x + ϵ у правій частині, щоб можна було просто інтерпретувати бета-коефіцієнти, які ми збираємося вивчити, то замість цього ми отримаємо лог-відношення шансів, або logit, у лівій частині - звідси і назва "логіт-модель":

$$ln(\frac{p}{1-p}) = \beta_0 + \beta_1 x + \epsilon$$

Логістична регресія може бути дещо неінтуїтивно зрозумілою, але її варто зрозуміти, оскільки вона буде згадуватися знову, коли ви будете інтерпретувати результати роботи нейронних мереж, що виконують завдання класифікації.

Використання результатів логістичної регресійної моделі для прийняття рішень

Результат логістичної регресійної моделі зверху виглядає як S-подібна крива, що показує P(Y=1) на основі значення X:



Щоб передбачити мітку Y - спам/не спам, рак/не рак, шахрайство/не шахрайство і т.д. - ви повинні встановити ймовірність відсікання, або поріг, для позитивного результату. Наприклад: "Якщо наша модель вважає, що ймовірність того, що цей лист є спамом, перевищує 70%, позначити його як спам. В іншому випадку - ні".

Поріг залежить від вашої толерантності до хибнопозитивних і хибнонегативних результатів. Якщо ви діагностуєте рак, у вас буде дуже низька толерантність до хибнонегативних результатів, адже навіть якщо є дуже мала ймовірність того, що пацієнт хворий на рак, ви захочете провести додаткові тести, щоб переконатися в цьому. Тому ви встановите дуже низький поріг для позитивного результату.

З іншого боку, у випадку шахрайських кредитних заявок толерантність до хибнопозитивних результатів може бути вищою, особливо для невеликих кредитів, оскільки подальша перевірка коштує дорого, а невеликий кредит може бути не вартий додаткових операційних витрат і непорозумінь для нешахрайських заявників, які відмічені для подальшої обробки.

Мінімізація втрат за допомогою логістичної регресії

Як і у випадку з лінійною регресією, ми використовуємо градієнтний спуск, щоб дізнатися бета-параметри, які мінімізують втрати.

У логістичній регресії функція витрат - це, по суті, міра того, як часто ви прогнозували 1, коли істинною відповіддю був 0, або навпаки. Нижче наведено регуляризовану функцію вартості, подібну до тієї, яку ми розглядали для лінійної регресії.

$$Cost = \frac{\sum_1^n y^i log(h_\beta(x^i)) + (1-y^i) log(1-h_\beta(x^i))}{2*n} + \lambda \sum_{i=1}^{2} \beta_i^2$$

Не панікуйте, коли бачите таке довге рівняння! Розбийте його на частини і подумайте про те, що відбувається в кожній частині концептуально. Тоді конкретика почне набувати сенсу.

Перша частина - це втрата даних, тобто наскільки сильно розходяться прогнози моделі з реальністю. Друга частина - це втрати від регуляризації, тобто наскільки ми караємо модель за те, що вона має великі параметри, які сильно впливають на певні ознаки (пам'ятайте, що це запобігає надмірному пристосуванню).

Ми мінімізуємо цю функцію витрат за допомогою градієнтного спуску, як описано вище, і вуаля! Ми побудували логістичну регресійну модель для максимально точного прогнозування класів.

Машини опорних векторів (SVM)

"Ми знову в кімнаті, повній кульок. Чому ми завжди знаходимося в кімнаті, повній кульок? Можу поклястися, що ми їх вже загубили".

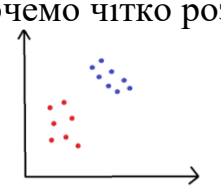
SVM - це остання параметрична модель, яку ми розглянемо. Вона зазвичай вирішує ту ж проблему, що і логістична регресія - класифікацію з двома класами - і дає схожі результати. Це варто розуміти, оскільки алгоритм є геометрично мотивованим за своєю природою, а не керується ймовірнісним мисленням.

Кілька прикладів завдань, які можуть вирішувати SVM:

- Це зображення кота чи собаки?
- Цей відгук позитивний чи негативний?
- Точки на 2D-площині червоні чи сині?

Ми використаємо третій приклад, щоб проілюструвати, як працюють SVM. Такі задачі називають іграшковими, тому що вони не є реальними - але нічого не є реальним, тому все гаразд.

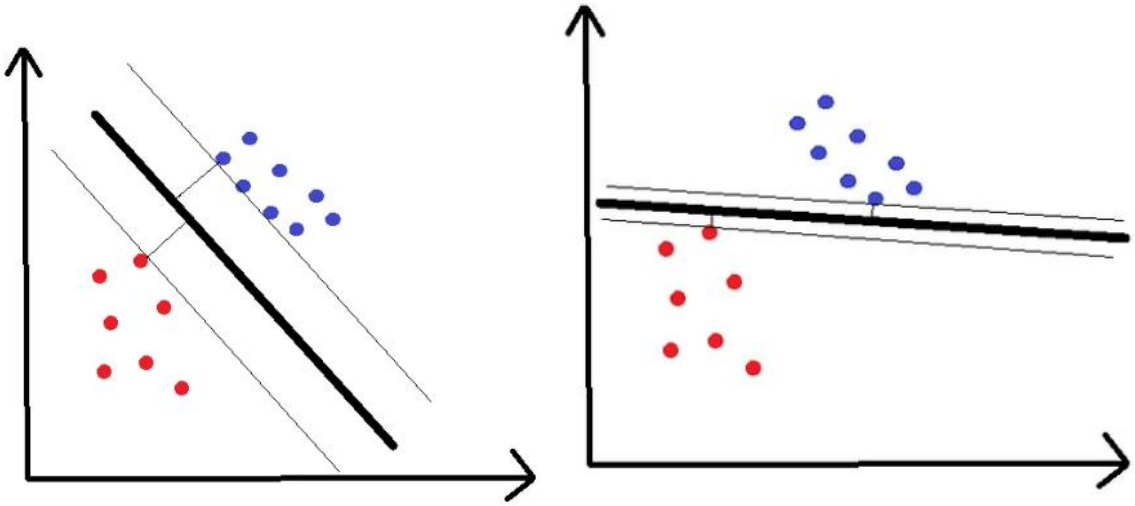
У цьому прикладі ми маємо точки у двовимірному просторі, які мають червоний або синій колір, і ми хочемо чітко розділити їх.



Навчальна вибірка зображена на графіку вище. Ми хочемо класифікувати нові, некласифіковані точки на цій площині. Для цього SVM використовують розділову лінію (або, в більш ніж двох вимірах, багатовимірну гіперплощину), щоб розділити простір на червону і синю зони. Ви вже можете уявити, як може виглядати розділова лінія на графіку вище.

Як саме ми обираємо, де провести лінію?

Нижче наведено два приклади такої лінії:



Сподіваємось, ви поділяєте інтуїцію, що перший рядок є кращим. Відстань до найближчої точки по обидва боки лінії називається запасом, і SVM намагається максимізувати цей запас. Ви можете думати про це як про простір

безпеки: чим більший цей простір, тим менша ймовірність того, що зашумлені точки будуть неправильно класифіковані.

Виходячи з цього короткого пояснення, виникає кілька важливих питань.

1. **Як працює математика, що стоїть за цим?**

Ми хочемо знайти оптимальну гіперплощину (у нашому 2D прикладі - лінію). Ця гіперплощина повинна (1) чітко розділяти дані, розміщуючи синні точки по один бік лінії, а червоні - по інший, і (2) максимізувати відступ. Це задача оптимізації. Розв'язок повинен відповідати обмеженню (1), максимізуючи при цьому маржу, як того вимагає (2).

Людська версія розв'язання цієї задачі полягає в тому, щоб взяти лінійку і продовжувати пробувати різні лінії, що розділяють всі точки, поки не отримаєте ту, яка максимізує маржу.

Гіперплощина розв'язку, яку ви отримаєте в результаті, визначається її положенням відносно певних x_i, які називаються опорними векторами, і зазвичай вони є найближчими до гіперплощини.

2. **Що станеться, якщо ви не зможете чисто розділити дані?**

Існує два способи вирішення цієї проблеми.

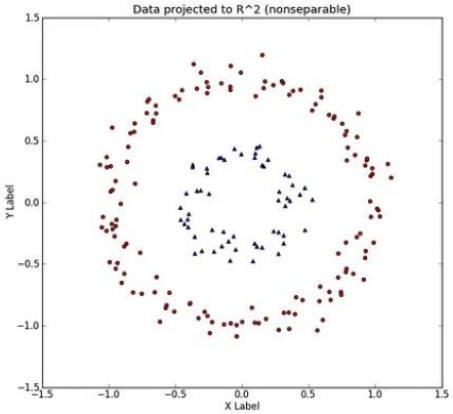
2.1. **Пом'якшити визначення "розділити"**.

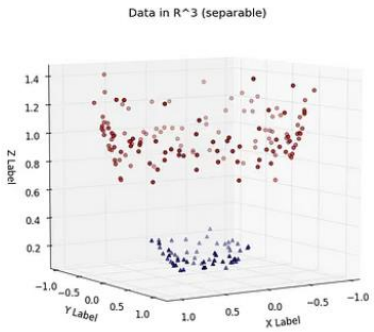
Ми допускаємо кілька помилок, тобто ми допускаємо кілька синіх точок у червоній зоні або кілька червоних точок у синій зоні. Ми робимо це, додаючи вартість C для неправильно класифікованих прикладів до нашої функції втрат. По суті, ми говоримо, що неправильно класифікувати точку допустимо, але це дорого коштує.

2.2. **Переводимо дані у вищу розмірність**.

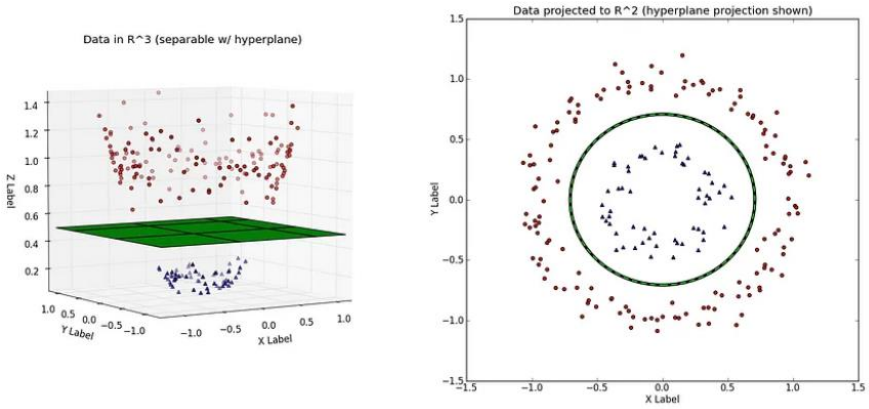
Ми можемо створювати нелінійні класифікатори, збільшуючи кількість вимірів, тобто включати x^2, x^3, навіть cos(x) тощо. Раптом у вас з'являться межі, які можуть виглядати більш звивистими, коли ми повернемо їх до представлення з меншою розмірністю.

Інтуїтивно це схоже на те, що на землі лежать червоні та сині камінці, і їх неможливо чітко розділити лінією - але якщо ви змусите всі червоні камінці відірватися від землі у правильному напрямку, ви зможете намалювати площину, яка їх розділить. Потім ви дозволите їм впасти назад на землю, знаючи, де закінчуються сині і починаються червоні.





Нероздільний набір даних у двовимірному просторі R^2 та той самий набір даних, відображений у тривимірному просторі з третім виміром x^2+y^2



Межа рішення показана зеленим кольором, спочатку в тривимірному просторі (ліворуч), а потім назад у двовимірний простір (праворуч). Те саме джерело, що й на попередньому зображенні.

Отже, SVM використовуються для класифікації з двома класами. Вони намагаються знайти площину, яка чітко розділяє два класи. Якщо це неможливо, ми або пом'якшуємо визначення "розділити", або переводимо дані у вищі виміри, щоб можна було чітко розділити дані.

Непараметричні учні.

Зараз все стане трохи... хиткішим.

На відміну від методів, які ми розглядали раніше - лінійної регресії, логістичної регресії та SVM, де форма моделі була заздалегідь визначена, - непараметричні нейронні мережі не мають структури моделі, визначеної апріорі. Ми не розмірковуємо про форму функції f, яку ми намагаємося вивчити, перед тим, як навчати модель, як ми це робили раніше з лінійною регресією. Замість цього, структура моделі визначається виключно з даних.

Ці моделі є більш гнучкими до форми навчальних даних, але іноді це відбувається ціною інтерпретованості. Незабаром це буде більш зрозуміло. Давайте почнемо.

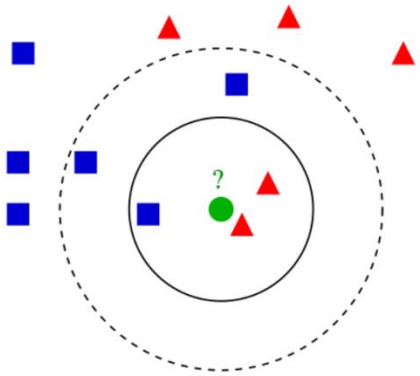
k-найближчих сусідів (k-NN)

"Ви є середнім значенням серед ваших k найближчих друзів".

k-NN здається майже занадто простим, щоб бути алгоритмом машинного навчання. Ідея полягає в тому, щоб позначити тестову точку даних x, знайшовши середнє значення (або моду) позначок k найближчих точок даних.

Погляньте на зображення нижче. Припустимо, ви хочете з'ясувати, чи є таємниче зелене коло червоним трикутником або синім квадратом. Що ви будете робити?

Можна спробувати придумати хитромудре рівняння, яке дивиться на те, де лежить Зелене Коло на координатній площині нижче, і робить відповідний прогноз. Або ж можна просто подивитися на трьох найближчих сусідів і здогадатися, що зелене коло, ймовірно, є червоним трикутником. Можна також розширити коло і подивитися на п'ять найближчих сусідів, і зробити прогноз таким чином (3/5 з п'яти найближчих сусідів - сині квадрати, тож ми можемо припустити, що загадкове зелене коло є синім квадратом, коли k=5).



k-NN ілюстрація з k=1, 3 та 5. Щоб класифікувати загадкове зелене коло (x) вище, подивіться на його єдиного найближчого сусіда - "червоний трикутник". Отже, можна припустити, що ŷ = "Червоний трикутник". При k=3, подивимось на 3 найближчих сусідів: їх мода знову "червоний трикутник", отже, ŷ = "червоний трикутник". При k=5 ми беремо моду 5 найближчих сусідів. Тепер зверніть увагу, що ŷ стає "Синім квадратом".

Це все. Це k найближчих сусідів. Ви дивитеся на k найближчих точок даних і берете середнє значення, якщо змінні неперервні (наприклад, ціни на житло), або моду, якщо вони категоричні (наприклад, кішка проти собаки).

Якщо ви хочете вгадати невідомі ціни на житло, ви можете просто взяти середнє значення деякої кількості географічно близьких будинків, і ви отримаєте досить гарні здогадки. Вони можуть навіть перевершити параметричну регресійну модель, побудовану якимось економістом, який оцінює коефіцієнти моделі для кількості ліжок/ванн, шкіл поблизу, відстані до громадського транспорту тощо.

Той факт, що k-NN не потребує заздалегідь визначеної параметричної функції f(X), яка пов'язує Y з X, робить його добре придатним для ситуацій, коли зв'язок занадто складний, щоб виразити його за допомогою простої лінійної моделі.

Як використовувати k-NN для прогнозування цін на житло:

1) Збережіть навчальні дані, матрицю X з характеристиками, такими як поштовий індекс, район, кількість спалень, квадратні фути, відстань до громадського транспорту тощо, та матрицю Y з відповідними цінами продажу.

2) Відсортуйте будинки у вашому навчальному наборі даних за схожістю з досліджуваним будинком на основі ознак у X. Нижче ми визначимо поняття "схожість".

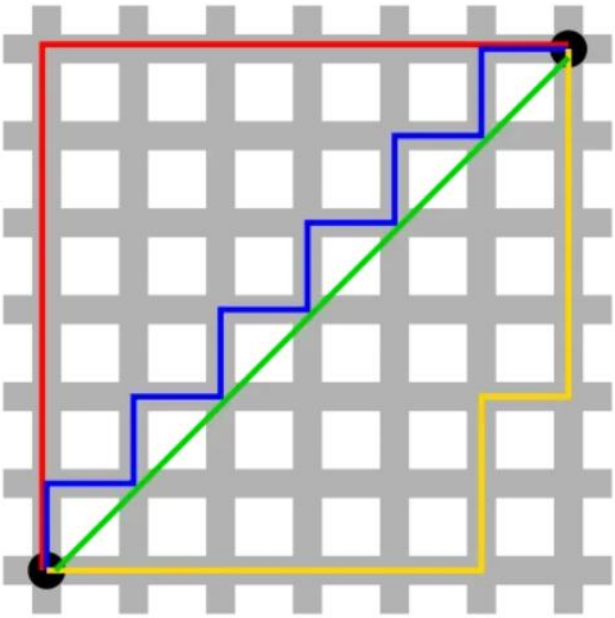
3) Візьміть середнє значення з k найближчих будинків. Це ваше припущення щодо ціни продажу (тобто ŷ)

Метрики відстані: визначення та розрахунок "близькості"

Як обчислити відстань від точки даних при пошуку "найближчих сусідів"? Як математично визначити, які з синіх квадратів і червоних трикутників у наведеному вище прикладі знаходяться найближче до зеленого кола, особливо якщо ви не можете просто намалювати гарний 2D-графік і подивитися на нього оком?

Найпростіша міра - евклідова відстань (пряма лінія, "як ворон летить"). Інша - Манхеттенська відстань, як відстань між міськими кварталами. Можна

уявити, що Манхеттенська відстань є більш корисною в моделі, що включає розрахунок вартості проїзду для водіїв Uber, наприклад.



Зелена лінія = Евклідова відстань. Синя лінія = відстань до Манхеттена

Пам'ятаєте теорему Піфагора про знаходження довжини гіпотенузи прямокутного трикутника?

$$a^2 + b^2 = c^2.$$

c - довжина гіпотенузи (зелена лінія вгорі). a і b - довжини інших сторін під прямим кутом (червоні лінії вгорі).

Розв'язуючи в термінах c, ми знаходимо довжину гіпотенузи, взявши квадратний корінь із суми квадратів довжин a і b, де a і b - ортогональні сторони трикутника (тобто вони знаходяться під кутом 90 градусів одна від одної, йдуть у перпендикулярних напрямках у просторі).

$$c = \sqrt{a^2 + b^2}.$$

Ця ідея знаходження довжини гіпотенузи за векторами у двох ортогональних напрямках узагальнюється на багато вимірів, і саме так ми виводимо формулу для евклідової відстані d(p,q) між точками p і q у n-вимірному просторі:

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Формула для евклідової відстані, що випливає з теореми Піфагора.

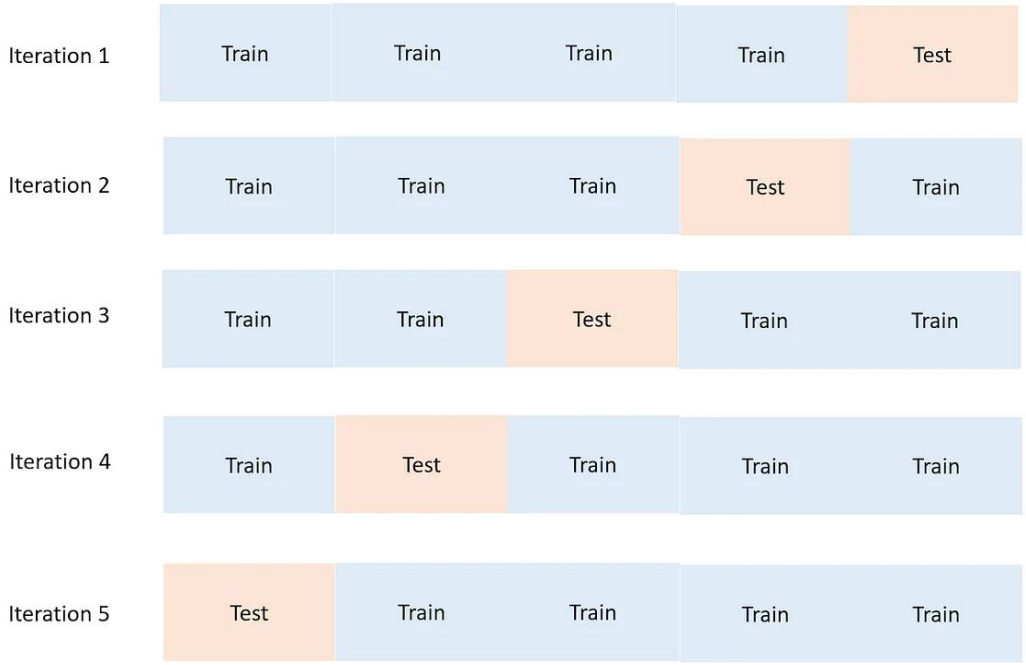
За допомогою цієї формули ви можете обчислити близькість усіх навчальних точок даних до точки даних, яку ви намагаєтеся позначити, і взяти середнє значення/моду k найближчих сусідів, щоб зробити свій прогноз.

Зазвичай вам не потрібно обчислювати метрики відстані вручну - швидкий пошук у Google покаже готові функції в NumPy або SciPy, які зроблять це за вас, наприклад, euclidean_dist = numpy.linalg.norm(p-q) - але цікаво спостерігати, як геометричні поняття з восьмого класу виявляються корисними для побудови ML-моделей сьогодні!

Вибір k: налаштування гіперпараметрів з перехресною перевіркою

Щоб вирішити, яке значення k використовувати, ви можете протестувати різні k-NN моделі з різними значеннями k з перехресною перевіркою:

1. Розділіть навчальні дані на сегменти і навчіть модель на всіх сегментах, крім одного; використовуйте залишений сегмент як "тестові" дані.
2. Подивіться, як працює ваша модель, порівнюючи прогнози (ŷ) з фактичними значеннями тестових даних (y).
3. Виберіть той варіант, який дає найменшу похибку в середньому по всіх ітераціях.



Перехресна перевірка проілюстрована. Кількість розбиття та ітерацій можна змінювати.

Вище значення k запобігає надмірному пристосуванню

Вищі значення k допомагають уникнути надмірного припасування, але якщо значення k занадто високе, ваша модель буде дуже упередженою і негнучкою. Візьмемо екстремальний приклад: якщо k = N (загальна кількість точок даних), модель просто тупо класифікує всі тестові дані як середнє значення або моду навчальних даних.

Якщо найпоширенішою твариною в наборі даних є шотландське висловухе кошеня, то k-NN, де k дорівнює N (кількість навчальних спостережень), передбачить, що кожна інша тварина у світі також є шотландським висловухим кошеням.

Де використовувати k-NN в реальному світі

Кілька прикладів, де можна використовувати k-NN:

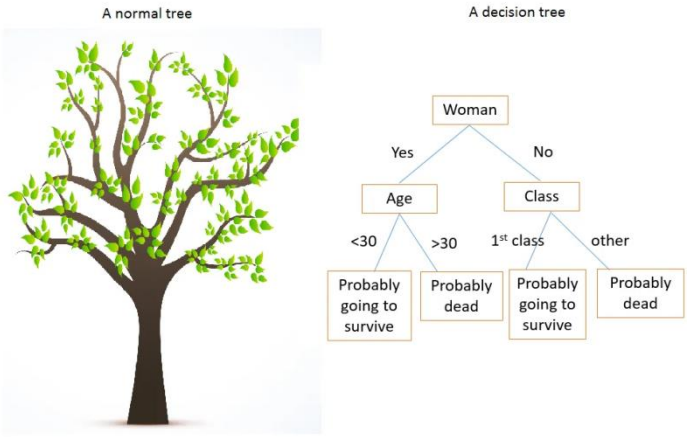
Класифікація: виявлення шахрайства. Модель може оновлюватися практично миттєво з новими навчальними прикладами, оскільки ви просто зберігаєте більше точок даних, що дозволяє швидко адаптуватися до нових методів шахрайства.

Регресія: прогнозування цін на житло. У прогнозуванні цін на житло буквально бути "найближчим сусідом" насправді є хорошим показником того, що ціни на житло будуть схожими. k-NN корисний у сферах, де фізична близькість має значення.

Імпутація відсутніх навчальних даних. Якщо в одному зі стовпців вашого .csv багато відсутніх значень, ви можете інтерполювати дані, взявши середнє значення або моду. k-NN може дати вам дещо точніший здогад для кожного відсутнього значення.

Дерева рішень, випадкові ліси

Створення хорошого дерева рішень схоже на гру "20 запитань".



Перше розділення в корені дерева рішень має бути схожим на перше запитання, яке ви маєте поставити в 20 запитаннях: ви хочете розділити дані якомога чіткіше, тим самим максимізуючи отримання інформації від цього розділення.

Якщо ваш друг каже: "Я думаю про іменник, постав мені до 20 запитань з відповідями "так" чи "ні", щоб здогадатися, що це", а ваше перше запитання - "це картопля?", то ви - дурень, тому що він скаже "ні", і ви не отримаєте майже ніякої інформації. Якщо тільки ви не знаєте, що ваш друг постійно думає про картоплю або думає про неї прямо зараз. Тоді ви добре попрацювали.

Натомість, питання на кшталт "чи є це об'єктом?" може мати більше сенсу.

Це схоже на те, як у лікарнях сортують пацієнтів або підходять до диференціальної діагностики. Спочатку ставлять кілька запитань і перевіряють основні життєві показники, щоб визначити, чи не збираєтесь ви померти, чи ні. Вони не починають робити біопсію, щоб перевірити, чи є у вас рак підшлункової залози, щойно ви переступили поріг лікарні.

Існують способи кількісної оцінки приросту інформації, так що ви можете оцінити кожне можливе розбиття навчальних даних і максимізувати

приріст інформації для кожного розбиття. Таким чином, ви зможете передбачити кожну мітку або значення якомога ефективніше.

Тепер давайте розглянемо конкретний набір даних і поговоримо про те, як ми вибираємо розбиття.

Набір даних "Титанік

Kaggle має набір даних про Титанік, який використовується для багатьох вступних занять з машинного навчання. Коли Титанік затонув, 1 502 з 2 224 пасажирів і членів екіпажу загинули. Хоча не обійшлося без везіння, жінки, діти та представники вищого класу мали більше шансів вижити. Якщо ви подивитеся на дерево рішень, наведене вище, то побачите, що воно певною мірою відображає цю варіативність залежно від статі, віку та класу.

Вибір розбиття в дереві рішень

Ентропія - це кількість невпорядкованості в множині (вимірюється індексом Джині або перехресною ентропією). Якщо значення дійсно змішані, ентропія велика; якщо ви можете чітко розділити значення, ентропія відсутня. Для кожного розділення в батьківському вузлі ви хочете, щоб дочірні вузли були якомога чистішими - мінімізувати ентропію.

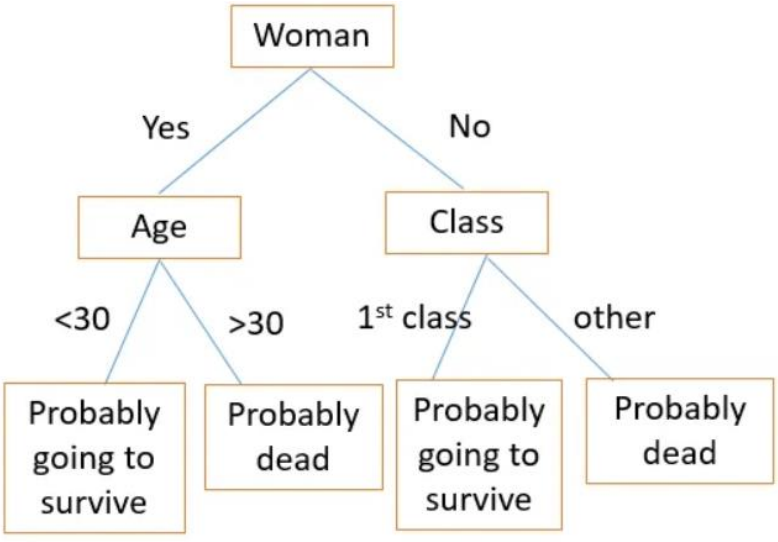
Наприклад, у "Титаніку" стать є визначальним фактором виживання, тому має сенс використовувати цю ознаку в першому розщепленні, оскільки саме вона призводить до найбільшого інформаційного приросту.

Давайте подивимось на наші змінні Титаніка:

Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Ми будуємо дерево, обираючи одну з цих змінних і розбиваючи набір даних відповідно до неї.



Перший спліт розділяє наш набір даних на чоловіків і жінок. Потім жіноча гілка знову розбивається за віком (розбиття, яке мінімізує ентропію). Аналогічно, гілка чоловіків розділяється за класом. Слідкуючи за деревом для нового пасажира, ви можете використовувати дерево, щоб зробити припущення про те, чи загинув він.

Приклад з "Титаніком" вирішує проблему класифікації ("вижити" чи "загинути"). Якби ми використовували дерева рішень для регресії - скажімо, для прогнозування цін на житло - ми б створили розбиття за найважливішими ознаками, які визначають ціни на житло. Скільки квадратних футів: більше чи менше ___? Скільки спалень і ванних кімнат: більше чи менше ___?

Потім, під час тестування, ви б пропустили конкретний будинок через усі розбиття і взяли б середнє значення всіх цін на житло у кінцевому вузлі листа (найнижчий вузол), де знаходиться будинок, як ваш прогноз щодо ціни продажу.

Існує декілька гіперпараметрів, які ви можете налаштувати у моделях дерев рішень, зокрема max_depth та max_leaf_nodes. Поради щодо визначення цих параметрів див. у модулі scikit-learn про дерева рішень.

Дерева рішень ефективні, тому що їх легко читати, вони потужні навіть для безладних даних і дешеві в обчислювальному плані для розгортання одразу після навчання. Дерева рішень також добре підходять для обробки змішаних даних (числових або категоріальних).

Проте навчання дерев рішень вимагає значних обчислювальних затрат, несе великий ризик надмірної адаптації і має тенденцію до пошуку локальних оптимумів, оскільки не може повернутися назад після того, як вони зробили розбиття. Щоб усунути ці недоліки, ми звернемося до методу, який ілюструє можливості об'єднання багатьох дерев рішень в одну модель.

Випадковий ліс: ансамбль дерев рішень

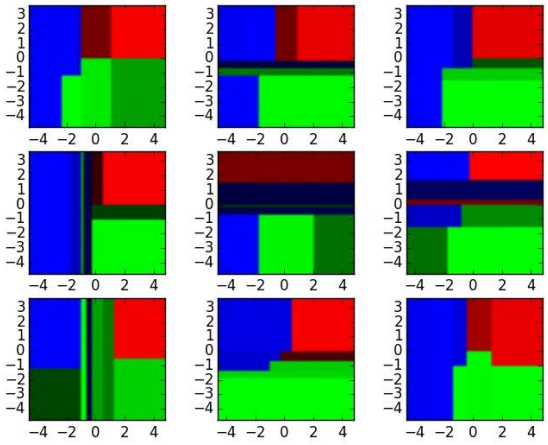
Модель, що складається з багатьох моделей, називається ансамблевою моделлю, і це, як правило, виграшна стратегія.

Одне дерево рішень може зробити багато помилкових рішень, оскільки воно має дуже чорно-білі судження. Випадковий ліс - це мета-оцінювач, який агрегує багато дерев рішень з деякими корисними модифікаціями:

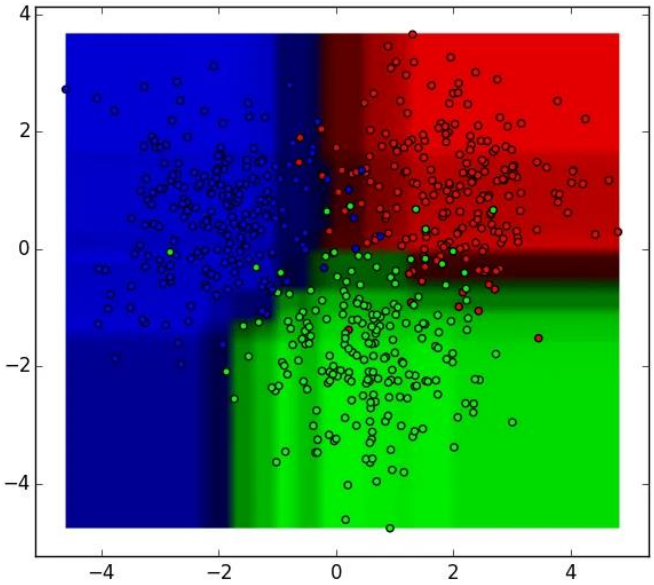
1. Кількість ознак, які можна розділити на кожному вузлі, обмежена певним відсотком від загальної кількості (це гіперпараметр, який ви можете вибрати - докладніше див. документацію до scikit-learn). Це гарантує, що ансамблева модель не покладається надто сильно на якусь окрему ознаку і справедливо використовує всі потенційно прогностичні ознаки.
2. Кожне дерево бере випадкову вибірку з вихідного набору даних під час генерації своїх розбиттів, додаючи додатковий елемент випадковості, який запобігає надмірному пристосуванню.

Ці модифікації також запобігають надмірній кореляції між деревами. Без пунктів 1 і 2 кожне дерево було б ідентичним, оскільки рекурсивне бінарне розбиття є детермінованим.

Для ілюстрації дивіться ці дев'ять класифікаторів дерев рішень нижче.



Ці класифікатори дерев рішень можна об'єднати в ансамбль випадкового лісу, який об'єднує їхні вхідні дані. Уявіть, що горизонтальна та вертикальна осі вихідних даних кожного дерева рішень - це ознаки x1 та x2. При певних значеннях кожної ознаки дерево рішень видає класифікацію "синій", "зелений", "червоний" і т.д.



Ці результати об'єднуються за допомогою модальних голосів або усереднення в єдину ансамблеву модель, яка в підсумку перевершує результати будь-якого окремого дерева рішень.

Випадкові ліси є чудовою відправною точкою для процесу моделювання, оскільки вони, як правило, мають високу продуктивність з високою толерантністю до менш очищених даних і можуть бути корисними для з'ясування того, які ознаки насправді мають значення серед багатьох ознак.

Існує багато інших розумних ансамблевих моделей, які поєднують дерева рішень і дають чудову продуктивність - подивіться на XGBoost (Extreme Gradient Boosting) як приклад.

ТЕМА 6. Навчання без вчителя та навчання з підкріпленням

Навчання без нагляду (вчителя)

Без нагляду винайшли трохи пізніше, у 90-х роках. Він використовується рідше, але іноді у нас просто немає вибору.

Марковані дані - це розкіш. Але що, якщо я хочу створити, скажімо, класифікатор автобусів? Мені що, вручну фотографувати мільйони автобусів на вулицях і маркувати кожен з них? Ні, на це піде ціле життя.

У цьому випадку є невелика надія на капіталізм. Завдяки соціальному розшаруванню у нас є мільйони дешевих робітників і сервісів на кшталт Mechanical Turk, які готові виконати ваше завдання за $0.05. І саме так у нас зазвичай все і робиться.

Або ви можете спробувати використати неконтрольоване навчання. Але я не можу пригадати жодного хорошого практичного застосування для нього. Зазвичай це корисно для дослідницького аналізу даних, але не як основний алгоритм. Спеціально навчена людина з оксфордським дипломом годує машину тонною сміття і спостерігає за нею. Чи є якісь кластери? Ні. Якісь видимі зв'язки? Ні. Тоді продовжуй. Ти ж хотів працювати в data science, так?

Як знайти основну структуру набору даних? Як його узагальнити та згрупувати найбільш корисно? Як ефективно представити дані у стислому форматі? Це цілі неконтрольованого навчання, яке називається "неконтрольованим", тому що ви починаєте з немаркованих даних (немає Y).

Приклади того, де можуть бути корисними методи неконтрольованого навчання:

- Рекламна платформа сегментує населення США на менші групи зі схожими демографічними характеристиками та купівельними звичками, щоб рекламодавці могли охопити свій цільовий ринок релевантною рекламою.
- Airbnb групує свої списки житла за районами, щоб користувачам було легше орієнтуватися в них.
- Команда з науки про дані зменшує кількість вимірів у великому наборі даних, щоб спростити моделювання та зменшити розмір файлу.

На відміну від керованого навчання, не завжди легко визначити, наскільки добре працює алгоритм неконтрольованого навчання. "Ефективність" часто є суб'єктивною і залежить від конкретної області.

Кластеризація

"Розділяє об'єкти за невідомими ознаками. Машина обирає найкращий спосіб"

Використовується сьогодні:

Для сегментації ринку (типи клієнтів, лояльність)

Для об'єднання близьких точок на карті

Для стиснення зображень

Для аналізу та маркування нових даних

Для виявлення аномальної поведінки

Популярні алгоритми: K-means_clustering, Mean-Shift, DBSCAN

Кластеризація - це класифікація без попередньо визначених класів. Це як ділити шкарпетки за кольорами, коли ви не пам'ятаєте всіх кольорів, які у вас є. Алгоритм кластеризації намагається знайти схожі (за певними ознаками) об'єкти та об'єднати їх у кластер. Ті, що мають багато схожих ознак, об'єднуються в один клас. З деякими алгоритмами ви навіть можете вказати точну кількість кластерів, які вам потрібні.

Чудовий приклад кластеризації - маркери на веб-картах. Коли ви шукаєте всі веганські ресторани поблизу, механізм кластеризації групує їх у блоки з номерами. Інакше ваш браузер завис би, намагаючись намалювати всі три мільйони веганських ресторанів у цьому хіпстерському центрі міста.

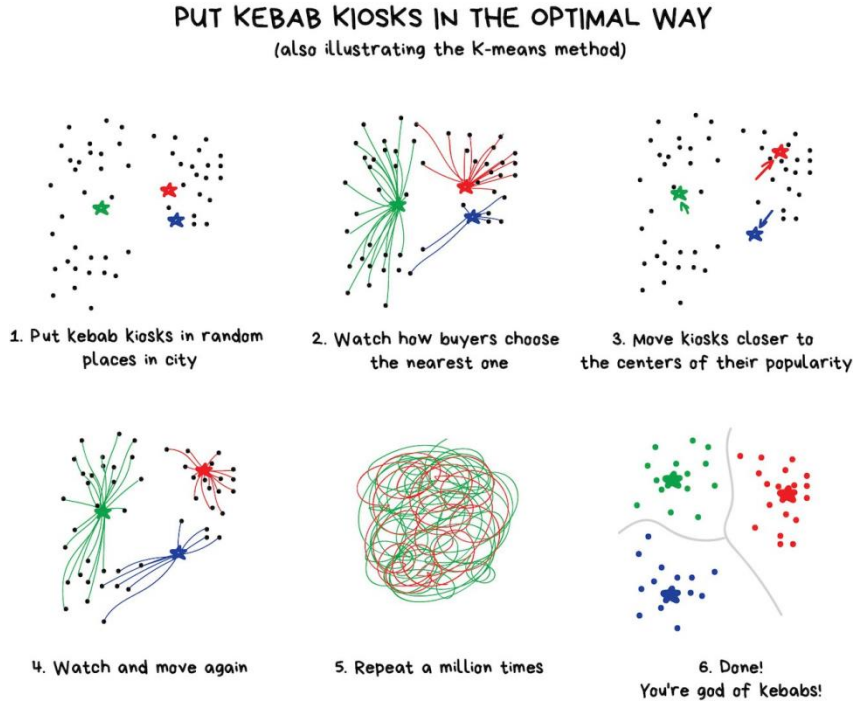
Apple Photos і Google Photos використовують складнішу кластеризацію. Вони шукають обличчя на фотографіях, щоб створити альбоми ваших друзів. Додаток не знає, скільки у вас друзів і як вони виглядають, але він намагається знайти спільні риси обличчя. Типова кластеризація.

Ще одне популярне питання - стиснення зображень. При збереженні зображення у PNG ви можете задати палітру, скажімо, 32 кольори. Це означає, що кластеризація знайде всі "червонуваті" пікселі, обчислить "середній червоний" і встановить його для всіх червоних пікселів. Менше кольорів - менший розмір файлу - вигода!

Однак, у вас можуть виникнути проблеми з такими кольорами, як Cyan-подібні кольори. Це зелений або синій? Тут на допомогу приходить алгоритм K-Means.

Він випадковим чином встановлює 32 кольорові точки в палітрі. Це центроїди. Решта точок позначаються як такі, що належать до найближчого центроїда. Таким чином, ми отримуємо своєрідні галактики навколо цих 32 кольорів. Потім ми переміщуємо центроїд до центру своєї галактики і повторюємо це доти, доки центроїди не перестануть рухатися.

Все готово. Скупчення визначені, стабільні, і їх рівно 32. Ось більш реальне пояснення:



Шукати центроїди зручно. Хоча в реальному житті кластери не завжди є колами. Уявімо, що ви геолог. І вам потрібно знайти на карті схожі мінерали. В такому випадку кластери можуть мати дивну форму і навіть бути гніздовими. Крім того, ви навіть не знаєте, скільки їх буде. 10? 100?

K-means тут не підходить, але DBSCAN може бути корисним. Скажімо, наші точки - це люди на міській площі. Знайдіть будь-яких трьох людей, що стоять близько один до одного, і попросіть їх взятися за руки. Потім попросіть їх почати хапати за руки тих сусідів, до яких вони можуть дотягнутися. І так далі, і так далі, поки ніхто більше не зможе взяти нікого за руку. Це наша перша група. Повторюйте процес, поки всі не будуть об'єднані в групи. Готово.

Просторова кластеризація додатків із шумом на основі щільності (DBSCAN) - це відомий алгоритм кластеризації даних, який широко використовується в інтелектуальному аналізі даних і машинному навчанні.

На основі набору точок (давайте мислити у двовимірному просторі, як показано на малюнку), DBSCAN групує точки, які знаходяться близько одна до одної на основі вимірювання відстані (зазвичай евклідової відстані) та мінімальної кількості точок. Він також позначає як викиди точки, що знаходяться в областях з низькою щільністю.

Параметри:

Алгоритм DBSCAN в основному вимагає 2 параметри:

eps: визначає, наскільки близько точки мають бути одна до одної, щоб вважатися частиною кластера. Це означає, що якщо відстань між двома точками менша або дорівнює цьому значенню (eps), то ці точки вважаються сусідами.

minPoints: мінімальна кількість точок для формування щільної області. Наприклад, якщо ми встановили параметр minPoints рівним 5, то для формування щільної області нам потрібно щонайменше 5 точок.

Оцінка параметрів:

Оцінка параметрів є проблемою для кожної задачі інтелектуального аналізу даних. Щоб вибрати правильні параметри, ми повинні розуміти, як вони використовуються, і мати хоча б базові попередні знання про набір даних, який буде використовуватися.

eps: якщо вибране значення eps занадто мале, велика частина даних не буде кластеризована. Вони вважатимуться викидами, оскільки не задовольняють кількості точок для створення щільної області. З іншого боку, якщо вибране значення занадто велике, кластери будуть зливатися і більшість об'єктів опиняться в одному кластері. Eps слід вибирати на основі відстані до набору даних (ми можемо використовувати графік k-відстаней, щоб знайти її), але загалом невеликі значення eps є кращими.

minPoints: Як правило, мінімальне значення minPoints може бути отримане з кількості вимірів (D) у наборі даних, оскільки minPoints ≥ D + 1. Більші значення зазвичай є кращими для наборів даних із шумом і формують більш значущі кластери. Мінімальне значення для minPoints має бути 3, але чим більший набір даних, тим більше значення minPoints слід вибирати.

Чому ми повинні використовувати DBSCAN?

Алгоритм DBSCAN слід використовувати для пошуку асоціацій і структур у даних, які важко знайти вручну, але які можуть бути релевантними і корисними для пошуку закономірностей і прогнозування тенденцій.

Методи кластеризації зазвичай використовуються в біології, медицині, соціальних науках, археології, маркетингу, розпізнаванні символів, системах управління і так далі.

Давайте розглянемо практичне застосування DBSCAN. Припустимо, ми займаємося електронною комерцією і хочемо покращити наші продажі, рекомендуючи нашим клієнтам відповідні товари. Ми не знаємо, що саме шукають наші клієнти, але на основі набору даних ми можемо передбачити і порекомендувати відповідний продукт конкретному клієнту.

Ми можемо застосувати DBSCAN до нашого набору даних (на основі бази даних електронної комерції) і знайти кластери на основі продуктів, які купували користувачі. Використовуючи ці кластери, ми можемо знайти схожість між клієнтами, наприклад, клієнт А купив 1 ручку, 1 книгу і 1 ножиці, а клієнт Б купив 1 книгу і 1 ножиці, тоді ми можемо порекомендувати 1 ручку клієнту Б. Це лише невеликий приклад використання DBSCAN, але він може бути використаний у багатьох додатках в декількох областях.

Як і класифікація, кластеризація може бути використана для виявлення аномалій. Користувач поводиться ненормально після реєстрації? Дозвольте машині тимчасово забанити його і створіть тікет для перевірки службою підтримки. Можливо, це бот. Нам навіть не потрібно знати, що таке "нормальна поведінка", ми просто завантажуємо всі дії користувача в нашу модель і дозволяємо машині вирішити, "типовий" це користувач чи ні.

Цей підхід працює не так добре, як класифікаційний, але спробувати ніколи не завадить.

Цікавим прикладом кластеризації в реальному світі є система кластеризації за життєвими стадіями Personicx від постачальника маркетингових даних Acxiom. Цей сервіс сегментує домогосподарства США на 70 окремих кластерів у межах 21 групи життєвих стадій, які використовуються рекламодавцями для таргетування реклами у Facebook, медійної реклами, прямих поштових розсилок тощо.

З їхнього технічного документа видно, що вони використовували центроїдну кластеризацію та аналіз головних компонент, обидва з яких є методами, що розглядаються в цьому розділі.

Ви можете собі уявити, наскільки доступ до цих кластерів є надзвичайно корисним для рекламодавців, які хочуть (1) зрозуміти свою існуючу клієнтську базу та (2) ефективно використовувати свої рекламні витрати, таргетуючи потенційних нових клієнтів з відповідними демографічними характеристиками, інтересами та стилем життя.

1Y STARTING OUT	**8X LARGE HOUSEHOLDS**	**15M TOP WEALTH**
Cluster 39 Setting Goals	Cluster 11 Schools & Shopping	Cluster 2 Established Elite
Cluster 45 Offices & Entertainment	Cluster 12 On the Go	Cluster 3 Corporate Connected
Cluster 57 Collegiate Crowd	Cluster 19 Country Comfort	
Cluster 58 Outdoor Fervor	Cluster 27 Tenured Proprietors	**16M LIVING WELL**
Cluster 67 First Steps		Cluster 14 Career Centered
	9B COMFORTABLE INDEPENDENCE	Cluster 15 Country Ways
2Y TAKING HOLD	Cluster 29 City Mixers	Cluster 23 Good Neighbors
Cluster 18 Climbing the Ladder	Cluster 35 Working & Active	
Cluster 21 Children First	Cluster 56 Metro Active	**17M BARGAIN HUNTERS**
Cluster 24 Career Building		Cluster 43 Work & Causes
Cluster 30 Out & About	**10B RURAL-METRO MIX**	Cluster 44 Open Houses
	Cluster 47 Rural Parents	Cluster 55 Community Life
3Y SETTLING DOWN	Cluster 53 Metro Strivers	Cluster 63 Staying Home
Cluster 34 Outward Bound	Cluster 60 Rural & Mobile	Cluster 68 Staying Healthy

Кластерізація k - means

Мета кластеризації - створити групи точок даних таким чином, щоб точки в різних кластерах були несхожими, а точки всередині кластера - схожими.

За допомогою кластеризації k-середніх ми хочемо об'єднати наші точки даних у k груп. Більше k створює менші групи з більшою деталізацією, менше k означає більші групи з меншою деталізацією.

Результатом роботи алгоритму буде набір "міток", які відносять кожну точку даних до однієї з k груп. У кластеризації за методом k-середніх ці групи визначаються шляхом створення центроїда для кожної групи. Центроїди - це як серце кластера, вони "захоплюють" найближчі до них точки і додають їх до кластера.

Уявіть собі людей, які з'являються на вечірці і незабаром стають центром уваги, тому що вони дуже привабливі. Якщо є лише одна така людина, то всі збираються навколо неї; якщо їх багато, то утворюється багато менших центрів активності.

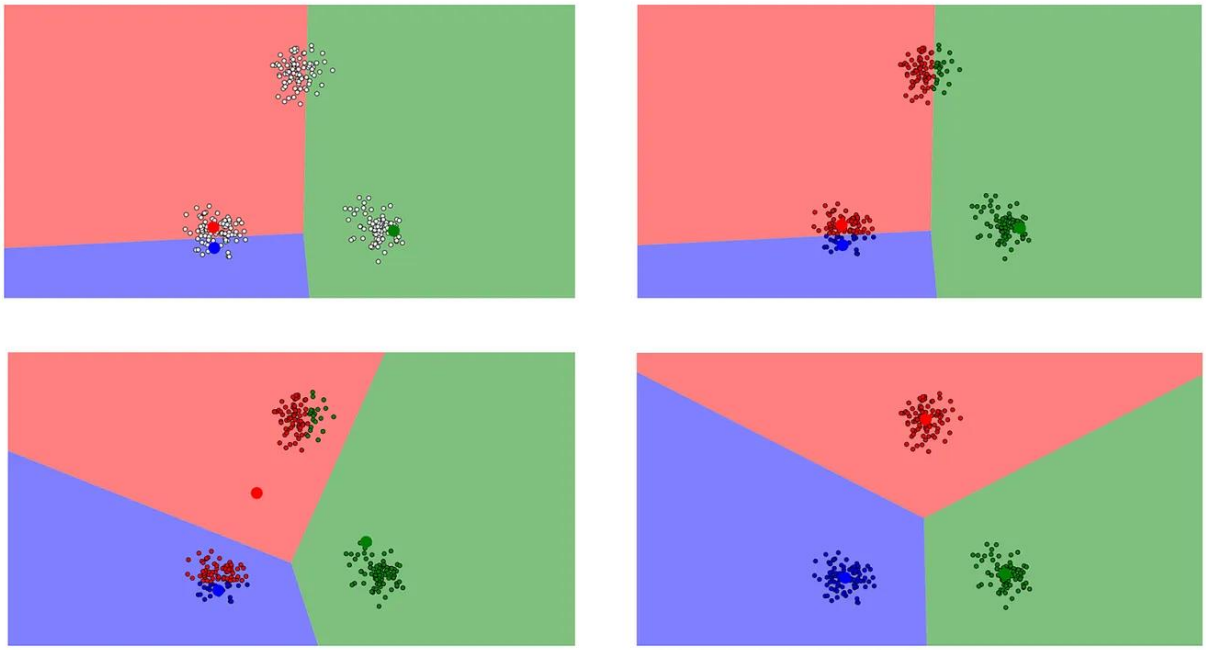
Ось кроки до кластеризації за методом k-середніх:

1. **Визначте k центроїдів**. Ініціалізуйте їх випадковим чином (існують також більш вигадливі алгоритми ініціалізації центроїдів, які в кінцевому підсумку збігаються більш ефективно).

2. **Знайдіть найближчий центроїд та оновіть призначення кластерів**. Віднесіть кожну точку даних до одного з k кластерів. Кожну точку даних віднесено до кластеру найближчого центроїда. Тут мірою "близькості" є гіперпараметр - найчастіше евклідова відстань.

3. **Перемістіть центроїди до центру їх кластерів**. Нове положення кожного центроїда обчислюється як середнє положення всіх точок у його кластері.

Повторюйте кроки 2 і 3 до тих пір, поки центроїд не перестане сильно зміщуватися на кожній ітерації (тобто, поки алгоритм не зійдеться).



Іншим реальним застосуванням кластеризації за методом k-середніх є класифікація рукописних цифр. Припустимо, що у нас є зображення цифр у вигляді довгого вектора яскравості пікселів. Припустимо, що зображення чорно-білі і мають розмір 64x64 пікселів. Кожен піксель представляє вимір. Отже, світ, у якому живуть ці зображення, має 64x64=4,096 вимірів. У цьому 4096-вимірному світі кластеризація за методом k-середніх дозволяє згрупувати зображення, які знаходяться близько одне до одного, і припустити, що вони представляють одну і ту ж цифру, що дозволяє досягти досить хороших результатів при розпізнаванні цифр.

Ієрархічна кластеризація

"Давайте зробимо так, щоб мільйон варіантів стали сімома варіантами. Або п'ятьма. Або двадцять? Вирішимо пізніше".

Ієрархічна кластеризація схожа на звичайну кластеризацію, за винятком того, що ви прагнете створити ієрархію кластерів. Це може бути корисно, коли вам потрібна гнучкість у визначенні кількості кластерів, які ви хочете отримати в кінцевому підсумку. Наприклад, уявіть собі групування товарів на онлайн-маркетплейсі, такому як Etsy або Amazon. На головній сторінці ви хочете мати кілька широких категорій товарів для простої навігації, але в міру того, як ви переходите до більш конкретних категорій покупок, ви хочете підвищити рівень деталізації, тобто більш чітких кластерів товарів.

Що стосується результатів роботи алгоритму, то на додаток до кластерних призначень ви також будуєте красиве дерево, яке розповідає вам про ієрархію між кластерами. Потім ви можете вибрати потрібну кількість кластерів з цього дерева.

Ось кроки для ієрархічної кластеризації:

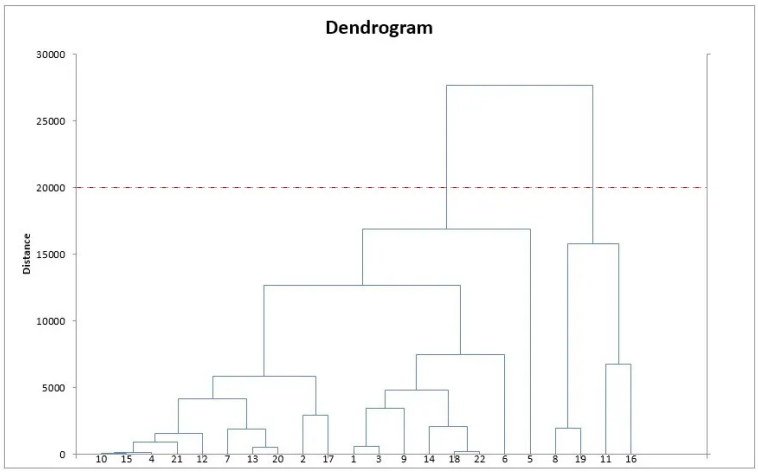
1. **Почніть з N кластерів**, по одному для кожної точки даних.

2. **Об'єднайте два кластери, які знаходяться найближче один до одного**. Тепер у вас є N-1 кластерів.

3. **Перерахуйте відстані між кластерами**. Існує кілька способів зробити це. Один з них (який називається кластеризацією за середнім зв'язком) полягає в тому, щоб вважати відстань між двома кластерами середньою відстанню між усіма їхніми членами.

4. **Повторюйте кроки 2 і 3, поки не отримаєте один кластер з N точок даних**. Ви отримаєте дерево (також відоме як дендрограма), подібне до наведеного нижче.

5. **Виберіть кількість кластерів і проведіть горизонтальну лінію на дендрограмі**. Наприклад, якщо ви хочете отримати k=2 кластери, вам слід провести горизонтальну лінію навколо "distance=20000". Ви отримаєте один кластер з точками даних 8, 9, 11, 16 і один кластер з рештою точок даних. Загалом, кількість кластерів, яку ви отримаєте, дорівнює кількості точок перетину вашої горизонтальної лінії з вертикальними лініями на дендрограмі.



Зменшення розмірності (узагальнення)

"Збирає специфічні функції в більш високорівневі"

У наш час використовується для:

- Рекомендаційні системи (★)
- Красиві візуалізації
- Тематичне моделювання та подібний пошук документів
- Аналіз підроблених зображень
- Управління ризиками

Популярні алгоритми: Аналіз головних компонент (PCA), сингулярна декомпозиція (SVD), прихований розподіл Діріхле (LDA), прихований семантичний аналіз (LSA, pLSA, GLSA), t-SNE (для візуалізації).

Раніше ці методи використовували затяті data scientists, яким потрібно було знайти "щось цікаве" у величезних купах цифр. Коли діаграми Excel не допомагали, вони змушували машини шукати закономірності. Так з'явилися методи Dimension Reduction або Feature Learning.

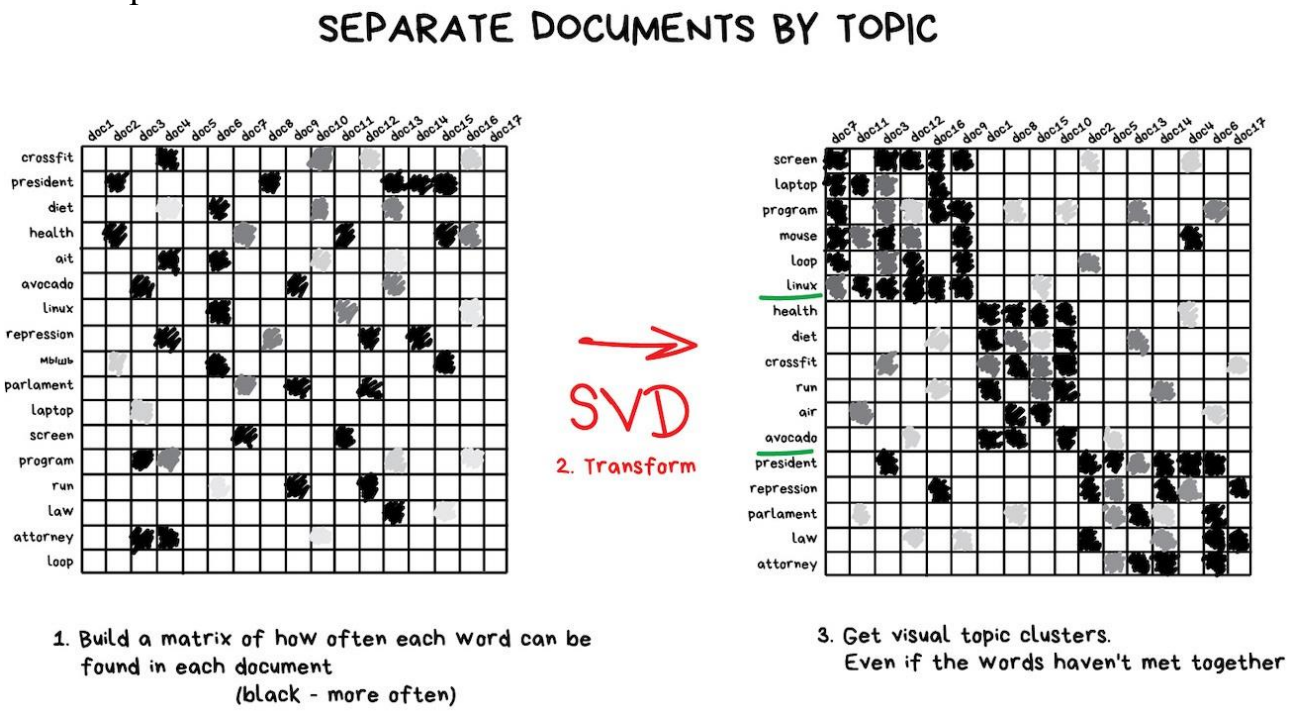
Людям завжди зручніше користуватися абстракціями, а не купою розрізнених функцій. Наприклад, ми можемо об'єднати всіх собак з трикутними вухами, довгими носами і великими хвостами в красиву абстракцію - "вівчарка". Так, ми втрачаємо частину інформації про конкретних вівчарок, але нова абстракція набагато корисніша для називання та пояснення. Як бонус, такі

"абстраговані" моделі швидше навчаються, менше перевантажуються і використовують меншу кількість функцій.

Ці алгоритми стали чудовим інструментом для тематичного моделювання. Ми можемо абстрагуватися від конкретних слів до їхніх значень. Це те, що робить латентний семантичний аналіз (LSA). Він базується на тому, як часто ви бачите слово в певній темі. Наприклад, у технічних статтях, безумовно, більше технічних термінів. Імена політиків найчастіше зустрічаються в політичних новинах тощо.

Так, ми можемо просто зробити кластери з усіх слів у статтях, але ми втратимо всі важливі зв'язки (наприклад, однакове значення слів "батарея" та "акумулятор" у різних документах). LSA впорається з цим належним чином, тому він і називається "латентною семантикою".

Отже, нам потрібно зв'язати слова і документи в одну ознаку, щоб зберегти ці приховані зв'язки - виявляється, що Singular decomposition (SVD) вирішує цю задачу, виявляючи корисні тематичні кластери зі слів, які ми бачимо разом.



LATENT SEMANTIC ANALYSIS (LSA)

Рекомендаційні системи та колаборативна фільтрація - ще одне надзвичайно популярне застосування методу зменшення розмірності. Здається, якщо ви використовуєте його для абстрагування користувацьких оцінок, ви отримаєте чудову систему для рекомендацій фільмів, музики, ігор і всього, що вам заманеться.

Навряд чи можливо повністю зрозуміти цю машинну абстракцію, але можна побачити деякі взаємозв'язки, якщо придивитися уважніше. Деякі з них пов'язані з віком користувача - діти більше грають у Minecraft і дивляться мультфільми; інші - з жанром фільму або хобі користувача.

Машини отримують ці високорівневі концепції, навіть не розуміючи їх, базуючись лише на знанні користувацьких оцінок.

"Це не щоденне збільшення, а щоденне зменшення. Відкидай несуттєве". - Брюс Лі

Зменшення розмірності дуже схоже на стиснення. Це спроба зменшити складність даних, зберігаючи при цьому якомога більше релевантної структури. Якщо взяти просте зображення розміром 128 x 128 x 3 пікселі (довжина x ширина x значення RGB), то це 49 152 виміри даних. Якщо ви зможете зменшити розмірність простору, в якому живуть ці зображення, не знищивши при цьому занадто багато змістовного контенту, то ви добре попрацювали над зменшенням розмірності.

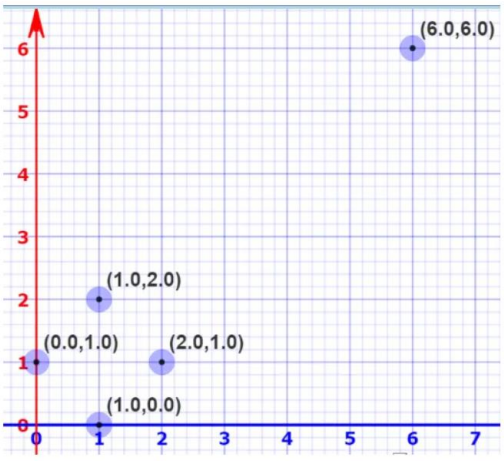
Ми розглянемо два найпоширеніші на практиці методи: аналіз головних компонент і декомпозицію сингулярних значень.

Аналіз головних компонент (PCA)

Спочатку трохи повторимо лінійну алгебру - поговоримо про простори та базиси.

Ви знайомі з координатною площиною з початком O(0,0) та базисними векторами i(1,0) і j(0,1). Виявляється, ви можете вибрати зовсім інший базис, і все одно вся математика буде працювати. Наприклад, ви можете залишити точку O за початок координат і вибрати базис векторів i'=(2,1) і j'=(1,2). Якщо у вас вистачить терпіння, ви переконаєтесь, що точка, позначена (2,2) в системі координат i', j', позначена (6, 6) в системі i, j.

Це означає, що ми можемо змінювати базис простору. Тепер уявіть собі простір набагато більшої розмірності. Наприклад, 50K вимірів. Ви можете вибрати базис для цього простору, а потім вибрати лише 200 найбільш значущих векторів цього базису. Ці вектори базису називаються головними компонентами, і вибрана вами підмножина утворює новий простір, який має меншу розмірність, ніж початковий, але зберігає якомога більшу складність даних.



Щоб вибрати найбільш значущі головні компоненти, ми дивимося, яку частину дисперсії даних вони охоплюють, і впорядковуємо їх за цією метрикою.

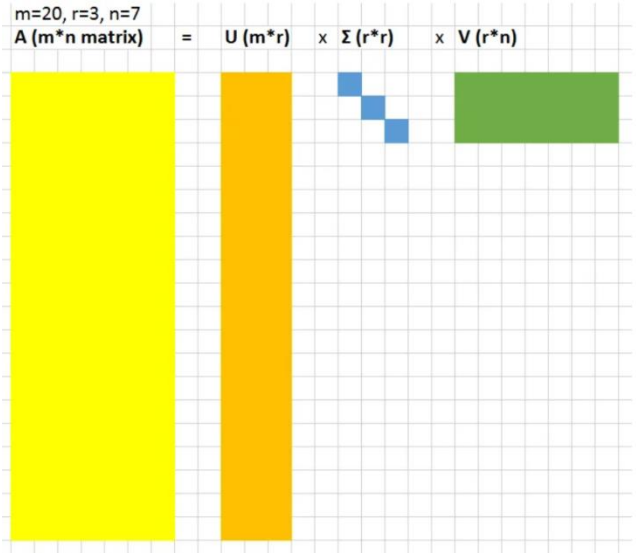
Інший спосіб мислення про це полягає в тому, що PCA відображає простір, в якому існують наші дані, щоб зробити їх більш стислими. Перетворений вимір є меншим за початковий вимір.

Використовуючи лише перші кілька вимірів переосмисленого простору, ми можемо почати розуміти організацію набору даних. Це і є обіцянка зменшення розмірності: зменшити складність (в даному випадку розмірність) при збереженні структури (дисперсії).

Розкладання за сингулярними значеннями (SVD)

Уявімо наші дані у вигляді великої матриці A = m x n. SVD - це обчислення, яке дозволяє розкласти цю велику матрицю на добуток 3 менших матриць (U=m x r, діагональна матриця Σ=r x r і V=r x n, де r - невелике число).

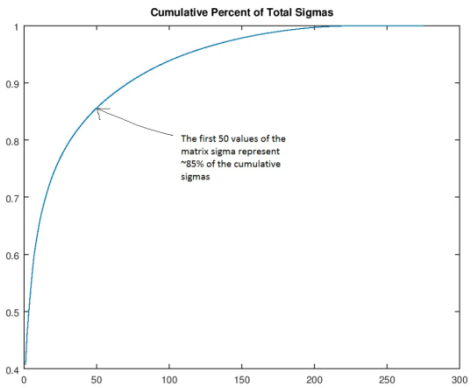
Ось більш наочна ілюстрація цього добутку для початку:



Значення у матриці Σ з діагоналлю r*r називаються сингулярними значеннями. Цікаво те, що ці сингулярні значення можна використовувати для стиснення вихідної матриці. Якщо відкинути найменші 20% сингулярних значень і відповідні стовпчики в матрицях U і V, то можна заощадити досить багато місця і при цьому отримати пристойне представлення базової матриці.

Щоб точніше зрозуміти, що це означає, давайте попрацюємо з зображенням собаки.

Ми скористаємося кодом, написаним у дописі Андрія Гіб'янського на SVD. Спочатку ми покажемо, що якщо впорядкувати сингулярні значення (значення матриці Σ) за величиною, то перші 50 сингулярних значень містять 85% величини всієї матриці Σ.



Ми можемо використати цей факт, щоб відкинути наступні 250 значень сигми (тобто встановити їх на 0) і залишити лише версію зображення собаки "рангу 50". Тут ми створимо зображення собак рангів 200, 100, 50, 30, 20, 10 і 3. Очевидно, що картинка стала меншою, але давайте погодимося, що собака 30-го рангу все ще хороша. Тепер давайте подивимось, якого стиснення ми досягнемо з цим собакою. Матриця вихідного зображення має розмір 305*275 = 83,875 значень. Собака 30-го рангу має 305*30+30+30*275=17 430 - майже в 5 разів менше значень з дуже незначною втратою якості зображення. Причиною наведених вище розрахунків є те, що ми також відкидаємо ті частини матриці U і V, які множаться на нулі при виконанні операції UΣ'V (де Σ' - це модифікована версія Σ, яка містить лише перші 30 значень).

Неконтрольоване навчання часто використовується для попередньої обробки даних. Зазвичай це означає стиснення даних зі збереженням сенсу, наприклад, за допомогою PCA або SVD, перед тим, як подати їх на глибоку нейронну мережу або інший алгоритм навчання з контролем.

Вивчення асоціативних правил

"Шукати закономірності в потоці замовлень"

Зараз використовується:

- Для прогнозування розпродажів і знижок
- Для аналізу товарів, що купуються разом
- Для розміщення товарів на полицях

- Для аналізу патернів веб-серфінгу

Популярні алгоритми: Апріорі, Евкліда, FP-зростання.

Сюди відносяться всі методи аналізу кошиків, автоматизації маркетингової стратегії та інші завдання, пов'язані з подіями. Коли у вас є послідовність чогось і ви хочете знайти в ній закономірності - спробуйте ці штуки.

Скажімо, покупець бере шість упаковок пива і йде до каси. Чи варто покласти арахіс по дорозі? Як часто люди купують їх разом? Так, це, ймовірно, працює для пива і арахісу, але які інші послідовності ми можемо передбачити? Чи може невелика зміна в розташуванні товарів призвести до значного збільшення прибутку?

Те ж саме стосується і електронної комерції. Там завдання ще цікавіше - що покупець купить наступного разу?

Незрозуміло, чому навчання на основі правил здається найменш розробленою категорією машинного навчання. Класичні методи ґрунтуються на суцільному перегляді всіх куплених товарів за допомогою дерев або множин. Алгоритми можуть лише шукати закономірності, але не можуть узагальнювати чи відтворювати їх на нових прикладах.

У реальному світі кожен великий ритейлер будує власне пропрієтарне рішення, тож ніяких революцій тут для вас не буде. Найвищий рівень технологій тут - рекомендаційні системи.

Навчання з підкріпленням

"Киньте робота в лабіринт і нехай він знайде вихід"

У наш час використовується для:

- Безпілотні автомобілі
- Роботи-пилососи
- Ігри
- Автоматизація торгівлі
- Управління ресурсами підприємства

Популярні алгоритми: Q-Learning, SARSA, DQN, A3C, Генетичний алгоритм

Нарешті, ми дійшли до чогось схожого на справжній штучний інтелект. У багатьох статтях навчання з підкріпленням розміщують десь посередині між контрольованим і неконтрольованим навчанням. У них немає нічого спільного! Навчання з підкріпленням використовується у випадках, коли ваша проблема взагалі не пов'язана з даними, але у вас є середовище, в якому ви живете. Наприклад, світ відеоігор або місто для безпілотного автомобіля.

Знання всіх правил дорожнього руху в світі не навчить автопілот правильно їздити по дорогах. Незалежно від того, скільки даних ми збираємо, ми все одно не можемо передбачити всі можливі ситуації. Саме тому його мета - мінімізувати помилки, а не передбачити всі ходи.

Виживання в навколишньому середовищі - основна ідея навчання з підкріпленням. Запустіть бідолашного маленького робота в реальне життя, карайте його за помилки і винагороджуйте за правильні вчинки. Так само, як ми вчимо наших дітей, чи не так?

Ефективніший спосіб - побудувати віртуальне місто і дозволити безпілотному автомобілю спочатку навчитися там усім його трюкам. Саме так ми зараз навчаємо автопілотів. Створіть віртуальне місто на основі реальної карти, заселіть його пішоходами і дайте машині навчитися вбивати якомога менше людей. Коли робот достатньо впевнено почуватиметься в цій штучній GTA, його можна буде випускати на реальні вулиці. Весело!

Існує два різних підходи - на основі моделі та без моделі.

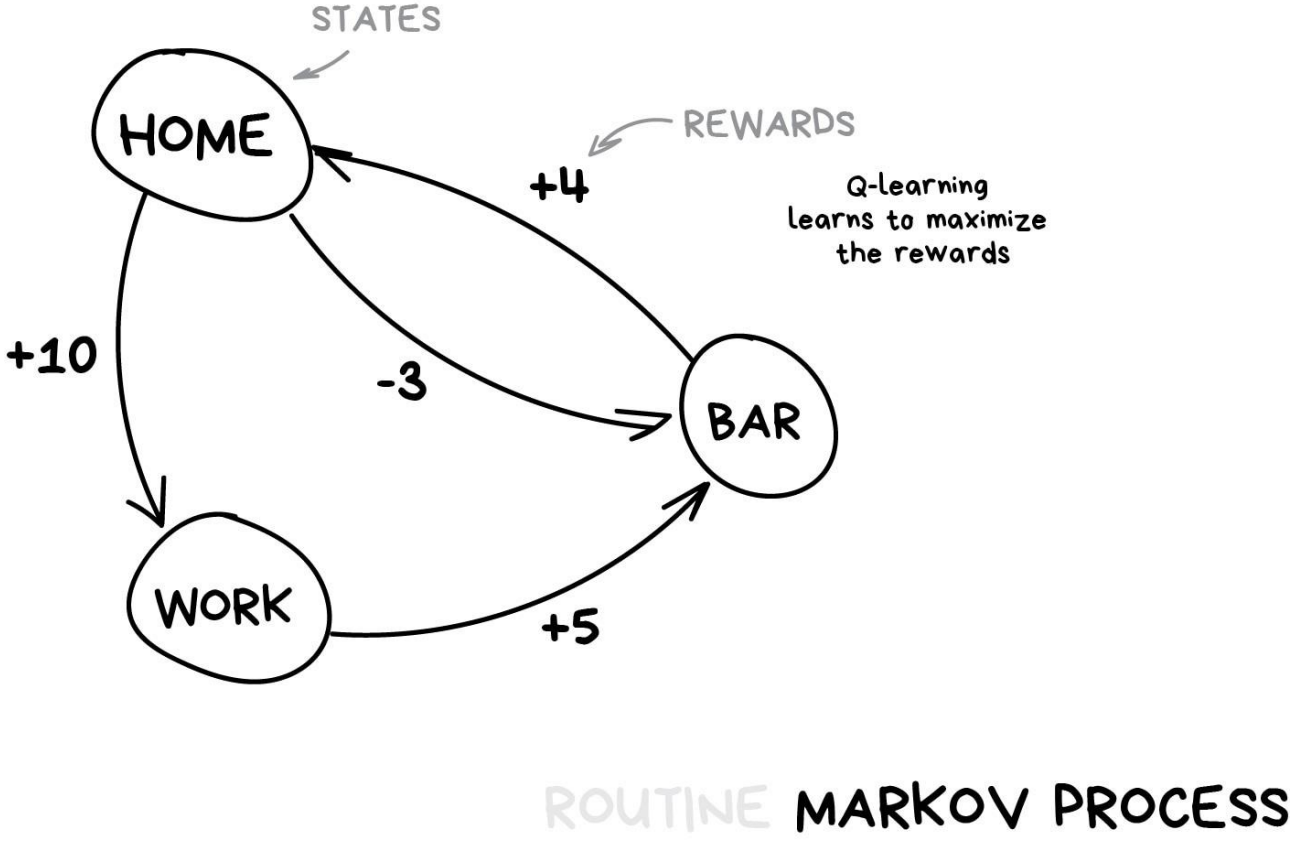
Model-Based означає, що машина повинна запам'ятовувати карту або її частини. Це досить застарілий підхід, оскільки для бідного самокерованого автомобіля неможливо запам'ятати всю планету.

При безмодельному навчанні автомобіль не запам'ятовує кожен рух, а намагається узагальнити ситуацію і діяти раціонально, отримуючи при цьому максимальну винагороду.

Пам'ятаєте новину про те, що штучний інтелект переміг найкращого гравця в гру Го? Хоча незадовго до цього було доведено, що кількість комбінацій у цій грі більша, ніж кількість атомів у Всесвіті.

Це означає, що машина не могла запам'ятати всі комбінації і таким чином виграти в го (як у шахах). На кожному кроці вона просто обирала найкращий хід для кожної ситуації, і їй це вдавалося досить добре, щоб обіграти людину.

Цей підхід є основною концепцією Q-навчання та його похідних (SARSA та DQN). "Q" в назві означає "Якість", оскільки робот вчиться виконувати найбільш "якісну" дію в кожній ситуації, а всі ситуації запам'ятовуються як простий марковський процес.



Така машина може тестувати мільярди ситуацій у віртуальному середовищі, запам'ятовуючи, які рішення призвели до більшої винагороди. Але як вона може відрізнити раніше бачені ситуації від абсолютно нової? Якщо безпілотний автомобіль стоїть на перехресті і світлофор загоряється зеленим - чи означає це, що він може їхати? А якщо по сусідній вулиці проїжджає карета швидкої допомоги?

Відповідь сьогодні - "ніхто не знає". Простої відповіді не існує. Дослідники постійно її шукають, але поки що знаходять лише обхідні шляхи. Одні кодують усі ситуації вручну, щоб вирішувати виняткові випадки, як-от проблема з тролейбусом. Інші заглиблюються вглиб і дозволяють нейронним мережам вирішувати цю проблему. Це привело нас до еволюції Q-навчання під назвою Deep Q-Network (DQN). Але вони також не є срібною кулею.

Навчання з підкріпленням для пересічної людини виглядало б як справжній штучний інтелект. Тому що воно змушує вас думати: "Вау, ця машина приймає рішення в реальних життєвих ситуаціях!". Ця тема зараз дуже популярна, вона розвивається з неймовірною швидкістю і перетинається з нейромережею, щоб точніше мити підлогу. Дивовижний світ технологій!

Не по темі. Колись дуже популярними були генетичні алгоритми. Це коли ми кидаємо купу роботів в одне середовище і змушуємо їх намагатися досягти мети, поки вони не помруть. Потім ми обираємо найкращих, схрещуємо їх, мутуємо деякі гени і повторно запускаємо симуляцію. Через кілька мільярдів років ми отримаємо розумну істоту. Напевно. Еволюція в найкращому вигляді.

Генетичні алгоритми розглядаються як частина навчання з підкріпленням, і вони мають найважливішу особливість, доведену десятиліттями практики: нікому до них немає діла.

Людство досі не змогло придумати завдання, де вони були б ефективнішими за інші методи.

У керованому навчанні навчальні дані надходять разом із ключем-відповіддю від якогось богоподібного "наглядача". Якби тільки життя працювало так само!

У навчанні з підкріпленням (RL) немає ключа-відповіді, але ваш агент навчання з підкріпленням все одно повинен вирішити, як діяти, щоб виконати своє завдання. За відсутності навчальних даних агент навчається на власному досвіді. Він збирає навчальні приклади ("ця дія була хорошою, ця дія була поганою") шляхом спроб і помилок, коли намагається виконати своє завдання, з метою максимізації довгострокової винагороди.

Розглянемо:

- Компроміс між дослідженням та експлуатацією
- Марковські процеси прийняття рішень (MDP), класичне середовище для завдань RL
- Q-навчання, політичне навчання та навчання з глибоким підкріпленням
- і, нарешті, проблему навчання на основі цінності

Найпростіший контекст, в якому можна подумати про навчання з підкріпленням, - це ігри з чіткою метою та системою балів.

Скажімо, ми граємо в гру, де наша мишка 🐭 шукає найвищу нагороду у вигляді сиру в кінці лабіринту (□ +1000 балів) або меншу нагороду у вигляді води по дорозі (●+10 балів). Тим часом, робо-миша намагається уникати місць, де її може вдарити струмом (⚡ -100 балів).

Трохи поблукавши лабіринтом, миша може знайти міні-рай з трьома джерелами води, скупченими біля входу, і витратити весь свій час на те, щоб скористатися цією знахідкою, постійно збираючи маленькі нагороди з цих джерел і ніколи не йдучи далі в лабіринт у пошуках більшого призу.

Але, як бачите, миша може пропустити ще кращий оазис далі в лабіринті, або ж головну нагороду - сир в кінці лабіринту!

Це підводить нас до компромісу між дослідженням та експлуатацією. Одна з простих стратегій дослідження полягає в тому, що миша більшу частину часу (скажімо, 80% часу) виконує найвідомішу дію, але час від часу досліджує новий, випадково вибраний напрямок, навіть якщо це може відвести її від відомої винагороди.

Ця стратегія називається стратегією епсилон-жадібності, де епсилон - це відсоток часу, протягом якого агент виконує випадково обрану дію замість того, щоб виконати дію, яка, найімовірніше, максимізує винагороду, враховуючи те, що він знає на даний момент (в даному випадку 20%). Зазвичай ми починаємо з великої кількості досліджень (тобто з більшого значення епсилон). З часом, коли миша дізнається більше про лабіринт і про те, які дії приносять найбільшу довгострокову винагороду, має сенс поступово зменшувати епсилон до 10% або навіть нижче, коли вона навчиться використовувати те, що знає.

Важливо пам'ятати, що винагорода не завжди приходить миттєво: у прикладі з роботом-мишею може бути довга ділянка лабіринту, яку потрібно пройти, і кілька точок прийняття рішення, перш ніж ви дійдете до сиру.

Марковські процеси прийняття рішень (MDP)

Блукання миші в лабіринті можна формалізувати як Марковський процес прийняття рішень, тобто процес, який має визначені ймовірності переходу зі стану в стан. Ми пояснимо це на прикладі нашого робота-миші. MDP включає в себе

- **Скінченну множину станів**. Це можливі положення нашої миші в лабіринті.
- **Набір дій, доступних у кожному стані**. Це {вперед, назад} в коридорі та {вперед, назад, ліворуч, праворуч} на перехресті.
- **Переходи між станами**. Наприклад, якщо на перехресті ви повернете ліворуч, то опинитеся в новій позиції. Це може бути набір ймовірностей, які пов'язані з більш ніж одним можливим станом (наприклад, коли ви використовуєте атаку в грі покемонів, ви можете або промахнутися, або завдати деякої шкоди, або завдати достатньої шкоди, щоб нокаутувати суперника).

Винагороди, пов'язані з кожним переходом. У прикладі з роботом-мишею більшість винагород дорівнює 0, але вони стають позитивними, якщо ви досягаєте точки, де є вода або сир, і негативними, якщо ви досягаєте точки, де вас б'є струмом.

Коефіцієнт дисконтування γ від 0 до 1, який кількісно вимірює різницю у важливості між негайною винагородою і майбутньою винагородою. Наприклад, якщо γ дорівнює 0,9, а винагорода за 3 кроки дорівнює 5, теперішня вартість цієї винагороди дорівнює $0{,}9^3*5$.

Відсутність пам'яті. Як тільки відомий поточний стан, історію подорожей миші лабіринтом можна стерти, оскільки поточний марковський стан містить всю корисну інформацію з історії. Іншими словами, "майбутнє не залежить від минулого, враховуючи теперішнє".

Тепер, коли ми знаємо, що таке MDP, ми можемо формалізувати мету миші. Ми намагаємося максимізувати суму винагород у довгостроковій перспективі:

$$\sum_{t=0}^{t=\infty} \gamma^t r(x(t), a(t))$$

Давайте подивимось на цю суму по частинах. Перш за все, ми підсумовуємо на всіх часових кроках t. Давайте поки що покладемо γ рівним 1 і забудемо про нього. r(x,a) - це функція винагороди. Для стану x і дії a (наприклад, поворот наліво на перехресті) вона дає вам винагороду, пов'язану з виконанням цієї дії a в стані x. Повертаючись до нашого рівняння, ми намагаємося максимізувати суму майбутніх винагород, виконуючи найкращу дію в кожному стані.

Тепер, коли ми поставили задачу навчання з підкріпленням і формалізували мету, давайте розглянемо деякі можливі рішення.

Q-навчання: вивчення функції дія-цінність

Q-навчання - це метод, який оцінює, яку дію слід виконати, на основі функції "дія-цінність", що визначає цінність перебування в певному стані та виконання певної дії в цьому стані.

У нас є функція Q, яка приймає на вхід один стан і одну дію і повертає очікувану винагороду за цю дію (і всі наступні дії) у цьому стані. До того, як ми досліджуємо середовище, Q дає однакове (довільне) фіксоване значення. Але потім, коли ми досліджуємо середовище більше, Q дає нам все краще і краще

наближення до вартості дії a в стані s. Ми оновлюємо нашу функцію Q по ходу роботи.

Це рівняння зі сторінки Вікіпедії про Q-навчання дуже добре пояснює все це. Воно показує, як ми оновлюємо значення Q на основі винагороди, яку ми отримуємо від нашого оточення:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Проігноруємо коефіцієнт дисконтування γ, знову встановивши його рівним 1. Перш за все, пам'ятайте, що Q має показувати повну суму винагород від вибору дії Q та всіх наступних оптимальних дій.

Тепер давайте пройдемося по рівнянню зліва направо. Коли ми виконуємо дію at у стані st, ми оновлюємо наше значення Q(st,at), додаючи до нього доданок. Цей доданок містить

Швидкість навчання **альфа**: це те, наскільки агресивними ми хочемо бути при оновленні нашого значення. Коли альфа близька до 0, ми оновлюємо значення не дуже агресивно. Коли альфа близька до 1, ми просто замінюємо старе значення на оновлене.

Винагорода - це винагорода, яку ми отримали, вчинивши дію в стані a. Отже, ми додаємо цю винагороду до нашої старої оцінки.

Ми також додаємо **оцінку майбутньої винагороди**, яка є максимально досяжною винагородою Q для всіх доступних дій в xt+1.

Нарешті, ми віднімаємо старе значення Q, щоб переконатися, що ми збільшуємо або зменшуємо тільки на різницю в оцінці (помножену на альфа, звичайно).

Тепер, коли ми маємо оцінку цінності для кожної пари стан-дія, ми можемо вибрати, яку дію виконати відповідно до нашої стратегії вибору дії (ми не обов'язково обираємо дію, яка кожного разу призводить до найбільш очікуваної винагороди, наприклад, при епсилон-жадібній стратегії дослідження ми б виконували випадкову дію певний відсоток часу).

У прикладі з роботом-мишею ми можемо використати Q-навчання, щоб з'ясувати значення кожної позиції в лабіринті та значення дій {вперед, назад, вліво, вправо} в кожній позиції. Потім ми можемо використати нашу стратегію вибору дій, щоб вибрати, що миша насправді робить на кожному кроці.

У підході Q-навчання ми вивчали функцію цінності, яка оцінювала цінність кожної пари "стан-дія".

Навчання політиці є більш прямолінійною альтернативою, в якій ми вивчаємо функцію політики, π, яка є прямою картою від кожного стану до найкращої відповідної дії в цьому стані. Подумайте про це як про поведінкову політику: "коли я спостерігаю стан s, найкраще зробити дію a". Наприклад, політика автономного транспортного засобу може ефективно включати щось на кшталт: "якщо я бачу жовте світло і перебуваю за 100 футів від перехрестя, я повинен загальмувати. В іншому випадку, продовжуйте рухатися вперед".

Змістовний модуль 2. Глибинне навчання. Data Minig

ТЕМА 7. Ансамблеві методи. Глибинне навчання

Ансамблеві методи

"Купка дерев вчиться виправляти помилки одне одного"

В даний час використовується для:

- Все, що підходить під класичні алгоритмічні підходи (але працює краще)
- Пошукові системи (★)
- Комп'ютерний зір
- Виявлення об'єктів

Популярні алгоритми: Random Forest, Gradient Boosting

Настав час для сучасних, дорослих методів. Ансамблі та нейронні мережі - два головні бійці, які прокладають нам шлях до сингулярності. Сьогодні вони дають найточніші результати і широко використовуються у виробництві.

Однак весь хайп сьогодні припав на нейронні мережі, а такі слова, як "бустінг" або "бекінг", хіба що хіпстери з TechCrunch знають.

Незважаючи на всю ефективність, ідея, що лежить в їх основі, надто проста. Якщо ви візьмете кілька неефективних алгоритмів і змусите їх виправляти помилки один одного, загальна якість системи буде вищою, ніж навіть найкращі окремі алгоритми.

Ви отримаєте ще кращі результати, якщо візьмете найбільш нестабільні алгоритми, які пророкують абсолютно різні результати на невеликому рівні шуму у вхідних даних. Наприклад, регресію та дерева рішень. Ці алгоритми настільки чутливі навіть до одного викиду у вхідних даних, що моделі можуть збожеволіти. Насправді, це те, що нам потрібно.

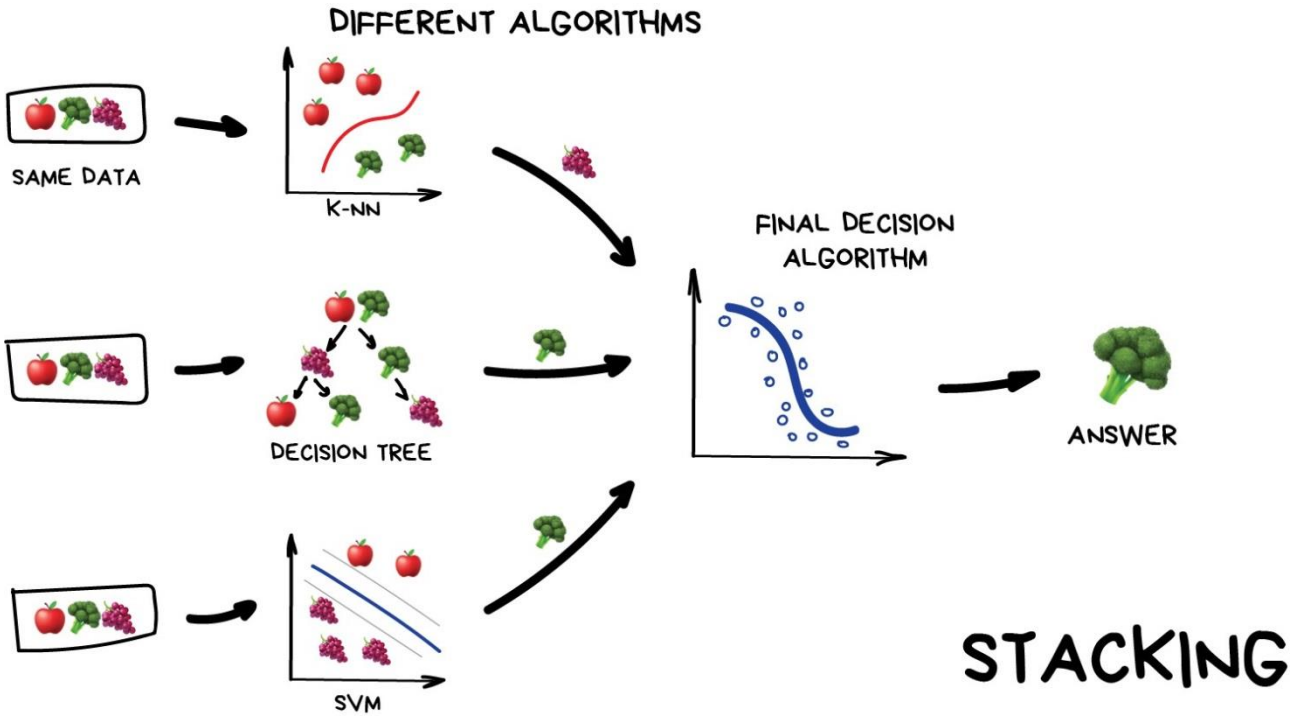
Ми можемо використовувати будь-який відомий нам алгоритм для створення ансамблю. Просто накидайте купу класифікаторів, приправте їх регресією і не забудьте виміряти точність. З мого досвіду: навіть не намагайтеся використовувати тут Байєса або kNN. Вони хоч і "тупі", але дуже стабільні. Це нудно і передбачувано.

Натомість є три перевірені в боях методи створення ансамблів.

Стекінг. Вихід декількох паралельних моделей передається на вхід останньої, яка приймає остаточне рішення. Як та дівчина, яка запитує своїх подруг, чи варто з вами зустрічатися, щоб потім самій прийняти остаточне рішення.

Наголос тут на слові "різні". Змішування однакових алгоритмів на одних і тих самих даних не має сенсу. Вибір алгоритмів повністю залежить від вас. Однак, для остаточної моделі прийняття рішень, регресія зазвичай є хорошим вибором.

Виходячи з мого досвіду, стекінг менш популярний на практиці, тому що два інших методи дають кращу точність.



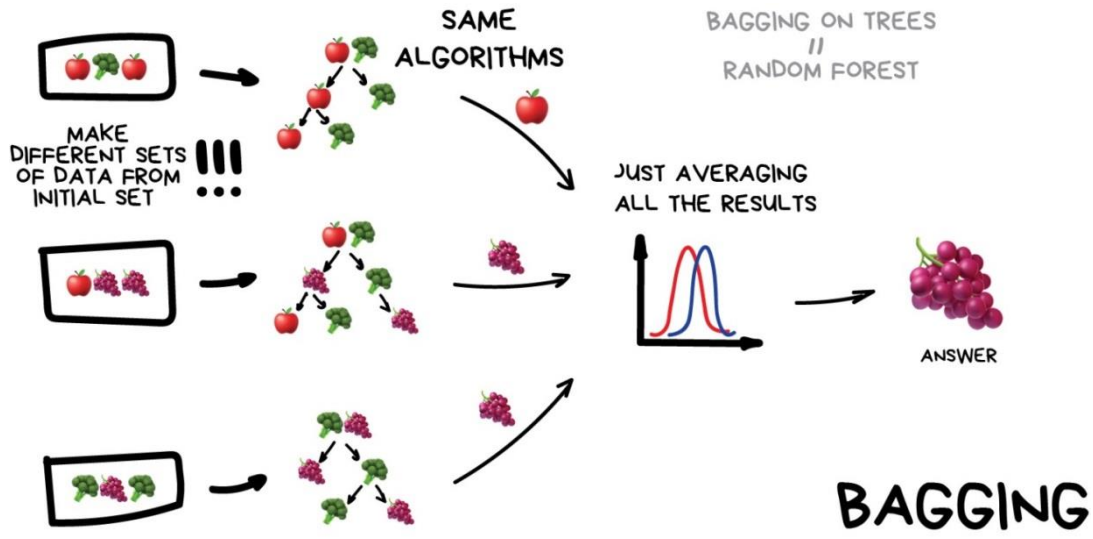
Bagging, також відоме як Bootstrap AGGregatING. Використовуйте той самий алгоритм, але тренуйте його на різних підмножинах вихідних даних. В результаті - лише усереднені відповіді.

Дані у випадкових підмножинах можуть повторюватися. Наприклад, з набору "1-2-3" ми можемо отримати підмножини "2-2-3", "1-2-2", "3-1-2" і так далі. Ми використовуємо ці нові набори даних, щоб навчити той самий алгоритм кілька разів, а потім передбачити остаточну відповідь за допомогою простого голосування більшістю голосів.

Найвідомішим прикладом пакування є алгоритм Random Forest, який просто пакує дані на деревах рішень (які були проілюстровані вище). Коли ви відкриваєте додаток камери на своєму телефоні і бачите, як вона малює квадратики навколо облич людей - це, ймовірно, результати роботи методу випадкового лісу.

Нейронні мережі були б надто повільними для роботи в реальному часі, але пакетна обробка є ідеальним рішенням, оскільки вона може обчислювати дерева на всіх шейдерах відеокарти або на цих нових модних процесорах ML.

У деяких завданнях здатність Random Forest працювати паралельно важливіша, ніж, наприклад, невелика втрата точності через прискорення. Особливо при обробці в реальному часі. Завжди є компроміс.

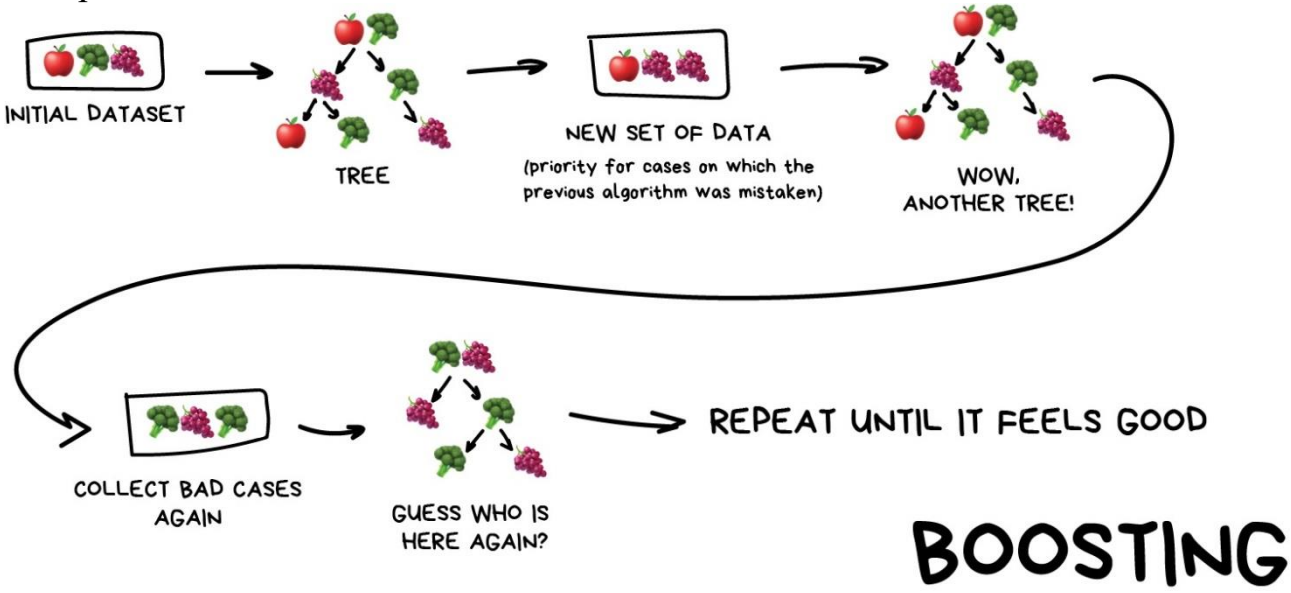


Алгоритми **Boosting** навчаються послідовно один за одним. Кожен наступний алгоритм приділяє найбільше уваги тим точкам даних, які були неправильно передбачені попереднім. Повторюємо до тих пір, поки не отримаємо позитивний результат.

Так само, як і при пакуванні, ми використовуємо підмножини наших даних, але цього разу вони не генеруються випадковим чином. Тепер у кожній підвибірці ми беремо частину даних, які попередній алгоритм не зміг обробити. Таким чином, ми змушуємо новий алгоритм вчитися виправляти помилки попереднього.

Алгоритми Boosting навчаються послідовно один за одним. Кожен наступний алгоритм приділяє найбільше уваги тим точкам даних, які були неправильно передбачені попереднім. Повторюйте доти, доки не отримаєте бажаного результату.

Так само, як і при пакуванні, ми використовуємо підмножини наших даних, але цього разу вони не генеруються випадковим чином. Тепер у кожній підвибірці ми беремо частину даних, які попередній алгоритм не зміг обробити. Таким чином, ми змушуємо новий алгоритм вчитися виправляти помилки попереднього.



Нейронні мережі та глибинне навчання

"Ми маємо тисячошарову мережу, десятки відеокарт, але досі не знаємо, куди це все застосувати. Давайте генерувати картинки!"

Використовується сьогодні для:

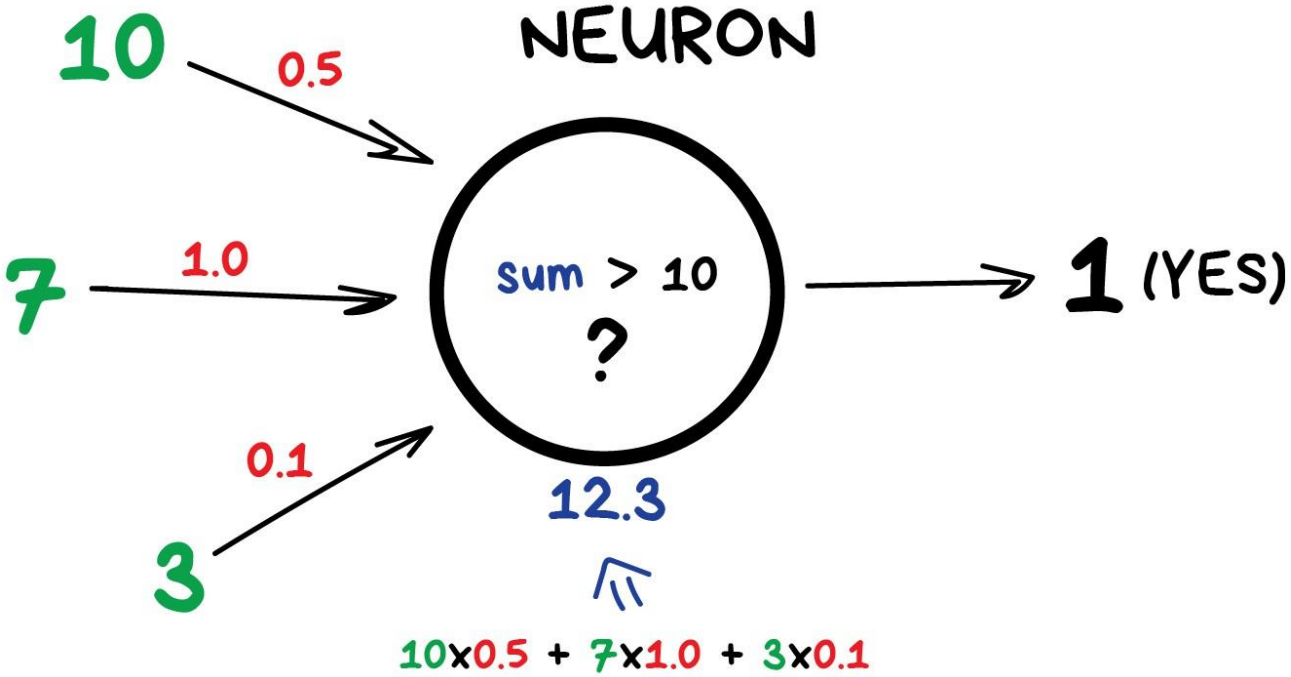
- Заміна всіх алгоритмів вище
- Ідентифікація об'єктів на фото та відео
- Розпізнавання та синтез мови
- Обробка зображень, передача стилю
- Машинний переклад

Популярні архітектури: Персептрон, згорткові мережі (CNN), рекурентні мережі (RNN), автокодери

Якщо ніхто ніколи не намагався пояснити вам нейронні мережі, використовуючи аналогії з "людським мозком", ви щасливі. Будь-яка нейронна мережа - це, по суті, набір нейронів і зв'язків між ними. Нейрон - це функція з купою входів і одним виходом. Його завдання - взяти всі числа зі свого входу, виконати над ними певну функцію і відправити результат на вихід.

Ось приклад простого, але корисного в реальному житті нейрона: підсумувати всі числа зі входів і, якщо ця сума більша за N, видати на виході 1. В іншому випадку - нуль.

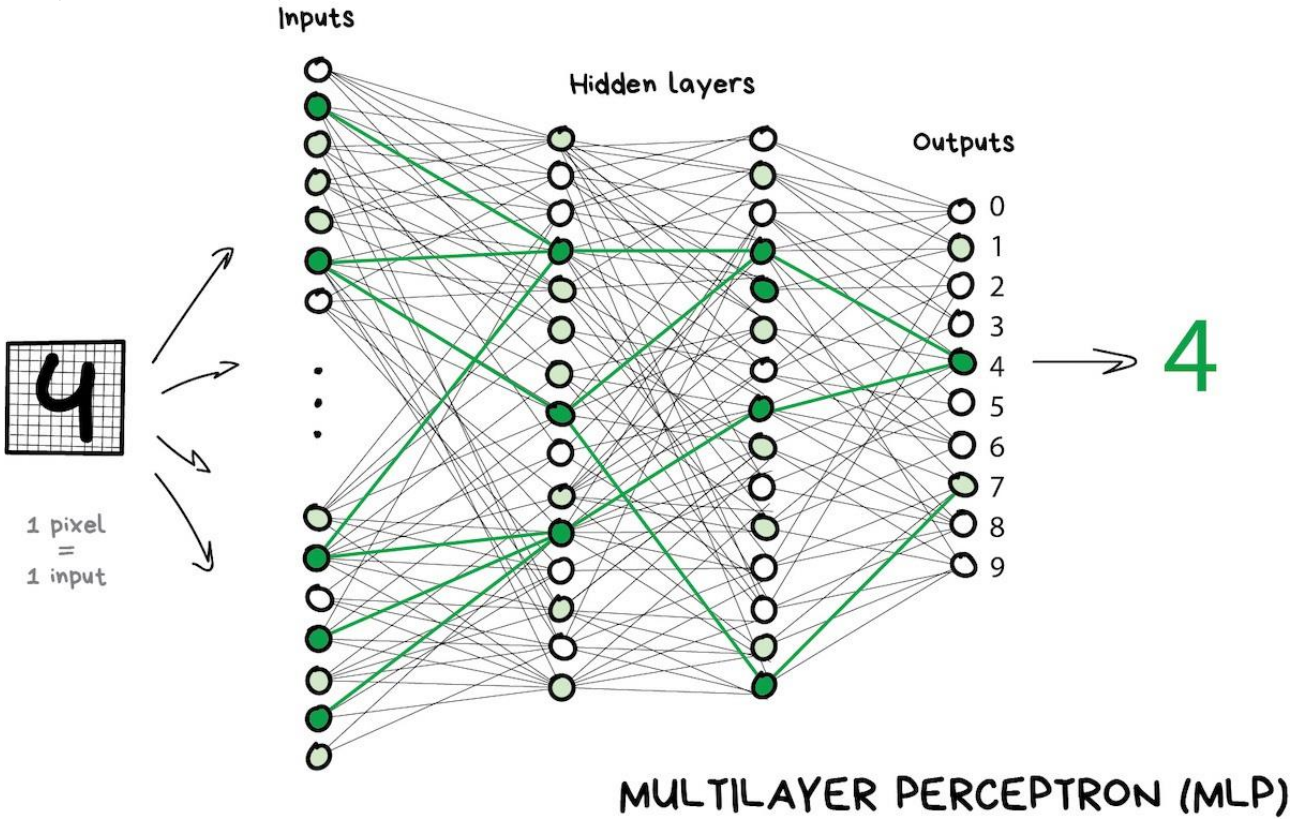
Зв'язки - це як канали між нейронами. Вони з'єднують виходи одного нейрона з входами іншого, щоб вони могли передавати один одному цифри. Кожне з'єднання має лише один параметр - вагу. Це як сила зв'язку для сигналу. Коли число 10 проходить через з'єднання з вагою 0.5, воно перетворюється на 5. Ці ваги говорять нейрону, що на один вхід потрібно реагувати більше, а на інший - менше. Ваги коригуються під час навчання - так мережа навчається. В принципі, це все, що потрібно знати.



Щоб мережа не впала в анархію, нейрони з'єднані шарами, а не випадковим чином. Усередині шару нейрони не пов'язані між собою, але вони

пов'язані з нейронами наступного і попереднього шарів. Дані в мережі йдуть строго в одному напрямку - від входів першого шару до виходів останнього.

Якщо накидати достатню кількість шарів і правильно розставити ваги, то вийде наступне: при подачі на вхід, скажімо, зображення рукописної цифри 4, чорні пікселі активують пов'язані з ними нейрони, ті активують наступні шари, і так далі, поки нарешті не засвітиться вихід, що відповідає за четвірку. Результат досягнуто.



При програмуванні в реальному житті ніхто не пише нейрони та зв'язки. Замість цього все представляється у вигляді матриць і обчислюється на основі множення матриць для кращої продуктивності.

Після того, як ми побудували мережу, наше завдання - призначити правильні шляхи, щоб нейрони правильно реагували на вхідні сигнали. Настав час згадати, що у нас є дані, які є зразками "входів" і правильних "виходів". Ми покажемо нашій мережі малюнок тієї самої цифри 4 і скажемо їй: "Адаптуй свої ваги так, щоб при отриманні цього входу на виході було 4".

Для початку всі ваги задаються випадковим чином. Після того, як ми показуємо їй цифру, вона видає випадкову відповідь, оскільки ваги ще не є правильними, і ми порівнюємо, наскільки цей результат відрізняється від правильного. Потім ми починаємо обходити мережу в зворотному напрямку від виходів до входів і кажемо кожному нейрону: "Гей, ти активувався тут, але ти зробив жахливу роботу, і далі все пішло не так, давай менше уваги приділяти цьому зв'язку, а більше тому, добре?".

Після сотень тисяч таких циклів "висновок-перевірка-покарання" з'являється надія, що ваги виправлені і діють за призначенням. Наукова назва цього підходу - Backpropagation, або "метод зворотного поширення помилки".

Цікаво, що на розробку цього методу пішло двадцять років. До цього ми ще якось вчили нейронні мережі.

Добре навчена нейронна мережа може імітувати роботу будь-якого з алгоритмів, описаних у цьому розділі (і часто працює точніше). Ця універсальність і зробила їх широко популярними. Нарешті у нас є архітектура людського мозку, яку, як вони сподівалися, потрібно просто зібрати з багатьох шарів і навчити їх на будь-яких можливих даних. Потім почалася перша AI-зима, потім вона відтанула, а потім накрила чергова хвиля розчарувань.

Виявилося, що мережі з великою кількістю шарів вимагають немислимих на той час обчислювальних потужностей. Сьогодні будь-який геймерський комп'ютер з гефорсами перевершує тогочасні дата-центри. Тож тоді люди не мали жодної надії отримати таку обчислювальну потужність, і нейронні мережі були величезним обломом.

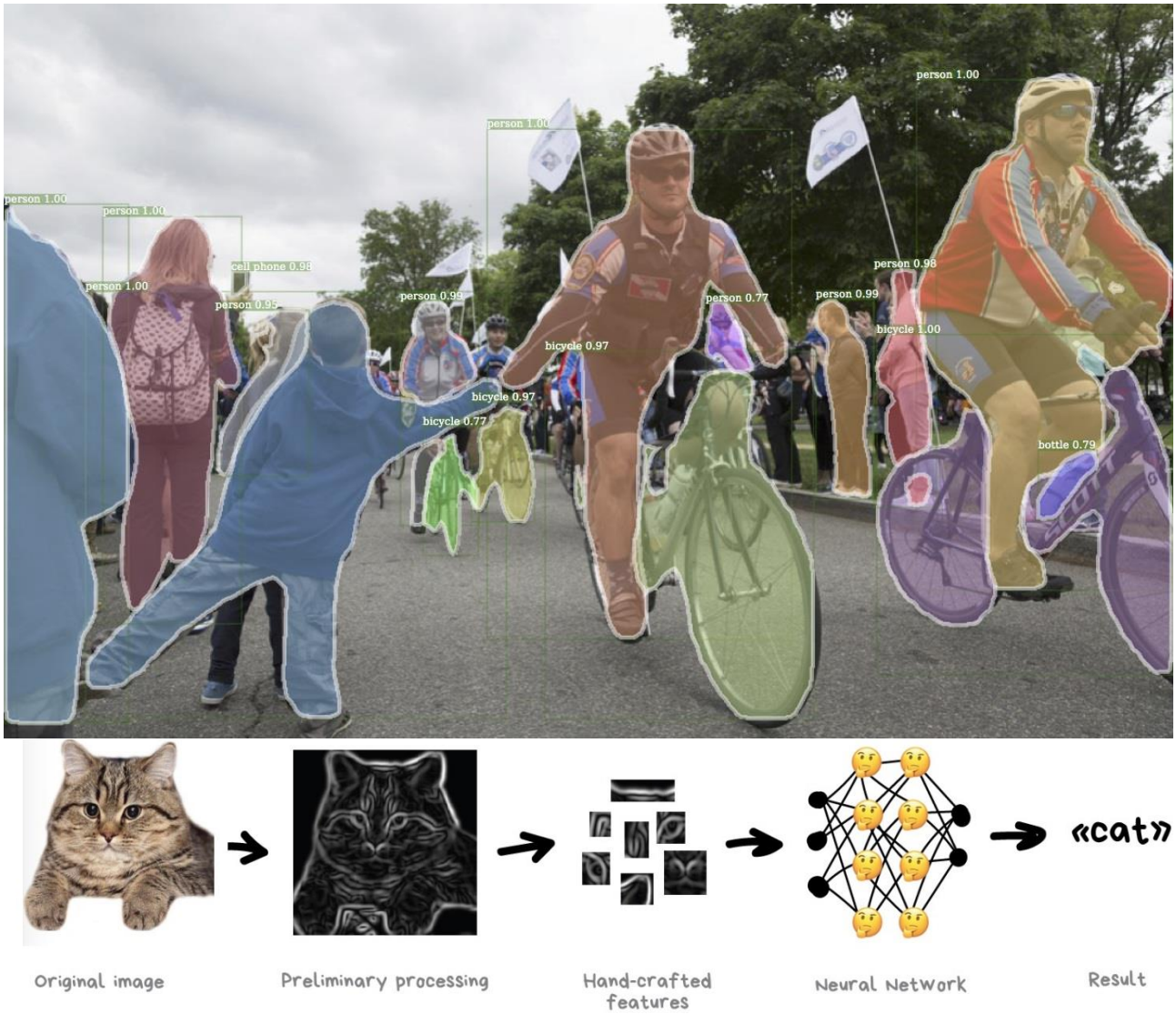
А потім десять років тому з'явилося глибоке навчання.

У 2012 році згорткові нейронні мережі здобули блискучу перемогу в конкурсі ImageNet, що змусило світ раптово згадати про методи глибокого навчання, описані ще в далеких 90-х роках. Тепер у нас є відеокарти!

Відмінності глибокого навчання від класичних нейронних мереж полягали в нових методах навчання, які могли працювати з більшими мережами. Зараз тільки теоретики намагаються розділити, яке навчання вважати глибоким, а яке не дуже. А ми, як практики, використовуємо популярні "глибокі" бібліотеки, такі як Keras, TensorFlow та PyTorch, навіть коли будуємо міні-мережу з п'ятьма шарами. Просто тому, що вони підходять краще, ніж усі інструменти, які були до цього. І ми просто називаємо їх нейронними мережами.

Згорткові нейронні мережі зараз в тренді. Їх використовують для пошуку об'єктів на фотографіях і відео, розпізнавання облич, перенесення стилю, генерації та покращення зображень, створення ефектів на кшталт сповільненої зйомки та покращення якості зображень. Сьогодні CNN використовуються у всіх випадках, коли йдеться про фотографії та відео. Навіть у вашому iPhone кілька таких мереж переглядають ваші знімки, щоб виявити на них об'єкти. Якщо є що виявляти.

Проблемою із зображеннями завжди була складність вилучення з них інформації. Текст можна розбивати на речення, шукати атрибути слів у спеціалізованих словниках тощо. Але зображення доводилося маркувати вручну, щоб навчити машину, де на цьому конкретному зображенні котячі вуха чи хвости. Такий підхід отримав назву "ручної роботи" і використовувався майже всіма.



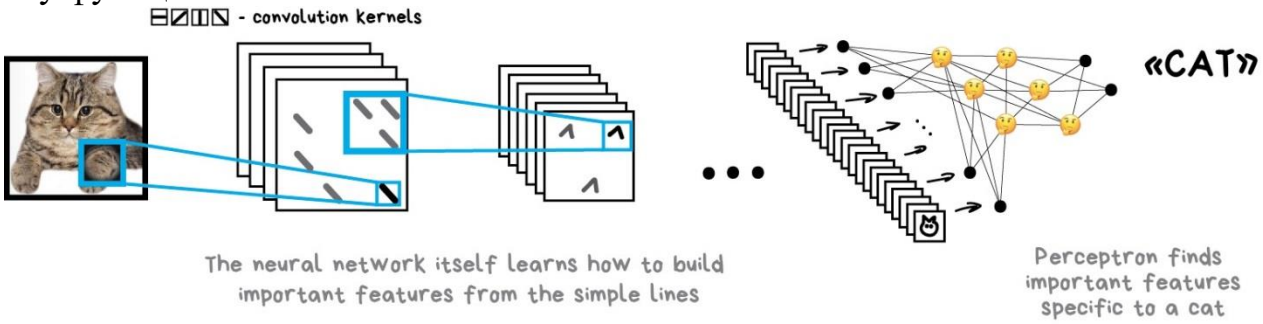
З ручною роботою є багато проблем. По-перше, якщо кіт притиснув вуха або відвернувся від камери - у вас проблеми, нейромережа нічого не побачить.

По-друге, спробуйте назвати на місці 10 різних ознак, які відрізняють котів від інших тварин. Я, наприклад, не можу цього зробити, але коли бачу, як повз мене вночі проноситься чорна пляма - навіть якщо я бачу її лише краєм ока - я точно відрізню кота від щура. Тому що люди дивляться не лише на форму вух чи кількість лап, а враховують багато різних ознак, про які вони навіть не замислюються. І тому не можуть пояснити це машині.

Це означає, що машині потрібно вивчити такі особливості самостійно, відштовхуючись від базових ліній. Ми зробимо так: спочатку розбиваємо все зображення на блоки 8x8 пікселів і призначаємо кожному з них тип домінуючої лінії - або горизонтальну [-], або вертикальну [|], або одну з діагоналей [/]. Також може статися, що деякі з них будуть дуже помітними - таке трапляється, і ми не завжди абсолютно впевнені в цьому.

На виході ми отримаємо кілька таблиць паличок, які фактично є найпростішими об'єктами, що представляють краї об'єктів на зображенні. Вони самі по собі є зображеннями, але побудовані з паличок. Тож ми можемо знову взяти блок 8x8 і подивитися, як вони поєднуються між собою. І знову, і знову...

Ця операція називається згортка, яка і дала назву методу. Згортку можна уявити як шар нейронної мережі, адже кожен нейрон може виконувати будь-яку функцію.



CONVOLUTIONAL NEURAL NETWORK (CNN)

Коли ми завантажуємо в нейромережу багато фотографій котів, вона автоматично присвоює більшу вагу тим комбінаціям паличок, які вона бачила найчастіше. Неважливо, чи це була пряма лінія котячої спини, чи геометрично складний об'єкт на кшталт котячої мордочки, щось буде дуже активним.

На виході ми поставимо простий персептрон, який буде дивитися на найбільш активні комбінації і на основі цього відрізняти котів від собак.

Принадність цієї ідеї в тому, що ми маємо нейромережу, яка сама шукає найбільш характерні риси об'єктів. Нам не потрібно вибирати їх вручну. Ми можемо завантажити в неї будь-яку кількість зображень будь-якого об'єкта, просто загугливши з її допомогою мільярди зображень, і наша мережа створить карти ознак з паличок і навчиться розрізняти будь-який об'єкт самостійно.

Рекурентні нейронні мережі (RNN)

Друга за популярністю архітектура на сьогоднішній день. Рекурентні мережі дали нам такі корисні речі, як нейронний машинний переклад, розпізнавання мови та синтез голосу в розумних помічниках. ШНМ найкраще підходять для послідовних даних, таких як голос, текст або музика.

Пам'ятаєте Microsoft Sam, олдскульний синтезатор мови з Windows XP? Цей кумедний хлопець складав слова літеру за літерою, намагаючись склеїти їх докупи. А тепер подивіться на Amazon Alexa або Assistant від Google. Вони не лише чітко вимовляють слова, але й розставляють правильні наголоси!

А все тому, що сучасні голосові помічники навчені говорити не по буквах, а цілими фразами відразу. Ми можемо взяти кілька озвучених текстів і навчити нейромережу генерувати аудіопослідовність, максимально наближену до оригінального мовлення.

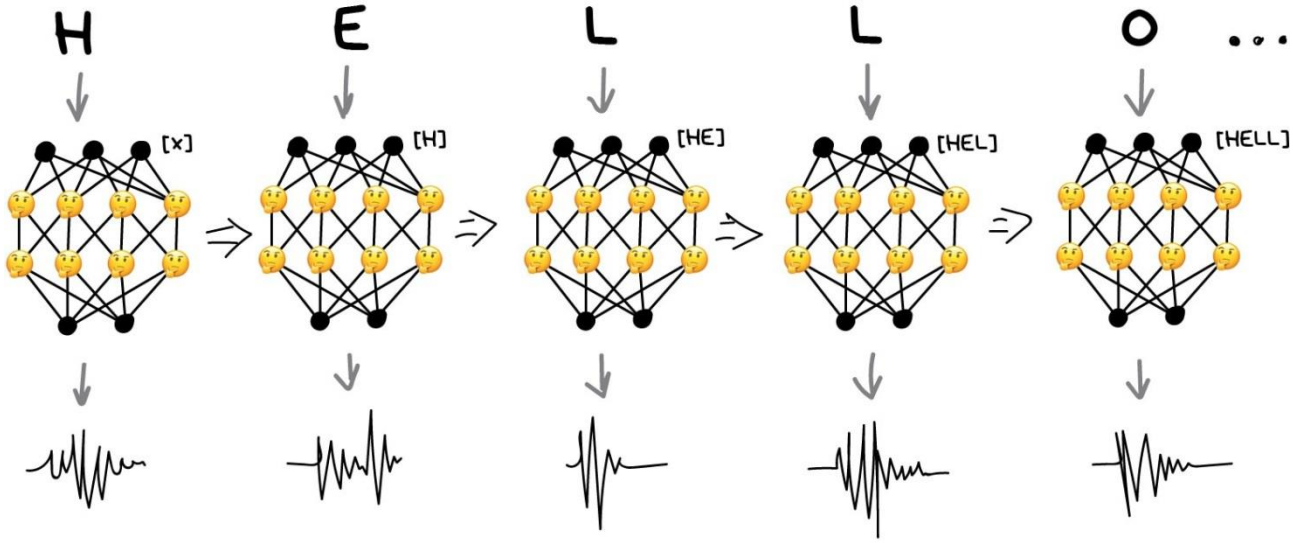
Іншими словами, ми використовуємо текст як вхід, а його аудіо - як бажаний вихід. Ми просимо нейромережу згенерувати певний звук для заданого тексту, потім порівнюємо його з оригіналом, виправляємо помилки і намагаємось максимально наблизитись до ідеалу.

Звучить як класичний процес навчання. Для цього підходить навіть персептрон. Але як визначити його виходи? Очевидно, що для кожної можливої фрази випускати один конкретний вихід - не варіант.

Тут нам допоможе той факт, що текст, мова або музика - це послідовності. Вони складаються з послідовних одиниць, таких як склади. Кожен з них звучить унікально, але залежить від попередніх. Втратьте цей зв'язок - і отримаєте дабстеп.

Ми можемо навчити персептрон генерувати ці унікальні звуки, але як він запам'ятає попередні відповіді? Ідея полягає в тому, щоб додати пам'ять до кожного нейрона і використовувати її як додатковий вхідний сигнал під час наступного прогону. Нейрон може зробити примітку для себе - агов, у нас тут була голосна, наступний звук має звучати вище (це дуже спрощений приклад).

Так з'явилися рекурентні мережі.



RECURRENT NEURAL NETWORK (RNN)

Цей підхід мав одну величезну проблему - коли всі нейрони пам'ятали свої минулі результати, кількість зв'язків у мережі ставала настільки великою, що технічно неможливо було налаштувати всі ваги.

Коли нейромережа не може забути, вона не може навчитися новому (люди мають таку ж ваду).

Перше рішення було простим: обмежити пам'ять нейронів. Скажімо, запам'ятовувати не більше 5 останніх результатів. Але це ламало всю ідею.

Пізніше з'явився набагато кращий підхід: використовувати спеціальні комірки, схожі на пам'ять комп'ютера. У кожну комірку можна записати число, зчитати його або скинути. Їх назвали комірками довготривалої та короткочасної пам'яті (LSTM).

Тепер, коли нейрону потрібно встановити нагадування, він ставить прапорець у цій комірці. Наприклад, "у слові був приголосний, наступного разу використовуйте інші правила вимови". Коли прапорець більше не потрібен, клітинки скидаються, залишаючи лише "довготривалі" зв'язки класичного перцептрону. Іншими словами, мережа навчена не тільки запам'ятовувати ваги, але й встановлювати ці нагадування.

Нейронні мережі та глибинне навчання (альтернативне бачення)

При глибокому навчанні ми все ще вивчаємо функцію f для відображення вхідних даних X у вихідні Y з мінімальними втратами на тестових даних, так само, як ми це робили весь цей час. Згадайте нашу початкову "постановку задачі" про навчання під контролем:

$Y = f(X) + \epsilon$

Навчання: машина дізнається f з помічених навчальних даних

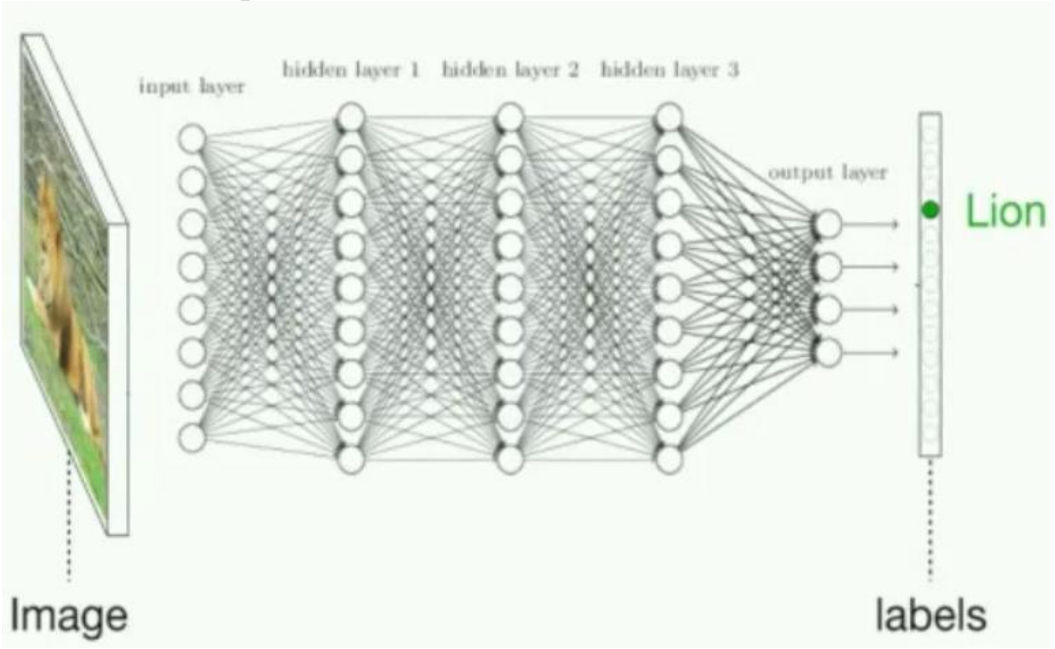
Тестування: машина прогнозує Y на основі немаркованих тестових даних

Реальний світ безладний, тому іноді f буває складною. У задачах з природною мовою великі обсяги словника означають велику кількість ознак. Проблеми зорового сприйняття пов'язані з великою кількістю візуальної інформації про пікселі. Ігри вимагають прийняття рішення на основі складних сценаріїв з багатьма можливими варіантами розвитку подій. Методи навчання, які ми розглядали раніше, добре працюють, коли дані, з якими ми працюємо, не надто складні, але незрозуміло, як їх узагальнити для подібних сценаріїв.

Глибоке навчання дійсно добре вивчає f, особливо в ситуаціях, коли дані є складними. Насправді, штучні нейронні мережі відомі як універсальні апроксиматори функцій, тому що вони здатні вивчити будь-яку функцію, незалежно від того, наскільки вона складна, за допомогою лише одного прихованого шару.

Давайте розглянемо проблему класифікації зображень. На вхід подається зображення, а на виході - клас (наприклад, собака, кішка, машина).

Графічно глибока нейронна мережа, яка вирішує задачу класифікації зображень, виглядає приблизно так:



Але насправді це гігантське математичне рівняння з мільйонами членів і безліччю параметрів. На вхід X подається, скажімо, зображення у відтінках сірого, представлене матрицею яскравостей пікселів w by h. Вихід Y - це вектор ймовірностей класів. Це означає, що на виході ми маємо ймовірність того, що кожен клас є правильною міткою. Якщо нейромережа працює добре, то найбільша ймовірність має бути для правильного класу. А шари посередині

просто виконують множення матриць, підсумовуючи активації x ваги з нелінійними перетвореннями (функціями активації) після кожного прихованого шару, щоб мережа могла вивчити нелінійну функцію.

Неймовірно, але ви можете використовувати градієнтний спуск точно так само, як ми це робили з лінійною регресією, щоб навчити ці параметри у спосіб, який мінімізує втрати. Отже, маючи багато прикладів і багато градієнтного спуску, модель може навчитися правильно класифікувати зображення тварин. І це, якщо коротко, і є "глибоким навчанням".

Де глибоке навчання працює добре, і трохи історії

Штучні нейронні мережі насправді існують вже давно. Їх застосування історично називали кібернетикою (1940-1960-ті роки), коннекціонізмом (1980-1990-ті роки), а потім вони увійшли в моду як глибоке навчання приблизно в 2006 році, коли нейронні мережі почали ставати, ну, "глибшими" (Goodfellow et al., 2016). Але лише нещодавно ми дійсно почали дряпати поверхню їхнього повного потенціалу.

За словами Андрея Карпаті (директора зі штучного інтелекту в компанії Tesla, якого ми схильні вважати шаманом глибокого навчання), загалом є "чотири окремі фактори, які стримують ШІ:

- Обчислення (очевидний: закон Мура, графічні процесори, ASIC),
- Дані (в хорошій формі, а не просто десь в інтернеті - наприклад, ImageNet),
- Алгоритми (дослідження та ідеї, наприклад, фон, CNN, LSTM), і
- Інфраструктура (програмне забезпечення під вами - Linux, TCP/IP, Git, ROS, PR2, AWS, AMT, TensorFlow тощо)". (Карпати, 2016).

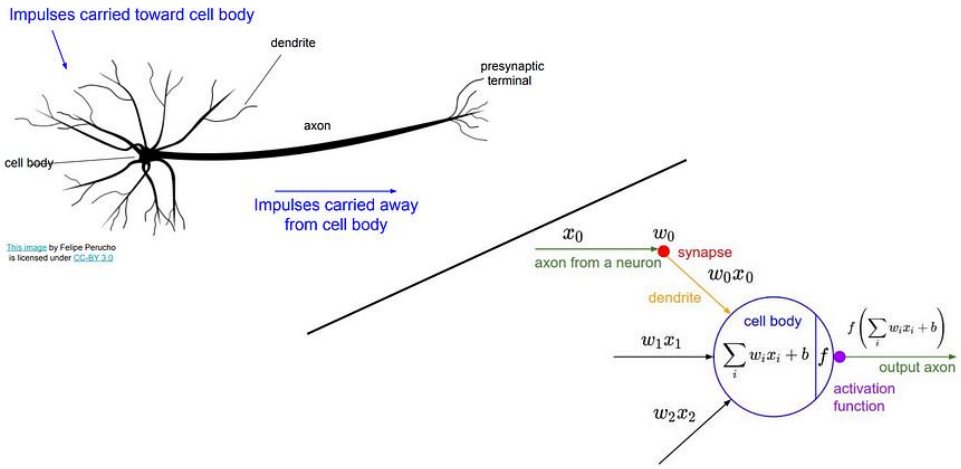
За останнє десятиліття або близько того весь потенціал глибокого навчання нарешті був розкритий завдяки досягненням у пунктах (1) і (2), що, в свою чергу, призвело до подальших проривів у пунктах (3) і (4) - і так цикл продовжується, і все більше людей стають на передові рубежі досліджень глибокого навчання на цьому шляху.

Нейрони, особливості навчання та рівні абстракції

Коли ви читаєте ці слова, ви не вивчаєте кожну літеру в кожному слові чи кожен піксель, що складає кожну літеру, щоб зрозуміти їхнє значення. Ви абстрагуєтесь від деталей і групуєте речі в поняття більш високого рівня: слова, фрази, речення, абзаци.

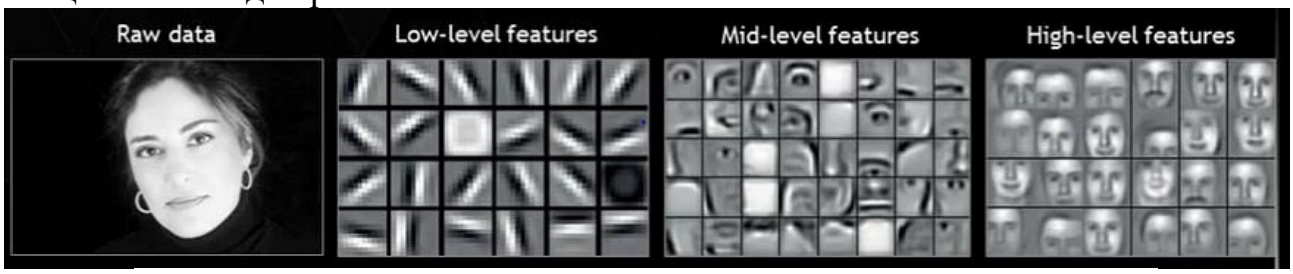
Те ж саме відбувається і в зоровій системі, причому не лише у людей, а й у зорових системах тварин загалом.

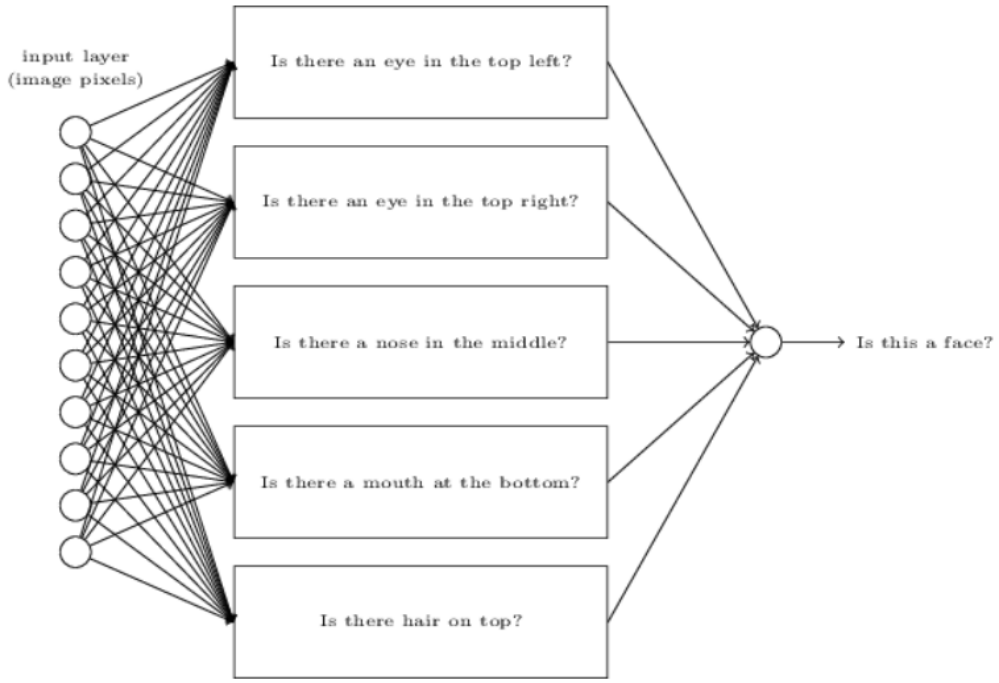
Мозок складається з нейронів, які "стріляють", випромінюючи електричні сигнали до інших нейронів після того, як достатньо "активуються". Ці нейрони є пластичними з точки зору того, наскільки сигнал від інших нейронів додасть до рівня активації нейрона (нечітко кажучи, ваги, що з'єднують нейрони один з одним, в кінцевому підсумку тренуються, щоб зробити нейронні зв'язки більш корисними, так само, як параметри в лінійній регресії можуть тренуватися, щоб покращити відображення від входу до виходу).



Ілюстрації біологічних та штучних нейронів, зроблені за допомогою Стенфордського комп'ютера CS231n. Цю аналогію не можна сприймати надто буквально - біологічні нейрони можуть робити те, чого не можуть штучні, і навпаки - але вона корисна для розуміння біологічного натхнення. Детальніше про біологічні та штучні нейрони читайте у Вікіпедії.

Наші біологічні мережі влаштовані ієрархічно, так що певні нейрони в кінцевому підсумку виявляють не вкрай конкретні риси навколишнього світу, а більш абстрактні риси, тобто патерни або угруповання більш низькорівневих ознак. Наприклад, фузіформна область обличчя у зоровій системі людини спеціалізована для розпізнавання облич.





Вгорі: ілюстрація вивчення все більш абстрактних ознак за допомогою NVIDIA. Внизу: схема того, як штучна нейронна мережа отримує вхідні дані у

вигляді пікселів, створює проміжні "нейрони" для виявлення ознак вищого рівня (наприклад, наявність носа) і об'єднує їхні виходи для створення кінцевого результату.

Цю ієрархічну структуру, яку демонструють біологічні нейронні мережі, було відкрито в 1950-х роках, коли дослідники Девід Хьюбел і Торстен Візель вивчали нейрони в зоровій корі котів. Вони не могли спостерігати активацію нейронів після впливу на кішку різноманітних стимулів: темних плям, світлих плям, помахів рук і навіть фотографій жінок у журналах. Але на своє розчарування, коли вони вийняли слайд з проектора під діагональним кутом, вони помітили деяку нейронну активність! Виявилося, що діагональні краї під дуже специфічним кутом викликали активацію певних нейронів.

Це має еволюційний сенс, оскільки природне середовище, як правило, галасливе і хаотичне (уявіть собі трав'янисту рівнину або скелясту місцевість). Отже, коли кішка в дикій природі сприймає "край", тобто лінію, яка контрастує з її фоном, це може вказувати на те, що об'єкт або істота знаходиться в полі її зору. Коли активується певна комбінація крайових нейронів, ці активації об'єднуються, щоб дати ще більш абстрактну активацію, і так далі, поки остаточна абстракція не стане корисним поняттям, таким як "птах" або "вовк".

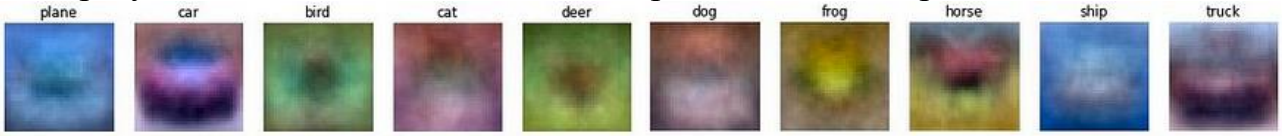
Ідея глибокої нейронної мережі полягає в тому, щоб імітувати подібну структуру за допомогою шарів штучних нейронів.

Чому лінійні моделі не працюють

На прикладі чудового Стенфордського курсу з глибокого навчання CS231n: Згорткові нейронні мережі та візуальне розпізнавання, уявіть, що ми хочемо навчити нейронну мережу класифікувати зображення з правильною однією з наступних міток: ["літак", "автомобіль", "птах", "кіт", "олень", "собака", "жаба", "кінь", "корабель", "вантажівка"].

Один з підходів може полягати в побудові "шаблону", або середнього зображення, для кожного класу зображень на основі навчальних прикладів, а потім використання алгоритму найближчих сусідів під час тестування для вимірювання відстані значень пікселів кожного некласифікованого зображення в сукупності до кожного шаблону. Цей підхід не передбачає жодних шарів абстракції. Це лінійна модель, яка поєднує всі різні орієнтації кожного типу зображення в одне усереднене розмиття.

Наприклад, ми беремо всі автомобілі - незалежно від того, чи стоять вони ліворуч, праворуч, по центру, і незалежно від їхнього кольору - і усереднюємо їх. В результаті шаблон виглядає досить розпливчастим і розмитим.

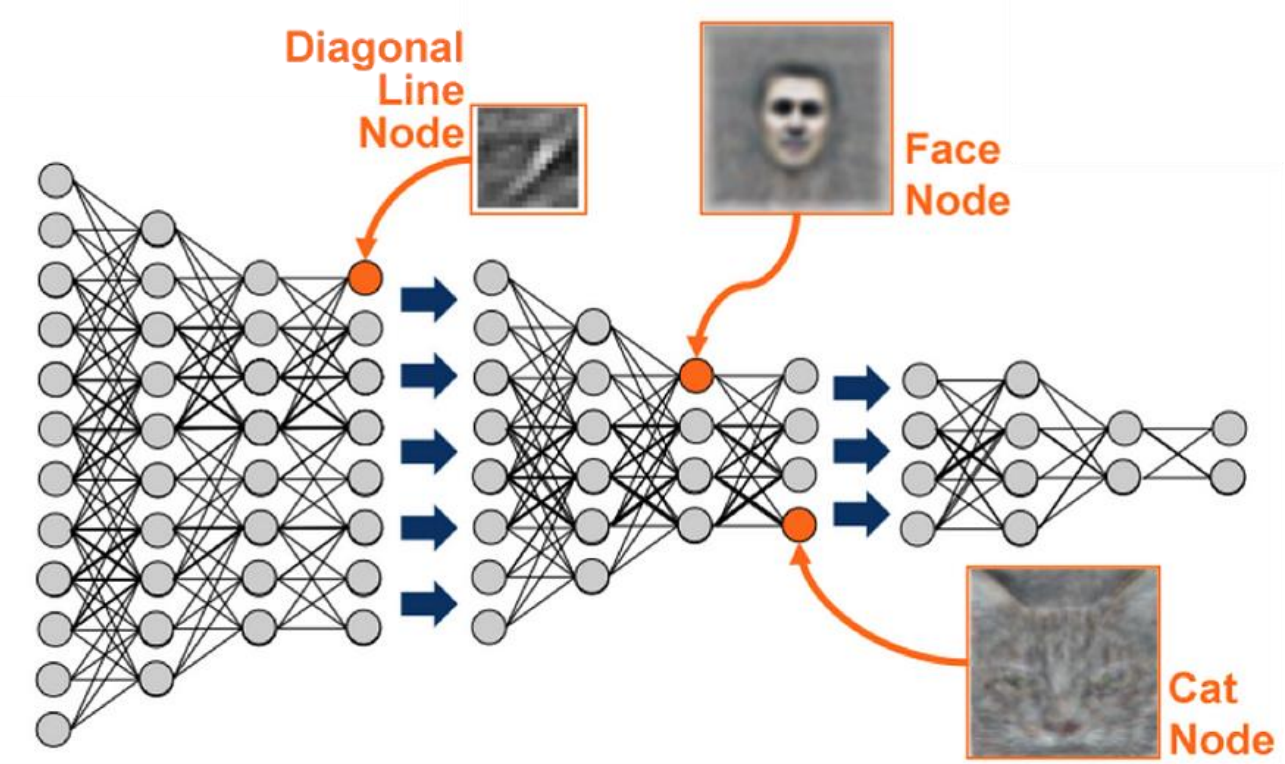


Зверніть увагу, що шаблон коня вище має дві голови. Нам це не дуже допомагає: ми хочемо мати можливість розпізнавати коней, що стоять праворуч або ліворуч, окремо, а потім, якщо виявлено будь-яку з цих ознак, ми хочемо сказати, що ми дивимося на коня. Таку гнучкість забезпечують глибокі нейронні мережі, як ми побачимо в наступному розділі.

Глибокі нейронні мережі підходять до проблеми класифікації зображень, використовуючи шари абстракції

Повторимо те, що ми пояснювали раніше в цьому розділі: вхідним шаром будуть необроблені яскравості пікселів зображення. Кінцевим шаром буде вихідний вектор ймовірностей класів (тобто ймовірність того, що зображення є "кішкою", "автомобілем", "конем" і т.д.).

Але замість того, щоб вивчати просту лінійну модель, що пов'язує вхід з виходом, ми будемо будувати проміжні приховані шари мережі, які будуть навчатися все більш абстрактним ознакам, що дозволить нам не втратити всі нюанси в складних даних.



Подібно до того, як ми описали мозок тварин, що виявляє абстрактні ознаки, штучні нейрони в прихованих шарах навчаться виявляти абстрактні концепції - будь-які концепції, які в кінцевому підсумку є найбільш корисними для захоплення найбільшої кількості інформації і мінімізації втрат в точності вихідних даних мережі (це приклад неконтрольованого навчання, що відбувається всередині мережі).

Це відбувається ціною інтерпретованості моделі, оскільки з додаванням нових прихованих шарів нейрони починають представляти все більш абстрактні і, зрештою, незрозумілі характеристики - до такої міри, що ви можете почути, що глибоке навчання називають "оптимізацією чорної скриньки", коли ви, по суті, просто пробуєте щось навмання і дивитеся, що виходить, не розуміючи, що відбувається всередині.

Лінійну регресію можна інтерпретувати, тому що ви вирішили, які особливості включити в модель. Глибокі нейронні мережі важче інтерпретувати, тому що функції вивчаються і ніде не пояснюються англійською мовою. Це все в уяві машини.

Деякі розширення та подальші концепції, на які варто звернути увагу

- **Програмні пакети для глибокого навчання**. Вам рідко доведеться реалізовувати всі частини нейронних мереж з нуля, оскільки існують бібліотеки та інструменти, які спрощують реалізацію глибинного навчання. Існує багато таких інструментів: TensorFlow, Caffe, Torch, Theano та інші.
- **Згорткові нейронні мережі** (CNN). ШНМ розроблені спеціально для обробки зображень і є ефективними для задач комп'ютерного зору. Вони також відіграють важливу роль у глибокому навчанні з підкріпленням. CNN натхненні тим, як працює зорова кора головного мозку тварин, і вони є основною темою курсу глибокого навчання, на який ми посилаємося в цій статті, Стенфордського курсу CS231n.
- **Рекурентні нейронні мережі** (RNN). ШНМ мають вбудовану пам'ять і добре підходять для вирішення мовних проблем. Вони також важливі для навчання з підкріпленням, оскільки дозволяють агенту відстежувати, де знаходяться речі і що відбувалося в минулому, навіть якщо всі ці елементи не видно одразу. Крістофер Олах написав чудову статтю про RNN та LSTM в контексті мовних проблем.
- **Глибоке навчання з підкріпленням**. Це одна з найцікавіших сфер досліджень глибокого навчання, яка лежить в основі таких нещодавніх досягнень, як перемога OpenAI над професійними гравцями в Dota 2 та AlphaGo від DeepMind, що перевершує людей у грі в го. Ми зануримося глибше в частині 5, але, по суті, мета полягає в тому, щоб застосувати всі методи, описані в цій статті, до проблеми навчання агента для максимізації винагороди. Це можна застосувати в будь-якому контексті, який можна гейміфікувати - від реальних ігор, таких як Counter Strike або Pacman, до безпілотних автомобілів, торгівлі акціями і (зрештою) до реального життя і реального світу.

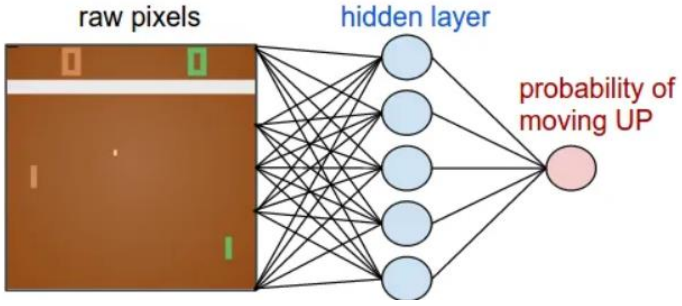
У підході Q-навчання ми вивчали функцію цінності, яка оцінювала цінність кожної пари "стан-дія".

Навчання політиці є більш прямолінійною альтернативою, в якій ми вивчаємо функцію політики, π, яка є прямою картою від кожного стану до найкращої відповідної дії в цьому стані. Подумайте про це як про поведінкову політику: "коли я спостерігаю стан s, найкраще зробити дію a". Наприклад, політика автономного транспортного засобу може ефективно включати щось на кшталт: "якщо я бачу жовте світло і перебуваю за 100 футів від перехрестя, я повинен загальмувати. В іншому випадку, продовжуйте рухатися вперед".

Отже, ми вивчаємо функцію, яка максимізує очікувану винагороду. Що ми знаємо, що дійсно добре навчається складних функцій? Глибокі нейронні мережі!

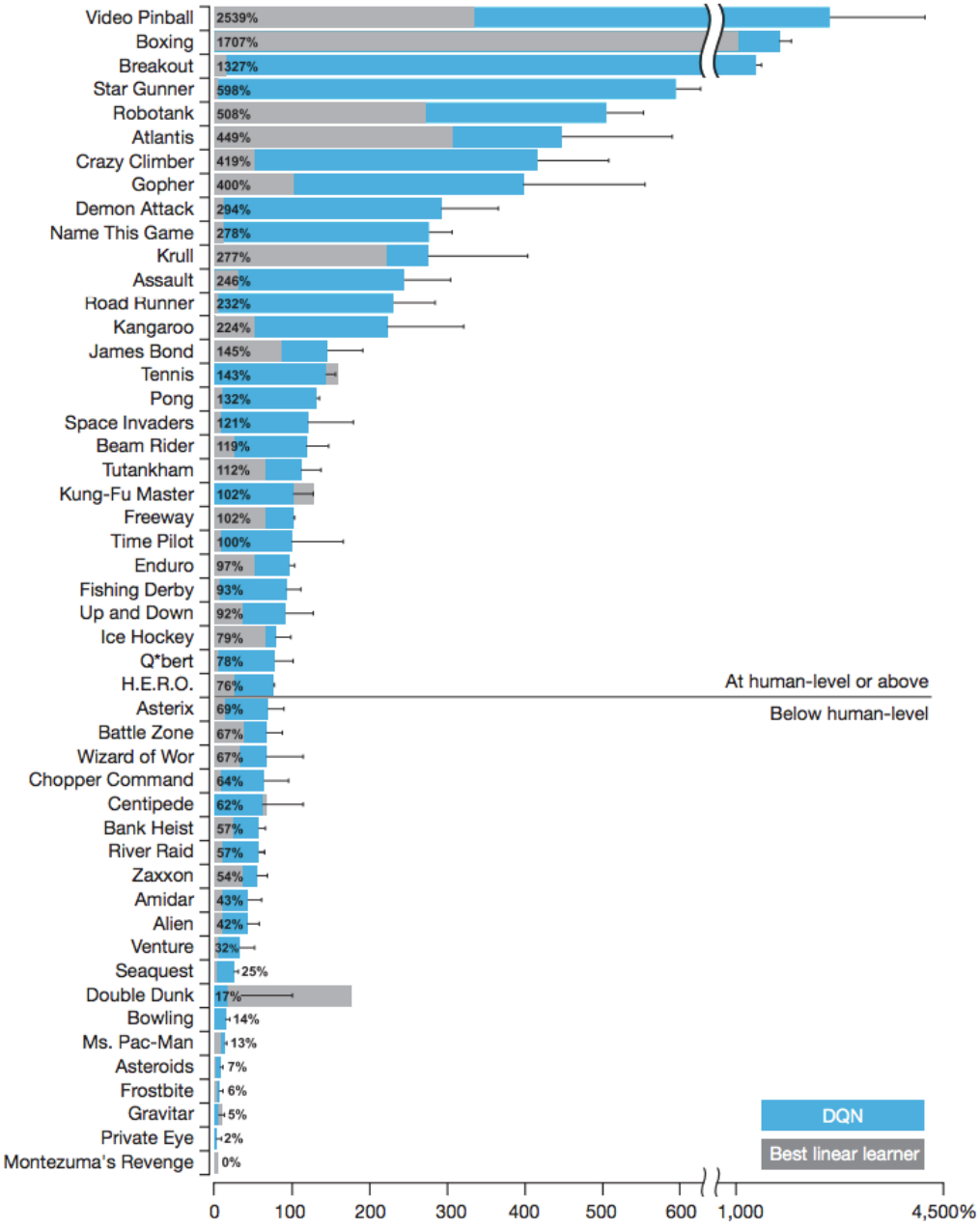
Стаття Андрія Карпатія "Понг з пікселів" надає чудовий приклад використання глибокого навчання з підкріпленням для вивчення політики для

гри Atari Pong, яка приймає необроблені пікселі з гри як вхідні дані (стан) і виводить ймовірність переміщення весла вгору або вниз (дія).



У мережі з градієнтом політики агент вивчає оптимальну політику, коригуючи свої ваги за допомогою градієнтного спуску на основі сигналів винагороди з навколишнього середовища.

DQN, A3C та досягнення в глибокому RL



Щоб допомогти вам сформувати певну інтуїцію щодо того, як відбувається прогрес у дослідженнях RL, наведемо кілька прикладів вдосконалення спроб нелінійних апроксиматорів Q-функції, які можуть покращити продуктивність і стабільність:

- Повторення досвіду, яке навчається шляхом рандомізації довшої послідовності попередніх спостережень і відповідної винагороди, щоб уникнути надмірного пристосування до нещодавнього досвіду. Ця ідея натхненна біологічним мозком: щури, які проходять лабіринти, наприклад, "відтворюють" патерни нейронної активності під час сну, щоб оптимізувати майбутню поведінку в лабіринті.
- Рекурентні нейронні мережі (RNN), що доповнюють DQN. Коли агент може бачити лише своє безпосереднє оточення (наприклад, робот-миша бачить лише певний сегмент лабіринту, тоді як з висоти пташиного польоту можна побачити весь лабіринт), агент повинен запам'ятовувати загальну картину, щоб пам'ятати, де які речі знаходяться.

Це схоже на те, як людські немовлята розвивають постійність об'єктів, щоб знати, що речі існують, навіть якщо вони залишають поле зору дитини. RNN є "рекурентними", тобто вони дозволяють інформації зберігатися на довгостроковій основі. У 2016 році, всього через рік після публікації статті про DQN, DeepMind представив ще один алгоритм під назвою Asynchronous Advantage Actor-Critic (A3C), який перевершив найсучасніші показники в іграх на Atari після тренування вдвічі коротшого часу (Mnih et al., 2016).

A3C - це алгоритм актор-критик, який поєднує в собі найкраще з обох підходів, які ми розглядали раніше: він використовує актора (політичну мережу, яка вирішує, як діяти) і критика (Q-мережу, яка вирішує, наскільки цінними є речі). Артур Джуліані (Arthur Juliani) чудово описав, як саме працює A3C. Зараз A3C є стартовим агентом OpenAI Universe Starter Agent.

З тих пір відбулася незліченна кількість захоплюючих проривів - від винайдення ШІ власної мови до навчання себе ходити по різноманітних місцевостях. Ця серія статей лише дряпає поверхню переднього краю RL, але, сподіваємося, вона послужить відправною точкою для подальших досліджень!

Питання інтелектуальності

Основна проблема полягає в тому, що питання "коли машини стануть розумнішими за нас і поневолять усіх?" від самого початку є хибним. У ньому занадто багато прихованих умов.

Ми говоримо "стануть розумнішими за нас", ніби маємо на увазі, що існує певна уніфікована шкала інтелекту. На вершині якої знаходиться людина, трохи нижче - собаки, а в самому низу вештаються дурні голуби.

Це неправильно.

Якби це було так, то кожна людина мала б у всьому перевершувати тварин, але це не так. Середньостатистична білка може запам'ятати тисячу схованок з горіхами - я навіть не можу згадати, де лежать мої ключі.

Тож інтелект - це набір різних навичок, а не єдина вимірювана величина? Або запам'ятовування місць, де заховані горіхи, не входить до інтелекту?

Ще більш цікаве для мене питання - чому ми вважаємо, що можливості людського мозку обмежені? В інтернеті є багато популярних графіків, де технологічний прогрес зображений у вигляді експоненти, а людські можливості є постійними. Але чи так це?

Гаразд, помножте 1680 на 950 прямо зараз в думках. Я знаю, що ви навіть не спробуєте. Але дай вам калькулятор - ви зробите це за дві секунди. Чи означає це, що калькулятор щойно розширив можливості вашого мозку?

Якщо так, то чи можу я продовжувати розширювати їх за допомогою інших машин? Наприклад, використовувати нотатки в телефоні, щоб не запам'ятовувати купу даних? О, схоже, що я це роблю прямо зараз. Я розширюю можливості свого мозку за допомогою машин.

Як вони працюють?

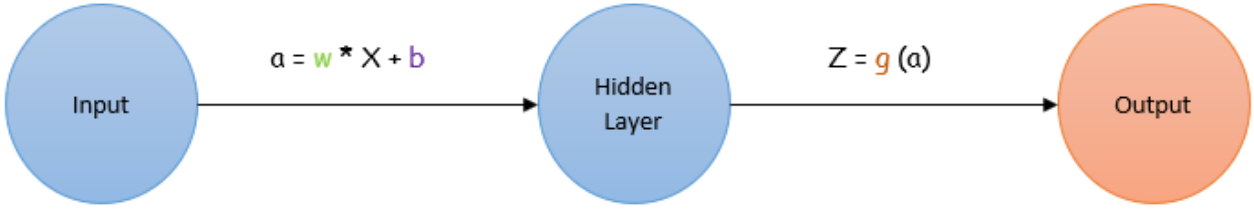
Ванільні ШНМ (штучні нейронні мережі) можна краще зрозуміти, розбивши їх на фундаментальні компоненти:

- Механізм прямого поширення
- Оптимізація ваги (за допомогою зворотного поширення)

Механізм прямого поширення

Цей компонент відповідає за те, щоб наша модель могла робити прогнози. У своїй найпростішій формі механізм прямого поширення починається із застосування лінійної функції до заданого входу, а потім використовує нелінійну функцію, яка називається "активація", для перетворення результату в бажаний вихід.

Давайте візуалізуємо ці операції, розглянувши просту модель, що складається з одного набору входів, одного прихованого нейрона і одного вихідного нейрона:



- X (вхідні дані): деяка матриця, що представляє дані, на яких ви хочете навчити мережу.
- W (ваги): деяка матриця, яку можна розглядати як нахил/швидкість зміни зв'язку між входом і виходом.
- b (зсув): деяка константа, подібна до "y-перехоплення".
- g(a) (активація): нелінійна функція, що застосовується до лінійного виходу "a" (деякі приклади найпопулярніших функцій активації):

- Випрямлена лінійна одиниця (ReLU): max{0, x}
- Сигмоїд: 1 / (1 + e^-x)
- Гіперболічний тангенс: tanh(x)

Коли ми маємо справу з даними, що складаються з декількох ознак, нам зазвичай потрібно більше прихованих шарів і нейронів, але ми все одно застосовуємо той самий порядок операцій, що описаний вище. Єдина відмінність полягає в тому, що наша модель стає значно складнішою, що в кінцевому підсумку ускладнює її навчання.

Оптимізація ваги

Як ми знаємо, що передбачення, зроблені механізмом прямого поширення, насправді правильні?

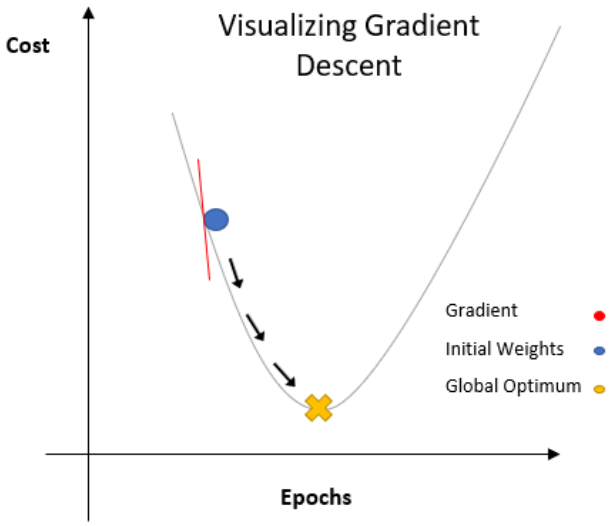
ШНМ навчаються краще наближати цільову змінну шляхом оптимізації вагових матриць, пов'язаних з предикторами. Оптимізація ваг моделі є набагато менш інтуїтивно зрозумілим процесом порівняно з компонентом прямого поширення, тому, не заглиблюючись у сувору математику, я представлю концептуальне розуміння, необхідне для подальшого розуміння.

Зворотне поширення - це вдосконалений математичний алгоритм, який поєднує правило ланцюга та часткові похідні відповідних змінних, щоб обчислити градієнт вагової матриці.

Тож як нам оптимізувати ваги тепер, коли ми маємо градієнт?

Стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD) використовує алгоритм зворотного поширення для налаштування ваг моделі шляхом мінімізації функції вартості при заданій швидкості збіжності (швидкості навчання) протягом обраної кількості навчальних ітерацій (епох). Оскільки градієнт є постійною величиною, то в кожній епосі ми змінюємо ваги до тих пір, поки вони не збігаються до глобального оптимального значення.

Сподіваємось, цей графік допоможе візуалізувати, чого прагне SGD протягом кожної епохи.



Тепер, коли у нас є необхідні базові знання, ми можемо нарешті почати будувати модель!

Я покажу простий посібник з побудови "ванільної" ШНМ з нуля (без використання будь-яких вбудованих модулів). Тип моделі, яку ми будемо будувати, буде простим регресором, оскільки змінні предиктора і відгуку мають числовий тип.

Чому з нуля?

Колись я ставив собі таке ж питання, але коли мені довелося написати власний код з нуля, я помітив, що почав розуміти базові концепції на набагато глибшому рівні (і це було весело).

Архітектура моделі

ШНМ, яку ми будуватимемо, складатиметься з одного набору входів, одного прихованого шару з 5 нейронів та кінцевого вихідного шару. Давайте розглянемо будівельні блоки, необхідні для побудови нашої моделі:

- Вхідні дані (X): 60 імітованих випадкових величин з неперервного рівномірного розподілу на інтервалі [-2, 2].
- Відгук (Y): деяка тригонометрична функція f(X)) = cos(2x +1)
- Функція активації: сигмоїда = 1 / (1 + e^-x)

- Вагові матриці (w1, w2): наші вагові матриці будуть випадковим чином згенеровані з неперервного рівномірного розподілу на [0, 1]. Існує дві первинні вагові матриці, які слід ініціалізувати:
- - Приховані ваги шарів: для кожного шару l приховані ваги повинні мати розмірність { N * (p +1) }, де N - кількість нейронів у даному шарі, а p - кількість різних предикторів у X.
- - Output Layer Weights: перетворює результати з прихованих шарів у відповідні розмірності: {dim(Y) * (N + 1)}, де dim(Y) - розмірність цільової змінної (у нашому випадку 1).
- Функція вартості: ми використаємо суму квадратів помилок (SSE) для оцінки якості прогнозування нашої моделі (квадратична вартість).

```
# Predictor variable as a set of uniform rv's over the interval [-2, 2]
x <- runif(60, min=-2, max=2)
# Response variable
y <- (cos(2*x + 1))
# Initialize the number of hidden neurons
hidden_neurons = 5
# Activation Function
sigmoid <- function(x) {     z = (1 / (1 + exp(-x)))    return(z)   }
# Randomly initializing the weights as i.i.d. N(0,1) rv's...
w1 = matrix(rnorm(2*hidden_neurons), nrow=hidden_neurons, ncol=2)
w2 = matrix(rnorm(hidden_neurons + 1), nrow=1, ncol=(hidden_neurons+1))
# Function for computing model error
modelError <- function(x, y, w1, w2, activation) {
# obtaining predictions
preds <- feedForward(x, w1, w2, activation)
# Calculating the error
SSE <- sum((y - preds) ** 2)
return (SSE)
}
```

Механізм прямого поширення

Тепер, коли у нас є змінні, ми визначаємо нашу власну функцію, яку ми будемо використовувати для прогнозування вхідного вектора.

```
# Function to obtain predicted outputs
feedForward <- function(x, w1, w2, activation) {
output <- rep(0, length(x))
for (i in 1:length(x)) {
a1 = w1 %*% matrix(rbind(1, x[i]), ncol=1)
z1 = activation(a1)
a2 = w2 %*% matrix(rbind(1, z1), ncol=1)
output[i] = a2  }
return(output)}
```

Зворотне розповсюдження

Спочатку визначимо похідну функції активації, необхідну для виконання зворотного розповсюдження:

Defining the derivative of our activation function needed for computing the gradient

derivativeActivation <- function(x) {
g = (sigmoid(x) * (1 - sigmoid(x)))
return(g)
}

Давайте уважно розглянемо необхідні операції, які потрібно виконати для того, щоб здійснити зворотне розмноження.

```
#Function for computing the gradients
backPropagation <- function(x, y, w1, w2, activation, derivativeActivation) {

  preds <- feedForward(x, w1, w2, activation) #predicted values

  derivCost <- -2*(y - preds) #Derivative of the cost function (first term)

  dW1 <- matrix(0,ncol=2,nrow=nrow(w1)) #Gradient for w1
  dW2 <- matrix(rep(0,length(x)*(dim(w2)[2])),nrow=length(x)) #Gradient matrix for w2

  # Computing the Gradient for W2
  for (i in 1:length(x)) {

    a1 = w1 %*% matrix(rbind(1, x[i]), ncol=1)
    da2dW2 = matrix(rbind(1, activation(a1)), nrow=1)
    dW2[i,] = derivCost[i] * da2dW2

  }

  #Computing the gradient for W1
  for (i in 1:length(x)) {

    a1 = w1 %*% matrix(rbind(1, x[i]), ncol=1)
    da2da1 = derivativeActivation(a1) * matrix(w2[,-1], ncol=1)
    da2dW1 = da2da1 %*% matrix(rbind(1, x[i]), nrow=1)

    dW1 = dW1 + derivCost[i] * da2dW1
  }

  # Storing gradients for w1, w2 in a list
  gradient <- list(dW1, colSums(dW2))

  return (gradient)
}
```

Розбір коду:

- predictions: прогнози моделі, необхідні для мінімізації функції вартості
- deriv_cost: оскільки ми використовуємо SSE як міру похибки моделі (квадратична вартість), взяття похідної від прогнозованих значень дає результат, зазначений вище
- (dW1, dW2): спочатку порожні матриці, які ми використовуємо для зберігання градієнтів вагових матриць.

Виконання зворотного поширення далеко не інтуїтивно зрозуміле, і це обчислення лише для одного набору вхідних даних, що проходять через прихований шар. Коли до мережі додається більше шарів, обчислення градієнту стає дуже дорогим в обчислювальному плані.

Навчання моделі (SGD)

Якщо ви помітили, ми ще не використали жодної з функцій, визначених у блоках коду. Тут ми зв'яжемо все разом, викликавши всі попередні функції для навчання моделі.

Ми визначаємо функцію Stochastic Gradient Descent для оптимізації ваг і відстеження помилки навчання.

По суті, ми будемо коригувати матриці ваг на кожній ітерації відповідно до обраної швидкості навчання (зазвичай це плаваюча величина від 0 до 1), в той же час ми будемо зберігати значення SSE в порожній матриці на кожній епосі.

```
# Defining our Stochastic Gradient Descent algorithm which will adjust our weight matrices
SGD <- function(x, y, w1, w2, activation, derivative, learnRate, epochs) {

  SSEvec <- rep(NA, epochs) #Empty array to store SSE values after each epoch
  SSEvec[1] = modelError(x, y, w1, w2, activation)

  for (j in 1:epochs) {

    for (i in 1:length(x)) {

      gradient <- backPropagation(x[i], y[i], w1, w2, activation, derivative)


      #Adjusting model parameters for a given number of epochs
      w1 <- w1 - learnRate * gradient[[1]]

      w2 <- w2 - learnRate * gradient[[2]]

    }

    SSEvec[j+1] <- modelError(x, y, w1, w2, activation)#Storing SSE values after each iteration

  }

  #Beta vector holding model parameters
  B <- list(w1,w2)
  result <- list(B, SSEvec)

  return(result)

}
```

Нарешті, ми отримуємо параметри нашої моделі за допомогою виклику функції SGD.

Running SGD function to obtain our optimized mode and parameters

model <- SGD(x, y, w1, w2, sigmoid, derivativeActivation, learnRate = 0.01, epochs = 200)

Графіки оцінювання моделей

Помилка навчання

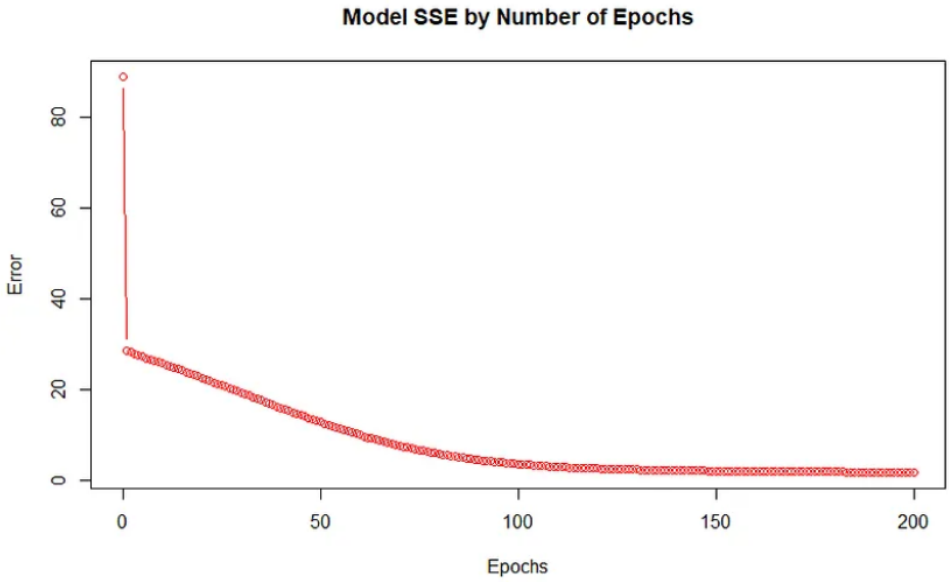
Ось графік залежності помилки навчання від кількості епох (200), ми бачимо значне зростання продуктивності на початку, за яким слідує постійне зменшення помилки моделі на кожній наступній ітерації.

Obtaining our adjusted SSE's for each epoch..

SSE <- model[[2]]

#Plotting the SSE from each epoch vs # epochs

plot(seq(0,(length(model[[2]])-1),1),SSE, main="Model SSE by Number of Epochs",xlab = "Epochs", ylab = "Error",col="red",type = "b")



Примітка: причина, чому ми не показуємо графік помилок тесту, полягає в тому, що у нас немає тестового набору, оскільки наші предиктори змодельовані на основі випадкового рівномірного розподілу, а не на основі реального набору даних.

Фактичні та прогнозовані значення

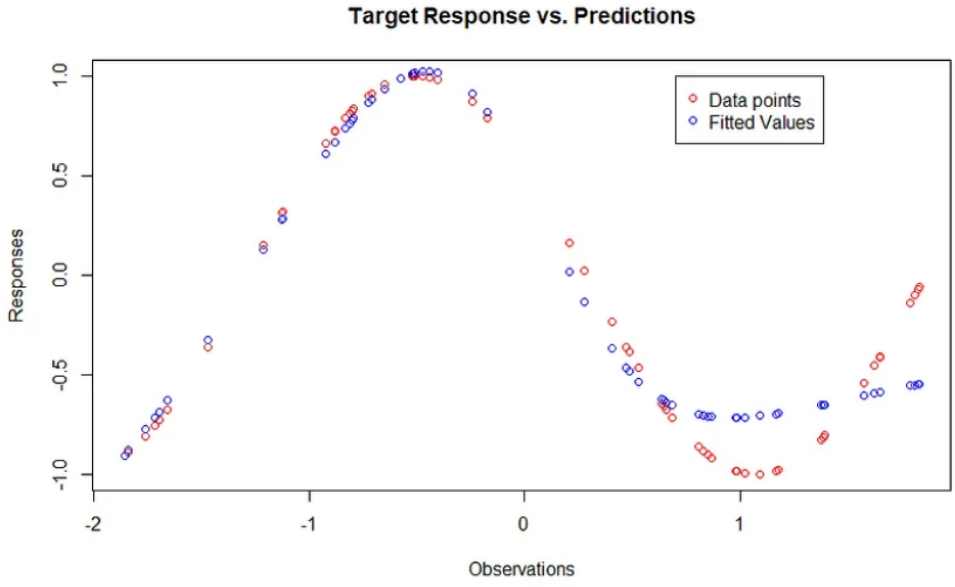
Давайте подивимось, як наша модель прогнозує цільову змінну, побудувавши графік вихідних значень відповідей у порівнянні з прогнозованими:

#Plotting training data against our model predictions

plot(x,y, main = "Target Response vs. Predictions",xlab="Observations", ylab="Responses", col="red")

lines(x, y_pred, col="blue", type="p")

legend(y=1,x=0.7, legend=c('Data points','Fitted Values'), col=c('red','blue') , lty=c(0,0), pch=c(1,1))



Непогано! Наша модель досить добре пояснює змінну відгуку, продуктивність дрейфує до правого хвоста, але ознак надмірного пристосування не спостерігається.

Створення штучних даних

```
Вектори предикторів
X <- matrix(c(
  0,0,1,1,
  0,1,1,0,
  1,0,1,1,
  1,1,1,0,
  1,0,0,1,
  0,1,0,1
),
ncol = 4,
byrow = TRUE
)
```

```
Спостережувані результати
y <- matrix(c(0, 0, 1,
        1, 0, 0,
        0, 1, 0,
        0, 1, 0,
        1, 0, 0,
        0, 0, 1),
        nrow = 3,
        byrow = FALSE
)
```

Згенеруйте випадковий вектор зі значеннями від 0 до 1, який буде використано як початкові ваги

n_nodes1 = 5 n_out = 3 rand_vector <- runif(nrow(X) * n_nodes1)

Перетворіть вектор вище в матрицю

rand_matrix <- matrix(rand_vector, nrow = nrow(X), ncol = n_nodes1, byrow = TRUE)

Цей список зберігає стан нашої нейронної мережі під час її навчання

```
my_nn <- list(
  # predictor variables
  input = X,
  # weights for layer 1
  weights1 = rand_matrix,
  # weights for layer 2
  weights2 = matrix(rep(0, times = n_out * ncol(rand_matrix)), ncol = n_out),
  # actual observed
  y = y,
  # matrix to store the predicted outcomes
  output = matrix(
    runif(length(y)),
    ncol = n_out
  )
)
```

Функції активації та похідні

Сигмоїдна функція активації. Функція активації перетворює входи шару на виходи.

```
sigmoid <- function(x) {
  1.0 / (1.0 + exp(-x))
}
```

Похідна сигмоїдної функції активації

```
sigmoid_derivative <- function(x) {
  x * (1.0 - x)
}
```

Функція активації Softmax

```
softM <- function(x) {
  nDim = length(x)
  res = rep(0, nDim)
  res = matrix(res, nrow = nrow(x), byrow = FALSE)
  for (j in 1:ncol(x)) {
    for (i in 1:nrow(x)) {
    xDiff = x[,j] - x[i,j]
    sumExpVect = sum(exp(xDiff))
    res[i,j] = 1/sumExpVect
    }
  }
  return(res)
}
```

Ми включимо похідну від Softmax як частину похідної від Cross-Entropy, тому не будемо визначати її окремо.

Функція втрат використовується для визначення якості підгонки моделі. Ми використовуємо функцію перехресної ентропії як функцію втрат.

```
loss_function <- function(nn) {
  -sum(nn$y * log(nn$output+0.0000005) ) # the additional constant here is to avoid running into log(0), which is undefined.
}
```

Для того, щоб мінімізувати функцію втрат, ми виконуємо пряме та зворотне розповсюдження.

Пряме та зворотне поширення

Пряме поширення застосовує функцію активації до шарів і виробляє передбачуваний результат.

```
feedforward <- function(nn) {
  nn$layer1 <- sigmoid(t(nn$weights1) %*% nn$input)
  nn$output <- softM(t(nn$weights2) %*% nn$layer1)
  nn
}
```

При зворотному поширенні береться прогнозований результат, отриманий в результаті кроку прямого поширення, і коригуються ваги шарів, щоб зменшити функцію втрат.

```
backprop <- function(nn) {
  # application of the chain rule to find derivative of the loss function with
  # respect to weights2 and weights1
  d_weights2 <- ( (nn$y - nn$output) %*% t(nn$layer1) )

  d_weights1 <- t(nn$y - nn$output) %*%  t(nn$weights2)
  d_weights1 <- (t(d_weights1) * sigmoid_derivative(nn$layer1)) %*% t(nn$input)

  # update the weights using the derivative (slope) of the loss function
  nn$weights1 <- nn$weights1 + t(d_weights1)
```

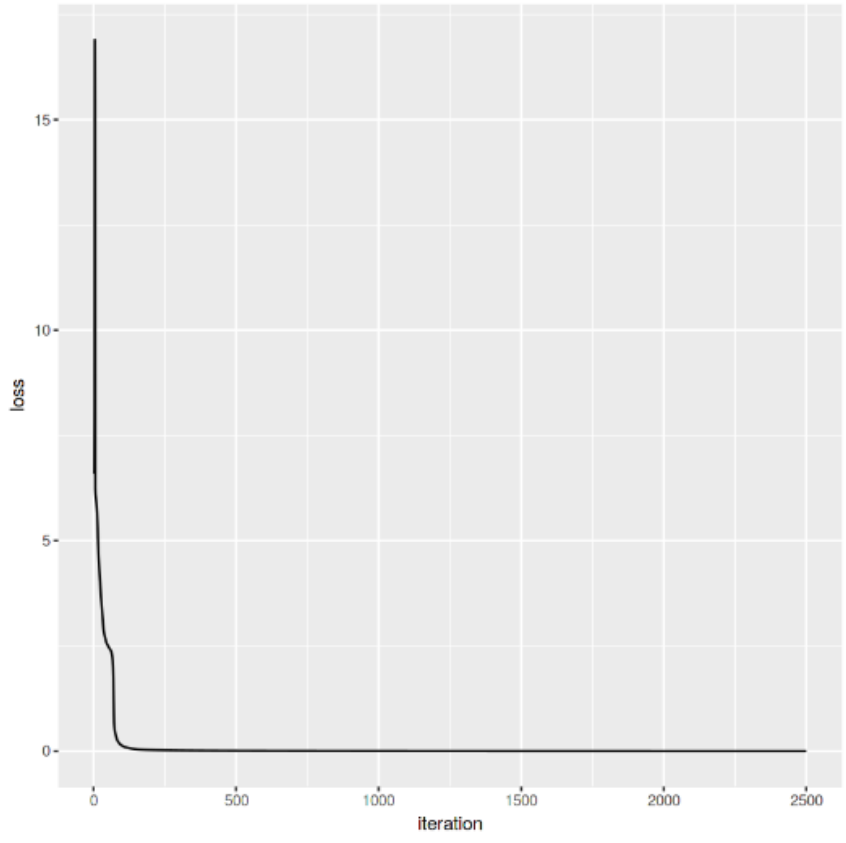
```
  nn$weights2 <- nn$weights2 + t(d_weights2)

  nn
}
```

Тепер ми готові навчати нашу модель. У процесі навчання неодноразово викликаються функції feedforward() і backprop(), щоб зменшити функцію втрат.

Тренування

```
Кількість разів для виконання прямого та зворотного розмноження
n <- 2500
фрейм даних для зберігання результатів функції втрат
loss_df <- data.frame(
  iteration = 1:n,
  loss = vector("numeric", length = n)
)
Поєднання прямого та зворотного поширення
for (i in seq_len(n)) {
  my_nn <- feedforward(my_nn)
  my_nn <- backprop(my_nn)
  # store the result of the loss function.  We will plot this later
  loss_df$loss[i] <- loss_function(my_nn)
}
```

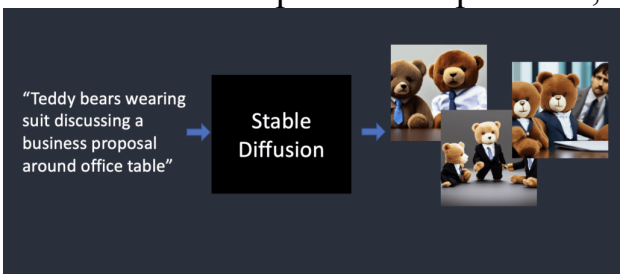
На графіку нижче показано результат функції втрат під час навчання моделі. Метою навчання моделі є мінімізація результату функції втрат (вісь Y), і ми бачимо, що в міру проходження ітерацій (вісь X) результат функції втрат наближається до нуля.



ТЕМА 9. Дифузійні нейронні мережі

Що може зробити Stable Diffusion?

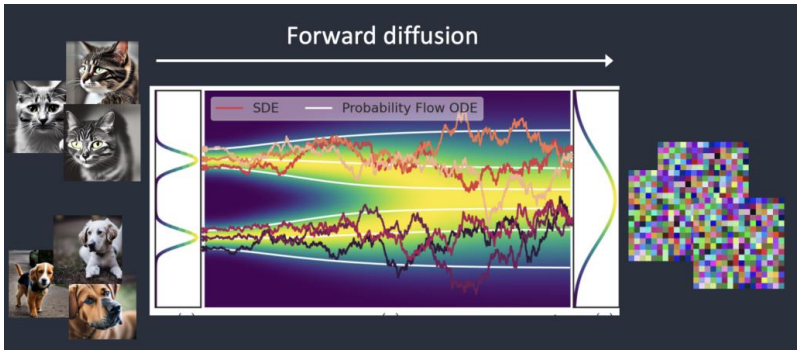
У найпростішій формі Stable Diffusion - це модель "текст-зображення". Дайте йому текстовий запит. Він поверне ШІ-зображення, що відповідає тексту.



Стабільна дифузія належить до класу моделей глибокого навчання, які називаються дифузійними моделями. Це генеративні моделі, тобто вони призначені для генерації нових даних, подібних до тих, які вони бачили під час навчання. У випадку зі Stable Diffusion даними є зображення.

Чому вона називається дифузійною моделлю? Тому що її математика дуже схожа на дифузію у фізиці. Пояснимо ідею.

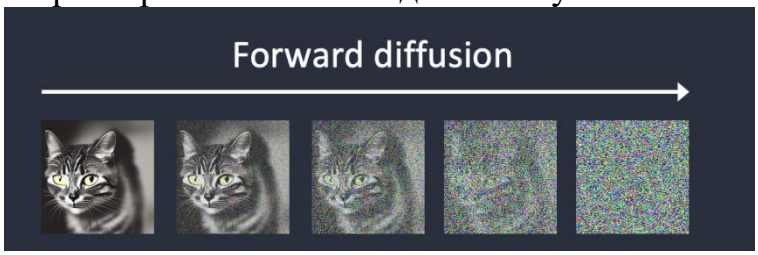
Скажімо, я навчив дифузійну модель лише на двох типах зображень: котиках і собачках. На малюнку нижче два піки зліва представляють групи зображень котів і собак.



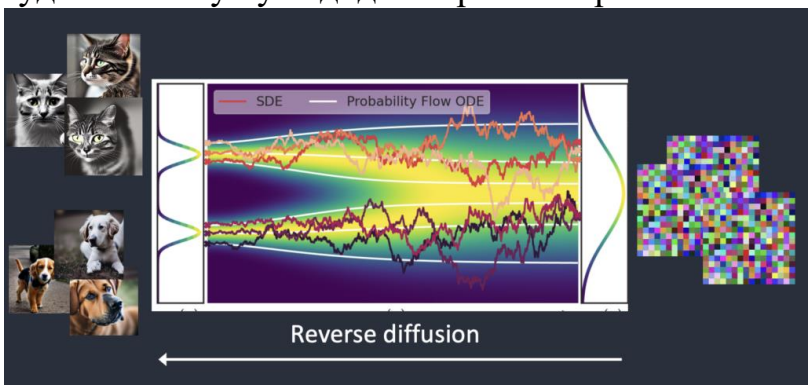
Процес прямої дифузії додає шум до навчального зображення, поступово перетворюючи його на нехарактерне шумове зображення. Процес прямого поширення перетворить будь-яке зображення кота чи собаки на шумове зображення. Зрештою, ви не зможете визначити, чи це спочатку собака, чи кішка (це важливо).

Це як якщо б крапля чорнила впала в склянку з водою. Крапля чорнила дифундує у воді. Через кілька хвилин вона хаотично розподіляється по всій воді. Ви вже не можете сказати, де вона спочатку впала - в центрі чи біля краю.

Нижче наведено приклад зображення, що зазнає прямої дифузії. Зображення кота перетворюється на випадковий шум.



Тепер настає найцікавіша частина. Що, якщо ми зможемо повернути дифузію назад? Як відтворення відео задом наперед. Повернутися назад у часі. Ми побачимо, куди спочатку була додана крапля чорнила.



Починаючи з шумного, безглуздого зображення, зворотна дифузія відновлює зображення кота або собаки. Це і є основна ідея.

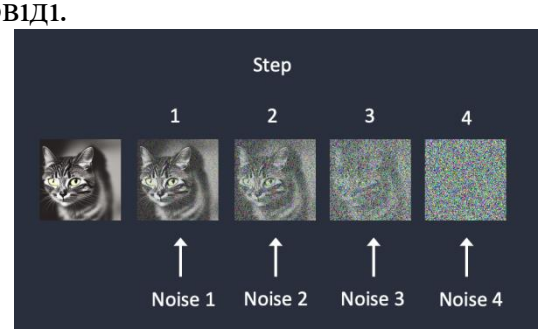
Технічно, кожен процес дифузії складається з двох частин: (1) дрейф і (2) випадковий рух. Зворотна дифузія дрейфує до зображення кота або собаки, але не до чогось середнього. Ось чому результатом може бути або кіт, або собака.

Як відбувається навчання

Ідея зворотної дифузії, безсумнівно, розумна та елегантна. Але питання на мільйон доларів: "Як це можна зробити?".

Щоб повернути дифузію назад, нам потрібно знати, скільки шуму додано до зображення. Відповідь полягає в тому, щоб навчити модель нейронної мережі передбачати додавання шуму. Вона називається предиктором шуму в Stable Diffusion. Це модель U-Net. Навчання відбувається наступним чином.

1. Виберіть навчальне зображення, наприклад, фотографію кота.
2. Згенеруйте випадкове шумове зображення.
3. Зіпсуйте навчальне зображення, додаючи це зашумлене зображення до певної кількості кроків.
4. Навчіть предиктор шуму повідомляти нам, скільки шуму було додано. Це робиться шляхом налаштування вагових коефіцієнтів і показу правильної відповіді.



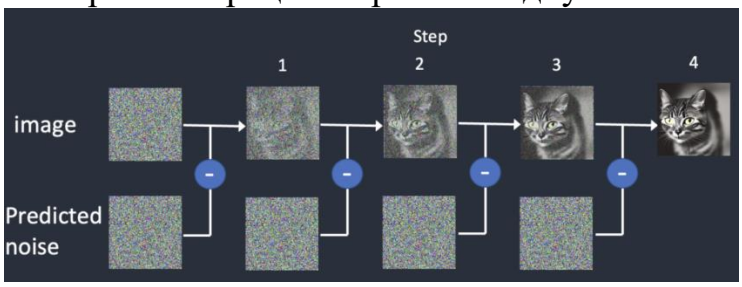
Після навчання ми маємо предиктор шуму, здатний оцінювати шум, доданий до зображення.

Зворотна дифузія

Тепер у нас є предиктор шуму. Як його використовувати?

Спочатку ми генеруємо абсолютно випадкове зображення і просимо предиктор шуму повідомити нам про рівень шуму. Потім ми віднімаємо цей передбачуваний шум від оригінального зображення. Повторіть цей процес кілька разів. Ви отримаєте зображення кота або собаки.

Ви можете помітити, що ми не можемо контролювати створення зображення кота чи собаки. Ми розглянемо це питання, коли будемо говорити про обумовленість. Наразі генерація зображень відбувається без умов.



Модель Stable Diffusion

Те, про що ми щойно говорили, НЕ є тим, як працює стабільна дифузія! Причина в тому, що вищезгаданий процес дифузії відбувається у просторі зображення. Він дуже, дуже повільний в обчислювальному плані. Ви не зможете запустити його на одному графічному процесорі, не кажучи вже про паршивий графічний процесор на вашому ноутбуці.

Простір зображення величезний. Подумайте про це: зображення 512×512 з трьома колірними каналами (червоним, зеленим і синім) - це 786,432-вимірний простір! (Стільки значень потрібно вказати для ОДНОГО зображення).

Дифузійні моделі, такі як Imagen від Google та DALL-E від Open AI, працюють у піксельному просторі. Вони використовують деякі трюки, щоб зробити модель швидшою, але цього все одно недостатньо.

Модель латентної дифузії

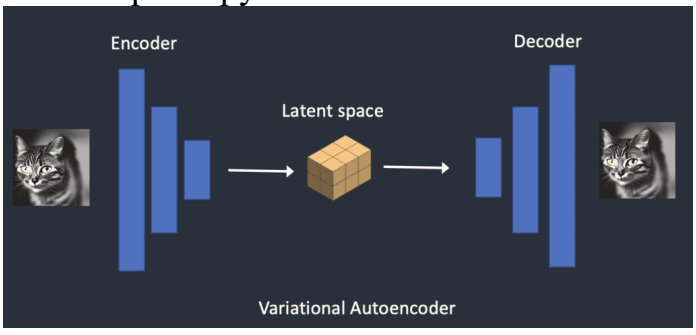
Стабільна дифузія призначена для вирішення проблеми швидкості. Ось як.

Стабільна дифузія - це модель прихованої дифузії. Замість того, щоб працювати у високорозмірному просторі зображення, вона спочатку стискає зображення у прихований простір. Прихований простір у 48 разів менший, тому він отримує перевагу від обробки набагато меншої кількості чисел. Ось чому він набагато швидший.

Варіаційний автокодер

Це робиться за допомогою техніки, яка називається варіаційним автокодером. Так, це саме те, чим є VAE-файли, але я поясню це пізніше.

Нейронна мережа варіаційного автокодування (VAE) складається з двох частин: (1) кодера і (2) декодера. Кодер стискає зображення до більш низьковимірного представлення в латентному просторі. Декодер відновлює зображення з латентного простору.



Прихований простір моделі стабільної дифузії має розмір 4x64x64, що в 48 разів менше, ніж піксельний простір зображення. Всі прямі та зворотні дифузії, про які ми говорили, насправді виконуються в латентному просторі.

Тому під час навчання, замість того, щоб генерувати зашумлене зображення, він генерує випадковий тензор у латентному просторі (латентний шум). Замість того, щоб спотворювати зображення шумом, він спотворює представлення зображення в латентному просторі за допомогою латентного шуму. Причиною цього є те, що це набагато швидше, оскільки латентний простір менший.

Роздільна здатність зображення

Роздільна здатність зображення відображається у розмірі тензора прихованого зображення. Розмір тензора прихованого зображення становить 4x64x64 лише для зображень 512×512. Для портретного зображення 768×512 він становить 4x96x64. Ось чому для створення більшого зображення потрібно більше оперативної пам'яті.

Оскільки Stable Diffusion v1 налаштовано на зображення 512×512, генерування зображень більших за 512×512 може призвести до появи дублікатів об'єктів, наприклад, сумнозвісних двох голів.

Чому можливий прихований простір?

Ви можете здивуватися, чому VAE може стискати зображення до набагато меншого прихованого простору без втрати інформації. Причина, як це не дивно, полягає в тому, що природні зображення не є випадковими. Вони мають високу регулярність: Обличчя має певне просторове співвідношення між очима, носом, щокою і ротом. Собака має 4 лапи і певну форму.

Іншими словами, висока розмірність зображень є артефактною. Природні зображення можна легко стиснути до набагато меншого прихованого простору без втрати інформації. У машинному навчанні це називається гіпотезою множини.

Гіпотеза множини стверджує, що багато багатовимірних наборів даних, які зустрічаються в реальному світі, насправді лежать уздовж низьковимірних латентних множин всередині цього багатовимірного простору. Як наслідок гіпотези множинності, багато наборів даних, які спочатку здаються такими, що потребують багато змінних для опису, насправді можуть бути описані порівняно невеликою кількістю змінних, подібно до локальної системи координат базової множини. Припускається, що цей принцип лежить в основі ефективності алгоритмів машинного навчання при описі багатовимірних наборів даних, враховуючи кілька загальних особливостей.

Гіпотеза множини пов'язана з ефективністю нелінійних методів зменшення розмірності в машинному навчанні. Багато методів зменшення розмірності припускають, що дані лежать уздовж низьковимірного підмноговиду, наприклад, ліплення множини, вирівнювання множини та регуляризація множини.

Основні наслідки цієї гіпотези полягають у тому, що

- Моделі машинного навчання повинні відповідати лише відносно простим, низьковимірним, високоструктурованим підпросторам у своєму потенційному вхідному просторі (латентні многовиди).
- В межах однієї з цих множин завжди можна інтерполювати між двома вхідними даними, тобто перетворити одну в іншу за допомогою безперервного шляху, вздовж якого всі точки потрапляють на множину.

Здатність до інтерполяції між вибірками є ключем до узагальнення в глибокому навчанні.

Зворотна дифузія в латентному просторі

Ось як працює латентна зворотна дифузія в Stable Diffusion.

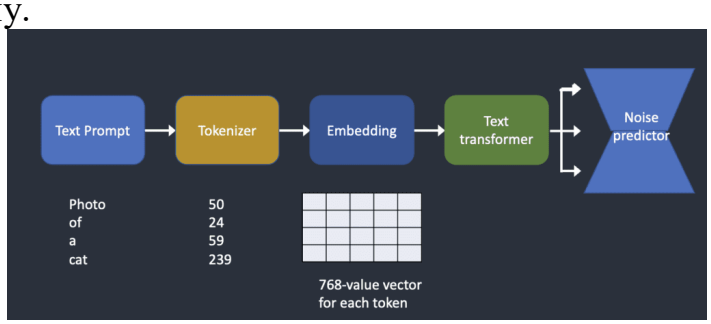
1. Генерується випадкова матриця латентного простору.
2. Предиктор шуму оцінює шум латентної матриці.
3. Потім оцінений шум віднімається від латентної матриці.
4. Кроки 2 і 3 повторюються до певного кроку дискретизації.
5. Декодер VAE перетворює латентну матрицю в кінцеве зображення.

Наше розуміння неповне: Де текстова підказка входить у зображення? Без нього Stable Diffusion не є моделлю перетворення тексту в зображення. Ви отримаєте зображення кота або собаки, але не зможете ним керувати.

Тут вступає в дію кондиціонування. Мета обумовленості - керувати предиктором шуму так, щоб передбачений шум давав нам те, що ми хочемо, після віднімання від зображення.

Обробка тексту (перетворення тексту в зображення)

Нижче наведено огляд того, як текстова підказка обробляється і подається в предиктор шуму. Токенізатор спочатку перетворює кожне слово в підказці на число, яке називається токеном. Потім кожен токен перетворюється на 768-значний вектор, який називається вбудовуванням. Потім вбудовування обробляються текстовим трансформатором і готові до використання предиктором шуму.



Токенізатор

Спочатку текстові підказки токенізуються за допомогою CLIP-токенізатора. CLIP - це модель глибокого навчання, розроблена Open AI для створення текстових описів будь-яких зображень. Stable Diffusion v1 використовує токенізатор CLIP.

Модель CLIP була запропонована в роботі Learning Transferable Visual Models From Natural Language Supervision (Навчання переносним візуальним моделям на основі спостереження за природною мовою). CLIP - це

мультимодальна модель зору та мови. Вона може бути використана для подібності зображення і тексту та для класифікації зображень з нульового кадру. CLIP використовує ViT-подібний трансформатор для отримання візуальних ознак і каузальну мовну модель для отримання текстових ознак. Потім текстові та візуальні ознаки проектуються на латентний простір з однаковою розмірністю. Точковий добуток між спроектованим зображенням і текстовими ознаками використовується як аналогічна оцінка.

Щоб подати зображення на кодер Transformer, кожне зображення розбивається на послідовність ділянок фіксованого розміру, що не перетинаються, які потім лінійно вбудовуються. Для представлення цілого зображення додається токен [CLS]. Автори також додають вбудовування абсолютного положення і подають отриману послідовність векторів на стандартний кодер Transformer.

Токенізація - це спосіб розуміння комп'ютером слів. Ми, люди, можемо читати слова, але комп'ютери можуть читати лише числа. Ось чому слова в текстовому запиті спочатку перетворюються на числа.

Токенізатор може токенізувати лише ті слова, які він бачив під час навчання. Наприклад, у моделі CLIP є слова "dream" і "beach", але немає "dreambeach". Токенізатор розбиває слово "dreambeach" на два токени "dream" і "beach". Отже, одне слово не завжди означає один токен!

Ще один нюанс - символ пробілу також є частиною токена. У наведеному вище випадку фраза "dream beach" створює дві лексеми "dream" і "[пробіл] beach". Ці лексеми відрізняються від лексеми "dreambeach", яка складається з "dream" і "beach" (без пробілу перед beach).

Модель стабільної дифузії обмежена використанням 75 лексем у підказці (тепер ви знаєте, що це не те саме, що 75 слів!).

Вбудовування

Stable diffusion v1 використовує модель ViT-L/14 Clip від Open AI. Вбудовування - це 768-значний вектор. Кожен токен має власний унікальний вектор вбудовування. Вбудовування фіксується моделлю CLIP, яка вивчається під час навчання.

Stable Diffusion v1 відноситься до певної конфігурації архітектури моделі, яка використовує автокодер з фактором дискретизації 8, 860M UNet і текстовий кодер CLIP ViT-L/14 для моделі дифузії. Модель була попередньо навчена на зображеннях 256x256, а потім доналаштована на зображеннях 512x512.

Навіщо нам потрібне вбудовування? Тому що деякі слова тісно пов'язані між собою. Ми хочемо скористатися цією інформацією. Наприклад, вставки man, gentleman і guy майже ідентичні, тому що їх можна використовувати як взаємозамінні. Моне, Мане і Дега малювали в стилі імпресіонізму, але по-різному. Імена мають близькі, але не ідентичні вбудування.

Це те ж саме вбудовування, яке ми обговорювали для запуску стилю за ключовим словом. Вбудовування можуть творити магію. Вчені довели, що знаходження правильних вбудовувань може викликати довільні об'єкти та стилі - це техніка тонкого налаштування, яка називається текстовою інверсією.

На етапі кодування тексту в більшості моделей перетворення тексту в зображення перший етап передбачає перетворення запиту в числове представлення. Зазвичай це робиться шляхом перетворення слів у токени, кожен з яких еквівалентний запису в словнику моделі. Потім ці записи перетворюються на "вбудовування" - безперервне векторне представлення для конкретного токена. Ці вбудовування зазвичай вивчаються в процесі навчання. У цій роботі знайдено нові вбудовування, які представляють конкретні, надані користувачем візуальні концепції. Потім ці вбудовування пов'язуються з новими псевдословами, які можуть бути включені в нові речення, як будь-яке інше слово. У певному сенсі, ми виконуємо інверсію у простір текстових вбудовувань застиглої моделі. Ми називаємо цей процес "текстовою інверсією".

Згодовування вбудовувань предиктору шуму

Перед тим, як потрапити до предиктора шуму, вбудований текст потрібно додатково обробити за допомогою текстового трансформатора. Трансформатор - це універсальний адаптер для обробки. У цьому випадку на його вхід надходять текстові вектори вбудовування, але це може бути і щось інше, наприклад, мітки класів, зображення або карти глибини. Трансформатор не лише здійснює подальшу обробку даних, але й надає механізм для включення різних модальностей кондиціонування.

Перехресна увага

Вихідні дані текстового перетворювача багаторазово використовуються предиктором шуму по всій U-Net. U-Net споживає його за допомогою механізму перехресної уваги. Саме там підказка зустрічається із зображенням.

Візьмемо для прикладу підказку "Чоловік з блакитними очима". Stable Diffusion об'єднує два слова "блакитний" і "очі" разом (самоуважність в межах запиту) так, що генерує чоловіка з блакитними очима, але не чоловіка в блакитній сорочці. Потім він використовує цю інформацію, щоб спрямувати зворотну дифузію на зображення з блакитними очима. (перехресна увага між підказкою та зображенням)

Примітка: Гіпермережа, метод точного налаштування моделей стабільної дифузії, перехоплює мережу перехресної уваги для вставки стилів. Моделі **LoRA** змінюють ваги модуля перехресної уваги для зміни стилів. Той факт, що модифікація цього модуля сама по собі може точно налаштувати модель Стабільної дифузії, говорить про те, наскільки важливим є цей модуль.

Інші зумовлення

Текстова підказка - не єдиний спосіб, яким можна обумовити модель стабільної дифузії.

Як текстове підказка, так і зображення глибини використовуються для налаштування моделі "глибина-зображення".

ControlNet обумовлює предиктор шуму виявленими контурами, людськими позами і т.д., і досягає відмінного контролю над генерацією зображень.

ТЕМА 10. Stable Diffusion. LoRa

LoRA

Моделі LoRA - це невеликі моделі стабільної дифузії, які застосовують незначні зміни до стандартних моделей контрольних точок. Зазвичай вони в 10-100 разів менші, ніж моделі контрольних точок. Це робить їх дуже привабливими для людей, які мають велику колекцію моделей.

LoRA (Low-Rank Adaptation - адаптація низького рангу) - це тренувальна техніка для тонкого налаштування моделей Stable Diffusion.

Але у нас вже є такі тренувальні техніки, як Dreambooth та текстова інверсія. У чому особливість LoRA? LoRA пропонує хороший компроміс між розміром файлу та потужністю навчання. Dreambooth є потужним, але призводить до створення великих файлів моделей (2-7 ГБ). Текстові інверсії крихітні (близько 100 КБ), але ви не можете зробити так багато.

LoRA знаходиться посередині. Розмір його файлів набагато більш керований (2 - 200 МБ), а потужність навчання пристойна.

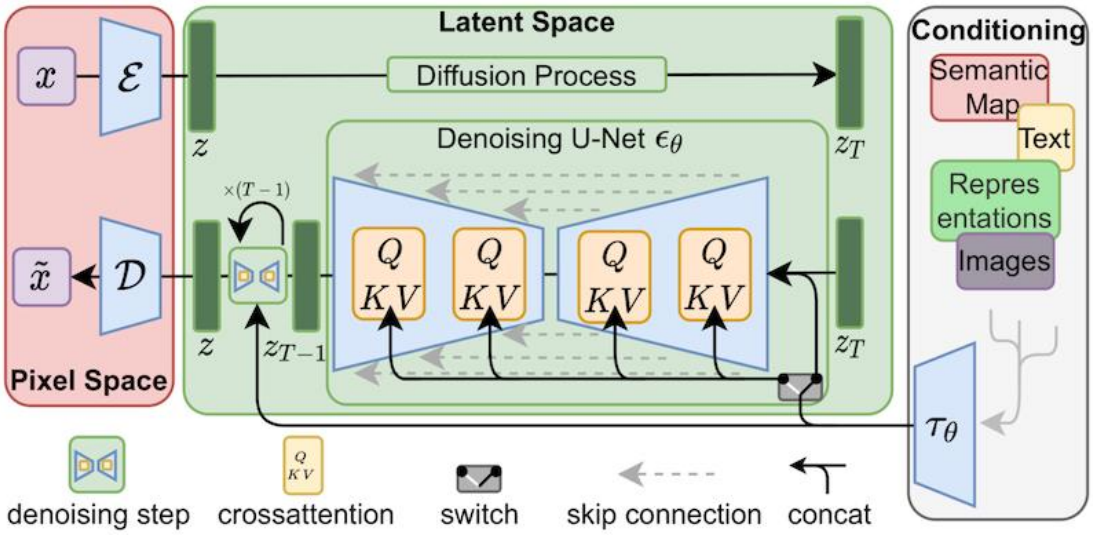
Як і текстову інверсію, модель LoRA не можна використовувати окремо. Вона повинна використовуватися з файлом контрольних точок моделі. LoRA модифікує стилі, застосовуючи невеликі зміни до супровідного файлу моделі.

LoRA - це чудовий спосіб налаштовувати художні моделі AI, не заповнюючи локальне сховище.

Як працює LoRA

LoRA застосовує невеликі зміни до найбільш критичної частини моделей стабільної дифузії: Шари перехресної уваги. Це та частина моделі, де зустрічаються зображення і підказка. Дослідники виявили, що для досягнення хорошого навчання достатньо тонко налаштувати цю частину моделі. Шари перехресної уваги - це жовті частини в архітектурі моделі Stable Diffusion нижче.

«*Важливою парадигмою обробки природної мови є широкомасштабне попереднє навчання на загальних даних предметної області та адаптація до конкретних завдань або доменів. Коли ми попередньо навчаємо більші моделі, повна точна настройка, яка перенавчає всі параметри моделі, стає менш доцільною. На прикладі GPT-3 175B - розгортання незалежних екземплярів доопрацьованих моделей, кожна з яких має 175B параметрів, є надто дорогим задоволенням. Ми пропонуємо низькорангову адаптацію (Low-Rank Adaptation, LoRA), яка заморожує ваги попередньо навченої моделі і вводить матриці декомпозиції рангів, що піддаються навчанню, в кожен рівень архітектури трансформатора, значно зменшуючи кількість параметрів, що піддаються навчанню, для наступних завдань.*»



LoRA точно налаштовує шари перехресної уваги (QKV-частини предиктора шуму U-Net)

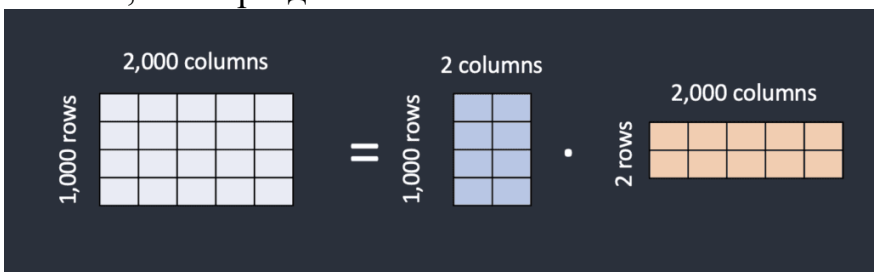
Ваги шару перехресної уваги розташовані в матрицях. Матриці - це просто набір чисел, розташованих у стовпчиках і рядках, як у електронній таблиці Excel. Модель LoRA точно налаштовує модель, додаючи свої ваги до цих матриць.

Як можна зменшити розмір файлів LoRA-моделі, якщо вони повинні зберігати однакову кількість ваг? Хитрість LoRA полягає в тому, що матриця розбивається на дві менші (низькорангові) матриці. Таким чином можна зберігати набагато менше чисел. Проілюструємо це на наступному прикладі.

Припустимо, що в моделі є матриця з 1 000 рядків і 2 000 стовпців. Це 2 000 000 чисел (1 000 x 2 000), які потрібно зберігати у файлі моделі. LoRA розбиває матрицю на матрицю 1,000 на 2 і матрицю 2 на 2,000. Це лише 6,000 чисел (1,000 x 2 + 2 x 2,000), що в 333 рази менше. Ось чому файли LoRA набагато менші.

У цьому прикладі ранг матриць дорівнює 2. Це набагато менше, ніж вихідні розміри, тому вони називаються матрицями низького рангу. Ранг може бути навіть 1.

Але чи є якась шкода від такого трюку? Дослідники виявили, що в перехресних шарах уваги це не сильно впливає на потужність точного налаштування. Отже, все гаразд.



Тепер ви знаєте всю внутрішню механіку стабільної дифузії, давайте розглянемо деякі приклади того, що відбувається під капотом.

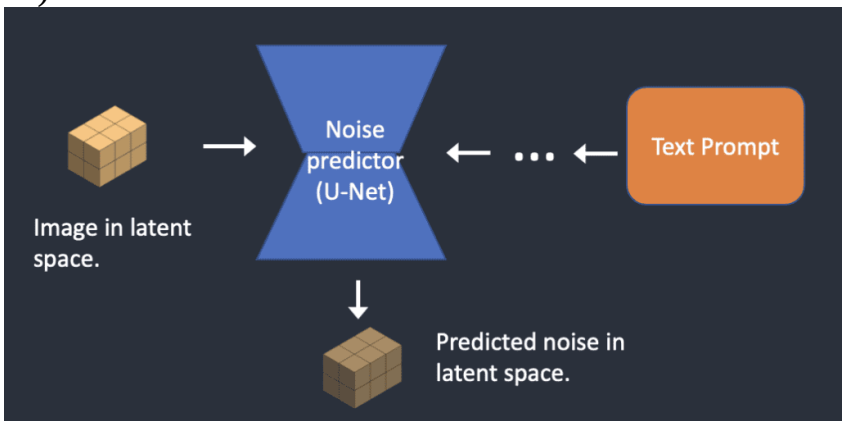
Stable Diffusion крок за кроком

Текст у зображення

У режимі "текст-зображення" ви даєте Stable Diffusion текстовий запит, а вона повертає зображення.

Крок 1. Stable Diffusion генерує випадковий тензор у латентному просторі. Ви керуєте цим тензором, задаючи початкове значення генератора випадкових чисел. Якщо ви встановите певне значення, ви завжди будете отримувати один і той самий випадковий тензор. Це і є ваше зображення в латентному просторі. Але поки що це все шум.

Крок 2. Предиктор шуму U-Net бере на вхід латентне зашумлене зображення і текстову підказку і пророкує шум, також в латентному просторі (тензор 4x64x64).



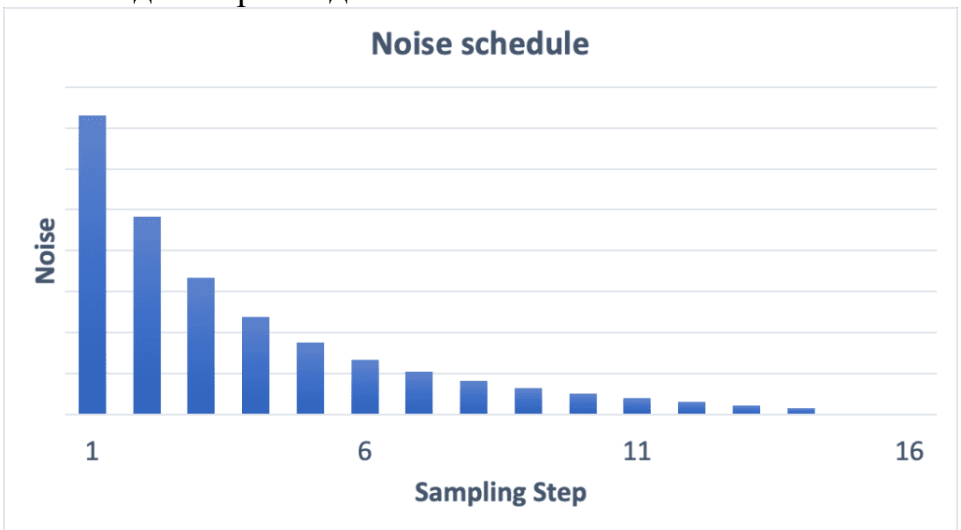
Крок 3. Віднімiть прихований шум від прихованого зображення. Це стане вашим новим латентним зображенням.

Кроки 2 і 3 повторюються для певної кількості кроків дискретизації, наприклад, 20 разів.

Крок 4. Нарешті, декодер VAE перетворює приховане зображення назад у піксельний простір. Це зображення, яке ви отримуєте після запуску Stable Diffusion.

Графік шуму

Зображення змінюється від зашумленого до чистого. Вам здається, що предиктор шуму погано працює на початкових етапах? Насправді, це лише часткова правда. Справжня причина в тому, що ми намагаємося досягти очікуваного шуму на кожному кроці дискретизації. Це називається графіком шуму. Нижче наведено приклад.



Графік шуму - це те, що ми визначаємо. Ми можемо віднімати однакову кількість шуму на кожному кроці. Або ми можемо віднімати більше на початку, як показано вище. На кожному кроці семплер віднімає рівно стільки шуму, щоб досягти очікуваного рівня шуму на наступному кроці. Це те, що ви бачите на покроковому зображенні.

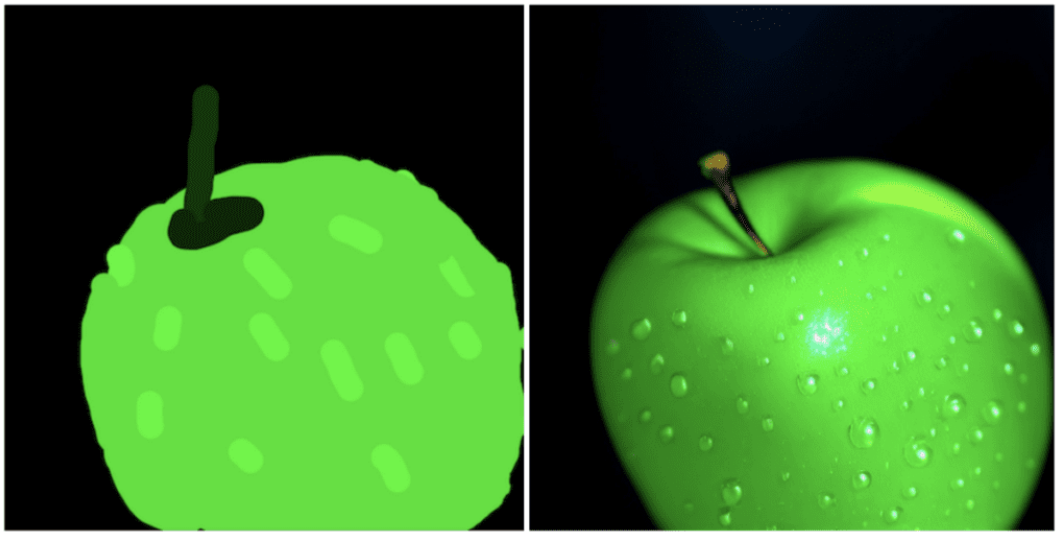
Зображення до зображення

Зображення до зображення перетворює одне зображення в інше за допомогою стабільної дифузії. Вперше це було запропоновано у методі SDEdit. SDEdit можна застосувати до будь-якої моделі дифузії. Таким чином, ми маємо зображення до зображення для Stable Diffusion (модель латентної дифузії).

«*Керований синтез зображень дозволяє звичайним користувачам створювати та редагувати фотореалістичні зображення з мінімальними зусиллями. Основна проблема полягає в тому, щоб збалансувати точність відтворення вхідних даних користувача (наприклад, намальованих від руки кольорових штрихів) і реалістичність синтезованого зображення. Існуючі методи на основі GAN намагаються досягти такого балансу, використовуючи або умовні GAN, або інверсії GAN, що є складним завданням і часто вимагає додаткових навчальних даних або функцій втрат для окремих застосувань.*

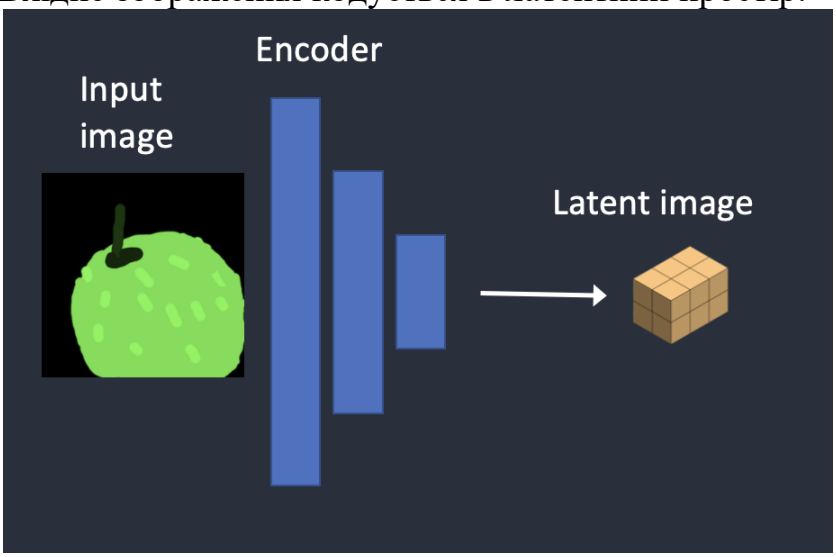
Для вирішення цих проблем було представлено новий метод синтезу та редагування зображень - стохастичне диференціальне редагування (SDEdit), заснований на генеративній моделі дифузії, який синтезує реалістичні зображення шляхом ітеративного згладжування за допомогою стохастичного диференціального рівняння (SDE). Отримавши вхідне зображення з інструкцією користувача будь-якого типу, SDEdit спочатку додає до нього шум, а потім за допомогою SDE знешумлює отримане зображення, щоб підвищити його реалістичність. SDEdit не вимагає спеціального навчання або інверсій і може природним чином досягти балансу між реалістичністю та достовірністю. SDEdit значно перевершує найсучасніші методи на основі GAN - до 98,09% за реалістичністю та 91,72% за загальною оцінкою задоволеності, згідно з дослідженням людського сприйняття, при виконанні різних завдань, включаючи синтез і редагування зображень на основі штрихів, а також компонування зображень.»

Вхідне зображення та текстова підказка подаються як вхідні дані у форматі "зображення до зображення". Наприклад, використовуючи цей аматорський малюнок і підказку "фото ідеального зеленого яблука зі стеблом, краплями води, драматичним освітленням" як вхідні дані, метод "зображення до зображення" може перетворити його на професійний малюнок:

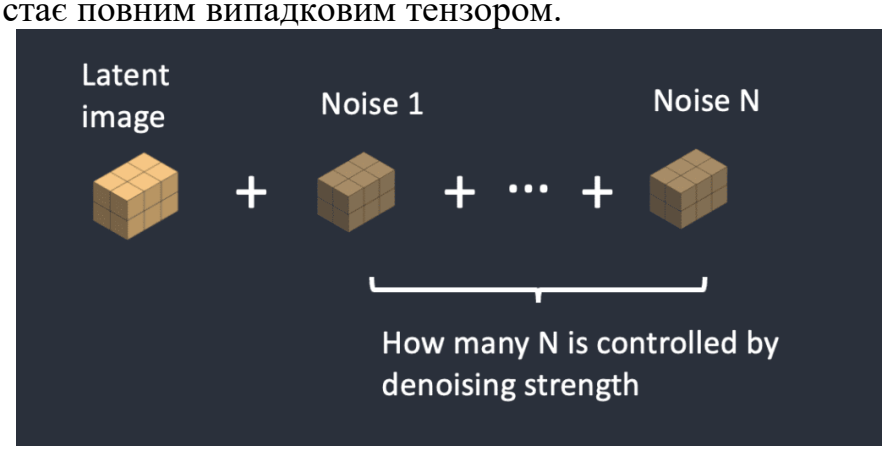


Ось покрокова інструкція.

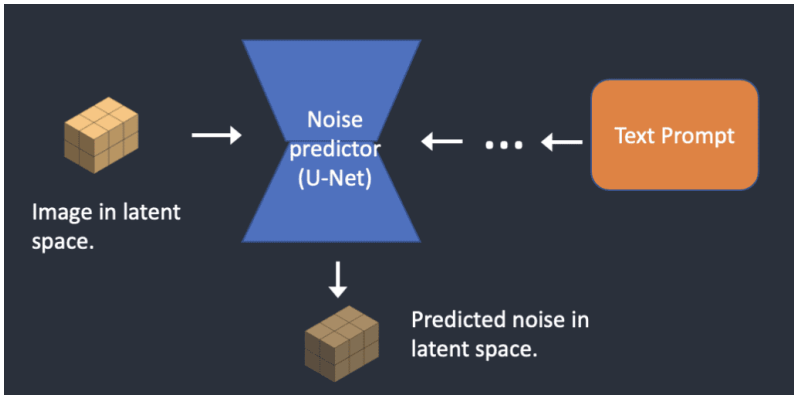
Крок 1. Вхідне зображення кодується в латентний простір.



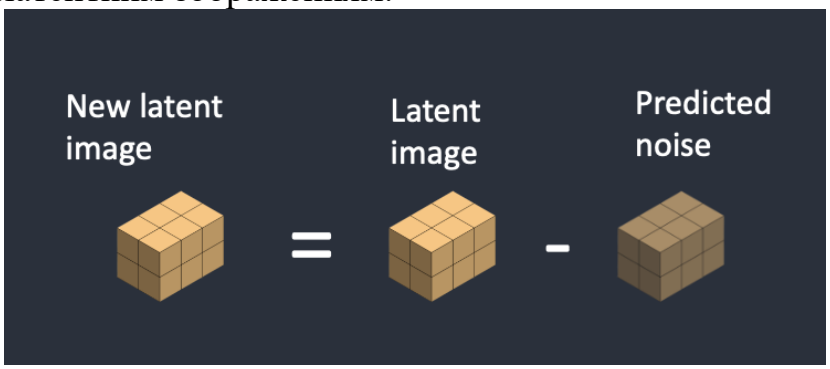
Крок 2. До прихованого зображення додається шум. Сила згладжування визначає, скільки шуму буде додано. Якщо вона дорівнює 0, шум не додається. Якщо 1, то додається максимальна кількість шуму, так що приховане зображення стає повним випадковим тензором.



Крок 3. Предиктор шуму U-Net приймає на вхід латентне зашумлене зображення і текстову підказку і пророкує шум у латентному просторі (тензор 4x64x64).



Крок 4. Віднімiть прихований шум від прихованого зображення. Це стане вашим новим латентним зображенням.



Кроки 3 і 4 повторюються для певної кількості кроків дискретизації, наприклад, 20 разів.

Крок 5. Нарешті, декодер VAE перетворює приховане зображення назад у піксельний простір. Це зображення, яке ви отримаєте після запуску перетворення "зображення в зображення".

Тепер ви знаєте, що таке перетворення зображення в зображення: Все, що воно робить - це встановлює початкове приховане зображення з невеликою кількістю шуму і вхідним зображенням. Встановлення сили знешумлення на 1 еквівалентно перетворенню текст-зображення, оскільки початкове приховане зображення є абсолютно випадковим.

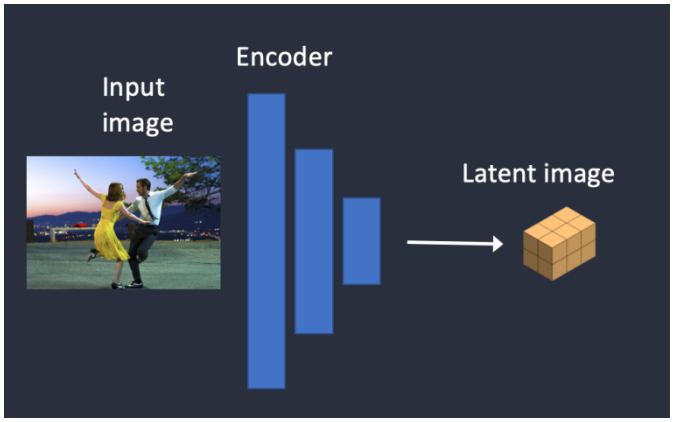
Домальовування

Домальовування - це лише окремий випадок накладання зображення на зображення. До частин зображення, які ви хочете зафарбувати, додається шум. Кількість шуму так само контролюється силою знешумлення.

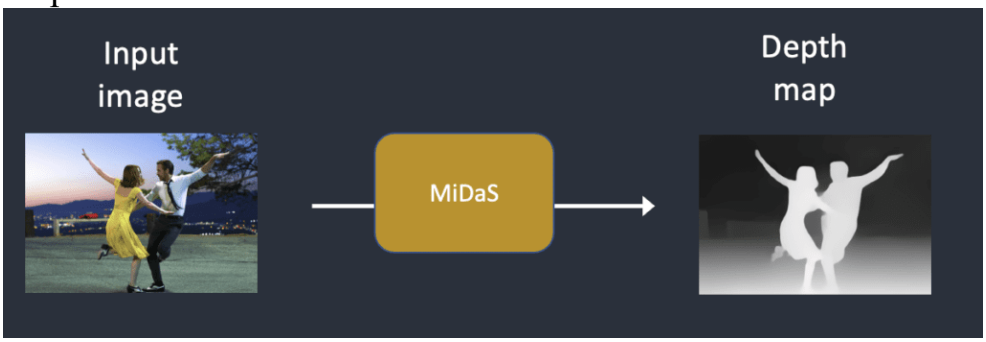
Глибина до зображення

Глибина до зображення - це вдосконалення методу "зображення до зображення"; він генерує нові зображення з додатковим кондиціонуванням за допомогою карти глибини.

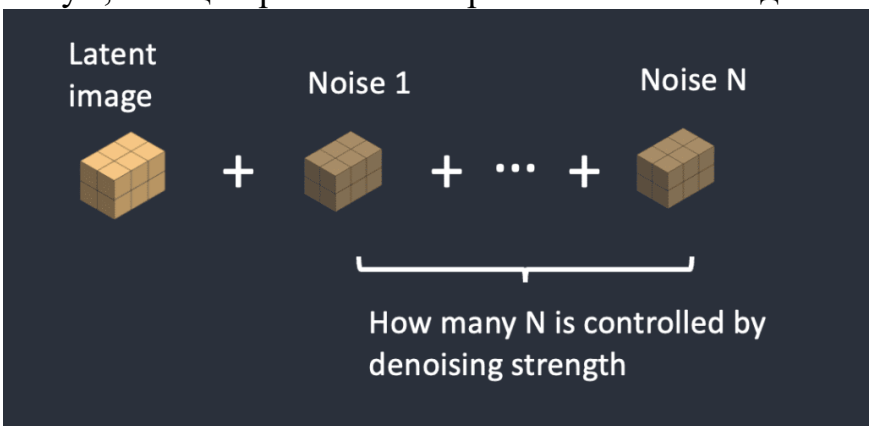
Крок 1. Вхідне зображення кодується в латентний стан



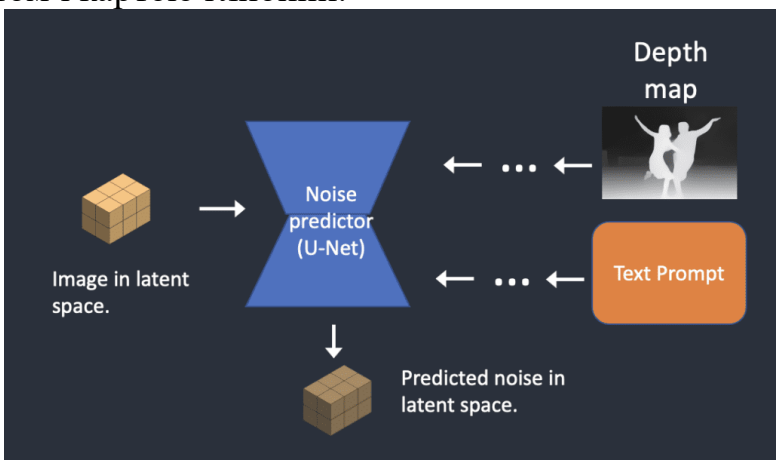
Крок 2. MiDaS (модель глибини ШІ) оцінює карту глибини на основі вхідного зображення.



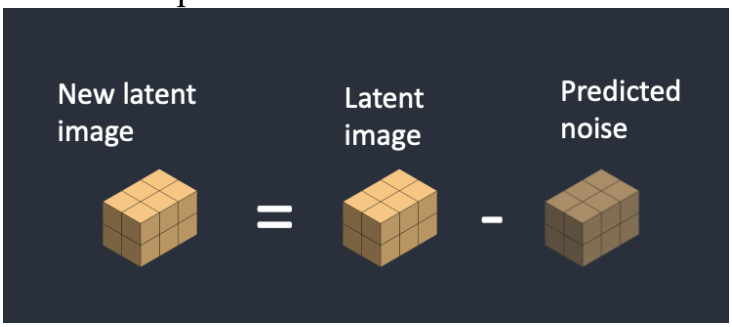
Крок 3. До прихованого зображення додається шум. Сила знешумлення контролює, скільки шуму буде додано. Якщо сила знешумлення дорівнює 0, шум не додається. Якщо сила знешумлення дорівнює 1, додається максимальний шум, так що приховане зображення стає випадковим тензором.



Крок 4. Предиктор шуму оцінює шум латентного простору, обумовлений текстовим запитом і картою глибини.

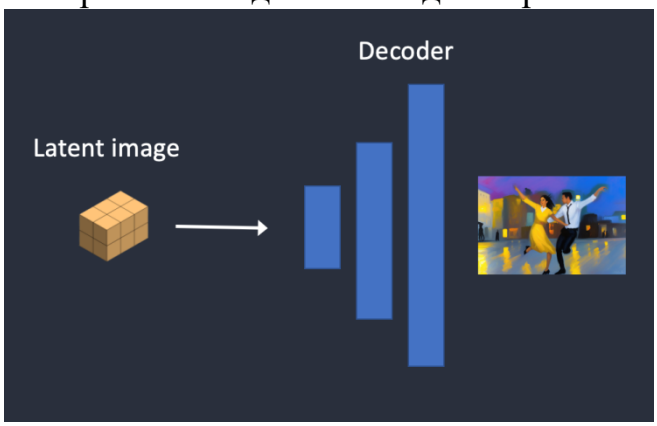


Крок 5. Віднімiть прихований шум від прихованого зображення. Це стане вашим новим латентним зображенням.



Кроки 4 і 5 повторюються для відповідної кількості кроків дискретизації.

Крок 6. Декодер VAE розшифровує приховане зображення. Тепер ви отримуєте остаточне зображення від глибини до зображення.



Що таке значення CFG

Ця тема не буде повною без пояснення безкласифікаторного наведення (CFG) – значення, з яким розробники ШІ працюють щодня. Щоб зрозуміти, що це таке, нам потрібно спочатку торкнутися його попередника – наведення за класифікатором.

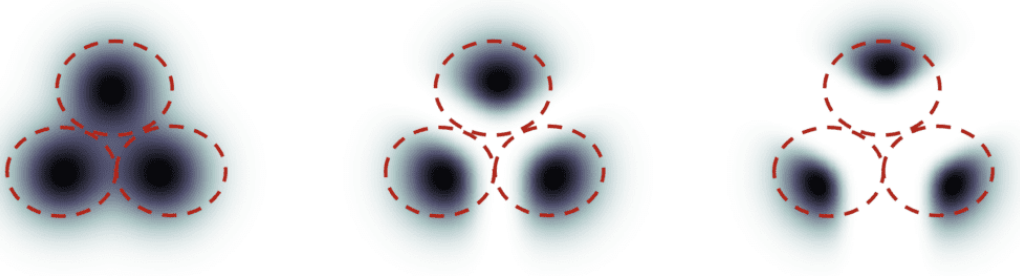
Керування за класифікатором

Наведення за класифікатором - це спосіб включення міток зображень у моделі дифузії. Ви можете використовувати мітку для керування процесом дифузії. Наприклад, мітка "кішка" керує процесом зворотної дифузії для створення фотографій котів.

Шкала наведення класифікатора - це параметр для контролю того, наскільки точно процес дифузії повинен слідувати за міткою.

Припустимо, що є 3 групи зображень з мітками "кішка", "собака" і "людина". Якщо дифузія буде некерованою, модель витягне зразки із загальної популяції кожної групи, але іноді вона може намалювати зображення, які можуть відповідати двом міткам, наприклад, хлопчика, який гладить собаку.

Наведення класифікатора. Зліва: без наведення. Посередині: мала шкала наведення. Праворуч: велика шкала наведення.



З високим рівнем наведення класифікатора зображення, створені дифузійною моделлю, будуть зміщені в бік екстремальних або однозначних прикладів. Якщо ви запитаєте модель про кота, вона поверне зображення, яке однозначно є котом і нічим іншим.

Шкала вказівки класифікатора контролює, наскільки точно дотримуються вказівки. На рисунку вище, вибірка праворуч має вищу шкалу наведення класифікатора, ніж та, що посередині. На практиці значення цієї шкали є просто множником члена дрейфу до даних з такою міткою.

Наведення без класифікатора

Хоча наведення за допомогою класифікатора досягло рекордної продуктивності, воно потребує додаткової моделі для забезпечення цього наведення. Це створює певні труднощі в навчанні.

Безкласифікаторне наведення, за словами його авторів, - це спосіб досягти "класифікаторного наведення без класифікатора". Замість того, щоб використовувати мітки класів і окрему модель для наведення, вони запропонували використовувати підписи до зображень і тренувати модель умовної дифузії, точно таку ж, яку ми обговорювали в процесі перетворення тексту в зображення.

Вони поставили частину класифікатора як умову для предиктора шуму U-Net, досягнувши так званого "безкласифікаторного" (тобто без окремого класифікатора зображень) наведення в процесі генерації зображень.

Текстова підказка забезпечує таке керування в процесі перетворення тексту в зображення.

Шкала наведення без класифікатора

Отже, ми маємо процес дифузії без класифікатора з використанням кондиціонування. Як нам контролювати, наскільки зображення, згенеровані ШІ, повинні слідувати підказкам?

Шкала безкласифікаторного наведення (CFG) - це значення, яке контролює, наскільки текстова підказка керує процесом дифузії. Генерація зображень ШІ є безумовною (тобто підказка ігнорується), коли шкала CFG дорівнює 0. Чим вища шкала CFG, тим більше дифузія спрямовується в бік підказки.

Модель SDXL

Модель SDXL є офіційним оновленням моделей v1 і v2. Модель випускається як програмне забезпечення з відкритим вихідним кодом.

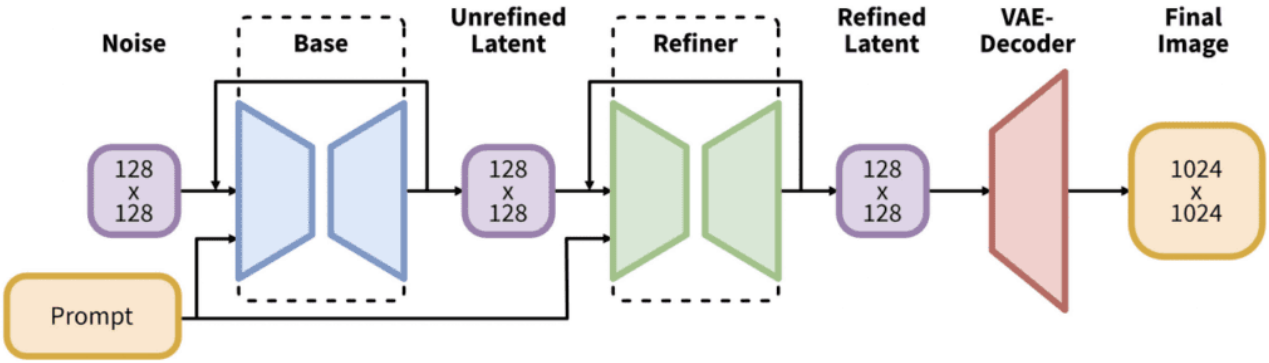
Це набагато більша модель. У світі ШІ ми можемо очікувати, що вона буде кращою. Загальна кількість параметрів моделі SDXL становить 6,6 мільярда, порівняно з 0,98 мільярда для моделі v1.5.

На практиці модель SDXL - це дві моделі. Ви запускаєте базову модель, а потім уточнюючу модель. Базова модель задає глобальний склад. Уточнююча модель додає більш дрібні деталі.

Ви можете запустити базову модель окремо, без доопрацювання.

Зміни в базовій моделі SDXL такі:

- Текстовий кодер поєднує в собі найбільшу модель OpenClip (ViT-G/14) та власну розробку OpenAI - CLIP ViT-L. Це розумний вибір, оскільки він робить SDXL легким для підказок, залишаючись при цьому потужним і здатним до навчання OpenClip.
- Нове налаштування розміру зображення, яке спрямоване на використання навчальних зображень розміром менше 256×256. Це значно збільшує навчальні дані, не відкидаючи 39% зображень.
- U-Net втричі більша за версію v1.5.
- Розмір зображення за замовчуванням - 1024×1024. Це в 4 рази більше, ніж 512×512 у моделі v1.5.



ТЕМА 11. Дані та шкали попередня обробка даних

Дані – у широкому розумінні можуть представляти собою факти, текст, графіки, картинки, звуки, аналогові або цифрові відео-сегменти тощо. Дані можуть бути отримані у результаті виміру, експерименту, арифметичних та логічних операцій.

Дані є необробленим матеріалом і повинні бути представлені у формі, придатній для зберігання, передачі та обробки.

Набір даних та їх атрибутів

При розв'язанні задач Data Mining набір даних зручно представити у вигляді двомірної таблиці (табл. 1).

Таблиця 1

Двомірна таблиця «об'єкт – атрибут»: набір даних

	Атрибути				
Об'єкти	Код клієнта	Вік	Сімейний стан	Дохід	Клас
	1	18	Одинокий	125	1
	2	22	Одружений	100	1
	3	30	Одинокий	70	1
	4	32	Одружений	120	1
	5	24	Вдівець	95	2
	6	25	Одружений	60	1
	7	32	Вдівець	220	1
	8	19	Одинокий	85	2
	9	22	Одружений	75	1
	10	40	Одинокий	90	2

Значення змінної – є проявом ознаки (виміру, характеристики).

Змінні можуть бути **числовими** даними, або **категоріальними** (якісними) даними.

Числові дані можуть бути:

Дискретними – даними, які є значеннями ознаки, загальна кількість яких є зліченою (може бути порахована натуральними числами) (тривалість маршруту трамвая – 10, 15, 30 хвилин).

Неперервними – даними, значення яких знаходяться у якомусь інтервалі (температура, ріст, вага, довжина).

При аналізі даних не завжди є можливість розглянути усю сукупність даних, що аналізуються, їх може бути дуже багато. У такому випадку розглядають тільки частину усієї сукупності даних – вибірку, розмір якої залежить від об'єктів, представлених у генеральній сукупності даних.

Генеральна сукупність – уся сукупність об'єктів, що вивчаються.

Вибірка – частина генеральної сукупності, відібрана певним способом з метою дослідження та отримання висновків про властивості і характеристики генеральної сукупності даних.

Параметри – числові характеристики генеральної сукупності.

Статистики – числові характеристики вибірки.

Шкали

У процесі підготовки даних у певній шкалі здійснюється вимір характеристик об'єктів, що досліджуються.

Вимір – процес привласнення чисел або символьних значень характеристикам об'єктів, які досліджуються відповідно до певного правила.

Шкала – правило, у відповідності з яким об'єктам привласнюються певні значення.

Шкали вимірів категоріальних даних:

1. **Номінальна шкала** – містить тільки категорії, дані у ній не можуть упорядковуватися, з ними не можна виконувати арифметичні та інші операції. Об'єкти класифіковані, мають словесні імена або умовні номери, які нічого не говорять про властивості об'єкта.

Для цієї шкали можна застосовувати тільки дві операції: «дорівнює» та «не дорівнює» (наприклад – стать, національність, клінічний діагноз, автомобільний номер).

2. **Дихотомічна шкала** – містить тільки дві категорії, зазвичай «так-ні» (окремий випадок номінальної шкали).

Для цієї шкали можна застосовувати дві операції: «дорівнює» та «не дорівнює» (наприклад – проживання у певному місті).

3. **Порядкова шкала** – шкала, у якій об'єктам привласнюються числа для позначення відносної позиції об'єктів, а не величини різниці між ними. Шкала дає можливість ранжувати значення змінних, відповідає на питання порядку слідування величин, а не їх числового порівняння. Об'єкти класифіковані, класи мають номера (закодовані).

Для цієї шкали можна застосовувати такі операції: «дорівнює», «не дорівнює», «більше», «менше» (наприклад – оцінка з предмету, твердість мінералів, військові ранги, посади).

Шкали вимірів числових даних:

1. **Інтервальна шкала** – шкала, у якій можуть бути обчислені різниці між значеннями величин, а їх відношення не має змісту. Шкала дозволяє знаходити різницю між двома величинами та має властивості двох попередніх шкал (номінальної та порядкової), а також дозволяє визначити кількісну зміну ознаки. Існує одиниця виміру, нуль не означає відсутність ознаки.

Для інтервальної шкали можна застосовувати такі операції: «дорівнює», «не дорівнює», «більше», «менше», «додавання», «віднімання» (наприклад – календарний час, температура по Цельсію).

2. **Шкала відношень** (пропорційна) – шкала, у якої є певна точка відліку та можливі відношення між значеннями шкали. Шкала має абсолютний нуль, який відповідає відсутності ознаки у об'єкта.

Для цієї шкали можна застосовувати такі операції: «дорівнює», «не дорівнює», «більше», «менше», «додавання», «віднімання», «множення», «ділення» (наприклад – ріст студентів, площа країни, вага, час, ціна, температура по Кельвіну).

Таблиця 2

Приклад використання різних шкал для множини вимірів різних об'єктів

№ об'єкта	Професія (*номінальна шкала*)	Середній бал (*інтервальна шкала*)	Освіта (*порядкова шкала*)
1	кур'єр	22	середня
2	вчений	55	вища
3	учитель	47	вища

Попередня обробка даних

Попередня обробка вхідних даних є виключно важливим етапом аналізу даних. Це мистецтво, яке приходить з досвідом.

Як стверджує один із засновників Data Mining, найбільш важливими та складними етапами Data Mining є: **очищення даних, попередня обробка та вибір змінних**. Ці етапи можуть займати до **80%** відведеного на розробку проекту часу.

Для того щоб максимально використати можливості інструментів Data Mining необхідно вибрати, очистити та перетворити дані, іноді інтегрувати інформацію, добуту із зовнішніх джерел, і встановити спеціальне середовище для роботи алгоритмів Data Mining. Результати інтелектуального аналізу даних в значній мірі залежать від рівня підготовки даних, а не від "чудових можливостей" якогось алгоритму або множини алгоритмів. Близько 75% роботи з технологією Data Mining припадає на збір даних.

При аналізі даних потрібно уміти добувати знання зовні та знаходити у них пробіли.

Перед використанням технології Data Mining необхідно ретельно проаналізувати її проблеми, обмеження та критичні питання, які пов'язані з нею, а також зрозуміти, чого ця технологія не може дати. ІАД не може замінити аналітика. Технологія не може дати відповіді на ті питання, які не були задані. Вона не може замінити аналітика, а всього лише дає йому потужний інструмент для полегшення і поліпшення його роботи.

Очищення даних

Процедура **очищення даних (data cleaning)** полягає у перевірці зібраних даних, виявлені та видаленні помилок з метою покращення якості даних.

Щоб не було за принципом: «сміття на вході, сміття на виході».

Перевірка даних на стадії очищення виконується з використанням комп'ютерної техніки дозволяє виявляти:

— дані, що виходять за межі певного діапазону: не можна використовувати в Data Mining, їх виправляють

— логічно непослідовні відповіді:

- респондент може вказати, що звичайно при міжміських розмовах користується спеціальною карткою, але при цьому відзначити, що жодного разу не дзвонив
- респондент указує, що часто користується якимсь товаром, і в той же час відзначає, що ніколи його не здобував

— екстремальні значення, шуми та викиди: значення, які різко відрізняються від усієї сукупності. Не завжди результат помилок, нерідко вони вказують на те, що існують певні проблеми з якістю зібраних даних. Аналітик оцінює їх вплив на результат – різні алгоритми по різному чутливі до викидів. Деякі інструменти мають вбудовані інструменти очистки від шумів.

о Наприклад, надмірно занижена оцінка певної торговельної марки може бути результатом того, що респондент просто не перебираючи позначив 1 по всіх її характеристиках (по рейтинговій шкалі від 1 до 7)

— Пропущені значення:

о Дані не були зібрані

о Дані не можна застосувати (дохід для дитини)

Способи обробки: 1) пропущені значення можна ігнорувати, 2) розрахувати нові значення, 3) виключити з аналізу, 4) замінити на можливі значення.

— Дублікати даних: записи з однаковими значеннями усіх атрибутів. Способи обробки: 1) видаляється уся група даних як недостовірна 2) заміна групи дублікатів на один унікальний запис.

Процедура очистки передбачає 1) проведення аналізу даних, 2) визначення порядку та правил перетворення даних, 3) підтвердження правильності обраних правил перетворення 4) заміна даних на очищені.

При наявності проблем у даних на етапі попередньої обробки вони повинні бути вирішені.

Розглянемо приклад у вигляді інформації про суб'єкти кредитування. Нехай є декілька об'єктів, про які відомо наступне: стать, вік, місто, дохід, освіта, індикатор повернення кредиту.

Задача кредитного скорінгу

Стать	Вік	Місто	Дохід	Освіта	Повернув?
Ч	25	Київ	15000	A	1
Ж	31	Київ	**9000000**	C	1
Ч	**10**	Київ	8760	C	0
A	28	Миколаїв	7800	B	0
Ч	23	**?**	6550	A	1
Ж	30	Одеса	**16782**	B	1

А, 10 – помилки статі та віку (статі А немає, кредит не могли дати неповнолітньому) відсутня інформація про місто проживання одного об'єкта.

У стовпці дохід два значення викликають питання – дуже велике (або помилка, або викид – реальне значення, дуже відрізняється від інших) та занадто детально вказане (усі інші округлені)

Не зрозуміло значення у стовпці освіта (їх потрібно узнати)

Обробка викидів

Як шукати викиди – як установити, що дохід у 900 000 – викид і його слід відфільтрувати.

Для цього слід установити наступне:

Перша квантиль Q1: таке значення, що рівно 25% об'єктів лежать лівіше від нього

Третя квантиль Q3: таке значення, що рівно 75% об'єктів лежать лівіше від нього

Інтерквауутильний розмах IQR = Q3 - Q1 – це є міра розкиду даних, чимось схожа на дисперсію.

Викиди лежать за межами інтервалу: [Q1 – 1.5*IQR; Q3 + 1.5*IQR]

Такий підхід є евристикою, обгрунтувань якісних немає, але його часто використовують на практиці.

Пропущені значення

Додамо у нашу таблицю декілька пропусків

Пропуск можна замінити на:

1) середній дохід по усій вибірці даних: 10978,4

2) ознаки стать та місто категоріальні, для них використовують інший підхід: місто, яке найбільш часто зустрічається: Київ та Ч.

Нормалізація та стандартизація даних

При вирішення задач аналізу даних доводиться мати справу з перетворенням даних у процесі їх підготовки для подальшого аналізу. У процесі перетворення даних використовують статистичні характеристики вибірки даних.

Деякі статистичні характеристики вибірки даних	
$x_1, x_2, \ldots, x_n$	вибірка n значень змінної x n – обсяг вибірки
$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$	середнє арифметичне
$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$	дисперсія
$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$	середньоквадратичне відхилення
$s = \sqrt{\frac{n}{n-1}\sigma^2}$	стандартне відхилення

Дані для аналізу можуть бути різнотипними, тому виникає необхідність приведення їх до однієї шкали, одного числового діапазону для подальшого використання. Вирішити поставлену задачу дозволяє нормалізація та стандартизація даних.

Нормування даних – це корегування вектора значень (набору вимірів) відповідно до деяких функцій перетворення, з метою зробити їх більш зручними для порівняння та сприйняття.

Наприклад, розділивши значення росту, виміряні у дюймах, на 2.54, ми одержимо виміри в метричній системі.

Нормалізація даних – дозволяє привести всі використовувані числові значення змінних так, щоб вони були у певному числовому діапазоні (наприклад, від 0 до 1), більш зручному для застосування до даних алгоритмів інтелектуального аналізу, а також для погодження діапазонів змін різних ознак.

Нормалізація даних направлена на надання усім змінним однакової ваги при застосуванні алгоритмів Data Mining. Характеристики, числове значення яких знаходиться у більшому числовому діапазоні, будуть мати більшу вагу, що буде впливати на результат аналізу.

Стандартизація даних передбачає таке перетворення даних, після якого кожна ознака має середнє 0 та дисперсію – 1.

Існує багато підходів до нормалізації даних. Різні види стандартизації та нормалізації передбачають перетворення даних по різному: із значень змінних вираховується їх середнє і ці значення діляться на стандартне відхилення (zстандартизація), лінійним перетворенням добиваються розкиду змінних в інтервалі [-1;1], [0;1], значення змінних ділять на середнє, максимум або стандартне відхилення.

Основні формули нормалізації та стандартизації

№	Формула	Характеристика формули
1	***min-max нормалізація*** $x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$	Область значень - [0,1]. Рекомендується икористовувати, якщо значення початкових даних рівномірно заповнюють область дослідження. Для деяких методів прогнозування формула неефективна у випадку рівності значень нулю або їх зосередження біля кінців відрізка [0,1]
	$x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)} \cdot (B - A) + A$	Різновидом ***min-max нормалізації*** є формула перетворення до діапазону [A,B]
2	***max- min нормалізація*** $x_i' = \frac{\max(x_i) - x_i}{\max(x_i) - \min(x_i)}$	Аналогічно першій, але дозволяє зворотно пропорційно розвернути шкалу, що зручно у випадках, коли більшість характеристик мають максимізуватися, а дана характеристика - мінімізуватися. Недоліки ті ж самі

3	***z- нормалізація*** (стандартизація) $x_i' = \frac{x_i - \overline{x_i}}{\sigma_i}$ $x_i' = \tau(x_i) = \frac{x_i - \overline{x}}{s_x}$	Відрізняється тим, що отримані в результаті застосування значення є безрозмірними, знаходяться з дисперсією та СКВ, рівними 1, на відрізку $\left[\frac{x_{min} - \overline{x}}{\sigma_x}; \frac{x_{max} - \overline{x}}{\sigma_x}\right]$ переважно в околі нуля. У перетворенні використовується $\overline{x}_i$ - вибіркове середнє значення i - тої змінної, σ_{x_i} - її ж вибіркове стандартне відхилення або середньоквадратичне відхилення *Корисно*: коли невідомі максимальні та мінімальні значення або є домінуючі аномалії (значення у певних проміжках)
4	$x_i' = \frac{2 \cdot (x_i - \min(x_i))}{\max(x_i) - \min(x_i)} - 1$	Область значень $[-1;1]$. Формула зручна для використання при прогнозуванні із застосуванням нейронних мереж, в яких активаційною функцією є гіперболічний тангенс. Має всі ті ж переваги й недоліки, що і перетворення 1 та 2
5	$x_i = \frac{1}{1 + e^{-x_i}}$	Область значень $(0;1)$. Використовується рідко, здебільшого для значного підсилення реакції на зміни значень в околі нуля. Функція є допоміжною, вона не позбавляє розмірності значення факторів
6	***Масштабування*** $x_i' = \tau(x_i) = \lambda \cdot x_i \quad \lambda \neq 0, \lambda = const$ Популярні константи: $\lambda = \frac{1}{\|x\|} \Rightarrow x' = \tau(x) = \frac{x}{\|x\|} \Rightarrow \|x'\| = 1$ $\lambda = 10^{-p}, p = \min_k \left\{ \frac{x_i}{10^k} : \max_{i=1..N} \left(\left\| \frac{x_i}{10^k} \right\| \right) \leq 1 \right\}$ $x' = \tau(x) = \frac{x}{10^p} \Rightarrow x' \in [-1,1]$	Зміна довжини вектора значень характеристики шляхом множення на константу. Масштабування важливе для метричних, лінійних, нейромережевих моделей. Не має значення для логічних методів (дерев рішень)

Дискретизація

Дискретизація числового атрибуту - заміна початкових значень на **інтервальні** або **концептуальні мітки**. Дозволяє неперервні дані перетворити у дискретні.

Приклад. Значення атрибуту «вік» можна замінити на:

- інтервальні мітки: 0-10, 11-20, …
- концептуальнімітки: молодий, дорослий, старий

Мітки можуть бути об'єднані в поняття більш високого рівня, визначаючи ієрархію понять числового атрибуту.

ТЕМА 12. Випадкові велечини

Перетворення категоріальних даних

Один з популярних підходів при роботі з цими ознаками – **Dummy – кодування**:

- Ознака xj категоріальна й приймає значення з множими значень $\boldsymbol{U} = \{\boldsymbol{u1}, \boldsymbol{u1}, \dots, \boldsymbol{um}\}$
- Створимо m нових бінарних ознак-індикаторів xj1 , …, xjm: xjk = [xj = uk]

При цьому к-й індикатор показує, чи дорівнює дана ознака на даному об’єкті значенню uk.

Приклад: нехай ознака кодує міста, при цьому можливо всього три міста:

U = {Київ, Харків, Одеса}

Кодуємо трьома бінарними ознаками:

Київ → (1, 0, 0) (буде закодовано вектором 1 0 0)

Харків → (0, 1, 0)

Одеса → (0, 0, 1)

Проблеми dummy – кодування:

1. Приклад 1. У виборці місто зустрічається тільки один раз, значить одна з кодуючи ознак прийме значення 1 тільки на одному об’єкті – ця ознака не має змісту. Для вирішення цієї проблеми значення ознак, що рідко зустрічаються, можна об’єднувати у одну категорію: тобто усі міста, які зустрічаються рідко, об’являють одним містом.

2. Приклад 2. Потрібно передбачити, чи натисне користувач по рекламному банеру. Ця задача має 4 ознаки:

- Ідентифікатор користувача
- Ідентифікатор банера
- Ідентифікатор сайту, на якому показано банер
- Ідентифікатор категорії банера

Це категоріальні ознаки, які можуть приймати багато різних значень. При використанні dummy – кодування буде отримано мільйон ознак. Це буде дуже велика вибірка, на якій буде дуже важно навчатися.

Для вирішення проблеми можна скористатися лічильниками. Ідея:

- Нехай по банеру u1 клікають частіше, ніж по банеру u2
- Це важлива ознака
- Замінимо категорії на ймовірності кліків

Приведемо приклад. Нехай є 7 обєктів, кожен з яких відноситься до класу 0 або класу 1. Та одна категоріальна ознака – місто.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
Місто	Київ	Одеса	Київ	Харків	Харків	Київ	Одеса
y	1	1	0	0	0	1	1

Оцінимо ймовірності значень категоріальної ознаки:

p(y = 1|Київ) = 2/3 = 0,67

p(y = 1|Харків) = 0/2 = 0
p(y = 1|Одеса) = 2/2 = 1
Далі робимо заміну кожного міста на зроблені оцінки:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
Місто	0,67	1	0,67	0	0	0,67	1
y	1	1	0	0	0	1	1

Перетворення даних номінальних шкал виміру

- Побудова ієрархій понять – полягає у узагальненні атрибутів більш загальними поняттями.

Наприклад: місто – країна

- Номінальні дані можуть приймати скінчену кількість значень без відношень порядку

Наприклад: вулиця.

- Побудова ієрархій може бути складною задачею та вимагати експертних знань з предметної області.
- Якщо дані зберігаються у базі даних, тоді багато ієрархій можна побудувати на основі схеми БД.

Статистичний аналіз

Статистика – наука збору, представлення, аналізу і розумної інтерпретації даних.

Метою аналізу даних є **знання** про об’єкт дослідження – виявлення корисної інформації, знайдення висновків, зважене прийняття рішень. Аналіз даних може мати багато аспектів та підходів, реалізовуватися за допомогою різних інструментів — в тому числі математичних, статистичних.

Статистика являє собою сукупність наукових методів, що дозволяють зрозуміти дані, дійти до їх суті.

Наприклад, у дослідженні вимірюється вага 100 пацієнтів. Це уже достатньо велика кількість спостережень, і просто глянувши на дані неможливо отримати швидке інформативне уявлення. Однак статистика може дати миттєву загальну картину даних – на основі доступної для сприйняття візуалізації або числового узагальнення — незалежно від кількості спостережень чи одиниць даних.

Статистичні методи – методи аналізу статистичних даних.

Статистичні методи дозволяють здійснити:

1) узагальнення даних
2) формулювання логічних висновків
3) передбачення залежностей між змінними
4) прогнозування.

Методи статистичного аналізу діляться на одномірні і багатомірні.

Одномірні методи передбачають аналіз кожної змінної окремо від інших.

Багатомірні методи містять взаємопов’язаний аналіз більш ніж однієї змінної.

За природою статистичні дані поділяють на:

— кількісні дані (метричні) є безперервними за своєю структурою, можуть бути виміряні за допомогою інтервальної шкали або шкали відношень.

— категоріальні дані (неметричні) – якісні дані з обмеженим числом унікальних значень і категорій; існує два види категоріальних даних: номінальні – використовуються для нумерації об'єктів і порядкові – дані, для яких існує природний порядок категорій.

Аналізуючи статистичні дані, ми зазвичай маємо справу з вибіркою даних, яка є частиною генеральної сукупності (сукупності усіх даних).

Статистики – числові характеристики вибірки.

Статистика є випадковою величиною, оскільки в її основі лежать вибіркові дані.

Випадкова величина – змінна величина, яка у результаті досліду (вимірювання) може приймати те чи інше числове значення.

Випадкова величина може бути неперервною та дискретною.

Основні **етапи статистичного аналізу**:

1) планування досліджень, результати яких можуть бути подані у вигляді випадкової вибірки;

одані у вигляді випадкової вибірки;

2) попереднє дослідження даних, що дозволяє в подальшому аналізі адекватно оцінити статистичні характеристики;

3) оцінка невідомих величин та функцій, яка базується на вихідних даних;

4) перевірка статистичних гіпотез, яка дозволяє на основі вибіркових даних оцінити невизначеність у виборі характеристик.

Попереднє дослідження даних включає: формування варіаційних рядів та гістограм, редагування даних (вилучення аномальних значень), ідентифікацію типів розподілів тощо.

Первинний статистичний аналіз: побудова варіаційного ряду, гістограми, полігону

При здійсненні аналізу невеликої вибірки даних, можна одночасно бачити її числові характеристики: мінімальне,максимальне значення тощо. А коли вибірка є надто великою, необхідно застосовувати статистичні методи для отримання розуміння про характеристики вибірки даних.

Статистичний аналіз одновимірних даних вимагає проведення первинного статистичного аналізу, що є необхідною складовою етапу попереднього дослідження даних, та розв'язання статистичної задачі відтворення функції розподілу.

Розглянемо обчислювальні процедури первинного статистичного аналізу.

Формування варіаційного ряду

Нехай задана вибірка n значень змінної x: {xi, i = 1, …n } де n – об'єм вибірки даних, xi – результати спостережень (вимірювань) реалізації випадкової величини .

Вибірка представлена значеннями змінної x у порядку їх отримання. Якщо значення вибірки розмістити у порядку їх зростання, одержимо впорядковану вибірку: x1, x2, …, xr – варіаційний ряд.

Розмах варіаційного ряду – дорівнює різниці Xmax-Xmin.

З варіаційним рядом легше працювати у силу його впорядкованості.

Варіанта варіаційного ряду – результат з вибірки даних, що не повторюється.

Частота варіанти (вага) – кількість елементів вибірки даних зі значенням, рівним цій частоті.

Відносна частота варіанти – відношення частоти варіанти до об'єму вибірки даних.

Для формування варіаційного ряду вибірку даних необхідно ранжувати – розмістити у порядку зростання та обчислити частоти і відносні частоти варіант:

$$x_1, \quad x_2, \quad \dots \quad x_r$$
$$n_1, \quad n_2, \quad \dots \quad n_r$$
$$p_1, \quad p_2, \quad \dots \quad p_r,$$

де x_i – варіанта $x_i < x_j$, якщо $i < j$

r – кількість варіант: $\sum_{l=1}^{r} n_l = N$

n_i – частота варіанти x_i

$p_i = \frac{n_i}{N}$ – відносна частота варіанти x_i : $\sum_{i=1}^{r} p_i = 1$

Приклад 1. Для вибірки {5, 2, 1, 3, 2, 8, 4, 5, 3, 2} варіаційний ряд матиме вигляд (об'єм вибірки n =10):

x_i	1	2	3	4	5	8
n_i	1	3	2	1	2	1
p_i	0,1	0,3	0,2	0,1	0,2	0,1

Відносні частоти є **емпіричними ймовірностями** відповідних значень змінної x вибіркових даних, а побудована з них таблиця – статистичним розподілом вибірки:

x_i	1	2	3	4	5	8
p_i	0,1	0,3	0,2	0,1	0,2	0,1

У випадку коли варіаційний ряд містить багато значень, застосовують групування даних: область вибіркових значень змінної x розбивають на інтервали та розраховують частоти попадання значень у кожен інтервал і відповідні відносні частоти.

Рекомендації стосовно кроку розбиття:

$$h = \frac{x_{\max} - x_{\min}}{M}$$

де M – кількість інтервалів розбиття. M повинно бути оптимальним (для 100 об'єктів – 8, для 50 – 5-6).

При $N < 100$ достатньо обмежитися застосуванням формули

$$M = \begin{cases} \left[\sqrt{N}\right], & \text{якщо } \left[\sqrt{N}\right] \text{ непарне,} \\ \left[\sqrt{N}\right] - 1, & \text{якщо } \left[\sqrt{N}\right] \text{ парне,} \end{cases}$$

де $[\cdot]$ – ціла частина.

Є формула Серджеса: M = 1 + 3,322*lgn.

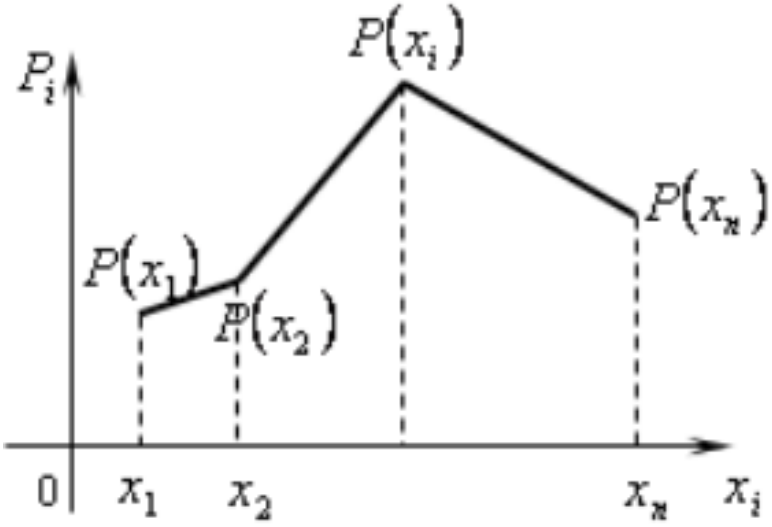
Крім того бажано слідкувати, щоб не було інтервалів, у які попало менше 5 значень.

Варіаційний ряд, побудований з допомогою групування, називають інтервальним.

Більшу інформативність несе зображення варіаційного ряду у вигляді гістограми відносних частот, яка дозволяє візуально швидко оцінити емпіричні ймовірності вибіркових даних. З цією метою здійснюється гістограмна оцінка.

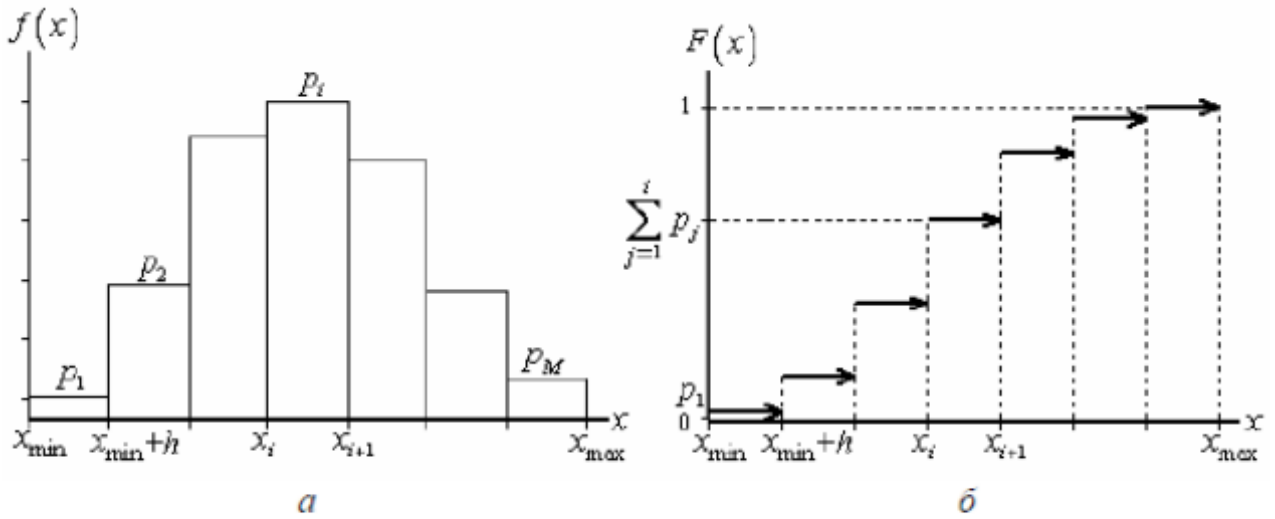
Графічне представлення вибірки

У випадку дискретної випадкової величини вибірку представляють у вигляді **полігону** – по осі абсцис відкладають можливі значення змінної x, а по осі ординат – їх відносні частоти, одержані точки з'єднують відрізками прямих.



Для інтервального варіаційного ряду (групованої вибірки) будується **гістограма** – графічне представлення даних у вигляді прямокутників, основами яких є інтервал групування, а висоти дорівнюють відносним частотам (частотам).

Побудувавши **кумуляту** – графік накопичених частот, отримуємо графічне зображення емпіричної функції розподілу – функції F(x), значення якої дорівнює ймовірності того, що випадкова величина приймає значення, менші за x.



а *б*

Рис. 1.1. Графічне подання результатів гістограмної оцінки:
а – гістограма відносних частот; *б* – графік емпіричної функції розподілу

Для неперервної величини задається **функція щільності ймовірності**, яка дозволяє визначити ймовірність випадкової величини потрапити у інтервал (a, b):

$$P\{a < \xi < b\} = \int_a^b f(x)dx,$$

де f(x) – щільність розподілу ймовірностей випадкової величини .

Побудована гістограма зображає **емпіричну щільність**. При n→ нескінченності, вона прямує до теоретичної щільності.

Числові характеристики випадкової величини

Як характеристики випадкової величини можна використовувати:

— **точкові оцінки** – такі оцінки параметрів вибірки, що визначаються одним числом (середні арифметичні, медіани);

— **інтервальні оцінки** – оцінки, які визначаються двома числами – межами інтервалу, до якого із заданою ймовірністю потрапляє оцінюваний параметр.

При малих обсягах вибірок та при їх значному відхиленні від нормального закону розподілу точкові оцінки можуть істотно відхилятися від істинних значень оцінюваних параметрів. Тому поряд з ними використовують інтервальні оцінки параметрів.

Числові характеристики положення: мода, медіана, математичне сподівання (середнє).

1. **Мода Мо** – значення випадкової величини, ймовірність якого максимальна.

Для дискретного варіаційного ряду – варіанта з максимальною частотою.

Для інтервального рядку – мода знаходиться всередині інтервалу з максимальною частотою. Для її обчислення користуються формулою лінійної інтерполяції.

Для неперервної випадкової величини – мода є значенням випадкової величини з найбільшою щільністю ймовірності.

2. **Медіана** – середина розподілу, така точка, що половина значень лежить ліворуч від неї, а половина – праворуч.

Медіана є найбільш загальною й фундаментальною характеристикою центра розподілу, оскільки вона базується на принципі симетрії.

Для дискретного варіаційного ряду – медіану Ме обчислюють за формулою:

$$M_e = \begin{cases} \dfrac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} & \text{якщо n – парне} \\ x_{\frac{n+1}{2}} & \text{якщо n – непарне} \end{cases}$$

Для інтервального ряду медіана – точка, у якій площа гістограми ділиться навпіл.

Для неперервної випадкової величини – медіана Ме визначається співвідношенням:

$$\int_{-\infty}^{M_e} f(x)dx = \int_{M_e}^{+\infty} f(x)dx$$

Для обчислення інших числових характеристик вибірки інтервальну таблицю вибірки замінюють на дискретну. У якості значень вказують середини інтервалів групування. Тоді і для дискретної і для інтервальної вибірки може бути задана таблиця частот або таблиця відносних частот – таблиця емпіричного розподілу вибірки.

3. **Математичне сподівання** (середнє значення) дискретної випадкової величини – сума добутків можливих значень на відповідні ймовірності:

$$M(x) = \sum_{i=1}^{n} x_i p(x_i)$$

Математичне сподівання неперервної випадкової величини – визначений інтеграл добутків можливих значень на відповідні ймовірності:

$$M(x) = \int_{-\infty}^{+\infty} xf(x)dx$$

Математичне сподівання наближено рівне середньому арифметичному вибірки і тим точніше, чим більше число спостережень.

$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ - як **середнє арифметичне**, якщо варіаційний ряд задано послідовністю

$\bar{x} = \frac{1}{n}\sum_{j=1}^{k} x_j m_j$ - якщо задана таблиця частот варіаційного ряду;

$\bar{x} = \sum_{j=1}^{k} x_j \frac{m_j}{n}$ - якщо задана таблиця відносних частот варіаційного ряду

Числові характеристики розсіювання, міри розкиду: дисперсія, стандартне (середньоквадратичне) відхилення.

Міри розкиду дають нам розуміння, наскільки добре, наприклад, середнє представляє весь набір даних. Якщо розкид значень у розподілі великий, то середнє не є таким репрезентативним, ніж якщо розкид даних малий.

1. **Дисперсія** – міра відхилень значень випадкової величини від центру розподілу (середнього), є математичне сподівання квадрата відхилення випадкової величини від свого математичного сподівання.

Дисперсія характеризує ступінь відхилення елементів сукупності від середнього в одиницях вимірювання відповідної ознаки.

Дисперсія дискретної випадкової величини є сума квадратів відхилень випадкової величини від її математичного сподівання, помножена на відповідні ймовірності:

$$\sigma^2 = D(x) = \sum_{i=1}^{n} \big(x_i - M(x)\big)^2 p(x_i)$$

Для неперервної випадкової величини дисперсія визначається за формулою

$$\sigma^2 = D(x) = \int_{-\infty}^{+\infty} \big(x - M(x)\big)^2 f(x)\, dx$$

Основною перевагою дисперсії як характеристики вибірки є те, що дисперсія суми статистично незалежних вибірок є сумою їх дисперсій:

$$\sigma_\Sigma^2 = \sum_{i=1}^{n} \sigma_i^2$$

незалежно від законів розподілу складових вибірок.

Дисперсію можна розрахувати за спрощеною формулою:

$$\sigma^2 = D(x) = M(x^2) - M^2(x)$$

для дискретної випадкової величини:

$$\sigma^2 = D(x) = \sum_{i=1}^{n} x_i^2 p(x_i) - M^2(x)$$

для неперервної випадкової величини:

$$\sigma^2 = D(x) = \int_{-\infty}^{+\infty} x^2 f(x)\, dx - M^2(x)$$

Якщо середнє значення оцінюють за самою вибіркою, то для розрахунку дисперсії використовують формулу:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Для обчислення скоригованої вибіркової дисперсії користуються формулою:

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

2. Середньоквадратичне (стандартне) відхилення

Середньоквадратичне (стандартне) відхилення дорівнює кореню квадратному з дисперсії випадкової величини:

$$s = \sigma = \sqrt{D(x)}$$

Корінь квадратний з дисперсії дозволяє отримати розмірність таку ж, як і у змінної.

Якщо середньоквадратичне відхилення оцінюють за самою вибіркою, то для розрахунку середньоквадратичного (стандартного) відхилення використовують скореговану формулу (при n<=50):

$$s = \sqrt{\frac{n}{n-1}\sigma^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

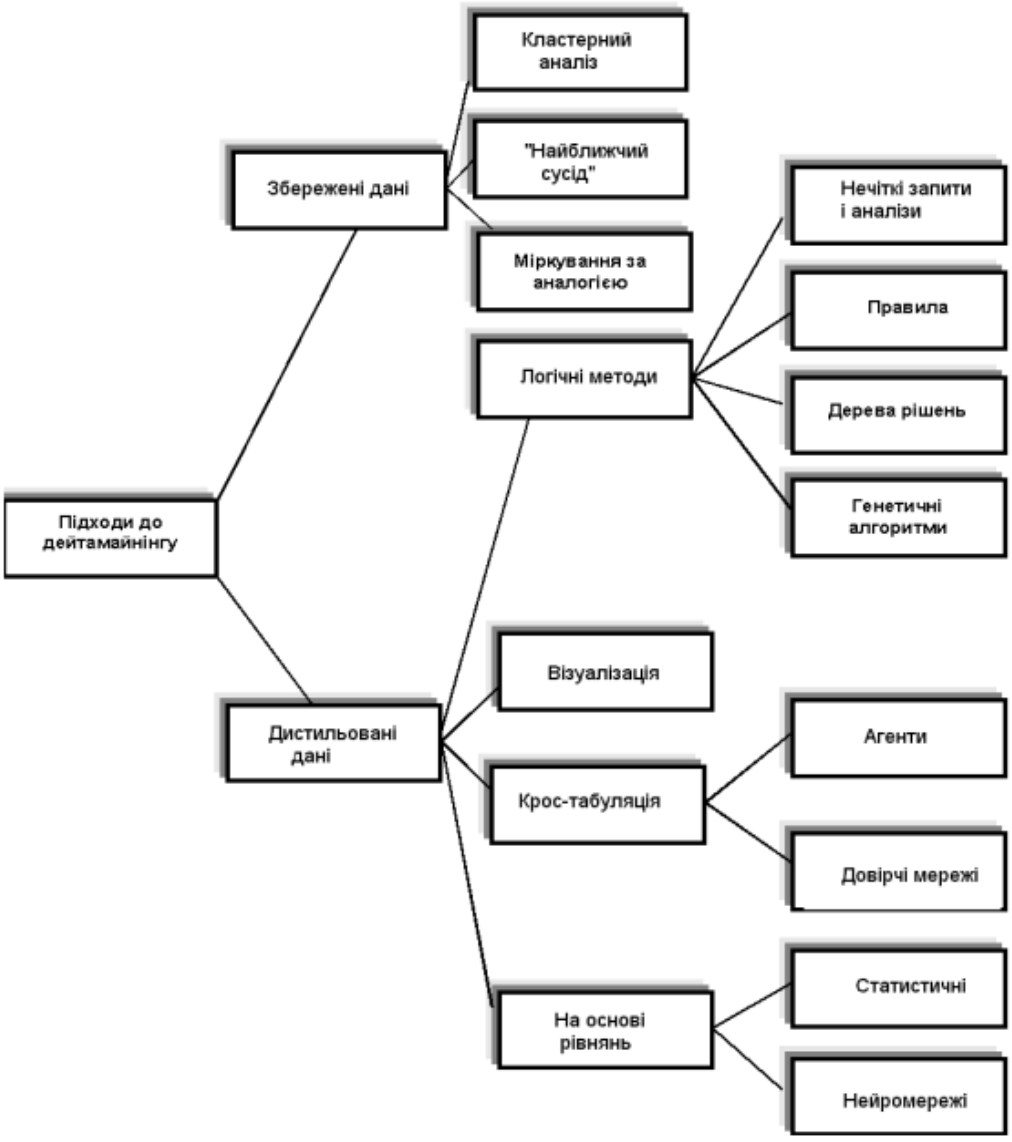
ТЕМА 13. Методи та стадії Data Mining. Закони розподілу

Класифікація методів Data Mining

Головна особливість Data Mining – поєднання широкого математичного інструментарію (від класичного статистичного аналізу до нових кібернетичних методів) і останніх досягнень у сфері інформаційних технологій.

В технології Data Mining гармонічно поєдналися строго формалізовані методи неформального аналізу, тобто кількісний та якісний аналіз.

До методів та алгоритмів Data Mining відносять наступні (рис. 1.): штучні нейронні мережі, дерева рішень, символьні правила, методи ближнього та k-ближнього сусіда, метод опорних векторів, байєсовські мережі, лінійна регресія, кореляційно-регресійний аналіз, ієрархічні та неієрархічні методи кластерного аналізу, методи пошуку асоціативних правил, метод обмеженого перебору, еволюційне програмування та генетичні алгоритми, методи візуалізації даних та інші.



Необхідно відмітити, що більшість методів були розроблені у рамках теорії штучного інтелекту.

Більшість аналітичних методів, які використовують в технології Data Mining – відомі математичні методи та алгоритми. Новим у їх застосуванні є можливість їх використання при розв'язанні конкретних проблем, обумовлених появою можливостей нових технічних та програмних засобів. Тому ці алгоритми ми будемо вивчати на лабораторних роботах з використанням таких програм, як SPSS, MathLab, MathCad, Excel, Statistica.

Метод – норма або правило, певний шлях, спосіб, прийом розв'язання задачі теоретичного, практичного, пізнавального, управлінського характеру.

Алгоритм – точний опис послідовності дій (кроків), які перетворюють вхідні дані у шуканий результат.

Методи Data Mining можна розділити на статистичні та кібернетичні.

До **статистичних** методів відносять:

1. Дескриптивний аналіз та опис вхідних даних;
2. Аналіз зв'язків і закономірностей: кореляційний, регресійний, факторний, дисперсійний аналіз;
3. Багатомірний статистичний аналіз: компонентний аналіз, дискримінантний аналіз, багатомірний регресійний аналіз, кластерний аналіз;
4. Аналіз часових рядів: динамічні моделі і прогнозування.

До **кібернетичних** методів відносять:

1. Штучні нейронні мережі (розпізнавання, кластеризація, прогноз);
2. Еволюційне програмування (в т.ч. алгоритми методу групового обліку аргументів);
3. Генетичні алгоритми (оптимізація);
4. Нечітка логіка;
5. Дерева рішень;
6. Системи обробки експертних знань.

Стадії технології Data Mining

Повний цикл технології Data Mining містить наступні етапи:

До основних стадій Data Mining відносять:

1. **Вільний пошук**. На цій стадії здійснюється дослідження набору даних з метою пошуку прихованих закономірностей – суттєвих, постійно повторюваних взаємозв'язків, які визначають етапи і форми процесу становлення та розвитку різних явищ та процесів. На цій стадії може здійснюватися валідація закономірностей – перевірка їх достовірності на частині даних, які не приймали участь у формування закономірностей. Такий прийом розділення даних на навчаючу та перевірочну множини часто використовують в методах дерев рішень та нейронних мереж.

У результаті вільного пошуку закономірностей, представленого діями умовної та асоціативної логіки, виявленням трендів та коливань, система сформує набір логічних правил «якщо, …, то …». При цьому потрібно задати цільову змінну.

2. **Прогностичне моделювання**. На цій стадії з використання результату попередньої стадії знайдені закономірності застосовують безпосередньо для прогнозування, передбачуючи невідомі значення та прогнозуючи розвиток процесів. У процесі прогностичного моделювання вирішуються задачі класифікації та прогнозування.

3. **Аналіз виключень**. На третій стадії здійснюється аналіз виключень або аномалій, виявлений у знайдених закономірностях. Для виявлення відхилень необхідно визначити норму, яка розраховується на стадії вільного пошуку.

Таблиця прикладів використання інтелектуального аналізу даних

	Інформаційні технології	Торгівля	Фінансова сфера
Класифікація			Оцінка кредитоспроможності
Регресія			Оцінка допустимого кредитного ліміту
Прогнозування		Прогнозування продаж	Прогнозування вартості акцій
Кластеризація		Сегментація клієнтів	Сегментація клієнтів
Визначення взаємозв'язків		Аналіз кошику покупця	
Аналіз послідовностей	Аналіз переходів по сторінкам веб-сайту		
Аналіз відхилень	Виявлення вторгнень в інформаційні системи		

Дейтамайнінг як процес виявлення в загальних масивах даних раніше невідомих, нетривіальних, практично корисних і доступних для інтерпретації знань, необхідних для прийняття рішень у різних галузях людської діяльності, практично має нічим не обмежені сфери застосування. Але, насамперед, методи ІАД нині більше всього цікавлять комерційні підприємства, оскільки рівень рентабельності від застосування дейтамайнінгу може досягати 1000 %.

Наприклад:

1) відомі повідомлення про економічний ефект, за якого прибутки у 10—70 раз перевищували первинні витрати, що становили від 350 до 750 тис. дол.;

2) є відомості про проект у 20 млн дол., який окупився всього за 4 місяці;

3) річна економія 700 тис. дол. за рахунок упровадження дейтамайнінгу в мережі універсамів у Великобританії.

Дейтамайнінг являє собою велику цінність для керівників і аналітиків у їх повсякденній діяльності. Ділові люди усвідомили, що за допомогою методів ДМ вони можуть отримати відчутні переваги в конкурентній боротьбі.

Найбільш яскравим прикладом успіху індустрії Data Mining є Google. Обидва його співзасновника займалися дослідженнями в цій області, і рання історія самого Google пов'язана з ІАД. Рекомендації на сайті Amazon.com ("покупці, що купили/ щошукали/ щоподивилися X, купили також Z") привели до величезного росту продаж.

Області застосування Data Mining, які виділяє один з його засновників та ідеологів:

– *реклама* – *біоінформатика* – *зв'язок з клієнтами* – *маркетинг* – *керування виробництвом* – *розваги й спорт*	– *виявлення шахрайства* – *е-коммерція* – *охорона здоров'я* – *інвестиції/цінні папери* – *телекомунікації* *вивчення веба*

Таким чином, області застосування ІАД (Data Mining):

1) **маркетинг** (Database marketers): ринкова сегментація, ідентифікація цільових груп, побудова профіля клієнта;

2) **банківська справа**: аналіз кредитних ринків, залучення та утримання клієнтів, управління ресурсами;

3) **кредитні компанії**: детекція підробок, формування «типової поведінки» власника кредитної карти, аналіз достовірності клієнтських рахунків, cross-selling програми;

4) **страхові компанії**: залучення та утримання клієнтів, прогнозування фінансових показників;

5) **роздрібна торгівля**: аналіз діяльності торгових точок, побудова профіля покупця, управління ресурсами;

6) **біржові трейдери**: вироблення оптимальної торгової стратегії, контроль рисків;

7) **телекомунікації та енергетика**: залучення клієнтів, цінова політика, аналіз відмов, прогнозування пікових навантажень, прогнозування надходження коштів;

8) **податкові служби і аудитори**: детекція підробок, прогнозування надходження в бюджет;

9) **фармацевтичні компанії**: прогнозування результатів майбутнього тестування препаратів, програми випробувань;

10) **медицина**: діагностика, вибір лікувальних впливів, прогнозування результату хірургічного втручання;

11) **управління виробництвом**: контроль якості, матеріально-технічне забезпечення, оптимізація технологічного процесу;

12) **учені та інженери**: побудова емпіричних моделей, заснованих на аналізі даних, розв'язання науковотехнічних задач.

До **основних проблем** технології ІАД відносять:

— технологія не може повністю замінити аналітика;

— складність розробки та експлуатації додатків (застосунків), основні аспекти складності:

- кваліфікація користувача
- складність підготовки даних
- великий процент хибних, недостовірних або безглуздих результатів
- висока вартість
- вимога наявності достатньої кількості репрезентативних даних

Перспективи подальшого розвитку технології ІАД:

— виділення типів проблемних областей з відповідаючими їм евристиками;

— створення формальних мов і логічних засобів, з допомогою яких будуть формалізуватися міркування;

— створення методів Data Mining, здатних не тільки добувати із даних закономірності, але й формувати певні теорії, які опираються на емпіричні дані;

— подолання істотного відставання можливостей інструментальних засобів Data Mining від теоретичних досягнень у цій сфері.

У прикладних задачах аналізу даних за даними вибірки часто необхідно визначити закон розподілу випадкової величини, що є одним із основних завдань статистичного аналізу.

Закон розподілу випадкової величини – співвідношення між значеннями випадкової величини та її ймовірностями.

Закон розподілу може бути заданий у вигляді таблиці, графіку, формули.

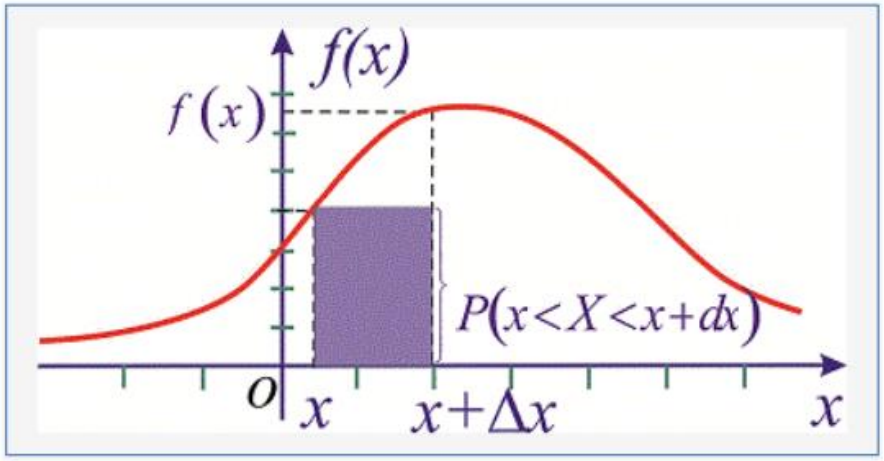
Функція розподілу (інтервальна функція розподілу) – це функція F(x), яка визначає ймовірність того, що випадкова величина X у результаті випробування (вимірювання) прийме значення, менше за x:

$$\boldsymbol{F(x) = p(X < x)}$$

Щільність ймовірності (диференціальна функція розподілу, функція розподілу щільності ймовірностей) – це функція f(x), яка характеризує щільність, з якою розподіляються значення величини у даній точці, та яка дорівнює першій похідній від функції розподілу:

$$\boldsymbol{f(x) = F(x)'}$$

Геометрично на графіку щільності ймовірностей f(x)dx відповідає площа прямокутника з основою dx і висотою f(x):



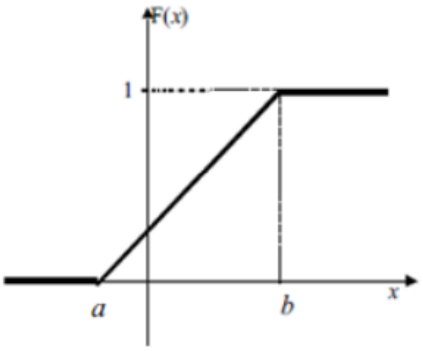
Рівномірний розподіл

Неперервна випадкова величина є рівномірно розподіленою на інтервалі [a, b], якщо її щільність ймовірності дорівнює деякій константі на цьому інтервалі й нулю поза ним.

Функція розподілу:

$$F(x)=\begin{cases}0, & x<a;\\ \dfrac{x-a}{b-a}, & a\le x\le b;\\ 1, & x>b.\end{cases}$$

Графік функції розподілу:



Математичне сподівання:

$$M(\xi)=\frac{b+a}{2}$$

Дисперсія:

$$D(\xi)=\frac{(b-a)^2}{12}$$

Середньоквадратичне відхилення:

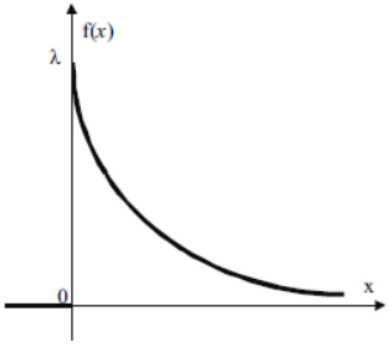
$$\sigma(\xi)=\frac{(b-a)}{2\sqrt{3}}$$

Експоненціальний розподіл

Неперервна випадкова величина має експоненціальний розподіл, якщо її щільність імовірності має вигляд:

$$f(x)=\begin{cases}\lambda\cdot e^{-\lambda\cdot x}, x\in[0,\infty)\\ 0\quad, x\in(-\infty,0)\end{cases}$$

Графік щільності ймовірності:



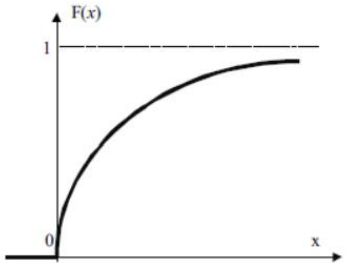
З показниковим розподілом мають справу при аналізі даних про тривалість безаварійної роботи різних машин і приладів, у теорії масового

обслуговування й надійності, у страховій справі, демографії й багатьох інших прикладних дисциплінах.

Функція розподілу:

$$F(x)=\begin{cases}0 & ,x\in(-\infty,0)\\ 1-e^{-\lambda x} & ,x\in[0,\infty)\end{cases}$$

Графік функції розподілу:



Математичне сподівання:

$$M(\xi)=\frac{1}{\lambda}$$

Дисперсія:

$$D(\xi)=\frac{1}{\lambda^2}$$

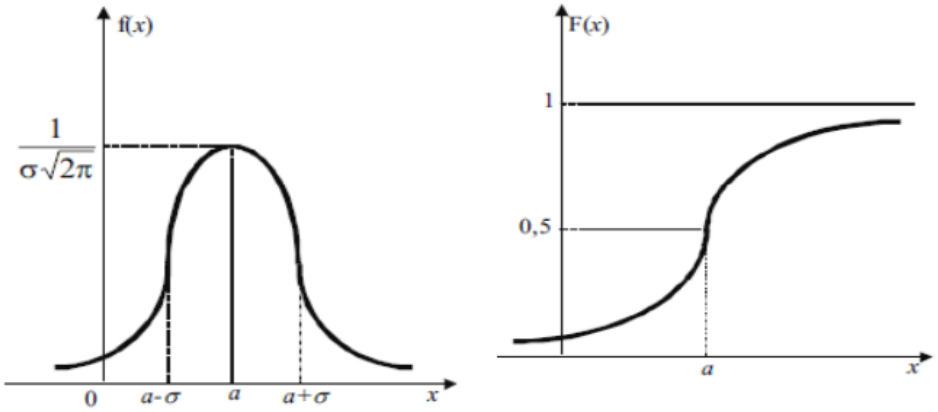
Середньоквадратичне відхилення:

$$\sigma(\xi)=\frac{1}{\lambda}$$

Нормальний розподіл

Безперервна випадкова величина розподілена нормально, якщо її щільність імовірності має вигляд:

Графік щільності ймовірності: Графік функції розподілу:



Таким чином, параметрами, які визначають нормальний розподіл, є математичне сподівання та середньоквадратичне відхилення.

Зміна параметра a зводиться до паралельного переносу графіка f(x) по осі OX. Для того щоб зрозуміти, як впливає параметр σ на цей графік, помітимо, що при зменшенні σ зростає fmax. Але площа фігури, обмежена графіком щільності ймовірностей і віссю OX, дорівнює 1, тому при зменшенні σ крива повинна швидше наближатися до осі OX далі від x = a і більш різко зростати поблизу цього значення.

Величину ξ - N(0,1) з нульовим середнім і одиничною дисперсією називають стандартною нормальною. Її щільність ймовірності й функція розподілу задаються формулами:

Нормальний розподіл або розподіл Гауса відіграє особливу роль, він є найбільш розповсюдженим на практиці, оскільки багато характеристик є нормально розподіленими

З нормальним розподілом легше працювати математичними засобами. Застосування нормального розподілу дозволяє виявити аномалії (підробки).

ТЕМА 14. Кластерний аналіз

Кластеризація (кластерний аналіз) – це задача розбиття множини об’єктів на групи, які називаються кластерами. Всередині кожної групи повинні виявитися "схожі" об’єкти, а об’єкти різних груп повинні бути як можна більше відмінні.

Перші розробки по кластерному аналізу відносяться до 30-х рр. минулого століття, але активний розвиток його методів та їх широке використання почалося в 60- 70-х роках XX століття з появою та розвитком ЕОМ. У процесі інтенсивного розвитку з’явилися нові методи, модифікації відомих алгоритмів, розширилася область застосування кластерного аналізу.

Головна **відмінність кластеризації від класифікації** полягає в тому, що перелік груп чітко не заданий і визначається в процесі роботи алгоритму.

Задачу кластерного аналізу доводиться розв’язувати на початкових етапах дослідження, коли про дані мало що відомо. Її рішення допомагає краще зрозуміти дані. Тому кластерний аналіз у списку задач Data Mining займає особливе положення – він дозволяє виділяти групи об’єктів, близьких за певними ознаками, без будь-якої попередньої інформації про розподіл об’єктів на групи. У цілому інтелектуальний аналіз тільки починається з автоматичного розбиття на кластери. Потім використовують інші методи Data Mining з метою визначення, що значить саме таке розбиття на кластери.

Кластерний аналіз є одним із методів багатомірного аналізу, який дозволяє об’єднувати в кластери змінні (об’єкти), схожі один на одного. Коли потрібно перетворити «гори» інформації у придатні для подальшого вивчення структури, використовують кластерний аналіз.

Переваги методу:

— дозволяє розбивати багатомірний ряд відразу по цілому набору параметрів;

— можна розглядати дані практично будь-якої природи (немає обмежень на вид досліджуваних об’єктів);

— можна обробляти значні обсяги інформації, різко стискати їх, робити компактними і наочними;

— може застосовуватися циклічно (проводиться доти, поки не буде досягнутий потрібний результат; а після кожного циклу можлива значна зміна спрямованості подальшого дослідження).

Кластерний аналіз має й свої **недоліки**:

— склад і кількість кластерів залежить від заданого критерію розбивки;

при перетворенні вхідного набору даних у компактні групи вхідна інформація може спотворюватися, окремі об’єкти можуть губити свою індивідуальність;

— часто ігнорується відсутність в аналізованій сукупності деяких значень кластерів.

Постановка задачі кластерного аналізу

Кластерний аналіз – це сукупність методів класифікації багатомірних спостережень або об'єктів, що засновані на визначенні поняття відстані між об'єктами з наступним виділенням з них груп спостережень (кластерів, таксонів). Вибір конкретного методу залежить від мети, з якою здійснюється кластеризація.

В цілому задача кластеризації є окремим випадком задачі **навчання без учителя**, що зводиться до розбивки наявної множини об'єктів даних на підмножини таким чином, щоб елементи однієї підмножини істотно відрізнялися по деякому набору властивостей від елементів всіх інших підмножин.

Формальний опис задачі кластеризації

Дана множина об'єктів $\boldsymbol{I}$:

$$\mathbf{I} = \{i_1, i_2, \ldots, i_j, \ldots, i_n\}, \quad \text{де}$$

n – кількість об'єктів кожен з яких представлений набором атрибутів (змінних, характеристик):

$i_j = \{x_{j1}, x_{j2}, \ldots, x_{jk} \ldots, x_{jm}\}$, m – кількість атрибутів

Кожна змінна (атрибут) може приймати значення з деякої множини:

$$x_{jk} = \{v^1_{jk}, v^2_{jk}, \ldots\}.$$

Потрібно побудувати множину кластерів $\boldsymbol{C}$:

$$C = \{c_1, c_2, \ldots, c_k, \ldots, c_g\},$$

у якій кожен кластер c_k містить схожі один на одного об'єкти множини $\boldsymbol{I}$:

$$c_k = \{i_j, i_p | \; i_j \in I, i_p \in I, d(i_j, i_p) < \sigma\}, \text{ де}$$

σ – величина, яка визначає міру близькості для включення об'єктів в один кластер;

$d(i_j, i_p)$ – міра близькості між об'єктами, яка називається ***відстанню***.

Необхідно знайти відображення $\boldsymbol{F}$ множини $\boldsymbol{I}$ на множину $\boldsymbol{C}$:

$$F: I \to C.$$

Невід'ємне $d(i_j, i_p)$ називається відстанню між об'єктами, коли виконуються наступні умови:

а) $d(i_j, i_p) > 0$ для усіх i_j та i_p

б) $d(i_j, i_p) = 0$ тоді і тільки тоді, коли $i_j = i_p$

в) $d(i_j, i_p) = d(i_p, i_j)$

г) $d(i_j, i_p) < d(i_j, i_r) + d(i_r, i_p)$

Якщо відстань $d(i_j, i_p)$ менше деякого значення σ, то кажуть, що об'єкти близькі та розміщують їх в один кластер. Якщо ні, то об'єкти будуть відмінні і їх розміщують у різних кластерах.

Більшість популярних алгоритмів при розв'язанні задачі кластеризації використовують **матрицю відстаней D**, рядки та стовпці якої відповідають елементам множини I.

Елементами матриці D є значення d(ij, ip) у рядку j та стовпці p. Матриця відстаней є симетричною відносно головної діагоналі, а її головна діагональ містить нулі:

$$D = \begin{pmatrix} 0 & d(i_1, i_2) & \dots & d(i_1, i_n) \\ d(i_2, i_1) & 0 & \dots & d(i_2, i_n) \\ \dots & \dots & \dots & \dots \\ d(i_n, i_1) & d(i_n, i_2) & \dots & 0 \end{pmatrix}$$

Процес кластерного аналізу

Етапи застосування кластерного аналізу у загальному вигляді є наступними:

1. Відбір вибірки об'єктів для кластеризації.

2. Визначення множини змінних, по яким будуть оцінюватися об'єкти вибірки. При необхідності – нормалізація та стандартизація значень змінних з метою отримання однакового внеску усіх змінних у розрахунок відстаней.

3. Обчислення значень міри подібності між об'єктами. На цьому етапі здійснюється вибір способу (формули) обчислення відстані між об'єктами (**метрики**).

4. Застосування методу кластерного аналізу для створення груп подібних об'єктів (кластерів). На цьому етапі здійснюється вибір методу та алгоритму кластеризації.

5. Подання результатів аналізу та перевірка їх вірогідності. Після отримання й аналізу результату можливе коригування обраної метрики й методу кластеризації для отримання оптимального результату.

Процес кластеризації залежить від обраного методу й майже завжди є **ітеративним**. Він може стати процесом, що включає цілий ряд експериментів на вибір різноманітних параметрів: міри відстані, типу стандартизації змінних, кількості кластерів і т.д. Однак кінцевою метою кластеризації є одержання змістовних відомостей про структуру досліджуваних даних.

Отримані результати вимагають подальшої інтерпретації, дослідження й вивчення властивостей і характеристик об'єктів для можливості точного опису сформованих кластерів.

У процесі кластеризації спочатку будують **вектор характеристик** для кожного об'єкту. Як правило, це набір числових значень (наприклад, вага, ріст людини). Однак є алгоритми, які працюють з якісними (категорійними) характеристиками. Сукупність об'єктів та змінних, які містять значення характеристик об'єктів, зручно представити у вигляді таблиці, кожен рядок якої відповідає одному об'єкту, а кожен стовпець – певній змінній, які міняються від

об’єкта до об’єкта (табл. 2.1). Така таблиця містить дані для **матриці вхідних даних**.

Таблиця 2.1

Набір даних, представлений у вигляді двомірної таблиці

	Характеристики			
Об’єкти	ПІП клієнта	Вік	Сімейний стан	Дохід
	Шевченко А.Д.	24	Одинокий	12500
	Новіцька Л.І.	22	Одружений	10520
	Іванов М.С.	35	Одружений	7580
	Савіцька Л.М.	45	Вдівець	6900

У разі необхідності проводять **нормалізацію** чи **стандартизацію** даних (це можна не робити у тих випадках, коли характеристики представлені числами, які знаходяться у одному числовому діапазоні).

Найбільш важким вважається визначення подібності об’єктів, які задаються введенням **відстані між об’єктами**. Відстань між об’єктами є критерієм, на основі якого здійснюється порівняння двох об’єктів та визначається їх подібність.

Міри відстаней

Визначення відстані між об’єктами є основним моментом дослідження, від якого залежать кінцеві варіанти розбиття на кластери.

розбиття на кластери. Об’єкт даних розглядається як точка в багатомірному метричному просторі, кожному виміру якого відповідає деяка властивість (атрибут) об’єкта, а **метрика** – є функція від значень даних властивостей.

Від типів вимірів цього простору, які можуть бути як числовими, так і категоріальними, залежить вибір **алгоритму кластеризації даних і використовувана метрика**. Цей вибір продиктований розходженнями в природі різних типів атрибутів.

Метрика – міра близькості об’єктів. Існує багато метрик.

Інтервальні (метричні) шкали

У задачах кластерного аналізу часто використовують наступні міри відстаней.

1. **Відстань Евкліда**: порівнює близькість двох об’єктів по великому числу ознак, є найбільш поширеною й обчислюється за формулою обчислення геометричної відстані у багатомірному просторі:

$$d_E(x_i, x_j) = \sqrt{\sum_{t=1}^{m} (x_{it} - x_{jt})^2}$$

З геометричної точки зору евклідова відстань буде безглуздою, якщо ознаки визначені у різних одиницях. Для коригування ситуації вдаються до нормалізації кожної ознаки.

2. **Відстань Хеммінга** (або **Манхетенська відстань**, відстань міських кварталів): є середнім різниць по координатам, дає ті ж результати, що й евклідова відстань, проте дозволяє зменшувати вплив окремих викидів (вони не підносяться до квадрату), є мірою відмінності об’єктів, що задаються атрибутивними ознаками, обчислюється за формулою:

$$d_H(x_i, x_j) = \sum_{t=1}^{m} \left| x_{it} - x_{jt} \right|$$

3. **Квадрат відстані Евкліда**: використовують для надання ваги більш віддаленим один від одного об'єктам, обчислюється за формулою:

$$d_E^2(x_i, x_j) = \sum_{t=1}^{m} (x_{it} - x_{jt})^2$$

4. **Відстань Чебишева**: є корисною у випадку, коли бажають визначити два об'єкти як різні, якщо вони відрізняються по одній координаті (по одному виміру), дорівнює максимальній відстані між об'єктами (є грубою мірою, частина інформації губиться), обчислюється за формулою:

$$d_\infty(x_i, x_j) = \max_{1 \le t \le m} \left| x_{it} - x_{jt} \right|.$$

5. **Степенева відстань**: є узагальненням Евклідової відстані, використовується для збільшення або зменшення ваги, яка відноситься до розмірності, для якої відповідні об'єкти сильно відрізняються, обчислюється за формулою:

$$d_{cm}(x_i, x_j) = \sqrt[r]{\sum_{t=1}^{m} (x_{it} - x_{jt})^p}$$

Де r та p – параметри, які визначає користувач.

Параметр p відповідає за поступове зважування різниць по окремим координатам, параметр r відповідальний за прогресивне зважування більших відстаней між об'єктами. Якщо обидва параметри r і p - рівні двом, то ця відстань збігається з відстанню Евкліда.

6. **Відстань Махаланобиса** (узагальнена евклідова відстань): це відстань від точки спостереження до центра ваги в багатомірному просторі ознак:

$$d_M(x_i, x_j) = (x_i - x_j) S^{-1} (x_i - x_j)^t,$$

де S – коваріаційна матриця (m × m) (m – кількість ознак об'єкта, відібрана для аналізу), яка розраховується за формулою

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^t$$

(n – кількість об'єктів, відібрана для аналізу).

7. **Пікова відстань**: передбачає незалежність між випадковими змінними, відстань в ортогональному просторі (на практиці змінні не є незалежними), обчислюється за формулою:

$$d_L(x_i, x_j) = \frac{1}{m} \sum_{t=1}^{m} \frac{\left| x_{it} - x_{jt} \right|}{x_{it} - x_{jt}}$$

Вибір метрики – відстані (критерію подібності) лежить повністю на дослідникові, який здійснює кластеризацію даних. При виборі різних мір результати кластеризації можуть суттєво відрізнятися.

Номінальні шкали

Для проведення **ієрархічного кластерного аналізу при використанні** шкал такого типу дані повинні бути підготовлені по-іншому, ніж було описано вище.

Кластерний аналіз такого типу виконується не на вхідній матриці даних, а на **таблиці спряженості**. У таблиці вхідних даних кожен рядок відносився до певного об'єкта. Тепер же рядок таблиці даних повинен відповідати категоріям однієї номінальної змінної, а стовпці - іншої.

Категорії першої змінної розглядаються як об'єкти, які треба розбити на кластери, а категорії другої - як змінні. У комірках таблиці містяться частоти - число об'єктів, у яких виявлено відповідне поєднання категорій.

Для порівняння між собою двох об'єктів – рядків таблиці спряженості (x і y), тобто для визначення відстані між ними, використовують частотні міри.

1. Міра χ2: обчислюється з використанням формули Пірсона, в якій розраховується сума квадратів стандартизованих залишків по всіх комірках таблиці спряженості, що належать двом певних об'єктам. Як відстань між категоріями використовується квадратний корінь із значення критерію χ2 :

$$disp(x,y) = \sqrt{\sum_k \frac{(f_o^k - f_t^k)^2}{f_t^k}}$$

де k - номер комірки в таблиці спряженості; $fe\ k$ - спостережувана частота в k-й комірці (наприклад, кількість об'єктів з таким поєднанням факторів; $ft\ k$- очікувана частота в k-й комірці.

Комірки з більш високими стандартизованими залишками роблять більш вагомий внесок у чисельне значення критерію χ2 , а отже, і у відстань між двома об'єктами, що відповідають перетину рядка та стовпця. Таким чином, чим більше великих стандартизованих залишків, тим більша відстань між рядками.

2. Міра φ 2 : при розрахунку відстані між двома рядками таблиці спряженості проводиться нормалізація; перед добуванням квадратного кореня вона поділяється на загальну суму спостережуваних частот, тобто загальне число об'єктів у двох рядках таблиці спряженості:

$$disp(x,y) = \sqrt{\frac{\sum_k \frac{(f_o^k - f_t^k)^2}{f_t^k}}{\sum_k f_t^k}}$$

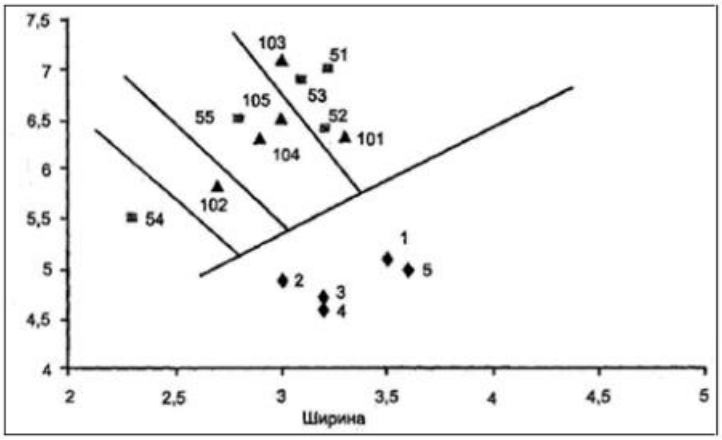
Міра φ2 змінюється від 0 до 1. Цей критерій рекомендований для таблиць спряженості з двох рядків і двох стовпців, проте в ієрархічної кластеризації осмислені результати виходять і при більшому числі стовпці таблиці спряженості.

Представлення результатів

Результатом кластерного аналізу є набір кластерів, що містять об'єкти вхідної множини даних. Кластерна модель повинна описувати кластери і належність кожного об'єкта до певного кластера.

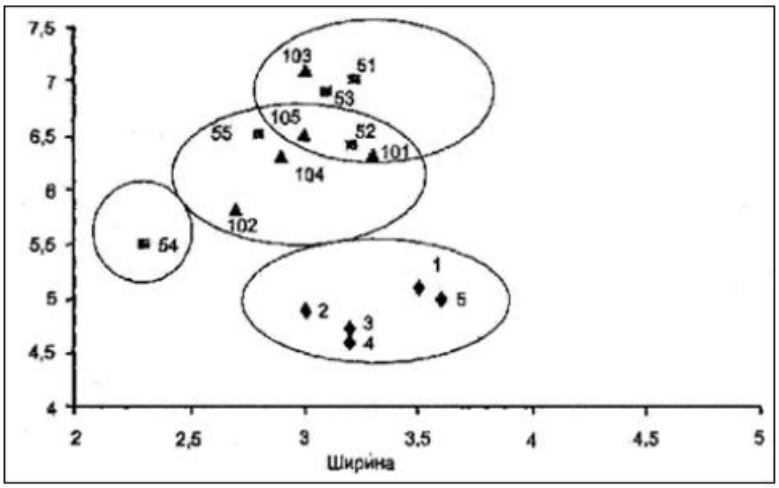
1. Для невеликого числа об'єктів, що характеризуються двома змінними, результати кластерного аналізу зображують графічно.

— Елементи представляються крапками, кластери розділяються прямими, які описуються лінійними функціями.



— Якщо кластери не можна розділити прямими, то рисуються ламані лінії, які описуються нелінійними функціями.

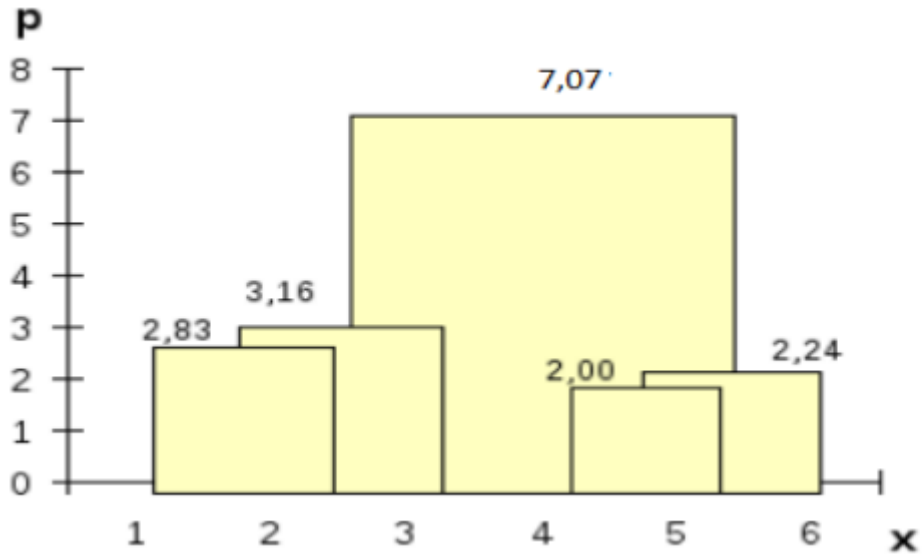
— У випадку якщо елемент може належати декільком кластерам, можна використати діаграми, які частково перекриваються.



2. Деякі алгоритми не просто відносять елемент до одного із кластерів, а визначають імовірність його приналежності. У цьому випадку зручніше представляти результат їх роботи у вигляді таблиці. У ній рядкам відповідають об’єкти вхідної множини, стовпцям– кластери, а в комірках вказують ймовірність належності об’єкта до кластера.

3. Результати кластеризації можуть бути представлені також у вигляді діаграми. Ряд алгоритмів кластеризации будують ієрархічні структури кластерів. Послідовність об’єднання кластерів графічно може бути представлена у вигляді дендрограми.

У таких структурах самий верхній рівень відповідає всій множині об’єктів, тобто одному-єдиному кластеру. На наступному рівні він ділиться на декілька кластерів. Кожний з них ділиться ще на декілька і т.д. Побудова такої ієрархії може відбуватися доти, поки кластери не будуть відповідати окремим об’єктам. Такі діаграми називаються дендограммами (dendrograms). Способів побудови дендограмм існує багато.



Приклад дендрограми

Огляд алгоритмів кластеризації

Велика кількість методів кластерного аналізу різниться не тільки мірами подібності, а й алгоритмами кластеризації. Можна виділити наступні класифікації алгоритмів кластеризації:

1. **Ієрархічні та плоскі алгоритми**:

a. **ієрархічні** алгоритми або алгоритми таксономії будують систему вкладених розбиттів вибірки на кластери й отримують на виході дерево кластерів, коренем якого є вся вибірка, а листами - найбільш дрібні кластери;

b. **Плоскі** алгоритми будують одне розбиття об’єктів вибірки на кластери що не перетинаються.

2. **Чіткі та нечіткі алгоритми**:

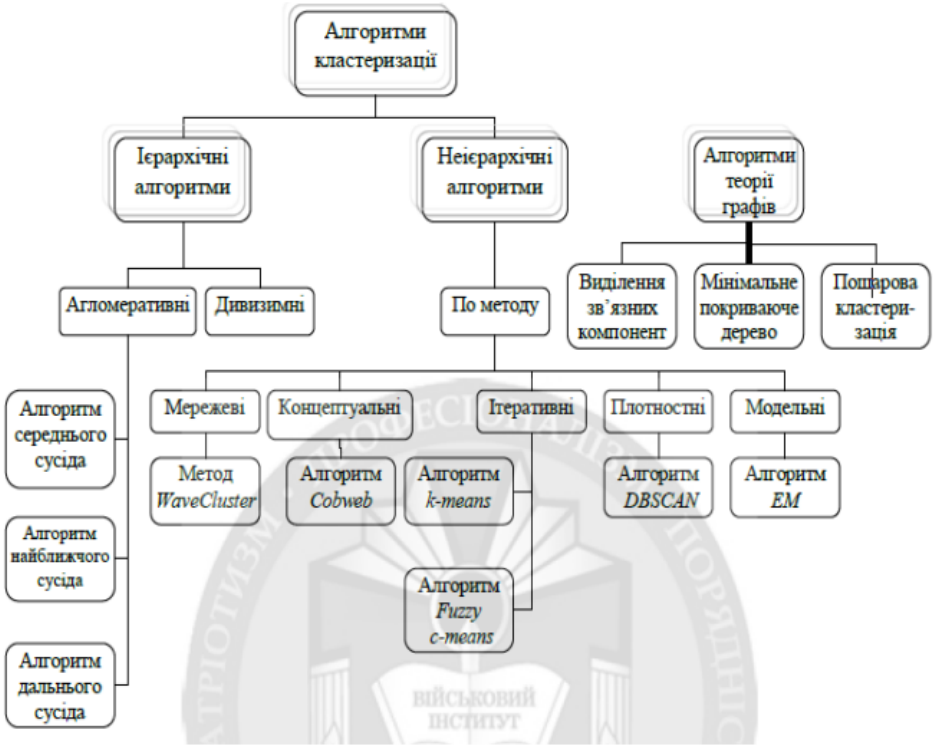
a. **чіткі** алгоритми (або непересічні) кожному об’єкту вибірки ставлять у відповідність номер кластера, тобто кожен об’єкт належить тільки одному кластеру;

b. **нечіткі** алгоритми (або пересічні) кожному об’єкту ставлять у відповідність набір дійсних значень, що показують ступінь відношення об’єкта до кластерів, тобто кожен об’єкт належить до кожного кластера з деякою ймовірністю.

Ієрархічні методи кластерного аналізу використовуються при невеликих обсягах наборів даних. Перевагою ієрархічних методів кластеризації є їх наочність.

До недоліку ієрархічних алгоритмів можна віднести систему повних розбивок, що може бути зайвою в контексті розв’язуваного завдання.

Класифікація алгоритмів та методів кластерного аналізу



Необхідність розробки методів і алгоритмів неієрархічної кластеризації обумовлена тим, що вони дозволяють досягти значної гнучкості у багатоваріантному розрахунку кластерів.

До **недоліків** цих методів можна віднести необхідність визначення початкової кількості та положення центрів кластерів і зазначення умови зупинки роботи алгоритмів.

Таблиця 1

Порівняльна таблиця алгоритмів

№ п/пр	*Алгоритм кластеризації*	*Форма кластерів*	*Вхідні дані*	*Результати*
1	Ієрархічний	Довільна	Число кластерів або поріг відстані для усікання ієрархії	Бінарне дерево кластерів
2	k-середніх	Гіпер-сфера	Число кластерів	Центри кластерів
3	c-середніх	Гіпер-сфера	Число кластерів, ступінь нечіткості	Центри кластерів, матриця належності
4	Виділення зв'язних компонент	Довільна	Поріг відстані R	Деревовидна структура кластерів
5	Мінімальне покриваюче дерево	Довільна	Число кластерів або поріг відстані для видалення ребер	Деревовидна структура кластерів
6	Пошарова кластеризація	Довільна	Послідовність порогів відстаней	Деревовидна структура кластерів з різними рівнями ієрархії

Оцінка якості кластеризації може бути проведена на основі наступних процедур:

— ручна перевірка;

— установлення контрольних точок і перевірка на отриманих кластерах;

— визначення стабільності кластеризації шляхом додавання в модель нових змінних;

— створення й порівняння кластерів з використанням різних методів: різні методи кластеризації можуть створювати різні кластери, і це є нормальним явищем. Однак створення схожих кластерів різними методами вказує на правильність кластеризації.

Застосування кластерного аналізу

Техніка кластеризації застосовується у самих різноманітних сферах. Головна задача – розбиття багатомірного ряду досліджуваних значень (об'єктів, змінних, ознак) на однорідні групи, кластери. Тобто, дані класифікуються та структуруються.

Питання, яке задає дослідник при використанні кластерного аналізу – як організувати багатомірну вибірку у наочні структури.

Приклади використання кластерного аналізу:

1. В біології – для визначення видів тварин на Землі

2. У медицині – для класифікації захворювань по групам симптомів і способам терапії

3. В психології – для визначення типів поведінки особистості у певних ситуаціях

4. В економічному аналізі – при вивченні та прогнозуванні економічної депресії, дослідженні кон'юнктури (наприклад, ринку окремих продуктів)

5. У різноманітних маркетингових дослідженнях

Кластерний аналіз має важливе значення у економічному аналізі. Він дозволяє вичленувати з великої сукупності періоди, де значення відповідних параметрів максимально близькі і де динаміка найбільш схожа.

Кластерний аналіз є кількісним інструментом дослідження соціально-економічних процесів, для опису яких необхідно багато характеристик. Він дозволяє вибірку розбити на декілька груп по досліджуваній ознаці, проаналізувати групи (як групуються змінні), групування об'єктів (як групуються об'єкти). З допомогою цього методу розв'язують задачі сегментації ринку, аналізують сільськогосподарські підприємства для порівняння продуктивності тощо.

ТЕМА 15. Алгоритми пошуку асоціативних правил

До **алгоритмів пошуку** асоціативних правил відносять:

1) **алгоритм AIS** – перший алгоритм пошуку асоціативних правил, розроблений у 1993 році співробітниками центру BM Almaden, генерує множини наборів та робить розрахунки під час сканування бази даних;

2) **алгоритм SETM** – передбачав використання мови SQL для обчислень, які також здійснювалися у процесі сканування БД;

3) **алгоритм Apriori** дозволяє ефективно розв'язувати задачу пошуку асоціативних правил, не розглядаючи набори, які зустрічаються рідко;

4) **алгоритми AprioriTid та AprioriHybrid** – вдосконалення алгоритму Apriori, передбачають здійснення перекодування елементів на етапах ітерації з метою економії обчислювальних ресурсів;

5) **алгоритм DHP** – передбачає ймовірнісний підрахунок наборів на кожному етапі алгоритму Apriori;

6) **алгоритми PARTITION, DIC** – сканують БД шляхом розподілу її на блоки, які не перетинаються.

Перед застосуванням алгоритмів проводять **попередню обробку даних** (рис. 1):

— приводять усі дані до бінарного вигляду (рис. 2): кожній транзакції відповідає бінарний вектор, кількість позицій якого рівна кількості елементів у базі даних, значення у яких рівне 1 – якщо елемент є у наборі елементів транзакції та 0 – якщо елемент у наборі відсутній;

— змінюють структуру даних (рис. 2): усі елементи упорядковують (у алфавітному порядку чи у порядку зростання (для числових даних)).

Номер транзакции	Наименование элемента	Количество
1001	A	2
1001	D	3
1001	E	1
1002	A	2
1002	F	1
1003	B	2
1003	A	2
1003	C	2
...	...	...

Рис. 1. Звичайний вигляд бази даних транзакцій

TID	A	B	C	D	E	F	G	H	I	K	...
1001	1	0	0	1	1	0	0	0	0	0	...
1002	1	0	0	0	0	1	0	0	0	0	...
1003	1	1	1	0	0	0	0	0	1	0	...

Рис. 2. Нормалізований вигляд бази даних транзакцій

Виявлення усіх асоціацій, підтримка й достовірність яких перевищують заданий мінімум, є задачею з високою обчислювальною трудомісткістю. Тому для **скорочення простору можливих рішень** застосовують спеціальні прийоми, найбільш поширеним з яких є **виявлення частих наборів**.

Генерація асоціативного правила розбивається на два кроки:

1. Мінімальний поріг підтримки використовується для пошуку усіх частих наборів елементів у базі даних, що містить транзакції.

2. Обмеження мінімальної достовірності застосовується для цих наборів для формування правила.

У багатьох прикладних областях елементи (об'єкти, предмети) множини I природним чином об'єднуються у групи, які у свою чергу можуть бути об'єднані у більш загальні групи, утворюючи ієрархічну структуру елементів (рис. 3).

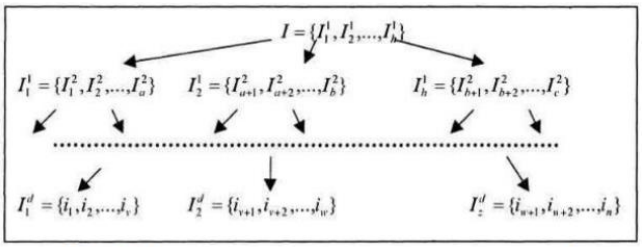


Рис. 3. Ієрархічне подання елементів множини I

Наявність ієрархії змінює уявлення про характеристики присутнього у транзакції елементу і – підтримка окремого елемента буде меншою, ніж підтримка групи, до якої він входить:

$$S(I) > S(i_j), \quad \text{де}$$

I – група в ієрархії;

i_j – j-тий елемент, що входить у дану групу.

Використання ієрархії при пошуку асоціативних правил дозволяє визначати зв'язки, що входять до більш високих рівнів ієрархії, оскільки підтримка набору може збільшуватися, якщо підраховується входження групи, а не її окремого елемента.

Загальна підтримка групи дорівнює сумі підтримки включених до неї елементів:

$$S(I) = \sum_{j=1}^{n} i_j ,$$

де n – число елементів у групі.

Приклад ієрархії продуктів представлено на рисунку 4.

Рис. 4. Ієрархія продуктів

На представленій ієрархічній схемі кетчуп та макаронні вироби мають багато підвидів. При обробці касових чеків отримують інформацію тільки про конкретний товар. Однак, якщо структура бази даних транзакцій дозволяє відображати ієрархію товарів, можна досліджувати усі утворені ними ієрархічні рівні.

Таким чином, аналіз транзакцій може бути розширений з метою отримання додаткових знань за рахунок здійснення пошуку:

— наборів, що часто зустрічаються;

— груп одного рівня ієрархії, що часто зустрічаються;

— змішаних наборів і груп рівнів ієрархії, що часто зустрічаються.

Ієрархічні асоціативні правила – асоціативні правила, виявлені для предметів, розташованих на різних ієрархічних рівнях

Синонімами є назви багаторівневі правила (multilevel rules) або узагальнені правила (generalized rules).

Стратегії ведення пошуку ієрархічних асоціативних правил

Існує кілька підходів до пошуку ієрархічних асоціативних правил.

Найчастіше використовуються спадні методи, коли часті предметні набори досліджуються на кожному ієрархічному рівні, починаючи з першого й закінчуючи рівнем з найбільшою деталізацією.

Тобто, як тільки виявляються всі часті предметні набори на першому рівні, починається пошук частих предметних наборів на другому й т.д.

На кожному рівні для знаходження частих наборів може використовуватися будь-який алгоритм, наприклад Apriori та його модифікації.

Відомо кілька стратегій ведення пошуку ієрархічних правил. Розглянемо їх.

1. **Використання однакового порогу мінімальної підтримки** S_{min}^{k} = const на всіх ієрархічних рівнях:

— **суть підходу**: при пошуку правил одноразово задається певний поріг мінімальної підтримки (наприклад, 5%), при досягненні якого набір вважається частим, у протилежному випадку – набір не включається до списку правил;

— **перевага підходу**: висока швидкість аналізу предметної області, обумовлена відмовою від оцінки часткових наборів, отриманих з тих з них, які недостатньо часто зустрічаються;

— **недолік підходу**: ризик пропустити тонкі асоціації на нижніх рівнях ієрархії, а саме перевагу користувачів одного виробника або зв'язку певних моделей товарів.

Наявний недолік намагаються обійти шляхом зниження мінімального рівня підтримки. Як наслідок - поява десятків і сотень незмістовних правил з низькою підтримкою й достовірністю.

2. **Зниження порогу мінімальної підтримки** при переході до нижчих ієрархічних рівнів. Можливі варіанти зниження: індивідуально для кожної підгрупи або функціонально.

— **функціональний вид** 1: пов’язаний з кількістю підкатегорій або з порядковим номером рівня (наприклад, якщо категорія «Комп'ютери» з рівнем підтримки $S^2 = 0{,}1$ ділиться на дві підкатегорії: «настільний» та «портативний», то в обох підкатегоріях $S^3 = S^2/2 = 0{,}05$);

— **функціональний вид 2**: пороги мінімальної підтримки встановлюються винятково в залежності від рівня, скільки б там підкатегорій не було:

$$S_{\min}^{k} = S_{\min}^{1} / k.$$

Останній підхід дозволяє піти набагато далі по ієрархії у пошуку асоціацій. Наприклад, якщо для всієї бази даних транзакцій прийнято мінімальний рівень підтримки $S^1_{min} = 0,2$, для інших категорій він буде рівним:

— для категорії «комп'ютери» (2 рівень): 0,2/2 = 0,1;

— для категорії «настільні комп'ютери» (3 рівень): 0,2/3=0,067;

— для категорії «настільні комп'ютери Dell» (4 рівень): 0,2/4=0,05 і т.д.;

3. **Міжрівнева фільтрація** заснована на понятті рівня проходу (level passage): відносно високий для верхніх рівнів ієрархії граничний рівень підтримки зберігається на нижніх у першому проході. Потім, у кожному проході, цей рівень знижується:

— ті товари низького рівня, які мають асоціації, ідентифікуються раніше, ніж їхні батьківські категорії, які, можливо, і не будуть мати необхідної підтримки;

— рекомендується виконувати три-чотири проходи бази даних транзакцій для сполучення рівнів довіри представників різних рівнів.

Властивість антимонотонності

Ця властивість підтримки – властивість антимонотонності, слугує для зменшення простору пошуку. Вона використовується у алгоритмі Apriori. Розглянемо її детальніше.

Властивість антимонотонності: підтримка набору елементів не може перевищувати підтримку будь-якої з його підмножин:

$$S_F \le S_E \ , \ \ якщо \ E \subset F$$

Наприклад, підтримка 3-елементного набору: {пиво, вода, чіпси} буде завжди меншою або рівною підтримці 2-елементних наборів:

{пиво, вода}, {вода, чіпси}, {пиво, чіпси}

Це пояснюється тим, що будь-яка транзакція, що містить {пиво, вода, чіпси}, містить також і набори {пиво, вода}, {вода, чіпси}, {пиво, чіпси}, причому обернене невірно.

Інше формулювання властивості антимонотонності: з ростом розміру набору елементів підтримка зменшується або залишається такою ж.

Із вище сказаного слідує, що будь-який k-елементний набір буде часто зустрічається тоді й тільки тоді, коли всі його (k-1)-елементні підмножини будуть часто зустрічаються.

Розглянемо малюнок, що ілюструє набір елементів I – {A, B, C, D} (рис. 5).

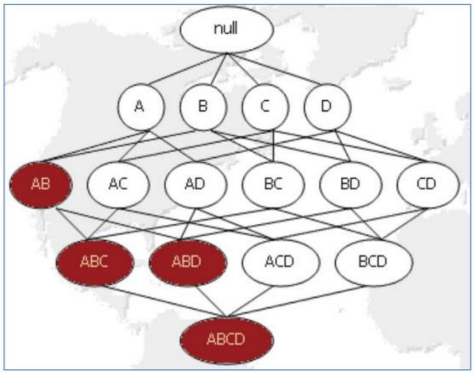


Рис. 5. Графічне зображення антимонотонності

Припустимо, що набір з елементів {A, B} має підтримку нижче заданого порогу й не є набором, який часто зустрічається. Тоді, відповідно до властивості антимонотонності, всі його супермножини також не є наборами, які часто зустрічаються й відкидаються. Вся ця вітка, починаючи з {A, B}, виділена фоном – вона відкидається. Використання цієї евристики дозволяє істотно скоротити простір пошуку.

Етапи алгоритму Apriori

Алгоритм Apriori визначає набори, які часто зустрічаються, за декілька етапів.

Кожен етап передбачає визначення усіх наборів елементів, які часто зустрічаються й складається із двох кроків:

1) **формування кандидатів** (candidate generation) ;

2) **підрахунок підтримки кандидатів** (candidate counting).

На і-му етапі:

1) на кроці формування кандидатів алгоритм створює множину кандидатів з і-елементних наборів, підтримка яких поки не обчислюється;

2) на кроці підрахунку кандидатів алгоритм сканує множину транзакцій:

— обчислюючи підтримку наборів-кандидатів;

— відкидаючи після сканування тих, підтримка яких менша за визначений користувачем мінімум

— зберігаючи ті і-елементні набори, які часто зустрічаються.

Під час 1-го етапу обрана множина наборів-кандидатів містить всі 1-елементні часті набори. Алгоритм обчислює їх підтримку під час кроку підрахунку кандидатів.

Опишемо алгоритм Apriori по крокам:

Крок 1. Привласнюється k = 1 і виконується відбір всіх 1- елементних наборів, у яких підтримка більша мінімально заданої користувачем S_{min}.

Крок 2. Привласнюється k = k + 1.

Крок 3. Якщо не вдається створити k-елементні набори, то алгоритм завершується, інакше виконується наступний крок.

Крок 4. Створюється множина k-елементних наборів кандидатів у часті набори:

— для цього необхідно об’єднати в k-елементні кандидати (k – 1) - елементні часті набори;

— кожен кандидат m Є M_k формується шляхом додавання до (k – 1)-елементного частого набору p останнього елемента з іншого (k – 1)-елементного частого набору q;

— елемент, який додається з набору q, по порядку є вищим, ніж останній елемент набору p;

— усі k – 2 елементи обох наборів p і q однакові.

Крок 5. Для кожної транзакції T з множиниD обираються кандидати з множини M_k , присутні у транзакції T – формується множина M_t .

Крок 6. У побудованій множині M_t видаляється кожен набір, підтримка якого менша за задану користувачем S_{min} – формується множина L_k . Повернення до кроку 2.

Результатом роботи алгоритму є об'єднання усіх множин L_k при усіх k: $\{L_1, L_2,..., L_k\}$.

Приклад застосування алгоритму Apriori

Розглянемо роботу алгоритму Apriori на прикладі множини об'єктів I = {шоколад, чіпси, кокоси, вода, пиво, горіхи}, які є товарами (табл. 1) при порогових мінімальній підтримці S_{min}=0,5 та мінімальній достовірності C_{min} = 0,75.

Таблиця 1

Прайс лист товарів

Ідентифікатор	Назва товару	Ціна
0	Шоколад	30.00
1	Чіпси	12.00
2	Кокоси	10.00
3	Вода	4.00
4	Пиво	14.00
5	Горіхи	15.00

Множина D транзакцій – наборів товарів з множини I представлена у таблиці 2.

Таблиця 2

Набори товарів у транзакціях множини ***D***

(*число транзакцій – 4*)

Номер транзакції	Номер товару	Найменування товару	Ціна
0	1	Чіпси	12.00
0	3	Вода	4.00
0	4	Пиво	14.00
1	2	Кокоси	10.00
1	3	Вода	4.00
1	5	Горіхи	15.00
2	5	Горіхи	15.00
2	2	Кокоси	10.00
2	1	Чіпси	12.00
2	2	Кокоси	10.00
2	3	Вода	4.00
3	2	Кокоси	10.00
3	5	Горіхи	15.00
3	2	Кокоси	10.00

Задача знаходження асоціативних правил розбивається на дві підзадачі:

1. Знаходження усіх наборів елементів, які задовольняють мінімальному порогу підтримки S_{min} = 0,5.

2. Генерація правил із знайдених наборів елементів з достовірністю, яка задовольняє заданому порогу C_{min} = 0,75.

1. Знаходження наборів елементів

Крок 1. Привласнюємо k = 1 та формуємо множину M_1 кандидатів (одноелементних наборів товарів), у яких підтримка більша за S_{min} = 0,5(табл. 3).

Таблиця 3

Множина M_1 одноелементних наборів товарів

№	Набір	Підтримка S
1	{0}	0/4 = 0
2	{1}	2/4 = 0,5
3	{2}	3/4 = 0,75
4	{3}	3/4 = 0,75
5	{4}	1/4 = 0,25
6	{5}	3/4 = 0,75

Заданій мінімальній підтримці відповідають кандидати 2, 3, 4 та 6 з товарами 1, 2, 3 та 5 відповідно: L_1 = {{1}, {2}, {3}, {5}}.

Відсікаються кандидати 1 та 6 з товарами 0 і 4: {{0}, {4}}.

Крок 2. Привласнюємо k = 2 та формуємо множину M_2 кандидатів (двоелементних наборів товарів), у яких підтримка більша за S_{min} = 0,5(табл. 4).

Таблиця 4

Множина M_2 двоелементних наборів товарів

№	Набір	Підтримка S
1	{1,2}	0,25
2	{1,3}	0,5
3	{1,5}	0,25
4	{2,3}	0,5
5	{2,5}	0,75
6	{3,5}	0,5

Заданій мінімальній підтримці відповідають кандидати 2, 4, 5 та 6 з товарами відповідно:

L_2 = {{1,3}, {2,3}, {2,5}, {1,5}}.

Відсікаються кандидати 1 та 3: {{1,2}, {4}}.

Крок 3. Привласнюємо k = 3 та формуємо множину M_3 кандидатів (трьохелементних наборів товарів), у яких підтримка більша за S_{min} = 0,5 (табл. 5).

Таблиця 5

Множина M_3 трьохелементних наборів товарів

№	Набір	Підтримка S
1	{2,3,5}	0,5

На даному кроці отримуємо: L_3 = {{2,3,5}}.

Так як 4-х елементні набори створити не вдасться, то алгоритм завершується

Результатом роботи алгоритму Apriori є множина наборів, серед яких буде здійснюватися пошук асоціативних правил:

$$L = L_1 \cup L_2 \cup L_3 = \{\{1\}, \{2\}, \{3\}, \{5\}, \{\{1,3\}, \{2,3\}, \{2,5\}, \{3,5\}, \{2,3,5\}\}.$$

2. Генерація правил

На отриманій множині наборів можуть бути згенеровані правила, представлені у таблиці 6. Для кожного з цих правил знаходять достовірність.

Таблиця 6

Згенеровані асоціативні правила

№	Правило	Достовірність С
1	«*якщо* 1 *то* 3»	1
2	«*якщо* 2 *то* 3»	2/3=0,67
3	«*якщо* 2 *то* 5»	1
4	«*якщо* 2 *то* 3 і 5»	2/3=0,67
5	«*якщо* 3 *то* 1»	2/3=0,67
6	«*якщо* 3 *то* 2»	2/3=0,67
7	«*якщо* 3 *то* 5»	2/3=0,67
8	«*якщо* 3 *то* 2 і 5»	2/3=0,67
9	«*якщо* 5 *то* 2»	1
10	«*якщо* 5 *то* 3»	2/3=0,67
11	«*якщо* 5 *то* 2 і 3»	2/3=0,67

Із згенерованих асоціативних правил залишаємо ті, достовірність яких не менша за C_{min} = 0,75. Цій умові задовольняють три правила:

«якщо 1 то 3», «якщо 2 то 5», «якщо 5 то 2».

Або:

Правило 1: «якщо чіпси то вода»

Правило 2: «якщо кокоси то горіхи»

Правило 3: «якщо горіхи то кокоси»

Асоціативні правила, що задовольняють мінімальному порогу підтримки S_{min} = 0,5, достовірність яких більша за задану порогову C_{min} = 0,75 знайдено.

ТЕМА 16. Генетичні алгоритми як інструмент data mining

Генетичні алгоритми як парадигма еволюційних обчислень

Проте нині генетичні алгоритми становлять стандартний інструментарій методів Data Mining для:

— комбінування шаблонів з правил індукції;

— налаштування та навчання нейронних мереж;

— пошук зразків у даних;

— відкриття шаблонів у тексті.

Генетичні алгоритми є частиною більш загальної групи методів – **еволюційних обчислень**, які поєднують різні варіанти використання еволюційних принципів для досягнення поставленої мети.

До основних принципів еволюційної теорії відносять: **спадковість, мінливість та природний відбір**.

Ключову роль в еволюції відіграє природний добір: найпристосованіші особи краще виживають і приносять більше потомства, ніж менш пристосовані.

Завдяки передаванню генетичної інформації, що називається генетичним успадковуванням, нащадки успадковують від батьків основні властивості.

Генетична інформація передається з допомогою хромосом (містяться у клітинах, основна частина - ДНК). Під час розмноження здійснюється передача генетичної інформації нащадкам шляхом **схрещування хромосом**.

На спадковість впливають також **мутації** – випадкові спадкові зміни організму (полягають у зміні деяких ланок ДНК). Якщо мутаці відбулася у статевій клітині, то зміна передається нащадкам.

Важливе місце в еволюційній теорії відводиться поняттю популяції як елементарній еволюційній одиниці. **Популяція** — це сукупність особин певного виду організмів, які здатні до вільного схрещування, населяють певну територію і деякою мірою ізольовані від сусідніх популяцій.

У рамках кожної популяції відбувається процес розмноження — **репродукції** (Reproduction), що являє собою комбінацію послідовностей (strings, хромосом) у популяції для створення нової послідовності (нащадка). Нащадок бере частини позицій генів від обох батьків і матиме частину ознак кожного із них.

Ідея створення обчислювальної системи, наділеної механізмами мінливості та добору, була досить привабливою. Еволюційні обчислення почалися з розробки моделей еволюційного процесу, серед яких можна виділити наступні парадигми:

— генетичні алгоритми;

— еволюційні стратегії: алгоритми, створені як методи рішення оптимізаційних задач, засновані на принципах природної еволюції, оперують дійсними числами;

— еволюційне програмування: модифікація генетичного алгоритму з урахуванням можливостей комп'ютерних програм. При використанні цього методу популяція складається з закодованих відповідним чином програм, що піддаються впливові генетичних операторів схрещування і мутації для

знаходження оптимального рішення, яким вважається програма, що щонайкраще вирішує поставлену задачу.

Основні поняття генетичних алгоритмів

Генетичний алгоритм – евристичний алгоритм пошуку, який здійснюється шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

При описі генетичних алгоритмів використовуються означення, запозичені з генетики.

Популяція – кінцева множина особей (об'єктів).

Покоління (нове покоління, покоління нащадків) – чергова популяція у генетичному алгоритмі.

Особи (об'єкти), що входять у популяцію, представлені хромосомами з закодованими у них множинами параметрів задачі – рішень (точки у просторі пошуку).

Хромосоми – упорядковані послідовності генів: ланцюжки символів, з якими здійснюються подальші операції.

Перед початком роботи генетичного алгоритму усі вхідні змінні представляються у вигляді хромосом. Для кодування параметрів частіше усього застосовують наступні методи:

— двійковий формат: під параметр виділяють N біт (N для кожного параметра може бути різним);

— формат з плаваючою точкою.

Отримані представлення кожного параметра викладають у ланцюжок (рядок) біт – хромосому.

Ген (властивість, знак, детектор) – атомарний елемент хромосоми.

Алель – значення гена у конкретній позиції.

Генотип (структура) – набір хромосом даної особи.

Функція пристосованості (fitness function, функція оцінки) – міра пристосованості даної особи в популяції.

Функція пристосованості впливає на функціонування генетичних алгоритмів, повинна мати точне і коректне означення. Функція пристосованості:

— у задачах оптимізації – оптимізується (точніше кажучи, максимізується) і називається цільовою функцією;

— у задачах мінімізації цільова функція перетворюється, і проблема зводиться до максимізації;

— у теорії керування функція пристосованості може приймати вид функції похибки;

— у теорії ігор – вартісної функції.

На кожній ітерації генетичного алгоритму оцінюється пристосованість кожної особи даної популяції за допомогою функції пристосованості, і на цій основі створюється наступна популяція особей, що складають множину потенційних рішень проблеми.

Функція пристосованості є важливим поняттям у генетичних алгоритмах. Вона дозволяє оцінити ступінь пристосованості конкретних особей у популяції

і вибрати з них найбільш пристосовані відповідно до еволюційного принципу виживання «найсильніших» (найкраще пристосувалися).

Для реалізації концепції вибору вводиться спосіб зіставлення різних хромосом. Популяція обробляється за допомогою процедур репродукції, мінливості (мутацій), генетичної композиції, які імітують біологічні процеси:

1) випадкові мутації даних в індивідуальних хромосомах: найчастіше шляхом інвертування випадкових бітів (ген хромосоми) або складніше (з врахуванням предметної області);

2) переходи і рекомбінації генетичного матеріалу, що міститься в індивідуальних батьківських хромосомах;

3) міграції генів.

Для здійснення процедур, що імітують біологічні процеси еволюції, використовують генетичні оператори:

— оператор селекції: здійснює відбір індивідів з поточної популяції;

— оператор кросовера (Cross-over): реалізує операцію комбінування, схрещування (змішування) хромосом шляхом замін значень генів і утворення нових хромосом на їх місцях;

— оператор мутації (Mutation): спонтанне перетворення (видозміна) генів у хромосомі;

— оператор редукції (): реалізує операцію доведення кількості особин поточної популяції до початково визначеної величини.

У процесі виконання процедур на кожній стадії еволюції виходять популяції з дедалі досконалішими індивідами.

Представлення даних

Генетичні оператори легко описуються і застосовуються, даючи розумні результати у разі представлення даних у вигляді рядків бітів.

Однак проблемою є вирішення питання: як на практиці зашифрувати достатньо складні структури даних в рядки бітів?

Розглянемо задачу класифікації – приклад про футбольний матч. Нехай результати гри «Динамо» знаходяться в залежності від чотирьох параметрів.

Класифікація, представлена у вигляді дерева, може бути записана у вигляді набору гіпотез, що відповідають листкам дерева.

Як закодувати гіпотезу у вигляді рядка бітів?

1.Виділити по одному біту на кожен атрибут і на цільову функцію. Це кодування є найбільш ефективним: кожен рядок має однакову довжину і кожен рядок має сенс.

Недоліки: як представити гіпотезу, в якій не всі значення атрибутів специфіковані, а, навпаки, від деяких з них нічого не залежить?

Якщо це неможливо, то дерево доведеться «розгортати» до повного набору гіпотез, що неефективно. Можна було б представляти гіпотези з атрибутами без фіксованих значень рядками меншої довжини, але тоді було б незрозуміло, які атрибути задані, а які — ні.

2. Рішенням проблеми, що виникла, буде наступна конструкція: для кожного атрибута треба виділити стільки бітів, скільки в нього можливих

значень. Нуль в тій чи іншій позиції означатиме, що дане значення не зустрічається в посилці правила, а одиниця — що зустрічається.

1) якщо всі біти, що відповідають значенням даного атрибута рівні 1, це означає, що значення атрибута не грає ніякої ролі для застосування цього правила;

2) для значення функції залишається один біт – правила, де цільова функція не визначена чи дозволено будь-яке її значення, позбавлені смислу.

3) таким чином, у випадку гри «Динамо» чотири бінарних атрибута дадуть 8 біт плюс один біт значення цільової функції.

Ця система кодування має одне тонке місце: що робити, якщо в результаті кроссовера чи мутації в гіпотезі буде отриманий набір із символів 00.

Така гіпотеза не може бути застосована ні разу — її умови відповідають порожній множині прикладів. Шляхи вирішення проблеми:

1) не дозволити появу таких гіпотез: повторяти кроссовер чи мутацію поки результат буде розумним;

2) залишати такі гіпотези в популяції, але присвоювати їм нульове значення функції Fitness: тоді ці гіпотези будуть не застосовані на наступному кроці відбору.

Одна гіпотеза – це не одна гілка, а декілька гілок дерева рішень. Як закодувати декілька правил?

При фіксованому наборі атрибутів всі правила будуть мати постійну довжину, рівну сумарній кількості можливих значень атрибутів плюс один біт на значення цільової функції (або більше, якщо цільова функція може приймати більше двох значень). Тому можна просто записати правила підряд — ніякої багатозначності це не внесе.

Загальний опис генетичного алгоритму

Генетичні алгоритми застосовують для пошуку рішень у дуже великих, складних просторах пошуку.

Задача кодується таким чином, щоб її рішення могло бути представлене у виді вектора – хромосоми.

Випадковим чином створюється початкова популяція – деяка кількість початкових векторів.

Особи початкової популяції оцінюються з використанням функції пристосованості (фітнес-функції), у результаті чого кожному векторові (особі) привласнюється певне значення: пристосованість, що визначає ймовірність виживання особи, представленої даним вектором.

З використанням отриманих значень пристосованості здійснюється селекція – обираються особи (вектори), допущені до схрещування.

До відібраних особей (векторів) застосовують генетичні оператори (в основному схрещування (crossoveri) та мутацію (mutation)), створюючи наступне покоління.

Особи наступного покоління також оцінюються, потім відбувається селекція, застосовуються генетичні оператори і т.д.

У генетичному алгоритмі зберігається принцип природного добору: чим пристосованіший індивідуум (чим більше відповідне йому значення цільової функції), тим з більшою ймовірністю він буде брати участь у схрещуванні.

Імітуючи еволюцію в комп'ютері, можна уникати небажаних подій (найпристосованіший тигр може загинути від пострілу мисливця) і завжди зберігати життя кращому з індивідуумів поточного покоління. У цьому полягає методика стратегії елітизму, коли в наступне покоління відбираються особини з найкращими показниками.

Так моделюється еволюційний процес, що продовжується кілька життєвих циклів – поколінь, поки не буде виконано критерій зупинки алгоритму.

Критерієм зупинки процесу здійснення генетичного алгоритму може бути одна з подій:

— сформовано задану кількість поколінь, відпущених на еволюцію;

— популяція досягла заданої якості(наприклад, значення якості всіх особин перевищило задану порогову величину);

— вичерпання часу, відпущеного на еволюцію;

— досягнутий деякий рівень збіжності: особини в популяції стали настільки подібними, що дальше їх поліпшення відсувається надзвичайно повільно, і тому продовження здійснення ітерацій генетичного алгоритму стає недоцільним.

Таким чином, стратегія отримання рішень за допомогою генетичних алгоритмів може бути реалізована такими кроками:

1) ініціалізація популяції;

2) вибір батьків для репродукції і генетичних операторів (мутації, кросовера та інших);

3) виконання операцій генерування проміжної популяції індивідуумів і оцінка їх придатності;

4) вибір членів популяції для отримання нової генерації;

5) повторення кроків 1-З, поки не буде досягнуте деяке правило зупинки алгоритму.

Етапи генетичного алгоритму

1. **Ініціалізація**: створення, генерація початкової популяції особин (індивідуумів, хромосом) випадковим чином. Індивіди на першому етапі не обов'язково повинні бути найпристосованіші (генетичний алгоритм все одно приведе до життєздатної популяції), достатньо, щоб для них можна було підрахувати функцію пристосованості.

2. **Генерація наступного покоління**: полягає у моделюванні еволюційного процесу:

2.1. Оцінювання: обчислення функцій пристосованості для особей поточної популяції.

2.2. Сортування батьківського покоління у відповідності із значенням функції пристосованості.

2.3. Застосування генетичних операторів, пока не згенеровано достатню кількість екземплярів нового покоління (доля виживших – тих, хто залишаться в живих, є параметром генетичного алгоритму):

— селекції: відбір батьків для репродукції – особин поточної популяції, ймовірність виживання яких найбільша (залежить від значення їх функції пристосованості), способи відбору є різні: 1) особи для схрещення можуть обиратися випадковим чином серед тих, ймовірність виживання яких є більшою 2) для кожного з батьків обирається пара претендентів, серед яких обирають того, хто є більш пристосованим;

— розмноження: шляхом схрещення, потомок, який перейде у популяцію наступного покоління, успадковує риси обох особин-батьків, їх хромосоми об'єднуються (оператор кроссовер);

— мутацій: шляхом спонтанної зміни деяких генів хромосом у кількох випадково обраних особин, (доля мутантів є параметром генетичного алгоритму);

— застосування інших генетичних операторів.

2.4. Перехід до нового покоління шляхом часкового або повного знищення старої популяції, популяція наступного покоління в більшості реалізацій генетичних алгоритмів містить стільки ж особин, скільки й початкова, але внаслідок відбору пристосованість (значення цільової функції) у ній в середньому вища.

У кожному наступному поколінні буде спостерігатися виникнення абсолютно нових розв'язків задачі. Серед них будуть кращі та гірші рішення, але завдяки процедурі добору кількість кращих розв'язків буде зростати.

3. **Перевірка критерію зупинки** генетичного алгоритму: якщо критерій зупинки не досягнуто, повернення до кроку 2, якщо досягнуто – перехід до наступного кроку.

4. **Зупинка алгоритму**: з кінцевої популяції вибирається та особина, яка має краще значення функції пристосованості (фітнес-функції) і є шуканим рішенням – результатом здійснення генетичного алгоритму. За рахунок того, що кінцева популяція краща, ніж початкова, отриманий результат являє собою поліпшене рішення.

Таким чином, основними характеристиками генетичного алгоритму є:

— розмір популяції;

— оператор кросовера і ймовірність його використання;

— оператор мутації і її ймовірність;

— оператор селекції;

— оператор редукції;

— правило (критерій) зупинки процесу виконання генетичного алгоритму.

Генетичні алгоритми зручні тим, що їх легко розпаралелити. Наприклад, можна розбити покоління на кілька груп і працювати з кожною з них незалежно, змінюючи час від часу кілька хромосом. Існують також інші методи розпаралелювання генетичних алгоритмів.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Звенігородський О. С., Зінченко О. В., Чичкарьов Є. А., Кисіль Т. М. Штучний інтелект. Вступний курс : навчальний посібник. Київ : ДУТ, 2022. 193 с. URL:https://duikt.edu.ua/uploads/l_492_92652604.pdf
2. Іванченко А. Види машинного навчання: посібник для початківців. Acer Corner. 2024. URL: https://blog.acer.com/ua/discussion/2054/vidi-mashinnogo-navchannya-posibnik-dlya-pochatkivciv
3. Кларк Е. Підручник зі штучного інтелекту для початківців: ознайомтеся з основами ШІ. Guru99. 2025. URL:https://www.guru99.com/uk/ai-tutorial.html
4. Кононова К. Ю. Машинне навчання: методи та моделі: підручник для бакалаврів, магістрів та докторів філософії спеціальності 051 «Економіка» / К. Ю. Кононова. Харків: ХНУ імені В. Н. Каразіна, 2020. 301 с.
5. Лабораторний практикум з систем штучного інтелекту / уклад. О. Мосіюк. Житомир : Вид-во ЖДУ ім. Івана Франка, 2024. 104 с. URL: https://eprints.zu.edu.ua/40845/1/PraktykumAI.pdf
6. Машинне навчання : навчальний посібник / за науковою редакцією д.т.н., проф., В. В. Пасічника ; Т. М. Басюк, В. В. Литвин, Л. М. Захарія, Н. Е. Кунанець. 3-тє видання, стереотипне. Львів : «Новий Світ-2000», 2026. 330 с.
7. Мокін В. Б., Дратований М. В. Наука про дані: машинне навчання та інтелектуальний аналіз даних : електронний навчальний посібник. Вінниця : ВНТУ, 2024. 263 с. URL:https://pdf.lib.vntu.edu.ua/books/2024/Mokin_2024_263.pdf
8. Нікольський Ю. В., Пасічник В. В., Щербина Ю. М. Системи штучного інтелекту : навчальний посібник. Львів : Магнолія 2006, 2024. 279 с.
9. Обчислювальний інтелект : навч. посібник / О. Ю. Заковоротний, Т. О. Орлова, Д. В. Гриньов ; Нац. техн. ун-т "Харків. політехн. ін-т". Харків : НТУ "ХПІ", 2024. 264 с. URI: https://repository.kpi.kharkov.ua/handle/KhPI-Press/85777.
10. Основи обчислювального інтелекту : лаб. практикум / уклад. О. Ю. Заковоротний, Т. О. Орлова, Д. В. Гриньов, В. М. Сергієнко ; Нац. техн. ун-т "Харків. політехн. ін-т". Харків : НТУ "ХПІ", 2023. 170 с. URI: https://repository.kpi.kharkov.ua/handle/KhPI-Press/74045.
11. Солодовник Г. В. Методи та системи штучного інтелекту. Харків : ТОВ «ДІСА ПЛЮС», 2021.177 с. https://surl.lu/ulafnc

12. Стратегія розвитку штучного інтелекту в Україні : монографія / за ред. А. І. Шевченка. Київ : Інститут проблем розвитку штучного інтелекту, 2023. 305 с. URI: https://jai.in.ua/archive/2023/ai_mono.pdf
13. Троцько В. В. Методи штучного інтелекту: навчально-методичний і практичний посібник. Київ : Університет економіки та права «КРОК», 2020. 86 с. URI: https://library.krok.edu.ua/media/library/category/navchalni-posibniki/trotsko_0001.pdf
14. Штучний інтелект. Нейромережева обробка інформації : архітектури, навчання, застосування : навчальний посібник у 2-х ч. : Ч. 1 / О. Г. Руденко, О. О. Безсонов, С. П. Євсеєв, О. Б. Ахієзер, Ю. І. Зайцев ; за заг. ред. С. П. Євсеєва. Харків : НТУ «ХПІ», Львів : «Новий Світ-2000», 2025. 426 с.

Додаткова

1. Bata, M., Carriveau, R., & Ting, D. S. K. (2020). Short-term water demand forecasting using hybrid supervised and unsupervised machine learning model. *Smart Water*, 5(1). https://doi.org/10.1186/s40713-020-00020-y
2. Jung, G., & Choi, S.-Y. (2021). Forecasting foreign exchange volatility using deep learning autoencoder-lstm techniques. *Complexity*, 2021, 1–16. https://doi.org/10.1155/2021/6647534
3. Liu, Q., Li, Z., Ji, Y., Martinez, L., Ul Haq, Z., Javaid, A., Lu, W., & Wang, J. (2019a). Forecasting the seasonality and trend of pulmonary tuberculosis in Jiangsu Province of China using advanced statistical time-series analyses. *Infection and Drug Resistance*, Volume 12, 2311–2322. https://doi.org/10.2147/idr.s207809
4. Petropoulos, F., Kourentzes, N., Nikolopoulos, K., & Siemsen, E. (2018). Judgmental selection of forecasting models. *Journal of Operations Management*, 60(1), 34–46. https://doi.org/10.1016/j.jom.2018.05.005
5. Rubio, L., & Alba, K. (2022). Forecasting selected colombian shares using a hybrid ARIMA-SVR model. *Mathematics*, 10(13), 2181. https://doi.org/10.3390/math10132181
6. Salah, O. M., Mahdi, G. J. M., & Al-Latif, I. A. A. (2021). A modified ARIMA model for forecasting chemical sales in the USA. *Journal of Physics: Conference Series*, 1879(3), 032008. https://doi.org/10.1088/1742-6596/1879/3/032008
7. Sun, J. (2021). Forecasting COVID-19 pandemic in Alberta, Canada using modified ARIMA models. *Computer Methods and Programs in Biomedicine Update*, 100029. https://doi.org/10.1016/j.cmpbup.2021.100029

8. Tripathi, M., Kumar, S., & Inani, S. K. (2020). Exchange rate forecasting using ensemble modeling for better policy implications. *Journal of Time Series Econometrics*. https://doi.org/10.1515/jtse-2020-0013
9. Wang, Y.-w., Shen, Z.-z., & Jiang, Y. (2018). Comparison of ARIMA and GM(1,1) models for prediction of hepatitis B in China. *Plos One*, 13(9), Стаття e0201987. https://doi.org/10.1371/journal.pone.0201987
10. Zhu, N., Zhang, D., Wang, W., Li, X., Yang, B., Song, J., Zhao, X., Huang, B., Shi, W., Lu, R., Niu, P., Zhan, F., Ma, X., Wang, D., Xu, W., Wu, G., Gao, G. F., & Tan, W. (2020). A novel coronavirus from patients with pneumonia in China, 2019. *New England Journal of Medicine*, 382(8), 727–733. https://doi.org/10.1056/nejmoa2001017

Навчальне видання

ШТУЧНИЙ ІНТЕЛЕКТ ТА МАШИННЕ НАВЧАННЯ

Конспект лекцій

Укладачі: **Шебаніна** Олена В’ячеславівна
Тищенко Світлана Іванівна
Пархоменко Олександр Юрійович
Жебко Олександр Олегович
Коломієць Андрій Миколайович

Формат 60х84 1/16. Ум. друк. арк. 2.94.
Наклад 50 прим. Зам. № _____

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54020, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб’єкта видавничої справи ДК № 4490 від 20.02.2013