Київський національний університет
імені Тараса Шевченка

Марченко О.О., Россада Т.В.

Актуальні проблеми Data Mining

Навчально-методичний посібник

Київ — 2017

УДК 004.8

Марченко О.О., Россада Т.В. Актуальні проблеми Data Mining**:** Навчальний посібник для студентів факультету комп'ютерних наук та кібернетики. — Київ. — 2017. — 150 с.

Рекомендовано до друку вченою радою факультету комп'ютерних наук та кібернетики
(протокол № 10 від 15 червня 2017 р.)

У посібнику розглянуто актуальні проблемі Data Mining. Виклад зосереджено на задачах класифікації, кластеризації та побудови асоціативних правил.

Призначений для студентів фізико-математичних та технічних спеціальностей вищих навчальних закладів.

Автори:

Марченко О.О. — доктор фізико-математичних наук, професор кафедри математичної інформатики факультету комп'ютерних наук та кібернетики

Россада Т.В. — кандидат фізико-математичних наук, асистент кафедри теорії та технології програмування факультету комп'ютерних наук та кібернетики

Рецензенти:

д.ф.-м.н., проф. **Глибовець М.М.**

д.т.н., проф. **Скобелєв В.Г.**

ВСТУП

Data is just like crude. It's valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc to create a valuable entity that drives profitable activity; so must data be broken down, analyzed for it to have value[1].

Data is the New Oil, By Michael Palmer[29]

Термін Data Mining[2] (укр. видобуток даних, інтелектуальний аналіз даних) введений[3] Григорієм Пятецким-Шапіро у 1989 році[37]. З його визначенням, Data Mining — це процес виявлення в сирих даних раніше невідомих, нетривіальних, практично корисних і доступних інтерпретації знань[4], необхідних для прийняття рішень у різних сферах людської діяльності.

Формально, Data Mining — це побудова *моделі даних* [24].

На сьогоднішній день існує декілька підходів до побудови моделей даних, а саме:

- *статистичний* (англ. Statistical Modelling): базується на теорії та зосереджується на перевірці гіпотез;
- *на основі машинного навчання* (англ. Machine Learning): евристичний, концентрується на поліпшенні роботи агентів;
- *обчислювальний* (по суті — інтелектуальний аналіз даніх): інтеграція теорії та евристик, сконцентрований на єдиному процесі аналізу даних, включає евристику

---

[1] Дані – це лише сира нафта, що є цінною, проте без переробки не може бути реально застосовна, доки не буде перетворена в газ, пластик, хімікати тощо, щоб створити цінність, яка зумовлює прибутковість; аналогічно дані треба проаналізувати та зрозуміти, щоб вони стали дійсно цінними ().

[2] У перекладі: видобуток даних, інтелектуальний аналіз даних, виявлення знань у базах даних (англ. Knowledge discovery in databases, KDD)

[3] У сучасному розумінні цього слова. Насправді першими, хто використав це поняття, хоча і у негативному сенсі, були статистики: так вони називали спроби вивести інформацію, що не підтримується по умові. Так, наприклад, Мендель — основоположник вчення про спадковість, підтасовував результати своїх експериментів (точніше завершував їх, коли отримував потрібний результат).

[4] Під терміном знання (knowledge) тут мається на увазі сукупність відомостей, що утворює цілісний опис, відповідний деякому рівню поінформованості по описуваному об'єкту. Використання знань (knowledge deployment) означає дійсне застосування знайдених знань для досягнення конкретних переваг (наприклад, в конкурентній боротьбі за ринок).

даних, навчання, інтеграцію та візуалізацію результатів.

Традиційні методи аналізу даних (статистичні методи) і аналітична обробка в реальному часі (Online Analytical Processing, далі — OLAP) в основному орієнтовані на перевірку заздалегідь сформульованих гіпотез (verification-driven data mining) і на «грубий» розвідувальний аналіз, що становить основу оперативної аналітичної обробки даних, у той час як одне з основних положень Data Mining — пошук неочевидних закономірностей. Інструменти Data Mining можуть знаходити такі закономірності і будувати гіпотези про взаємозв'язки самостійно. Оскільки формулювання гіпотези щодо залежностей є найскладнішим завданням, перевага Data Mining в порівнянні з іншими методами аналізу є очевидним.

Суть і мету інтелектуального аналізу даних можна охарактеризувати таким чином: це технологія, призначена для пошуку у великих обсягах даних (англ. Big Data) *неочевидних*, *об'єктивних* і *корисних на практиці* закономірностей. Неочевидних — тобто таких, що не виявляються стандартними методами обробки інформації або експертним шляхом. Об'єктивних — тобто таких, будуть повністю відповідати дійсності (на відміну від суб'єктивної експертної думки). Корисних на практиці — тобто таких, яким можна знайти практичне застосування.

Нерідко Data Mining ототожнюють з виявленням знань у базах даних (англ. Knowledge Discovery in Databases), хоча більш правильно вважати Data Mining одним із кроків цього процесу (див. рис. 1).

Рисунок 1. Зв'язок Data Mining з іншими областями

Інтелектуальний аналіз даних — потужний засіб у рекламі, у бізнесі, у економіці тощо. Це не просто область науки, це область життєдіяльності, що є як науковою, оскільки включає в себе наукові дисципліни, так і прикладною, оскільки спеціалісту у сфері інтелектуального аналізу даних необхідні навички програмування та алгоритміки. Крім цього, це в якомусь сенсі культура та мистецтво, оскільки алгоритми Data Mining вимагають постійно шукати нові шляхи вирішення проблем.

Розділ 1. Основні поняття Data Mining

1.1. Поняття даних

Дослідження у сфері інтелектуального аналізу даних середини XX століття були зосереджені в основному на дослідженні табличних даних. Проте, з часом виникла потреба в обробці та аналізі як звичайних, так і специфічних текстових даних: html-документів, xml-файлів, jsn-файлів тощо. Особливість обробки та аналізу текстових даних у тому, що останні містять інформацію і в своїй структурі, і у семантичних зв'язках, що, в свою чергу, вимагає досліджень у сфері баз знань. Окрім того, сучасний стан розвитку інформаційних технологій зумовлює дослідження цікавих та складних об'єктів, що раніше не представляли інтересу, наприклад, соціальних мереж. Технологія Data Mining передбачає аналіз даних представлених у різних формах, придатних для зберігання, передачі та обробки.

У широкому розумінні, *дані* представляють собою факти, текст, графіки, картинки, звуки, аналогові або цифрові відеосегменти. Дані можуть бути отримані в результаті *вимірів*, *експериментів*, *арифметичних* і *логічних операцій*. Іншими словами, *дані* — це необроблений матеріал, що надається постачальниками даних і використовується споживачами для формування інформації на основі даних.

Найчастіше зустрічаються дані, що складаються із записів (англ. Record Data). Прикладами таких наборів даних є *табличні, матричні, документальні* та *транзакційні* (або *операційні*).

Табличні дані складаються із *записів*, кожен з яких складається з фіксованого набору *атрибутів. Транзакційні дані* представляють собою особливий тип даних, де кожен запис, що є *транзакцією*, включає *набір значень*. Прикладами *графічних даних* є: WWW-дані, молекулярні структури, графи, карти тощо. Перевагою графічних даних є простота сприйняття у порівнянні з табличними та матричними даними.

Особливим чином організовані і збережені в електронному вигляді дані називають *базою даних* (англ. Database). Під «особливим чином» мається на увазі такий спосіб, що дає можливість полегшити їх пошук і доступ до них для одного або декількох застосунків. Така організація передбачає наявність мінімальної надмірності даних.

Бази даних є одним з різновидів інформаційних технологій, а також формою зберігання даних. Метою створення баз даних є побудова такої системи, що була б незалежною від програмного забезпечення та застосовуваних технічних засобів. Побудова такої системи має забезпечувати несуперечливу і цілісну інформацію. При проектуванні бази даних передбачається багатоцільове її використання. У найпростішому випадку база даних представляються у вигляді системи двовимірних таблиць.

1.1.1. Набір даних і їх атрибутів

Найпростіший і досить популярний формат зберігання даних — *таблиці*. Вони широко використовуються при дослідженні та аналізі даних.

Таблиця (англ. Table) — це набір елементів даних (*значень*), які організовані з використанням моделі вертикальних стовпчиків (з різними іменами) і горизонтальних рядків (див. табл. 1.1.1.1). По горизонталі таблиці розташовуються *атрибути* об'єкта (його ознаки). По вертикалі таблиці — власне *об'єкти*. Об'єкт описується як набір атрибутів. Об'єкт також відомий як запис, випадок, приклад, рядок таблиці.

Атрибут (англ. Attribute) — це властивість, що характеризує об'єкт (наприклад: колір очей людини, температура води тощо). Атрибут також називають полем таблиці, вимірюванням, характеристикою.

Змінна (англ. Variable) — властивість або характеристика, загальна для всіх досліджуваних об'єктів, прояв якої може змінюватися від об'єкта до об'єкта. *Значення* (англ. Value) змінної є проявом ознаки.

Аналіз даних, як правило, не передбачає можливості розглянути всю цікаву сукупність об'єктів. Обробка та вивчення великих обсягів даних є дорогим процесом, який вимагає значних затрат часу, а також неминуче призводить до помилок, пов'язаних з людським фактором. Для більшості задач буває цілком достатньо розглянути деяку частину всієї сукупності — *вибірку*. Розмір вибірки при цьому залежить від різноманітності об'єктів, представлених в *генеральної сукупності* (англ. Population) — всій сукупності досліджуваних об'єктів.

Таким чином *вибірка* (англ. Sample) — це частина генеральної сукупності, певним способом відібрана з метою дослідження і отримання висновків про властивості та

характеристики генеральної сукупності. У вибірці повинні бути представлені різні комбінації і елементи генеральної сукупності.

Числові характеристики генеральної сукупності називають *параметрами*, а числові характеристики вибірки — *статистиками*.

Досить часто дослідження ґрунтуються на *гіпотезах*, які перевіряються за допомогою даних. *Гіпотеза* — це припущення щодо параметрів сукупності об'єктів, яке повинно бути перевірено на її частині. Гіпотеза є частково обґрунтованою закономірністю знань, що служить або для зв'язку між різними емпіричними фактами, або для пояснення факту групи фактів.

1.1.2. Вимірювання

У процесі підготовки даних вимірюється характеристики об'єкту. *Вимірювання* — процес присвоєння чисел характеристикам досліджуваних об'єктів згідно з визначеним правилом — *шкалою*.

Зазвичай виділяють п'ять типів шкал: *номінальна, порядкова, інтервальна, відносна* і *дихотомічна*.

Номінальна шкала (англ. Nominal Scale) містить тільки категорії; дані в ній не можуть бути впорядковані, над ними неможливо проводити арифметичні дії. Номінальна шкала складається з назв, категорій, імен для класифікації і сортування об'єктів або спостережень за певною ознакою. Для цієї шкали можна застосовувати лише прості операції порівняння: дорівнює (=) та не дорівнює (≠). Прикладами номінальної шкали є: назви факультетів, прізвища тощо.

Порядкова шкала (англ. Ordinal Scale) характеризується тим, що в ній числа привласнюють об'єктам для позначення відносної позиції об'єктів, але не величини відмінностей між ними, тобто вимірювання в порядковій шкалі містять інформацію тільки про порядок величин, але не відповідає на питання типу "наскільки одна величина більша за іншу". Для цієї шкали можна застосовувати лише такі операції, як: дорівнює (=), не дорівнює (≠), більше (>) та менше (<). Приклади порядкової шкали: освіта (початкова, середня, вища), вчене звання (доцент, старшій дослідник, професор) тощо.

В *інтервальній шкалі* (англ. Interval Scale) різниці між значеннями можуть бути обчислені, проте їхні відношення не мають сенсу. Ця шкала дозволяє знаходити різницю між двома величинами, має властивості номінальної і порядкової шкал, а

також дозволяє визначити кількісну зміну ознаки. Інтервальна шкала, на відміну від дискретних номінальної і порядкової, є неперервною. Вона дозволяє здійснювати точні вимірювання ознаки і виконувати арифметичні операції додавання, віднімання, множення, ділення. Для цієї шкали можна застосовувати лише такі операції, як дорівнює (=), не дорівнює (≠), більше (>), менше (<), операції додавання (+) і віднімання (-). Прикладом такої шкали є температура повітря: можна говорити, що -10°C на 20 градусів менше, ніж +20°C, проте порівняння у скільки разів менше не має сенсу.

Відносна шкала (англ. Ratio Scale) характеризується існуванням певної точки відліку, що наповнює сенсом відношення між значеннями. Для цієї шкали можна застосовувати такі операції, як дорівнює (=), не дорівнює (≠), більше (>), менше (<), операції додавання (+) і віднімання (-), множення (*) та ділення (/). Прикладом такої шкали є заробітна плата.

Дихотомічна шкала (англ. Dichotomous Scale) містить тільки дві категорії. Приклад такої шкали: стать (чоловіча і жіноча). Це частковий випадок категоріальної шкали.

Приклад використання різних шкал для вимірювання властивостей різних об'єктів, в даному випадку температурних умов, наведено в таблиці 1.1.2.1.

1.2. Особливості обробки даних

Зазвичай застосування Data Mining передбачає аналіз великих об'ємів даних. У загальному випадку це не так, проте немає сумнівів, що необхідність пошуку закономірностей у великих базах даних суттєво ускладнює задачу.

Не існує універсальних методів аналізу або алгоритмів, що придатні для обробки будь-яких обсягів інформації. Методи аналізу даних істотно відрізняються один від одного по продуктивності, якості результатів, зручності застосування і вимогам до даних. Оптимізація може проводитися на різних рівнях: обладнання, бази даних, аналітична платформа, підготовка вихідних даних, спеціалізовані алгоритми. Аналіз великого обсягу даних вимагає особливого підходу, тому що технічно складно їх переробити за допомогою тільки «грубої сили», тобто використання більш потужного устаткування.

Таблица 1.1.1.1 Приклад двовимірної таблиці "об'єкт-атрибут"

	Атрибути				
Обєкти	*Код клієнта*	*Вік*	*Сімейний стан*	*Дохід*	*Клас*
	1	18	Single	125	1
	2	22	Married	100	1
	3	30	Single	70	1
	4	32	Married	120	1
	5	24	Divorced	95	2
	6	25	Married	60	1
	7	32	Divorced	220	1
	8	19	Single	85	2
	9	22	Married	75	1
	10	40	Single	90	2

Таблица 1.1.2.1. Множина вимірювань властивостей різних об'єктів

Номер об'єкта	*Професія (номінальна шкала)*	*Середній бал (інтервальна шкала)*	*Освіта (порядкова шкала)*
1	Слюсар	22	Середня
2	Вчений	55	Вища
3	Вчитель	47	Вища

Однією із ефективних стратегій обробки великих обсягів даних є розбиття даних на сегменти і побудову моделей для кожного сегмента окремо з подальшим об'єднанням результатів. Подібний підхід дозволяє підвищити швидкість аналізу і знизити вимоги до пам'яті завдяки обробці менших обсягів даних в один прохід. Крім того, в цьому випадку аналітичну обробку можна розпаралелити, що позитивно позначається на витраченому часі.

Крім підвищення швидкості цей підхід має і ще одну важливу перевагу — кілька відносно простих моделей окремо легше створювати і підтримувати, ніж одну велику. Можна запускати моделі поетапно, отримуючи таким чином перші результати в максимально стислі терміни[7[.

1.2.1. Модель MapReduce

MapReduce — програмна модель, призначена для паралельної обробки великих об’ємів даних за рахунок поділу задачі на незалежні частини[14]. Основна ідея підходу — розділити код на:

- обчислення і обробку даних;
- масштабування, розпаралелення і обробку відмов.

Термін MapReduce означав спочатку власну технологію Google, але зараз став загальновживаним і використовується для означення моделі програмування. Бібліотеки MapReduce були створені для різних мов програмування. Однією із найпопулярніших вільних імплементацій є Apache Hadoop[14].

Програма MapReduce складається із функції map та reduce. Функція map ставить у відповідність списку інший список. При цьому сам список не міняється, а створюється новий (див. рис. 1.2.1.1). Функція Reduce ставить у відповідність списку одне значення, наприклад, функція додавання (див. рис. 1.2.1.2). Сам список також не міняється.

В MapReduce до вхідного списку послідовно застосовуються функції MapReduce. Списки складаються із пар ключ-значення. Обробка елементів списку з різними ключами функціями Reduce виконується окремо. Для кожного ключа генерується окреме результуюче значення.

Розглянемо застосування технології MapReduce на прикладі реалізація функції підрахунку кількості слів у списку.

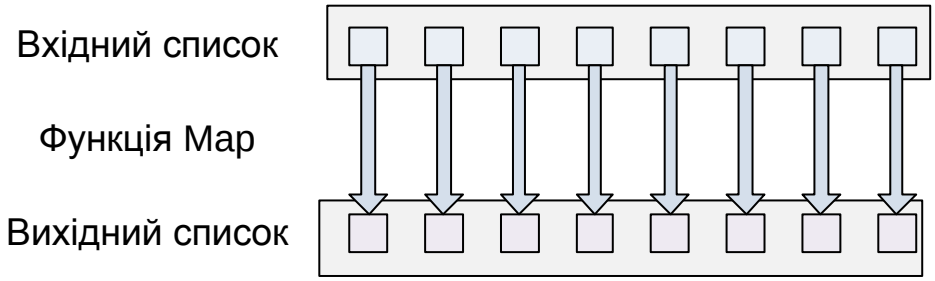


Рисунок 1.2.1.1. Функція Map

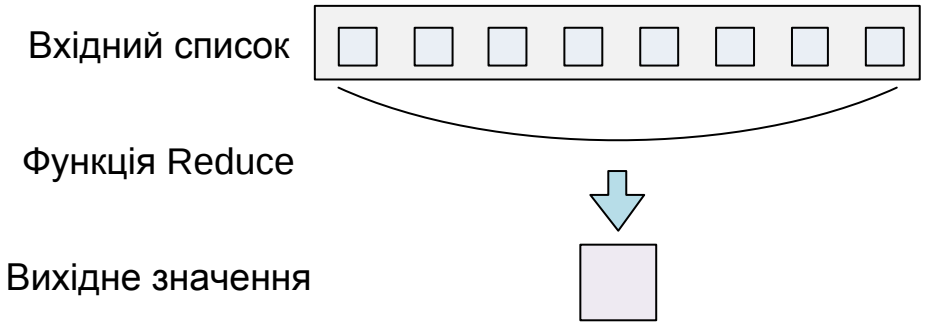


Рисунок 1.2.1.2. Функція Reduce

Маємо на вході:

```
file1: Hello World Bye World
file2: Hello Hadoop Goodbye Hadoop
```

На виході маємо отримати:

```
Bye  1
Goodbye  1
Hadoop   2
Hello    2
World    2
```

Функція *map* має генерувати списки пар: (ключ, значення), що виглядає наступним чином:

```
map (filename, file-contents):
    for each word in file-contents:
        emit (word, 1)
```

Функція *reduce* має «збирати» дані за ключем:

```
reduce (word, values):
          sum = 0
          for each value in values:
               sum = sum + value
          emit (word, sum)
```

Потік даних в MapReduce наступний (див. рис. 1.2.1.3):

- вхідні файли завантажуються в HDFS і розподіляються по вузлам;
- на кожному вузлі запускаються вхідні файли (будь-який процес map може опрацьовувати будь-який файл);
- процеси map генерують проміжні списки пар ключ-значення;
- пари ключ-значення передаються по мережі для обробки Reduce;
- усі значення з однаковим ключем передаються одному процесу Reduce;
- вихідні дані у вигляді файлів записуються в HDFS.

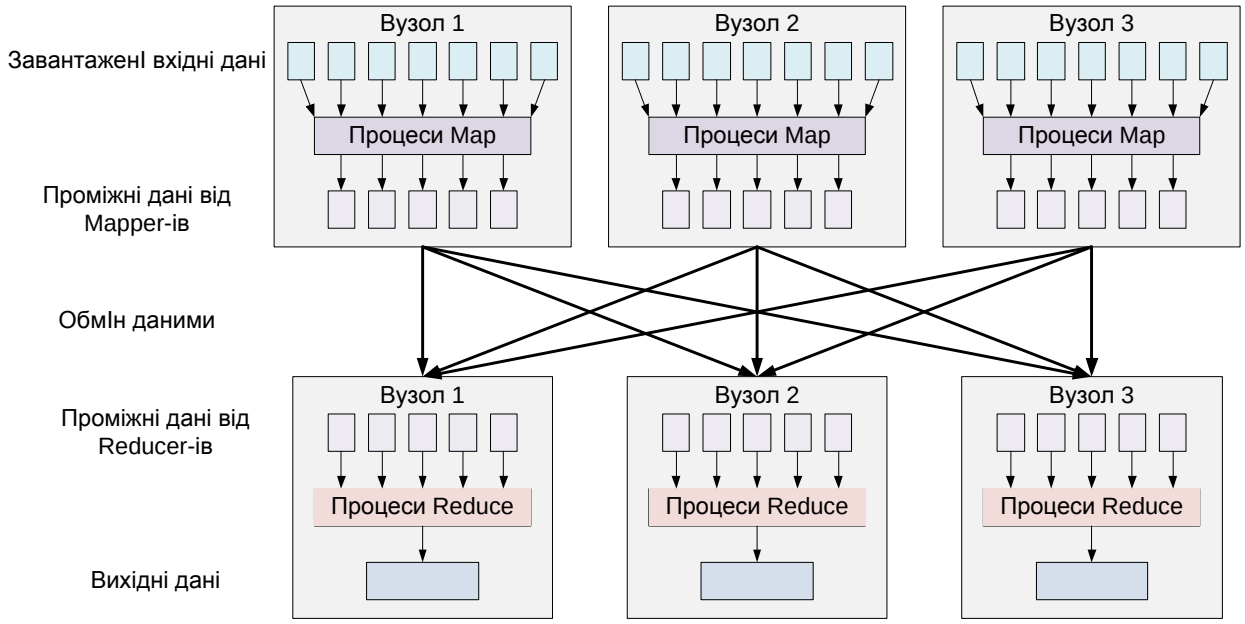


Рисунок 1.2.1.3. Потік даних в MapReduce

Управління потоком даних в MapReduce виконується автоматично, програміст не може змінювати потік даних. Окремі компоненти задачі не обмінюються між собою даними (інакше неможливе автоматичне відновлення після збою). У випадку збою вузла процеси Map і Reduce автоматично перезапускаються на новому вузлі.

1.3. Задачі Data Mining

Розглянемо основні *задачі,* які вирішуються методами Data Mining:

- *класифікація* — віднесення об'єктів (спостережень, подій) до одного з заздалегідь відомих *класів*;
- *регресія* (в тому числі *задачі прогнозування*) — встановлення *залежності* безперервних вихідних від вхідних змінних;
- *кластеризація* — угруповання об'єктів (спостережень, подій) на основі даних (властивостей), що описують сутність цих об'єктів, у *кластери*. Об'єкти всередині кластера повинні бути «схожими» один на одного і відрізнятися від об'єктів, що ввійшли в інші кластери. Чим більше схожі об'єкти всередині кластера і чим більше відмінностей між кластерами, тим точніша кластеризація;
- *асоціація* — виявлення закономірностей між пов'язаними подіями. Прикладом такої закономірності служить правило, яке вказує, що з події *X* слід подія *Y*. Такі правила називаються *асоціативними*. Вперше ця задача була запропонована для знаходження типових шаблонів покупок, що здійснюються в супермаркетах, тому іноді її ще називають *аналізом ринкової корзини* (market basket analysis);
- *послідовні шаблони* — встановлення закономірностей між пов'язаними в часі подіями, тобто виявлення залежності, що якщо відбудеться подія *X*, то через заданий час відбудеться подія *Y*;
- *аналіз відхилень* — виявлення найбільш нехарактерних шаблонів.

Проблеми бізнес-аналізу формулюються по-іншому, але рішення більшості з них зводиться до тієї чи іншої задачі Data Mining або до їх комбінації. Наприклад, оцінка ризиків — це вирішення завдання регресії або класифікації, сегментація

ринку — кластеризація, стимулювання попиту — асоціативні правила. Фактично, задачі Data Mining є елементами, з яких можна зібрати рішення переважної більшості реальних бізнес завдань.

Розглянемо застосування Data Mining на прикладі аналізу даних про час між виверженнями і тривалістю виверження одного із найвідоміших гейзерів — Олд Фейтфула (англ. Old Faithful) із Єлтонського національного парку у штаті Вайомінг що у США. Розглянемо дані, подані схематично на рисунку 1.3.1.

Приклад 1.3.1. На рисунку 1.3.1 ми бачимо, що точки групуються у два кластери. В одному кластері знаходяться точки, що відповідають виверженню з малою тривалістю і малим часом очікування. В іншому — з великою тривалістю і великим часом очікування.

На рисунку 1.3.2 виділені кластери та центри мас. Для знаходження центру ваги необхідно знайти середнє значення кожної ознаки.

Розглянемо метод центрів ваг для вирішення задачі кластеризації. Нехай множину об'єктів потрібно розбити на K кластерів. Спочатку довільним чином розіб'ємо об'єкти на K класів. У кожній групі знайдемо центр ваги і проведемо перегрупування. Для цього помістимо кожен об'єкт в той клас, до центру тяжіння якого він ближче всіх. Після перегрупування всіх об'єктів знову обчислимо центри тяжкості і повторимо ітерацію. Ітерації повторюються до тих пір, поки розбиття на класи перестане змінюватися.

До описаного методу близький метод медіан. Основна відмінність методу в тому, що замість центрів тяжіння використовуються медіани. Медіаною заданого набору точок називається та з них, яка ближче всіх до центру тяжіння.

1.4. Етапи Data Mining

Етапи аналізу розглянемо на прикладі стандарту CRISP-Data Mining (англ. Cross Industry Standard Process for Data Mining, див. рис. 1.4.1). За цим стандартом обробка та аналіз даніх відбувається поетапно. Розглянемо детально кожен етап.

Спочатку, на *етапі розуміння даних* (англ. Data Understanding) виявляємо і задаємо ознаки об'єктів. Позначимо D — *множину об'єктів* (англ. Data Set), $d \in D$ — *навчаючий об'єкт*, $\phi_j : D \to F_j$ — *ознаки*.

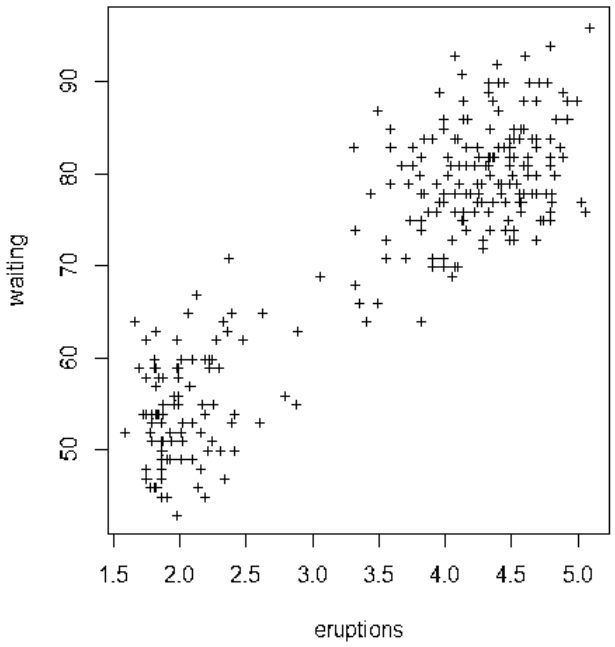


Рисунок 1.3.1. Діаграма, що представляє дані про тривалість виверження і проміжки між виверженнями гейзера

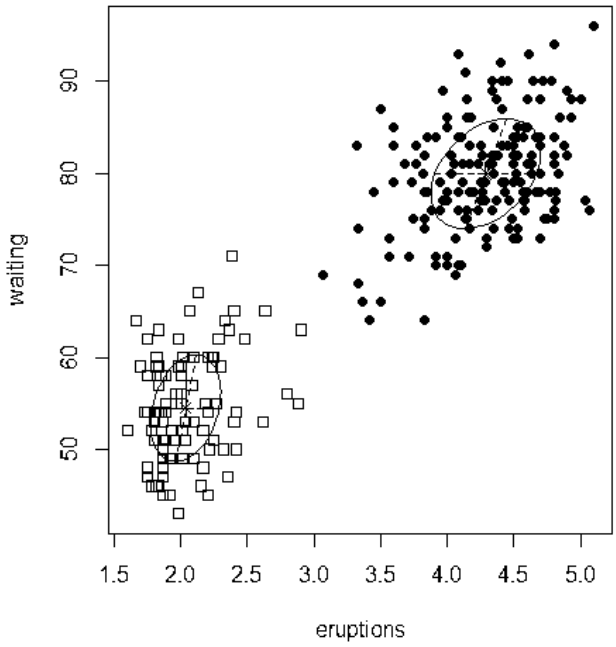


Рисунок 1.3.2. Дані розбиті на два класи. Жирними фігурами відзначені центри ваги кластерів

Ознаки можуть бути таких видів:

- *бінарні* (англ. Binary): $F_j = \{true, false\}$;
- *номінальні* (англ. Categorical): F_j — скінченне;
- *порядкові* (англ. Ordinal): F_j — скінченне, впорядковане;
- *кількісні* (англ. Numerical): $F_j = \mathbb{R}$

Тоді d_i представляється наступним чином:

$$\overline{x}_i = \{\phi_1(d_i), \ldots, \phi_n(d_i)) \in X$$

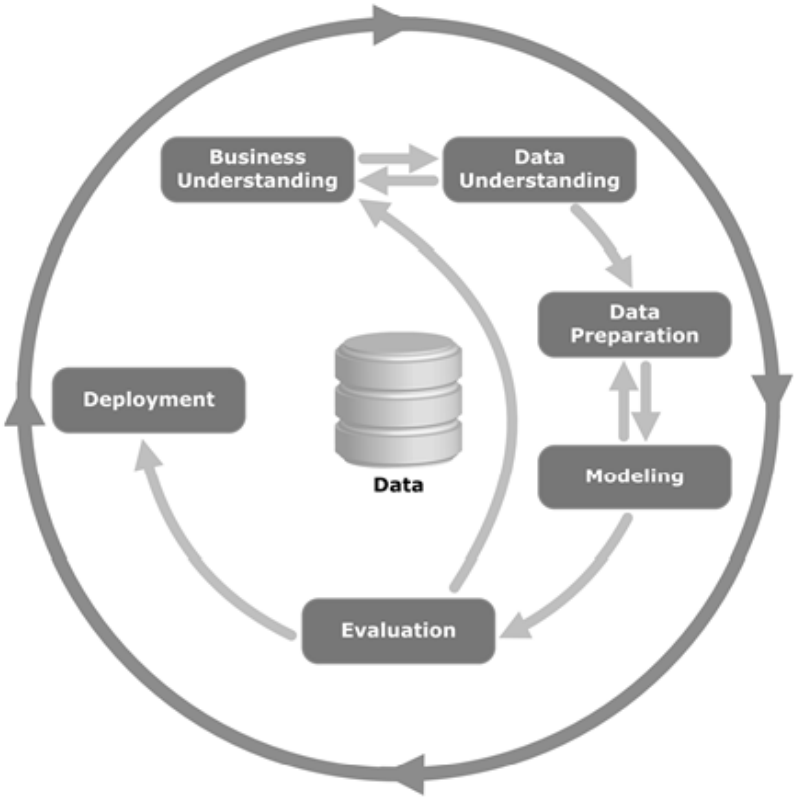


Рисунок 1.4.1. Етапи Cross Industry Standard Process for Data Mining

На *етапі підготовки даних* (англ. Data Preparating) відбувається наступне: видалення шуму, заповнення відсутніх значень (для коректної роботи алгоритму), трансформація факторів (ознак), вибір факторів (деякі ознаки можуть однозначно впливати на роботу алгоритму), використання апріорних знань. У результаті:

$$X = \begin{pmatrix} \overline{x}_1 \\ \dots \\ \overline{x}_N \end{pmatrix}, x_i \in \mathcal{X}, \quad Y = \begin{pmatrix} y_1 \\ \dots \\ y_N \end{pmatrix}, y_i \in \mathcal{Y}$$

Етап *модулювання* (аггл. Modeling) передбачає побудову моделі. Модель — сімейство параметричних функцій вигляду:

$$H = \{h(\overline{x}, \theta) : \mathcal{X} \times \Theta \to \mathcal{Y}\}$$

Алгоритм навчання — вибір найкращих параметрів θ^*:

$$A(X, Y) : (\mathcal{X} \times \mathcal{Y})^N \to \Theta$$

У результаті:

$$h^*(\overline{x}) = h(x, \theta^*)$$

На етапі *перевірки якості моделі* (англ. Evaluation) узгоджуємо, на що робити акцент. Є кілька важливих питань, на які потрібно дати відповідь з огляду на поставлену задачу та вимоги до неї.

По перше, потрібно зрозуміти, що для даної конкретної задачі важливіше: *точність* чи *апроксимація*? Більшість моделей важко навчити точно, або це складно в плані обчислення.

По друге, треба визначити, що має більше значення: *міра помилки* (англ. Bias) чи *міра стабільності (англ. Variance)?* Вважається, що якщо модель стабільна, але іноді помиляється — це не критично для більшості задач. Проте якщо модель не помиляється, але працює не стабільно — це не завжди добре.

По третє, на що варто зосередити увагу, на *інтерпретованість* чи на *якість?* Іноді алгоритм справляється з задачею якісно, але не зрозуміло, що відбувається. Проте інколи прозорий алгоритм дає не точний результат.

Останній етап — власне реалізація проекту (англ. *Deployment*).

Важливо, що в основі методів лежать деякі припущення. Якщо припущення хибні — то і результат незадовільний. Тому, перед розробкою алгоритмів, важливо ретельно дослідити властивості даних.

Розвідувальний аналіз, насправді, може займати *до половини часу*, виділеного на аналіз.

1.5. Методи Data Mining

Основна особливість Data Mining — це поєднання широкого математичного інструментарію (від класичного статистичного аналізу до нових кібернетичних методів) і останніх досягнень у сфері інформаційних технологій. У технології Data Mining гармонійно поєдналися строго формалізовані методи та методи неформального аналізу, тобто кількісний та якісний аналіз даних.

1.4.1. Класифікація методів Data Mining

До методів[5] і алгоритмів інтелектуального аналізу даних відносяться наступні: штучні нейронні мережі, дерева рішень, символьні правила, методи найближчого сусіда і k-найближчого сусіда, метод опорних векторів, байєсовські мережі, лінійна регресія, кореляційно-регресійний аналіз; ієрархічні методи кластерного аналізу, неієрархічні методи кластерного аналізу, в тому числі алгоритми k-середніх і k-медіани; методи пошуку асоціативних правил, у тому числі алгоритм Apriori; метод обмеженого перебору, еволюційне програмування і генетичні алгоритми, різноманітні методи візуалізації даних тощо.

Більшість аналітичних методів, що використовуються в технології Data Mining — це відомі математичні алгоритми. Новим в їх застосуванні є можливість їх використання при вирішенні тих чи інших конкретних проблем, обумовлена можливостями, які з'явилися завдяки розвитку технічних і програмних засобів. Слід зазначити, що більшість методів інтелектуального аналізу даних були розроблені в рамках теорії штучного інтелекту.

Розділимо методи інтелектуального аналізу даних на *статистичні* на *кібернетичні*. Під статистичними методами маємо на увазі наступні:

- *попередній аналіз природи статистичних даних* (перевірка гіпотез стаціонарності, нормальності, незалежності, однорідності, оцінка виду функції розподілу, її параметрів і т.п.);
- *виявлення зв'язків і закономірностей* (лінійний і нелінійний регресійний аналіз, кореляційний аналіз та ін.);

[5] Метод (англ. Method) являє собою норму або правило, певний шлях, спосіб, прийом рішень задачі теоретичного, практичного, пізнавального, управлінського характеру.

- *багатовимірний статистичний аналіз* (лінійний і нелінійний дискримінантний аналіз, кластерний аналіз, компонентний аналіз, факторний аналіз та ін.);
- *динамічні моделі і прогноз на основі часових рядів.*

Другий напрямок — це множина підходів, об'єднаних ідеєю *комп'ютерної математики* та використання теорії *штучного інтелекту*. До цієї групи відносимо такі методи:

- *штучні нейронні мережі* (розпізнавання, кластеризація, прогноз);
- *еволюційне програмування* (в т.ч. алгоритми методу групового обліку аргументів);
- *генетичні алгоритми* (оптимізація);
- *асоціативна пам'ять* (пошук аналогів, прототипів);
- *нечітка логіка*;
- *дерева рішень*;
- *системи обробки експертних знань*.

Методи Data Mining також можна класифікувати за задачами. Відповідно до такої класифікації виділяємо дві групи. Перша з них — це підрозділ методів на *вирішення задач сегментації* (тобто задачі класифікації і кластеризації) і задачі *прогнозування*. У відповідності з другою класифікацією, методи Data Mining можуть бути спрямовані на отримання *описових* і *прогнозуючих* результатів. Описові методи служать для знаходження *шаблонів* (або *зразків)*, що описують дані, які піддаються інтерпретації з точки зору аналітика. До методів, спрямованим на отримання описових результатів, відносяться ітеративні методи кластерного аналізу, в тому числі: алгоритм k-середніх, k-медіани, ієрархічні методи кластерного аналізу, самоорганізаційні карти Кохонена, методи крос-табличної візуалізації, різні методи візуалізації тощо.

Прогнозуючі методи використовують значення одних змінних для передбачення/прогнозування невідомих (пропущених) або майбутніх значень інших (цільових) змінних. До методів, спрямованим на отримання прогнозуючих результатів, відносяться такі методи: нейронні мережі, дерева рішень, лінійна регресія, метод найближчого сусіда, метод опорних векторів та ін.

1.4.2. Властивості методів Data Mining

Різні методи Data Mining характеризуються певними властивостями, які можуть бути визначальними при виборі

методу аналізу даних. Методи можна порівнювати між собою, оцінюючи характеристики їх властивостей. Серед основних властивостей і характеристик методів Data Mining розглянемо такі: *точність, масштабованість, інтерпретованість, перевіряємість, трудомісткість, гнучкість, швидкість і популярність.*

Масштабованість — властивість обчислювальної системи, що забезпечує передбачуваний ріст системних характеристик, наприклад, швидкості реакції, загальної продуктивності тощо, при додаванні до неї обчислювальних ресурсів.

Більшість інструментів Data Mining, пропонованих зараз на ринку програмного забезпечення, реалізують одразу кілька методів, наприклад: дерева рішень, індукцію правил і візуалізацію, або ж нейронні мережі, карти Кохонена та візуалізацію.

В універсальних прикладних статистичних пакетах (наприклад, SPSS, SAS, STATGRAPHICS, Statistica тощо) реалізується широкий спектр найрізноманітніших методів (як статистичних, так і кібернетичних). Слід враховувати, що для можливості їх використання, а також для інтерпретації результатів роботи статистичних методів (кореляційного, регресійного, факторного, дисперсійного аналізу тощо) потрібні спеціальні знання в галузі статистики.

Універсальність того чи іншого інструменту часто накладає певні обмеження на його можливості. Перевагою використання таких універсальних пакетів є можливість відносно легко порівнювати результати побудованих моделей, отримані різними методами. Така можливість реалізована, наприклад, в пакеті Statistica, де порівняння засноване на так званій «конкурентній оцінці моделей». Ця оцінка полягає в застосуванні різних моделей до одного і того ж набору даних для порівняння їх характеристик та вибору найкращої з них.

Висновки до розділу 1

Data Mining — це побудова моделі даних.

Дані — це необроблений матеріал, що надається постачальниками даних і використовується споживачами для формування інформації на основі даних. *Табличні дані* складаються із записів, кожен з яких складається з фіксованого набору атрибутів. *Транзакційні дані* — особливий тип даних, де кожен

запис, що є транзакцією, включає набір значень. *Графічні дані*: WWW-дані, молекулярні структури, графи, карти тощо.

Вимірювання — процес присвоєння чисел характеристикам досліджуваних об'єктів згідно з визначеним правилом — *шкалою*. Існує п'ять типів шкал: *номінальна, порядкова, інтервальна, відносна* і *дихотомічна*.

MapReduce — програмна модель, призначена для паралельної обробки великих об’ємів даних за рахунок поділу задачі на незалежні частини.

Основні *задачі,* які вирішуються методами Data Mining: *класифікація, регресія, кластеризація, асоціація, послідовні шаблони, аналіз відхилень.*

Статистичні методи Data Mining: попередній аналіз природи статистичних даних, виявлення зв'язків і закономірностей, багатовимірний статистичний аналіз, динамічні моделі і прогноз на основі часових рядів.

Кібернетичні методи Data Mining: штучні нейронні мережі, еволюційне програмування, генетичні алгоритми, асоціативна пам'ять, нечітка логіка, дерева рішень, системи обробки експертних знань.

Основні властивості методів Data Mining: точність, масштабованість, інтерпретованість, перевіряємість, трудомісткість, гнучкість, швидкість і популярність

Питання до розділу 1

1. Дайте визначення поняттю Data Mining?
2. Хто і в якому смислі вперше використав термін Data Mining?
3. Дайте визначення поняттю даних. Які типи даних ви знаєте?
4. Що собою являє об’єкт та атрибут в табличних даних?
5. Які типи шкал ви знаєте? В чому особливість кожної з них?
6. Що таке MapReduce?
7. Які задачі, які вирішуються методами Data Mining?
8. Що собою являє Cross Industry Standard Process for Data Mining? Які його основні етапи?

9. Які методи застосовуються в Data Mining? Чим відрізняються статистичні та кібернетичні методи Data Mining?
10. Що собою являє попередній аналіз даних?
11. Які властивості методів Data Mining ви знаєте?
12. Що таке масштабованість системи?

Розділ 2. Задача класифікації даних

2.1. Постановка задачі класифікації даних

Класифікація — це задача розбиття множини об'єктів або спостережень на апріорно задані групи, звані *класи*, всередині кожної з яких вони передбачаються схожими один на одного, мають приблизно однакові властивості і ознаки. При цьому рішення отримується на основі аналізу значень *атрибутів*.

Класифікація є однією з найважливіших задач інтелектуального аналізу даних. Вона застосовується в маркетингу при оцінці кредитоспроможності позичальників, визначенні лояльності клієнтів, розпізнаванні образів, медичній діагностиці та багатьох інших додатках. Якщо аналітику відомі властивості об'єктів кожного класу, то якщо нове спостереження належить до певного класу, дані властивості автоматично поширюються і на нього.

Якщо число класів обмежене двома, то має місце *бінарна класифікація*, до якої можуть бути зведені складніші задачі. Наприклад, замість визначення таких ступенів кредитного ризику, як «Високий», «Середній» або «Низький», можна використовувати всього два — «Видати» або «Відмовити».

При вирішенні задач класифікації необхідно віднести наявні *статичні зразки* (наприклад, характеристики ситуації на ринку, дані медогляду, інформацію про клієнта) до певних класів. Допустимі кілька способів подання даних. Найбільш поширеним є такий, при якому зразок представляється вектором. Компоненти цього вектора являють собою різні характеристики зразка, які впливають на прийняття рішення про те, до якого класу можна віднести даний зразок. Наприклад, для медичних задач компонентами цього вектора можуть бути дані з медичної карти хворого. Таким чином, на підставі деякої інформації про зразок, необхідно визначити, до якого класу його можна віднести. Класифікатор таким чином відносить об'єкт до одного з класів відповідно до визначеного розбиттям N-мірного простору, який називається *простором входів*, і розмірність цього простору визначається кількістю компонент вектора.

Для розв'язання заданої задачі насамперед, потрібно визначити рівень складності системи, що є непростою задачею в ситуації, коли кількість зразків значно обмежена. Можна виділити три основні рівні складності. Перший (найпростіший) — коли класи можна розділити прямими лініями (або гіпер-

площинами, якщо простір входів має розмірність більше двох) — *лінійна роздільність*. У другому випадку класи неможливо розділити лініями (площинами), але їх можливо відокремити за допомогою більш складного поділу — *нелінійна роздільність*. У третьому випадку класи перетинаються і можна говорити лише про *ймовірнісну роздільність.*

В ідеальному варіанті після попередньої обробки ми маємо отримати лінійно роздільну задачу, оскільки це значно спрощує побудову класифікатора. На жаль, при вирішенні реальних задач ми маємо обмежену кількість зразків, на основі яких проводиться побудова класифікатора. При цьому ми не можемо провести таку передобробку даних, при якій буде досягнута лінійна роздільність зразків.

Розрізняють такі задачі класифікації:

- *допоміжну (штучну) класифікацію*, яка здійснюється за зовнішньою ознакою і служить для надання множині предметів (процесів, явищ) потрібного порядку;
- *природну класифікацію*, яка здійснюється за істотними ознаками, що характеризують внутрішню спільність предметів і явищ; вона є результатом і важливим засобом наукового дослідження, оскільки передбачає і закріплює результати вивчення закономірностей об'єктів, що класифікуються.

В залежності від обраних ознак, їх поєднання і процедури розподілу понять, класифікація може бути:

- *простою*, за якої поділ родового поняття відбувається тільки за ознакою і тільки один раз до розкриття всіх видів;
- *складною*, що застосовується для поділу одного поняття за різними ознаками і синтезу таких простих поділів в єдине ціле.

Прикладом простої класифікації є *дихотомія*, при якій членами поділу можуть бути два поняття, кожне з яких виключає інше, тобто дотримується *принцип виключення третього*: $A \& \neg A$ = False. Прикладом складної класифікації є періодична система хімічних елементів.

Таким чином, для проведення класифікації повинні бути присутніми ознаки, що характеризують групу, до якої належить та чи інша подія або об'єкт. Зазвичай при цьому на підставі

аналізу вже класифікованих подій формулюються певні правила.

Класифікація відноситься до *стратегії навчання з учителем* (англ. Supervised Learning), яке також називають контрольованим або керованим навчанням.

Завданням класифікації часто називають пророкування категоріальної залежної змінної (тобто залежною змінною, яка є категорією) на основі вибірки безперервних і/або категоріальних змінних. Наприклад, можна передбачити, хто з клієнтів фірми є потенційним покупцем певного товару, а хто — ні; хто скористається послугою фірми, а хто — ні тощо. Цей тип завдань відноситься до завдань бінарної класифікації, в них залежна змінна може приймати тільки два значення (наприклад: так чи ні; 0 або 1).

Інший варіант класифікації виникає, якщо залежна змінна може приймати значення з деякої множини визначених класів. Наприклад, коли необхідно передбачити, яку марку автомобіля захоче купити клієнт. У цих випадках розглядається множина класів для залежної змінної.

Класифікація може бути *одномірною* (за однією ознакою) і *багатовимірною* (за двома і більше ознаками). Багатовимірна класифікація була розроблена біологами для вирішення проблем дискримінації для класифікації організмів. Однією з перших робіт, присвячених цьому напрямку, вважають роботу Р. Фішера «Генетична теорія природного відбору» 1930-го року, в якій організми поділялися на підвиди залежно від результатів вимірювань їх фізичних параметрів.

Розглянемо задачу класифікації на простому прикладі.

Приклад 2.1.1. Припустимо, є база даних про клієнтів туристичного агентства з інформацією про вік і дохід за місяць. Є рекламний матеріал двох видів: більш дорогий і комфортний відпочинок і дешевший, молодіжний відпочинок. Відповідно, визначені два класу клієнтів: клас 1 і клас 2. База даних приведена в таблиці 2.1.1. Необхідно визначити, до якого класу належить новий клієнт і який з двох видів рекламних матеріалів йому варто надсилати.

Для наочності представимо нашу базу даних в двомірному просторі (вік і дохід) у вигляді множини об'єктів, що належать класам 1 і 2. На рисунку 2.1.1 наведені об'єкти з двох класів, виділені різними кольорами.

Задача полягає в тому, щоб визначити, до якого класу належить новий клієнт, що на рисунку 2.1.1 позначений білою міткою.

В інтелектуальному аналізі даних для розв'язання задачі класифікації використовуються такі моделі, як: нейронні мережі, дерева рішень, машини опорних векторів, метод k-найближчих сусідів, алгоритми покриття тощо, при побудові яких застосовується навчання з учителем, коли вихідна змінна (мітка класу) задана для кожного спостереження. Формально класифікація проводиться на основі розбиття простору ознак на області, в межах кожної з яких багатовимірні вектори розглядаються як ідентичні. Іншими словами, якщо об'єкт потрапив в область простору, асоційовану з певним класом, він до нього і відноситься (див. рис. 2.1.2).

2.2. Точність класифікації: оцінка рівня помилок

Оцінка точності класифікації може проводитися за допомогою *крос-перевірки*.

Крос-перевірка (Cross-validation) — це процедура оцінки точності класифікації на даних з тестової множини, яку також називають крос-перевірочною. Точність класифікації тестової множини порівнюється з точністю класифікації навчальної множини. Якщо класифікація тестової множини дає приблизно такі ж результати по точності, як і класифікація навчальної множини, вважається, що дана модель пройшла крос-перевірку.

Поділ на навчальну і тестову множини здійснюється шляхом ділення вибірки у певній пропорції, наприклад навчальна множина — дві третини даних і тестова — одна третина даних. Цей спосіб слід використовувати для вибірок з великою кількістю прикладів. Якщо ж вибірка має малі обсяги, рекомендується застосовувати спеціальні методи, при використанні яких навчальна і тестова вибірки можуть частково перетинатися.

Таблиця 2.1.1. База даних клієнтів туристичного агентства

Код клієнта	Вік	Прибуток	Клас
1	18	25	1
2	22	100	1
3	30	70	1
4	32	120	1
5	24	15	2
6	25	22	1
7	32	50	2
8	19	45	2
9	22	75	1
10	40	90	2

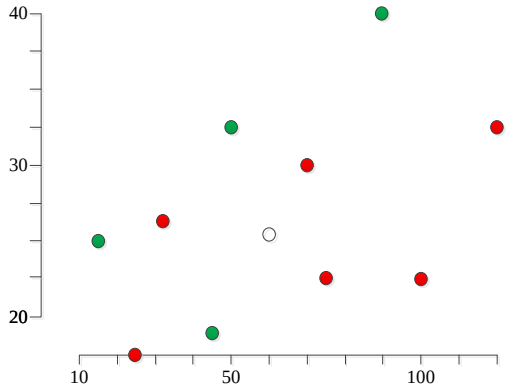
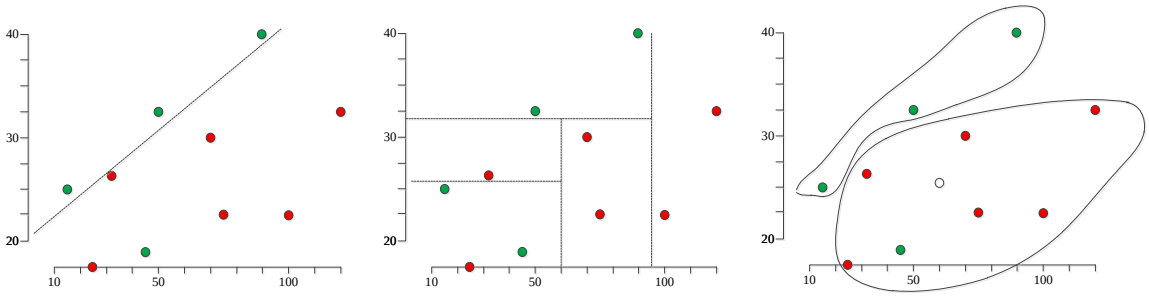


Рисунок 2.1.1. Множини об'єктів бази даних в двомірному вимірі



а) б) в)

Рис 2.1.2. Розв’язання задачі класифікації методами а) лінійної регресії б) дерев рішень в) нейронних мереж

Оцінювання методів слід проводити за такими характеристиками: *швидкість, робастність, інтерпретованість, надійність*.

Швидкість характеризує час, що необхідний на створення моделі та її використання.

Робастність, тобто стійкість до будь-яких порушень вихідних передумов, означає можливість роботи з зашумленими даними і пропущеними значеннями в даних.

Інтерпретованість забезпечує можливість розуміння моделі аналітиком.

Надійність методів класифікації передбачає можливість роботи цих методів при наявності в наборі даних шумів і викидів.

2.3. Алгоритм 1-rule

Алгоритм побудови елементарних правил (1R, OneR, 1-rule) — простий алгоритм побудови правил для класифікації об'єкта, цей алгоритм будує правила за значенням тільки однієї незалежної змінної.

Алгоритм по суті є пошуком змінної, яка дозволила б максимальною точністю класифікувати об'єкти. Для цього необхідно перевірити кожну змінну шляхом обчислення «помилки» — кількості об'єктів, що не задовольняють правилу з тих, що мають значення цієї змінної. У підсумку обирається змінна з найменшою помилкою.

Алгоритм побудови елементарних правил вважається найпростішим серед алгоритмів класифікації. Розглянемо його на прикладі.

Приклад 2.3.1. Для даних з таблиці 2.3.1 правила формуються з усіх можливих значень всіх незалежних змінних. Для кожного правила рахується помилка: скільки об'єктів не задовольняють правилу з тих, що мають значення цієї змінної. Правило з найменшою помилкою вважається найкращим. При класифікації розглядається лише найкраще правило.

Наприклад, для правила: «якщо назва = тематична, то стаття = релевантна» виконується наступне: 3 тематичних заголовка у не релевантних статей, 6 тематичних заголовків всього, отже помилка: ½.

Таблиця 2.3.1. Приклад даних

Назва	*Ключові слова*	*Тематика підходить*	*Стаття*
Тематична	<50%	Ні	Не релевантна
Тематична	>50%	Ні	Релевантна
Не тематична	>50%	Так	Релевантна
Тематична	>50%	Ні	Релевантна
Не тематична	<50%	Так	Не релевантна
Не тематична	>50%	Ні	Релевантна
Тематична	<50%	Так	Не релевантна
Не тематична	>50%	Ні	Не релевантна
Тематична	>50%	Так	Релевантна
Тематична	>50%	так	Не релевантна

Розглянемо правило «Якщо ключових слів>50%, то стаття = релевантна». Йому відповідає 5 із 7 варіантів, тобто точність – 69%, що достатньо ефективно.

2.4. Наївний баєсівський класифікатор

Наївний байєсівський класифікатор один з найпростіших з алгоритмів класифікації. Тим не менш, досить часто він працює краще за більш складні алгоритми.

Розглянемо множину відомих об'єктів:

$$D=\{d_q, d_2, \ldots, d_m\},$$

кожен з яких має деякий набір ознак з множини всіх ознак:

$$F=\{f_1, f_2, \ldots, f_q\},$$

а також одну мітку з множини міток:

$$C=\{c_1, c_2, \ldots, c_r\}.$$

Нашим завданням є обчислення найбільш ймовірного класу/мітки невідомого вхідного об'єкту d, опираючись на набір його ознак:

$$F_d = \{f_{d1}, f_{d2}, \ldots, f_{dn}\}.$$

Іншими словами, нам необхідно обчислити таке значення випадкової змінної *c*, при якому досягається апостеріорний максимум (англ. Maximum Aposteriori Probability, MAP).

Таким чином, модель наївного байєсівського класифікатора приймає два припущення:

- порядок проходження ознак об'єкта не має значення;
- ймовірності ознак не залежать одне від одного при даному класі:

Для прикладу візьмемо задачу визначення статі по імені. Звичайно, щоб визначити стать можна створити великий список імен з мітками статі. Але цей список в будь-якому випадку буде не вичерпний. Для того щоб вирішити цю проблему, можна «натренувати» модель по маркованих іменах.

Нехай є рядок тексту *d*. Крім того, є класи *C*, до одного з яких ми повинні віднести рядок. Нам необхідно знайти такий клас *c*, при якому його ймовірність для даного рядка була б максимальна. Математично це записується так:

$$c = \arg\max_{c \in C} P(c \mid d)$$

Обчислити $P(c|d)$ складно. Але можна скористатися теоремою Байеса і перейти до непрямих ймовірностям:

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

Оскільки ми шукаємо максимум від функції, то знаменник нас не цікавить (він в даному випадку константа). Крім того, треба поглянути на рядок *d*. Зазвичай, немає сенсу працювати з усім рядком. Набагато ефективніше виділити з нього певні ознаки (англ. features). Таким чином формула прийме вигляд:

$$P(c \mid f_1 f_2 .. f_n) = \frac{P(f_1 f_2 .. f_n \mid c)P(c)}{P(f_1 f_2 .. f_n)}$$

Тут включаємо «наївне» припущення про те, що змінні *F* залежать тільки від класу *C*, і не залежать один від одного. Це сильне спрощення, але найчастіше це працює. Чисельник прийме вигляд:

$$P(c)P(f_1 \mid c)P(f_2 \mid cf_1)..P(f_n \mid cf_1 f_2 .. f_n) = P(c)P(f_1 \mid c)P(f_2 \mid c)...P(f_n \mid c) =$$

$$= P(c)\prod_i P(f_i \mid c)$$

Фінальна формула прийме вигляд:

$$c = \arg\max_{c \in C} P(c \mid f_1 f_2 .. f_n) = \arg\max_{c \in C} P(c) \prod_i p(f_i \mid c)$$

Щоб під час програмування не оперувати з числами, близькими до нуля, використаємо логарифм: так як логарифм монотонно зростає для будь-якого $x>0$, то максимум будь-якої функції $f(x)$ буде ідентичний максимуму $ln(f(x))$:

$$c = \arg\max_{c \in C} P(c) \prod_i p(f_i \mid c) = \arg\max_{c \in C} \ln(P(c) \prod_i p(f_i \mid c)) = \arg\max_{c \in C} \ln(P(c)) + \ln(\prod_i p(f_i \mid c))$$

Тобто все що потрібно зробити, це обчислити ймовірності $P(c)$ і $P(d|c)$. Обчислення цих параметрів називається тренуванням класифікатора.

2.5. Застосування нейронних мереж для задач класифікації

Розв'язання задачі класифікації є одним з найважливіших застосувань нейронних мереж.

2.5.1. Штучні нейронні мережі

Штучні нейронні мережі — здатні до навчання системи, що імітують діяльність людського мозку[40].

Незважаючи на велику різноманітність варіантів нейронних мереж, всі вони мають спільні риси. Так, всі вони, так само, як і мозок людини, складаються з великого числа зв'язаних між собою однотипних елементів — нейронів, які імітують нейрони головного мозку. На рисунку 2.5.1.1 показана схема нейрону.

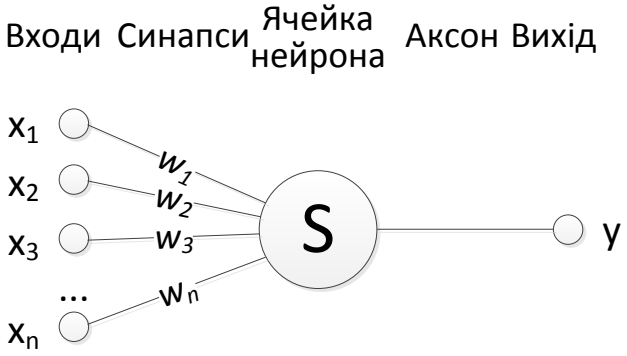


Рисунок 2.5.1.1 Схема штучного нейрону

З рисунку видно, що штучний нейрон, так само, як і живий, складається із синапсів, що пов'язують входи нейрону з ядром; ядра нейрона, яке здійснює обробку вхідних сигналів і аксона, що пов'язує нейрон з нейронами наступного шару. Кожен синапс має вагу, яка визначає, наскільки відповідний

вхід нейрону впливає на його стан. Стан нейрону визначається за формулою:

$$S = \sum_{i=1}^{n} x_i w_i ,$$

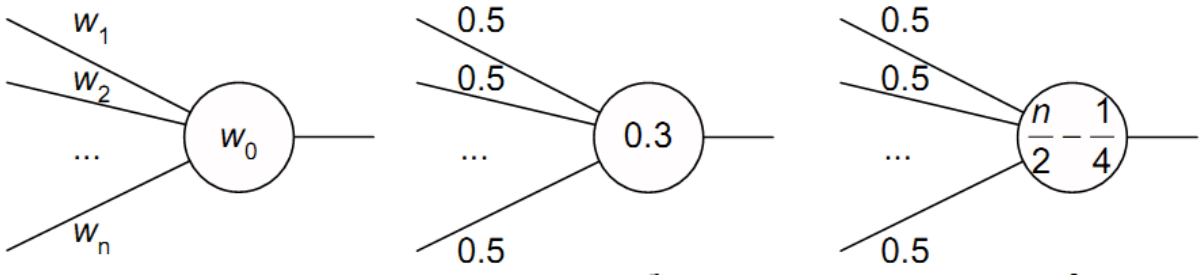
де n — число входів нейрона, x_i — значення і-го входу нейрона, w_i — вага і-го синапса.

Далі визначається значення аксона за формулою:

$$Y = f(S),$$

де f — деяка функція, що називається активаційною.

У випадку лінійної нейронної мережі, перцептрон підраховує S та порівнює його із заданим значенням — порогом активації (англ. threshold) видає 1, якщо $S > w_0$ (рисунок 2.5.1.2.а). Будь-яка булева функція може бути представлена у вигляді побудованної з перцептронів штучної мережі глибини 2. Приклад диз'юнкції та кон'юнкції подано на рисунку 2.5.1.2.б-в.



а) загальна схема б) диз'юнкція в) кон'юнкція

Рисунок 2.5.1.2 Приклади перцептронів

Найчастіше у якості активаційної функції використовується так звана сигмоїда, що має наступний вигляд:

$$f(x) = \frac{1}{1 + e^{-\alpha x}}$$

Основна перевага цієї функції в тому, що вона диференційована на всій області визначення і має дуже просту похідну:

$$f'(x) = \alpha f(x)(1 - f(X))$$

При зменшенні параметра сигмоїда стає більш пологою, вироджуючись в горизонтальну лінію на рівні 0,5 при a=0. При збільшенні сигмоїда все більше наближується до функції одиничного стрибка.

2.5.2. Нейронні мережі зворотного поширення

Нейронні мережі зворотного поширення — це найпотужніший інструмент пошуку закономірностей, прогнозування, якісного аналізу. Свою назву вони отримали за використання алгоритму навчання, в якому помилка поширюється від кінцевого шару до початкового, тобто в напрямку, протилежному напрямку поширення сигналу при нормальному функціонуванні мережі.

Нейронна мережа зворотного поширення складається з декількох шарів нейронів, причому кожен нейрон шару i пов'язаний з кожним нейроном шару $i+1$, тобто мова йде про повнозв'язну нейронну мережу.

У загальному випадку задача навчання нейронної мережі зводиться до знаходження функціональної залежності $Y = f(X)$, де X — вхідний, а Y — вихідний вектори. У загальному випадку така задача, при обмеженому наборі вхідних даних, має нескінченну множину розв'язків. Для обмеження простору пошуку при навчанні ставиться задача мінімізації цільової функції помилки нейронної мережі, яка знаходиться за методом найменших квадратів:

$$E(w) = \frac{1}{2}\sum_{j=1}^{p}(y_j - d_j)^2 ,$$

де y_j — значення j-го виходу нейромережі, d_j — цільове значення j-го виходу, p — число нейронів у вихідному шарі.

Навчання нейронної мережі здійснюється методом градієнтного спуску, тобто на кожній ітерації зміна ваги здійснюється за формулою:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}} ,$$

де η — параметр, що визначає швидкість навчання. При цьому:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial S_j} \cdot \frac{\partial S_j}{\partial w_{ij}} ,$$

де y_j — значення виходу j-го нейрона, S_j — середньозважена сума вхідних сигналів. При цьому:

$$\frac{\partial S_j}{\partial w_{ij}} \equiv x_i ,$$

де x_i — значення i-го входу нейрону.

Далі розглянемо множник $\frac{\partial E}{\partial y_j}$:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial S_k} \cdot \frac{\partial S_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial S_k} \cdot w_{jk}^{(n+1)},$$

де k — число нейронів в шарі n+1.

Введемо допоміжну змінну:

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j} = \frac{dy_j}{dS_j}$$

Тоді ми зможемо визначити рекурсивну формулу для визначення n-ого шару:

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \right] \frac{dy_j}{dS_j}.$$

Визначення останнього шару нейронної мережі не є складною задачею, оскільки нам відомий цільовий вектор, тобто вектор тих значень, які повинна видавати нейронна мережа при заданому наборі вхідних значень:

$$\delta_j^{(n)} = (y_i^{(N)} - d_i) \cdot \frac{dy_i}{dS_i}.$$

Нарешті, формула має вигляд:

$$\Delta w_{ij}^{(n)} = -\eta \cdot \delta_j^{(n)} \cdot x_i^n.$$

Розглянемо тепер повний алгоритм навчання нейронної мережі:

1. подати на вхід нейронної мережі один із потрібних образів і визначити значення виходів нейронів нейронної мережі;
2. розрахувати для вихідного шару за формулою:

$$\delta_j^{(n)} = (y_i^{(N)} - d_i) \cdot \frac{dy_i}{dS_i}$$

і розрахувати зміни ваг вихідного шару N за формулою:

$$\Delta w_{ij}^{(n)} = -\eta \cdot \delta_j^{(n)} \cdot x_i^n;$$

3. Розрахувати за формулами:

$$\delta_j^{(n)} = \left[\sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \right] \frac{dy_j}{dS_j} \text{ і } \Delta w_{ij}^{(n)} = -\eta \cdot \delta_j^{(n)} \cdot x_i^n$$

відповідно і $\Delta w_{ij}^{(N)}$ для інших шарів нейронної мережі, $n = N\text{-}1..1$;

4. Скорегувати всі ваги нейронної мережі за формулою:

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t);$$

5. Якщо помилка суттєва — перейти до кроку 1.

На етапі 2 до нейронної мережі по черзі у випадковому порядку подаються вектори з навчальної послідовності.

2.5.3. Подання вхідних даних для штучних нейронних мереж

Основна відмінність нейронних мереж від інших алгоритмів машинного навчання в тому, що для них усі вхідні і вихідні параметри представлені у вигляді чисел з плаваючою точкою (зазвичай в діапазоні [0..1]). Проте зазвичай дані предметної області часто мають інше кодування. Наприклад, це можуть бути числа в довільному діапазоні, дати, символьні рядки тощо. Таким чином, дані можуть бути як кількісними, так і якісними. В такому випадку дані потрібно перед обробити, представивши у відповідному форматі.

Для побудови класифікатора необхідно визначити, які параметри впливають на прийняття рішення про те, до якого класу належить зразок. При цьому можуть виникнути дві проблеми. По-перше, якщо кількість параметрів невелика, то може виникнути ситуація, при якій один і той же набір вихідних даних відповідає прикладам, які знаходяться в різних класах. Тоді нейронну мережу навчити неможливо, оскільки неможливо знайти мінімум, який відповідає такому набору вихідних даних, і система буде працювати не коректно. Вихідні дані обов'язково мають бути несуперечливі. Для вирішення цієї проблеми за зазначеної ситуації необхідно збільшити розмірність простору ознак: кількість компонент вхідного вектора, відповідного зразку. Але тоді при збільшенні розмірності простору ознак може виникнути ситуація, коли число прикладів може стати недостатнім для навчання мережі, і вона, замість узагальнення, просто запам'ятає приклади з навчальної вибірки і не зможе коректно функціонувати. Таким чином, при визначенні ознак необхідно знайти компроміс з їх кількістю.

Далі необхідно визначити *спосіб подання вхідних даних* для нейронної мережі, тобто визначити спосіб *нормування*. Нормування необхідне, оскільки нейронні мережі працюють з даними, представленими числами в діапазоні 0..1, а вихідні дані можуть мати довільний діапазон або взагалі бути нечисловими даними. При цьому можливі різні способи, починаючи від простого лінійного перетворення в необхідний діапазон і закінчуючи багатовимірним аналізом параметрів і нелінійної нормуваннями залежно від впливу параметрів один на одного.

Завдання класифікації при наявності двох класів може бути вирішена на нейронній мережі з одним нейроном у вихідному шарі, який може приймати одне з двох значень 0 або 1, залежно від того, до якого класу належить зразок. За наявності декількох класів виникає проблема, пов'язана з поданням цих даних для виходу мережі. Найбільш простим способом представлення вихідних даних у такому випадку є вектор, компоненти якого відповідають різним номерами класів. При цьому *i*-а компонента вектора відповідає *i*-му класу. Всі інші компоненти при цьому мають бути нульові. Тоді, наприклад, другому класу будуть відповідати 1 на 2 виході мережі і 0 на решті інших. При інтерпретації результату зазвичай вважається, що номер класу визначається номером виходу мережі, на якому з'явилося максимальне значення. Наприклад, якщо в мережі з трьома виходами ми маємо вектор вихідних значень (0.2,0.6,0.4), то ми бачимо, що максимальне значення має друга компонента вектора, значить клас, до якого відноситься цей приклад — 2. При такому способі кодування іноді вводиться також поняття достовірності (англ. Confidence) нейронної мережі — наскільки впевнено можна стверджувати, що даний приклад відноситься до даного класу. Найбільш простий спосіб визначення достовірності полягає у визначенні різниці між максимальним значенням виходу і значенням іншого виходу, яке є найближчим до максимального. Наприклад, для розглянутого вище прикладу достовірність, що приклад відноситься до другого класу, визначиться як різниця між другою і третьою компонентою вектора і дорівнює 0.6-0.4 = 0.2, тобто 20%. Відповідно чим вища достовірність, впевненість, тим більша ймовірність того, що мережа дала правильну відповідь. Цей метод кодування є найпростішим, але не завжди найоптимальнішим способом представлення даних.

Відомі й інші способи. Наприклад, вихідний вектор являє собою номер кластера, записаний в двійковій формі. Тоді при

наявності 8-ми класів нам буде потрібен вектор з 3 елементів, і, скажімо, 3-му класу буде відповідати вектор 011. Але при цьому у разі отримання невірного значення на одному з виходів ми можемо отримати невірну класифікацію (невірний номер кластеру), тому має сенс збільшити відстань між двома кластерами за рахунок використання кодування виходу за кодом Хеммінга, який підвищить надійність класифікації.

Інший підхід полягає в розбитті задачі з k класами на $\frac{k\times(k-1)}{2}$ підзадач з двома класами (2×2 кодування) кожна. Під підзадачею в даному випадку розуміється те, що мережа визначає наявність однієї з компонент вектора. Тобто вихідний вектор розбивається на групи по два компоненти в кожній таким чином, щоб у них увійшли всі можливі комбінації компонент вихідного вектора. Число цих груп можна визначити як кількість невпорядкованих вибірок по два з вихідних компонент. з комбінаторики:

$$A_k^n = \frac{k!}{n!(k-n)!} = \frac{k!}{2!(k-2)!} = \frac{k(k-1)}{2}.$$

Тоді, наприклад, для задачі з чотирма класами ми маємо 6 виходів (підзадач) розподілених так, як показано в таблиці 2.5.3.1, де 1 на виході говорить про наявність однієї з компонент. Тоді ми можемо перейти до номеру класу за результатом розрахунку мережею наступним чином: визначаємо, які комбінації отримали одиничне (близьке до одиниці) значення виходу (тобто які підзадачі у нас активувалися), і вважаємо, що номер класу буде той, який увійшов в найбільшу кількість активованих підзадач (див. табл. 2.5.3.2).

Таблиця 2.5.3.1.

№ підзадачі	Компоненти виходу
1	1-2
2	1-3
3	1-4
4	2-3
5	2-4
6	3-4

Таблиця 2.5.3.2.

№ класу	Активовані виходи
1	1,2,3
2	1,4,5
3	2,4,6
4	3,5,6

Це кодування в багатьох задачах дає кращий результат, ніж класичний спосіб кодування.

Велике значення має правильний вибір *обсягу мережі*. Побудувати невелику і якісну модель часто буває просто неможливо, а велика модель буде просто запам'ятовувати приклади з навчальної вибірки і не робити апроксимацію, що, природно, призведе до некоректної роботи класифікатора.

Існують два основні підходи до побудови нейронної мережі — конструктивний і деструктивний. При першому з них спочатку беруть мережу мінімального розміру, і поступово збільшують її до досягнення необхідної точності. При цьому на кожному кроці її знову навчають. Також існує так званий метод каскадної кореляції, при якому після закінчення епохи відбувається коригування архітектури мережі з метою мінімізації помилки.

При деструктивному підході спочатку береться мережа завищеного обсягу, і потім з неї видаляються вузли та зв'язки, що майже не впливають на рішення. При цьому корисно пам'ятати наступне правило: число прикладів в навчальній множині повинно бути більше числа ваг, що настроюються. Інакше замість узагальнення мережа просто запам'ятає дані і втратить здатність до класифікації — результат буде невизначений для прикладів, які не увійшли до навчальну вибірку.

При виборі *архітектури мережі* звичайно випробовується кілька конфігурацій з різною кількістю елементів. При цьому основним показником є обсяг навчальної множини і узагальнююча здатність мережі. Зазвичай використовується алгоритм навчання зворотного поширення (англ. Back Propagation) з підтверджуючою множиною.

2.5.4. Алгоритм побудови класифікатора на основі нейронних мереж

Алгоритм побудови класифікатора на основі нейронних мереж має наступні кроки.

1. Робота з даними:
   - 1.1. скласти базу даних із прикладів, характерних для даної задачі;
   - 1.2. розбити всю сукупність даних на дві множини: навчальну і тестову (або на 3: навчальну, тестову і підтверджуючу).
2. Попередня обробка:

2.1. вибрати систему ознак, характерних для даної задачі, і перетворити дані відповідним чином для подачі на вхід мережі (нормування, стандартизація і т.д.). В результаті бажано отримати лінійно відокремлюваний простір множини зразків;

2.2. вибрати систему кодування вихідних значень (класичне кодування, 2 на 2 кодування і т.д.)

3. Конструювання, навчання та оцінка якості мережі:

3.1. вибрати топологію мережі, тобто кількість шарів, число нейронів у шарах і т.д.;

3.2. вибрати функцію активації нейронів (наприклад «сигмоїду»);

3.3. вибрати алгоритм навчання мережі;

3.4. оцінити якість роботи мережі на основі підтверджуючої множини або іншим критерієм, оптимізувати архітектуру (зменшення ваг, проріджування простору ознак тощо);

3.5. зупиниться на варіанті мережі, що забезпечує найкращу спроможність до узагальнення та оцінити якість роботи по тестовій множині.

4. Використання та діагностика:

4.1. з'ясувати степінь впливу різних факторів на прийняте рішення (евристичний підхід);

4.2. переконатися, що мережа дає необхідну точність класифікації (кількість неправильно розпізнаних прикладів нечисленна);

4.3. при необхідності повернуться на етап 2, змінивши спосіб представлення зразків або змінивши базу даних;

4.4. практично використовувати мережу для вирішення поставленої задачі.

Для того, щоб побудувати якісний класифікатор, необхідно мати якісні дані. Жоден з методів побудови класифікаторів, заснований на нейронних мережах або статистичний, ніколи не дасть класифікатор потрібної якості, якщо наявний набір прикладів не буде достатньо повним і представницьким для задачі, з якою доведеться працювати системі.

2.6. Дерева прийняття рішень

Дерева прийняття рішень — це спосіб представлення *правил* в ієрархічній структурі, де кожному об'єкту відповідає єдиний вузол, що дає результуюче рішення. Під правилом розуміється логічна конструкція, представлена у вигляді «якщо ... то ...» (див. рис. 2.6.1).

Дерева прийняття рішень (дерева класифікацій, регресійні дерева) — один з методів автоматичного аналізу даних. Перші ідеї створення дерев рішень сходять до робіт Ховленда (Hoveland) і Ханта (Hunt) кінця 50-х років XX століття. Однак, основоположною роботою, що дала імпульс для розвитку цього напрямку, стала книга Ханта (Hunt, EB), Мерін (Marin J.) і Стоуна (Stone, PJ) «Experiments in Induction», що побачила світ у 1966р.

Ситуації, в яких варто застосовувати дерева прийняття рішень, зазвичай виглядають наступним чином: є множіна випадків, кожен з яких описується деяким кінцевим набором дискретних атрибутів, і в кожному випадку дано значення деякої (невідомої) булевої функції, що залежить від цих атрибутів. Задача: створити економічну конструкцію, яка б описувала цю функцію і дозволяла класифікувати нові дані.

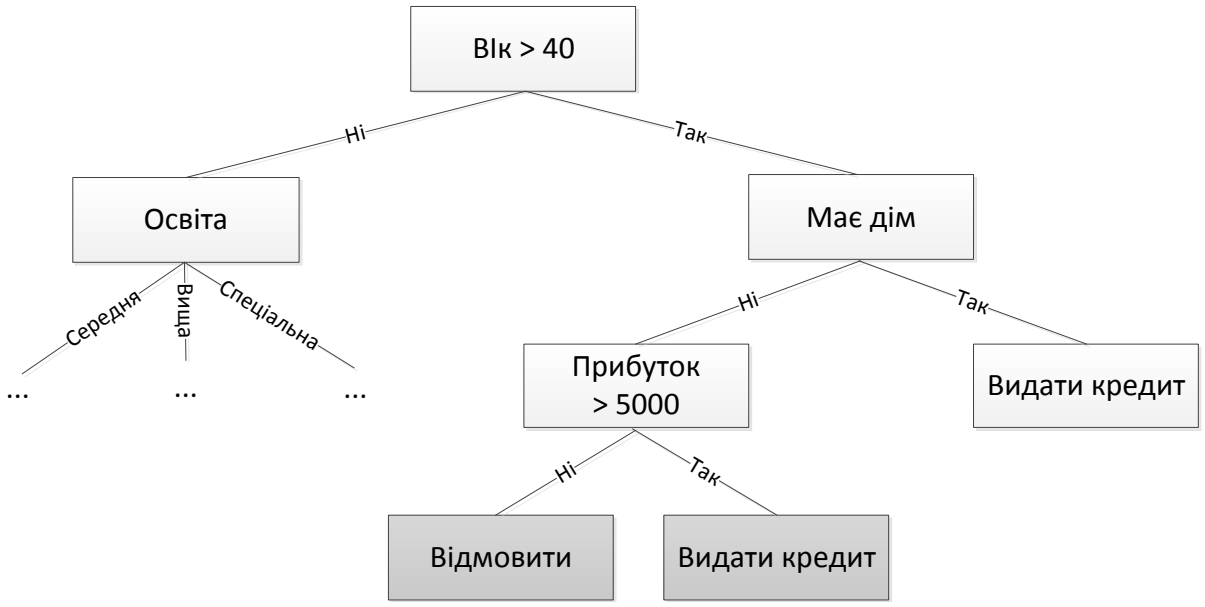


Рисунок 2.6.1. Приклад дерева прийняття рішень

Область застосування дерев рішень у даний час широка, але всі задачі, які вирішуються цим апаратом можуть бути об'єднані в наступні три класи:

- *опис даних,* при якому дерева рішень дозволяють зберігати інформацію про дані в компактній формі, у

вигляді дерева рішень, що містить точний опис об'єктів;

- *класифікація,* з якою дерева рішень відмінно справляються; проте цільова змінна для даної задачі має мати дискретні значення;
- *Регресія; я*кщо цільова змінна має безперервні значення, дерева рішень дозволяють встановити залежність цільової змінної від незалежних (вхідних) змінних; наприклад, до цього класу відносяться завдання чисельного прогнозування (передбачення значень цільової змінної).

Розглянемо застосування методу дерев прийняття рішень на прикладі задачі прогнозування.

Приклад 2.6.1. Припустимо, маючи дані про виступи футбольних клубів за останній час, ми бажаємо знати, чи виграє клуб «Динамо» свій наступний матч. Ми знаємо, що, насправді, це залежить від багатьох параметрів; проте перерахувати їх усі — задача безнадійна, тому обмежимося такими:

- чи вище суперник по турнірній таблиці;
- чи вдома грається матч;
- чи пропускає матч хтось з лідерів команди;
- чи йде дощ.

У нас є деяка статистика (див. табл. 2.6.1) і тепер ми хочемо зрозуміти, чи виграє «Динамо» в наступній грі. Припустимо, що сьогоднішній суперник стоїть нижче «Динамо» по турнірній таблиці, гра відбувається в гостях, лідери на місці і дощу немає. Спробуємо побудувати дерево прийняття рішень. В його вузлах, що не є листям, знаходяться атрибути, за якими різняться випадки. В листі знаходиться значення цільової функції. По ребрам будемо спускатися, щоб класифікувати випадки, які вже маємо.

Використаємо атрибути в порядку, в якому вони вказані в таблиці. Тоді вийде структура, зображена на рисунку 2.6.2.а).

Використання дерева рішень для відповіді на питання, що нас цікавить, зводиться до того, щоб пройтись по дереву згори вниз і визначити, в який з листків потрапляє ситуація, що нас цікавить.

Для того, щоб реалізувати машинне навчання, треба змінювати дерево прийняття рішень у відповідності до нової інформації. Наприклад, якщо «Динамо» програв, то нічого

змінювати не треба. Інакше довелось би доповнити дерево ще одним вузлом.

Побудоване дерево не ідеальне. Спробуємо взяти за основу інший атрибут, наприклад, помістимо в корінь дерева питання, чи йде дощ. Тоді глибина дерева буде дорівнювати 2 (рисунок 2.6.2.б).

2.6.1. Етапи побудови дерев рішень

При побудові дерев рішень особлива увага приділяється наступним питанням: вибір критерію атрибуту, за яким буде відбуватися розбиття, зупинки навчання і відсікання гілок. Розглянемо всі ці питання по порядку.

Яким чином слід вибрати ознаки. Для побудови дерева прийняття рішення на кожному внутрішньому вузлі необхідно знайти таку умову (перевірку), яка б розбивала множину, асоційовану з цим вузлом, на підмножини якомога більш оптимальним способом. В якості такої перевірки повинен бути вибраний один з атрибутів. Загальне правило для вибору атрибуту можна сформулювати наступним чином: обраний атрибут повинен розбити множину так, щоб результуючі підмножини складалися з об'єктів, що належать до одного класу, або були максимально наближені до цього, тобто кількість об'єктів з інших класів («домішок») у кожному з цих множин була якомога меншою.

Інтуїтивно зрозуміло, що для отримання оптимального дерева прийняття рішень, необхідно на кожному кроці обирати атрибути, що «найкраще» характеризують цільову функцію. Ця вимога формалізується за допомогою поняття *ентропії.* Припустимо, що є деяка множина A з n елементів, m з яких мають деяку властивість S. Тоді ентропія множини A по відношенню до властивості S — це:

$$H(A,S) = -\frac{m}{n}\log_2\frac{m}{n} - \frac{n-m}{n}\log_2\frac{n-m}{n}$$

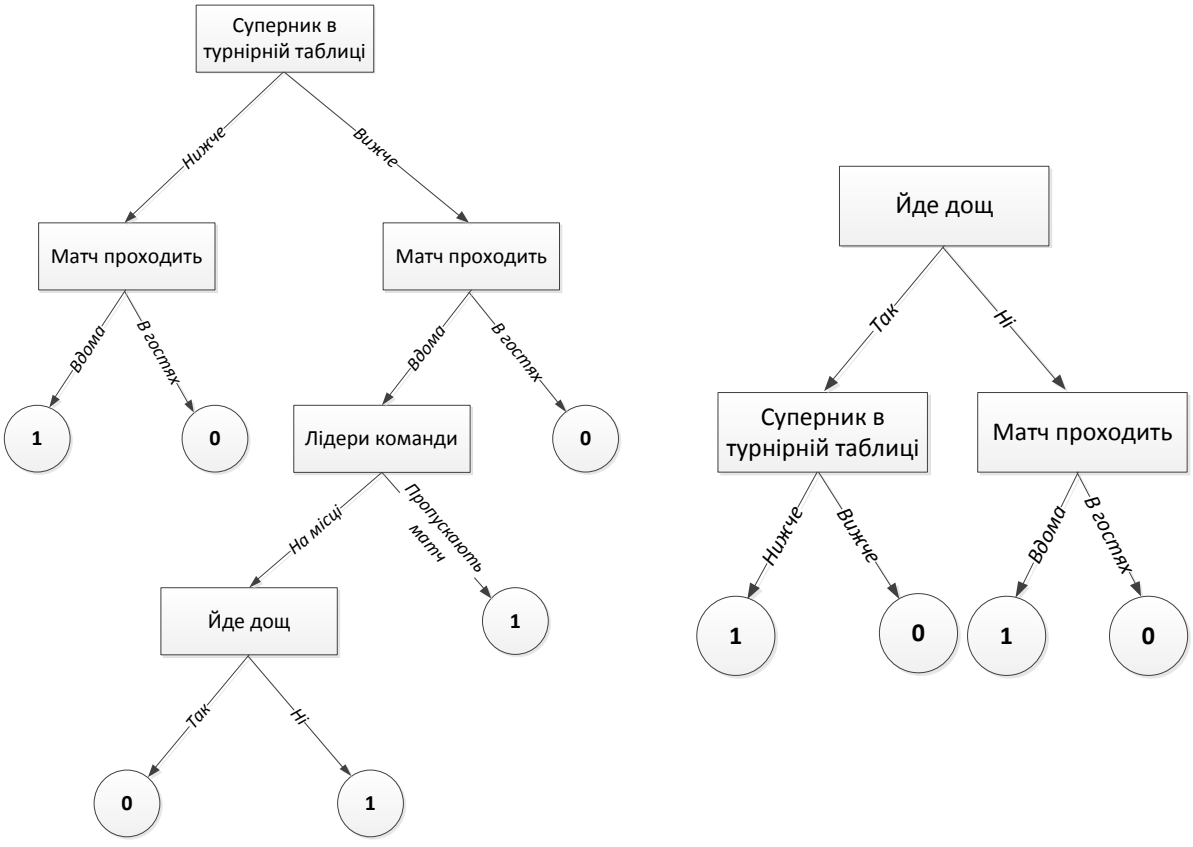
Тобто ентропія залежить від пропорції, в якій розділяється множина. По мірі зростання цієї пропорції від 0 до ½ ентропія також зростає, а після ½ — симетрично спадає.

Якщо властивість S не бінарна, а може приймати s різних значень, кожне з яких реалізується в m_i випадках, то ентропія узагальнюється таким чином:

$$H(A,S) - -\sum_{i=1}^{s}\frac{m_i}{n}\log\frac{m_i}{n}$$

Таблиця 2.6.1. Приклад даних

Суперник	Граємо	Лідери	Дощ	Перемога
Вище	Вдома	На місці	Так	Ні
Вище	Вдома	На місці	Ні	Так
Вище	Вдома	Пропускають	Ні	Так
Нижче	Вдома	Пропускають	Ні	Так
Нижче	В гостях	Пропускають	Ні	Ні
Нижче	Дома	пропускають	Так	Так
Вище	В гостях	На місці	Так	Ні
Нижче	В гостях	На місці	Ні	???



а) б)

Рисунок 2.6.2. Приклади дерева прийняття рішень

Поняття ентропії тісно пов'язане з теорією інформації. Грубо кажучи, ентропія — це середня кількість біт, які необхідні, щоб закодувати атрибут *S* елементу множини *A*. Якщо ймовірність появи *S* рівна ½, то ентропія рівна 1, і необхідний повноцінний біт, а якщо *S* з'являється нерівномірно, то можна закодувати послідовність елементів *A* ефективніше.

Таким чином, при виборі атрибуту для класифікації, треба обирати його таким чином, щоб після класифікації ентропія стала якомога меншою. Ентропія при цьому буде різною у різних нащадків, і загальну суму треба порахувати з урахуванням того, скільки результатів залишилось у розгляді у кожного з нащадків.

Загальноприйнятим в теорії дерев прийняття рішень є *теоретико-інформаційний критерій* вибору відповідного атрибуту. Припустимо, що множина *A* елементів, деякі з яких мають властивість *S*, класифіковано за допомогою атрибута *Q*, що має *q* можливих значень. Тоді приріст інформації (англ. Information Gain) визначається наступним чином:

$$\mathrm{Gain(A,Q)} = \mathrm{H(A,S)} - \sum_{i=1}^{q} \frac{|A_i|}{A} \cdot H(A_i, S) \ ,$$

де A_i — множина елементів *A*, на яких атрибут *Q* приймає значення *i*.

На кожному кроці жадібний алгоритм має обирати той атрибут, для якого приріст інформації максимальний.

Приклад 2.6.1.1. Для описаного вище прикладу визначимо оптимальний атрибут. Для цього порахуємо вихідну ентропію (хоча для максимізації приросту це, насправді, не потрібно):

$$H(A, Перемога) = -\frac{4}{7}\log_2\frac{4}{7} - \frac{3}{7}\log_2\frac{3}{7} \approx 0.9852$$

Тепер обчислимо приріст інформації для різних атрибутів:

$$Gain(A, Суперник) = H(A, Перемога) - \frac{4}{7}H(A_{вище}, Перемога) - \frac{3}{7}H(A_{нижче}, Перемога) \approx$$

$$\approx 0.9852 - \frac{4}{7}(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}) - \frac{3}{7}(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}) \approx 0.0202 \ .$$

$$Gain(A, Граємо) = H(A, Перемога) - \frac{5}{7}H(A_{вдома}, Перемога) - \frac{2}{7}H(A_{в\_гостях}, Перемога) \approx 0.4696$$

$$Gain(A, Лідери) = H(A, Перемога) - \frac{3}{7}H(A_{на\_місці}, Перемога) - \frac{4}{7}H(A_{пропускають}, Перемога) \approx 0{,}1281$$

$$Gain(A, Дощ) = H(A, Перемога) - \frac{3}{7}H(A_{так}, Перемога) - \frac{4}{7}H(A_{ні}, Перемога) \approx 0{,}1281$$

Таким чином, вирахування приросту інформації показує нам, що спочатку треба класифікувати по тому, чи вдома матч, чи в гостях. Це виглядає логічно: одна з гілок одразу обривається. Однак, якщо використовувати алгоритм далі, то глибина дерева буде рівна трьом, тобто одного атрибуту виявляється недостатньо, щоб виділити цей випадок з п'яти.

Розбивати далі вузол або відзначити його як лист. На додаток до основного методу побудови дерев рішень були запропоновані наступні правила.

- Використання статистичних методів для оцінки доцільності подальшого розбиття, так звана «рання зупинка» (англ. Prepruning). Врешті решт, «рання зупинка» процесу побудови приваблива в плані економії часу навчання, але тут доречно зробити одне важливе застереження: цей підхід будує менш точні класифікаційні моделі і тому рання зупинка вкрай небажана. Тому краще замість зупинки застосовувати відсікання.
- Обмежити глибину дерева, тобто зупинити подальшу побудову, якщо розбиття веде до дерева з глибиною, що перевищує задане значення.
- Розбиття має бути нетривіальним, тобто отримані в результаті вузли повинні містити не менше заданої кількості прикладів.

Цей список евристичних правил можна продовжити, але на сьогоднішній день не існує такого, яке б мало велику практичну цінність. До цього питання слід підходити обережно, оскільки багато хто з них застосовні лише в якихось окремих випадках.

Яким чином гілки дерева повинні відсікатися. Дуже часто алгоритми побудови дерев рішень дають складні дерева, які «переповнені даними», мають багато вузлів і гілок. Такі «гіллясті» дерева дуже важко зрозуміти. До того ж гіллясте дерево, що має багато вузлів, розбиває навчальну множину на все більшу кількість підмножин, що складаються з все меншої кількості об'єктів.

Цінність правила, справедливого скажімо для 2-3 об'єктів, вкрай низька, і в цілях аналізу даних таке правило практично непридатне. Набагато краще мати дерево, яке складається з малої кількості вузлів, яким би відповідало велика кількість об'єктів з навчальної вибірки. І тут виникає питання: а чи не побудувати всі можливі варіанти дерев, що відповідають

навчаючій множині, і з них вибрати дерево з найменшою глибиною? На жаль, ця задача є NP-повною, це було показано Л. Хайфілем і Р. Рівестом [23], і, як відомо, цей клас задач не має ефективних методів вирішення.

Для вирішення вищеописаної проблеми часто застосовується так зване відсікання гілок (англ. Pruning).

Будемо розуміти точність дерева рішень як відношення правильно класифікованих об'єктів при навчанні до загальної кількості об'єктів з навчальної множини, а помилку — як кількість неправильно класифікованих. Припустимо, що нам відомий спосіб оцінки помилки дерева, гілок і листя. Тоді, можна використовувати наступне:

- побудувати дерево;
- відсікти або замінити піддеревом ті гілки, які не приведуть до зростання помилки.

На відміну від процесу побудови, відсікання гілок відбувається знизу вгору, рухаючись з листя дерева, відзначаючи вузли як листя, або замінюючи їх піддеревом.

Хоча відсікання не є панацеєю, але в більшості практичних завдань дає добрі результати, що дозволяє говорити про правомірність використання подібної методики.

Іноді навіть усічені дерева можуть бути все ще складні для сприйняття. У такому випадку, можна вдатися до методики вилучення правил з дерева з наступним створенням наборів правил, що описують класи. Для вилучення правил необхідно досліджувати всі шляхи від кореня до кожного аркуша дерева. Кожен такий шлях дасть правило, де умовами будуть перевірки з вузлів зустрілися на шляху.

2.6.2. Дерева прийняття рішень і булеві функції

Дерево прийняття рішень можна читати як булеву функцію. Вона буде природнім чином виражатися в кон'юктивній нормальній формі (КНФ). Наприклад, дерево на рисунку 2.6.2.а відповідає функції:

$$((\text{Суперник} = \text{Нижче}) \wedge (\text{Граємо} = \text{Вдома})) \vee$$
$$\vee((\text{Суперник} = \text{Вище}) \wedge (\text{Граємо} = \text{Вдома}) \wedge (\text{Лідери} = \text{Пропускають})) \vee$$
$$\vee((\text{Дощ} = \text{Так}) \wedge (\text{Суперник} = \text{Нижче})) \vee$$
$$\vee((\text{Дощ} = \text{Так}) \wedge (\text{Граємо} = \text{Вдома}))$$

2.6.3. Алгоритм побудови дерев прийняття рішень ID3

Алгоритм ID3 був придуманий Джоном Р. Квинланом [33]. У ньому, в порівнянні з попередніми прикладами, зустрічаються два нетривіальних, але важливих узагальнення. По перше, алгоритм оброблює ситуацію, коли одному і тому ж набору атрибутів відповідають декілька випадків з різними результатами. Тоді, коли ми дійдемо до кінця дерева, атрибутів, котрі могли б розділити результати, не залишається. Для цього пункт 4 алгоритму — рішення буде прийматися більшістю.

Інше узагальнення — у атрибута може зустрітися декілька варіантів. До того ж, може статися, що якийсь з цих варіантів не реалізується, у такому випадку будемо заповнювати відповідний лист у залежності від того, яких результатів було більше в його предка (пункт 7б алгоритму).

ID3 (A,S,Q):

1. створюємо корінь дерева;
2. якщо S виконується на всіх елементах A — поставити в корінь мітку 1 і вийти;
3. якщо S не виконується на жодному елементі A — поставити в корінь мітку 0 і вийти;
4. Якщо Q = 0, то:
    a. якщо S виконується на половині чи більшій частині A — поставити в корінь мітку 1 і вийти;
    b. якщо S не виконується на більшій частині A — поставити в корінь мітку 0 і вийти;
5. вибрати $Q_i \in Q$ для якого Gain(A,Q_i) — максимальний$
6. gоставити в корінь мітку Q_i;
7. для кожного значення q атрибута Q_i:
    a. додати нового нащадка кореня і помітити відповідне висхідне ребро міткою *q;*
    b. якщо в A немає випадків, для яких Q_i приймає значення q (тобто $|A_q|$=0), то помітити цього нащадка в залежності від того, на якій частині A виконується S (аналогічно пункту 4);
    c. інакше запустити ID3($A_q,S,Q\backslash\{Q_i\}$) і додати його в результат як піддерево з коренем в цьому нащадку.

2.6.4. Переваги використання дерев рішень

Переваги використання дерев рішень:

- генерація правил в областях, де експерту важко формалізувати свої знання;
- швидкий процес навчання;
- витяг правил природною мовою;
- інтуїтивно зрозуміла класифікаційна модель;
- висока точність прогнозу, порівнянна з іншими методами (статистика, нейронні мережі);
- побудова непараметричних моделей.

В силу цих та багатьох інших причин, методологія дерев рішень є важливим інструментом у роботі кожного фахівця, що займається аналізом даних, незалежно від того практик він чи теоретик.

2.6.5. Проблеми з критерієм приросту інформації

Приріст інформації досить часто обирає атрибути, в яких більше значень. Наприклад, припустимо, що в таблиці ігор були також записані і дати ігор, причому у кожного матчу своя дата. Тоді приріст інформації:

$$Gain(A, Дата) = H(A, Перемога) - \sum_{i=1}^{n} \frac{1}{n} H(A_{Дата=i}, Перемога) = H(A, Перемога)$$

тому що в кожній із гілок лише один випадок, і ентропія у кожній гілці дорівнює нулю. Таким чином, приріст інформації в цьому випадку буде максимальним з можливих. Але отримане дерево буде абсолютно некорисним: навіть якщо дата і мала відношення до сили команди, все-одно дати, що пройшли, більше ніколи не повторяться.

В таких випадках треба модифікувати критерій приросту інформації, щоб він міг упоратися з цією проблемою: треба зменшувати важливість атрибута з ростом його значень.

Розглянемо критерій *Gain Ratio*, що враховує не тільки кількість інформації, необхідної для запису результату, але і кількість інформації, необхідної для розділення по поточному атрибуту. Ця поправка виглядає таким чином:

$$\mathrm{SplitInfo(A,Q)} = -\sum_{i=1}^{q} \frac{|A_i|}{A} \cdot \log_2 \frac{|A_i|}{A},$$

а сам критерій — як максимізація величини:

$$\mathrm{GainRatio(A,Q)} = \frac{\mathrm{Gain(A,Q)}}{\mathrm{SplitInfo(A,Q)}}$$

Ще один критерій, що часто використовується — *індекс Гіні.* Для набору тестів A і властивості S, що має s значень, цей індекс підраховується наступним чином:

$$\mathrm{Gini}(\mathrm{A},\mathrm{S}) = 1 - \sum_{i=1}^{s} \frac{|A_i|}{A}$$

Відповідно, для набору тестів A і атрибуту Q, що має q значень і цільової властивості S, що має s значень, індекс вираховується наступним чином:

$$\mathrm{Gini}(\mathrm{A},\mathrm{Q},\mathrm{S}) = Gini(A,S) - \sum_{i=1}^{q} \frac{|A_i|}{A} \cdot Gini(A,S)$$

Індекс Гіні компенсує частково ухил критерію приросту інформації в сторону вибору найбільш гіллястих атрибутів. Але в іншому вони досить схожі; індекс Гіні і критерій приросту інформації не узгоджуються лише на ~2% випадків.

2.6.6. Перенавчання

Якщо дерево будується за допомогою алгоритму ID3 чи аналогічного йому, отримане дерево задовольняє всім даним, що маємо. Але на практиці в результаті часто отримується дуже деталізовані дерева, і кількість помилок при подальшому використанні побудованого дерева росте. Це явище називається перенавчанням (англ. Overfitting).

Ефект перенавчання легко проілюструвати на наступному прикладі. Припустимо. Що гра «Динамо» в ясну погоду взагалі ні від чого більше не залежить: «Динамо» виграє у 90% випадків. Але серед вхідних даних маємо одну поразку вдома в ясний день. Тоді ID3 старанно врахує всі (що не мають відношення до суті справи) «причини» цієї поразки і буде в подальшому прогнозувати, що «Динамо» програє в аналогічних ситуаціях. Насправді ж «Динамо» буде вигравати з тією ж ймовірність 90%. Таким чином, дерево рішень не буде виконувати свою задачу.

Можна, звичайно, просто штучно зупиняти ріст дерева в глибину. Але коли і на якому етапі це варто робити? Зазвичай, щоб уникнути перенавчання, використовують так назване обрізання (англ. Pruning): спочатку дозволяють дереву рости далі, а згодом обрізають лишні гілки. Обрізання дерева в деякому сенсі заключається в тому, що піддерево з коренем у цьому вузлі видаляється з дерева, а у вузол поміщається мітка з тим результатом, яких в цьому вузлі більшість. Залишилось тільки зрозуміти, які гілки видаляти.

Подамо один із можливих методів. Розіб'ємо вихідні дані на множину, на якій дерево буде будуватися, і множину, на якій будемо тестувати дерево. По першій частині побудуємо дерево прийняття рішень. Тепер у нас є дерево і набір еталонних тестів, правильні відповіді на які нам відомі. Головною мірою успіху для нас буде те, наскільки добре дерево справляється з тесами. Відповідно, можна просто розглянути кожну вершину як кандидата на обрізання і перевірити, чи стане урізане дерево краще справлятися з тестами. На кожному кроці ми будемо вибирати таку вершину, обрізання якої максимально покращує поведінку дерева прийняття рішень. Ці вершини будемо залишати обрізаними і продовжувати до того часу, поки обрізання будь-якої вершини не буде призводити до більш слабкого результату на еталонних тестах.

Є і інші методи боротьби з перенавчанням.

2.6.7. Області застосувань дерев рішень

Дерева рішень є прекрасним інструментом в системах підтримки прийняття рішень, інтелектуального аналізу даних. До складу багатьох пакетів, призначених для інтелектуального аналізу даних, вже включені методи побудови дерев рішень. В областях, де висока ціна помилки, вони послужать відмінною підмогою аналітика чи керівника

Дерева рішень успішно застосовуються для вирішення практичних завдань в наступних областях.

- Банківська справа: оцінка кредитоспроможності клієнтів банку при видачі кредитів.
- Промисловість: контроль за якістю продукції (виявлення дефектів), випробування без руйнувань (наприклад перевірка якості зварювання) і т.д.
- Медицина: діагностика різних захворювань.
- Молекулярна біологія: аналіз будови амінокислот.

Це далеко не повний список областей де можна використовувати дерева рішень. Не дослідженими залишаються ще багато потенційних областей застосування.

2.7. Алгоритм найближчого сусіда

Людина, зустрічаючись з новою задачею, використовує свій життєвий досвід, згадує аналогічні ситуації, які колись з ним відбувалися. Про властивості нового об'єкта ми судимо, покладаючись на схожі знайомі спостереження. Наприклад,

зустрівши іноземця на вулиці, ми можемо здогадатися про його походження по акценту, жестах і зовнішності. Для цього необхідно згадати найбільш схожої людини на нього, походження якого відомо.

Так, подібно до наведеного вище прикладу, подібність об'єктів лежить в основі *алгоритму k-найближчих сусідів* (англ. k-nearest neighbor algorithm, KNN). Алгоритм здатний виділити серед всіх спостережень k відомих об'єктів (k-найближчих сусідів), схожих на новий невідомий раніше об'єкт. На основі класів найближчих сусідів виноситься рішення щодо нового об'єкта. Важливим завданням даного алгоритму є підбір коефіцієнта k — кількості записів, які будуть вважатися схожими.

Алгоритм KNN широко застосовується в Data Mining. Формулюється алгоритм наступним чином. Нехай є n спостережень, кожному з яких відповідає запис у таблиці. Усі записи належать якого-небудь класу. Необхідно визначити клас для нового запису.

2.7.1. Суть алгоритму

На першому кроці алгоритму слід задати число k — кількість *найближчих сусідів*. Якщо прийняти $k = 1$, то алгоритм втратить узагальнюючу здатність (тобто здатність видавати правильний результат для даних, що не зустрічалися раніше в алгоритмі), оскільки новому запису буде присвоєний клас, близький до нього. Якщо встановити занадто велике значення, то багато локальних особливостей не буде виявлено.

На другому кроці знаходяться k записів з мінімальною відстанню до вектора ознак нового об'єкта (пошук сусідів). Функція для розрахунку відстані повинна відповідати наступним правилам:

- $d(x,y) \geq 0$,
- $d(x,y) = 0$ тоді і тільки тоді, коли $x=y$;
- $d(x,y) = d(y,x)$;
- $d(x,z) \leq d(x, y) + d(y, z)$, за умови, що точки x, y, z лежать на одній прямій.

Для впорядкованих значень атрибутів знаходиться *Евклідова відстань*:

$$D_E = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \ ,$$

де n — кількість атрибутів.

Для строкових змінних, які не можуть бути впорядковані, може бути застосована функція відмінності, яка задається наступним чином:

$$d(x, y) = \begin{cases} 0, & x = y \\ 1, & x \neq y \end{cases}.$$

Часто перед розрахунком відстані необхідна *нормалізація*. Наведемо деякі корисні формули.

Мінімаксна нормалізація:

$$X^{\times} = \frac{X - \min(X)}{\max(X) - \min(X)} .$$

Нормалізація за допомогою стандартного відхилення:

$$X^{\times} = \frac{X - X_{cp}}{\sigma_x} .$$

де σ_x — стандартне відхилення, X_{cp} — середнє значення.

При знаходженні відстані іноді враховують значущість атрибутів. Вона визначається експертом або аналітиком суб'єктивно, покладаючись на власний досвід. У такому випадку при знаходженні відстані кожен *i*-ий квадрат різниці в сумі множиться на коефіцієнт Z_i. Наприклад, якщо атрибут *A* в три рази важливіше атрибута *B* (Z_A=3, Z_B=1), то відстань буде знаходитися наступним чином:

$$D_E = \sqrt{3 \cdot (x_A - y_A)^2 + (x_B - y_B)^2} \ .$$

Подібний прийом називають *розтягуванням осей* (англ. Stretching the Axes), що дозволяє знизити помилку класифікації.

Слід зазначити, що якщо для запису *A* найближчим сусідом є *B*, то це не означає, що *B* — найближчий сусід *A*. Дана ситуація представлена на рисунку 2.7.1.1: при *k* = 1 найближчій для точки *B* буде точка *A*, а для *A* — *X*; при збільшенні коефіцієнта до *k* = 7, точка *B* так само не буде входити в число сусідів.

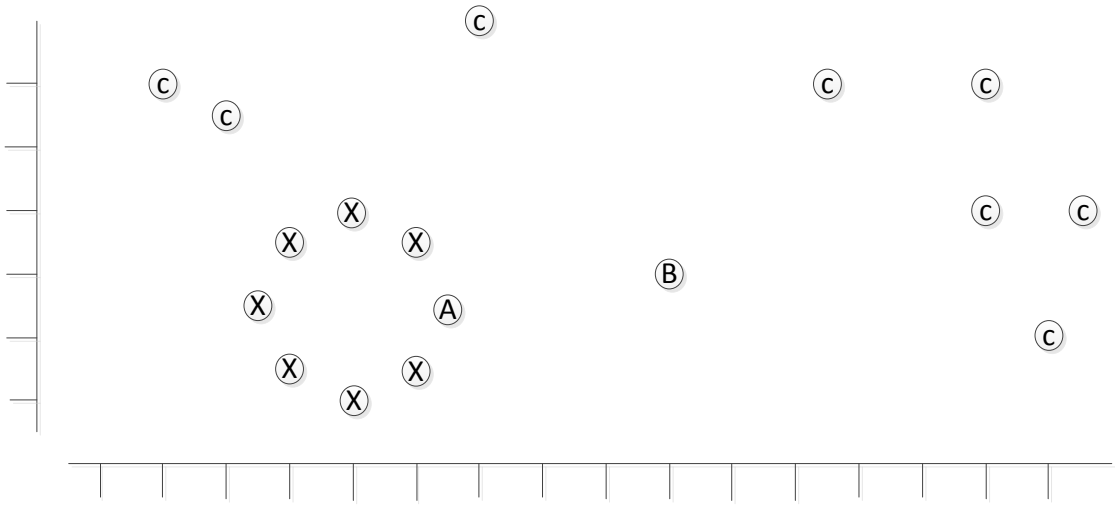


Рисунок 2.7.1.1 Найближчі сусіди A і B

На наступному кроці, коли знайдені записи, найбільш схожі на нову, необхідно вирішити, як вони впливають на клас нового запису. Для цього використовується *функція поєднання* (англ. Combination Function). Одним з основних варіантів такої функції є *просте незважене голосування* (англ. Simple Unweighted Voting).

Спочатку, знайшовши число *k*, ми дізналися, скільки записів буде мати право голосу при визначенні класу. Потім ми виявили ці записи, відстань від яких до нової виявилося мінімальним. Тепер можна приступити до простого *незваженого голосування*.

Відстань від кожного запису при голосуванні тут більше не грає ролі. Всі мають рівні права у визначенні класу. Запис віддає перевагу тому класу, до якого вона належить, а той, який набере найбільшу кількість голосів, присвоюється нового запису.

Але що робити у випадку, якщо декілька класів набрали рівну кількість голосів? Цю проблему знімає *зважене голосування* (англ. Weighted Voting). У такій ситуації враховується також і відстань до нового запису. Чим менше відстань, тим більше значущий внесок вносить голос. Голоси за клас знаходяться за наступною формулою:

$$votes(class) = \sum_{i=1}^{n} \frac{1}{d^2(X, Y_i)},$$

де $d^2(X, Y_i)$ — квадрат відстаней від відомого запису Y_i до нового X, n — кількість відомих записів, для яких розраховуються голоси, *class* — назва класу.

Клас, який набрав найбільшу кількість голосів, присуджується новому запису. При цьому ймовірність того, що кілька класів наберуть однакові голоси, набагато нижче. Цілком очевидно, що при $k = 1$ нового запису присвоюється клас найближчого сусіда.

2.7.2. Гіпотеза компактності

Якщо міра подібності об'єктів введена досить вдало, то схожі об'єкти набагато частіше лежать в одному класі, ніж в різних. В цьому випадку межа між класами має досить просту форму, а класи утворюють компактно локалізовані області в просторі об'єктів[6].

2.7.3. Вагова функція

Для даного алгоритму вагову функцію можна обчислювати по різному. Нехай задана навчальна вибірка пар «об'єкт-відповідь»:

$$X^m = \{(x_1,y_1),\dots,(x_m,y_m)\}$$

Нехай на множині об'єктів задана функція відстані $\rho(x,x')$. Ця функція повинна бути досить адекватною моделлю подібності об'єктів. Чим більше значення цієї функції, тим менш схожими є два об'єкти x, x'.

Для довільного об'єкта u розташуємо об'єкти навчальної вибірки x_i в порядку зростання відстаней до u:

$$\rho(u,x_{1;u}) \le \rho(u,x_{2;u}) \le \dots \le \rho(u,x_{m;u}),$$

де через $x_{i;u}$ позначається той об'єкт навчальної вибірки, який є i-м сусідом об'єкта u. Аналогічне позначення введемо і для відповіді на i-м сусіда: $y_{;u}$. Таким чином, довільний об'єкт u породжує свою перенумерацію вибірки. У найбільш загальному вигляді алгоритм найближчих сусідів є

$$a(u) = \arg\max_{y \in Y} \sum_{i=1}^{m} [y(x_{i;u}) = y] w(i,u),$$

де $w(i,u)$ — задана вагова функція, яка оцінює ступінь важливості і-го сусіда для класифікації об'єкта u. Природно вважати, що ця функція невід'ємна і не зростає по i.

По-різному задаючи вагову функцію, можна отримувати різні варіанти методу найближчих сусідів (таблиця 2.7.3.1).

---

[6] Зауважимо, що в математичному аналізі компактними називаються обмежені замкнуті множини. Гіпотеза компактності не має нічого спільного з цим поняттям, і повинна розумітися швидше в «побутовому» сенсі цього слова.

Таблиця 2.7.3.1. Варіанти методу найближчих сусідів

Вагова функція	*Назва методу*
$w(i,u)=[i=1]$	найпростіший метод найближчого сусіда
$w(i,u)=[i\le k]$	метод k найближчих сусідів
$w(i,u)=[i\le k]q^i$	метод k експоненціально зважених найближчих сусідів, де передбачається q <1
$w(i,u)=K\left(\frac{\rho(u,x_{i;u})}{h}\right)$	метод парзенівського вікна фіксованої ширини h
$w(i,u)=K\left(\frac{\rho(u,x_{i;u})}{\rho(u,x_{k+1;u})}\right)$	метод парзенівського вікна змінної ширини
$w(i,u)=K\left(\frac{\rho(u,x_{i;u})}{h(x_{i;u})}\right)$	метод потенційних функцій, в якому ширина вікна $h(x_i)$ залежить не від об'єкта класифікації, а від навчального об'єкта x_i

В таблиці 2.7.3.1 $K(r)$ — задана невід'ємна монотонно незростаюча функція на $[0,+\infty)$, *ядро згладжування.*

2.7.4. Застосування алгоритму

Алгоритм k-найближчих сусідів має широке застосування. Наприклад:

- виявлення шахрайства: нові випадки шахрайства можуть бути схожі на ті, які відбувалися колись у минулому, і алгоритм KNN може розпізнати їх для подальшого розгляду;
- передбачення відгуку клієнтів: можна визначити відгук нових клієнтів за даними з минулого;
- медицина: алгоритм може класифікувати пацієнтів за різними показниками, ґрунтуючись на даних минулих періодів;
- інші завдання, що вимагають класифікацію об'єктів.

2.7.5. Основні проблеми алгоритму і шляхи їх вирішення

2.7.5.1. Вибір числа сусідів k

При $k = 1$ алгоритм найближчого сусіда нестійкий до шумових викидів: він дає помилкові класифікації не тільки на самих об'єктах-викидах, а й на найближчих до них об'єктах інших класів. При $k = m$, навпаки, алгоритм надмірно стійкий і вироджується в константу. Таким чином, крайні значення k небажані. На практиці оптимальне значення параметра k

визначають за критерієм змінного контролю, найчастіше — методом виключення об'єктів по одному (англ. Leave-One-Out Cross-Validation).

2.7.5.2. Відсів шумів (викидів)

Зазвичай об'єкти навчання не є рівноцінними. Серед них можуть знаходитися типові представники класів — *еталони*. Якщо класифікується об'єкт близький до ідеалу, то, швидше за все, він належить до того ж класу. Ще одна категорія об'єктів — *неінформативні* або *периферійні*. Вони щільно оточені іншими об'єктами того ж класу. Якщо їх видалити з вибірки, це практично не позначиться на якості класифікації. Нарешті, до вибірки може потрапити кілька шумових викидів — об'єктів, що знаходяться «в гущі» чужого класу. Як правило, їх видалення тільки покращує якість класифікації.

Видалення з вибірки шумових і неінформативних об'єктів дає кілька переваг одночасно: підвищується якість класифікації, скорочується обсяг збережених даних і зменшується час класифікації, що витрачається на пошук найближчих еталонів.

2.7.5.3. Надвеликі вибірки

Надвеликі вибірки ($m \geq 10^3$) створюють кілька чисто технічних проблем: необхідно не тільки зберігати великий обсяг даних, але і вміти швидко знаходити серед них k найближчих сусідів довільного об'єкта u.

Проблема вирішується двома способами:

- вибірка проріджується шляхом викидання неінформативних об'єктів;
- застосовуються спеціальні індекси і ефективні структури даних для швидкого пошуку найближчих сусідів (наприклад, kd-дерева).

2.7.5.4. Проблема вибору метрики

Це найскладніша з проблем. У практичних завданнях класифікації рідко зустрічаються такі «ідеальні випадки», коли заздалегідь відома хороша функція відстані $p(x,x')$. Якщо об'єкти описуються числовими векторами, часто беруть евклідову метрику. Цей вибір, як правило, нічим не обґрунтований — просто це перше, що спадає на думку. При цьому необхідно пам'ятати, що всі ознаки мають бути виміряні «в одному масштабі», а найкраще — віднормовані. В іншому випадку озна-

ка з найбільшими числовими значеннями буде домінувати в метриці, інші ознаки, фактично, враховуватися не будуть.

Однак і нормування є вельми сумнівною евристикою, так як залишається питання: «невже всі ознаки однаково значущі і повинні враховуватися приблизно з однаковою вагою?»

Якщо ознак занадто багато, а відстань обчислюється як сума відхилень за окремими ознаками, то виникає проблема *прокляття розмірності.* Суми великого числа відхилень з великою ймовірністю мають дуже близькі значення (відповідно до закону великих чисел). Виходить, що в просторі високої розмірності всі об'єкти приблизно однаково далекі один від одного; вибір *k* найближчих сусідів стає практично довільним.

Проблема вирішується шляхом відбору відносно невеликого числа інформативних ознак (англ. Features Selection). В алгоритмах обчислення оцінок будується множина різних наборів ознак (т. зв. опорних множин), для кожного будується своя функція близькості, потім по всіх функцій близькості проводиться голосування.

2.7.6. Переваги і недоліки алгоритму

На закінчення відзначимо достоїнства і недоліки алгоритму KNN. Перерахуємо позитивні особливості.

- Алгоритм стійкий до аномальних викидів, оскільки ймовірність потрапляння такого запису в число *k*-найближчих сусідів мала. Якщо ж це сталося, то вплив на голосування (особливо зважене) (при *k>2*) також, швидше за все, буде незначним, і, отже, малим буде і вплив на підсумок класифікації.
- Програмна реалізація алгоритму відносно проста.
- Результат роботи алгоритму легко піддається інтерпретації. Експертам у різних областях цілком зрозуміла логіка роботи алгоритму, заснована на знаходженні схожих об'єктів.
- Можливість модифікації алгоритму, шляхом використання найбільш придатних функцій поєднання і метрик дозволяє підлаштувати алгоритм під конкретну задачу.

Алгоритм KNN має і низкою недоліків. По-перше, набір даних, використовуваний для алгоритму, повинен бути репрезентативним. По-друге, модель не можна «відокремити» від даних: для класифікації нового прикладу потрібно вико-

ристовувати всі приклади. Ця особливість сильно обмежує використання алгоритму. Крім того, метод найближчих сусідів заснований на явному зберіганні всіх навчальних об'єктів.

Висновки до розділу 2

Класифікація — задача розбиття множини об'єктів або спостережень на апріорно задані групи, звані класами, всередині кожної з яких вони передбачаються схожими один на одного, мають приблизно однакові властивості і ознаки. При цьому рішення отримується на основі аналізу значень атрибутів (ознак).

Розрізняють *допоміжну (штучну)* та *природну класифікацію*. Класифікація також може бути *одномірною* (за однією ознакою) і *багатовимірною.*

Класифікація відноситься до *стратегії навчання з учителем.*

Оцінка точності класифікації може проводитися за допомогою *крос-перевірки.*

Оцінювання методів слід проводити, виходячи з таких характеристик: *швидкість, робастність, інтерпретованість, надійність.*

Алгоритм побудови елементарних правил (1R, OneR, 1-rule) — простий алгоритм формування правил для класифікації об'єкта, цей алгоритм будує правила за значенням тільки однієї незалежної змінної.

Наївний байєсівський класифікатор один з найпростіших з алгоритмів класифікації. Тим не менш, досить часто він працює не гірше, а то і краще складніших алгоритмів.

Штучні нейронні мережі — здатні до самонавчання системи, що імітують діяльність людського мозку. Нейронні мережі зворотного поширення — це найпотужніший інструмент пошуку закономірностей, прогнозування, якісного аналізу.

Дерева прийняття рішень — це спосіб представлення правил в ієрархічній, послідовної структурі, де кожному об'єкту відповідає єдиний вузол, що дає рішення.

Алгоритм KNN здатний виділити серед всіх спостережень k відомих об'єктів (k-найближчих сусідів), схожих на новий невідомий раніше об'єкт.

Питання до розділу 2

1. В чому полягає задача класифікації даних?
2. Які типи класифікації ви знаєте?
3. Яким чином перевіряється точність класифікації?
4. Чим відрізняються робастність та надійність?
5. Сформулюйте алгоритм побудови елементарних правил.
6. Яка закономірність лежіть в основі наївного байєсівського класифікатора?
7. Про що говорить формула Баєса?
8. Сформулюйте алгоритм наївного байєсівського класифікатора.
9. Що таке штучні нейронні мережі?
10. Яка будова перцептрона?
11. Що таке активаційна функція?
12. Функції якого типу найчастіше застосовуються у якості активаційної функції і чому?
13. Яким чином штучні нейронні мережі застосовуються для розв'язання задачі класифікації?
14. Сформулюйте алгоритм побудові дерева прийняття рішень?
15. Що таке ентропія і як вона застосовується при побудові дерева прийняття рішень?
16. В чому основна ідея алгоритму к-найближчих сусідів?
17. Які алгоритми KNN ви знаєте і чим вони відрізняються?

Вправи до розділу 2

1. Перевірте правила «Якщо ключових слів < 50%, то стаття релевантна» та «Якщо тематика сайту не підходить, то стаття не релевантна» для даних з таблиці 1.2.
2. Для даних з таблиць, поданих на рисунку 2.1.a-f побудувати дерево прийняття рішень (S=1) використовуючи теоретико-інформаційний критерій вибору атрибута та знайти клас останнього об'єкта.
3. Намалювати дерево прийняття рішень, що відповідає функції a-f:

a. $x \vee (y \wedge \bar{z})$ b. $(x \vee y) \wedge (\bar{y} \vee z)$

c. $(x \vee \bar{y}) \wedge (y \vee \bar{z} \vee t)$ d. $(x \wedge \bar{y}) \vee (y \wedge \bar{z} \wedge t)$

e. $(\bar{x} \vee y) \wedge (\bar{y} \vee z \vee \bar{t})$ f. $(x \wedge \bar{y} \wedge t) \vee (\bar{x} \wedge \bar{z}) \vee (x \wedge y \wedge z \wedge t)$

4. Для булевих функцій із 3 намалювати перцептрони.
5. Побудувати мережу із перцептронів глибини 2, що реалізує функцію XOR.
4. Скориставшись алгоритмом найближчого сусіда, визначити клас точки X для даних, поданих на рисунку 2.2.a-2.2.d.

Q1	Q2	Q3	Q4	S
0	0	1	0	0
1	0	0	1	1
2	0	1	0	1
0	1	0	0	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	0
2	0	0	0	1
2	1	1	0	1
0	1	1	1	0
2	1	1	1	?

a

Q1	Q2	Q3	Q4	S
0	0	1	0	0
1	0	0	1	1
2	0	1	0	1
0	1	0	0	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	0
2	0	0	0	1
2	1	1	0	1
0	1	1	1	0
2	1	1	1	?

b

Q1	Q2	Q3	Q4	S
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
0	1	1	1	?

c

Q1	Q2	Q3	Q4	S
0	0	1	0	0
1	0	2	1	1
2	0	1	0	1
0	1	0	0	1
0	1	1	0	1
0	1	2	1	0
1	0	0	1	0
2	0	0	0	1
2	1	2	0	1
0	1	1	1	0
2	1	1	1	?

d

Q1	Q2	Q3	Q4	S
0	0	1	0	0
1	0	0	1	1
2	0	1	0	1
0	1	0	0	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	0
2	0	0	0	1
2	1	1	0	1
0	1	1	1	0
2	1	1	1	?

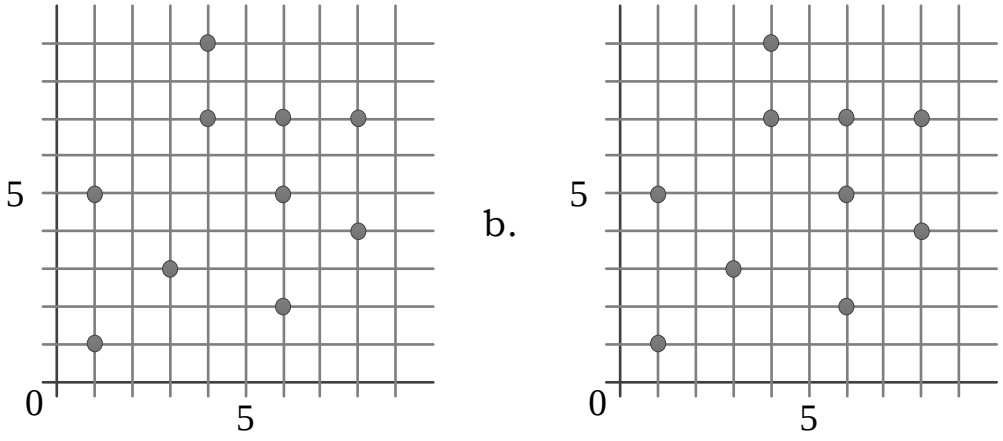
e

Q1	Q2	Q3	Q4	S
0	0	1	0	0
1	0	0	1	1
2	0	1	0	1
0	1	0	0	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	0
2	0	0	0	1
2	1	1	0	1
0	1	1	1	0
2	1	1	1	?

f

Рисунок 2.1. Вхідні дані

a. b.



c. d.

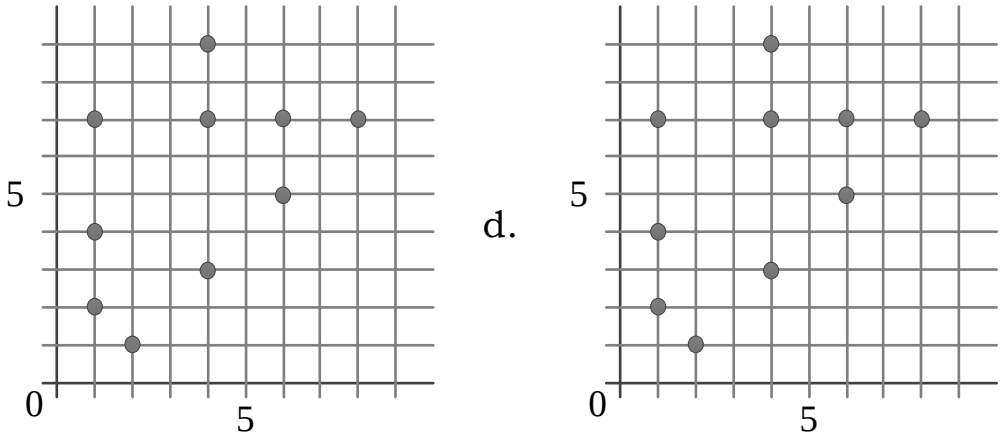


Рисунок 2.2. Вхідні дані

Розділ 3. Пошук асоціативних правил

3.1. Задача пошуку асоціативних правил

Асоціативні правила дозволяють знаходити закономірності між пов'язаними подіями. Прикладом такого правила, служить твердження, що покупець, що придбає «Хліб», придбає і «Молоко» з імовірністю 75%. Перший алгоритм пошуку асоціативних правил, що називався AIS був розроблений в 1993 році співробітниками дослідницького центру IBM Almaden. З цією піонерської роботи зріс інтерес до асоціативних правилам; на середину 90-х років минулого століття припав пік дослідницьких робіт в цій області, і з тих пір щороку з'являлося по декілька алгоритмів.

Вперше ця задача була запропонована для знаходження типових шаблонів покупок, що здійснюються в супермаркетах, тому іноді її ще називають аналізом ринкової корзини (англ. Market Basket Analysis).

3.1.1. Асоціативні правила

Нехай є база даних, що складається з купівельних транзакцій. Кожна транзакція — це набір товарів, куплених покупцем за один візит. Таку транзакцію ще називають ринковою корзиною.

Нехай $I = \{i_1, i_2, i_3, \ldots, i_n\}$ — множина (набір) товарів (елементів), D — множина транзакцій, де кожна транзакція T — набір елементів з I, $T \subseteq I$. Кожна транзакція являє собою бінарний вектор, де $t[k]=1$, якщо i_k присутній в транзакції, інакше $t[k] = 0$. Будемо говорити, що транзакція T містить X, деякий набір елементів з I, якщо $X \subset T$.

Асоціативним правилом називається імплікація $X \Rightarrow Y$, де $X \subset I$, $Y \subset I$ і $X \cap Y = \varnothing$.

Правило $X \Rightarrow Y$ *має підтримку* (англ. Support) s, якщо s% транзакцій з D містять $X \cup Y$, $supp(X \Rightarrow Y) = supp(X \cup Y)$.

Достовірність (англ. Confidence) правила вказує, яка ймовірність того, що з X слідує Y. Правило $X \Rightarrow Y$ справедливе з достовірність c, якщо c% транзакцій з D, що містять X, також містять Y, $conf(X \Rightarrow Y) = supp(X \cup Y) / supp(X)$.

Іншими словами, метою даного аналізу є встановлення наступних залежностей: якщо в транзакції зустрівся деякий набір елементів X, то на підставі цього можна зробити висновок

про те, що інший набір елементів Y також же повинен з'явитися в цій транзакції. Встановлення таких залежностей дає нам можливість знаходити дуже прості і інтуїтивно зрозумілі правила.

Алгоритми пошуку асоціативних правил призначені для знаходження всіх правил $X \Rightarrow Y$, причому підтримка і достовірність цих правил повинні бути вище деяких наперед визначених порогів, званих відповідно *мінімальною підтримкою* (minsupport) і *мінімальної достовірністю* (minconfidence).

Задача знаходження асоціативних правил розбивається на дві підзадачі:

- знаходження всіх наборів елементів, які задовольняють порогу minsupport; такі набори елементів називаються такими, що часто зустрічаються;
- генерація правил з наборів елементів, знайдених згідно з достовірністю, що задовольняє порогу minconfidence.

Один з перших алгоритмів, що ефективно вирішує подібний клас задач — це алгоритм APriori. Крім цього алгоритму останнім часом був розроблений ряд інших алгоритмів: DHP, Partition, DIC та інші.

Значення для параметрів мінімальна підтримка і мінімальна достовірність вибираються таким чином, щоб обмежити кількість знайдених правил. Якщо підтримка має велике значення, то алгоритми знаходитимуть правила, добре відомі аналітикам або настільки очевидні, що немає ніякого сенсу проводити такий аналіз. З іншого боку, низьке значення підтримки веде до генерації величезної кількості правил, що, звичайно, вимагає істотних обчислювальних ресурсів. Тим не менше, більшість цікавих правил знаходиться саме при низькому значенні порогу підтримки. Хоча занадто низьке значення підтримки веде до генерації статистично необґрунтованих правил.

Пошук асоціативних правил зовсім не тривіальна задача, як може здатися на перший погляд. Одна з проблем — алгоритмічна складність при знаходженні наборів елементів, що часто зустрічаються, оскільки із зростанням числа елементів в I ($|I|$) експоненціально зростає число потенційних наборів елементів.

3.1.2. Узагальнені асоціативні правила

Під час пошуку асоціативних правил ми припускали, що всі аналізовані елементи однорідні. Повертаючись до аналізу ринкового кошика, це товари, що мають абсолютно однакові атрибути, за винятком назви. Однак не складе великих труднощів доповнити транзакцію інформацією про те, в яку товарну групу входить товар і побудувати ієрархію товарів. Наведемо приклад такого *угрупування* (таксономії) у вигляді ієрархічної моделі на рисунку 3.1.2.1.

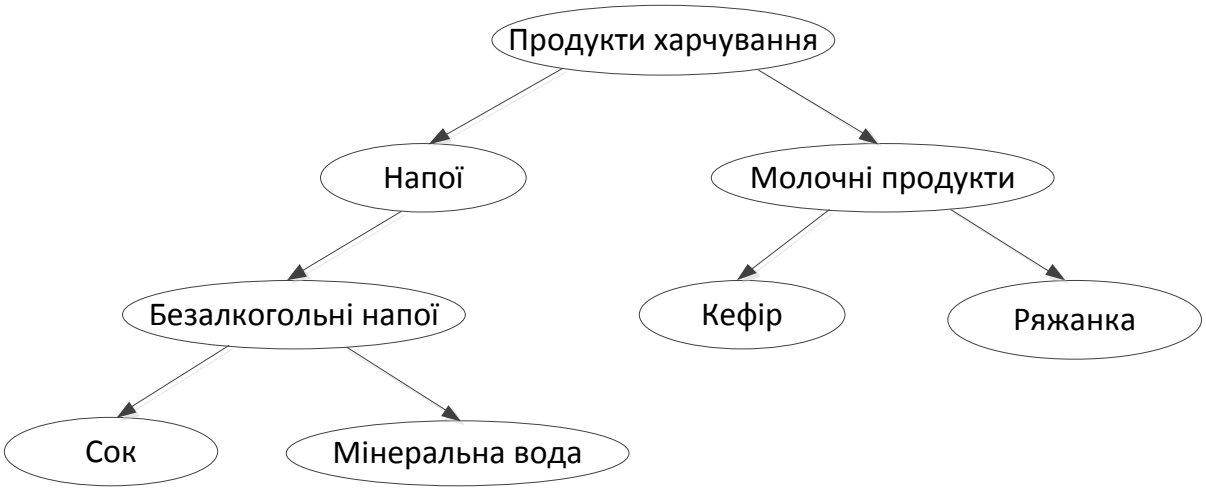


Рисунок 3.1.2.1. Ієрархічна модель

Нехай нам дана база транзакцій D і відомо в які групи (таксони) входять елементи. Тоді можна витягувати з даних правила, що зв'язують групи з групами, окремі елементи з групами і т.д.

Наприклад, якщо Покупець купив товар з групи «Безалкогольні напої», то він купить і товар з групи «Молочні продукти» або «Сок». Ці правила носять назву *узагальнених асоціативних правил*.

Загальним асоціативним правилом (англ. Generalized Association Rules) називається імплікація $X \Rightarrow Y$, де $X \subset I$, $Y \subset I$ і $X \cap Y = \varnothing$ і жодний з елементів, що входять в набір Y не є предком жодного елемента, що входять в X. Підтримка і достовірність рахуються так само, як і у випадку асоціативних правил.

Введення додаткової інформації про угрупування елементів у вигляді ієрархії дає наступні переваги.

- Це допомагає встановити асоціативні правила не тільки між окремими елементами, а й між різними рівнями ієрархії (групами).

- Окремі елементи можуть мати недостатню підтримку, але в цілому група може задовольняти порогу minsupport.

Для знаходження таких правил можна використовувати будь-який з вищеназваних алгоритмів. Для цього кожну транзакцію потрібно доповнити всіма предками кожного елемента, що входить в транзакцію. Однак, застосування «в лоб» цих алгоритмів неминуче призведе до наступних проблем:

1. елементи на верхніх рівнях ієрархії прагнуть до значно більших значень підтримки у порівнянні з елементами на нижніх рівнях;
2. з додаванням в транзакції груп збільшилася кількість атрибутів і відповідно розмірність вхідного простору; це ускладнює завдання, а також веде до генерації більшої кількості правил;
3. поява надлишкових правил, що суперечать визначенню узагальненого асоціативного правила, наприклад, «Сок»-«Прохолодні напої». Очевидно, що практична цінність такого "відкриття" нульова при 100% достовірності; отже, потрібні спеціальні оператори, що видаляють подібні надмірні правила.

Для знаходження узагальнених асоціативних правил бажано використання спеціалізованого алгоритму, який усуває вищеописані проблеми і до того ж працює в 2-5 разів швидше, ніж стандартний APriori.

Групувати елементи можна не тільки по входженню в певну товарну групу, а й за іншими характеристиками, наприклад за ціною (дешево, дорого), бренду і т.д.

3.1.3. Чисельні асоціативні правила

Задача пошуку асоціативних правил істотно спрощена. По суті все зводилося до того, присутній в транзакції елемент чи ні. Тобто якщо розглядати випадок ринкової корзини, то ми розглядали два стани: куплено товар чи ні, проігнорувавши, наприклад, інформацію про те, скільки було куплено, хто купив, характеристики покупця і т.д. І можна сказати, що розглядали «булеві» асоціативні правила. Якщо взяти будь-яку базу даних, кожна транзакція складається з різних типів даних: числових, категоріальних і т.д. Для обробки таких записів і витягання чисельних асоціативних правил був запропонований алгоритм пошуку.

Приклад чисельного асоціативного правила (англ. Quantitative Association Rules): [Вік: 30-35] і [Сімейний стан: одружений] [Місячний дохід: 1000-1500 тугриків].

Крім описаних вище асоціативних правил існують непрямі асоціативні правила, асоціативні правила с запереченням, тимчасові асоціативні правила для подій пов'язаних у часі та інші.

Як було сказано, завдання пошуку асоціативних правил вперше була представлена для аналізу ринкової кошика. Асоціативні правила ефективно використовуються в сегментації покупців по поведінці при покупках, аналізі переваг клієнтів, плануванні розташування товарів в супермаркетах, крос-маркетингу, адресній розсилці. Однак, сфера застосування цих алгоритмів не обмежується лише однією торгівлею. Їх також успішно застосовують і в інших областях: медицині, для аналізу відвідувань веб-сторінок (англ. Web Mining), для аналізу тексту (англ. Text Mining), для аналізу даних по перепису населення, в аналізі і прогнозуванні збоїв телекомунікаційного устаткування і т.д.

3.2. Алгоритм Apriori та його різновиди

Для того, щоб було можливо застосувати алгоритм Apriori, необхідно провести попередню обробку даних:

- привести всі дані до бінарного вигляду (табл. 3.2.1 та 3.2.2);
- змінити структуру даних.

Розглянемо таблицю 3.2.1. Кількість стовпців в таблиці дорівнює кількості елементів, присутніх у множині транзакцій *D*. Кожен запис відповідає транзакції, де у відповідному стовпці стоїть 1, якщо елемент присутній в транзакції, і 0 в іншому випадку.

Зауважимо, що початковий вигляд таблиці може бути відмінним від наведеного в таблиці 3.2.1. Головне, щоб дані були перетворені до нормалізованого вигляду, інакше алгоритм не застосовується.

Таблиця 3.2.1. Звичайний вигляд бази даних транзакцій

Номер транзакції	*Назва елементу*	*Кількість*
1001	A	2
1001	D	3
1001	E	1
1002	A	2
1002	F	1
1003	B	2
1003	A	2
1003	C	2
…	…	…

Таблиця 3.2.2. Нормалізований 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	…
1003	1	0	0	0	0	1	0	0	0	0	…
1003	1	1	1	0	0	0	0	0	1	0	…

Отже, дані перетворені (табл. 3.2.2), тепер можна приступити до опису самого алгоритму. Як було сказано, такі алгоритми працюють в два етапи, не є винятком і розглянутий нами алгоритм Apriori. На першому кроці необхідно знайти набори елементів, що часто зустрічаються, а потім, на другому, витягти з них правила. Кількість елементів у наборі будемо називати розміром набору, а набір, що складається з *k* елементів — k-елементним набором.

3.2.1. Властивість анти-монотонності

Виявлення наборів елементів, які часто зустрічаються — операція, що вимагає багато обчислювальних ресурсів і, відповідно, часу. Примітивний підхід до вирішення даного завдання — простий перебір всіх можливих наборів елементів. Це $O(2|I|)$ операцій, де $|I|$ — кількість елементів. Apriori використовує одну з властивостей підтримки, що говорить: підтримка будь-якого набору елементів не може перевищувати мінімальної підтримки будь-якої з його підмножин. Наприклад, підтримка 3-елементного набору {Хліб, Масло, Молоко} буде завжди менше або дорівнює підтримці 2-елементних наборів {Хліб, Масло}, {Хліб, Молоко}, {Масло, Молоко}. Справа в тому, що будь-яка транзакція, яка містить {Хліб, Масло, Молоко}, також повинна містити {Хліб, Масло}, {Хліб, Молоко}, {Масло, Молоко}, причому зворотне не вірно.

Ця властивість носить назву *анти-монотонності* і служить для зниження розмірності простору пошуку. Не май ми в наявності такої властивості, знаходження багатоелементних наборів було б практично нездійсненним завданням у зв'язку з ростом обчислень.

Властивості анти-монотонності можна дати і інше формулювання: зі зростанням розміру набору елементів підтримка зменшується, або залишається такою ж. З усього вищесказаного випливає, що *k*-елементний набір буде часто зустрічається тоді і тільки тоді, коли всі його (*k*-1)-елементні підмножини будуть часто зустрічаються.

Всі можливі набори елементів з I можна представити у вигляді решітки, що починається з порожнього множини, потім на 1 рівні 1-елементні набори, на 2-му — 2-елементні і т.д. На k рівні представлені *k*-елементні набори, пов'язані з усіма своїми (*k*-1)-елементними підмножинами.

Проаналізуємо рисунок 3.2.1.1, що ілюструє набір елементів I — {A, B, C, D}. Припустимо, що набір з елементів {A,

B} має підтримку нижче заданого порогу і, відповідно, не часто зустрічається. Тоді, відповідно до властивості анти-монотонності, всі його супермножини також не є такими, що часто зустрічаються, і відкидаються. Вся ця гілка, починаючи з {A, B}, виділена фоном. Використання цієї евристики дозволяє істотно скоротити простір пошуку.

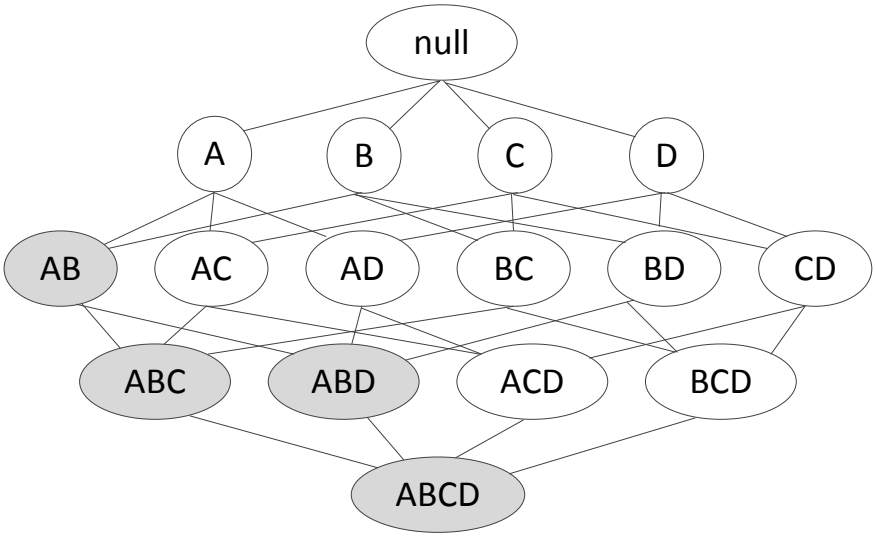


Рисунок 3.2.1.1. Набір елементів I — {A, B, C, D}.

3.2.2. Реалізація алгоритму

На першому кроці алгоритму підраховуються 1-елементні набори, що часто зустрічаються. Для цього необхідно пройтися по всьому набору даних і підрахувати для них підтримку, тобто скільки разів зустрічається в базі.

Наступні кроки будуть складатися з двох частин: генерації наборів елементів, що потенційно часто зустрічаються (їх називають кандидатами) і підрахунку підтримки для кандидатів.

Описаний вище алгоритм можна записати у вигляді наступного псевдо-коду:

```
F1 = { 1-елементні набори, що часто зустрічаються}
для (k=2; Fk-1 <> ∅; k++) {
 Ck = Apriorigen(Fk-1) // генерація кандидатів
 для всіх транзакцій t ∈ T {
      Ct = subset(Ck, t) // видалення  надлишкових правил
      для всіх кандидатів c ∈ Ct
      c.count ++
 }
```

```
 Fk = { c ∈ Ck | c.count >= minsupport}
 // відбір кандидатів
}
Результат ∪ Fk
```

Опишемо функцію генерації кандидатів. На цей раз немає ніякої необхідності знову звертатися до бази даних. Для того, щоб отримати k-елементні набори, скористаємося (k-1)-елементними наборами, які були визначені на попередньому кроці і часто зустрічаються.

Згадаймо, що наш початковий набір зберігається у впорядкованому вигляді. Генерація кандидатів також буде складатися з двох кроків.

1. Об'єднання. Кожен кандидат Ck буде формуватися шляхом розширення набору розміру (k-1) додаванням елемента з іншого (k-1) — елементного набору.

   Наведемо алгоритм цієї функції Apriorigen у вигляді невеликого SQL-подібного запиту.

   ```
   insert into Ck
       select p.item1, p.item2, …, p.itemk-1, q.itemk-1
       From Fk-1 p, Fk-1 q
       where p.item1= q.item1, p.item2 = q.item2, … ,
       p.itemk-2 = q.itemk-2, p.itemk-1 < q.itemk-1
   ```

2. Видалення надлишкових правил. На підставі властивості анти-монотонності, слід видалити всі набори с C_k якщо хоча б одне з його (k-1) підмножин не часто зустрічається.

Після генерації кандидатів наступним завданням є підрахунок підтримки для кожного кандидата. Очевидно, що кількість кандидатів може бути дуже великою і потрібен ефективний спосіб підрахунку. Найбільш тривіальний спосіб — порівняти кожну транзакцію з кожним кандидатом. Але це далеко не найкраще рішення. Набагато швидше і ефективніше використовувати підхід, що базується на зберіганні кандидатів у хеш-дереві. Внутрішні вузли дерева містять хеш-таблиці з покажчиками на нащадків, а листя — на кандидатів. Це дерево нам стане в нагоді для швидкого підрахунку підтримки для кандидатів.

Хеш-дерево будується кожен раз, коли формуються кандидати. Спочатку дерево складається тільки з кореня, який є

листом, і не містить ніяких кандидатів-наборів. Кожен раз коли формується новий кандидат, він заноситься в корінь дерева і так до тих пір, поки кількість кандидатів в корені-листі не перевищить певного порогу. Як тільки кількість кандидатів стає більше порога, корінь перетворюється в хеш-таблицю, тобто стає внутрішнім вузлом, і для нього створюються нащадки-листя. І всі приклади розподіляються по вузлах-нащадкам згідно хеш-значенням елементів, що входять в набір, і т.д. Кожен новий кандидат хешується на внутрішніх вузлах, поки він не досягне першого вузла-листа, де він і буде зберігатися, поки кількість наборів знову ж таки не перевищить порогу.

Хеш-дерево з кандидатами-наборами побудовано, тепер, використовуючи хеш-дерево, легко підрахувати підтримку для кожного кандидата. Для цього потрібно "пропустити" кожну транзакцію через дерево і збільшити лічильники для тих кандидатів, чиї елементи також містяться і в транзакції, тобто $C_k \cap T_i = C_k$. На кореневому рівні хеш-функція застосовується до кожного елементу з транзакції. Далі, на другому рівні, хеш-функція застосовується до других елементам і т.д. На k-рівні хешується k-елемент. І так до тих пір, поки не досягнемо листа. Якщо кандидат, що зберігається в листі, є підмножиною розглянутої транзакції, тоді збільшуємо лічильник підтримки цього кандидата на одиницю.

Після того, як кожна транзакція з вихідного набору даних "пропущена" через дерево, можна перевірити чи задовольняють значення підтримки кандидатів мінімального порогу. Кандидати, для яких ця умова виконується, переносяться в розряд часто зустрічаються. Крім того, слід запам'ятати і підтримку набору, вона нам стане в нагоді при витяганні правил. Ці ж дії застосовуються для знаходження (k+1)-елементних наборів і т.д.

Після того як знайдені всі часто зустрічаються набори елементів, можна приступити безпосередньо до генерації правил.

Витяг правил — менш трудомістке завдання. По-перше, для підрахунку достовірності правила досить знати підтримку самого набору і множини, що лежить в умові правила. Наприклад, є набір {A, B, C}, що часто зустрічається, і потрібно підрахувати достовірність для правила $AB \Rightarrow C$. Підтримка самого набору нам відома, але і його множина {A, B}, що лежить в умові правила, також є такою, що часто зустрічається, в силу властивості анти-монотонності, і значить його підтримка нам

відома. Тоді ми легко зможемо підрахувати достовірність. Це позбавляє нас від небажаного перегляду бази транзакцій, який потрібен в тому випадку якби це підтримка була невідома.

Щоб витягти правило з часто зустрічається набору F, слід знайти всі його не порожні підмножини. І для кожної підмножини s ми зможемо сформулювати правило $s \Rightarrow (F — s)$, якщо достовірність правила $conf(s \Rightarrow (F — s)) = supp(F)/supp(s)$ не менше порога minconf.

Зауважимо, що чисельник залишається постійним. Тоді достовірність має мінімальне значення, якщо знаменник має максимальне значення, а це відбувається в тому випадку, коли в умові правила є набір, що складається з одного елемента. Всі супермножини даної множини мають меншу або рівну підтримку і, відповідно, більше значення достовірності. Ця властивість може бути використано при витяганні правил. Якщо ми почнемо витягувати правила, розглядаючи спочатку тільки один елемент в умові правила, і це правило має необхідну підтримку, тоді всі правила, де в умові стоять супермножини цього елемента, також мають значення достовірності вище заданого порогу. Наприклад, якщо правило $A \Rightarrow BCDE$ задовольняє мінімального порогу достовірності minconf, тоді $AB \Rightarrow CDE$ також задовольняє. Для того, щоб витягти всі правила використовується рекурсивна процедура. Важливе зауваження: будь-яке правило, складене з часто зустрічається набору, повинно містити всі елементи набору. Наприклад, якщо набір складається з елементів {A, B, C}, то правило $A \Rightarrow B$ не повинно розглядатися.

Приклад 3.1. На основі даних, представлених в таблиці 3.2.2.1, треба знайти закономірності між покупками.

Присвоїмо значенням товарів змінні: хліб = a, молоко = b, печиво = c, сметана = d, колбаса = e, цукерки = f (табл. 3.2.2.2).

Розглянемо набір товарів (itemset), що включає в себе, наприклад {хліб, молоко, печиво}: abc = {a, b, c}. Цей набір зустрічається і базі три рази, тобто підтримка цього набору рівна 3: sup(abc)=3. При мінімальному рівні підтримки minsupport=3 цей набор є таким, що часто зустрічається.

Для даного набору товару підтримка у процентному відношенні рівна 50%: SUP(abc) = (3/6)*100% = 50%. Таким чином, набір представляє інтерес.

Розглянемо правило «З покупки молока слідує покупка печива». Підтримка даного правила дорівнює 3, або 50%.

Таблиця 3.2.2.1. Транзакційна база даних

TID	*Товари*
100	Хліб, молоко, печиво
200	Молоко, сметана
300	Молоко, хліб, сметана, печиво
400	Колбаса, сметана
500	Хліб, молоко, печиво, сметана
600	Цукерки

Таблица 3.2.2.2. Набори товарів, що часто зустрічаються

TID	*Набір*
100	a, b, c
200	b, d
300	b, a, d, c
400	e, d
500	a, b, c, d
00	f

Число транзакцій, що містить молоко, рівно 4, а число транзакцій, що містить печиво — 3. Достовірність правила — ¾ *100% , тобто 75 %.

Робота алгоритму Apriori для даного прикладу зображена на рисунку 3.2.2.3.

На першому етапі відбувається формування одноелементних кандидатів. Далі алгоритм підраховує підтримку одноелементних наборів. Набори з рівнем підтримки менше встановленого, тобто 3, відсікаються. В нашому прикладі це набори e і f, що маються підтримку 1. Решта наборів вважаються такими, що часто зустрічаються, це набори a, b, c, d.

Далі відбувається формування двохелементних кандидатів, підрахунок їх підтримки і відсічення наборів з рівнем підтримки менше 3. Ті товари, що залишились, вважаються такими, що часто зустрічаються: ab, ac, bd.

На останньому етапі алгоритм формує трьохелементні набори товарів abc, abd, bcd, acd. Після цього алгоритм відсікає кандидатів, що не зможуть стати такими, що часто зустрічаються, на основі інформації про відсічених кандидатів на минулих етапах.

Так набори ad, bc, cd були відкинуті, тому алгоритм не розглядає abd, bcd, acd.

Висновки до розділу 3

Асоціативним правилом називається імплікація $X \Rightarrow Y$, де $X \subset I$, $Y \subset I$ і $X \cap Y = \varnothing$. Правило $X \Rightarrow Y$ *має підтримку* s (support), якщо s% транзакцій з D містять $X \cup Y$, $supp(X \Rightarrow Y) = supp(X \cup Y)$. *Достовірність* правила вказує, яка ймовірність того, що з X слідує Y. Правило $X \Rightarrow Y$ справедливе з достовірність (confidence) c, якщо c% транзакцій з D, що містять X, також містять Y, $conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$. Асоціативні правила дозволяють знаходити закономірності між пов'язаними подіями.

Алгоритми пошуку асоціативних правил призначені для знаходження всіх правил $X \Rightarrow Y$, причому підтримка і достовірність цих правил повинні бути вище деяких наперед визначених порогів, званих відповідно *мінімальною підтримкою* (minsupport) і *мінімальної достовірністю* (minconfidence).

Рисунок 3.2.2.3. Етапи агоритму Apriori

Задача знаходження асоціативних правил розбивається на дві під задачі:

1. *знаходження всіх наборів елементів*, які задовольняють порогу minsupport;
2. *генерація правил з наборів елементів*, знайдених згідно п.1. з достовірністю, що задовольняє порогу minconfidence.

Загальним асоціативним правилом (Generalized Association Rules) називається імплікація $X \Rightarrow Y$, де $X \subset I$, $Y \subset I$ і $X \cap Y = \varnothing$ і де жодний з елементів, що входять в набір Y не є предком жодного елемента, що входять в X.

Властивість анті-монотонності: підтримка будь-якого набору елементів не може перевищувати мінімальної підтримки будь-якої з його підмножин.

Питання до розділу 3

1. Сформулюйте задачу пошуку асоціативних правил.
2. Як ще називають задачу пошуку асоціативних правил?
3. Що таке асоціативне правило?
4. Як визначається достовірність та підтримка асоціативного правила?
5. Що таке узагальнені асоціативні правила?
6. Які типи асоціативних правил ви знаєте?
7. Дайте означення узагальненим та чисельним асоціативним правилам.
8. Які кроки в алгоритмі Apriori?
9. В чому заключається властивість анти-монотонності, що використовується алгоритмом Apriori?
10. Як застосовується властивість анти-монотонності в алгоритмі Apriori?
11. Які варіації алгоритму Apriori ви знаєте?

Вправи до розділу 3

Використавши алгоритм Apriori, побудуйте асоціативні правила для даних, поданих на рисунку 3.1.a-d.

a

1	0 1 3 4 7
2	0 1 2
3	0 3 4 5 6
4	0 1 2 3 4 5 7
5	1 2 3 4 5
6	1 3 4
7	3 7 9
8	0 1 2 3 7
9	1 2 3 6 7
10	0 7 8

b

1	0 1 2 5 7 9
2	1 2 7 8
3	0 5 9 8 9
4	1 2 4 5 7
5	0 1 2 5 9
6	0 1 2 9
7	0 3 4 5 8 9
8	3 4 1 2 7 8
9	5 6 7
10	0 5 6 8 9

c

1	0 1 3 4 7
2	0 1 2
3	0 3 4 5 6
4	0 1 2 3 4 5 7
5	1 2 3 4 5
6	1 3 4
7	3 7 9
8	0 1 2 3 7
9	1 2 3 6 7
10	0 7 8

d

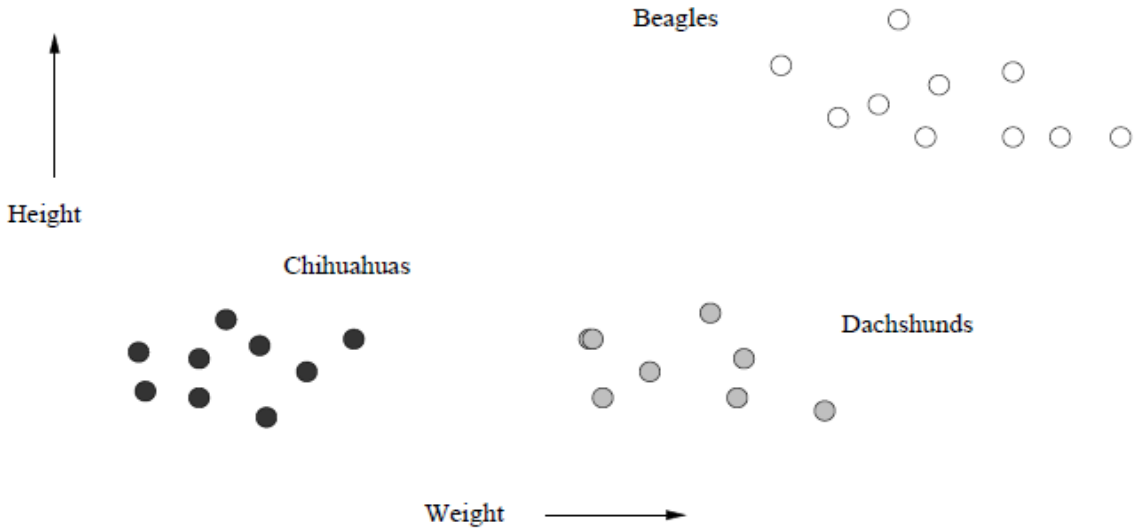
1	0 1 3 4 7
2	0 1 2
3	0 3 4 5 6
4	0 1 2 3 4 5 7
5	1 2 3 4 5
6	1 3 4
7	3 7 9
8	0 1 2 3 7
9	1 2 3 6 7
10	0 7 8

Рисунок 3.1. Вхідні дані для вправи

Розділ 4. Кластеризація даних методами Data Mining

Під кластеризацією зазвичай мають на увазі процес перевірки набору «точок» та групування точок у «кластери» згідно якоїсь міри довжини. Суть полягає у тому, що точки з одного кластера матимуть невелику відстань одна з одною, а точки з різних кластерів навпаки — велику. Під «точками» мається на увазі *n*-вимірний вектор характеристик. Приклад кластерів подано на рисунку 4.1.

Даний розділ присвячений методам знаходження кластерів у даних. Особлива увага приділяється методам обробки великих об'ємів даних та/або випадкам багатомірного або неевклідового простору.



Малюнок 4.1: Висоти та вага собак трьох різних порід

4.1. Кластерні технології

4.1.1. Точки, простори та відстані

Придатний до кластеризації набір даних представляє собою множину точок, які належать деякому простору. Простором вважаємо універсальну множину, з якої беруться точки нашого набору даних (наприклад, евклідовий простір має багато важливих властивостей, які можуть бути корисними при кластеризації; зокрема, точки евклідового простору є векторами дійсних чисел). Довжина вектора визначається кількістю вимірів простору. Компонентами вектора є координати відповідних точок.

Усі простори, для яких може бути виконана кластеризація,

мають міру довжини, яка задає відстань між двома точками у просторі. Загальна Евклідова відстань (квадратній корінь сум квадратів різниці між координатами точок у кожному вимірі) працює для всіх Евклідових просторі, як і інші методи виміру відстаней у Евклідових просторах: манхетанська (сума магнітуд різниць у кожному вимірі) та L∞-відстань (максимальна магнітуда різниць в будь-якому просторі).

Приклад 4.1. Класичне використання кластеризації часто включають в себе маловимірні Евклідові простори. Наприклад, рисунок 4.1 демонструє висоту та вагу собак різних порід. Без знання про породи, просто поглянувши на діаграму, можна бачимо, що собаки підпадають під три кластери, і ці кластери відповідають трьом породам. З малим обсягом даних будь-який алгоритм кластеризації буде будувати коректні кластери, а проста побудова графіку буде достатньо наглядною.

Сучасні задачі кластеризації вже на нестільки прості. Вони можуть включати Евклідові простори з дуже великою кількістю вимірів або навіть неевклідові простори. Наприклад кластери документів за темами, які основані на частоті входження, незвичних словах. Кластер «кіноманів» за типами фільмів, що їм подобаються.

Розглянемо основні вимоги для функції відстані між двома точками.

1. Відстані повинні бути додатніми.
2. Нульовою може буті лише відстань від будь-якої точки до самої себе.
3. Відстань симетрична: відстань від A до B співпадає з відстанню від B до A.
4. Порядок підрахунку відстаней між точками немає значення.
5. Для міра відстані виконується трикутна нерівність: відстань від A до B та до C не менша за відстань від A до C напряму.

4.1.2. Стратегії кластеризації

Алгоритми кластеризації можна розділити на дві групи згідно двох фундаментально різних стратегій.

1. Ієрархічні або агломеративні алгоритми стартують кожен із власної точки у кластері. Кластери об’єднуються за близькістю використовуючи одне з

багатьох визначень «близькості». Об’єднання зупиняється коли подальше об’єднання приводить до небажаних результатів. Наприклад, ми можемо зупинитися, коли маємо заздалегідь визначену кількість кластерів, або ми можемо використовувати міру компактності кластерів і відмовитись від побудови кластера шляхом об’єднання двох менших кластерів, якщо результуючий кластер має точки на дуже великій відстані одна від одної.

2. Інший клас алгоритмів включає в себе етап ініціалізації початкових точок, що визначають кластери. Точки задаються у будь-якому порядку і кожна приписується до кластера, до якого вона найбільше підходить. Цей процес зазвичай займає коротку фазу, в якій задаються початкові кластери. Деякі види алгоритмів також дозволяють об’єднання або розділення кластерів, або дозволять позбутися точок, які знаходяться на віддаленні від усіх кластерів.

Алгоритми кластеризації також можна розділити за наступними критеріями.

1. Працює алгоритму у Евклідовому просторі або в довільній системі виміру. В евклідовому просторі можна сумувати набір точок за їх центроїдою — середнім арифметичним точок. В неевклідовому просторі не існує визначення центроїди і потрібні інші шляхи сумування кластерів.
2. Використовує алгоритм лише первинну пам'ять у випадку малого обсягу даних, або йому буде потрібна вторинна пам'ять, якщо даних забагато. Алгоритми для великих обсягів даних часто використовують скорочення, оскільки неможливо проаналізувати всі пари точок у первинній пам'яті. Також важливо сумувати кластери в основній пам'яті, оскільки утримувати усі точки усіх кластерів в первинній пам'яті неможливо.

4.1.3. Багатовимірні евклідові простори та «прокляттям вимірності».

Багатовимірні евклідові простори мають декілька неявних властивостей, які інколи називають «прокляттям вимірності». Неевклідові простори зазвичай також мають ці властивості. Одним з втілень «прокляття» в багатовимірних просторах

полягає у тому, що майже всі пари точок рівновіддалені одна від одної. Іншим проявом є те, що майже будь-які два вектори є практічно ортогональними.

4.1.3.1. Розподіл відстаней у багатовимірних просторах

Нехай існує d-вимірний евклідів простір. Припустимо, що ми вибираємо n будь-яких точок в одиночному кубі, іншими словами точки $[x_1, x_2, ..., x_d]$, де кожен x_i знаходиться у проміжку від 0 до 1. Якщо $d = 1$, то ми розміщуємо випадкові точки на лінії довжиною 1. Ми очікуємо, що деякі пари точок будуть дуже близько, іншими словами, ці точки будуть послідовними на лінії. Ми також очікуємо, що деякі точки будуть віддалені — точки, що знаходяться на протилежних кінцях лінії. Середня відстань між парами точок становить 1/3.

Припустимо, що d достатньо велике. Евклідова відстань між двома випадковими точками $[x_1, x_2, ..., x_d]$, та $[y_1, y_2, ..., y_d]$, становить:

$$\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

Тут кожен x_i та y_i є випадковою змінною, обраною на проміжку від 0 до 1. Оскільки d велике, ми можемо очікувати, що для деяких i $|x_i - y_i|$ буде близьким до 1. Це встановлює нижню межу 1 на відстань між майже будь-якими двома випадковими точками. Більш того, точніший аргумент може накласти точнішу нижню межу на відстані між усіма, крім зникаючи малої кількості пар точок. Однак, найбільша відстань між двома точками становить $\sqrt{d}$, звісно можна заперечити, що така ж мала кількість пар точок не буде мати таку верхню межу. Більш того, більшість точок, буде мати відстань, близьку до середньої.

Якщо ж не існує жодної пари близьких точок, то побудова кластера стає набагато складнішою. Не існує чіткого обґрунтування для групування пар точок тим або іншим чином. Звісно, дані можуть бути не випадковим, і можуть існувати корисні кластери навіть у багатовимірних просторах. Однак, аргумент випадкових даних передбачає, що знаходження цих кластерів серед великої кількості точок на майже однаковій відстані є дуже складним.

4.1.3.2. Кути між векторами

Знову припустимо, що ми маємо три випадкові точки A, B і C в d-вимірному просторі, де d достатньо велике. Припустимо, що точки не лежать в одиничному кубі; вони можуть бути будь-де у просторі. Чому дорівнює кут ABC ? Ми можемо припустити, що A це точка $[x_1, x_2, ..., x_d]$ і C це точка$[y_1, y_2, ..., y_d]$, а B — основа. Кут ABC це скалярний добуток A та C розділений на добуток довжини векторів A та C . Тобто косинус:

$$\frac{\sum_{i=0}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}}$$

З ростом d знаменник зростає лінійно в d, але знаменник є сумою випадкових значень, які можуть бути як позитивними так і негативними. Тому очікуване значення знаменника — 0, і, з ростом d, його стандартне відхилення зростає лише до $\sqrt{d}$. Тому для великих d косинус кута між будь-якими двома векторами майже напевно буде близьким до 0, що означає, що кут буде близьким до 90 градусів.

Важливим наслідком з ортогональності випадкових векторів є те, що для трьох випадкових точок A, B і C, знаючи відстань від A до B дорівнює d_1, а відстань від B до C — d_2, ми можемо припустити, що відстань від A до C приблизно дорівнює $\sqrt{d_1^2 + d_2^2}$. Це правило не спрацьовує, навіть приблизно, якщо вимірів небагато. В крайньому випадку, якщо d = 1 тоді відстань від A до C обов'язково буде дорівнювати d_1+d_2, якщо A та C знаходяться по різни боки від B, або $|d_1 + d_2|$, якщо з одного боку.

4.2. Постановка задачі кластеризації

Кластеризація — це автоматичне розбиття елементів деякої множини на групи в залежності від їх подібності. Елементами множини може бути все, що завгодно, наприклад, дані або вектори характеристик. Власне групи прийнято називати кластерами.

Кластеризація (об'єднання в групи схожих об'єктів) є однією із фундаментальних задач Data Mining. Список прикладних областей, де вона застосовується, широкий: сегментація зображень, маркетинг, боротьба з шахрайством, прогнозування, аналіз текстів тощо. Задачу кластеризації в тому чи іншому вигляді формулювали в таких наукових напрямках, як

статистика, розпізнавання образів, оптимізація, машинне навчання. Звідси різноманіття синонімів поняттю кластер — клас, таксон, згущення.

Також кластеризація є важливою формою абстракції даних.

Крім того, кластеризація є розділом сучасної теоретичної інформатики, що бурхливо розвивається, і в цій області можна отримати ряд цікавих дослідницьких результатів.

Кластеризація розбиває множину об'єктів на групи, які визначаються лише її результатом. Класифікація відносить кожен об'єкт до однієї із заздалегідь визначених груп.

Кластеризація включає в себе наступні етапи (рисунок 4.2).

1. Виділення характеристик
2. Визначення метрики
3. Розбиття об'єктів на групи
4. Представлення результатів

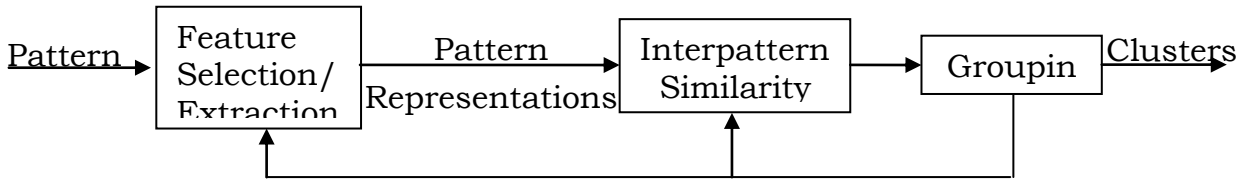


Рисунок 4.2 Етапи кластеризації

Для початку необхідно *обрати властивості*, які характеризують наші об'єкти. Далі — спробувати зменшити розмірність простору характеристичних векторів, тобто виділити найбільш вагомі властивості об'єктів. Зменшення розмірності пришвидшує процес кластеризації, а в деяких випадках дозволяє візуально оцінювати результати.

Виділені характеристики потрібно нормалізувати.

Далі всі об'єкти представляються у вигляді характеристичних векторів. Надалі будемо ототожнювати об'єкт з цього характеристичним вектором.

Наступним етапом є *вибір метрики*, за якою будемо визначати схожість об'єктів.

Метрика обирається в залежності від:

1. Простору, в я кому розташовані об'єкти
2. Неявних характеристик кластерів

Наприклад, якщо всі координати об'єкта неперервні і дійсні, а кластери мають представляти собою дещо на кшталт гіперсфер, то використовується класична метрика Евкліда:

$$d_2(x_i, x_j) = \sqrt{\sum_{k=1}^{d} (x_{ik} - x_{jk})^2} = \left\| x_i - x_j \right\|_2$$

Для категорійних атрибутів поширена міра подібності Чекановського-Серенсена і Жаккара:

$$|t_1 \cap t_2| / |t_1 \cup t_2|$$

При виконанні кластеризації важливо, скільки в результаті має бути побудовано кластерів. Передбачається, що кластеризація має виявити природні локальні згущення об'єктів. Тому число кластерів є параметром, що часто суттєво ускладнює вид алгоритму, якщо воно невідоме; і таким, що суттєво впливає на якість результату, якщо воно відоме.

Проблема вибору числа кластерів зовсім нетривіальна. Достатньо сказати, що для отримання задовільного теоретичного рішення часто вимагається зробити сильне припущення про властивості деякого заздалегідь заданого сімейства розподілів. Але на початку дослідження про дані заздалегідь нічого невідомо, тому алгоритми кластеризації зазвичай будуються як деякий спосіб перебору числа кластерів, і визначення його оптимальних значень в процесі перебору.

Число методів розбиття на множини і кластери достатньо велике. Всі їх можна розділити на ієрархічні та неієрархічні.

В неієрархічних алгоритмах характер їх роботи і умову зупинки треба завчасно регламентувати часто достатньо великим числом параметрів, що іноді важко. Але в таким алгоритмах досягається гнучкість у варіюванні кластеризації і зазвичай визначається числом кластерів.

З іншої сторони, коли об'єкти характеризуються великим числом ознак (параметрів), то важливе значення набуває задача групування ознак. Вхідна інформація знаходиться в квадратній матриці зв'язків ознак.

В ієрархічних алгоритмах фактично відмовляються від визначення числа кластерів, будуючи повне дерево вкладених кластерів (дендрограму). Число кластерів визначається з припущень, що не відносяться до алгоритму. Труднощі таких алгоритмів: вибір близькості кластерів, проблема інверсій індексації, негнучкість, яка зазвичай небажана.

4.2. Ієрархічні алгоритми кластеризації

Розглянемо ієрархічну кластеризацію в Евклідовому просторі. Цей алгоритм може бути використаний тільки для відносно невеликих наборів даних, проте існують покращення. В не-Евклідовому просторі існують деякі додаткові труднощі, пов'язані з ієрархічною кластеризацією. Тому введемо поняття «кластероїду» і метод, згідно якого можна представити кластер без центроїди або середньої точки у кластері.

Будь-який алгоритм ієрархічної кластеризації працює наступним чином. На початку вважаємо, що кожна точка належіть окремому кластеру. На кожному кроці більші кластери утворюються шляхом об'єднання двох менших кластерів. Необхідно наперед визначити наступне:

1. як будуть представлені кластери?
2. як визначити які два кластери об'єднати?
3. коли закінчиться об'єднання кластерів?

Як тільки ми вирішимо ці питання, алгоритм може буди представлений наступним чином:

```
WHILE it is not time to stop DO:
    pick the best two clusters to  merge;
    combine those two clusters into one cluster;
END;
```

4.2.1. Базвий алгоритм ієрархічної кластеризації

Почнемо з припущення, що наш простір — евклідовий. Це дозволяє представити кластер через його центроїду або середнє точок у ньому. Слід зауважити, що у кластері з однією точкою вона і буде центроїдою, тому кластери можуть буди одразу ініціалізовані. Тоді ми можемо використати правило об'єднання, згідно якого відстань між кластерами — це відстань між їхніми центроїдами, і будемо обирати кластери з найменшою відстанню. Існують також і інші шляхи знаходження відстані між кластерами, також пари кластерів можна визначати за іншими критеріями, не тільки за відстанню.

Приклад 4.2.1.1. Розглянемо базову ієрархічну кластеризацію на прикладі обробки даних з рисунку 4.2.1.1. Ці точки знаходяться в 2-вимірному Евклідовому просторі і кожна

точка названа за своїми (*x*, *y*) координатами. Спочатку кожна точка є кластером і центроїдою цього кластера. З усіх пар точок, дві пари лежать найближче одна до одної: (10,5) та (11,4) або (11,4) та (12,3). Кожна має відстань $\sqrt{2}$. Об'єднаємо (11,4) з (12,3). Результат показаний на рисунку 4.2.1.2 включно з центроїдою нового кластера у (11.5, 3.5).

Може здатися, що точка (10,5) об'єднується з новим кластером, оскільки вона близька до (11,4). Але правило відстані вимагає від нас порівняння відстаней між центроїдами, і відстань від (10,5) до центроїди нового кластеру буде становити $1.5\sqrt{2}$, що трохи більше за 2. Тому тепер двома найближчими кластерами є (4,8) та (4,10). Ми об'єднуємо їх у кластер з центроїдою (4,9).

На цьому етапі, дві найближчі центроїди це (10,5) та (11.5,3.5), тому ми об'єднуємо ці кластери. Результатом стає кластер з трьох точок (10,5), (11,4) та (12,3). Центроїд цього кластера — (11,4), що приходиться на одну з точок кластера, але це лише випадковість. Вигляд кластерів показаний на рисунку 4.2.1.3.

Тепер існує кілька центроїд з відстанню $\sqrt{5}$, і вони є найближчими. На рисунку 4.2.1.4 подано результат вибору трьох з них:

1. (6,8) об'єднується з кластером з центроїдою (4,9).
2. (2,2) об'єднується з (3,4).
3. (9,3) об'єднується з триелементним кластером з центроїдою (11,4).

Ми можемо продовжити подальше об'єднання кластерів. Далі ми роздивимось різні стоп-правила.

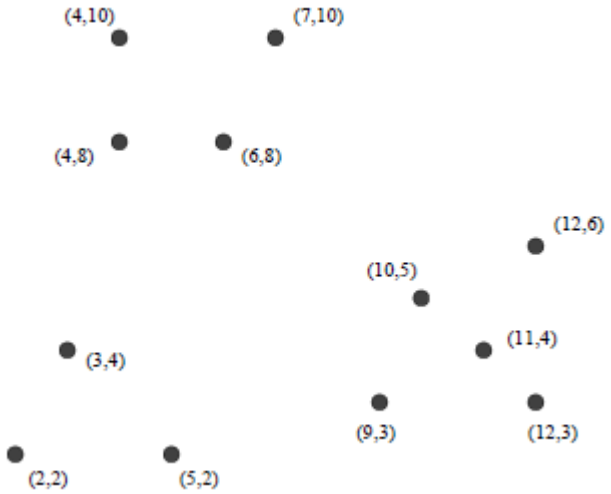


Рисунок 4.2.1.1. Приклад даних

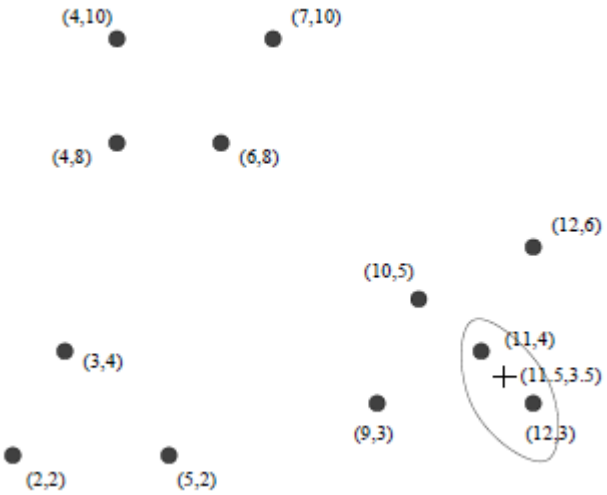


Рисунок 4.2.1.2. Перший крок кластеризації

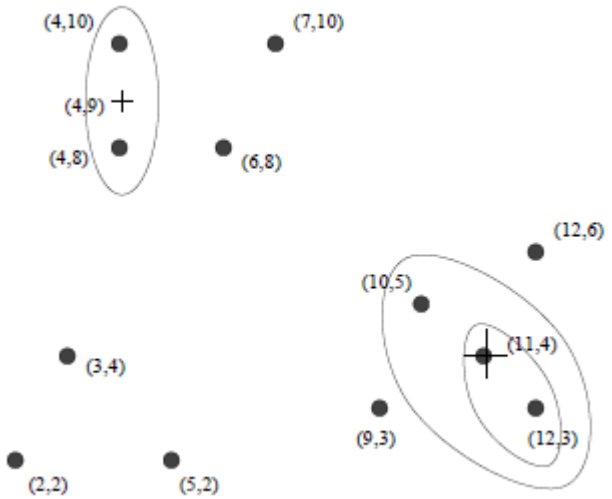


Рисунок 4.2.1.3. Проміжний етап кластеризації

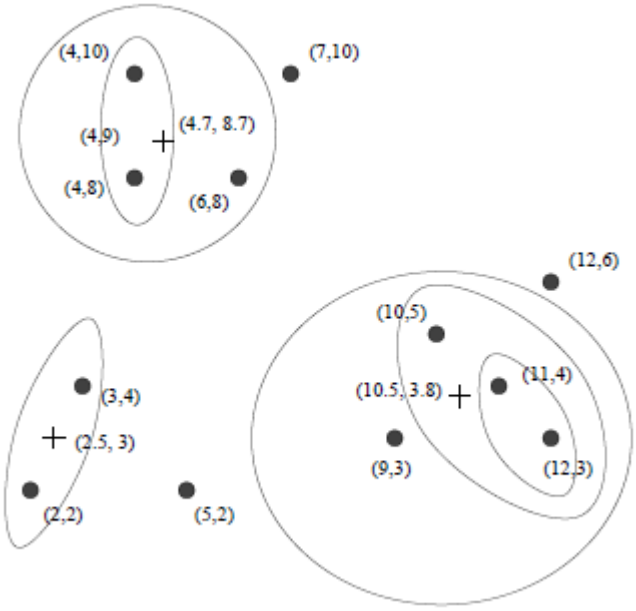


Рисунок 4.2.1.4. Проміжний етап кластеризації

Існує декілька підходів до зупинки процесу кластеризації.

1. Можемо знати скільки кластерів буде у даних. Наприклад, якщо ми знаємо, що дані про собак беруться з чіхуа-хуа, такс та біглів, тоді ми знаємо, що зупинитись слід на трьох кластерах.
2. Можемо зупинитись, коли найкраща комбінація існуючих кластерів не є адекватною. Різні шляхи визначення адекватності ми розглянемо далі, але для прикладу можемо вважати, що будь-який кластер не має середню відстань від центроїди до його точок більшу за якийсь визначений ліміт. Цей підхід має місце тільки якщо ми впевнені, що кластер не розтягнутий на велику відстань.
3. Можемо продовжити кластеризацію, доки ми не отримаємо один кластер. Однак, наявність одного кластера з усіма точками немає сенсу. Скоріш ми отримаємо дерево, яке відображає шлях, за яким точки об'єднувались. Такий результат має сенс в деяких застосуваннях, де точки — це геноми створінь, а відстань між ними — різниці між геномами. Тоді це дерево буде відображати еволюцію цих створінь, що, частіш за все показує виникнення двох видів з одного предка.

Приклад 4.2.1.2. Якщо ми завершимо клестерізацію даних з прикладу 4.2.1.1, то отримаємо дерево зв'язків, подане на рисунку 4.2.1.5. □

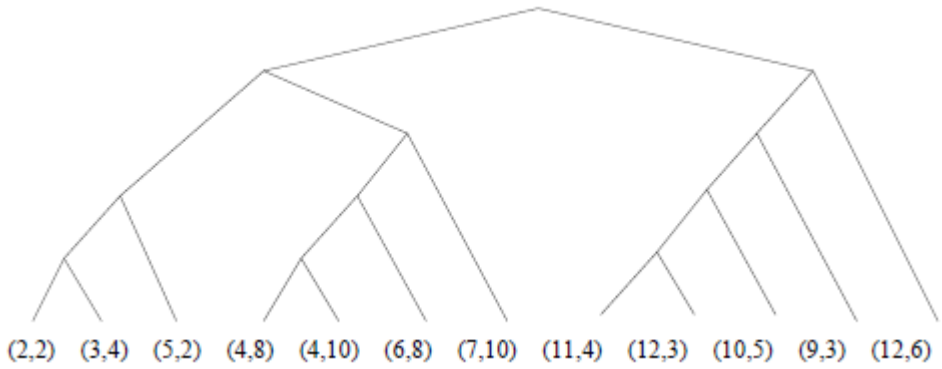


Рисунок 4.2.1.5. Дерево зв'язків даних

4.2.2. Алгоритми Single-link та Complete-link

Результатом роботи ієрархічних алгоритмів є дендограма (ієрархія), що дозволяє розбити множину об'єктів на будь-яку кількість кластерів. Два найбільш популярних алгоритми (будуються «знизу-вверх»).

1. Single-link — на кожному кроці поєднує два кластери з найменшою відстанню між будь-якими двома представниками
2. Complete-link — на кожному кроці поєднує два кластери з найменшою відстанню між двома найбільш віддаленими представниками

Приклад 4.2.2.1. Для даних, зображених на рисунку 4.2.2.1, кроки алгоритми відповідних алгоритмів представлені на рисунку 4.2.2.2.

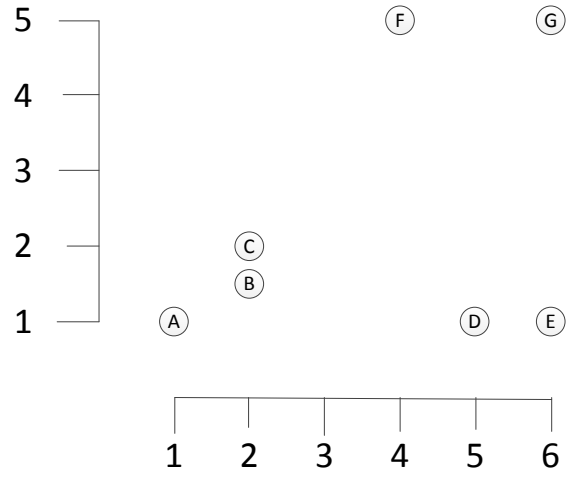
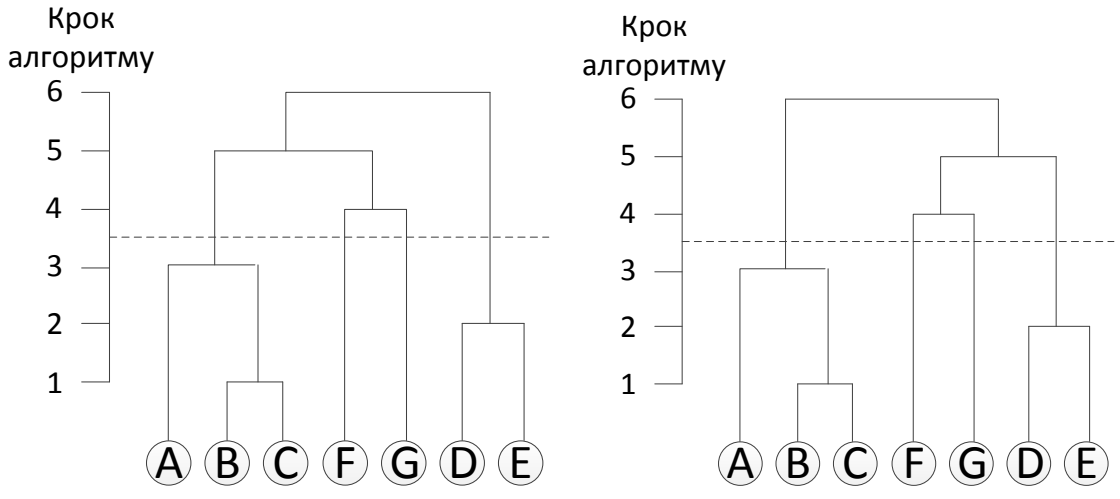


Рисунок 4.2.2.1. Приклад даних



(а) (б)

Рисунок 4.2.2.2. Кроки алгоритму (а) Single-link та (б) Complete-link

Існує декілька методів розрахунку відстаней з використанням старих значень відстаней для поєднаних кластерів, що відрізняються коефіцієнтами в формулі (табл. 4.2.2.1):

$$d_{rs} = \alpha_p d_{ps} + \alpha_q d_{qs} + \beta d_{pq} + \gamma \left| d_{ps} - d_{qs} \right| .$$

Якщо кластери p і q обєднуються в кластер *r* і треба розрахувати відстань від нового кластера до кластера s, застосування того чи ынщшого методу залежить від способа визначення відстані між класами. Ці методи розрізняються значеннями коефіцієнтів α_p, α_q, β, γ.

Таблиця 4.2.2.1. Методи розрахунку відстаней

Назва методу	α_p	α_q	β	γ
Nearest neighbor: відстань між найближчими сусідами, найближчими об'єктами кластерів	1/2	1/2	0	-1/2
Furthest neighbor: відстань між найвіддаленішими об'єктами кластерів	1/2	1/2	0	1/2
Median clustering: центроідний метод, центр визначається як середнє усіх обєктів	1/2	1/2	-1/4	0
Between-groups linkage: середня відстань між кластерами	1/2	1/2	0	0
Within-groups linkage: середня відстань між усіма обєктами кластерів з врахуванням відстаней всередині кластерів	$\frac{k_p}{k_p + k_q}$	$\frac{k_p}{k_p + k_q}$	0	0
Centroid clustering: відстань між центрами кластерів	$\frac{k_p}{k_p + k_q}$	$\frac{k_p}{k_p + k_q}$	$\frac{-k_p k_q}{k_p + k_q}$	0
Ward's method: метод Уорда, в якості выдстані між кластерами береться приріст суми квадратів відстаней до центрів кластерів, отриманий в результаті їх обєднання	$\frac{k_r + k_p}{k_r + k_p + k_q}$	$\frac{k_r + k_p}{k_r + k_p + k_q}$	$\frac{-k_r}{k_r + k_p + k_q}$	0

4.2.3. Ефективність ієрархічної кластеризації

Базовий алгоритм ієрархічної кластеризації не дуже ефективний. На кожному кроці нам потрібно розраховувати відстань між усіма парами кластерів, щоб знайти найкращу пару для об'єднання. Перший крок займає $O(n^2)$, а кожний наступний крок займають час, пропорційний до $(n-1)^2$, $(n-2)^2$, … . Сума квадратів від 1 до n становить $O(n^3)$, отже цей алгоритм кубічний. Тому він може використовуватись лише для невеликих наборів точок.

Однак існує ефективне застосування, про яке нам треба знати.

1. Починаємо ми попереднім шляхом, знаходячи відстані між усіма парами точок, отже цей крок $O(n^2)$.
2. Сформуємо пари та їх відстані у пріоритетній черзі, задля можливості знайти найменшу відстань за один крок. Ця операція також $O(n^2)$.
3. Коли ми об'єднуємо два кластери C та D, ми видаляємо з черги усі входження, які мають відношення до цих кластерів; ця робота займає $O(n \log n)$, оскільки нам треба зробити 2n вилучень, то вилучення з черги пріоритетів можна виконати за $O(\log n)$.
4. Тоді ми розраховуємо усі відстані між новими та вже існуючими кластерами. Це робиться за $O(n \log n)$, оскільки існує щонайбільше n нових входжень до черги пріоритетів, внесення цих входжень робиться за $O(\log n)$.

Оскільки останні два кроки робляться максимум n разів, а перші два кроки тільки раз, то загальний час роботи $O(n^2 \log n)$. Це краще за $O(n^3)$, але це все ще накладає жорсткі обмеження на розмір n.

4.2.4 Альтернативні правила для управління ієрархічною кластеризацією

Ми вже знаємо правило обрання кластерів за відстанню: знайти найближчу пару и об'єднати її у кластер. Існують і інші варіанти, представлені далі.

1. Вважати відстань між двома кластерами мінімальною відстанню між будь-якими двома точками, одна з яких лежить у кластері. Наприклад для рисунку 7.3 наступною кластеризована була б точка (10,5) з кластером з двох точок, оскільки (10,5) має відстань

$\sqrt{2}$ і жодна пара не кластеризованих точок не лежить ближче.

2. Вважати відстань між кластерами середньою відстанню усіх пар точок, кожна від окремого кластера.
3. Радіус кластера це максимальна відстань між усіма точками та центроїдою. Об'єднання кластерів за мінімальним радіусом результуючого кластера. Інший варіант — за мінімальною середньою відстанню між центроїдою та точками результуючого кластера. Ще один варіант — використовувати суми квадратів відстаней між точками та центроїдою.
4. Діаметр кластера — це максимальна відстань між двома точками цього кластера. Слід зазначити, що радіус та діаметр кластера не пов'язані прямо, як у колі, але вони мають тенденцію бути пропорційними. Ми можемо об'єднувати кластери за критерієм мінімального діаметру результуючого кластера. Можливі варіації та аналоги цього правила.

Приклад 4.2.4.1. Нехай існує кластер з п'яти точок справа на рисунку 4.2.1.1. Центроїда цих точок — (10.8,4.2). Існує 2 однаково віддалені від центроїди точки: (9,3) та (12,6), обидві мають відстань 2.16. Отже радіус — 2.16. Щоб знайти діаметр, треба знайти дві найвіддаленіші точки. Це знову (9,3) та (12,6). Їхня відстань 4.24, отже це і є діаметр. Зверніть увагу, що діаметр не вдвічі більше радіуса, хоча й дуже близький до цього. Причина цього у тому, що центроїда не лежить на прямій між (9,3) та (12,6).

У нас також є можливість вибрати критерій зупинки об'єднання. Ми вже зазначили критерій «зупинитись, коли залишилось *k* кластерів» у випадку, коли задане *k*. Інші критерії подані далі.

1. Зупинитись, коли діаметр результуючого кластера перевищує поріг. Це також можна робити для радіуса, або будь-якого варіанта радіусу зазначених вище.
2. Зупинитись, коли щільність результуючих кластерів буде менше якогось зазначеного порогу. Щільність можна задати декількома шляхами. Грубо кажучи, це кількість точок на кластер. Відношення вираховується через кількість точок, розділених на потужність

діаметра або радіуса кластера. Коректною потужністю може бути вимірність кластера. Інколи, 1 або 2 вибираються як потужність, не зважаючи на розмірність кластера.

3. Зупинитись, коли є підстави вважати результуючий кластер наступного об'єднання поганим. Наприклад, можна відстежувати середній діаметр нинішніх кластерів. Доки ми комбінуємо точки у кластері, їх середнє буде зростати. Але якщо об'єднати два кластери, що не повинні об'єднуватись, тоді середній радіус зросте різким стрибком.

Приклад 4.2.4.2. Ще раз проаналізуємо рисунок 4.2.1.1.

Він має три природні кластери. Ми вирахували діаметр найбільшого — п'яти точок з права — в прикладі 4.2.4.1; 4.24. Діаметр три точкового кластера знизу зліва 3, відстань між (2,2) та (5,2). Діаметр чотирьох точкового кластеру згори зліва 3.61. Середній діаметр 3.62 був досягнутий від 0 за дев'ять об'єднань, такий ріст є відносно повільним: 0.4 за об'єднання.

Якщо нам потрібно об'єднати два з цих кластерів, то найкращим варіантом буде об'єднати два лівих кластера. Діаметр результуючого кластера 9.43, це відстань між точками (2,2) та (7,10). Тепер, середнє діаметрів 6.84. Це середнє підскочило за один крок майже на стільки, на скільки до цього за дев'ять попередніх кроків. Це порівняння вказує на те, що останнє об'єднання було недоцільним і його потрібно відмінити і зупинити алгоритм.

4.2.5. Ієрархічна кластеризація у неевклідових просторах

У випадку неевклідових просторів нам потрібні деякі відстані, що вираховуються з точок, такі як жаккардова, косинусальна або редакційна. Проблема виникає у представленні кластера, оскільки ми не можемо замінити набор точок їх центроїдою.

Приклад 4.2.5.1. Проаналізуємо проблему представлення на прикладі редакційної відстані, нехай ми намагаємось об'єднати рядки abcd та aecdb. Їх відстань становить 3, тому вони можуть бути об'єднані. Однак не існує рядка, який б представляв їх середнє, або був би природною серединою між ними. Ми можемо використати один з рядків, через який ми проходимо у процесі перетворення одного рядка в інший методом одиночного вставлення або видалення елементу,

наприклад aebcd, або будь-який інший. До того ж при формуванні кластерів з більш ніж двох рядків поняття «шлях між двома» перестає мати сенс.

Не маючи можливості об'єднувати точки в неевклідових просторах, нашим єдиним вибором стає підбір точки з кластеру для його представлення. В ідеалі, ця точка лежить близько до усіх інших точок кластеру, тому в деяких випадках вона лежить у «центрі» кластера. Будемо називати таку точку «кластероїдою». Вибирати кл астероїду можна декількома шляхами, кожен з яких призначений мінімізувати відстань між кластероїдою та іншими точками в кластері. Зазвичай кластероїдою вибирають точку, що мінімізує:

1. суму відстаней до усіх інших точок у кластері;
2. максимальну відстань до іншої точки у кластері;
3. суму квадратів відстаней до інших точок у кластері.

Приклад 4.2.5.2. Припустимо, що ми використовуємо редакційну відстань, і наш кластер складається з чотирьох точок abcd, aecdb, abecb, та ecdab. Їх відстані представлені у наступній таблиці 4.2.5.1.

Таблиця 4.2.5.1. Відстані між точками

	ecdab	*abecd*	*aecdb*
abcd	5	3	3
aecdb	5	2	
abecb	4		

Якщо ми застосуємо критерії кластероїди для кожної з точок, то ми отримаємо результат, поданій в таблиці 4.2.5.2.:

Таблиця 4.2.5.1. Результат застосування критеріїв кл астероїди

Point	*Sum*	*Max*	*Sum-Sq*
abcd	11	5	43
aecdb	7	3	17
abecd	9	4	29
ecdab	11	5	45

З результатів ми бачимо, що будь-який з критеріїв обирає точку aecdb кластероїдою. Зазвичай різні критерії призводять до вибору різних точок.

Варіанти знаходження відстані між кластерами можуть бути використані і в неевклідових просторах, тільки замість центроїди ми використовуємо кластероїду. Наприклад, ми можемо об'єднати два кластера з найближчими кл астероїдами. Ми також можемо використовувати середню мінімальну відстань між усіма парами точок у кластерах.

Інший критерій, пов'язаний з виміром щільності кластера, оснований на радіусі або діаметрі. Обидва варіанти мають місце у неевклідовому просторі. Діаметр все ще максимальна відстань між двома будь-якими точками у кластері. Радіус може бути знайдений при використанні кластероїди замість центроїди. Більш того, має сенс використовувати той самий метод оцінки для радіусу, що й для знаходження кластероїди. Наприклад, якщо ми знаходимо кластероїду як точку з найменшими сумами квадратів відстаней до інших вершин, то радіус буде знаходитись як сума квадратів (або їх квадратний корінь).

Жодний з критеріїв зупинки об'єднання кластерів, які булі розглянуто, не використовував центроїду, крім поняття радіусу і ми вже переконались, що радіус має сенс у неевклідових просторах. Тому не існує істотної різниці в критеріях зупинки при переході до неевклідових просторів.

4.3. Неієрархічні алгоритми кластеризації

Значну популярність при розвязанні задач кластеризації набули алгоритми, що базуються на пошуці оптимального в деякому сенсі розбиття множин даних на кластери. В багатьох задач в силу своїх переваг використовуються саме алгоритми побудови розбиття. Дані алгоритми намагаються згрупувати дані в кластери таким чином, щоб цільова функція алгоритма розбиття досягала екстремуму (мінімума). Розглянемо основні алгоритми кластеризаціїї, що базуються на методах розбиття. В цих алгоритмах використані наступні базові поняття:

- Навчальна множина (вхідна множина даних) M, на якій будується розбиття;
- Метрика відстані

$$d_A^2\left(m_j, c^{(i)}\right) = \left\| m_j - c^{(i)} \right\|_A^2 = \left(m_j - c^{(i)}\right)^t A(m_j - c^{(i)})$$

де матриця A визначає спосіб підрахунку відстані. Наприклад, для одиничної матриці будемо використовувати відстань по Евкліду.

- Вектор центрів кластерів C;
- Матриця розбиття по кластерам U;
- Цільова функція $J=J(M, d, C, U)$;
- Набір обмежень.

Дані, що досліджуються, представляються як множина векторів в багатовимірному просторі, що описує деякі об'єкти предметної області. Кожен вектор складається з набору координат, що іменуються атрибутами. Кожен атрибут має свою природу (числову або категоріальну) і свою множину значень. Множина значень у випадку числової природи атрибута задається у вигляді одного або декількох інтервалів числової осі. У випадку категоріальної природи атрибута множина значень задається перерахунком усіх можливих значень.

Велике значення при підгготовці даних до кластеризації має їх очистка та норумвання. Очистка даних проводиться як по вхідній множині, так ві по множині атрибутів. Очистка даних по множині атрибутів необхідна для виключення залежних атрибутів, які не вплинуть на результат, але збільшать час, необхідний на обробку даних. Нормування необхідне для того, чтоб га результати кластеризації не впливали різні області значень окремих атрибутів. При цьому числові і категоріальні атрибути нормується по рзіному. В результаті кожен атрибут представляється значенням з одиничного інтервалу.

Метрика відстані — множливо, найважливіше поняття, що використовуються в кластеризації. Саме за допомогою відстані між вхідними векторами визначається їх схожість аботрізниця. Є багато способів визначення відстані: евклідова, Хеммінга, відстань Махаланобіса та ін. Вибір способу підрахунку відстані залежить від природи обєктів що використовуються і беспосередньо впливає на результат.

Цільова функція — це функція, мінімізація якої дає розвязок задачі кластеризації. Послідовність дій, що реалізує пошук мінімума цільової функції, є алгоритмом кластеризації.

Матриця розбиття — основний результат неієрархічнох кластеризації. Матриця розбиття представляє собою таблицю, де кожна ячейка містить значення функції належності даного вектора заданому кластеру. На основі цієї матриці отримується

результуюче розбиття. В більшості алгоритмів крім матриці належності в якості результату породжується множина центрів кластерів. Центр кластеру — це вектор, степінь належності якого заданому кластеру максимальна. Як правило, центрів кластерів немає у вхідній множині.

Набір обмежень пов'язаний з умовами, що накладаються на значення елементів матриця належності. Ці обмеження визначаються алгоритмом, що використовується для кластеризації.

4.3.1. Алгоритм k-Means (Hard-c-means)

Даний алгоритм складається із таких кроків:

1. Випадково обрати k точок, які будуть початковими центрами мас кластерів
2. Віднести кожен об’єкт до кластеру з найближчим центром мас
3. Перерахувати центри мас кластерів згідно з поточним членством
4. Якщо критерій зупинки алгоритму не виконується, повернутися до кроку 2.

У якості критерію зупинки зазвичай обирають один з двух варіантів:

1. Немає переходу об’єктів з кластера в кластер на кроці 2
2. Мінімальна зміна середньоквадратичної помилки

Алгоритм чутливий до вибору центрів мас.

Розглянемо на прикладі. Для даних, зображених на рисунку 4.3.1 кроки алгоритму представлені на рисунку 4.3.2.

Важливо відмітити, що метод k-середніх гарно працює, якщо дані по своїй природі діляться на компактні, приблизно сферичні групи.

Даний алгоритм є прообразом майже всіх алгоритмів нечіткої кластеризації і його формалізація допоможе кращому розумінню принципів, закладених в більш складні алгоритми.

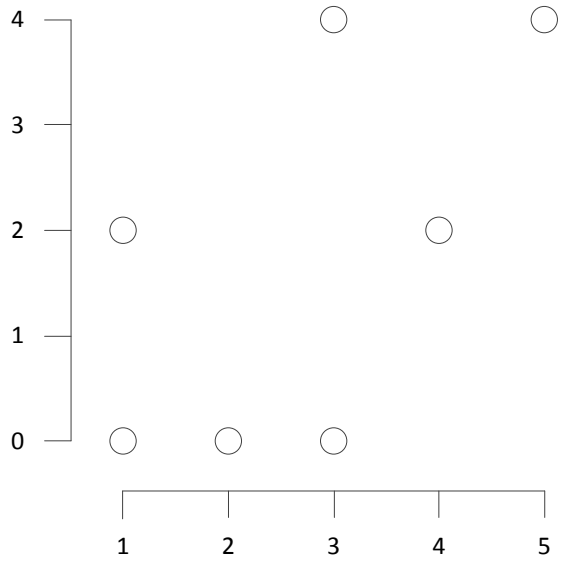


Рисунок 4.3.1. Приклад даних

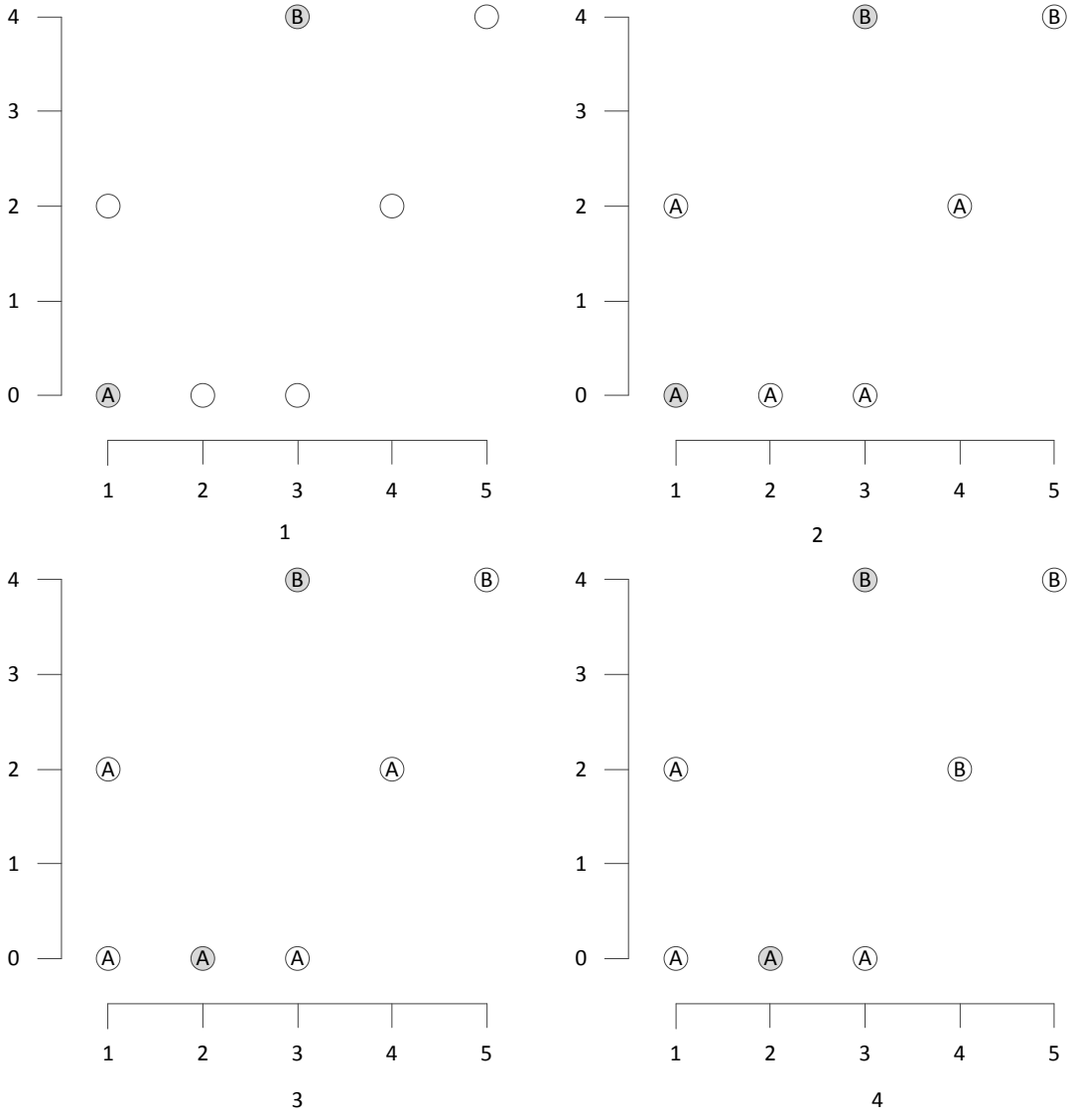


Рисунок 4.3.2. Кроки алгоритму k-means

Базові визначення і поняття в рамках даного алгоритму мають вигляд:

- Навчальна множина $M = \{m_j\}_{j=1}^{d}$, d — кількість точок (векторів) даних;
- Метрика відстані по формулі:
  - $d_A^2\left(m_j, c^{(i)}\right) = \left\| m_j - c^{(i)} \right\|_A^2 = \left(m_j - c^{(i)}\right)^t A(m_j - c^{(i)})$
- Вектор центрів кластерів $C = \{c^{(i)}\}_{i=1}^{c}$, де

$$c^{(i)} = \frac{\sum_{j=1}^{d} u_{ij} m_j}{\sum_{j=1}^{d} u_{ij}}, \; 1 \le i \le c;$$

- Матриця розбиття:

$$U = \{u_{ij}\}, \text{ де}$$

$$u_{ij}^{(l)} = \begin{cases} 1 & d(m_j, c_i^{(l)}) = \min\limits_{l \le k \le c} d(m_j, c_i^{(l)}) \\ 0 & \textit{інакше} \end{cases}$$

- Цільова функція:

$$J(M, U, C) = \sum_{i=1}^{c} \sum_{j=1}^{d} u_{ij} d_A^2(m_j, c^{(i)})$$

- Набір обмежені:

$$u_{ij} \in \{0,1\}; \; \sum_{i=1}^{c} u_{ij} = 1; 0 < \sum_{i=1}^{d} u_{ij} < d$$

який визначає, що кожен вектор даних може належати тільки одному кластеру і не належати іншим. В кожному кластері міститься не менше однієї точки, але менше загальної кількості кластерів.

Конструктивно алгоритм представляє собою ітераційну процедуру наступного вигляду.

Крок 1. Проініціювати початкове розбиття (наприклад, випадковим чином). Вибрати точність δ (використовується в умові завершення алгоритму), про ініціювати номер ітерації l = 0.

Крок 2. Визначити центри кластери по наступній формулі:

$$c_l^{(i)} = \frac{\sum_{j=1}^{d} u_{ij}^{(l-1)} \cdot m_j}{\sum_{j=1}^{d} u_{ij}^{(l-1)}}, \ 1 \le i \le c$$

Крок 3. Оновити матрицю розбиття з тим, щоб мінімізувати квадрати помилок, використовуючи формулу:

$$u_{ij}^{(l)} = \begin{cases} 1 & d(m_j, c_i^{(l)}) = \min_{l \le k \le c} d(m_j, c_i^{(l)}) \\ 0 & \textit{інакше} \end{cases}$$

Крок 4. Перевірити умову $\left\| U^{(l)} - U^{(l-1)} \right\| < \delta$. Якщо умова виконується, завершити процес. Інакше — перейти до кроку 2 з номером ітерації l+1.

Основний недолік даного алгоритму в силу дискретного характеру елементів матриці розбиття — великий розмір простору розбиття. Одним із способів боротьби з даним недоліком є представлення елементів матриці числами з одиничного інтервалу Тобто належність елемента даних кластеру має визначатися функцією належності — елемент може належати декільком кластерам з різною стелінню належності.

4.3.1.1. Підбір правильного k

Ми можемо не знати коректне значення *k*, яке можна використати у k-means кластеризації. Однак, якщо ми виміряємо ефективність кластеризації для різних значень *k*, ми можемо в певній мірі вгадати найкраще значення *k*. Розглянемо вплив росту відповідності кластера на ріст його радіуса або діаметра, при якому різкий ріст цих значень сигналізує про досягнення оптимальної кількість кластерів (рисунок 4.3.1.1.1).

Якщо ми не знаємо коректного значення k, ми можемо знайти хороше значення через кількість операцій кластеризації, що зростає логарифмічно, доки залишається коректним. Почнемо з запуску k-середнього алгоритму для k=1, 2, 4, 8, … . Колись будуть досягнуті величини v та $2v$ між якими є невелике зменшення діаметру, або іншої міри відповідності кластера. Ми можемо завершити тим, що значення k задається на проміжку між $v/2$ та v. Якщо ви використаєте бінарний пошук на цьому проміжку, ви можете знайти найкраще значення *k* за $\log 2v$

операцій кластеризації, тобто усього за $2\log 2v$ операції. Оскільки реальне значення k щонайменше *v/2*, ми маємо кількість операцій, логарифмічну до *k*.

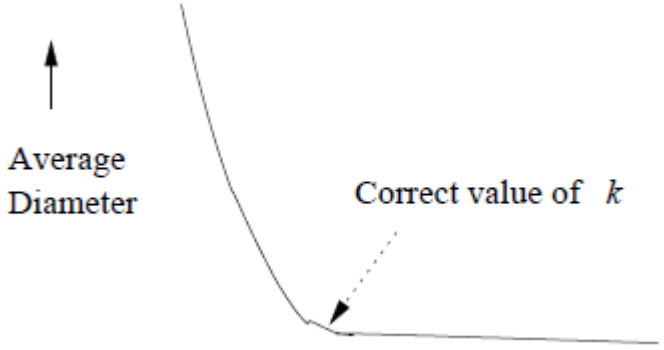


Рисунок 4.3.1.1.1. Середній діаметр або інша величина починає різко зростати із досягненням нижньої межі кількості адекватних кластерів

Оскільки визначення «не сильно змінюється» є неточним, ми не можемо сказати точно, коли воно зміниться сильно. Однак бінарний пошук може бути використаний, якщо припустити, що термін «не сильно змінюється» уточнений формулою. Ми знаємо, що існує велика зміна у проміжку між *v/2*та *v*, інакше ми б не виконували кластеризацію для *2*v кластерів. Припустимо, що у якийсь момент ми звузили проміжок *k* до проміжку між *x* та *y*. Нехай *z*=(*x*+*y*)/2. Запустимо кластеризацію с *z* вихідними кластерами. Якщо не існує значної різниці між *z* та y, тоді істинне значення *k* лежить між *x* та *z*. Тоді ми можемо рекурсивно зменшувати проміжок, щоб знайти коректне значення *k*. З іншого боку, якщо існує відчутна різниця між *z* та *y*, тоді використовуємо бінарний пошук у проміжку *z* та *y*.

4.3.2. Алгоритм Бредлі, Файяда та Рейна

Цей алгоритм, який ми будемо називати BFR (Bradley, Fayyad, Reina), за ініціалами його авторів, є варіантом k-середнього алгоритму, призначений для кластеризації даних у багатовимірних евклідових просторах. Він робить сильні припущення щодо форми кластерів: вони повинні бути нормально розподілені навколо центроїди. Середнє значення та стандартне відхилення кластера може відрізнятися для різних вимірів, але виміри повинні бути незалежними. Наприклад у двох вимірах кластер може бути у формі «сигари», але «сигару» не можна обертати по обох осях. Суть передана на рисунку

4.3.2.1.

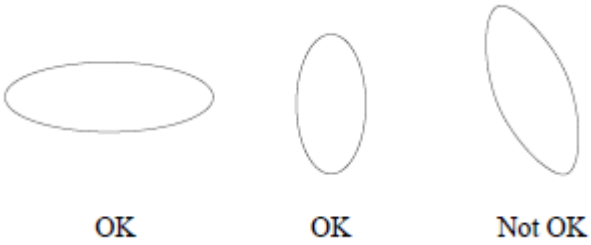


Рисунок 4.3.2.1. Кластери даних, для яких може бути використаний BFR-алгоритм повинні мати стандартне відхилення, що розповсюджується за осями, але осі кластера повинні відповідати осям простору.

BFR-алгоритм починається з вибору k точок, використовуючи один з методів, описаних вище. Тоді точки з файлу даних зчитується частинами. Це можуть бути як частини з розподіленої файлової системи або за файлом, який містить інформацію про розбиття даних на частини відповідного розміру. Кожна частина повинна складатися з такої кількості точок, щоб основної пам’яті вистачило для їх обробки. Також у основній пам’яті зберігаються дані про всі k кластерів та деякі інші дані, тому навіть основна пам'ять не може повністю бути виділена на зберігання частини. Дані у основній пам’яті окрім частини, діляться на три типи.

1. Відкинутий набір: Це прості підсумовані відомості про кластери. Ми звернемося до способів підсумування кластерів згодом. Зверніть увагу, що хоча підсумки кластеру не «відкинуті»; Більш того, вони дуже важливі. Однак, точки, що входять до підсумку відкинуті і не мають ніякого представлення у основній пам’яті крім цього підсумку.
2. Стиснутий набір: Це підсумки, подібні до підсумків кластера, але для набору точок, що лежать близько одна до одної, але не належать жодному кластеру. Ці точки також відкинуті, у сенсі, що вони не з’являються в основній пам’яті ніде крім підсумку. Ми будемо називати такі набори точок міні кластерами.
3. Збережений набір: Деякі точки неможливо ані призначити до кластеру, ані вони не лежать досить близько до інших точок, з якими вони можуть бути представлені у виді стиснутого набору. Ці точки зберігаються в основній пам’яті так, як вони і з’являються у вхідному файлі.

Рисунок 4.3.2.2 відображає, як обробляються представлені точки.

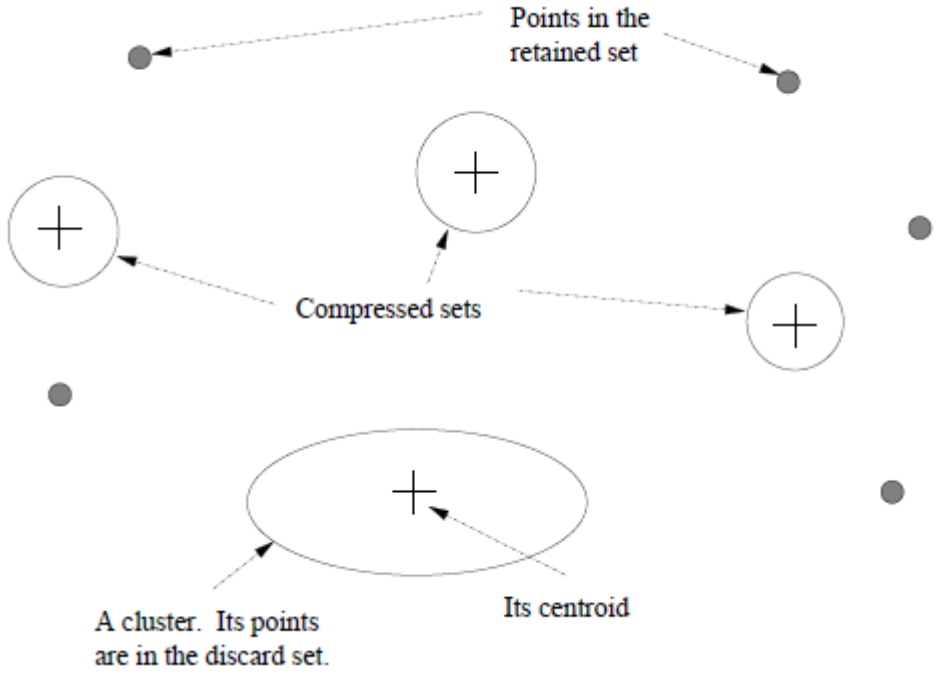


Рисунок 4.3.2.2. Точки у відкинутому, стиснутому та збереженому наборі.

Відкинутий та стиснутий набори представлені $2d + 1$ значеннями, якщо дані d-вимірні. Ці значення це:

a) кількість представлених точок, N;

b) сума компонентів усіх точок у кожному вимірі; ці дані представлені у вигляді вектора SUM довжини d, i компоненти в i-тому вимірі — SUM_i;

c) сума квадратів компонентів усіх точок у кожному вимірі; ці дані представлені у вигляді вектору SUMSQ довжини d, і компоненти в i-тому вимірі — $SUMSQ_i$;

Нашою метою є представлення набору точок за їх порядком, їхньою центроїдою та стандартним відхиленням в кожному вимірі. Однак, ці $2d + 1$ значень надають нам цю статистику. N — це порядок. Координатами центроїди в i-тому вимірі є SUM_i/N. Дисперсія у i-тому просторі це:

$$SUMSQ_i / N - (SUM_i/N)^2$$

Variance:

$$SUMSQ_i/N - (SUM_i/N)^2$$

Приклад 4.3.2.1. Припустимо кластер складається з точок

(5, 1), (6, −2), та (7, 0). Тоді N = 3, SUM = [18, −1], та SUMSQ = [110, 5]. Центроїда це SUM/N , або [6, −1/3]. Дисперсія в першому вимірі це 110/3 – $(18/3)^2$=0.667, отже стандартне відхилення це $\sqrt{0.667}$=0.816. У другому вимірі дисперсія дорівнює 5/3 – $(-1/3)^2$ = 1.56, отже стандартне відхилення становить 1.25.

Подивимось, що відбувається під час обробки частин точок.

4.3.2.1. Переваги представлення N , SUM, SUMSQ

Існують відчутні переваги представлення набору точок у BFR-алгоритмі порівняно із зберіганням *N*, центроїди та стандартного відхилення у кожному вимірі. Враховуйте, що нам потрібно при додаванні точки до кластера. *N* збільшується на 1.

Але ми також можемо додати вектор, що представляє розташування точки у SUM щоб отримати новий SUM і ми можемо додати квадрати і-тої компоненти вектора до $SUMSQ_i$, щоб отримати новий $SUMSQ_i$. У випадку, якщо замість SUM ми використовуємо центроїду, тоді ми не могли б фіксувати центроїду з урахуванням нової точки без додаткових розрахунків, що залучають N та перерахунок стандартного відхилення також став би складнішим. Також, якщо ми хочемо об'єднати два набори, ми можемо додати відповідні величини N, SUM та SUMSQ, але при використанні центроїди та стандартних відхилень як представлення, розрахунку стали б набагато складнішими.

1. По-перше, усі точки, що є достатньо близькими до центроїди кластера, додаються до цього кластера. Як було сказано вище, додати інформацію про точку, що представляє кластер, до N, SUM та SUMSQ просто. Потім ми відпускаємо точку. Поняття «достатньо близької» буде розглянуте найближчим чином.
2. Для точок, що не лежать достатньо близько до будь-якої центроїди, ми об'єднуємо їх у кластер з точками зі збереженим набором. Можно використати будь-який алгоритм кластеризації у основній пам'яті. Ми повинні використати деякі критерії щоб визначити резонність об'єднання двох точок у кластер або двух кластерів у один. Кластери, що мають більше ніж одну точку сумуються та додаються до зжатого набору. Кластери з однією точкою стають точками з поверненого набору.

3. Тепер ми маємо міні кластери, отримані з нашим спроб кластеризувати нові точки та старий збережений набір і ми маємо міні кластери за старого зжатого набору. Навіть якщо міні кластери неможливо об'єднати з жодним з k кластерів, вони можуть об'єднуватись між собою. Слід зазначити, що форма представлення стиснених наборів (N, SUM та SUMSQ) полегшує розрахунок статистики, наприклад кандидати серед міні кластерів на об'єднання.
4. Точки, що приписані до кластеру або міні кластеру записуються у вторинну пам'ять.

Нарешті, якщо це остання частина вхідних даних, потрібно зробити щось із збереженим та поверненим наборами. Ми можемо роздивлятися їх как залишки і не кластеризувати. Або ми можемо приписати кожну точку у збереженому наборі до найближчої центроїди. Ми можемо об'єднати кожен міні кластер з кластером, чия центроїда найближча до центроїди міні кластера. Важливим рішенням є вибір, чи є точка p достатньо близькою до одного з k кластерів, щоб мало сенс їх об'єднання. Існує два підходи.

a) Додати p до кластера, якщо не тільки центроїда лежить найближче до p, але і не існує точок, об'єднання яких з кластерами утворить ближчу центроїду. Це рішення є складним статистичним розрахунком. Воно передбачає задання точок навмання та заздалегідь відому кількість точок, що будуть оброблятися. Перевагою є те, що якщо ми знайдемо таку центроїду, то ми можемо додати точку p до кластера і забути про неї.
b) Ми можемо вимірювати ймовірність того, що p належить до кластера, вона б знаходилась навіть на великих відстанях до центроїди. Цей розрахунок базується на припущенні того, що кластер складається з нормально розподілених точок по всі осям у просторі. Це дозволяє нам розраховувати відстань Махаланобіса між точками, яку розглянемо далі.

Відстань Махаланобіса це відстань між точкою та центроїдою кластера, нормалізована за стандартним відхиленням кластера у кожному вимірі. Оскільки BFR-алгоритм припускає відповідність осей кластера до осей простору, розрахунок відстані Махаланобіса є досить простим. Нехай

p=[$p_1, p_2,.., p_d$] — точка, а c=[$c_1, c_2,.., c_d$] — центроїда кластера. Нехай σ_i — стандартне відхилення точок кластера у і-тому вимірі. Тоді відстань Махаланобіса між p та c:

$$\sqrt{\sum_{i=1}^{d}\left(\frac{p_i - c_i}{\sigma_i}\right)^2}$$

Отже ми нормалізуємо відстань між p та c в і-тому вимірі розділивши її на стандартне відхилення кластера у цьому вимірі. Залишок формули об'єднує нормалізовані відстані в кожному вимірі звичайним для евклідового простора шляхом. Щоб додати p до кластеру, ми розраховуємо відстань Махаланобіса між p та центроїдою кожного кластера. Ми обираємо кластера, чия центроїда має найменшу відстань Махаланобіса і ми додаємо p до кластера, якщо її відстань Махаланобіса не перевищую поріг.

Наприклад, ми обираємо чотири пороги, якщо дані розподілені нормально, тоді ймовірність значення перевищити чотири пороги менше ніж одні мільйонна. Тому, якщо точки кластера розподілені нормально, тоді ймовірність невдалого включення точки менше ніж 10^{-6}. І така точка буде приписана до кластера у будь-якому випадку, доки вона не опиниться ближче до іншої центроїди з міграцією центроїд із ростом кластерів.

4.3.3. Метод найближчого сусіда

Цей метод є одним із найстаріших методів кластеризації. Він був створений у 1978 році.

Для кожного об'єкту зовні кластеру робимо наступне

1. Знаходимо найближчого сусіда, кластер якого визначений
2. Якщо відстань до цього сусіда менше порогу, то відносимо в той же кластер. Інакше створюємо ще один кластер.

Далі розглядаємо результат і за необхідності збільшуємо поріг. Наприклад, якщо багато кластерів з одного об'єкта.

4.3.4. CURE — алгоритм

Розглянемо інший алгоритм з класу призначення точок для багатомасштабної кластеризації. Цей алгоритм, під назвою

CURE (Clustering Using REpresentatives), працює у евклідовому просторі. Але він не залежить від форми кластерів; вони не повинні бути нормально розподіленими, можуть мати дивні вигини, S-подібну форму, або навіть форму кільця. Замість використання центроїди для представлення кластерів, використовується набір представляючи точок.

Приклад 4.3.4.1. Рисунок 4.3.4.1 ілюструє два кластери. Внутрішній кластер це звичайне коло, а другий — кільце навколо круга. Таке розташування не є повністю патологічним. Створіння з іншої галактики може бачити нашу сонячну систему як внутрішній кластер (планети) та зовнішнє кільце (пояс Койпера), и пустоту між ними.

Ініціалізація CURE

1. Взяти невелику частину даних та кластеризувати його в основній пам'яті. В принципі, можна використати будь-який метод кластеризації, але оскільки CURE розроблений для обробки дивно сформованих кластерів, його частіш за все використовують при ієрархічному методі, коли кластери об’єднуються коли вони мають близьку пару точок.
2. Вибрати невеликий набір точок з кожного кластера як представляючи точки. Ці точки слід обирати якомога далі одна від одної.
3. Зсунути кожну з представляючих точок на фіксовану відстань між їх поточною позицією та центроїдою їх кластера. Мабуть 20% найкращий зсув. Цей шаг можливий тільки у евклідовому просторі, інакше між цими точками неможливо буде провести пряму.

Приклад 4.3.4.2. Ми можемо використати ієрархічну кластеризацію на даних з рисунку 4.3.4.1. Якщо відстань між кластерами розраховувалася б за найкоротшою відстанню між точками, кожна зі свого кластера, тоді ми змогли б коректно знайти два кластери. Тобто точки з кільця залишалися б разом, точки з кола залишались разом, але частини кільця та кола завжди знаходились би далеко. Зазначте, що якби відстань між кластерами розраховувалась за відстанню між центроїдами, то ми могли б не отримати коректний результат. Причиною цього є те, що центроїди обох кластерів лежать у центрі діаграми.

Для наступного кроку ми беремо представляючи точки. Якщо вибірка з кожного кластеру досить велика, ми можемо

розрахувати точки кластера за найбільшою відстанню одна від одної, якщо вони лежать на границі кластера. Рисунок 7.13 ілюструє таку вибірку точок. Нарешті, ми зсуваємо представляючи точки на фіксованій проміжок до центроїди кластера. Зазначте, на рисунку 4.3.4.2. обидва кластера мають центроїди посередині діаграми. Тому представляючи точки з кільця присуваються до кола, що на й було потрібно. Завершальні позиції представляючи точок з рисунку 4.3.4.2 зображені на рисунку 4.3.4.3.

Завершення CURE-алгоритму

Наступною фазою CURE є злиття двох кластерів, якщо вони мають пару представляючи точок, кожна зі свого кластера, що лежать достатньо близько. Користувач може сам задати відстань, що відповідає поняттю «близько». Шаг злиття може тривати доки не залишиться досить близьких кластерів.

Приклад 4.3.4.3. Ситуація на рисунку 4.3.4.3 слугує наглядною ілюстрацією. Можливо, кільце та коло можна об’єднати, оскільки їх центроїди лежать близько. Наприклад, якби прогалина між кільцем та колом була б менше, об’єднання було б не тільки можливим, а й логічним.

У випадку рисунку 4.3.4.3 вибір між зсувом відстані до центроїди, на який ми зсуваємо представляючи точки та вибором наскільки далеко повинні бути представляючи точки двох кластерів, щоб їх об’єднати. Разом надає нам розуміння, чи можливо об’єднати ці два кластери в один, або залишити два кластери.

Останнім кроком у CURE є присвоєння точок. Кожна точка *p* береться із вторинної пам'яті та порівнюється з представляючи ми точками. Ми присвоюємо точку кластеру, якщо вона лежить достатньо близько до результуючої точки цього кластера.

Приклад 4.3.4.4. В нашому прикладі, точки в середині кільця будуть ближче до його представляючи точок ніж до представляючи точок круга. Те ж саме і для круга, точки в ньому будуть лежати ближче до його представляючи точок. Точка що не лежить всередині кільця або кола буде приписана до кільця, якщо вона знаходиться за кільцем. Якщо ж така точка знаходиться між кільцем та колом, вона буде приписана до одного або іншого, в залежності від її близькості до представляючи точок.

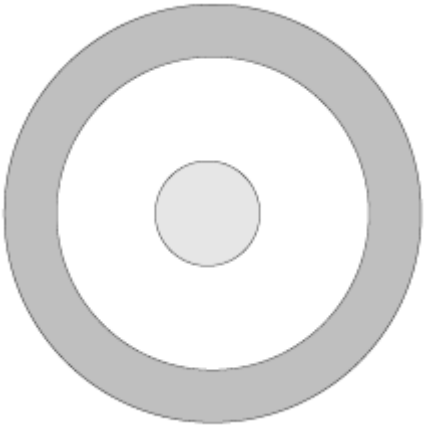


Рисунок 4.3.4.1. Два кластери, один навколо другого

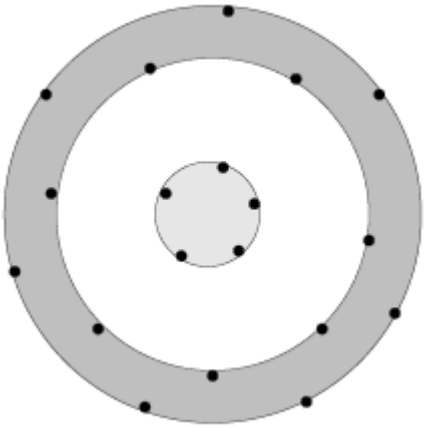


Рисунок 4.3.4.2. Вибрати представляючи точки, якомога далі одна від одної

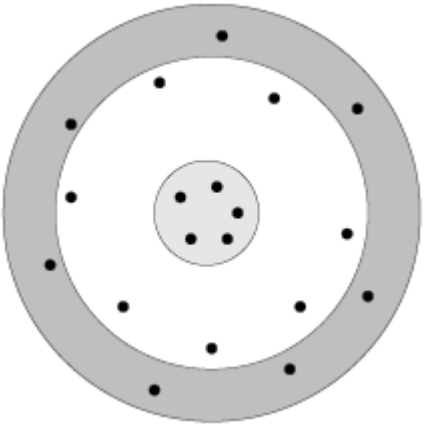


Рисунок 4.3.4.3. Зсув представляючи точок на 20% ближче до центроїди

4.4. Кластеризація у неевклідових просторах

Далі ми розглянемо алгоритм, що працює з даними не з основної пам'яті, і при цьому не потребує евклідового простору. GRGPF-алгоритм (Ganti, Ramakrishnan, Gehrke, Powell, French) використовує принципи як ієрархічного так і точкового підходів. Як CURE, він представляє кластери у пам'яті за точками. Але водночас він намагається організовувати кластери у дерево, і приписує точку до кластеру, шляхом проходу вниз по дереву. Листи дерев містять підсумки кластерів. Акцент робиться на знаходженні близькості кластерів за їх розташування у листах дерева.

4.4.1. Представлення кластерів у GRGPF-алгоритмі

З призначенням нових точок до кластерів, вони збільшуються. Більшість точок зберігають на диску і не приймають участі у призначенні точок, хоча вони можуть завантажуватись. У пам'яті кластер представлений декількома значеннями:

1. N — кількість точок у кластері;
2. кластероїда — точка, що належить кластеру з мінімальною сумою квадратів відстаней до інших точок (ROWSUM);
3. ROWSUM кластероїди кластера;
4. константа k — деяка кількість точок, близьких до кл астероїди;
5. K точок, що лежать найдальше від кл астероїди.

4.4.2. Ініціалізація кластерного дерева

Кластери представляються у вигляді дерева, вузли якого можуть бути дуже великими. Кожен лист дерева вміщає стільки представлень кластера, скільки може вмістити.

Внутрішній вузол вміщає набір кластероїд кластерів, представлених у під деревах, а також вказівки на корені цих піддерев. Набір має фіксований розмір, тому кожен вузол буде мати однакову кількість синів.

Ми запускаємо кластер-дерево беручи з основної пам'яті набір даних та ієрархії кластеризації. Результатом кластеризації стане дерево T, але воно не буде результатом GRGPF-алгоритму. Скоріш з нього ми вибираємо якусь кількість вузлів, що представлятимуть кластери. Це будуть початкові кластери для GRGPF-алгоритму, вони розміщаються у листку дерева. Тоді ми групуємо кластери з загальним вузлом з T у внутрішні вузли

представляю чого дерева.

4.4.3. Додавання точок у GRGPF-алгоритмі

Тепер ми зчитуємо точки із вторинної пам'яті і приєднуємо їх до найближчого кластера. Ми починаємо з вершини і порівнюємо набори кластероїд з усіх дітей вершини. Та дитина, центроїда якої найближче до точки розглядається наступною, коли ми дістаємося до вузла ми порівнюємо набори центроїд дітей і йдемо по дитині, що має найближчу кластероїду. Нарешті, коли ми досягаємо листа, з якого ми вибираємо кластер з найближчою кластероїдою і приєднуємо до нього точку.

Тобто:

1. додати 1 до N;
2. додати квадрат відстані від точки до кожного з q вузлів — ROWSUM(q).

Ми також розраховуємо ROWSUM точки.

$$\text{ROWSUM}(p) = \text{ROWSUM}(c) + \text{N}d^2(p, c)$$

де $d(p, c)$ — відстань між точкою та кластероїдою.

4.4.4. Розділення та об'єднання кластерів

GRGPF-алгоритм накладає обмеження на радіус кластера:

$$\sqrt{ROWSUM(c)/N}$$

де c — кластероїда, N — кількість точок у кластері. При перевищенні радіуса кластера межі, кластер розбивається на два. Точки кластерів повертаються в основну пам'ять, де діляться на дві частини шляхом мінімізації ROWSUM.

Результатом стане те, що лист тепер представляє два кластери, якщо лист не може вмістити обидва кластери, його треба розбити на два листи.

Найгіршим випадком всього цього стане те, що дерево не буде вміщуватись у основну пам'ять. Тоді зробити можна лише одне: зменшити його шляхом збільшення обмеження на розмір радіусу кластера і об'єднати деякі кластери. Зазвичай достатньо об'єднати лише кластери «поблизу» один від одного, в сенсі, кластери, що знаходяться на одному листку. Для об'єднання кластерів C1 і C2 у кластер C ми припускаємо, що кластероїдою C буде одна з найвіддаленіших точок від кластероїди C1 або C2.

$$ROWSUM_C(p) = ROWSUM_{C1}(p) +$$
$$+ N_{C2}(d^2(\text{p}, C_1) + d^2(C_1, C_2)) + ROWSUM_{C2}(C_2)$$

Потім ми повинні завершити розрахунок значень об'єднаного кластера. Ми повинні враховувати усі точки нового кластера, для яких ми знаємо їх ROWSUM. Це центроїди кластерів, k точок, що лежать найближче до кластероїд і k точок, найвіддаленіших від кластероїд, за винятком точки, що і є кластероїдою. Для кожної з цих $4k+1$ точок ми можемо вирахувати відстані. Ми обираємо k з найменшими відстанями на роль «найближчих» точок і k з найбільшими на роль «найдальших». Їх ROWSUM можна розрахувати за зазначеною вище формулою.

4.5. Кластеризація у потоках та паралелізм

У цьому розділі ми роздивимось як можна кластеризувати потік.

4.5.1. Модель розрахунку потоку

Ми припускаємо, що кожен елемент потоку є точкою у просторі. Кожне вікно складається з останніх N точок. Нашою метою є попередня кластеризація під наборів точок у потоці. Існує багато варіантів утворення такої впорядкованості, в залежності від того, що ми беремо за означення цілісності кластера.

Ми не робимо жодних обмежень щодо простору, в якому можуть лежати точки потоку. Він може бути евклідовим, тоді відповіддю буде порядок центроїд кластерів, або неевклідовою, тоді відповіддю буде порядок кластероїд кластерів.

Проблема значно спрощується, якщо припустити, що усі обрані точки з потоку мають незмінну статистику. Тоді набору з потоку достатньо, щоб утворити кластери, і можна на деякий час забути про потік. Однак потокова модель вважає зміну статистики точок з плином часу.

4.5.2. Алгоритм кластеризації потоку

В цьому розділі ми розглянемо значно спрощений BDMO-алгоритм (Babcock, Datar, Motwani, O'Callaghan). Повна версія алгоритму потребує значно складніших структур, які призначені забезпечити результат навіть у найгірших випадках.

BDMO-алгоритм базується на методології підрахунку потоків, ось основні подібні етапи.

1. Точки потоку розбиваються та сумуються комірками потужності два.

2. Як і раніше, розміри комірок підлягають обмеженню свого розміру.
3. Розмір комірок не можна зменшувати із плином часу.
4. Комірки складаються з:
    a. розміру комірки;
    b. часової мітки комірки.
    c. наборів записів, що представляє кластери, з яких були взяті точки комірки; ці записи складаються:
        i. кількості точок у кластері;
        ii. центроїди або кластероїди кластера;
        iii. інших необхідних параметрів.

Ініціалізація комірок

Розмір найменшої комірки буде становити p, з потужністю 2. Отже, кожний p-тий елемент потоку створює нову комірку з останніми p точками. Часова мітка цієї комірки є часовою міткою останньої точки з неї. Ми можемо лишити кожну точку в кластері, або можемо кластеризувати точки згідно до обраної стратегії кластеризації.

Злиття комірок

Після створення нової комірки потрібно оглянути вже створені. По-перше, якщо якась комірка матиме часовий штамп більше за N часових одиниць відносно нинішнього часу, то таку комірку можна викинути із списку. По-друге, ми могли створити три комірки розмір p, в випадку чого ми повинні об'єднати два старших з них. Об'єднання створює комірку розміру $2p$, тобто ми отримуємо ріст розмірів із злиттям.

Щоб злити дві послідовні комірки, нам потрібно зробити наступне:

1. Розмір комірки вдвічі більший за розмір комірок, що зливаються.
2. Часова мітка нової комірки є часова мітка останньої комірки.
3. Ми повинні враховувати, чи варто об'єднувати кластери, і якщо так, ми повинні розрахувати параметри нового кластера.

Відповідь запитам

Пригадайте, що під запитом мається на увазі запит

кластерам про останні m точок з потоку, де $m \leq N$. Через обрану нами стратегію злиття комірок, ми не завжди зможемо знайти ці m точок. Однак, якщо ми виберемо найменший набір комірок, що покривають m точок, ми не будемо включати до них більше ніж 2m точок. У якості відповіді запиту ми створимо центроїди або кластероїди усіх точок у обраній комірці. Щоб отримати відносно точний результат відносно m з кластерів, ми припустимо, що точки між $2m$ та $m+1$ не будуть відчутно відрізнятись за статистикою.

Обравши комірки, ми змішуємо усі кластери. Потім ми використовуємо одну з методологій для злиття кластерів.

Кластеризація у паралельному оточенні

Тепер ми швидко оглянемо можливості паралелізації у кластеризації. Припустимо нам дана дуже велика кількість точок і ми хочемо використати паралелізм, щоб вирахувати центроїди їх кластерів. Найпростішим підходом буде використання map-reduce стратегії, але частіше за все нам потрібно виконати лише reduce.

Почніть зі створення багатьох Map завдань. Кожне завдання приписується до під набору точок. Метою Map функції є кластеризація точок. Його результатом є ключовий набір пар з фіксованим ключем 1 та значенням, що є описом одного кластера. Цей опис може бути створений будь яким із методів, запропонованих у розділі 7.6.2.

Оскільки усі ключові пари мають однаковий ключ, можу бути лише одне Reduce завдання. Це завдання отримує описи кластерів з кожного Map завдання і об'єднує їх між собою.

4.4. Застосування нейронних мереж

Іноді для розв'язання задач кластеризації застосовують нейронні мережі. У цього підходу є наступні особливості

- Штучні нейронні мережі легко працюють у розподілених системах з істотною паралелізацією у силу своєї природи
- Штучні нейронні мережі оперують числами, тому вони можуть проводити розбиття на кластери тільки для об'єктів з чисельними векторами характери сит
- Оскільки штучні нейронні мережі підстроюють свої вагові коефіцієнти, базуючись на вхідних даних, це

допомагає зробити вибір значимих характеристик (етап 1 кластеризації) менш суб'єктивним

Прикладом можуть служити самоорганізаційні карти Кохонена. Вони є аналогом алгоритма k-Means.

4.4.1. Самоорганізаційна карта Кохонена

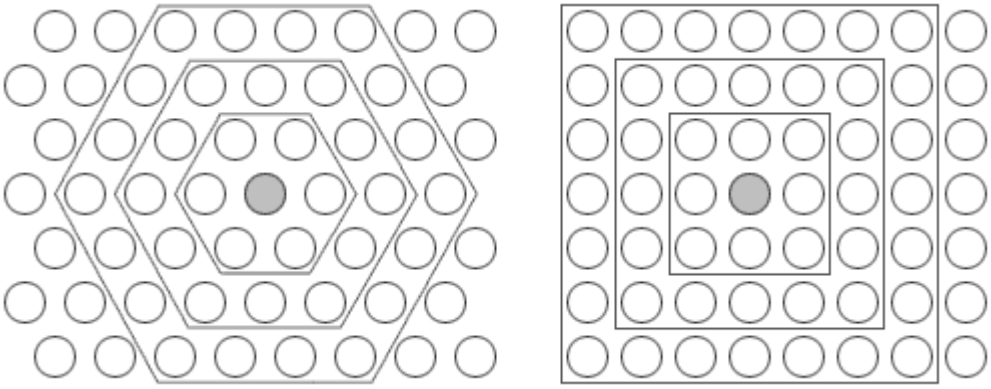
Самоорганізаційна карта Кохонена (*Self-organizing map* — SOM) — це один з різновидів нейромережевих алгоритмів. Основною відмінністю даної є те, що при навчанні використовується метод навчання без учителя, тобто результат навчання залежить тільки від структури вхідних даних. Нейронні мережі даного типу часто застосовуються для вирішення найрізноманітніших завдань , від відновлення пропусків у даних до аналізу даних і пошуку закономірностей, наприклад, у фінансовій задачі. У даній статті ми розглянемо принципи функціонування та деякі аспекти використання самоорганізаційних карт.

Алгоритм функціонування самоорганізаційних являє собою один з варіантів кластеризації багатовимірних векторів. Важливою відмінністю алгоритму SOM є те, що в ньому всі нейрони (вузли, центри класів тощо) впорядковані в деяку структуру (зазвичай двовимірну сітку) . При цьому в ході навчання модифікується не тільки нейрон-переможець, але і його сусіди, але в меншій степені. За рахунок цього SOM можна вважати одним з методів проектування багатовимірного простору в простір з більш низькою розмірністю. При використанні цього алгоритму вектори, схожі у вихідному просторі, виявляються поряд і на отриманій карті.

SOM передбачає використання впорядкованої структури нейронів. Зазвичай використовуються одно- і двовимірні сітки. При цьому кожен нейрон являє собою n-мірний вектор-стовпець $\bar{w} = [w_1, w_2, .., w_n]^T$, де n визначається розмірністю початкового простору (розмірністю вхідних векторів). Застосування одно- і двовимірних сіток пов'язано з тим, що виникають проблеми при відображенні просторових структур більшої розмірності (при цьому знову виникають проблеми споніженіем розмірності до двовимірної, представимо на моніторі).

Зазвичай нейрони розташовуються у вузлах двовимірної сітки з прямокутними або шестикутними осередками. При цьому, як було сказано вище, нейрони також взаємодіють один з одним. Величина цієї взаємодії визначається відстанню між

нейронами на карті. На рис 4.7. подано приклади відстані для шестикутної і чотирикутної сіток.



(а) (б)

Рисунок 4.7. Приклади відстані для (а) шестикутної і (б) чотирикутної сіток.

При цьому легко помітити, що для шестикутної сітки відстань між нейронами більше збігається з евклідовим відстанню, аніж для чотирикутної сітки.

Кількість нейронів у сітці визначає ступінь деталізації результату роботи алгоритму, і в кінцевому рахунку від цього залежить точність узагальнюючої здатності карти.

4.4.2. Початкова ініціалізація карти

При реалізації алгоритму SOM заздалегідь задається конфігурація сітки (прямокутна або шестикутна), а також кількість нейронів в мережі. Деякі джерела рекомендують використовувати максимально можливу кількість нейронів в карті. При цьому початковий радіус навчання (neighborhood) значною мірою впливає на здатність узагальнення за допомогою отриманої карти. У випадку, коли кількість вузлів карти перевищує кількість прикладів в навчальній вибірці, то успіх використання алгоритму у великій мірі залежить від відповідного вибору початкового радіуса навчання. Однак, у випадку, коли розмір карти становить десятки тисяч нейронів, час, необхідний на навчання карти зазвичай буває занадто великим для вирішення практичних завдань, таким чином необхідно досягати допустимого компромісу при виборі кількості вузлів.

Перед початком навчання картки необхідно проініціалізувати вагові коефіцієнти нейронів. Вдало вибраний спосіб ініціалізації може істотно прискорити навчання, і

призвести до отримання більш якісних результатів. Існують три способи ініціювання початкових ваг.

1. Ініціалізація випадковими значеннями, коли всім вагам даються малі випадкові величини.
2. Ініціалізація прикладами, коли в якості початкових значень задаються значення випадково вибраних прикладів з навчальної вибірки
3. Лінійна ініціалізація. У цьому випадку ваги ініціюються значеннями векторів, лінійно впорядкованих уздовж лінійного підпростору, що проходить між двома головних власними векторами вихідного набору даних. Власні вектори можуть бути знайдені наприклад за допомогою процедури Грама-Шмідта.

4.4.3. Навчання

Навчання складається з послідовності корекцій векторів, що представляють собою нейрони. На кожному кроці навчання з вихідного набору даних випадково вибирається один з векторів, а потім проводиться пошук найбільш схожого на нього вектора коефіцієнтів нейронів. При цьому вибирається нейрон-переможець, який найбільш схожий на вектор входів. Під схожістю в даній задачі розуміється відстань між векторами, зазвичай обчислюється в евклідовому просторі. Таким чином, якщо позначить нейрон-переможець як c, то отримаємо $\|\bar{x} - \bar{w}_c\| = \min_i \{\|\bar{x} - \bar{w}_i\|\}$.

Після того, як знайдений нейрон-переможець, проводиться коригування ваг нейромережі. При цьому вектор, що описує нейрон-переможець і вектори, що описують його сусідів у сітці переміщуються в напрямку вхідного вектора. Це проілюстровано на рисунку 4.8. для двовимірного вектора.

Координати вхідного вектора помічені хрестом, координати вузлів карти після модифікації відображені сірим кольором. Вид сітки після модифікації відображений штриховими лініями.

При цьому для модифікації вагових коефіцієнтів використовується формула:

$$\bar{w}_i(t+1) = \bar{w}_i(t) + h_{ci}(t) \cdot \left[\bar{x}(t) - \bar{w}(t)\right]$$

де t позначає номер епохи (дискретний час). При цьому вектор $\bar{x}(t)$ вибирається випадково з навчальної вибірки на ітерації *t*. Функція *h(t)* називається функцією сусідства нейронів. Ця

функція являє собою незростаючу функцію від часу і відстані між нейроном-переможцем і сусідніми нейронами в сітці. Ця функція розбивається на дві частини: власне функцію відстані і функції швидкості навчання від часу.

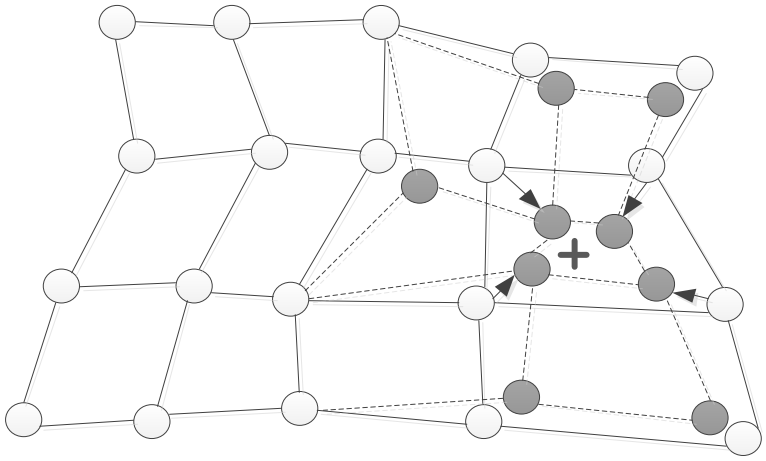


Рисунок 4.8. Переміщення в напрямку вхідного вектора

Звичайно застосовується одні з двох функцій від відстані: проста константа:

$$h(d,t)=\begin{cases} const, & d \le \sigma(t) \\ 0, & d > \sigma(t) \end{cases},$$

або Функція Гауса:

$$h(d,t)=e^{-\frac{d^2}{2\sigma^2(t)}}$$

При цьому кращий результат виходить при використанні функція Гауса.

Розглянемо тепер функцію швидкості навчання $a(t)$. Найбільш часто використовуються два варіанти цієї функції: лінійна і обернено пропорційна часу вигляду:

$$a(t)=\frac{A}{t+B},$$

де A і B це константи. Застосування цієї функції призводить до того, що всі вектори з навчальної вибірки вносять приблизно рівний внесок у результат навчання.

Навчання складається з двох основних фаз: на первинному етапі вибирається досить велике значення швидкості навчання і радіуса навчання, що дозволяє розташувати вектора нейронів відповідно до розподілу

прикладів у вибірці, а потім проводиться точне підстроювання ваг, коли значення параметрів швидкості навчання багато менше початкових. У разі використання лінійної ініціалізації первісний етап грубого підстроювання може бути пропущений.

4.4.4. Застосування алгоритму

Оскільки алгоритм SOM поєднує в собі два основних напрямки — векторне квантування і проектування, то можна знайти і основні застосування цього алгоритму. Дану методику можна використовувати для пошуку та аналізу закономірностей у вихідних даних. При цьому, після того, як нейрони розміщені на карті, отримана карта може бути відображена. Розглянемо різні способи відображення отриманої карти.

При цьому методі відображення отриману карту можна представити у вигляді листкового пирога, кожен шар якого являє собою розмальовку, породжену однією з компонент вихідних даних. Отриманий набір розмальовок може використовуватися для аналізу закономірностей, наявних між компонентами набору даних. Після формування карти ми отримуємо набір вузлів, який можна відобразити у вигляді двовимірної картинки. При цьому кожному вузлу карти можна поставити у відповідність ділянку на малюнку, чотирьох або шестикутний, координати якого визначаються координатами відповідного вузла в решітці.

Тепер для візуалізації залишилося тільки визначити колір осередків цієї картинки. Для цього і використовуються значення компонент. Найпростіший варіант — використання градацій сірого. У цьому випадку осередки, відповідні вузлам карти, в які потрапили елементи з мінімальними значеннями компонента або не потрапило взагалі жодного запису, будуть зображені чорним кольором, а осередки, в які потрапили записи з максимальними значеннями такого компонента, будуть відповідати осередку білого кольору. В принципі можна використовувати будь-яку градієнтну палітру для розмальовки.

Отримані розмальовки в сукупності утворюють атлас, що відображає розташування компонент, зв'язки між ними, а також відносне розташування різних значень компонент.

4.4.5. Відображення кластерів

Кластером буде група векторів, відстань між якими всередині цієї групи менше, ніж відстань до сусідніх груп. Структура кластерів при використанні алгоритму SOM може

бути відображена шляхом візуалізації відстані між опорними векторами (ваговими коефіцієнтами нейронів).

При використанні цього методу найчастіше використовується уніфікована матриця відстаней (u-matrix): обчислюється відстань між вектором ваг нейрону в сітці і його найближчими сусідами. Потім ці значення використовуються для визначення кольору, яким цей вузол буде зображений. Зазвичай використовують градації сірого, причому чим більше відстань, тим темніше зображений вузол. При цьому вузлам з найбільшими відстанями між ними та сусідами відповідає чорний колір, а навколишнім вузлам — білий.

4.5. Адаптивні методи кластеризації

Для того, щоб алгоритм кластеризації побудував набір кластерів, необхідно знати їх кількість. Міняючи цю кількість, можемо отримати множину рівноцінних результатів. Коли про кількість кластерів інформації немає, виникає проблема найкращого розбиття, а це нетривіальна задача. Полегшити задачу можна, якщо додати в алгоритм деякий адаптивний механізм відбору оптимального розв'язку. Вибір оптимального рішення буде базуватися на понятті якості кластеризації. Експертний вибір базується на оцінці спеціалістів, але така оцінка часто об'єктивно неможлива через великий об'єм і складність даних. Тому важливу роль грають формальні критерії оцінки якості кластеризації.

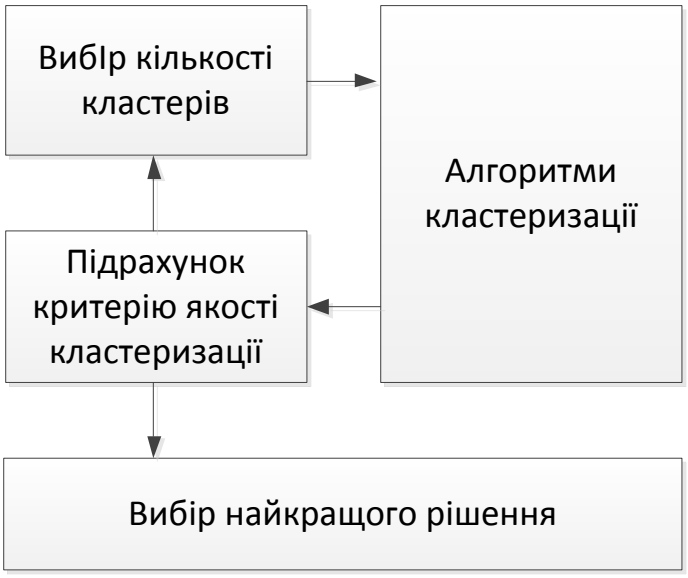


Рисунок 4.9. Загальна схема процедури адаптивної кластеризації

Ключовим елементом в адаптивній кластеризації є вибір критерію, по якому буде проводитися оцінка якості кластеризації (рисунок 4.9). Наведемо деякі з них.

Показники чіткості досягають максимуму при найбільш чіткому розбитті.

Коефіцієнт розбиття:

$$PC = \frac{\sum_{q=1}^{Q}\sum_{k=1}^{K} u_{ij}^2}{Q}, \quad PC \in \left[\frac{1}{k}, 1\right].$$

Модифікований коефіцієнт розбиття:

$$PC_M = \frac{\sum_{q=1}^{Q}\sum_{k=1}^{K} u_{ij}^2}{Q} - \frac{1}{K}, \quad PC \in \left[0, \frac{K-1}{K}\right]$$

Обидва коефіцієнти мають недолік: їх область значень напряму залежить від кількості кластерів.

Індекс чіткості:

$$CL = \frac{K \cdot PC - 1}{K - 1}, \quad CL \in [0,1]$$

Ентропія розбиття досягає мінімуму при найбільшій впорядкованості в системі.

Ентропія розбиття:

$$PE = \frac{\sum_{q=1}^{Q}\sum_{k=1}^{K} u_{qk} \ln(u_{qk})}{Q}, \quad PC \in \left[0, \ln K\right]$$

В загальному випадку розбиття на меншу кількість кластерів дасть менше значення ентропії, тому критерій видозмінюють.

Модифікована ентропія:

$$PE_M = \frac{\sum_{q=1}^{Q}\sum_{k=1}^{K} u_{qk} \ln(u_{qk})}{Q \ln K} = \frac{PE}{\ln K}, \quad PC \in \left[0,1\right]$$

Показник компактності та ізольованості:

$$CS = \frac{\sum_{q=1}^{Q}\sum_{k=1}^{K} u_{qk}^2 \cdot d^2(x_q, c_k)}{Q \cdot \min\left\{ d^2(c_i, c_j) \middle| i, j \in \overline{1, K}, \quad i \neq j \right\}}$$

Чим менше значення критерію — тим більш компактні кластери.

Індекс ефективності

Максимум цього критерію дає оптимальну кількість кластерів. Критерій будується з двох частин.

Міжкластерні відмінності (великі при оптимальній *K*):

$$\sum_{k=1}^{K}\sum_{q=1}^{Q} u_{qk}^2 d^2(c_k, \bar{x})$$

Відмінності всередині кластерів (малі при оптимальній *K*):

$$\sum_{k=1}^{K}\sum_{q=1}^{Q} u_{qk}^2 d^2(x_q, c_k)$$

Комбінуючи ці частини, отримаємо критерій:

$$PI = \sum_{k=1}^{K}\sum_{q=1}^{Q} u_{qk}^2 \left(d^2(c_k, \bar{x}) - d^2(x_q, c_k) \right)$$

Тут $\bar{x}$ — середнє арифметичне всіх вхідних векторів.

4.6. Нечіткі алгоритми кластеризації

Чітка кластеризація — кластеризація, яка кожен об'єкт відносить тільки одному кластеру. *Нечітка кластеризація* — кластеризація, при якій для кожного об'єкту x_i визначається $f_{i,k} \cdot f_{i,k}$ — дійсне значення, що показує степінь належності x_i кластеру j (рисунок 4.10).

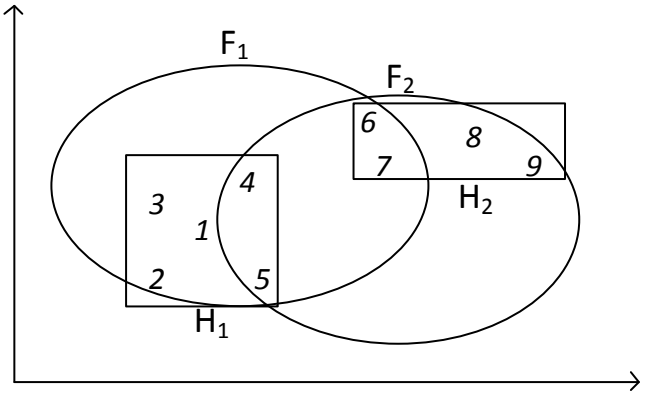


Рисунок 4.10. Приклад результату нечіткої кластеризації:

F_1 = {(1,0.9),(2,0.8), (3,0,7), (4,0.6),(5,0.55),(6,0.2),(7,0.2),(8,0.0), (9,0.0)}
F_2 = {(1,0.0),(2,0.0),(3,0.0),(4,0.1),(5,0.15),(6,0.4),(7,0.35),(8,0.1),(9,0.9)}

4.6.1. Алгоритм нечіткої кластеризації

Алгоритм наступний:

1. Вибрати початкове нечітке розбиття n об'єктів на k кластерів шляхом вибору матриці належності U розміру $n \times k$. Зазвичай U_{ij} належить [0,1].
2. Використовуючи матрицю U, знайти значення критерію нечіткої помилки.

Наприклад:

$$E^2(X,U) = \sum_{i=1}^{N}\sum_{k=1}^{K} U_{ik} \left\| x_i^{(j)} - c_k \right\|^2$$

де c_k — «центр мас» нечіткого кластера k:

$$c_k = \sum_{i=1}^{N} U_{ik} x_i$$

3. Перегрупувати об'єкти з метою зменшення цього значення критерію нечіткої похибки.
4. Повертатися в пункт 2 до того часу, поки зміни матриці U не стануть несуттєвими.

4.7. Застосування генетичних алгоритмів

Загальна схема даного підходу:

1. Вибрати початкову випадкову популяцію множини розширень і отримати оцінку якості для кожного рішення (зазвичай вона пропорційна $1/e^2$)
2. Створити і оцінити наступну популяцію рішень, використовуючи еволюційні оператори: Оператор вибору — з великою ймовірністю віддає перевагу хорошим рішенням; Оператор рекомбінації («кроссовер») — створює нове рішення на основі комбінацій із існуючих; Оператор мутації — створює нове рішення на основі випадкової незначної зміни одного із існуючих
3. Повторювати крок 2 до отримання потрібного результату.

Головною перевагою генетичних алгоритмів в даному застосуванні є те, що вони шукають глобально оптимальне рішення.

Більшість популярних алгоритмів оптимізації обирають початкове рішення, яке згодом міняють у ту чи іншу сторону. Таким чином отримуємо хороше розбиття, але не завжди — найоптимальніше.

Оператори рекомбінації і мутації дозволяють отримати рішення, сильно не схожі на вхідні — таким чином здійснюється глобальний пошук.

4.7.1. Основні визначення

Як відомо, ідея еволюції Дарвіна полягає в природньому відборі: найменш пристосовані організми помирають раніше і у великих кількостях, найбільш пристосовані виживають і дають потомство. Їх потомство вже опиняється в середньому більш пристосованими до оточуючого середовища, але серед них знову виділяються найбільш пристосовані особи, і так далі.

А генетичному алгоритмі відбувається те саме. Є деякий простір гіпотез, із яких ми маємо обрати кращу. Є функція пристосованості Fitness, яка визначає, наскільки добре той чи інший організм пристосований до «оточуючого середовища». Є набір генетичних операцій, за допомогою яких на світ з'являються нові особи. І, нарешті, є деякі цільові значення $\text{Fitness}_{\max}$, до якого ми прагнемо — як тільки мі його досягнемо, роботу з алгоритмом можна буде завершити.

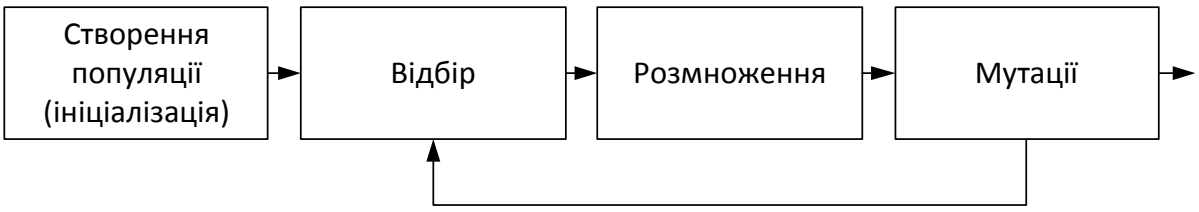


Рисунок 4.11. Загальна схема генетичного алгоритму

На рисунку 4.11 зображена загальна схема генетичного алгоритму Розглянемо її детальніше.

4.7.2. Схема генетичного алгоритму

1. Ініціалізація. Перед першим кроком створюємо випадковим чином деяку початкову популяцію; навіть якщо вона виявить неконкурентноздатною, генетичний алгоритм все одно достатньо швидко переведе її до життєздатної популяції. Таким чином,

на першому кроці можна особливо не намагатися зробити дуже вже пристосованих індивідів, достатньо, щоб на них можна було підрахувати функцію Fitness. Результатом першого кроку є популяція H, що складається з N індивідів.

2. Відбір. На етапі відбору необхідно із всієї популяції вибрати певну її долю, яка залишиться «в живих» на цьому етапі еволюції. Є різні способи проводити відбір, головне, що ймовірність виживання особи h має залежати від значення функції пристосованості Fitness(h). Сама доля вижившиs зазвичай є параметром генетичного алгоритму, і її просто задають попередньо. За результатами відбору із N індивідів популяції H має залишитись sN індивідів, які ввійдуть в результуючу популяцію H'.
3. Розмноження. Розмноження в генетичних алгоритмах звичайне — щоб зробити нащадка потрібно декілька батьків; зазвичай потрібно рівно два. Розмноження в різних алгоритмах визначається по-різному — воно, звичайно, залежить від представлених даних. Головна вимога — щоб потомок чи потомки мали можливість успадкувати риси обох батьків, «змішавши» їх деяким достатньо розумним чином. Щоб провести операцію розмноження, необхідно вибрати (1-s)p/2 пар гіпотез із H і провести з ними розмноження, отримавши по два потомки від кожної пари.

   Чому особи для розмноження зазвичай вибираються з усієї популяції H, а не з вижившиs на першому кроці? Справа в тому, що основна проблема багатьох генетичних алгоритмів — нестача різноманіття (diversity) в індивідах. Достатньо швидко виділяється один-єдиний генотип, який представляє собою локальний максимум, а згодом всі елементи популяції програють йому відбір і вся популяція «забивається» копіями цього індивіда. Є різні способи боротьбі з цим; один з них — вибір для розмноження не лише найбільш пристосованих, а всіх.
4. Мутації. До мутацій відноситься все те саме, що і до розмноження: є деяка доля мутантів m, що є параметром генетичного алгоритму, і на кроці мутації треба вибрати mN індивідів, а потім змінити їх у

відповідності із заздалегідь визначеними операціями мутації.

4.7.3. Генетичні операції

Спочатку припустимо, що всі елементи популяції зашифровані у вигляді бітових рядків. З бітовими рядками можна робити практично все, що завгодно; давайте розглянемо, як на рядках працюють стандартні генетичні операції.

З ініціалізацією все зрозуміло: породжувати випадкові бітові рядки нескладно. Для відбору конкретна форма представлення даних не сильно важлива: достатньо вирахувати функцію пристосованості Fitness.

При розмноженні особа має унаслідувати риси обох предків. На бітових рядках це можна реалізувати достатньо природним чином: зібрати результуючий рядок із частин рядків-батьків. Така операція називається *кроссовером* (crossover). Щоб зробити кроссовер двох рядків, треба вибрати певну *маску,* а згодом у відповідності до цієї маски вибрати ті чи інші біти батьків.

Визначення. Для двох рядків однакової довжини $x_1..x_n$ і $y_1..y_n$ результатом кроссовера з маскою $m_1...m_n$ є рядок $z_1..z_n$, де

$$z_i = \begin{cases} x_i, & \text{если } m_i = 1, \\ y_i, & \text{если } m_i = 0. \end{cases}$$

В залежності відтого, як вибирається маска, розглядаються декілька видів кроссоверу.

1. Одноточковий кроссовер (single-point crossover). Найпростіший вид кроссовера; в ньому випадковим чином вибирається одна позиція в рядочку і маска складається з одиниць, що лівіше цієї позиціх і нулів — правіше.
2. Двоточковий кроссовер (double-point crossover). Те саме, що і в попередньому випадку, але випадковим чином обираються дві позиціх, і між цими двома позиціями беруться біти одного з батьків, а зовні вибраних позицій — іншого.
3. Однорідний кроссовер (uniform crossover). Тут маска обирається випадковим чином, рівномірно.

Ми розглянули різні види кроссоверів на бітових рядках. Залишилось зрозуміти, якими можуть бути мутації. Найбільш

розповсюджена мутація — інвертування випадкового біту рядка. Її вже достатньо для багатьох за стосунків. Для того, щоб визначити інші типи мутацій, треба більш явно знати структуру предметної області.

4.7.4. Представлення даних

В попередньому параграфі ми припускали, що елементи популяції генетичного алгоритму — це рядки бітів. Дійсно, в такому випадку вся генетичні операції легко описуються і застосовуються, даючи розумні результати. Теоретично цього достатньо: зрозуміло, що будь-яку скільки завгодно складну структуру можна відобразити в слова алфавіту із двох символів. Але 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. Тоді класифікація, представлена у вигляді дерева, може бути записана у вигляді набору гіпотез, що відповідають листкам дерева. Як закодувати гіпотезу у вигляді рядка бітів? Перша думка — просто виділити по одному біту на кожен атрибут і на цільову функцію. Це кодування, очевидно, є найбільш ефективним, кожен рядок має однакову довжину і кожен рядок має сенс. Але його недоліки також очевидні: як представити гіпотезу, в якій не всі значення атрибутів специфіковані, а, навпаки, від деяких з них нічого не залежить? Якщо це неможливо, то будьяке дерево доведеться «розгортати» до повного набору гіпотез, що явно 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.

В даному випадку загальноприйнятим 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 означає, що дане значення не зустрічається в посилці правила, а одиниця — що зустрічається. В такій нотації якщо всі біти, що відповідають значенням даного атрибута рівні 1, це означає, що значення його (атрибута) не грає ніякої ролі для застосування цього правила. Для значення функції ми залишаємо один біт — правила, де цільова функція не визначена чи дозволено будь-яке її значення, позбавлені смислу. Таким чином, у випадку прикладу з гравцями «Динамо» чотири бінарних атрибута дадуть

8 біт посилки плюс один біт значення цільової функції. У цієї системи кодування залишилось лише 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. Тут є два шляхи. Перший — просто не дозволити появу таких гіпотез: повторяти кроссовер чи мутацію поки результат буде розумним. Другий — залишати такі гіпотези в популяції, але присвоювати їм нульове значення функції Fitness; тоді ці гіпотези будуть «виполоті» на наступному кроці відбору. Обидва підходи мають право на існування.

Наведемо приклад більш специфічної мутації, що залежить від представлення даних У випадку задачі класифікації досить успішною виявляється мутація видалення з посилки правила випадково вибраного атрибута (заміна будь-якого рядка, що відповідає цьому атрибуту, на рядок з одиниць). Така мутація відповідає узагальненню отриманих гіпотез і досить часто призводить до хороших результатів.

Хоча це все ще не зовсім те, що потрібно для розв'язання задачі. До цього часу ми описували окремі правила, окремі гілки дерева прийняття рішень. Але одна гіпотеза — це не одна гілка, а декілька (всі гілки дерева).

Як закодувати декілька правил? Тут, на щастя, відповідь проста: 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.

Набагато серйозніша проблема в тому, що в різних деревах різна кількість гілок і, відповідно, в різних гіпотезах буде різна кількість правил.

Як же робити кроссовер з рядками змінної довжини? Нам потрібно не лише мати можливість схрестити два рядки довжиною $l_1 n$ і $l_2 n$ кожна, де n — довжина правила, а li — кількість правил в кожній гіпотезі, але і отримати в результаті рядок довжиною l_n — буде неприємно отримати гіпотезу, в якій дробове число правил. Причому бажано, щоб це l вибиралось більш менш випадково з проміжку довжиною від 1 до l_1+l_2, а не дорівнювало завжди, наприклад max $\{l_1,l_2\}$.

Одне з можливих рішень таке: будемо використовувати двоточковий кроссовер так, щоб зберегти постійну відстань до границь правил. Тоді при заміні ділянки першої гіпотези на ділянку другої гіпотези, хоча їх довжини можуть бути різними, їх довжина по модулю n буде постійною Цей принцип простіше за все пояснити на прикладі.

Приклад. Кроссовер на рядках змінної довжини.

Припустимо, що в нашому прикладі приймають участь правила довжини 5, і ми випадково вибрали дві точки з першої гіпотези:

0[0101 110]10

Тоді у другій гіпотезі треба вибрати такі точки, щоб відстань від лівої точки до лівого краю правила і від правої точки до правого краю правила були такими ж. Наприклад, в правилі довжини 15 можуть бути варіанти:

1[10]11	01010	01110	1[1011	010]10	01110
1[1011	01010	011]10	11011	0[10]10	01110
11011	0[1010	011]10	11011	01010	0[11]10

Тепер кроссовер буде породжувати коректні гіпотези:

Вихідні рядки		Результат				
0[0101	110]01	01010				
1[10]11	01010	01110	10101	11011	01010	01110

4.7.5. Відбір

Ми вже розглянули всі елементи типового генетичного алгоритму на прикладі задачі класифікації, крім одного — власне природнього відбору. Більше того, мі навіть не визначити функцію пристосованості Fitness. Цю функцію ми будемо визначати наступним чином:

$$\text{Fitness}(h) = \left(\frac{\text{Correct}(h)}{\text{TotalExamples}}\right)^2,$$

де Correct(h) — кількість прикладів, правильно розкласифікованих гіпотезою h, TotalExamples — загальна кількість прикладів. Відзначимо, що такая функція пристосованності прекрасна справляється з проблемою гіпотез, що не відповідають ніяким конкретним прикладам — вони не зможуть класифікувати жодного прикладу, і їх пристосованність буде рівна 0.

Як же тепер вибрати найбільш пристосованих індивідів (нам треба відібрати sN індивідів з популяції розміру N)? Головна біда генетичних алгоритмів — недостатня гнучкість, яка призводить до того, що популяція стає сильно однорідною і не може вибратись з локального максимума. Тому просто вібирати верхні sN особи по функції Fitness — не найкраща ідея.

Зазвичай використовують один з двох найбільш популярних методів. Перший з них — метод рулетки (roulette wheel selection). В цьому методі у кожної гіпотези з ненульовою функцією пристосованності є шанс бути вибранною, і ймовірність її виживання пропорційна її функції пристосування: у кожної гіпотещи h_i ймовірність бути вибранною:

$$\Pr(h_i) = \frac{\mathrm{Fitness}(h_i)}{\sum_{j=1}^{N} \mathrm{Fitness}(h_j)}.$$

При цьому одна і та сама особа може бути вибрана декілька разів; алгоритм просто проводить вибір із заданою ймовірністю в продовж потрібної кількості ітерацій (sN). Цей метод — один із класичних, але на практиці використовується не так часто. По перше, такий семплінг достатньо дорогий обчислювально, а по-друге, цьому методу зазвичай не вистачає різноманіття в гіпоезах, які отримується на виході. Другу проблему можна частково вирішити, якщо перейти від методу рулетки до рангового методу. В ранговому методі гіпотези (індивіди) спочатку сортуються по пристосованності, а потім використовується такий же семплінг; але ймовірність бути вибраною у гіпотези прямо пропорційна не абсолютному значенню її пристосованності, а її рангу. Навіть якщо гіпотеза далеко відривається від своїх конкурентів по значенню Fitness, ранговий метод лише поставить її на перше місце, нехтуючи разницю в абсолютних значеннях.

Інший метод, що часто використовується — турнірний метод (toutnament selection). Спочатку виберемо випадково дві гіпотези. Потім з деякою фіксованою ймовірністю p (зазвичай $p>1/2$) виживає більш пристосована, з ймовірністью 1-p — менш пристосована гіпотеза.

Далі спробуємо поєднати все те, що до цього часу вивчали, в єдину схему алгоритму. Розглянемо приклад генетичного алгоритму, в якому вибрали метод рулетки в якості методу природнього відбору і інтвертування випадкового біта — в якості єдиного типу мутацій.

Genetic(N, p, s, m, Fitness, $Fitness_{max}$)

1. Створити *N* випадкових гіпотез $H = \{h_1, \ldots, h_n\}$
2. Для кожної гіпотези *h* із *H* підрахувати *Fitness(h)*
3. Поки $max_h Fitness(h) < Fitness_{max}$:
   - *3.1. H' = ø*
   - *3.2.* Випадково вибрати *sN* гіпотез із *H* та додати їх в *H'*. Ймовірність вибрати гіпотезу h_i:

$$\Pr(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^{N} \text{Fitness}(h_j)}$$

   - 3.3. Випадково обрати *((1-s)p)/2* пар гіпотез із *H* з тими ж ймовірностями. Для кожної пари (h_i, h_j) запустити операцію кросс-валідації и додаті її в результат *H'*
   - 3.4. Рівномірно вибрати *mN* випадкових гіпотез з *H'* і у кожній з них інвертувати випадковий біт
   - 3.5. *H = H'*
   - 3.6. Для кожної гіпотези *h* із *H* підрахувати *Fitness(h)*
4. Видати $argmax_h Fitness(h)$

Як відомо, Дарвін був не єдиним вченим, що запропонував схему еволюції. Жан Батист Ламарк в своїй книзі «Філософія зоології», що була опублікована у 1809 році, вже висловлював основні ідеї еволюційного розвитку видів. При житті Ламарка його думки не отримали підтримки серед наукової спільноти. Згадали про нього тільки після виходу книги Дарвіна, коли реального наукового значення його ідеї вже не мали — усе вказувало на те, що правим був Дарвін, а не Ламарк.

Основною ідеєю Ламарка було те, що організми міняються під дією оточуючого середовища і умов їх життєдіяльності. Головна відмінність від теорії Дарвіна в тому, що по Ламарку види можуть мінятися протягом свого життя, а не лише на генетичному рівні.

Хоча теорія Ламарка не витримують біологічної критики, хто сказав, що вона зовсім даремна для штучного інтелекту? Фактично, всі програми машинного навчання це ламаркіанські види, що навчаються протягом життя, а не за рахунок природнього відбору. І в рамках загальної парадигми генетич-

них алгоритмів ми також можемо поєднати природній відбір з навчанням «on-the-fly».

Для цього потрібно буде в якості гіпотез (індивідів популяції) використовувати який-небудь із апаратів машинного навчання. А згодом навчати їх на тестових прикладах. Зазвичай в якості такого апарата застосовують нейронні мережі: крім іншого, їх ще легко і навчити «трохи», так, щоб «генетичний код» не згубився від навчання; а, наприклад, дерева прийняття рішень після роботи алгоритма ID3 усі були б однакові.

Інша цікава думка, що прийшла в штучний інтелект з біології — це так називає мий ефект Болдуіна. Він пов'язаний із здатністю організму навчатися впродовж життя. Як вплине така можливість на еволюцію? Вона може «згладити» залежність пристосованості від генотипу: чим краще навчається особа, тим менше вона залежить від фенотипу. Наприклад, людина в процесі свого життя навчається таким різним речам, що в результаті від генотипу конкретної людини залежить в його житті дуже мало і зовсім не зрозуміло, які мутації можуть виявитися для людини сприятливими и бути закріпленими природнім відбором.

Болдуінівський ефект не є, як може здаватися, розвитком ламаркіанських ідей. Вивчені навики не передаються у спадок.

Щодо штучного інтелекту, болдуінівський ефект виражається в тому, що в природньому відборі приймають участь індивіди, що навчаються, причому їх здатність до навчання також є предметом відбору.

Приклад. Генетичний алгоритм, що реалізує ефект Болдуіна.

Опишемо конкретну схему побудови генетичного алгоритму на нейронних мережах.

Кожен елемент популяції — нейронна мережа глибини 2 с N входами, M нейронами на скритому рівні і N нейронами на виході. Таким чином у такого індивіда утворюються $(N + 1) \times M + (M + 1) \times N$ генів, що відповідають вагам нейронної мережі («плюс одиниці» утворюються в наслідок того, що у кожного нейрону є ще вага w_0, що додає константу до зваженої суми входів). Більш того, у кожного індивіда є ще стільки ж бінарних генів (генів пластичності), що визначаються, чи можемо ми відповідну вагу міняти в процесі навчання чи ні.

Кожна ітерація алгоритму складається з двох частин. Протягом першої частини індивіди навчаються на тестових

прикладах; при цьому навчання враховує також гени пластичності: для ваги w навчання відбувається по формулі

$$\Delta w = -\eta p \sum_{v \in V} \frac{\partial E_v(w)}{\partial w},$$

де η — швидкість начання, V — тестові приклади, E_v — функція помилки, а p — відповідний вазі w ген пластичності. Таким чином, якщо цей ген рівен 1, то вага може мінятися в процесі навчання, а якщо він равен 0, то вага протягом навчання залишається постійною.

На другому етапі навчання нейронні мережі приймають участь в генетичних операціях. Ці операції виконуються з функцією пристосованності:

$$\text{Fitness} = 1.0 - \frac{1}{N2^N} \sum_{v \in V} \sum_{i=0}^{N-1} (\text{Out}_{v,i} - \text{Target}_{v,i})^2,$$

де $\text{Target}_{v,i}$ — значення цільової функції на і-м виході мережі, а $\text{Out}_{v,I}$ — реальний вхід і-того нейрону мережі.

Відбір і розмноження ведуться досить простим способом: найгірший індивід популяції замінюється копією найкращої. Головна відмінність — у мутації. Виділяють мутації двох типів:

1. *мутації ваг ребер* міняють випадково вибрану вагу на випадкове значення з інтервалу *[-d,d];*
2. мутації генів пластичності — звичайне інвертування випадкового біту.

Висновки до розділу 4

Кластеризація: Кластери це корисний набір даних, який має певну форму у просторі. Щоб кластеризувати точки нам потрібно знати відстань між ними у цьому просторі. В ідеалі, точки з одного кластера знаходяться близько, а з різних — далеко.

Алгоритми кластеризації: Алгоритми кластеризації мають одну або дві форми. Ієрархічні алгоритми кластеризації починаються з усіма точками, як власними кластерами і близькі кластери об'єднуються. Алгоритми кластеризації із призначенням точок оглядають усі точки, і приєднують їх до кластера, до якого вони найкраще підходять.

Прокляття вимірності: Точки у багатовимірних евклідових просторах, як і точки у неевклідових просторах часто

поводяться непередбачувано. Дві неочікувані властивості цих просторі, це — однакова відстань між випадковими точками і ортогональність випадкових векторів.

Центроїди та кластероїди: В евклідових просторах, можна знайти середнє серед їх елементів — центроїду. В неевклідових просторах, неможливо знайти «середню» точку, тому серед усіх точок кластера обирається представляюча точка — кластероїда.

Вибір кластероїди: Існує багато способів визначити таку точку. Наприклад, точка з найменшою сумою відстаней до інших точок, найменшою сумою квадратів відстаней, найменша максимальна відстань до будь-якої точки.

Радіус та діаметр: Незалежно, чи є простір евклідовим, або ні, ми можемо знайти радіус кластера, як максимальну відстань від центроїди/кластероїди до будь-якої точки у кластері. Діаметр можна знайти як максимальну відстань між двома точками у кластері.

Ієрархічна кластеризація: Сімейство цих алгоритмів має багато варіантів, які діляться на два види. Перший — вибирати шляхи злиття кластерів. Другий — вирішити, коли зупинити процес злиття.

Вибір кластерів для злиття: Однією із стратегій вибору найкращої пари кластерів для злиття в ієрархічній кластеризації, це брати кластери з найближчими центроїдами/кластероїдами. Інщий підхід полягає у виборі кластерів з найближчими точками. Третій — використовувати середню відстань між точками у двох кластерах.

Зупинка процесу злиття: Ієрархічна кластеризація може продовжуватись, доки не буде досягнена задана кількість кластерів. Також можна об'єднувати кластери, доки не залишиться досить близьких кластерів для злиття. Ще один підхід — злиття, доки результуючий кластер матиме досить високу «щільність».

K-середні алгоритми: Це сімейство алгоритмів належить до алгоритмів призначення точок, і працює у евклідовому просторі. Припускається, що існує саме k кластерів. Після формування початкових k кластерів і їх центроїд, точки беруться одна за одною та приєднуються до кластера з найближчою центроїдою

Ініціалізація K-середнього алгоритму: Одним з підходів визначення початкових k центроїд є вибір першої точки, а потім

підібрати k-1 точок, що лежать якомога далі від неї. Або взяти малу вибірку точок і використовуючи ієрархічну кластеризацію, щоб об’єднати їх у k кластерів.

Підбір K у K-середніх алгоритмах: якщо невідома кількість кластерів, ми можемо використати бінарний пошук, роблячи k-середню кластеризацію з різними значеннями k.

BFR-алгоритм: Це варіант k-середнього алгоритму, для великого обсягу даних, що не вміщаються в основну пам'ять.

Представлення кластерів у BFR-алгоритмі: Точки читаються з диску по частинам. Кластери представлені в основній пам’яті кількістю точок, вектор сумою всіх точок, вектором суми квадратів компонент точок у кожному просторі.

Обробка точок у BFR-алгоритмі: Більшість точок в основній пам'яті будуть приписані до ближчого кластера, і параметри кластера зміняться з урахуванням цих точок.

CURE-алгоритм: Це алгоритм типу призначення точок. Він працює з кластерами будь-якої форми у евклідовому просторі. Він працює з даними, що не вміщаються в основну пам'ять.

Представлення кластерів у CURE : Алгоритм починається з кластеризації малого набору точок. Потім вибираються представляючи точки для кожного кластера. Згодом представляючи точки зсуваються до центроїди кластера, тому вони лежать у середині кластера.

Обробка точок у CURE: Після створення представляючи точок у кожному кластері, увесь набор точок можна зчитати зразу і призначений до кластерів. Точка приєднується до того кластера, прадставляюча точка якого лежить найближче.

GRGPF-алгоритм: Це алгоритм типу призначення точок. Він працює з даними, що не вміщаються в основну пам'ять і не потребує евклідового простору.

Представлення кластерів у GRGPF-алгоритмі : Кластер представляється кількість точок у ньому, кластероїдою, набором найближчих до неї точок і набором найдальших від неї точок.

Організація дерева кластерів у GRGPF-алгоритмі: Кластери представляються у вигляді дерева, те вузли є блоками диску і містять інформацію щодо багатьох кластерів. Листи містять представлення якомога більшої кількості кластерів, а вузли містять набор кластероїд суміжних кластерів.

Обробка точок у GPGRF: Після ініціалізації кластерів з набору точок, ми вводимо кожну точку у кластер з найближчою

кластероїдою. Через структуру дерева, ми починаємо з вершини і спускаємось вниз доки не дійдемо до листа, що містить кластер з найближчою кластероїдою, і приєднуємо точку до нього.

Представлення комірок у BDMO: Розмір комірки рівний кількості точок, що вона представляє. Комірка містить тільки представлення кластерів цих точок, не самих точок. Представлення кластерів включає кількість точок, центроїду\кластероїду та іншу необхідну інформацію для злиття.

Об'єднання комірок у BDMO-алгоритмі: Коли виникає потреба об'єднати комірки, ми знаходимо найкращі кластери для об'єднання, кожен зі своєї комірки і об'єднуємо їх парами. Якщо потік розвивається повільно, тоді послідовні комірки будуть мати однакові центроїди, тому такі об'єднання мають сенс.

Відповідь на запити у BDMO: Запит це довжина суфіксу у ковзному вікні. Ми беремо усі кластери в усіх комірках, що хоча б частково лежать у суфіксі і зливаємо їх за будь-якою стратегією.

Кластеризація використовуючи Map-Reduce: Ми можемо розділити дані на частини і кластеризувати кожну частину паралельно, використовуючи Map завдання. Кластери з кожного Map завдання можуть бути кластеризовані у єдиному Reduce завданні.

Питання до розділу 4

1. Сформулюйте задачу кластеризації
2. В чому різниця між класифікацією і кластеризацією?
3. До якого типу алгоритмів відноситься Single-link? Сформулюйте основні принципи та наведіть приклад використання
4. До якого типу алгоритмів відносяться Single-link та Complete-link? Сформулюйте основні принципи та наведіть приклад використання
5. Основні кроки алгоритму k-Means
6. В чому заключаэться метод найближчого сусіда?
7. Яка структура само організаційних карт Кохонена?
8. Яким чином застосовуються нейронні мережі при розв'язанні задачі кластеризації?
9. В чому заклечається принцип роботи алгоритму Clope?

Вправи до розділу 4

1. Довести, що відстань між двома навмання вибраними точками на лінії довжиною 1, буде дорівнювати 1/3.
2. Якщо вибрати дві точки з одиночного кубу навмання, якою буде їх Евклідова відстань?
3. Нехає існує d-вимірний Евклідовий простір. Враховуючи вектори, компоненти яких є або +1, або -1 в кожному вимірі. Довжина кожного вектору становить √d ,і добуток їх довжин (знаменник у формулі косинусу кута між ними) дорівнює d. Якщо ми виберемо кожну компоненту незалежно, і вони будуть скоріш за все або +1, або -1, який розподіл значень знаменника формули (суми добутків відповідних елементів кожного вектору)? Що ви можете сказати щодо очікуваного значення косинусу кута між векторами з ростом d?
4. Провести ієрархічну кластеризацію одновимірного набору точок 1, 4, 9, 16, 25, 36, 49, 64, 81, припускаючи, що кластери представлені своїми центроїдами і на кожному кроці об'єднуються кластери з найближчими центроїдами.
5. Як змінилася б кластеризація з прикладу 4.2.1.2 якби за відстань між кластерами ми вважали б:
   a. мінімальну відстань між будь-якими двома точками, кожна з різного кластеру;
   b. середнє відстаней між парами точок, кожна з різного кластеру.
6. Повторити кластеризацію з прикладу 4.2.1.2**,** вибираючи для об'єднання два кластери, результуючий кластер яких буде мати:
   a. найменший радіус;
   b. найменший діаметр.
7. Розрахувати щільність для кожного з трьох кластерів з рисунку 4.2.1.1, якщо «щільність» це кількість точок, розділена на:
   a. квадрат радіуса;
   b. діаметр.
8. Якою буде щільність згідно (a) та (b), з кластерів, які утворюються об'єднанням будь-яких двох з даних

трьох кластерів. Чи означають різниці у щільностях, що кластери потрібно або не потрібно об'єднувати?

9. Ми можемо знайти кластероїду кластера навіть у евклідовому просторі. Знайти кластероїди всіх трьох кластерів з рисунку 4.2.1.1, припускаючи, що критерій вибору кластероїди — точка з мінімальною сумою відстаней до інших точок кластера.
10. Нехай існує простір рядків у редакційних відстанях, які функціонують як міра відстані. Навести приклад набору рядків, такого, що вибір кластероїди шляхом мінімізації сум відстаней до інших точок, буде відрізнятися від кластероїди, обраної шляхом мінімізації максимальної відстані до інших точок.
11. Для точок рисунку 4.2.1.1, якщо обрати три початкові точки і першою обрати точку (3,4), які інші точки буде обрано?
12. Довести, що обираючи будь-які початкові точки на рисунку 4.2.1.1, якщо обирати три початкові точки методом з розділу 7.3.2 ми отримаємо точки в кожному з трьох кластерів.
13. Наведіть приклад набору даних та вибірки початкових *k* центроїд, таких, що коли точки приписуються до ближчого кластера, хоча б одна з початкових k точок буде приписана до іншого кластера.
14. Для трьох кластерів з рисунку 4.2.1.1:
    a. розрахувати представлення кластера як у BFR-алгоритмі, тобто розрахувати N, SUM та SUMSQ;
    b. розрахувати дисперсію та стандартне відхилення кожного кластера у двох вимірах.
15. Припустимо, що кластер тривимірних точок має стандартне відхилення 2,3 та 5, в трьох вимірах, в цьому порядку. Розрахуйте відстань Махаланобіса між (0,0,0) та (1,-3,4).
16. Розбити на 3 кластери за допомогою алгоритмів Single-link та Complete-link дані з рисунку 4.1.a-4.1.e.
17. Розбити на кластери за допомогою алгоритму k-means дані з рисунку 4.1.a-4.1.e.
18. Розбити на кластери методом найближчого сусіда дані з рисунку 4.1.a-4.1.e.

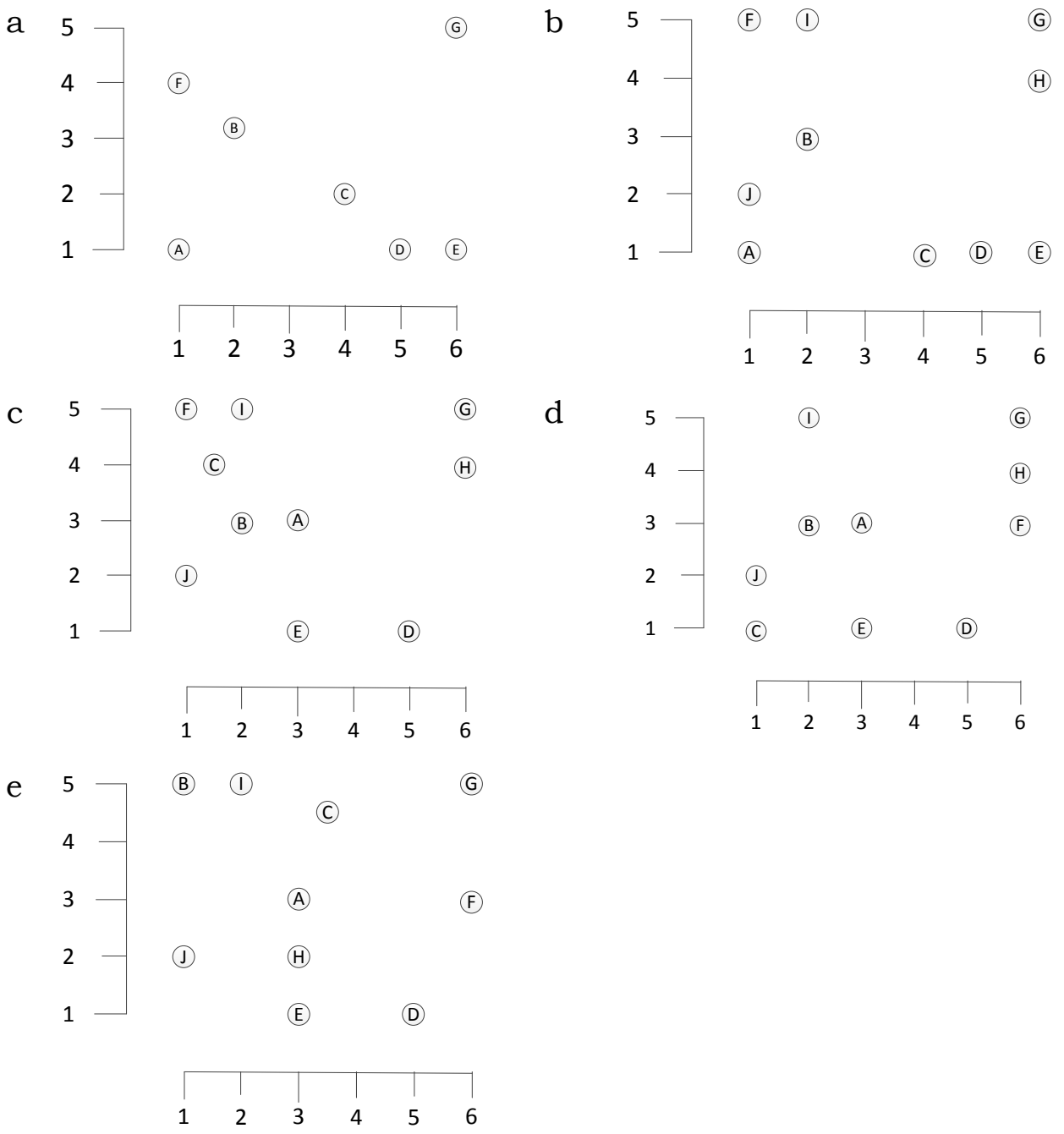


Рисунок 4.1. Вхідні дані для вправи

Використані джерела

1. Agrawa R. Fast Discovery of Association Rules / R. Agrawal, R. Srikant // In Proc. of the 20th International Conference on VLDB, Santiago, Chile, September 1994.
2. Agrawal R. Mining Associations between Sets of Items in Massive Databases / R. Agrawal, T. Imielinski, A. Swami // In Proc. of the 1993 ACM-SIGMOD Int'l Conf. on Management of Data, 207-216.
3. Agrawal R. Mining Generalized Association Rules / R. Srikant, R. Agrawal // In Proc. of the 21th International Conference on VLDB, Zurich, Switzerland, 1995.
4. Agrawal R. Mining quantitative association rules in large relational tables / R. Srikant, R. Agrawal //In Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, June 1996.
5. Babcock B. Maintaining variance and k-medians over data stream windows / B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan // Proc. ACM Symp. on Principles of Database Systems, pp. 234–243, 2003.
6. Baestaens D.E. Neural Network Solution for Trading in Financial Markets / Dirk Emma Baestaens, Willem Max Van Den Bergh, Douglas Wood // Pitman publishing
7. BaseGroup Labs. Технологии анализа даннх // Електронний ресурс. Режим доступу: https://basegroup.ru
8. Berry Michael J. A. Data Mining techniques: for marketing, sales, and customer relationship management / Michael J.A. Berry, Gordon Linoff // 2nd ed.
9. Bilmes J. A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models / Tech. Report ICSI-TR-97-021. — 1997.
10. Bradley P.S. Scaling EM (Expectation-Maximization) Clustering to Large Databases / Paul S. Bradley, Usama M. Fayyad, Cory A. // Microsoft Research. — 1999.
11. Bradley P. Scaling Clustering Algorithms to Large Databases / Bradley P., Fayyad U., Reina C. //Proc. 4th

Int'l Conf. Knowledge Discovery and Data Mining, AAAI Press. — Menlo Park, Calif. — 1998.

12. Brin S. Dynamic Itemset Counting and Implication Rules for Market Basket Data // In Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press, New York, 1997.
13. Buntine W. A theory of classification rules. Електронний ресурс. — 1992.
14. Dean J. MapReduce: Simplified Data Processing on Large Clusters / Jeffrey Dean and Sanjay Ghemawat // OSDI'04: Sixth Symposium on Operating System Design and Implementation. --- San Francisco, CA. --- December, 2004.
15. Ganti V. CACTUS — Clustering Categorical Data Using Summaries. / Ganti V., Gerhke J., Ramakrishan R. // In Proc KDD'99. — 1999.
16. Ganti V. Clustering large datasets in arbitrary metric spaces / V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French // Proc. Intl. Conf. on Data Engineering, pp. 502–511, 1999.
17. Garcia-Molina H. Database Systems: The Complete Book Second Edition / H. Garcia-Molina, J. D. Ullman, and J. Widom // Prentice-Hall, Upper Saddle River, NJ, 2009.
18. Guha S. CURE: An Efficient Clustering Algorithm for Large Databases / Guha S., Rastogi R.,Shim K. // Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Pres. — New York. — 1998.
19. Hipp J. Algorithms for Association Rule Mining — A General Survey and Comparison / J. Hipp, U. Guntzer, and G. Nakaeizadeh // In Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000.
20. Hristev R. M. Artifical Neural Networks
21. Huang Z. A fast clustering algorithm to claster very large categorical data sets in Data Mining. Research Issues on on Data Mining and KDD. — 1997.
22. Huang Z. Clustering large data sets with mixed numeric and categorical values / In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining. — 1997.

23. Hyafil, L. Constructing optimal binary decision trees is NP–complete / Hyafil, L., Rivest, R. // Inf. Process. Lett. 5(1), 15–17 (1976)
24. Larose D.T. Discovering knowledge in data: an introduction to Data Mining // Електронний ресурс. Режим доступу на 31.08.2017: https://www.jstatsoft.org/article/view/v016b01/v16b01.pdf
25. Leskovec J. Mining of Massive Datasets / Jure Leskovec Anand Rajaraman, Jeffrey David Ullman // Stanford Univ. — 2010.
26. Machine Learning, Neural and Statistical Classification / Editors D. Mitchie et.al. // 1994.
27. Milenova B. Clustering large databases with numeric and nominal values using orthogonal projections / Milenova, B., Campos, M. // Oracle Data Mining Technologies. — 2002.
28. Murthy S. Automatic construction of decision trees from data: A Multi-disciplinary survey // 1997.
29. Navathe S. An Efficient Algorithm for Mining Association Rules in Large Databases / Savasere, E. Omiecinski, and S. Navathe //In Proc. 21st Int'l Conf. Very Large Data Bases, Morgan Kaufmann, San Francisco, 1995.
30. Palmer M. Data is the New Oil / Michael Palmer // CMO News, Michael Palmer, Reinventing Marketing, Technology's Impact, The Customer. — 2006. — Електронний ресурс. — Режим доступу на 31.08.2017: http://ana.blogs.com/maestros/2006/11/data_is_the_new.html
31. Park J.S. An Effective HashBased Algorithm for Mining Association Rules / J.S. Park, M.-S. Chen, and S.Y. Philip // In Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press, New York, 1995.
32. Quinlan J.R. C4.5: Programs for Machine learning / J. Ross Quinlan // Morgan Kaufmann Publishers. — 1993.
33. Quinlan, J.R. Induction of Decision Trees // Mach. Learn. 1. — V.1. — Mar. 1986. — P. 81–106.

34. Wang K. Clustering transactions using large items / Wang, K., Xu, C.. Liu, B. // In Proc. CIKM'99. — Kansas, Missouri. — 1999.

35. Zhang T. Birch: An Efficient Data Clustering Method for Large Databases / Zhang T., Ramakrishnan R., Livny M. // Proc. ACM SIGMOD Int'l Conf. Management of Data, ACM Press. — New York. — 1996.

36. Айвазян С.А. Прикладная статистика и основы эконометрики / С.А. Айвазян, В.С Мхитарян // М. Юнити, 1998

37. Барсегян А.А. Методы и модели анализа данных: OLAP и Data Mining / А.А. Барсегян, М.С. Куприянов, В.В. Степаненко, И.И. Холод // СПб. — 2004.

38. Ганти В. Добыча данных в сверхбольших базах данных / В. Ганти, Й. Герке, Р. Рамакришнан // Открытые системы. — №9-10. — 1999.

39. Дюк В.А. Применение технологий интеллектуального анализа данных в естественнонаучных, технических и гуманитарных областях / В. А. Дюк, А. В. Флегонтов, И. К. Фомина // Известия Российского государственного педагогического университета им. А.И. Герцена. Область наук: Информатика. — 2011. — С. 77-84. --- Режим доступу на 31.08.2017: https://cyberleninka.ru/article/n/primenenie-tehnologiy-intellektualnogo-analiza-dannyh-v-estestvennonauchnyh-tehnicheskih-i-gumanitarnyh-oblastyah

40. Стариков А. Нейронные сети — математический апарат // Режим доступа: http://www.basegroup.ru/library/analysis/neural/math

41. Шеннон К. Работы по теории информации и кибернетике // М. Иностранная литература, 1963.

42. Методичні матеріали, лекції та статті по темі із відкритих джерел

ЗМІСТ

Навчальне видання

Марченко Олександр Олександрович
Россада Тетяна Володимирівна

Актуальні проблеми Data Mining

Навчально-методичний посібник

Редактор С. Бреусов