МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Кафедра прикладної математики та
моделювання складних систем

ДИПЛОМНА РОБОТА
тема роботи:

МОДЕЛЮВАННЯ ПРОБЛЕМ КЛАСИФІКАЦІЇ З ПІДБОРОМ ОПТИМАЛЬНОГО АЛГОРИТМУ

Завідувач випускової кафедри:	д. ф.-м. н., проф. О. В. Лисенко
Керівник роботи:	к. ф.-м. н. Т. С. Сушко
Рецензент:	к. ф.-м. н., с. н. с. О. Б. Лисенко
Виконавець:	студентка факультету ЕлІТ, гр. ПМ.м.н-71 Ю. О. Шевченко

Затверджено на засіданні кафедри "__" _________ 20__ р., протокол №

Суми 2019 р.

Реферат

Дипломна робота магістра складається з 82 с., 7 малюнків, 99 джерел, 2 додатків.

Мета – побудова математичної моделі проблем класифікації з підбором оптимального алгоритму.

В роботі проведено аналітичний огляд сучасних методів побудови класифікаторів. Наведено математичні основи побудови моделей такими методами: метод головних компонент (PCA), дерево прийняття рішення (CART), нейронні мережі прямого поширення(FF NN), наївний баєсів класифікатор (NB), метод опорних векторів (SVM).

Побудовано власний алгоритм класифікації даних, що ґрунтується на методі k-найближчих сусідів. Розроблено програмний комплекс, що включає в себе перехресну перевірку, реалізації розглянутих методів та вибір фінального методу використовуючи жадібний алгоритм.

КЛАСИФІКАЦІЯ, КЛАСИФІКАТОР, K-CROSS VALIDATION, ПЕРЕХРЕСНА ПЕРЕВІРКА, ГОЛОВНІ КОМПОНЕНТИ, БАЄС, NAIVE BAYES, НЕЙРОННІ МЕРЕЖІ, NEURAL NETWORK, FEED FORWARD, BACKPROPOGATION, ДЕРЕВО ПРИЙНЯТТЯ РІШЕННЯ, DECISION TREE, МЕТОД ОПОРНИХ ВЕКТОРІВ, SUPPORT VECTOR MACHINE.

ЗМІСТ

1. Аналітичний огляд 4
2. Постановка задачі 9
3. Методики 10
3.1 Метод головних компонент 10
3.2 Перехресна перевірка 14
3.3 Метод опорних векторів 17
3.4 Наївний баєсів класифікатор 21
3.4 Штучні нейронні мережі та алгоритм зворотного поширення помилки 26
3.5 Дерево прийняття рішення 32
3.7 Модифікований метод *k*-найближчих сусідів 38
4. Практична реалізація 43
4.1 Туристичне страхування в Сінгапурі 43
4.2 Дощ в Австралії 50
4.3 Оцінка автомобіля 55
4.4 Загальний випадок 57
Висновок 62
Список використаної літератури 63
ДОДАТОК А 69
ДОДАТОК Б 78

1. Аналітичний огляд

Інтелектуальний аналіз даних - це процес аналізу наборів даних у новий спосіб пошуку несподіваних залежностей та зв’язків. Зв’язки та залежності, отримані за допомогою інтелектуального аналізу даних, часто називають моделями або шаблонами, які витягують неявні, невідомі та потенційні корисні відомості з даних. Це потрібно для того, щоб передбачити майбутні тенденції і поведінку, прийняти проактивні рішення і відповісти на бізнес-питання, на які витрачається занадто багато часу[1]. Були вивчені різні методи інтелектуального аналізу даних для обробки та аналізу декількох типів шаблонів даних, де найбільш популярними завданнями інтелектуального аналізу даних є класифікація, узагальнення, видобування правил асоціації та кластеризація[2].

Для забезпечення значущих результатів інтелектуального аналізу даних необхідно розуміти оброблювані дані. На підходи інтелектуального аналізу даних зазвичай впливають декілька факторів, такі як шуми, які включають нульові або нетипові значення. Відповідно до зміни характеру даних, що підлягають обробці, були введені деякі розширення для інтелектуального аналізу даних: отримання просторових даних; отримання веб-ресурсів та отримання веб-контенту, поведінки користувачів та конкретна інформація в Інтернеті відповідно та отримання великий даних, що є розвиненою галуззю Big Data аналітики[3–6].

Для побудови успішної моделі інтелектуального аналізу даних існують основні фази, які слід слідувати, як показано в резюме, представленому на малюнку 1.

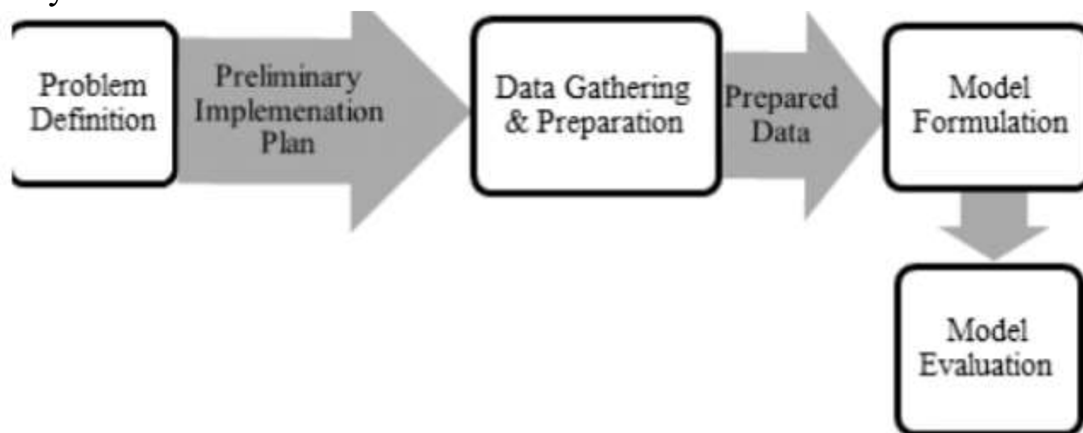

Рис. 1: Фази побудови моделі інтелектуального аналізу даних

По-перше, необхідно визначити, яку проблему має вирішити інтелектуальний аналіз даних, що зосереджується на розумінні цілей та вимог. Після того, дані мають бути структуровані у вигляді завдання інтелектуального аналізу даних і повинен бути розроблений початковий план

реалізації. По-друге, починається фаза збору та підготовки даних, де мають бути вказані джерела даних і формат даних. Далі, на етапі моделювання можна застосовувати різні методи. Таким чином, для однієї категорії процесу інтелектуального аналізу даних може бути застосовано багато підходів з урахуванням різноманітності форм даних. Нарешті, починається процес побудови моделі та її оцінка. На цій фазі застосовуються різні методи моделювання для оцінки того, наскільки модель задовольняє цілям[7, 8].

Задача класифікації визначає клас, до якого належить нове спостереження. Якщо дано набір навчальних даних, який має кілька атрибутів, модель ідентифікується як функція значень інших атрибутів. Для цього необхідний навчальний набір правильно ідентифікованих спостережень. Класифікація застосовується для автоматичного присвоєння об'єктам попередньо визначеним класам, наприклад: класифікувати транзакції кредитних карток як легітимні чи шахрайські, або класифікувати новини, як фінанси, розваги, спорт тощо.

Найбільш поширеними підходами, які використовувалися при вирішенні проблем реального світу, є методи, що базуються на деревах прийняття рішень[9], нейронних мережах[10], а також на методі опорних векторів (SVM), наївному баєсовому класифікаторі та *k*-найближчих сусідів (KNN)[10].

Термін «дерево класифікації і регресії» (CART) - це парасольковий термін, який використовується для позначення обох вищевказаних процедур, вперше введених Breiman у 1984 році[11]. Методи, що базуються на деревах прийняття рішень, виводять значущі правила для інтелектуальної інформації, щоб використовувати їх для класифікації даних. Одним з найбільш популярних алгоритмів є CART (дерево класифікації та регресії), ID3 (ітераційний дихотомізатор 3) і C4.5[9]. ID3 і CART були винайдені самостійно приблизно в один і той же час (між 1970 і 1980 рр.), але дотримуються аналогічного підходу для навчання дерева рішень.

Нейронні мережі, що також використовуються в класифікації через їх здатність витягувати значущу інформацію з складних даних, застосовуються для виявлення закономірностей, які вважаються занадто складними для виконання людиною. Вони складаються з мереж «нейронів», що мають подібну нейронну структуру, як у мозку. Вперше модель нейронної мережі на основі математики та алгоритмів була створена в 1943 році вченими МакКалох та Пітц[12], що отримала назву «порогова логіка». Ця модель відкрила шлях для дослідження нейронних мереж на два підходи. Один з підходів зосереджувався на біологічних процесах у мозку, а інший - на застосуванні нейронних мереж для штучного інтелекту. Ця робота привела

до роботи над нервовими мережами та їх зв'язком з кінцевими автоматами[13].

Ключовим тригером для відновлення інтересу до нейронних мереж і навчання був алгоритм зворотного розповсюдження похибки (Вербос, 1975), зробивши навчання багатошарових мереж здійсненним і ефективним. У середині 1980-х років паралельна розподілена обробка набирає популярність під назвою «зв'язування». Румельхарт і МакКлелланд (1986) описали використання зв'язності для моделювання нейронних процесів[14].

Проблема зникаючого градієнта впливає на багатошарові мережі прямого поширення, які використовують зворотне поширення, а також рекурентні нейронні мережі (RNN)[15, 16]. Оскільки помилки поширюються від шару до шару, вони стискаються експоненціально з числом шарів, що перешкоджає налаштуванню ваг нейронів, що ґрунтується на цих помилках, особливо впливаючи на глибокі мережі. Щоб подолати цю проблему, Шмідхубер прийняв багаторівневу ієрархію мереж (1992), попередньо підготувавши один рівень одночасно без нагляду і тонкого налаштування за допомогою зворотного поширення похибки. Бенке (2003) спирався тільки на знак градієнта (Rprop)[17] у проблемах, таких як реконструкція зображення і локалізація обличчя. Хінтон та інші (2006) запропонували вивчити високорівневе представлення, використовуючи послідовні шари бінарних або реальних латентних змінних з обмеженою машиною Больцмана[18], щоб змоделювати кожен шар. Як тільки достатньо багато шарів було вивчено, глибока архітектура може бути використана в якості генеративної моделі, відтворюючи дані при виборі моделі[19, 20].

Раніше проблеми в підготовці глибоких нейронних мереж успішно вирішувалися такими методами, як безконтрольна попередня підготовка, а доступна обчислювальна потужність зростала за рахунок використання графічних процесорів і розподілених обчислень. Нейронні мережі були розгорнуті у великих масштабах, зокрема в проблемах розпізнавання зображень і візуалізації. Це стало відоме як "глибоке навчання".

SVM визначають межі прийняття рішень залежно від концепції рішень, що розділяє об'єкти, що належать до різних класів. Оригінальний алгоритм SVM був винайдений Володимиром Н. Вапником і Олексієм Я. Червоненкісом. Однак у 1992 році Бернхард Е. Босер, Ізабель М. Гайон та Володимир Н. Вапник запропонували спосіб створення нелінійних класифікаторів, застосовуючи прийом ядра (спочатку запропонований Айзерменом та ін.[21]) до максимально роздільної гіперплощини[22]. Отриманий алгоритм формально схожий, за винятком того, що кожен скалярний добуток замінюється нелінійною функцією ядра. Це дозволяє

алгоритму підігнати гіперплощину максимальної роздільності в перетвореному просторі ознак. Нинішній стандарт реалізації (м'яка роздільність) був запропонований Корінною Кортесом і Вапником у 1993 році і опублікований у 1995 році[23].

Наївний класифікатор Баєса є прямолінійним імовірнісним класифікатором, який застосовує теорему Баєса і передбачає сильні незалежні зв'язки між ознаками[24]. Незважаючи на їх наївний дизайн і, очевидно, спрощені припущення, наївні класифікатори Баєса працювали досить добре в багатьох складних реальних ситуаціях. У 2004 році аналіз проблеми баєсівської класифікації показав, що існують обгрунтовані теоретичні причини для очевидно неправдоподібної ефективності наївних класифікаторів Баєса[25]. Проте всебічне порівняння з іншими алгоритмами класифікації в 2006 році показало, що класифікація Баєса перевершується іншими підходами, такими як дерева або випадкові ліси[26]. Ренні та інші обговорювали проблеми з поліноміальним припущенням в контексті класифікації документів і можливих шляхів полегшення цих проблем, включаючи використання tf – idf ваг замість сировинних частот і нормалізації довжини документа, для отримання наївного класифікатора Баєса, який є конкурентоспроможним з вектором підтримки машини[27].

Класифікація *K*-найближчого сусіда (*kNN*) є одним з найбільш фундаментальних і простих методів класифікації і повинна бути одним з перших варіантів для дослідження класифікації, коли є мало або взагалі немає попередніх знань про розподіл даних. Класифікація *K*-найближчих сусідів була розроблена з необхідності виконувати дискримінантний аналіз, коли достовірні параметричні оцінки щільності ймовірностей невідомі або їх важко визначити. У неопублікованій доповіді ВПС США з авіаційної медицини в 1951 році Фікс і Ходжес ввели непараметричний метод класифікації об'єктів, який з тих пір став відомим правилом *k*-найближчого сусіда[28]. Пізніше, у 1967 році, були розроблені деякі формальні властивості правила k-найближчого сусіда; Наприклад, було показано, що для $k = 1, n \to \infty$ похибка класифікації *k*-найближчих сусідів обмежена зверху подвійною частотою помилок Баєса[29]. Після встановлення таких формальних властивостей було проведено довгу лінію дослідження, включаючи нові підходи до відхилення[30], уточнення відносно частоти помилок Баєса, були використані дистанційні зважені підходи[31, 32], методи м'якого обчислення[33] і нечіткі методи[34, 35].

Використовується також метод повторного інкрементного обрізання для зменшення помилок (RIPPER) для класифікації об'єктів за допомогою набору правил "if". . . then . . . else…”. Ця методика генерує модель, що

складається з правил ресурсів, побудованих для виявлення майбутніх прикладів відхилень [5], [23], [35].

Ансамблі класифікаторів представляють концепцію агрегування декількох класифікаторів як новий підхід до підвищення продуктивності класифікаторів, які працюють індивідуально[36]. Ці класифікатори можуть базуватися на різноманітних методологіях класифікації, досягаючи різних показників правильно класифікованих об’єктів. Bagging є прикладом для ансамблів класифікаторів для bootstrap агрегації. Це метод генерування ансамблю моделей, побудованих з bootstrap реплікованих об’єктів[37]. Випадковий ліс - це ще один ансамбль класифікаторів, що складається з багатьох дерев рішень, і виводить вузол класу окремими деревами. Для багатьох наборів даних він дає високоточний результат і може ефективно працювати на великих базах даних[10]. Ліси обертання, з іншого боку, використовують виділення ознак з метою побудови класифікаторських ансамблів. Для базового класифікатора тренувальні дані створюються шляхом поділу набору ознак на підмножини k, а потім застосування аналізу основних компонентів (PCA) на кожному піднаборі. Основні компоненти зазвичай резервуються для підтримки інформаційної мінливості. Тому для формулювання нових ознак базового класифікатора використовуються нові отримані осі k[38].

PCA був винайдений у 1901 році Карлом Пірсоном[39] як аналог теореми про головну вісь в механіці; пізніше він був самостійно розроблений і названий Гарольдом в 1930-х роках[40]. Залежно від сфери застосування, його також називають дискретним перетворенням Карунена-Лоева (KLT) в обробці сигналів, перетворенням Готелінґа у багатовимірному контролі якості, належної ортогональної декомпозиції (POD) в машинобудуванні, розкладанням сингулярних значень (SVD) матриці X [41], розкладанням власних значень (EVD) матриці $X^T X$ в лінійній алгебрі, факторним аналізом[42], теоремою Екарт-Янга[43], або емпіричними ортогональними функціями в метеорологічній науці, розкладанням емпіричних власних функцій[44], емпіричним компонентним аналізом[45], спектральним розкладанням в шумі і вібрації, емпіричним модальним аналіз в структурній динаміці. PCA тісно пов'язаний з факторним аналізом. Факторний аналіз, як правило, включає в себе більше доменних специфічних припущень про основну структуру і вирішує власні вектори з дещо іншою матрицею. PCA також пов'язаний з канонічним кореляційним аналізом (CCA). CCA визначає системи координат, які оптимально описують перехресну коваріацію між двома наборами даних, тоді як PCA визначає нову ортогональну систему координат, яка оптимально описує дисперсію в одному наборі даних[46, 47].

2. Постановка задачі

Розглянемо набір даних df=[m x n]. Основна ідея полягає в побудові моделі, що буде класифікувати дані з максимальною точністю. Для досягнення цієї мети будуть використані наступні алгоритми:

- Метод опорних векторів
- Метод наївної класифікації
- Нейронні мережі прямого поширення помилки
- Дерево прийняття рішення
- Бустинг-метод

Для отримання адекватної моделі, вона повинна бути уточнена за допомогою перехресної перевірки. Дана перевірка розбиває дані на декілька фрагментів та проводить навчання на кожному фрагменті окремо. Це забезпечує повне покриття даних та отримання більш точних результатів.

Враховуючи, що набори даних являють собою велику кількість ознак, частіше за все ці ознаки пов’язані і немає необхідності використовувати їх усі. Тому є необхідність у використанні компонентного аналізу для зменшення кількості ознак. Виділення головних компонент зменшить розмірність даних, але при достатній кількості компонент можна досягти майже повного збереження інформативності даних.

Одним з найпопулярніших методів є метод *k*-найближчих сусідів. Основним недоліком цього методу є необхідність підбору кількості сусідів. Ми пропонуємо модифікацію цього методу шляхом узагальнення усіх даних. Ми пропонуємо замінити процес вибору кількості сусідів на побудову вектору ваг, що буде враховувати відстань до об’єкту.

На основі отриманої моделі буде виділено основні критерії вибору моделі будуть побудовані основні залежності, що покажуть як обирати метод класифікації не проводячи велику кількість додаткових досліджень.

3. Методики

3.1 Метод головних компонент

Аналіз головних компонент (PCA) – це статистична процедура, яка використовує ортогональні перетворення для перетворення набору можливих корельованих змінних (сутностей, кожен з яких приймає різні числові значення) у набір значень лінійно некорельованих змінних, званих головними компонентами. Якщо спостерігаються n спостережень з змінними p, то кількість окремих головних компонентів є $\min(n-1, p)$.

PCA в основному використовується як інструмент у дослідницькому аналізі даних і для створення прогностичних моделей. PCA може бути здійснена шляхом розкладання власних значень коваріаційної (або кореляційної) матриці даних або сингулярного розкладання матриці даних, як правило, після етапу нормалізації вихідних даних. Нормалізація кожного атрибута складається з середнього центрування: вирахування кожного значення даних з вимірюваної середньої величини її змінної, так що його емпіричне середнє значення (середнє) дорівнює нулю, і, можливо, нормалізувати дисперсію кожної змінної, щоб зробити її рівною 1. Результати PCA зазвичай обговорюються з точки зору компонентних балів, які іноді називають коефіцієнтами факторів (перетворені значення змінних, що відповідають конкретній точці даних), і навантаження (вага, за якою кожна стандартизована початкова змінна повинна бути множиною для отримання оцінки компонента)[48]. Якщо показники компонентів стандартизовані до одиничної дисперсії, навантаження повинні містити дисперсію даних у них (а це величина власних значень). Якщо оцінки компонентів не є стандартизованими (тому вони містять дисперсію даних), то навантаження повинні бути масштабовані одиницями ("нормалізовані"), і ці ваги називаються власними векторами; вони є косинусами ортогонального обертання змінних у головні компоненти або назад.

PCA є найпростішим з багатоваріантних методів на основі власних векторів. Часто його функціонування можна розглядати як розкриття внутрішньої структури даних таким чином, що найкраще пояснює відхилення в даних. Якщо багатоваріантний набір даних візуалізується у вигляді набору координат у високовимірному просторі даних (1 вісь на змінну), PCA може надати користувачеві більш низьку розмірну картину, проекцію цього об'єкта при перегляді з найбільш інформативної точки зору. Це робиться, використовуючи лише перші кілька основних компонентів, щоб зменшити розмірність трансформованих даних.

Нехай дано матрицю змінних X розміром ($I \times J$), де I – число образів (рядків), а J – число незалежних змінних (стовпців), яких, як правило, багато.

(J>>1). В методі головних компонент використовуються нові змінні $t_a \left(a = 1,...,A \right)$, які є лінійною комбінацією початкових змінних $x_j \left(j = 1,...,J \right)$

$$t_a = \sum_{k=1}^{J} p_{ak} x_k \quad (1)$$

За допомогою нових змінних матриця X розкладається на добуток двох матриць T і P:

$$X = TP^T + E = \sum_{a=1}^{A} t_a p_a^T + E, \quad (2)$$

де матриця T є матрицею *оцінок* (scores) розмірністю (I×A), матриця P є матрицею *навантажень* (loadings) з розмірністю (J×A), а матриця E – матриця *залишків* розмірністю (I×J).

Нові змінні t_a називаються головними компонентами (Principal Components), тому і сам метод називається методом головних компонент (PCA). Число A називається числом головних компонент (PC). Ця величина є меншою ніж число змінних J і числа образів I.

Важливою властивістю PCA є ортогональність (незалежність) головних компонент. Тому матриця оцінок T не перебудовується при збільшенні числа компонент, а до неї просто додається ще один стовпець, що відповідає новому напрямку. Теж відбувається і з матрицею навантажень P.

Найчастіше для побудови PCA оцінок і навантажень, використовується рекурентний алгоритм NIPALS, який на кожному кроці обчислює одну компоненту.

Спочатку вихідна матриця X центрується і перетворюється в матрицю $E_0, a = 0$. Далі застосовують наступний алгоритм:

1)Обираємо початковий вектор t

2) $p' = \frac{t' E_a}{t' t}$

3) $p = \frac{p}{\left(p' p \right)^{\frac{1}{2}}}$

4) $t = E_a \frac{p}{p' p}$

5) Перевіряємо збіжність. Якщо умова збіжності не виконується, то повертаємось до пункту 2.

Після обчислення черговий (a-ої) компоненти, вважаємо $t_a = t,\ p_a = p$. Для отримання наступного компонента треба обчислити залишки $E_{a+1} = E_a - tp^T$ і застосувати до них той же алгоритм, замінивши індекс а на (а + 1). Після того, як побудовано простір з головних компонент, ми можемо спроектувати данні на головні компоненти та отримати X_{new}, іншими словами – визначити матрицю оцінок T_{new}:

$$T_{new} = X_{new}P \qquad (3)$$

Метод головних компонент тісно пов'язаний з іншим розкладом - по сингулярним значенням, SVD. В останньому випадку вихідна матриця X розкладається в добуток трьох матриць

$$X = USV^T \qquad (4)$$

де U - матриця, утворена ортонормованими власними векторами u_r матриці XX^T, що відповідають власним значенням λ_r:

$$XX^T u_r = \lambda_r u_r, \qquad (5)$$

матриця V - матриця, утворена ортонормованими власними векторами v_r матриці $X^T X$:

$$X^T X v_r = \lambda_r v_r, \qquad (6)$$

матриця S - позитивно визначена діагональна матрицю, елементи якої є сингулярними значеннями $\sigma_1 \geq ... \geq \sigma_R \geq 0$ рівні квадратним кореням з власних значень λ_r:

$$\sigma_r = \sqrt{\lambda_r}. \qquad (7)$$

Тоді зв'язок між PCA і SVD визначається наступними простими співвідношеннями

$$\begin{aligned} T &= US \\ P &= V \end{aligned} \qquad (8)$$

Цільовий підхід до оцінки числа головних компонент по необхідній долі загальної дисперсії формально може бути використаний завжди, але неявно вважається, що немає розділення на «сигнал» та «шум», та будь-яка

наперед задана точність має сенс. Тому часто більш продуктивною є евристика, що базується на гіпотезі про наявність «сигналу» з порівняно малою розмірність та відносно великою амплітудою та «шуму» з великою розмірністю та відносно малою амплітудою. З цієї точки зору метод головних компонент працює як фільтр: сигнал міститься, в основному, в проекції на перші головні компоненти, а в решті пропорція шуму набагато вище, ніж сигналу.

Виникає питання: як оцінити кількість необхідних компонент, якщо співвідношення «сигнал»/«шум» невідомо?

Найпростіший та найстаріший метод відбору головних компонент дає правило Кайзера: значущими є ті компоненти, для яких

$$\lambda_i > \frac{1}{n} trC ,$$

де C - матриця, складена з власних значень λ_i, тобто

$$\lambda_i > \bar{\lambda}, \ \bar{\lambda} = \frac{1}{n}\sum_{j=1}^{n}\lambda_j ,$$

(середню вибіркову дисперсію координат вектора даних). Правило Кайзера працює ефективно у простих випадках, коли є декілька головних компонент, які набагато більше середньої вибіркової дисперсії, а інші власні числа менше нього. В більш складних випадках метод може позначати велику кількість компонент як значимі.

Одним з найбільш популярних евристичних підходів до оцінки кількості головних компонент є правило зламаної тростини[49]. Набір нормованих на одиничну суму власних значень $\left(\frac{\lambda_i}{trC}, (i = 1,...,n)\right)$ порівнюється з розподілом довжин обломків тростини одиничної довжини, зламаної в $n-1$-й випадковій точці (точки розлому обираються незалежно та рівнорозподілені по довжині тростини).

3.2 Перехресна перевірка

Перехресна перевірка, яка іноді називається оцінкою обертання, [50–52] являє собою різні методи валідації моделі для оцінки узагальнення незалежних даних, використовуючи статистичні процедури. В основному перехресна перевірка використовується для оцінки того, на скільки точно навчена модель буде працювати на практиці. У задачі прогнозування, зазвичай, дається модель набору відомих даних, на основі яких проводиться навчання (навчальний набір даних), і набору даних невідомих даних (або перших побачених даних), на основі яких перевіряється модель (називається набором даних перевірки чи тестуванням). [53, 54] Мета перехресної перевірки полягає в тому, щоб перевірити здатність моделі передбачати нові дані, які не були використані при навчання, для того, щоб позначити проблеми, такі як перенавчання або зміщення вибору [55], і дати уявлення про те, як модель буде узагальнюватися дані до незалежного набору (тобто як поведе себе модель з реальними даними).

На першому етапі перехресна перевірка включає розбиття вибірки даних на додаткові підмножини: навчальну вибірку та тестовий набір. Щоб зменшити варіабельність, у більшості методів виконується кілька етапів перехресної перевірки з використанням різних розділів, а результати перевірки об'єднуються (наприклад, усереднюються) по етапах, щоб дати оцінку прогностичної продуктивності моделі.

У підсумку, перехресна перевірка об'єднує (середні) показники придатності в прогнозі, щоб отримати більш точну оцінку продуктивності прогнозування моделі.

Припустимо, у нас є набір даних та модель, яка може бути навчена на цих даних. Якщо ми візьмемо незалежну вибірку даних перевірки з тієї ж самої популяції, що й навчальні дані, то, як правило, виявиться, що модель не відповідає даних перевірки, а також відповідає навчальним даним. Розмір цієї різниці, ймовірно, буде великим, особливо коли розмір набору навчальних даних невеликий, або коли кількість параметрів у моделі велике. Перехресна перевірка - це спосіб оцінити розмір цього ефекту.

Якщо модель правильно визначена, то вона може бути розглянута при м'яких припущеннях, а саме очікуване значення MSE для навчального набору складає $\frac{n-p-1}{n+p+1}<1$ помножене на очікуване значення MSE для валідаційного набору[56] (очікуване значення приймається за розподілом навчальних множин). Таким чином, якщо ми обчислимо MSE на навчальному наборі, ми отримаємо оптимістично упереджену оцінку того, наскільки добре модель

буде відповідати незалежному набору даних. Ця упереджена оцінка називається оцінкою підбору вибірки, тоді як оцінка перехресної перевірки є оцінкою поза зразком.

Таким чином, перехресна перевірка є загальноприйнятим способом прогнозування продуктивності моделі на недоступних даних з використанням чисельного розрахунку замість теоретичного аналізу.

Можна виділити два типи перехресної перевірки, вичерпну та невичерпну перехресну перевірку.

Вичерпними методами перехресної перевірки є методи перехресної перевірки, які вивчають і перевіряють всі можливі способи розділення вихідного зразка на набір для навчання та перевірки.

Перехресна перевірка залишкового виходу (LpO CV) передбачає використання p спостережень як набору валідації, а решту спостережень - як навчальний набір. Це повторюється на всіх способах розрізання вихідного зразка на валідаційному наборі p спостережень і навчальному наборі.

Перехресна перевірка LpO вимагає підготовки та перевірки моделей C_p^n разів, де n - кількість спостережень у вихідному зразку, C_p^n - біноміальний коефіцієнт. Для $p > 1$ і для навіть помірно великого n LpO CV може стати обчислювально непридатним. Наприклад, з n = 100 і p = 30 $C_{30}^{100} = 3 \cdot 10^{25}$.

Перехресна перевірка залишків (LOOCV) є часним випадком перехресної перевірки залишення з p = 1.

Перехресна перевірка LOO вимагає меншого часу обчислення, ніж перехресна перевірка LpO, тому що існує тільки C_1^n наборів для перевірки. Проте, n проходів можуть все ще вимагати досить велику кількість часу для обчислення, в цьому випадку можуть бути більш доцільними інші підходи, такі як к-кратна перехресна перевірка.

Невичерпні методи перехресної перевірки не обчислюють усі способи розбиття вихідної вибірки. Ці методи є наближенням перехресної перевірки залишків.

У *k*-кратної перехресної перевірки вихідний зразок розподіляється випадковим чином на *k* груп з однаковим розміром. З *k* груп є одна група, яка зберігається як тестовий набір, а решта *(k – 1)* груп використовуються як навчальні дані. Процес перехресної перевірки повторюється *k* разів, при цьому кожна з *k* груп використовується точно один раз як тестовий набір. Результати *k* можуть бути усереднені для отримання єдиної оцінки. Перевагою цього методу перед іншими методами є те, що всі спостереження використовуються як для навчання, так і для перевірки, і кожне спостереження використовується для валідації точно один раз. Зазвичай

використовується 10-кратна перехресна перевірка[57], але в загальному випадку *k* залишається нефіксованим параметром.

У стратифікованій *k*-кратній перехресної перевірці набори вибирають таким чином, щоб середнє значення відгуку було приблизно однаковим у всіх наборах. У випадку двійкової класифікації, це означає, що кожен набір містить приблизно однакові пропорції двох типів міток класу.

У методі затримки ми випадковим чином призначаємо точки даних двома множинами d_0 і d_1, які зазвичай називаються навчальним набором і тестовим набором відповідно. Розмір кожного з наборів є довільним, хоча зазвичай тестовий набір менше, ніж навчальний набір. Потім тренуємо (будуємо модель) на d_0 і тестуємо (оцінюємо її продуктивність) на d_1.

У типовій перехресній перевірці результати кількох прогонів модельного тестування усереднюються разом. На відміну від цього, метод затримки включає в себе лише один запуск. Його слід використовувати з обережністю, оскільки без такого усереднення декількох прогонів можна досягти дуже хибних результатів. Індикатор точності прогнозування (F*), як і коефіцієнти моделі, мають тенденцію бути нестабільними, оскільки вони не будуть згладжуватися кількома ітераціями.

Повторна перевірка випадкових під-вибірок, також відома як перехресна перевірка Монте-Карло[58], випадково розподіляє набір даних на дані навчання та перевірки. Для кожного такого поділу модель підходить до навчальних даних, і точність прогнозування оцінюється з використанням даних перевірки. Потім результати усереднюються за розбиттям. Перевагою цього методу є те, що частка поділу навчання / перевірки не залежить від кількості ітерацій. Недоліком цього способу є те, що деякі спостереження ніколи не можуть бути обрані в наборі валідації, тоді як інші можуть бути обрані більш ніж один раз. Іншими словами, підмножини перевірки можуть перекриватися. Цей метод також демонструє варіацію Монте-Карло, що означає, що результати будуть змінюватися, якщо аналіз повторюється з різними випадковими розбиттями.

3.3 Метод опорних векторів

Метод опорних векторів (SVM) є методом навчання з учителем, що використовуються для класифікації та регресійного аналізу. Враховуючи набір навчальних прикладів, кожен з яких позначений як приналежний до однієї або іншої з двох категорій, алгоритм навчання SVM створює модель, яка класифікує нові набори даних. Модель SVM представляє дані у вигляді точок у просторі, відображених таким чином, що приклади точки різних категорій мають чіткий максимально широкий розрив. Потім нові приклади відображаються в цьому самому просторі і передбачається, що вони належать до категорії, заснованої на тому, на якій стороні розриву вони потрапляють.

На додаток до виконання лінійної класифікації, SVM можуть ефективно виконувати нелінійну класифікацію, використовуючи так зване перетворення ядра.

Більш формально, SVM створює гіперплощину або набір гіперплощин, який може бути використаний для класифікації, регресії або інших задач, таких як виявлення викидів. Інтуїтивно, хороше поділ досягається за допомогою гіперплощини, яка має найбільшу відстань до найближчої точки тренувальних даних будь-якого класу (так званий функціональний запас), оскільки, як правило, чим більше відстань від гіперплощини до об’єкту, тим менше помилка узагальнення класифікатора[59].

Нехай дано початковий набір даних $(\bar{x}_1, y_1),...,(\bar{x}_n, y_1)$, де y_i приймає значення 1 або -1 в залежності від того, до якого класу належить точка $\bar{x}_i$. Кожна точка $\bar{x}_i$ являє собою p-вимірний вектор. Ми хочемо знайти гіперплощину, що розділяє об’єкти на різні класи та знаходиться на максимальній відстані від 2х найближчих точок, що належать різним класам.

Будь-яку гіперплощину можна записати як набір точок $\bar{x}$, що задовольняє рівняння $\bar{w} \cdot \bar{x} - b = 0$, де $\bar{w}$ - це (не обов'язково нормалізований) вектор нормалі до гіперплощини. Параметр $\frac{b}{|w|}$ визначає зміщення гіперплощини від початку координат вздовж нормального вектора $\bar{w}$.

Якщо навчальні дані лінійно відокремлюються (жорстка класифікація), ми можемо виділити дві паралельні гіперплощини, які розділяють два класи даних, так що відстань між ними буде максимальною. Область, обмежена цими двома гіперплощинами, називається "відступом", а гіперплощина, що має максимальну відступ до найближчих точок сусідніх класів, являє собою

гіперплощину, що лежить між ними. З нормалізованим або стандартизованим набором даних ці гіперплощини можна описати рівняннями

- $\overline{w} \cdot \overline{x} - b = 1$ (для точок, що знаходяться вище гіперплощини та мають $y_i = 1$)
- $\overline{w} \cdot \overline{x} - b = -1$ (для точок, що знаходяться нижче гіперплощини та мають $y_i = -1$)

Геометрично відстань між цими двома гіперплощинами дорівнює $\frac{2}{\left\|\overline{w}\right\|}$, тому для максимізації відстані між площинами нам необхідно мінімізувати $\left\|\overline{w}\right\|$. Для того, щоб забезпечити правильно розташування точок відносно прямої, введено додаткове обмеження:

$$y_i \cdot \left(\overline{w} \cdot \overline{x_i} - b\right) \geq 1,\ \forall i = 1,...,n \qquad (9)$$

Отже, задача звелась до наступної задачі оптимізації:
«Мінімізувати $\left\|\overline{w}\right\|$ за умови, що $y_i \cdot \left(\overline{w} \cdot \overline{x_i} - b\right) \geq 1,\ \forall i = 1,...,n$»

Важливим наслідком цього геометричного опису є те, що гіперплощину максимальної відстані повністю визначають ті $\overline{x_i}$, які лежать найближчі до неї. Ці точки $\overline{x_i}$ називаються векторами підтримки.

Щоб поширити SVM на випадки, коли дані не є лінійно відокремленими (м'яка класифікація), ми вводимо функцію завісних втрат,

$$\max\left(0, 1 - y_i \cdot \left(\overline{w} \cdot \overline{x_i} - b\right)\right) \qquad (10)$$

Ця функція дорівнює нулю, якщо обмеження (9) виконується, іншими словами, якщо $\overline{x_i}$ лежить на правильній стороні поля. Для даних на неправильній стороні поля значення функції пропорційно відстані до гіперплощини.

Тому нам необхідно мінімізувати наступну функцію

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i\left(\overline{w} \cdot \overline{x}_i - b\right)\right)\right] + \lambda\left\|\overline{w}\right\|^2 \qquad (11)$$

де параметр λ визначає компроміс між збільшенням розміру поля та забезпеченням того, що $\overline{x}_i$ лежить на правильній стороні поля. Таким чином, для досить малих значень λ, другий доданок функції втрати стане незначним, отже, він буде вести себе подібно до лінійно роздільної задачі.

У 1992 році буде винайдено нелінійний класифікатор, використовуючи перетворення ядра. Фінальний алгоритм майже нічим не відрізнявся від початкового, окрім того, що скалярний добуток компонент був замінений на нелінійну ядрову функцію. У цьому випадку, класифікатор також залишався гіперплощиною, але вже в просторі, утвореному ядровою функцією, причому у початковому просторі задача лінійної класифікації не може бути вирішена.

Слід зазначити, що робота в багатовимірному просторі збільшує похибку узагальнення, хоча при достатній кількості зразків алгоритм досі працює добре[60].

Найчастіше використовуються наступні ядра:

- Поліноміальне ядро:

$$k\left(\overline{x}_i,\overline{x}_j\right)=\left(\overline{x}_i\cdot\overline{x}_j+c\right)^d;$$

- Гауссова радіальна базисна функція:

$$k\left(\overline{x}_i,\overline{x}_j\right)=\exp\left(-\gamma\left\|\overline{x}_i-\overline{x}_j\right\|^2\right),\gamma>0;$$

- Гіперболічний тангенс:

$$k\left(\overline{x}_i,\overline{x}_j\right)=\tanh\left(\kappa\overline{x}_i\cdot\overline{x}_j+c\right)$$

для деяких $\kappa>0,c<0$.

Обчислимо м’який класифікатор SVM мінімізуючи наступний виразу

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0,1-y_i\left(\overline{w}\cdot\overline{x}_i-b\right)\right)\right]+\lambda\left\|\overline{w}\right\|^2 \tag{12}$$

Класичний підхід, який передбачає зведення задачі (12) до задачі квадратичного програмування.

Мінімізація (12) може бути переписана як обмежена задача оптимізації з диференційованою цільовою функцією наступним чином.

Для кожного $i=1,...,n$ введемо змінну $\zeta_i=\max\left(0,1-y_i\left(\overline{w}\cdot\overline{x}_i-b\right)\right)$.

Зазначимо, що ζ_i - це найменше невід'ємне число, що задовольняє наступну нерівність $y_i\left(\overline{w}\cdot\overline{x}_i-b\right)\geq 1-\zeta_i$.

Таким чином, ми можемо переписати задачу оптимізації наступним чином:

$$\begin{aligned}&\frac{1}{n}\sum_{i=1}^{n}\zeta_i+\lambda\left\|\overline{w}\right\|^2\to\min\\&y_i\left(\overline{w}\cdot\overline{x}_i-b\right)\geq 1-\zeta_i\\&\zeta_i\geq 0,\forall i=1,...,n\end{aligned} \tag{13}$$

Використовуючи метод множників Лагранжа, отримуємо спрощену задачу

$$f\left(c_1,...,c_n\right)=\sum_{i=1}^{n}c_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}y_i c_i\left(\overline{x}_i\cdot\overline{x}_j\right)y_j c_j\to\max$$

$$\sum_{i=1}^{n}c_i y_i=0 \qquad (14)$$

$$0\le c_i\le\frac{1}{2n\lambda},\forall i=1,...,n.$$

Ця задача називається двоїстою задачею. Оскільки двоїста задача максимізації є квадратичною функцією відносно c_i, що підлягає лінійним обмеженням, вона ефективно розв'язується алгоритмами квадратичного програмування.

Для знаходження початкових параметрів використаємо наступні співвідношення:

$$\overline{w}=\sum_{i=1}^{n}c_i y_i\overline{x}_i$$

$$b=\overline{w}\cdot\overline{x}_i-y_i \qquad (15)$$

Серед інших методів хотілось би відмітити метод релевантних векторів (Relevance Vector Machine, RVM). На відміну від SVM метод релевантних векторів обчислює ймовірність, з якою об’єкт належить даному класу. Тобто, якщо SVM каже "x належить до класу A", то RVM каже "x належить до класу A з ймовірністю p та до класу B з ймовірністю 1-p".

3.4 Наївний баєсів класифікатор

Наївні баєсові класифікатори відносяться до сімейства простих ймовірнісних класифікаторів. Вони засновані на теоремі Баєса з наївними припущеннями про незалежність між ознаками.

Цей класифікатор знайшов своє широке використання на початку 1960 років[61] і залишається до сих пір популярним у різних областях. З відповідною попередньою обробкою даний класифікатор є конкурентоспроможним з більш просунутими методами, включаючи SVM. Він також знаходить застосування в отриманні автоматичного медичному діагнозі[62].

Наївні баєсові класифікатори є високо масштабованими, вимагаючи ряду параметрів лінійних за кількістю змінних (ознак) у навчальній задачі. Навчання з максимальною ймовірністю можна здійснити, оцінивши вираз закритої форми[63], який приймає лінійний час, а ітераційне наближення, яке займає велику кількість часу та використовується для багатьох інших типів класифікаторів.

Існує ціле сімейство алгоритмів, заснованих на загальному принципі: всі наївні баєсові класифікатори припускають, що значення конкретної ознаки не залежить від значення будь-якої іншої ознаки, враховуючи змінну класу. Наприклад, плід може вважатися яблуком, якщо він червоний, круглий і близько 10 см в діаметрі. Наївний класифікатор Баєса розглядає кожну з цих ознак незалежно від вірогідності того, що цей плід є яблуком, незалежно від можливих кореляцій між кольором, округлості та особливостями діаметра.

Для деяких типів ймовірнісних моделей, наївні баєсові класифікатори а можуть бути дуже ефективними в керованому навчання з учителем. У багатьох практичних застосуваннях оцінка параметрів для наївних баєсових моделей використовує метод максимальної ймовірності; іншими словами, можна працювати з наївною моделлю Баєса, не використовуючи баєсовську ймовірність або будь-які баєсовські методи.

Перевагою наївного Баєсу є те, що для оцінки параметрів, необхідних для класифікації, потрібно лише невелика кількість навчальних даних.

Взагалі, наївний баєсів класифікатор - це модель умовної ймовірності: якщо класифікуватись об'єкт, який представляє собою вектор $x = (x_1, ..., x_n)$ з n ознаками, то класифікатор знайде для цього об'єкту ймовірність $p(C_k | x_1, ..., x_n)$ для кожного з K можливих класів C_k[64]. Проблема з вищенаведеною постановкою полягає в тому, що якщо кількість ознак n велике або якщо ознака може приймати велику кількість значень, то базування такої моделі на таблицях ймовірностей неможливо, тому виникає необхідність переформулювати модель, щоб зробити її більш

сприйнятливою. Використовуючи теорему Баєса, умовна ймовірність може бути розкладена як

$$p(C_k|x) = \frac{p(C_k)\,p(x \mid C_k)}{p(x)}. \tag{16}$$

На практиці існує інтерес тільки в чисельнику цього дробу, тому що знаменник не залежить від C і заданих значень ознак x_i, тому знаменник є фактично постійним . Чисельник еквівалентний спільній ймовірнісній моделі $p(C_k, x_1, ..., x_n)$, яку можна переписати наступним чином, використовуючи правило ланцюга для повторних застосувань визначення умовної ймовірності:

$$\begin{aligned} p(C_k, x_1, ..., x_n) &= p(x_1, ..., x_n, C_k) \\ &= p(x_1 \mid x_2, ..., x_n, C_k) = p(x_2, ..., x_n, C_k) \\ &= p(x_1 \mid x_2, ..., x_n, C_k)\,p(x_2 \mid x_3, ..., x_n, C_k)\,p(x_3, ..., x_n, C_k) \\ &= \ldots \\ &= p(x_1 \mid x_2, ..., x_n, C_k)\,p(x_2 \mid x_3, ..., x_n, C_k)\ldots p(x_{n-1} \mid x_n, C_k)\,p(x_n \mid C_k)\,p(C_k) \end{aligned}$$

Врахуємо «наївні» припущення про умовну незалежність ознак: припустимо, що кожна ознака x_i умовно незалежна від будь-якої іншої функції x_j для $i \neq j$та обраного класу C_k. Це призводить до наступного наближення:

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) \approx p(x_i \mid C_k). \tag{17}$$

Таким чином, загальна модель може бути виражена наступним чином:

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &\approx p(C_k)\,p(x_1 \mid C_k)\,p(x_2 \mid C_k)\ldots p(x_n \mid C_k) \\ &= p(C_k)\prod_{i=1}^{n} p(x_i \mid C_k). \end{aligned} \tag{18}$$

Це означає, що за наведеними вище припущеннями про незалежність, умовний розподіл над змінною класу C_k:

$$p(C_k \mid x_1, ..., x_n) = \frac{1}{Z}\,p(C_k)\prod_{i=1}^{n} p(x_i \mid C_k) \tag{19}$$

де змінна $Z = p(x) = \sum_k p(C_k)\, p(x \mid C_k)$ є коефіцієнтом масштабування, що залежить тільки від ознак $x_1, \ldots, x_n$.

Відповідний класифікатор є функцією, яка присвоює мітку класу C_k для деяких k таким чином:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\arg\max}\, p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k). \tag{20}$$

Щоб оцінити параметри розподілу ознаки, необхідно припустити розподіл або генерувати непараметричні моделі для ознак з навчального набору[65]. Припущення щодо розподілу ознак називаються моделлю подій наївного баєсового класифікатора. Для дискретних функцій, подібних тим, що зустрічаються в класифікації документів (включаючи фільтрацію спаму), популярними є мультиноміальний розподіл та розподіл Бернуллі. Ці припущення призводять до двох різних моделей, які часто плутають[24, 66].

При роботі з безперервними даними припускається, що значення, пов'язані з кожним класом, розподіляються відповідно до розподілу Гауса. Наприклад, припустимо, що навчальні дані містять безперервний атрибут x. Спочатку сегментуємо дані класом, а потім обчислимо середнє значення і дисперсію у кожному класі x. Нехай μ_k і σ_k^2 - середнє значення та дисперсія x, асоційовані з класом C_k. Припустимо, що ми маємо деяку кількість спостережень v. Ймовірнісний розподіл v для класу C_k, $p(x = v \mid C_k)$, можна обчислити, підставивши v у рівняння для нормального розподілу:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}} \tag{21}$$

У моделі з великою кількістю різних значень ознак, вектори ознак представляються у вигляді частот, з якими деякі події були згенеровані $(p_1,...,p_n)$, де p_i - ймовірність того, що відбувається подія i (або K таких значень у багатокласовому випадку). Вектор ознак $x=(x_1,...,x_n)$ тоді представляється у вигляді гістограми, де x_i - частоти виникнення події і в конкретному випадку. Цю модель зазвичай використовують для класифікації документів, з подіями, що представляють собою появу слова в одному документі. Ймовірність спостереження за гістограмою х задається наступним чином:

$$p(x \mid C_k)=\frac{\left(\sum_i x_i\right)!}{\prod_i x_i\,!}\prod_i p_{ki}^{x_i} \tag{22}$$

Мультиноміальний наївний класифікатор Баєса стає лінійним, коли він виражається в логарифмічному просторі[27]:

$$\begin{aligned}\log p(C_k \mid x) &\propto \log\left(p(C_k)\prod_{i=1}^{n} p_{ki}{}^{x_i}\right)\\ &=\log p(C_k)+\sum_{i=1}^{n} x_i\cdot\log p_{ki}\\ &=b+w_k{}^T x\end{aligned} \tag{23}$$

де $b=\log p(C_k)$, $w_{ki}=\log p_{ki}$.

Якщо задане значення класу і ознака ніколи не зустрічається разом у навчальних даних, то оцінка на основі частоти буде дорівнює нулю. Це проблематично, тому що воно знищить всю інформацію в інші ймовірності, коли вони множиться. Тому часто бажано включати корекцію малих вибірок у всіх оцінках ймовірностей, так що жодна ймовірність ніколи не буде рівною нулю. Цей спосіб регуляризації наївних баєсів називається згладжуванням Лапласа.

У багатовимірній моделі події Бернуллі ознаки являють собою незалежні булеві змінні. Як і мультиноміальна модель, ця модель використовується для задач класифікації документів[66], але замість частотних характеристик використовуються бінарні характеристики термінів. Якщо x_i, що виражає виникнення або відсутність i-го терміну в тексті, то ймовірність, що надається класу C_k, задається[66]

$$p(x \mid C_k) = \prod_{i=1}^{n} p_{ki}^{x_i} (1 - p_{ki})^{(1 - x_i)} \qquad (24)$$

де p_{ki} - ймовірність класу C_k, що генерує термін x_i. Ця модель особливо популярна для класифікації коротких текстів. Вона має перевагу явного моделювання відсутності термінів.

3.4 Штучні нейронні мережі та алгоритм зворотного поширення помилки

Штучні нейронні мережі – це обчислювальні системи, нечітко натхненні біологічними нейронними мережами, які утворюють мозок тварин[67, 68]. Сама нейронна мережа не є алгоритмом, а скоріше основа для багатьох різних алгоритмів машинного навчання для спільної роботи та обробки складних вхідних даних[69]. Такі системи «навчаються» виконувати завдання, розглядаючи приклади, як правило, не запрограмовані з якимись конкретними правилами. Наприклад, при розпізнаванні зображень вони можуть навчитися ідентифікувати зображення, які містять кішок, аналізуючи приклади зображень, які були позначені вручну як кіт на малюнку чи немає, використовуючи результати для ідентифікації кішок в інших зображеннях. Вони автоматично генерують ідентифікаційні характеристики з навчального матеріалу, який вони обробляють.

ШНМ базуються на колекції з'єднаних з'єднань або вузлів, які називаються штучними нейронами, які моделюють нейрони в біологічному мозку. Кожне з'єднання, подібно синапсам в біологічному мозку, може передавати сигнал від одного штучного нейрона до іншого. Штучний нейрон, який отримує сигнал, може обробляти його, а потім передавати інформацію іншим нейронам, пов'язаним з ним.

У звичайних реалізаціях ШНМ сигнал при з'єднанні між штучними нейронами є дійсним числом, і вихід кожної штучної нейронної системи обчислюється деякою нелінійною функцією суми її входів. Зв'язки між штучними нейронами називаються «ребрами». Штучні нейрони і ребра, як правило, мають вагу, що регулюється в процесі навчання. Штучні нейрони можуть мати такий поріг, що сигнал надсилається тільки, якщо агрегатний сигнал перетинає цей поріг. Як правило, штучні нейрони агрегуються в шари. Різні шари можуть виконувати різні види перетворень на своїх входах.

Початковою метою створення ШНМ було вирішення проблем так само, як і людський мозок. Однак з плином часу увага переходить до виконання конкретних завдань, що призводять до відхилень від біології. Штучні нейронні мережі використовувалися на різних завданнях, включаючи комп'ютерне бачення, розпізнавання мовлення, машинний переклад, фільтрацію соціальних мереж, ігрові та відеоігри та медичну діагностику.

Алгоритм зворотного поширення – це метод, що використовується в штучних нейронних мережах для обчислення градієнта, необхідного для розрахунку ваг[67]. У алгоритмі зворотного поширення помилка обчислюється на виході і поширюється назад по шарах мережі. Цей алгоритм зазвичай використовується для навчання глибоких нейронних мереж[12, 69].

Цей алгоритм є узагальненням дельта-правила до багатошарової прямої мережі, що стало можливим завдяки використанню правила ланцюга для ітеративного обчислення градієнтів для кожного шару. Вона тісно пов’язана з алгоритмом Гауса – Ньютона і є частиною продовження досліджень в області нейропроникності. Алгоритм зворотного поширення є особливим випадком більш загальної техніки, званої автоматичної диференціацією. У контексті навчання, він зазвичай використовується разом з алгоритмом оптимізації градієнтного спуску для коригування ваг нейронів шляхом обчислення градієнта функції втрати.

Щоб зрозуміти математичне підґрунтя алгоритму зворотного розповсюдження, розглянемо просту нейронну мережу з двома вхідними нейронами, одним вихідним нейроном і без прихованих нейронів. Кожен нейрон використовує лінійну функцію активації (на відміну від більшості нейронних мереж, в яких функція активації з входів до виходів є нелінійною).

Спочатку перед тренуванням ваги будуть встановлюватися випадковим чином. Тоді нейрон дізнається з навчальних прикладів, які в даному випадку складаються з наборів (x_1, x_2, t), де x_1 і x_2 є входами в мережу і t є коректним виходом. Початкова мережа, задана як x_1 і x_2, обчислить вихід y, який, ймовірно, відрізняється від t (з урахуванням випадкових ваг). У загальному випадку, помилка результату складатиме:

$$E = (t - y)^2 \tag{25}$$

де E - невідповідність або помилка.

Проблему зіставлення входів до виходів можна звести до задачі оптимізації пошуку функції, яка дасть мінімальну помилку. Однак вихід нейрона залежить від зваженої суми всіх її входів:

$$y = x_1 w_1 + x_2 w_2 \tag{26}$$

де w_1 і w_2 - ваги, що поєднують вхідні нейрони з вихідним нейроном. Тому помилка також залежить від вхідних ваг до нейрона.

Один з алгоритмів для пошуку набору ваг, що мінімізує помилку, що використовується найчастіше, є градієнтним спуском, а алгоритм зворотного поширення потім використовується обчислення найшвидшого напрямку спуску.

Метод градієнтного спуску передбачає обчислення похідної функції помилки щодо ваг мережі.

Припускаючи один вихідний нейрон, функція квадрата помилки:

$$E = \frac{1}{2}\left(t - y\right)^2 \quad (27)$$

де E - квадратична помилка, t є цільовим виходом для навчальної вибірки, і y - фактичний вихід вихідного нейрона.

Коефіцієнт $\frac{1}{2}$ враховується для скасування сталої при диференціації.

Для кожного нейрона j, вихід якого o_j визначається як

$$o_j = \varphi\left(net_j\right) = \varphi\left(\sum_{k=1}^{n} w_{ki} o_k\right) \quad (28)$$

Вхідний сигнал net_j є зваженою сумою виходів o_k нейронів попереднього шару. Якщо нейрон знаходиться у першому шарі після вхідного шару, то o_k є просто входом x_k в мережу. Змінна w_{kj} позначає вагу між нейроном k попереднього шару і нейроном j поточного шару.

Функція активації φ нелінійна і диференційована. Зазвичай використовується функція активації у вигляді логістичної функції:

$$\varphi(z) = \frac{1}{1+e^{-z}}, \quad (29)$$

яка має зручну похідну:

$$\frac{d\varphi(z)}{dz} = \varphi(z)\left(1-\varphi(z)\right) \quad (30)$$

Обчислення часткової похідної помилки відносно ваги w_{ij} здійснюється за допомогою правила ланцюга двічі:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \quad (31)$$

У даній формулі, тільки net_j залежить від w_{ij}, тому

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i. \quad (32)$$

Якщо нейрон знаходиться в першому шарі після вхідного шару, то $o_i = x_i$.

Похідна виходу нейрона j відносно його входу є просто частковою похідною функції активації (припускаючи, що використовується логістична функція):

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j} \varphi\left(net_j\right) = \varphi\left(net_j\right)\left(1-\varphi\left(net_j\right)\right) \quad (33)$$

Перший доданок є простим для обчислення, якщо нейрон знаходиться у вихідному шарі, тому що тоді

$$\begin{cases} o_j = y \\ \frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}\left(t-y\right)^2 = y-t \end{cases}. \qquad (34)$$

Якщо *j* нейрон знаходиться в довільному внутрішньому шарі мережі, знаходження похідної *E* по відношенню до o_j є менш очевидним. Розглядаючи *E* як функцію, що залежить від входів всіх нейронів отримуємо вхід від нейрона *j*:

$$\frac{\partial E\left(o_j\right)}{\partial o_j} = \frac{\partial E\left(net_1, net_2, ..., net_n\right)}{\partial o_j} \qquad (35)$$

Повна похідна відносно o_j, отримуємо рекурсивний вираз для похідної:

$$\frac{\partial E}{\partial o_j} = \sum_l \left(\frac{\partial E}{\partial net_l}\frac{\partial net_l}{\partial o_j}\right) = \sum_l \left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l} w_{jl}\right) \qquad (36)$$

Отже, похідна по відношенню до o_j може бути обчислена, якщо всі похідні відносно виходів o_l наступного шару - відомі ближче до вихідного нейрона.

Маємо:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \begin{cases} \left(o_j - t_j\right) o_j \left(1 - o_j\right), j \in W_o \\ \left(\sum_l w_{jl}\delta_l\right) o_j \left(1 - o_j\right), otherwise. \end{cases} \qquad (37)$$

де W_o - нейрони зовнішного шару.

Оновлення ваг w_{ij}, використовуючи градієнтний спуск, відбувається шляхом вибору швидкості навчання, $\eta > 0$. Зміна ваги повинна відображати вплив *E* на збільшення або зменшення w_{ij}. Новий Δw_{ij} додається до старих ваг, а добуток швидкості навчання і градієнта, помноженого на *-1*, гарантує, що w_{ij} змінюється таким чином, що завжди зменшується *E*. Маємо:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j o_i . \tag{38}$$

Хотілось би зазначити, що вченими було виділено 2 фундаментальних обмеження на нейронні мережі такого типу: відсутність у них здатності до узагальнення своїх характеристик на нові ситуації, а також нездатність аналізувати складні ситуації з зовнішньому середовищі шляхом розділення їх на більш прості задачі.

Нейронні мережі такого типу мають свої особливості, що відрізняє їх від інших типів схожих нейронних мереж:

- Використовується нелінійна функція активації
- Кількість шарів, що навчаються більше одного
- Сигнали, що поступають на вхід та отримується на виході, не бінарні, а можуть кодуватись у десяткових числах
- Похибка мережі визначається не як число неправильно розпізнаних образів, а як деяка статистична міра нев'язки між отриманим та шуканим результатом
- Навчання проводиться не до відсутності похибки, а до стабілізації вагових коефіцієнтів або переривається раніше, щоб запобігти перенавчанню моделі

3.5 Дерево прийняття рішення

Дерева прийняття рішень - це один з методів прогнозування, що використовуються в статистиці, інтелектуальному аналізі даних і машинному навчанні. Дерева, де цільова змінна може приймати дискретний набір значень, називаються деревом класифікації. У цих деревних структурах листя являють собою мітки класів, а гілки являють собою кон'юнкції ознак, які ведуть до тих міток класу. Дерева рішень, де цільова змінна може приймати безперервний набір значень (зазвичай реальні числа), називаються деревами регресії.

Дерево прийняття рішень є методом, який зазвичай використовується в інтелектуальному аналізі даних[70]. Метою є створення моделі, яка передбачає значення цільової змінної на основі декількох вхідних змінних. Кожен внутрішній вузол відповідає одній з вхідних змінних, а ребра будуються для кожного з можливих значень цієї вхідної змінної. Кожен листок являє собою значення цільової змінної з урахуванням значень вхідних змінних, представлених шляхом від кореня до листа.

Дерево прийняття рішень - це просте подання для класифікації прикладів. Для цього припустимо, що всі вхідні ознаки мають кінцеві дискретні області, і є єдина цільова функція, яка називається "класифікацією". Дерево рішень або дерево класифікації - це дерево, в якому кожен внутрішній (нелистовий) вузол позначений елементом введення. Дуги, що надходять з вузла, позначеного функцією введення, позначені кожним з можливих значень цільової або вихідної функції, або дуга призводить до підпорядкованого вузла прийняття рішення на іншій вхідній функції. Кожен листок дерева позначається класом або розподілом ймовірностей над класами, що означає, що набір даних був класифікований деревом або в певний клас, або в певний розподіл ймовірностей (який, якщо дерево рішень добре - побудований, має перекіс до певних підгруп класів).

Дерево можна "навчити" шляхом поділу набору елементів на підмножини на основі значень атрибутів. Цей процес повторюється на кожному похідному підмножині рекурсивним чином, який називається рекурсивним розбиттям. Рекурсія завершується, коли підмножина у вузлі має всі однакові значення цільової змінної, або коли розщеплення більше не додає значення до прогнозів. Цей процес індукції дерев рішень згори-вниз (TDIDT)[71] є прикладом жадібного алгоритму, і він є найпоширенішою стратегією для вивчення дерев рішень з даних.

В області інтелектуального аналізу даних, дерева рішень можуть бути описані також як комбінація математичних і обчислювальних методів, що допомагають опису, категоризації та узагальненню даного набору даних.

Дані надходять у формі:

$$(x, Y) = (x_1, x_2, ..., x_n, Y).$$

де Y - цільова змінна, яку ми намагаємося зрозуміти, класифікувати або узагальнити; вектор x складається з ознак $x_1, x_2, ..., x_n$, які використовуються для цього завдання.

Дерева рішень, що використовуються в інтелектуальному аналізі даних, мають два основних типи:

- Аналіз класифікаційного дерева - це коли прогнозований результат - клас (дискретний), до якого належать дані.
- Аналіз регресійного дерева - це коли прогнозований результат можна вважати реальним числом (наприклад, ціна будинку або тривалість перебування пацієнта в лікарні).

Деякі методи, які часто називають методами ансамблю, створюють більше одного дерева рішень:

- Посилені дерева: Поступово будують ансамбль шляхом навчання кожного нового екземпляра, щоб підкреслити випадки попереднього неправильного моделювання. Типовим прикладом є AdaBoost. Вони можуть бути використані для проблем регресійного типу та класифікаційного типу[72, 73].
- Bootstrap агреговані дерева рішень - ранній метод ансамблю, будує кілька дерев рішень, неодноразово повторно підбираючи дані тренування з заміною, і використовуючи консенсус для прогнозування[74].
- Класифікатор випадкових лісів - це специфічний тип Bootstrap агрегації.
- Ліс обертання: кожне дерево рішень навчається шляхом першого застосування аналізу основних компонентів (PCA) на випадковому підмножині вхідних функцій[75].

Дерево рішень є структурою, подібною до потокової діаграми, де кожен внутрішній (нелистовий) вузол позначає тест на атрибут, кожна гілка представляє результат тесту, а кожен листок містить мітку класу. Найвищим вершиною дерева є кореневий вузол.

Існує багато конкретних алгоритмів дерева рішень. Серед них:

- ID3 (Ітеративний дихотомізатор 3)
- C4.5 (наступник ID3)

- CART (дерево класифікації та регресії)
- CHAID (автоматичний детектор CHi-squared). Виконує багаторівневі розбиття при обчисленні дерев класифікації[76].
- MARS: краще розширює дерева рішень для обробки цифрових даних.
- Дерева умовних висновків. Підхід на основі статистики, що використовує непараметричні тести як критерії поділу, виправлений для багаторазового тестування, щоб уникнути перенавчання. Цей підхід призводить до неупередженого вибору предиктора і не вимагає обрізки[77, 78].

Алгоритми побудови дерев рішень зазвичай працюють згори-вниз, вибираючи змінну на кожному кроці, що найкраще розбиває набір елементів. Різні алгоритми використовують різні метрики для вимірювання "найкращого". Вони зазвичай вимірюють однорідність цільової змінної в межах підмножин.

Використовуваний алгоритмом класифікації та регресії дерева, домішка Джині є мірою того, як часто випадково вибраний елемент з набору буде неправильно позначений, якщо він був випадково позначений відповідно до розподілу міток у підмножині. Домішка Джині може бути обчислена шляхом підсумовування ймовірності p_i елемента з міткою i, обраної в часі ймовірності $\sum_{k \neq i} p_k = 1 - p_i$ помилки в класифікації цього пункту. Він досягає свого мінімуму (нуль), коли всі випадки в вузлі потрапляють в одну цільову категорію. Щоб обчислити домішки Джині для набору елементів з J класами, припустимо $i \in \{1, 2, ..., J\}$ і p_i - частка елементів, позначених класом i у наборі.

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i (1 - p_i) = \\ = \sum_{i=1}^{J} (p_i - p_i^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = \\ = 1 - \sum_{i=1}^{J} p_i^2 \quad (39)$$

Використовуються алгоритмами деревоподібної генерації ID3, C4.5 та C5.0. Інформаційний підхід базується на концепції ентропії та інформаційного змісту з теорії інформації.

Ентропія визначається наступним чином

$$H(T)=I_E(p_1,p_2,...,p_J)=-\sum_{i=1}^{J} p_i \log_2 p_i \quad (40)$$

де $p_1,p_2,...,p_J$ - відсоток кожного класу, присутнього в дочірньому вузлі, який є результатом розбиття в дереві[79].

$$\overbrace{IG(T,a)}^{Information\,Gain} = \overbrace{H(T)}^{Entropy(parent)} - \overbrace{H(T\mid a)}^{Weighted\,Sum\,of\,Entropy(children)} = \\ = -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_a p(a) \sum_{i=1}^{J} -\Pr(i\mid a)\log_2 \Pr(i\mid a) \quad (41)$$

Інформаційний приріст використовується, щоб вирішити, яку функцію слід розділити на кожному етапі побудови дерева. Щоб зробити це, на кожному кроці ми повинні вибрати розділ, який призводить до чистих дочірніх вузлів. Зазвичай використовується міра чистоти називається інформаційне значення, яке вимірюється в бітах. Для кожного вузла дерева інформаційне значення "представляє очікуваний обсяг інформації, необхідний для визначення того, чи слід класифікувати новий екземпляр так чи ні, враховуючи, що приклад досяг цього вузла"[79].

Розглянемо приклад набору даних з чотирма атрибутами: погода (сонячна, похмура, дощова), температура (жарко, м'яко, прохолодно), вологість (висока, нормальна) і вітряно (так, ні), з двійковою (так чи ні) цільовою змінною, «грати», та 14 наборами даних. Щоб побудувати дерево рішень на цих даних, нам необхідно порівняти інформаційний приріст кожного з чотирьох дерев, кожен розбитий на одну з чотирьох ознак. Розділення з найвищим інформаційним коефіцієнтом буде прийнято за перше роздылення, і процес буде продовжуватися, поки всі діти-вузли не будуть чистими, або поки інформаційний приріст не буде дорівнює 0.

Розділимо дані, використовуючи змінну «вітряно». У цьому наборі даних є шість точок даних з істинним вітряним значенням, три з яких мають значення змінної «грати» - так, а три мають значення змінної «грати» - ні. Вісім залишившихся точок даних з негативним значенням змінної «вітряно» мають 2 негативних значення та шість позитивних значень змінної «грати».

Для значення «вітряно»=так обчислимо ентропію

$$I_E([3,3])=-\frac{3}{6}\log_2\frac{3}{6}-\frac{3}{6}\log_2\frac{3}{6}=-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}=1$$

Для значення «вітряно»=ні, де було вісім точок: шість так і два ні, маємо

$$I_E([6,2]) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8113$$

Щоб знайти інформацію про розділення, ми беремо середньозважене значення цих двох чисел на основі того, скільки спостережень потрапило в який вузол.

$$I_E([3,3],[6,2]) = \frac{6}{14}\cdot 1 + \frac{8}{14}\cdot 0.8113 = 0.8922$$

Щоб знайти інформаційний приріст розбиття за змінною «вітряно», ми повинні спочатку обчислити інформаційне значення в даних перед розбиттям. Оригінальні дані містили дев'ять так і п'ять ні.

$$I_E([9,5]) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.9403$$

Тепер ми можемо обчислити інформаційний приріст, досягнутий шляхом розбиття на змінною «вітряно»:

$$IG_1 = I_E([9,5]) - I_E([3,3],[6,2]) = 0.9403 - 0.8922 = 0.0481$$

Щоб побудувати дерево, необхідно розрахувати інформаційний приріст кожного можливого першого розбиття. Найкращий перший розкол - це той, який забезпечує найбільшу інформаційну вигоду. Цей процес повторюється для кожного нечистого вузла до завершення дерева.

Серед інших методів інтелектуального аналізу даних, дерева рішень мають різні переваги:

- Прості для розуміння та інтерпретації. Люди можуть зрозуміти модель дерева рішень після короткого пояснення. Дерева також можуть відображатися графічно таким чином, що їх легко інтерпретувати[80].
- Необхідна невелика підготовка даних. Інші методи часто вимагають нормалізації даних. Оскільки дерева можуть обробляти якісні предиктори, немає необхідності створювати фіктивні змінні[80].
- Використовують модель білої коробки. Якщо дана ситуація спостерігається в моделі, пояснення умови легко пояснюється булевою логікою.
- Можлива перевірка моделі з використанням статистичних тестів. Це дає можливість врахувати надійність моделі.

- Нестатистичний підхід, що не робить припущень щодо даних тренувань або залишків прогнозування; наприклад, відсутність припущень щодо розподілу, незалежності або постійної дисперсії.
- Добре виконується з великими наборами даних. Великі обсяги даних можуть бути проаналізовані з використанням стандартних обчислювальних ресурсів у розумний час.
- Вбудований вибір ознаки. Додаткові нерелевантні ознаки майже не будуть використовуватись, тому їх можна буде видалили в наступних прогонках.
- Дерева рішень можуть наблизити будь-яку функцію (напр. XOR)[81].

На дерева прийняття рішень накладається ряд обмежень:

- Дерева можуть бути дуже нестійкими. Невелика зміна даних тренувань може призвести до великих змін у дереві і, відповідно, до остаточних прогнозів[80].
- Проблема вивчення оптимального дерева рішень є NP-повною за декількома аспектами оптимальності навіть для простих концепцій[82, 83]. Практичні алгоритми навчання на основі дерева рішень засновані на евристиці, такі як жадібний алгоритм, де локально оптимальні рішення приймаються на кожному вузлі.
- Дані, що приймають участь у прийнятті рішення, можуть створювати надскладні дерева, які добре узагальнюються лише навчальну вибірку[84]. Механізми, такі як обрізка, необхідні для уникнення цієї проблеми (за винятком деяких алгоритмів, таких як підхід умовного висновку, який не вимагає обрізки)[77, 78].
- Для даних, що включають категоріальні змінні з різним числом рівнів, інформаційний приріст у деревах рішень зміщений на користь атрибутів з більшим рівнем[85]. Тим не менш, питання вибору упередженого предиктора уникається підходом «умовного виводу»[77], двоступеневим підходом[86] або адаптивним вибором функцій виходу з одного виходу[87].

3.7 Модифікований метод *k*-найближчих сусідів

У розпізнаванні образів алгоритм k-найближчих сусідів (kNN) є непараметричним методом, що використовується для класифікації та регресії[88]. В обох випадках вхідні дані складаються з k найбільш близьких прикладів навчання в просторі ознак. Вихід залежить від того, чи використовується k-NN для класифікації або регресії:

- У класифікації kNN вихід є клас, до якого відноситься даний об’єкт. Об'єкт класифікується множинним голосуванням його сусідів, при цьому об'єкт призначається класу, найбільш поширеному серед його найближчих сусідів (k - ціле позитивне число, зазвичай невелике). Якщо k = 1, то об'єкт просто присвоюється класу того самого найближчого сусіда.
- У kNN регресії вихід є значенням властивості для об'єкта. Це значення є середнім значенням k найближчих сусідів.

Метод найближчих сусідів є прикладом ледачого навчання, де функція лише апроксимується локально і все обчислення відкладено до класифікації. Алгоритм kNN є одним з найпростіших алгоритмів машинного навчання. Сусіди обираються з набору об'єктів, для яких відомий клас (для класифікації kNN) або значення властивості об'єкта (для kNN регресії). Це можна розглядати як навчальну вибірку для алгоритму, хоча явного кроку навчання цей метод не має.

Особливість алгоритму kNN полягає в тому, що він чутливий до локальної структури даних.

Припустимо, що у нас є пари $(X_1,Y_1),(X_2,Y_2),...,(X_n,Y_n)$, де Y - класова мітка X. З огляду на деяку норму $\|\cdot\|$ в просторі R^d та точку $x \in R^d$ точки з набору $(X_1,Y_1),(X_2,Y_2),...,(X_n,Y_n)$ будуть переупорядковані наступним чином $\|X_1 - x\| \le ... \le \|X_n - x\|$.

У фазі класифікації k - константа, визначена користувачем, а вектор, шо подається на вхід алгоритму (запит або контрольна точка), класифікується шляхом присвоєння мітки, яка є найбільш частою серед k навчальних наборів, найближчих до цієї точки запиту.

Зазвичай використовується метрика відстані для безперервних змінних - евклідова відстань. Для дискретних змінних, таких як класифікація тексту, можна використовувати іншу метрику, наприклад метрику перекриття (або відстань Хеммінга). Часто точність класифікації kNN може бути значно покращена, якщо метрику відстані вивчається за допомогою спеціалізованих

алгоритмів, таких як аналіз великих граничних сусідів або компонентів сусідства.

Недолік основної класифікації "більшості" відбувається тоді, коли розподіл класів є перекісним. Тобто, приклади більш частого класу мають тенденцію домінувати в прогнозуванні нового прикладу, оскільки вони мають тенденцію бути спільними серед найближчих сусідів через їх велику кількість[89]. Одним із шляхів подолання цієї проблеми є зважування класифікації з урахуванням відстані до кожної з її найближчих сусідів. Способом подолання перекосу є абстракція в поданні даних. Наприклад, у самоорганізованій карті (SOM) кожен вузол є представником (центром) кластера подібних точок, незалежно від їх щільності у вихідних даних навчання. Потім K-NN може бути застосований до SOM.

Вибір найкращого значення k залежить від даних; загалом, великі значення k зменшують вплив шуму на класифікацію[90], але роблять межі між класами менш виразними. Добре k можна вибрати різними евристичними методами. Особливий випадок, коли клас передбачається класом найближчої навчальної вибірки (тобто, коли k = 1), називається алгоритмом найближчого сусіда.

Точність алгоритму kNN може суттєво погіршитися завдяки наявності шумних або невідповідних ознак, або якщо шкала ознак не узгоджується з їх важливістю. Багато досліджень були спрямовані на вибір або масштабування ознак для поліпшення класифікації. Особливо популярним підходом є використання еволюційних алгоритмів для оптимізації масштабування функцій[91]. Інший популярний підхід полягає у масштабуванні ознак шляхом взаємної інформації навчальних даних з навчальними класами.

У двійкових (двох класових) проблемах класифікації корисно вибрати k, щоб бути непарним числом, оскільки це уникає прив'язаних голосів. Одним з популярних способів вибору емпірично оптимального k в цьому параметрі є метод bootstrap[92].

Класифікатор методу k-найближчих сусідів можна розглядати як присвоєння k сусідам ваги $\frac{1}{k}$, причому усі інші отримають ваги 0. Це можна узагальнити до класифікованих найближчих сусідів класифікаторів. Тобто, кожен i-й найближчий сусід отримує вагу w_{ni} за умови, що $\sum_{i=1}^{n} w_{ni} = 1$. Аналогічний результат щодо сильної узгодженості класифікованих найближчих сусідів також має місце[93].

Нашою задачею є побудова класифікатора γ, що оснований на використанні відстані від точок тренувального набору до об'єкту для

побудови ваг. Ми знаємо, що усі ваги повинні задовольняти правилу $\sum_{i=1}^{n} w_i = 1$.

Використовуючи наявні умови, для *i*-ї точки, ваги будуть мати вигляд:

$$w_i = \frac{d_i^{-p}}{\sum_{j=1}^{n} d_j^{-p}} \tag{42}$$

Тоді, для класифікації нової нового об'єкту використаємо правило

$$Y = \gamma(X) = \sum_{i=1}^{n} w_i y_i = \begin{cases} C_1, \gamma \geq \frac{(C_1 - C_2)}{2} \\ C_2, \gamma < \frac{(C_1 - C_2)}{2} \end{cases}. \tag{43}$$

Якщо вибірка маленька, то метод зводиться до методу найближчого сусіда. Ми експериментально отримали, що при $p = 2$, ми отримали гарний результат, причому точність моделі є співставною з іншими методами і є адекватною.

Коли кількість ознак у вхідних є занадто великою і є підозра, що вони є надлишковими, то вхідні дані будуть перетворені в набір зменшених репрезентацій (також називається вектором особливостей). Перетворення вхідних даних у набір ознак називається вилученням ознак . Якщо вилучені ознаки вибираються ретельно, то очікується, що набір ознак витягуватиме відповідну інформацію з вхідних даних, щоб виконати бажане завдання, використовуючи для цього зменшене подання, а не повний розмір введення. .

Приклад типового конвеєра computer vision обчислень для розпізнавання осіб з використанням kNN, включаючи етапи попередньої обробки вилучення ознак і зменшення розмірів (зазвичай реалізовані разом з OpenCV):

- Виявлення обличчя Хаара
- Аналіз відстеження середнього зсуву
- Проекція PCA або Fisher LDA в просторі ознак, за яким йде класифікація kNN

Для високовимірних даних (наприклад, з числом розмірів більше 10) зменшення розмірності зазвичай виконується до застосування алгоритму kNN, щоб уникнути наслідків прокляття розмірності[94].

Прокляття розмірності в контексті kNN в основному означає, що евклідова відстань не є корисною при великих розмірностях, тому що всі вектори майже рівновіддалені до вектора пошукового запиту (уявіть кілька точок, що лежать більш-менш по колу з точкою запиту в центрі; відстань від запиту до всіх точок даних у просторі пошуку майже однакова).

Виділення ознак і зменшення розміру можуть бути об'єднані в одну стадії з використанням аналізу головних компонент (PC), лінійного дискримінантного аналізу (LDA), або канонічного кореляційного аналіз (CCA) в якості стадії попередньої обробки, а потім з допомогою кластеризації kNN на ознаки в просторі зменшеної розмірності. У машинному навчанні цей процес також називається низьковимірним вбудовуванням[95].

Для наборів даних дуже високих розмірів (наприклад, при виконанні пошуку подібності на потоках живого відео, даних ДНК або високих часових рядів), виконують швидкий приблизний пошук k -NN, використовуючи чутливість до локалізації, "випадкові прогнози"[96], «ескізи»[97] або інші високоякісні способи пошуку подібності з інструментарію VLDB, що можуть бути єдиним можливим варіантом.

Зменшення даних є однією з найважливіших проблем для роботи з величезними наборами даних. Зазвичай для точної класифікації потрібні лише деякі точки даних. Ці дані називаються прототипами і можуть бути знайдені наступним чином:

- Виділяється відхилення класів, тобто навчальних даних, які класифіковані неправильно за допомогою kNN (для заданого k)
- Окрему частину даних відокремлюють на дві групи:
  - прототипи, що використовуються для прийняття рішень класифікації,
  - поглинені точки, які можна правильно класифікувати за допомогою kNN з використанням прототипів.

Поглинені точки потім можуть бути вилучені з навчального набору.

Навчальний приклад, оточений прикладами інших класів, називається відхиленням класу. До причин викидів класів належать:

- випадкова помилка
- недостатньо навчальних прикладів цього класу (замість кластера з'являється окремий приклад)
- відсутні важливі особливості (класи розділені в інших вимірах, які ми не знаємо)
- надто багато навчальних прикладів інших класів (незбалансовані класи), які створюють «вороже» тло для даного малого класу

Класи викидів виробляють шум. Вони можуть бути виявлені і відокремлені для подальшого аналізу. З урахуванням двох натуральних чисел, $k > r > 0$, навчальним прикладом називається (k, r) NN клас-залишок, якщо його K найближчих сусідів включають в себе більше, ніж r прикладів інших класів.

Конденсований найближчий сусід (CNN, алгоритм Харта) є алгоритмом, призначеним для зменшення набору даних для kNN класифікації[98]. Він вибирає набір прототипів U з навчальних даних, так що 1NN з U може класифікувати приклади майже так само точно, як і 1NN може класифікувати цілий набір даних.

Враховуючи навчальний набір *X*, CNN працює ітеративно:

- Розглядається навчальна вибірка *X*, шукається елемент *x*, чий найближчий прототип від U має іншу мітку, ніж *x*.
- *x* видаліть з *X* і додається до U
- Повторіть розгляд вибірки, доки у U не буде додано більше прототипів .

Для класифікації використовуйте U замість *X*. Приклади, які не є прототипами, називаються "поглиненими" точками.

Ефективно переглядати навчальні приклади в порядку зменшення граничного співвідношення[99]. Граничне співвідношення прикладу навчання визначається як

$$a(x) = \frac{\|x' - y\|}{\|x - y\|} \qquad (44)$$

де $\|x - y\|$- відстань до найближчого прикладу y, що має інший колір, ніж x , і $\|x' - y\|$ - відстань від y до його найближчого прикладу x ' з тією ж міткою, що і x .

Граничне співвідношення використовується як атрибутом початкової точки *x*.

4. Практична реалізація

4.1 Туристичне страхування в Сінгапурі

Розглянемо набір даних, що являє собою дані про туристичне страхування в Сінгапурі. Дані представлені наступним чином:

travel insurance.csv (4.36 MB)

	Agency	Agency...	Distrib...	Product...	Claim	Duration	Destina...	Net Sal...	Commi...	Gender	Age
1	CBH	Travel Agency	Offline	Comprehensive Plan	No	186	MALAYSIA	-29	9.57	F	81
2	CBH	Travel Agency	Offline	Comprehensive Plan	No	186	MALAYSIA	-29	9.57	F	71
3	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	No	65	AUSTRALIA	-49.5	29.7		32
4	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	No	60	AUSTRALIA	-39.6	23.76		32
5	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	No	79	ITALY	-19.8	11.88		41
6	JZI	Airlines	Online	Value Plan	No	66	UNITED STATES	-121	42.35	F	44

Для побудови моделі нам необхідно провести попередню обробку даних. Як ми можемо бачити дані мають наступну структуру:

1. Цільовий вектор: Статус заявки (Claim.Status)
2. Назва агенства (Agency)
3. Тип агенства (Agency.Type)
4. Канал збуту туристичного страхування (Distribution.Channel)
5. Ім'я продукту туристичного страхування (Product.Name)
6. Тривалість подорожі (Duration)
7. Місце призначення (Destination)
8. Обсяг продажів полісів страхування подорожей (Net.Sales)
9. Комісійні, що надійшли до агентства туристичного страхування (Commission)
10. Стать людини, що хоче отримати страхування (Gender)
11. Вік людини, що хоче отримати страхування (Age)

Цільовий вектор має 2 класи: 'yes', 'no', тому ми можемо прийняти їх за бінарну комбінацію: 1 - 'yes', 0 - 'no'.

Назва агенства не впливає на цільовий вектор, тому ми можемо видалити цю ознаку з таблиці. Цей крок зробить наші дані більш незалежними.

Дані містять 5 категоріальних змінних: 'Agency.Type' (Travel Agency, Airlines), 'Distribution.Channel' (Online, Offline), 'Product.Name' (24 категорії), 'Destination', 'Gender' (M, F). Усі категорійні змінні необхідно замінити на числові значення, що будуть відповідати тим чи іншим категоріям. Інші ж ознаки є числовими, тому вони не потребують попередньої обробки.

Після обробки змінних, необхідно впевнитись що дані не мають пропусків та елементів, що повторюються. Для цього використаємо наступні функції:

```
data= data.drop_duplicates()
data=data.dropna()
```

Так як методи не можуть сприймати символьну інформацію, перевіримо, щоб після виконання підготовчих операції, усі змінні мали числовий характери. Для цього використаємо функцію

```
X.dtypes

Agency Type                int64
Distribution Channel       int64
Product Name               int64
Duration                   int64
Destination                int64
Net Sales                float64
Commision (in value)     float64
Gender                   float64
Age                        int64
dtype: object
```

Як ми бачимо, усі змінні є числовими, тому можемо вважати, що попередня підготовка даних завершена і ми можемо починати процес моделювання.

Процес моделювання розпочнемо з виділення головних компонент. Для цього необхідно попередньо нормалізувати дані. Для цього використаємо мінімізацію по mix-max критерію:

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

```
from sklearn.preprocessing import normalize
X = pd.DataFrame(normalize(X, norm='max', axis=1))
X["target"] = Y
X.head()
```

	0	1	2	3	4	5	6	7	8	target
0	0.005376	0.005376	0.005376	1.0	0.005376	-0.155914	0.051452	0.005376	0.435484	0
1	0.005376	0.005376	0.005376	1.0	0.005376	-0.155914	0.051452	0.005376	0.381720	0
2	0.015385	0.030769	0.030769	1.0	0.030769	-0.761538	0.456923	0.007692	0.492308	0
3	0.016667	0.033333	0.033333	1.0	0.033333	-0.660000	0.396000	0.008333	0.533333	0
4	0.012658	0.025316	0.025316	1.0	0.037975	-0.250633	0.150380	0.006329	0.518987	0

Виділимо головні компоненти даних. Для цього знайдемо власні значення моделі:

```
Y = X.iloc[:,-1]
X = X.iloc[:,0:data.shape[1]-1]
pca = PCA(n_components=X.shape[1])
pca.fit(X)
eigenvalues = pca.explained_variance_ratio_
eigenvalues
```

```
array([4.07857327e-01, 2.79896588e-01, 1.29451772e-01, 8.97832149e-02,
       7.82748988e-02, 1.41579844e-02, 3.93053965e-04, 1.15138800e-04,
       7.00228604e-05])
```

Кожне значення в даному масиві показує частику інформативності, яку несе в собі кожна компонента. В теорії, можна обрати компоненти, що несуть в собі 67% інформативності і не приведе до втрати важливої інформації. Ми ж вважаємо, що 67% даних – це дуже малий відсоток, тому обирали компоненти, які сумарно несуть у собі більше 80% інформативності. У деяких випадках дані були дуже сильно зв'язані між собою і більше 80% інформативності припадало на 1 одну компоненту. В такому випадку ми обирали мінімум 2 компоненти.

```
count = 0
sum = 0
for eig in eigenvalues:
    sum = sum + eig
    count = count + 1
    if sum > 0.8:
        break
if count == 1:
    count = 2
count

3

sum

0.817205686301105
```

У даному випадку кількість компонент склала 3 з інформативністю 81.72%.

Тому, дані після вибору головних компонент будуть мати вигляд

```
pca = PCA(n_components=count)
pca.fit(X)
transformX = pd.DataFrame(pca.transform(X))
transformX.head()
```

	0	1	2
0	0.843407	0.378402	0.245331
1	0.867712	0.352360	0.268310
2	1.055779	0.870330	0.494028
3	0.995167	0.813061	0.427391

З теорії інтелектуального аналізу даних відомо, що застосовуючи метод ансамблів, точність майбутнього прогнозу зростає. Тому є необхідність у використанні методу ансамблів, а враховуючи, що одним з найновіших є метод опорних векторів, використаємо його для побудови ансамблю. Було експериментально доведено, що метод опорних векторів, оснований на радіальному ядрі Гауса, дають непоганий результат в плані точності.

Для проведення процесу навчання було створено 2 функції. Функція getMethod, повертала об'єкт-класифікатор, використовуючи назву методу.

Функція func слугує функцією тренування моделі. За допомогою цієї функції ми навчаємо модель, отримуємо прогноз та знаходимо попередню помилку розрахунків.

```
def getMethod(str):
    if str == "DecisionTreeClassifier":
        return DecisionTreeClassifier()
    if str == "GaussianNB":
        return GaussianNB()
    if str == "SVM":
        return SVC(gamma='auto')
    if str == "AdaBoost":
        return AdaBoostClassifier(DecisionTreeClassifier(), algorithm='SAM
ME', n_estimators=200)
    if str == "Neural Networks":
        return MLPClassifier(alpha=1)

def func(str, X_train, Y_train, X_test, Y_test, methods, mse):
    cl = getMethod(str)
    cl.fit(X_train, Y_train)
    c = methods.get(str, list())
    c.append(cl)
    methods[str] = c
    dtpred = cl.predict(X_test)
    m = mse.get(str, list())
    m.append(mean_squared_error(Y_test, dtpred))
    mse[str] = m
    return (methods, mse, dtpred)
```

Для ансамблю використаємо 60% даних. Ці дані стануть тренувальною вибіркою і на їх основі буде побудовано нова змінна:

```
seq = [int(x) for x in transformX.shape[0]*np.random.rand(round(0.6*transf
ormX.shape[0]),1)]
train_fold = transformX.iloc[seq]
newX = transformX.drop('target', axis=1)

train_fold_output = data.iloc[seq, -1]
train_fold_input = train_fold.drop('target', axis=1)

s = func('SVM',
                          train_fold_input,
                          train_fold_output,
                          newX,
                          data.iloc[:, -1],
                          dict(),
                          dict())
transformX = transformX.drop('target', axis=1)
transformX['svm_rgb'] = s[2]

transformX.head()
```

	0	1	2	svm_rgb
0	0.843407	0.378402	0.245331	0
1	0.867712	0.352360	0.268310	0
2	1.055779	0.870330	0.494028	0
3	0.995167	0.813061	0.427391	0
4	0.834930	0.492346	0.243186	0

Ансамбль готовий, тому у нас є все необхідне для пошуку оптимального методу класифікації. Для класифікації було використано наступні методи:

- Дерево прийняття рішення
- Гаусівський Наївниї баєсів класифікатор
- Метод опорних векторів з радіальним ядром Гауса
- Нейронні мережі прямого поширення помилки з коефіцієнтом навчання, рівним 1
- Бустинг-класифікатор на основі дерева прийняття рішення

В якості перехресної перевірки було використано метод 10-кратної крос-валідації. Під час прогонки моделі набір даних було поділено на 10 частин, кожна з яких була протестована. Це дало можливість моделі навчитись на усіх даних. На кожному етапі було навчено 5 моделей та пораховано помилку на для кожної моделі.

```
start_time = time.time()
skf = StratifiedKFold(n_splits=10, random_state = True)
methods = {}
mse = {}
fold_count = 0
for train, test in skf.split(transformX, data.iloc[:, -1]):
    print("Processing fold %s" % fold_count)
    train_fold_input = transformX.iloc[train]
    test_fold_input = transformX.iloc[test]

    train_fold_output = data.iloc[train, -1]
    test_fold_output = data.iloc[test, -1]

    list_of_methods  =  ["DecisionTreeClassifier",  "GaussianNB",  "SVM", 
"AdaBoost", "Neural Networks"]

    for method in list_of_methods:
        s = func(method,
                          train_fold_input,
                          train_fold_output,
                          test_fold_input,
                          test_fold_output,
                          methods,
                          mse)
        methods = s[0]
```

```
        mse = s[1]

  # Done with the fold
    fold_count += 1
print("--- %s seconds ---" % (time.time() - start_time))

Processing fold 0
Processing fold 1
Processing fold 2
Processing fold 3
Processing fold 4
Processing fold 5
Processing fold 6
Processing fold 7
Processing fold 8
Processing fold 9
--- 11277.264970779419 seconds ---
```

Маємо наступні результати. Модель була навчена за 3 год. 7 хв. 57 сек. Розглянемо середню похибку по методам:

```
def takeFirst(elem):
    return elem[0]

errors=[]
for k,v in mse.items():
    res = np.mean(v)
    errors.append((res, k))
errors.sort(key = takeFirst)
errors

[(0.016660028315490037, 'GaussianNB'),
 (0.016660028315490037, 'SVM'),
 (0.016660028315490037, 'Neural Networks'),
 (0.026174988158263173, 'AdaBoost'),
 (0.03360953148593516, 'DecisionTreeClassifier')]
```

Розглянемо поставлену задачу зі сторони ледачої класифікації, а саме проведемо класифікацію за допомогою моделі, яку ми розробили.

На основі побудованих головних компонент, виберемо 60% даних початкового набору та на їх основі виконаємо класифікацію наступним чином:

```
seq = [int(x) for x in transformX.shape[0]*np.random.rand(round(0.6*transf
ormX.shape[0]),1)]
transformX['target'] = Y
train_fold = transformX.iloc[seq]
newX = transformX.drop('target', axis=1)

trainY = train_fold['target']
trainX = train_fold.drop('target', axis=1)

def method(trainX, trainY, testX):
    outs = []
```

```
    for (idx1, testRow) in testX.iterrows():
        temp = trainX
        temp = temp.sub(testRow, axis=1)**2
        d = temp.sum(axis=1)
        d = [np.sum(d) if x==0 else x for x in d]
        w=[1/(x**2) for x in d]
        sumW=np.sum(w)
        if np.sum(trainY*w)/sumW > 0.5:
            out = 1
        else:
            out = 0
        outs.append(out)
    return outs
outs = method(trainX, trainY, testX)
mean_squared_error(outs, testY)
```

```
0.027353420687156 57
```

Як ми бачимо, найкращий результат дали наступні методи:

- Наївний баєсів класифікатор,
- Метод опорних векторів,
- Нейронні мережі прямого поширення.

Даний результат обумовлений тим, що відношення класів є нерівномірним, тому модифікований метод k-найближчих сусідів дає поганий результат. Також дерева прийняття рішення та бустинг-метод на його основі не дали гарного результату, бо ці методи більш орієнтовані на категорійний розподіл даних. З іншого боку, нейронні мережі орієнтовні на будь-які дані, тому вони гарно справляються з поставленою задачею. Щодо методу опорних векторів, цей метод дає гарні результати через те, обране нами ядро має гарний роздільний результат. Наївний баєсів класифікатор же має гарну роздільну здатність через те, що він базується на умовних ймовірностях і припускає лінійну незалежність даних, чого ми досягли шляхом виділення головних компонент.

4.2 Дощ в Австралії

Розглянемо набір даних, що являє собою набір погодних показників Австралії. Основною задачею є обробка показників для прогнозування дощу в наступних день. Дані представлені наступним чином:

	Date	Location	MinTemp	MaxTe...	Rainfall	Evapor...	Sunshine	WindG...	WindG...	WindDi...	WindDi...	WindS...	WindS...	#
	07													
8	2008-12-08	Albury	7.7	26.7	0	NA	NA	W	35	SSE	W	6	17	
9	2008-12-09	Albury	9.7	31.9	0	NA	NA	NNW	80	SE	NW	7	28	
10	2008-12-10	Albury	13.1	30.1	1.4	NA	NA	W	28	S	SSE	15	11	
11	2008-12-11	Albury	13.4	30.4	0	NA	NA	N	30	SSE	ESE	17	6	
12	2008-12-12	Albury	15.9	21.7	2.2	NA	NA	NNE	31	NE	ENE	15	13	
13	2008-12-13	Albury	15.9	18.6	15.6	NA	NA	W	61	NNW	NNW	28	28	
14	2008-12-14	Albury	12.6	21	3.6	NA	NA	SW	44	W	SSW	24	20	
15	2008-12-16	Albury	9.8	27.7	NA	NA	NA	WNW	50	NA	WNW	NA	22	
16	2008-12-17	Albury	14.1	20.9	0	NA	NA	ENE	22	SSW	E	11	9	
17	2008-12-18	Albury	13.5	22.9	16.8	NA	NA	W	63	N	WNW	6	20	
18	2008-12-19	Albury	11.2	22.5	10.6	NA	NA	SSE	43	WSW	SW	24	17	
19	2008-12-20	Albury	9.8	25.6	0	NA	NA	SSE	26	SE	NNW	17	6	
20	2008-12-	Albury	11.5	29.3	0	NA	NA	S	24	SE	SE	9	9	

Дані мають наступну структуру:

- Date: дата проведення дослідження
- Location: назва місцевості
- MinTemp: мінімальна температура в градусах цельсію
- MaxTemp: максимальна температура в градусах цельсію
- Rainfall: кількість опадів, що випала за день
- Evaporation: випаровування (мм) протягом 24 годин до 9 ранку
- Sunshine: кількість годин яскравого сонячного світла в день
- WindGustDir: напрямок сильного пориву вітру протягом 24 годин до полудня
- WindGustSpeed: швидкість (км/год) сильного пориву вітру протягом 24 годин до полудня
- WindDir9am: напрямок вітру о 9 ранку
- WindDir3pm: напрямок вітру о 3 дня
- WindSpeed9am: швидкість вітру (км/год) в середньому за 10 хвилин до 9 ранку

- WindSpeed3pm: швидкість вітру (км/год) в середньому за 10 хвилин до 9 ранку
- Humidity9am: вологість о 9 ранку у відсотках
- Humidity3pm: вологість о 3 дня у відсотках
- Pressure9am: Атмосферний тиск (hpa) відносно середнього рівня моря о 9 ранку
- Pressure3pm: Атмосферний тиск (hpa) відносно середнього рівня моря о 3 дня
- Cloud9am: частка неба, затьмарена хмарою в 9 ранку. Вимірюється в "октах". Міра 0 свідчить про повністю чисте небо, а 8 означає, що воно є повністю похмуре.
- Cloud3pm: частка неба, затьмарена хмарою о 3 дня.
- Temp9am: температура в Цельсіях о 9 ранку
- Temp3pm: температура в Цельсіях о 3 дня
- RainToday: логічна: 1, якщо опадів (мм) протягом 24 годин до 9 ранку перевищує 1 мм, інакше 0
- RISK_MM: кількість дощу у наступний день
- RainTomorrow: цільова змінна, показує чи буде дощ завтра.
- The target variable. Did it rain tomorrow?

Перші дві змінних не несуть важливої інформації для прогнозу, тому ми можемо їх видалити.

Змінні, що містять напрямок вітру є категорійними, тому замінимо їх числовими аналогами.

Як ми можемо бачити, дані містять багато пропусків. Враховуючи те, що ми не компетентні у заміні цих пропусків, ми видаляємо рядки з пропусками, що може потягти за собою деяке погіршення точності результату.

Маємо:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	...	Humidity3pm	Press
0	17.9	35.2	0.0	12.0	12.3	1	48.0	1	1	6.0	...	13.0	
1	18.4	28.9	0.0	14.8	13.0	2	37.0	2	2	19.0	...	8.0	
2	19.4	37.6	0.0	10.8	10.6	3	46.0	3	3	30.0	...	22.0	
3	21.9	38.4	0.0	11.4	12.2	4	31.0	4	4	6.0	...	22.0	
4	24.2	41.0	0.0	11.2	8.4	4	35.0	5	5	17.0	...	15.0	
5	27.1	36.1	0.0	13.0	0.0	5	43.0	6	5	7.0	...	19.0	
6	23.3	34.0	0.0	9.8	12.6	1	41.0	7	2	17.0	...	15.0	
7	16.1	34.2	0.0	14.6	13.2	6	37.0	8	6	15.0	...	9.0	
8	19.0	35.5	0.0	12.0	12.3	7	48.0	1	4	30.0	...	28.0	
9	19.7	35.5	0.0	11.0	12.7	8	41.0	3	4	15.0	...	14.0	
10	20.9	37.8	0.0	12.8	13.2	9	30.0	8	7	11.0	...	9.0	
11	23.9	38.1	0.0	13.8	12.1	7	39.0	9	8	24.0	...	15.0	
12	24.9	41.2	0.0	14.8	13.0	1	43.0	6	3	17.0	...	15.0	
13	25.2	40.5	0.0	16.4	10.3	10	44.0	10	1	13.0	...	15.0	
14	21.6	34.2	0.0	17.4	13.1	10	44.0	7	6	17.0	...	8.0	
15	18.4	31.8	0.0	16.0	12.9	2	33.0	7	1	17.0	...	5.0	
16	17.9	34.2	0.0	12.0	11.3	6	61.0	9	9	22.0	...	19.0	
17	21.4	37.5	0.0	14.8	6.9	3	43.0	1	10	26.0	...	29.0	
18	23.3	39.4	4.8	12.0	10.9	11	59.0	6	11	19.0	...	14.0	

Після видалення пропусків та неінформативних змінних, набір даних має наступних розмір: 56420 rows × 22 columns

Розпочнемо обробку даних з перевірки типу даних.

```
MinTemp          float64
MaxTemp          float64
Rainfall         float64
Evaporation      float64
Sunshine         float64
WindGustDir        int64
WindGustSpeed    float64
WindDir9am         int64
WindDir3pm         int64
WindSpeed9am     float64
WindSpeed3pm     float64
Humidity9am      float64
Humidity3pm      float64
Pressure9am      float64
Pressure3pm      float64
Cloud9am         float64
Cloud3pm         float64
Temp9am          float64
Temp3pm          float64
RainToday          int64
RISK_MM          float64
```

Як ми бачимо, усі змінні є числовими, тому ми можемо починати процес виділення головних компонент та навчання.

Для виділення головних компонент, нормалізуємо дані за допомогою min-max критерію:

```
from sklearn.preprocessing import normalize
X = pd.DataFrame(normalize(X, norm='max', axis=1))
X["target"] = Y
X.head()
```

	0	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17
0	0.017788	0.034980	0.0	0.011925	0.012223	0.000994	0.047699	0.000994	0.000994	0.005962	...	0.012919	1.0	0.998112	0.001987	0.004969	0.026433
1	0.018166	0.028532	0.0	0.014612	0.012834	0.001975	0.036529	0.001975	0.001975	0.018758	...	0.007898	1.0	0.999210	0.000987	0.000987	0.020041
2	0.019164	0.037143	0.0	0.010669	0.010471	0.002964	0.045441	0.002964	0.002964	0.029635	...	0.021733	1.0	0.996938	0.000988	0.005927	0.028351
3	0.021625	0.037918	0.0	0.011257	0.012047	0.003950	0.030611	0.003950	0.003950	0.005925	...	0.021724	1.0	0.996445	0.000987	0.004937	0.028735
4	0.023944	0.040566	0.0	0.011081	0.008311	0.003958	0.034629	0.004947	0.004947	0.016820	...	0.014841	1.0	0.996735	0.000989	0.005936	0.033244

5 rows × 22 columns

Процедура нормалізації є обов'язковим кроком, тому що під час виділення головних компонент будуються проекції кожної змінної на вісь і по проекції знаходиться дисперсія даних відносно осі. Якщо ж дані не будуть нормалізовані, кожна зі змінних має свою розмірність і через це неможливо визначити яка зі змінних має більшу дисперсію відносно своєї проекції.

Для виділення головних компонент розглянемо власні значення даних:

```
array([4.68159203e-01, 1.81244239e-01, 1.05138533e-01, 6.54263213e-02,
       4.66892259e-02, 2.64750658e-02, 2.34781743e-02, 1.99073633e-02,
       1.45050497e-02, 1.40278989e-02, 1.27985366e-02, 8.16837127e-03,
       4.40521222e-03, 3.38391727e-03, 1.89297349e-03, 1.73341794e-03,
       1.12211286e-03, 8.45387334e-04, 3.39614748e-04, 1.93014473e-04,
       6.63672302e-05])
```

Як ми можемо бачити, 4 компоненти є значимими і складають 82% інформативності.

У цих компонентах, дані мають наступний вигляд:

	0	1	2	3
0	-0.059461	0.001490	0.009731	-0.004106
1	-0.054548	-0.006320	-0.000363	-0.002168
2	-0.041161	0.005495	0.009254	0.005580
3	-0.042177	-0.016752	0.019483	0.000924
4	-0.060119	-0.005198	0.022465	-0.005410

На основі отриманих даних, побудуємо ансамбль, використовуючи метод опорних векторів з радіальним ядром Гауса. Маємо:

	0	1	2	3	svm_rgb
0	-0.059461	0.001490	0.009731	-0.004106	0
1	-0.054548	-0.006320	-0.000363	-0.002168	0
2	-0.041161	0.005495	0.009254	0.005580	0
3	-0.042177	-0.016752	0.019483	0.000924	0
4	-0.060119	-0.005198	0.022465	-0.005410	0

На основі цих даних, проведемо процес навчання моделі, використовуючи методи, що перераховані вище. Процес навчання зайняв 1 год. 15 хв. 36 сек. Після навчання маємо наступний результат:

```
[(0.1966679336618802, 'GaussianNB'),
 (0.2010991876046989, 'SVM'),
 (0.2010991876046989, 'Neural Networks'),
 (0.22688583882817587, 'AdaBoost'),
 (0.22763018662031737, 'DecisionTreeClassifier')]
```

Розглядаючи ледачий алгоритм, процес навчання зайняв набагато більше часу. Це зумовлено великою кількістю матричних операцій. З іншого боку, ми отримали наступну точність:

```
0.16409074796171572
```

Маємо, найкращу точність дали методи: Модифікований метод k-найближчих сусідів, Наївний баєсів класифікатор Гауса, Метод опорних векторів, Нейронні мережі.

Враховуючи те, що в даному випадку розподіл класів був більш рівномірним, даний результат є очікуваним.

4.3 Оцінка автомобіля

Одним з найцікавіших прикладів є набір даних оцінки автомобілів. Ці дані мають наступний вигляд:

	vhigh_2	2	2_2	small	low	unacc
1	vhigh	2	2	small	med	unacc
2	vhigh	2	2	small	high	unacc
3	vhigh	2	2	med	low	unacc
4	vhigh	2	2	med	med	unacc
5	vhigh	2	2	med	high	unacc
6	vhigh	2	2	big	low	unacc
7	vhigh	2	2	big	med	unacc
8	vhigh	2	2	big	high	unacc
9	vhigh	2	4	small	low	unacc
10	vhigh	2	4	small	med	unacc
11	vhigh	2	4	small	high	unacc
12	vhigh	2	4	med	low	unacc
13	vhigh	2	4	med	med	unacc
14	vhigh	2	4	med	high	unacc

Цей набір даних цікавий тим, що усі змінні є категорійними:

- Buying: v-high, high, med, low
- Maint: v-high, high, med, low
- Doors: 2, 3, 4, 5-more
- Persons: 2, 4, more
- lug_boot: small, med, big
- safety: low, med, high

Цільова змінна представлена у якості булевої змінної та показує оцінку автомобіля з точки зору прийнятно/неприйнятно. Співвідношення класів 70/30. База складається з 1728 об’єктів.

Виконаємо попередню підготовку даних, зведемо їх до числових значень, нормалізуємо та виділимо головні компоненти.

Ми отримали наступні власні значення системи:

```
array([0.32417932, 0.18203989, 0.18172083, 0.12948686, 0.10166703,
       0.08090607])
```

Значимими є 4 компоненти, що несуть у собі 81.74% інформативності.

У нових координатах, дані мають наступний вигляд:

	0	1	2	3
0	-0.006157	-4.793559e-15	0.110727	0.114456
1	0.148832	-3.905380e-15	0.091516	0.430881
2	-0.046674	-7.768031e-15	0.183901	0.366443
3	0.148832	-3.683336e-15	0.091516	0.430881
4	0.303822	-2.795157e-15	0.072305	0.747305

Побудуємо ансамбль на основі методу опорних векторів з радіальним ядром Гауса та почнемо процес навчання.

Навчання моделі зайняло 3 хв. 57 сек. Ми отримали наступну точність методів:

```
[(0.25296074741228664, 'AdaBoost'),
 (0.2609792982927813, 'Neural Networks'),
 (0.2696565398575077, 'SVM'),
 (0.27081260922166955, 'GaussianNB'),
 (0.2940146525070574, 'DecisionTreeClassifier')]
```

Розглянемо ці дані з боку ледачого навчання. Так як у ледачого навчання фактично немає процесу навчання, немає необхідності виконувати перехресну перевірку.

Ми отримали наступну точність:

```
0.21180555555555555
```

Як ми можемо бачити, найкращу точність дали наступні методи:

- Модифікований метод найближчих сусідів
- Бустинг-метод на основі дерева прийняття рішення
- Нейронні мережі

Ці методи дали гарний результат, тому що усі дані були категорійними. Нейронні мережі запам’ятовують об’єкти, фактично вони роблять зліпок усіх даних і можуть досить точно їх розпізнати. Бустинг-метод дав гарний результат, бо усі дані були категорійними. А методу найближчих сусідів дав гарний результат, бо відстань між об’єктами була досить велика і був відсутній процес зашумованості даних.

4.4 Загальний випадок

В ході виконання даної роботи, було протестовано 26 наборів даних. Ми обирали дані різної розмірності, для того, щоб результат був максимально точним. Кількість об'єктів в наборах коливалась від 90 до 63326 об'єктів. Для точності експерименту ми намагались охопити усі діапазони, на що вказує частотна діаграма:

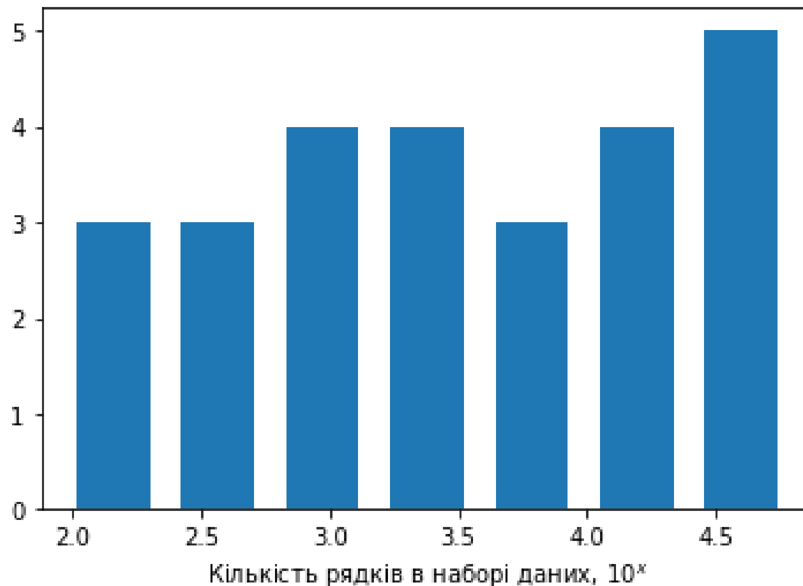

Рис. 2 Частотна діаграма кількості об'єктів в наборах даних

Також ми намагались обирати набори даних, що мають різну кількість ознак. Кількість ознак коливалось від 6 до 33. Частотна гістограма має вигляд:

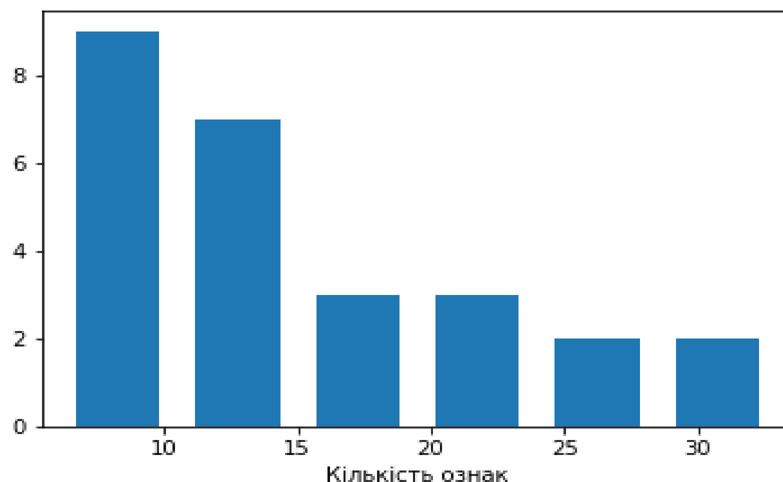

Рис. 3 Частотна діаграма розподілу ознак в наборах даних

Після обробки даних, ми отримали різні показними похибок. Значення похибки поливалось від набору до набору:

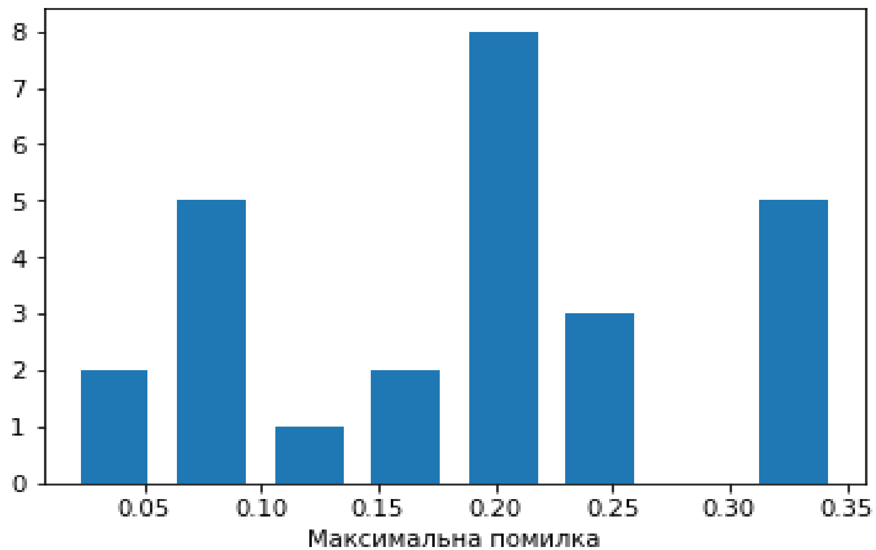

Рис. 4 Розподіл похибок

Як ми можемо побачити, середнє значення похибки знаходиться в районі відмітки 0.2 і його значення має найбільша кількість наборів. Середнє значення по всім базам склало 0.18574365808684087.

Слід зазначити, що модель тестувалась на 6 методах і різниця між найкращою та найгіршою точністю не була однакова. В середньому ця різниця складала 0.13078861977116715, але були й такі моделі, де ця різниця досягала 0,45.

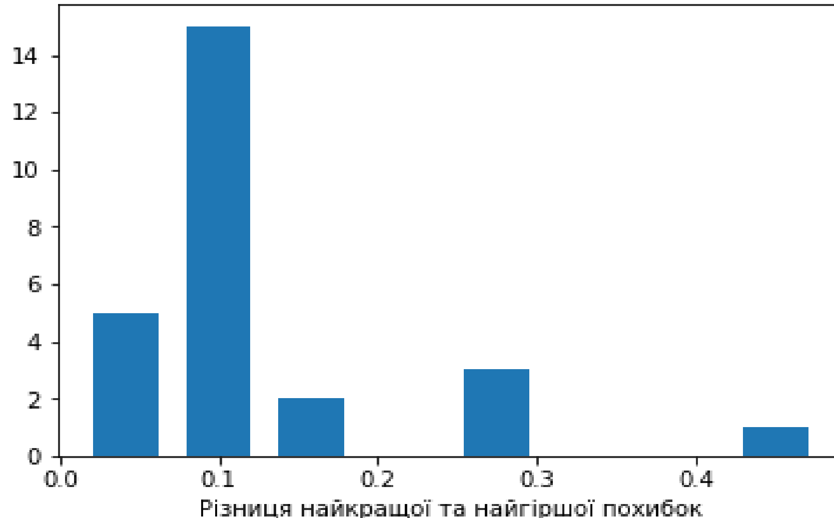

Рис. 5 Різниця похибок методів в наборах даних

Цей результат зумовлений тим, що дані були погано підготовленими початково. Для покращення цього результату спеціаліста необхідно додати нові описові ознаки, що покращать точність класифікації.

Розглянемо отримані результати. Для кожного набору даних було проведено перехресну перевірку на 5 методах. Результатом кожної прогонки була навчена модель, яка могла класифікувати дані з певною точністю. Після

перехресної перевірки дані зводились в єдиний результат, а точність усереднювалась. Для кожного набору даних було отримано 51 класифікатор, з яких 50 моделей отриманих за допомогою 10-кратної перехресної перевірки, та одна модель ледачої класифікації.

Отримані результати показали, що для кожного набору даних можна виділити 3 методи, точність яких буде майже однаковою. Середня різниця між першими 3 методами складає 0.013638207925217296. В загальному випадку 61,5% наборів даних має майже нульову відмінність у похибках:

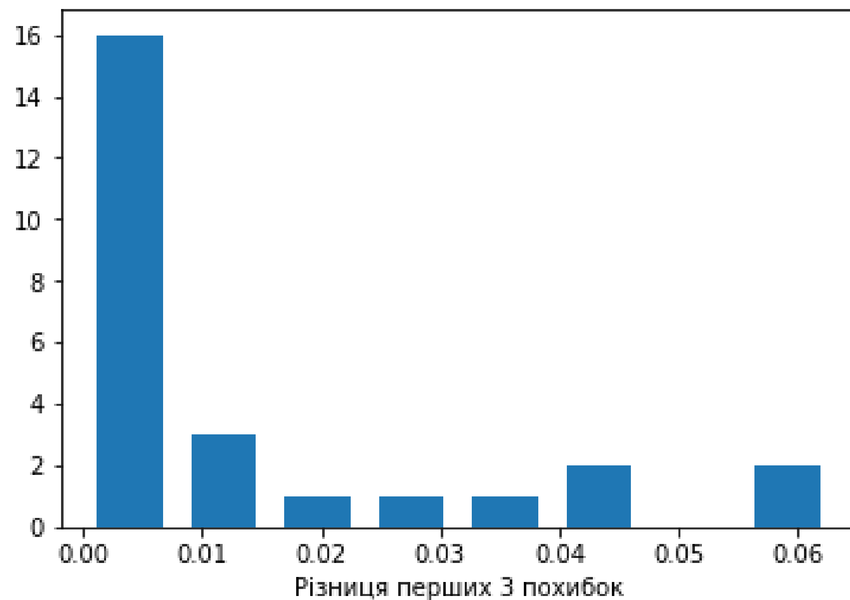

Рис. 6 Відхилення похибок перших 3 методів

Опираючись на концепт рівності похибок, ми отримали наступні результати:

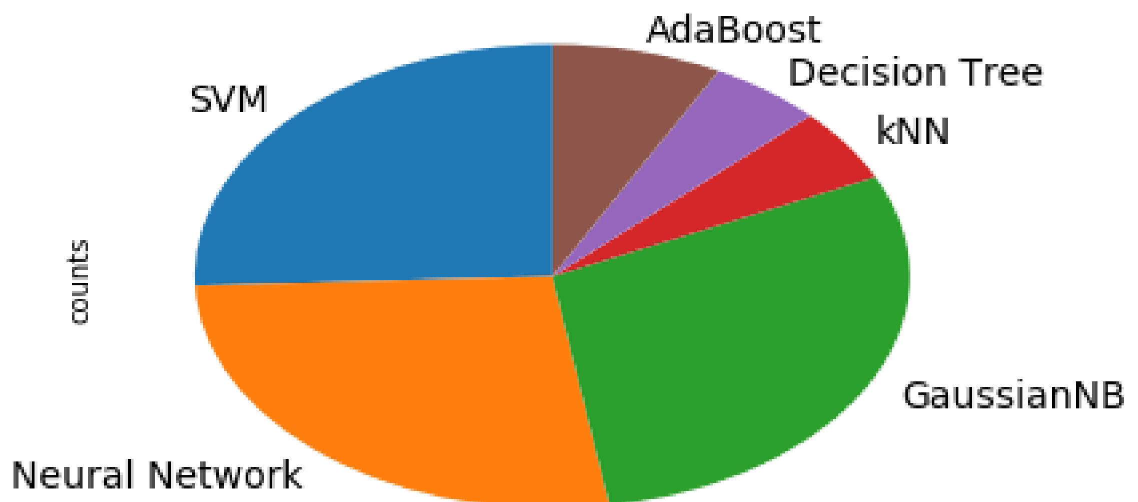

Рис. 7 Розподіл методів з найбільшою точністю.

Як ми можемо бачити, більшість займає наївний баєсів класифікатор Гауса. Він зустрічається у 23 наборах даних. Цей класифікатор працює ефективно, якщо більшість ознак – дискретні. Якщо ж дані мають усі категорійні змінні, як було у 3 наборах даних, то даний класифікатор не можна використовувати. Замість нього необхідно було використати звичайний наївний баєсів класифікатор. Ми досягли гарного результату, використовуючи наївний баєсів класифікатор, завдяки виділенню головних компонент. Цей процес забезпечив отримання умови незалежності компонент, що є базовою засадою наївної класифікації.

На другому місці ми маємо нейронні мережі. Через їх здатність робити зліпок даних. Фактично нейронна мережа, виконуючи процес навчання, запам'ятовую кожен об'єкт, знаходячи таке значення ваг, що при прогонці вже знайомого об'єкту, буде точно класифікувати його. Ця особливість нейронних мереж була перейнята від реальних нейронних мереж і допомагає штучним нейронним мережам показувати гарний результат. Однією з важливих особливостей також є те, що нейронним мережам байдуже, з якими даними працювати. Під час процесу навчання, нейронна мережа зможе запам'ятати як категорійні дані, так і дискретні.

На третьому місці ми бачимо метод опорних векторів. Даний результат зумовлений тим, що нам вдалось гарно підібрати ядро метода. Хочеться зазначити, якщо використати лінійне ядро, ти результат був би набагато гіршим. Даний метод дає гарні результати завдяки тому, що він будується на засадах оптимізації, тобто, для вирішення поставленої задачі необхідно звести поставлену задачу до задачі квадратичної оптимізації. Цей процес допомагає досить точно розділити дані, хоча і потребує досить велику кількість обчислень. Для досягнення більш кращого результату можна спробувати замінити радіальне ядро Гауса на тригонометричний тангенс чи поліноміальне ядро.

Решта ж методів, а саме бустинг-метод, дерево прийняття рішень та модифікований метод найближчих сусідів, дали гарний результат лише на методах, що містять лише категорійні дані, або їх частка на багато більша за дискретні. Метод найближчих сусідів може бути використаний також на дискретних даних з правильним підбором коефіцієнта p. Ми тестували модель при p=2. При зростанні даного коефіцієнта, на наборах даних, що ми мали, при збільшенні p, точність зростала на долі відсотка або ж зменшувалась взагалі. Це може бути зумовлено тим, що дані були не точні, тобто мав місце людський фактор, що вплинуло на вимірювальну точність.

Щодо дерев прийняття рішення та бустинг-методу, що будувався також на деревах прийняття рішення, можна сказати, що для побудови гарної

моделі, дані повинні бути роздільними. Це зумовлюється тим, що для кожного розділення дерево будує нову гілку і з кожним розділенням дерево збільшується у розмірах, що є досить затратним і не завжди необхідним. Тому деревам легше працювати з категорійними даними і класифікація таких даних деревами прийняття рішення є досить ефективним.

Отже, ми можемо виділити наступні особливості:

- Якщо система містить лише категорійні дані, то найкращим варіантом є використання методів: дерево прийняття рішення, рандомний ліс, як частинний випадок бустинг-методу та модифікованого методу найближчих сусідів;
- Якщо ж система містить дискретні дані, то найкращим варіантом буде використати метод опорних векторів, нейронні мережі та метод наївної баєсової класифікації Гауса разом з виділенням головних компонент. Виділення головних компонент забезпечує основні засади використання баєсового класифікатора та зменшує розмірність даних, без втрати інформації.
- Метод найближчих сусідів може бути відрегульований шляхом підбору параметру p, причому цим самим, можна добитись гарної точності. Цей метод гарно працює, якщо кількість даних перевищує 50000 і розподіл класів перевищує 20/80.

Висновок

В роботі була розглянута задача оптимізації проблем класифікації. Задача звелась до побудови математичної моделі, що, використовуючи метод головних компонент, допомагає системі підібрати метод класифікації, що дає найкращу точність.

Методом головних компонент було відібрано компоненти, що несуть у собі більше 80% інформативності. Якщо ж на 80% інформативності припадала лише одна компонента, опиралось 2 компоненти, щоб задача не зводилась до лінійної роздільності.

Для побудови моделі було використано наступні методи:

- Наївний баєсів класифікатор Гауса;
- Нейронні мережі прямого поширення помилки;
- Метод опорних векторів;
- Дерево прийняття рішення;
- Бустинг-метод на основі дерев прийняття рішення;
- Модифікований метод найближчих сусідів, що спирається на відстані між сусідами.

Адекватність моделі було перевірено за допомогою 10-тикратної перехресної валідації.

Отримана модель була запрограмована за допомогою мови програмування Python. На основі отриманої моделі було проаналізовано 26 наборів даних.

Отримані результати дають можливість зробити наступні висновки:

- При класифікації даних, що містять лише категорійні дані, необхідно використовувати методи, що основані не деревах прийняття рішення або модифікований метод найближчих сусідів;
- При класифікації дискретних даних, найкраще працюють нейронні мережі, метод опорних векторів та наївний баєсів класифікатор Гауса.
- Модифікований метод найближчих сусідів може бути використаний для дискретний даних з попереднім підбором параметра p.
- Для класифікації категорійних даних можна також використати метод звичайної наївної класифікації, що також забезпечить гарний результат, якщо його використати разом з методом головних компонент.

Список використаної літератури

1. Xu, S., & Dezhi, C. (2013). 2013 Third International Conference on Intelligent System Design and Engineering Applications ISDEA 2013.
2. Jiawei, H., Kamber, M., & Pei, J. (2006). Data mining, southeast Asia edition: Concepts and techniques.
3. Suthaharan, S. (2014). Big data classification: problems and challenges in network intrusion prediction with machine learning. *ACM SIGMETRICS Performance Evaluation Review*, *41.4*, 70–73.
4. Deren, L., & Wang, S. (2005). Concepts, principles and applications of spatial data mining and knowledge discovery. In *Proceedings of the International Symposium on Spatio-Temporal Modeling*. Beijing, China.
5. Zaki, M. J., & Wagner, M. J. (2014). Data Mining and Analysis: Fundamental Concepts and Algorithms. *Cambridge University Press*.
6. Washio, T., & Motoda, H. (2003). State of the art of graph-based data mining. *ACM SIGKDD Explorations Newsletter 5.1*, 59–68.
7. Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., … Liu, B. (2007). Top 10 algorithms in data mining. *Springer-Verlag*.
8. Hegland, M. (2001). Data mining techniques. *Acta Numerica 2001*, *10*, 313–355.
9. Murdopo, A. (2013). *Distributed Decision Tree Learning for Mining Big Data Streams*.
10. Tjoa, A. M., Paryudi, I., & Ashari, A. (2013). Performance Comparison between Naïve Bayes, Decision Tree and k-Nearest Neighbor in Searching Alternative Design in an Energy Simulation Tool. *Journal of IJACSA, IJACSA (International Journal of Advanced Computer Science and Applications*, *4*(11).
11. Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
12. McCulloch, W., & Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. *Bulletin of Mathematical Biophysics*, *5*(4), 115–133.
13. Kleene, S. C. (1956). *Representation of Events in Nerve Nets and Finite Automata*. *Annals of Mathematics Studies* (Vol. 34). Princeton University Press.
14. Rumelhart, D. ., & McClelland, J. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge: MIT Press.
15. Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen*. Technische Univ, Munich.
16. Hochreiter, S., John, F., & Kremer, S. C. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In *A Field Guide to Dynamical Recurrent Networks*. John Wiley & Sons.

17. Behnke, S. (2003). *Hierarchical Neural Networks for Image Interpretation*. (Springer, Ed.)*Lecture Notes in Computer Science*.
18. Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, *1*, 194–281.
19. Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.
20. Hinton, G. (2009). Deep belief networks. *Scholarpedia*, *4*(5), 5947.
21. Aizerman, M. A., Braverman, E. M., & Rozonoer, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, *25*, 821–837.
22. Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In C. '92 (Ed.), *Proceedings of the fifth annual workshop on Computational learning theory* (p. 144).
23. Cortes, C., & Vapnik, V. N. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.
24. Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). Spam filtering with Naive Bayes—which Naive Bayes? In (CEAS) 17 (Ed.), *Third conference on email and anti-spam*.
25. Zhang, H. (2004). The Optimality of Naive Bayes. In *FLAIRS2004 conference*.
26. Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proc. 23rd International Conference on Machine Learning*.
27. Rennie, J., Shih, L., Teevan, J., & Karger, D. (2003). Tackling the poor assumptions of Naive Bayes classifiers. In *ICML*.
28. Fix, E., & Hodges, J. L. (1951). *Discriminatory analysis, nonparametric discrimination: Consistency properties*.
29. Cover, T.M., Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory*, *IT-13*(1), 21–27.
30. Hellman, M. E. (1970). The nearest neighbor classification rule with a reject option. *IEEE Trans. Syst. Man Cybern.*, *3*, 179–185.
31. Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *IEEE Trans. Syst. Man Cybern.*, *SMC-6*, 325–327.
32. Bailey, T., & Jain, A. (1978). A note on distance-weighted k-nearest neighbor rules. *IEEE Trans. Systems, Man, Cybernetics*, *8*, 311–313.
33. Bermejo, S., & Cabestany, J. (2000). Adaptive soft k-nearest-neighbour classifiers. *Pattern Recognition*, *33*, 1999–2005.
34. Jozwik, A. (1983). A learning scheme for a fuzzy k-nn rule. *Pattern Recognition Letters*, *1*, 287–289.
35. Keller, J. M., Gray, M. R., & Givens, J. A. (1985). A fuzzy k-nn neighbor algorithm. *IEEE Trans. Syst. Man Cybern.*, *SMC-15*(4), 580–585.

36. Kleiner, A., Talwalkar, A., Sarkar, P., & Jordan, M. (2012). *The big data bootstrap.*
37. Machová, K., Barcak, F., & Bednár, P. (2006). A bagging method using decision trees in the role of base classifiers. *Acta Polytechnica Hungarica*, *3.2*, 121–132.
38. Kuncheva, L. I., & Rodríguez, J. J. (2007). An experimental study on rotation forest ensembles. In *Multiple Classifier Systems* (pp. 459–468). Berlin Heidelberg: Springer.
39. Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, *2*(11), 559–572.
40. Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*, 417–441, 498–520.
41. Golub, G. H., & Van Loan, C. F. (1983). *Matrix Computations. The Johns Hopkins*. Maryland: University Press.
42. Jolliffe, I. T. (2002). *Principal Component Analysis*. *Springer Series in Statistics* (2nd ed.). NY: Springer.
43. Harman, H. H. (1960). *Modern factor analysis*. Oxford, England: Univ. of Chicago Press.
44. Sirovich, L. (1987). Turbulence and the dynamics of coherent structures. *Quarterly of Applied Mathematics*, 561–590.
45. Lorenz, E. N. (1956). *Empirical Orthogonal Functions and Statistical Weather Prediction. Technical report, Statistical Forecast Project Report 1.* Massachusetts: Department of Meteorology, MIT.
46. Barnett, T. P., & Preisendorfer, R. (1987). Origins and levels of monthly and seasonal forecast skill for United States surface air temperatures determined by canonical correlation analysis. *Monthly Weather Review*, *115*(9).
47. Hsu, D., Kakade, S. M., & Zhang, T. (2008). *A spectral algorithm for learning hidden markov models*.
48. Shaw, P. J. A. (2003). Multivariate statistics for the Environmental Sciences. *Hodder-Arnold*.
49. Cangelosi, R., & Goriely, A. (2007). Component retention in principal component analysis with application to cDNA microarray data. *Biology Direct*, *2*(2).
50. Geisser, S. (1993). *Predictive Inference*. NY: Chapman and Hall.
51. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In M. Kaufmann (Ed.), *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1137–1143). San Mateo, CA.
52. Devijver, P. A., & Kittler, J. (1982). *Pattern Recognition: A Statistical Approach*. London, GB: Prentice-Hall.
53. "What is the difference between test set and validation set? (2018).

54. Newbie question: Confused about train, validation and test data! (2015).
55. Cawley, G. C., & Talbot, N. L. C. (2010). On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, *11*, 2079–2107.
56. Trippa, L., Waldron, L., Huttenhower, C., & Parmigiani, G. (2015). "Bayesian nonparametric cross-study validation of prediction methods. *The Annals of Applied Statistics*, *9*, 402–428.
57. McLachlan, G. J., Do, K.-A., & Ambroise, C. (2004). Analyzing microarray gene expression data. *Wiley*.
58. Dubitzky, W., Granzow, M., & Berrar, D. (2007). Fundamentals of data mining in genomics and proteomics. *Springer Science & Business Media*, 178.
59. Hastie, T., Tibshirani, R., & Friedman, J. (n.d.). The elements of Statistical Learning, 134.
60. Jin, C., & Wang, L. (2012). Dimensionality dependent PAC-Bayes margin bound. *Advances in Neural Information Processing Systems*.
61. Maron, M. E. (1961). Automatic Indexing: An Experimental Inquiry. *Journal of the ACM*, *8*(3), 404–417.
62. Rish, I. (2001). An empirical study of the naive Bayes classifier. In *IJCAI Workshop on Empirical Methods in AI.*
63. Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Prentice Hall.
64. Murty, N. M., & Devi, S. V. (2011). *Pattern Recognition: An Algorithmic Approach*.
65. John, G. H., & Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. In *Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence* (pp. 338–345). Morgan Kaufmann.
66. McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. *AAAI-98 workshop on learning for text categorization.*, *752*.
67. Artificial Neural Networks as Models of Neural Information Processing | Frontiers Research Topic. (2018).
68. Encephalos Journal. (2019). Retrieved from www.encephalos.gr
69. Build with AI | DeepAI. (2018). *DeepAI*. Retrieved from https://deepai.org/learn
70. Rokach, L., & Maimon, O. (2008). *Data mining with decision trees: theory and applications*. World Scientific Pub Co Inc.
71. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.
72. Friedman, J. H. (1999). *Stochastic gradient boosting*. Stanford University.
73. Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). *The elements of statistical learning : Data mining, inference, and prediction*. New York:

Springer Verlag.
74. Breiman, L. (1996). Bagging Predictors. *Machine Learning*, *24*(2), 123–140.
75. Rodriguez, J. J., Kuncheva, L. I., & Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(10), 1619–1630.
76. Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, *29*(2), 119–127.
77. Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased Recursive Partitioning: A Conditional Inference Framework. *Journal of Computational and Graphical Statistics*, *15*(3), 651–674.
78. Strobl, C., Malley, J., & Tutz, G. (2009). "An Introduction to Recursive Partitioning: Rationale, Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests. *Psychological Methods*, *14*(4), 323–348.
79. Witten, I., Frank, E., & Hall, M. (2011). *Data Mining*. Burlington, MA: Morgan Kaufmann.
80. Gareth, J., Witten, D., & Hastie, T. (2015). *An Introduction to Statistical Learning*. New York: Springer.
81. Mehtaa, D., & Raghavan, V. (2002). Decision tree approximations of Boolean functions. *Theoretical Computer Science*, *270*(1–2), 60–623.
82. Hyafil, L., & Rivest, R. L. (1976). Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, *5*(1), 15–17.
83. S., M. (1998). "Automatic construction of decision trees from data: A multidisciplinary survey. *Data Mining and Knowledge Discovery*.
84. Principles of Data Mining. (2007).
85. Deng, H., Runger, G., & Tuv, E. (2011). Bias of importance measures for multi-valued attributes and solutions. In *21st International Conference on Artificial Neural Networks (ICANN)* (pp. 293–300).
86. Brandmaier, A. M., Oertzen, T. von, McArdle, J. J., & Lindenberger, U. (2012). Structural equation model trees. *Psychological Methods*, *18*(1), 71–86.
87. Painsky, A., & Rosset, S. (2017). Cross-Validated Variable Selection in Tree-Based Methods Improves Predictive Performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(11), 2142–2153.
88. Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, *46*(3), 175–185.
89. Coomans, D., & Massart, D. L. (1982). Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, *136*, 15–27. doi:10.1016/S0003-2670(01)95359-0
90. Everitt, B. S., Landau, S., Leese, M., & Stahl, D. (2011). *Miscellaneous Clustering Methods, in Cluster Analysis*. (L. John Wiley & Sons, Ed.) (5th

ed.). Chichester, UK.

91. Nigsch, F., Bender, A., van Buuren, B., Tissen, J., Nigsch, E., & Mitchell, J. (2006). Melting point prediction employing k-nearest neighbor algorithms and genetic parameter optimization. *Journal of Chemical Information and Modeling*, *46*(6), 2412–2422. doi:10.1021/ci060149f
92. Hall, P., Park, B., & Samworth, R. (2008). Choice of neighbor order in nearest-neighbor classification. *Annals of Statistics.*, *36*(5), 2135–2152. doi:10.1214/07-AOS537
93. Stone, C. J. (1977). Consistent nonparametric regression. *Annals of Statistics*, *5*(4), 595–620. doi:10.1214/aos/1176343886
94. Beyer, K. (1999). When is “nearest neighbor” meaningful? In *Database Theory—ICDT’99* (pp. 217–235).
95. Shaw, B., & Jebara, T. (2009). Structure preserving embedding. In *26th Annual International Conference on Machine Learning ACM*.
96. Bingham, E., & Heikki, M. (2001). Random projection in dimensionality reduction: applications to image and text data. In *The seventh ACM SIGKDD international conference on Knowledge discovery and data mining. ACM*.
97. Shasha, D. (2004). *High Performance Discovery in Time Series. Springer*. Berlin.
98. Hart, P. E. (1968). The Condensed Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, *18*, 515–516. doi:10.1109/TIT.1968.1054155
99. Mirkes, E. M. (2011). *KNN and Potential Energy: applet. University of Leicester*. University of Leicester.

ДОДАТОК А

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "1. Р—С‡РёС‚СѓРІР°РЅРЅСЏ С„Р°Р№Р»Сѓ"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "import pandas as pd\n",
    "import numpy as np\n",
    "path = 'DataSets/Rain in Australia/weatherAUS.csv'\n",
    "data = pd.read_csv(path, na_values='?', na_filter=True)\n",
    "#data=data.drop(columns=['Unnamed: 0.1', 'Unnamed: 0'])\n",
    "data.head()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "2. РџС–РґРіРѕС‚РѕРІРєР° РґР°РЅРёС…"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn import preprocessing \n",
    "from sklearn.preprocessing import MinMaxScaler "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "data= data.drop_duplicates()  \n",
    "data=data.dropna()"
   ]
  },
```

```
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "df=pd.DataFrame(data, index=None)\n",
  "df.index=range(df.shape[0])\n",
  "data=df\n",
  "data"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "for i in range(0,1027):\n",
  "    if data.iloc[i, 8] == '0':\n",
  "        data.iloc[i, 8] = 0\n",
  "    else:\n",
  "        data.iloc[i, 8] = 1\n",
  "data.iloc[:, 8].unique()"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "data.iloc[:,2]=data.iloc[:,2].replace('3+',3)\n",
  "data.iloc[:,2]=pd.to_numeric(data.iloc[:,2])\n",
  "data.iloc[:,2]"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "Y = data.iloc[:,-1]\n",
  "X = data.iloc[:,0:data.shape[1]-1]\n",
  "X.iloc[:, 2] = X.iloc[:,2].replace('5more', 5)\n",
  "X.iloc[:,3] = X.iloc[:,3].replace('more', 6)\n",
  "X.iloc[:,2]=pd.to_numeric(X.iloc[:,2])\n",
  "X.iloc[:,3]=pd.to_numeric(X.iloc[:,3])\n",
  "X.dtypes"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.preprocessing import normalize\n",
```

```
   "X = pd.DataFrame(normalize(X, norm='max', axis=1))\n",
   "X[\"target\"] = Y\n",
   "X.head()"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "3. Р’РёРґС–Р»РµРЅРЅСЏ РіРѕР»РѕРІРЅРёС… РєРѕРјРїРѕРЅРµРЅС‚"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "from sklearn.decomposition import PCA"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "Y = X.iloc[:,-1]\n",
   "X = X.iloc[:,0:data.shape[1]-1]\n",
   "pca = PCA(n_components=X.shape[1])\n",
   "pca.fit(X)\n",
   "eigenvalues = pca.explained_variance_ratio_\n",
   "count = 0\n",
   "sum = 0\n",
   "for eig in eigenvalues:\n",
   "    sum = sum + eig\n",
   "    count = count + 1\n",
   "    if sum > 0.8:\n",
   "        break\n",
   "if count == 1:\n",
   "    count = 2\n",
   "count"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "sum"
  ]
 },
 {
  "cell_type": "code",
```

```
 "execution_count": null,
 "metadata": {
  "scrolled": true
 },
 "outputs": [],
 "source": [
  "eigenvalues"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {
  "scrolled": true
 },
 "outputs": [],
 "source": [
  "pca = PCA(n_components=count)\n",
  "pca.fit(X)\n",
  "transformX = pd.DataFrame(pca.transform(X))\n",
  "transformX.head()"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "4. РџРµСЂРµС…СЂРµСЃРЅР° РїРµСЂРµРІС–СЂРєР°"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.model_selection import StratifiedKFold\n",
  "from sklearn.metrics import mean_squared_error"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.tree import DecisionTreeClassifier  \n",
  "from sklearn.naive_bayes import GaussianNB\n",
  "from sklearn.svm import SVC\n",
  "from sklearn.ensemble import AdaBoostClassifier\n",
  "from sklearn.neural_network import MLPClassifier"
 ]
},
{
 "cell_type": "code",
```

```
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "def getMethod(str):\n",
    "    if str == \"DecisionTreeClassifier\":\n",
    "        return DecisionTreeClassifier() \n",
    "    if str == \"GaussianNB\":\n",
    "        return GaussianNB()\n",
    "    if str == \"SVM\":                    \n",
    "        return SVC(gamma='auto')\n",
    "    if str == \"AdaBoost\":                 \n",
    "        return AdaBoostClassifier(DecisionTreeClassifier(),
algorithm='SAMME', n_estimators=200)\n",
    "    if str == \"Neural Networks\":\n",
    "        return MLPClassifier(alpha=1)\n",
    "    \n",
    "def func(str, X_train, Y_train, X_test, Y_test, methods,
mse):\n",
    "    cl = getMethod(str)\n",
    "    cl.fit(X_train, Y_train)\n",
    "    c = methods.get(str, list())\n",
    "    c.append(cl)\n",
    "    methods[str] = c\n",
    "    dtpred = cl.predict(X_test)  \n",
    "    m = mse.get(str, list())\n",
    "    m.append(mean_squared_error(Y_test, dtpred))\n",
    "    mse[str] = m\n",
    "    return (methods, mse, dtpred)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "transformX['target'] = data.iloc[:,-1]\n",
    "transformX.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#transformX = transformX.drop('target', axis=1)\n",
    "\n",
    "seq = [int(x) for x in
transformX.shape[0]*np.random.rand(round(0.6*transformX.shape[0]),1)]\
n",
    "train_fold = transformX.iloc[seq]\n",
    "newX = transformX.drop('target', axis=1)\n",
    " \n",
```

```
    "train_fold_output = data.iloc[seq, -1]\n",
    "train_fold_input = train_fold.drop('target', axis=1)\n",
    "    \n",
    "s = func('SVM', \n",
    "                          train_fold_input, \n",
    "                          train_fold_output, \n",
    "                          newX,\n",
    "                          data.iloc[:, -1],\n",
    "                          dict(),\n",
    "                          dict())\n",
    "transformX = transformX.drop('target', axis=1)\n",
    "transformX['svm_rgb'] = s[2]\n",
    "\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "transformX.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import time"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "start_time = time.time()\n",
    "skf = StratifiedKFold(n_splits=10, random_state = True)\n",
    "methods = {}\n",
    "mse = {}\n",
    "fold_count = 0\n",
    "for train, test in skf.split(transformX, data.iloc[:, -1]):\n",
    "    print(\"Processing fold %s\" % fold_count)\n",
    "    train_fold_input = transformX.iloc[train]\n",
    "    test_fold_input = transformX.iloc[test]\n",
    " \n",
    "    train_fold_output = data.iloc[train, -1]\n",
    "    test_fold_output = data.iloc[test, -1]\n",
    "          \n",
    "    list_of_methods = [\"DecisionTreeClassifier\",
\"GaussianNB\", \"SVM\", \"AdaBoost\", \"Neural Networks\"]\n",
    "    \n",
```

```
  "    for method in list_of_methods:\n",
  "        s = func(method, \n",
  "                            train_fold_input, \n",
  "                            train_fold_output, \n",
  "                            test_fold_input, \n",
  "                            test_fold_output, \n",
  "                            methods,\n",
  "                            mse)\n",
  "        methods = s[0]\n",
  "        mse = s[1]\n",
  " \n",
  "  # Done with the fold\n",
  "    fold_count += 1\n",
  "print(\"--- %s seconds ---\" % (time.time() - start_time))"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "def takeFirst(elem):\n",
  "    return elem[0]\n",
  "\n",
  "errors=[]\n",
  "for k,v in mse.items():\n",
  "    res = np.mean(v)\n",
  "    errors.append((res, k))\n",
  "errors.sort(key = takeFirst)\n",
  "errors"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "def finalModel(data, errors, methods):\n",
  "    getModels = methods[errors[0][1]]\n",
  "    df = pd.DataFrame()\n",
  "    i=0\n",
  "    for model in getModels:\n",
  "        pred = model.predict(data)  \n",
  "        df[str(i)]=pred\n",
  "        i=i+1\n",
  "    return df    "
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "transformX = transformX.drop('target', axis=1)"
 ]
},
```

```
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "df = finalModel(transformX, errors, methods)\n",
  "result = [round(x) for x in df.mean(axis=1)]\n",
  "mean_squared_error(result, Y)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "import json"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "with open('mse.txt') as json_file:  \n",
  "    ms = json.load(json_file)\n",
  "ms"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "#mthds[path]=methods\n",
  "ms[path]=mse"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "ms"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "metadata": {},
 "outputs": [],
 "source": [
  "with open('mse.txt', 'w') as outfile:  \n",
```

```
    "    json.dump(ms, outfile)"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.7.0"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```

ДОДАТОК Б

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import json"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "with open('mse.txt') as json_file:  \n",
    "    mse = json.load(json_file)\n",
    "mse"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "paths = list(mse.keys())\n",
    "len(paths)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import pandas as pd\n",
    "import numpy as np\n",
    "i = 26\n",
    "path = paths[i]\n",
    "data = pd.read_csv(path, na_values='?', na_filter=True)\n",
    "data.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
```

```
  "source": [
   "from sklearn import preprocessing \n",
   "from sklearn.preprocessing import MinMaxScaler "
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "#data=data.drop(columns=['Unnamed: 0.1', 'Unnamed: 0'])\n",
   "data.head()"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "data= data.drop_duplicates()    \n",
   "Y = data.iloc[:,-1]\n",
   "X = data.iloc[:,0:data.shape[1]-1]\n",
   "X.head()\n",
   "X.shape[0]"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "X[\"target\"] = Y\n",
   "X=X.dropna() \n",
   "X.head()\n",
   "X.shape[0]"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "#X.iloc[:, 2] = X.iloc[:,2].replace('5more', 5)\n",
   "#X.iloc[:,3] = X.iloc[:,3].replace('more', 6)\n",
   "#X.iloc[:,2]=pd.to_numeric(X.iloc[:,2])\n",
   "#X.iloc[:,3]=pd.to_numeric(X.iloc[:,3])\n",
   "X.dtypes"
  ]
 },
 {
  "cell_type": "code",
```

```
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#for i in range(X['boat'].shape[0]):\n",
    "#    if X.iloc[i,8] == 0:\n",
    "#        X.iloc[i,8] = 0\n",
    "#    else:\n",
    "#        X.iloc[i,8] = 1\n",
    "#X['Dependents']=X['Dependents'].replace('3+', 4)\n",
    "#X['Dependents']=pd.to_numeric(X['Dependents'])\n",
    "#X.dtypes"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "from sklearn.preprocessing import normalize\n",
    "X.iloc[:,0:data.shape[1]-1] = normalize(X.iloc[:,0:data.shape[1]-
1], norm='max', axis=1)\n",
    "X.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn.decomposition import PCA"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "Y = X.iloc[:,-1]\n",
    "X = X.iloc[:,0:data.shape[1]-1]\n",
    "pca = PCA(n_components=X.shape[1])\n",
    "pca.fit(X)\n",
    "eigenvalues = pca.explained_variance_ratio_\n",
    "count = 0\n",
    "sum = 0\n",
    "for eig in eigenvalues:\n",
    "    sum = sum + eig\n",
    "    count = count + 1\n",
    "    if sum > 0.8:\n",
    "        break\n",
```

```
   "if count == 1:\n",
   "    count = 2"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
   "scrolled": true
  },
  "outputs": [],
  "source": [
   "pca = PCA(n_components=count)\n",
   "pca.fit(X)\n",
   "transformX = pd.DataFrame(pca.transform(X))\n",
   "transformX.head()"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "import matplotlib.pyplot as plt"
  ]
 },
 {
  "cell_type": "markdown",
  "metadata": {},
  "source": [
   "plt.scatter(transformX[0], transformX[1], \n",
   "            c=Y.map({0: 'blue', 1: 'orange'}));\n",
   "plt.scatter(transformX[0], transformX[2], \n",
   "            c=Y.map({0: 'blue', 1: 'orange'}));\n",
   "plt.scatter(transformX[0], transformX[3], \n",
   "            c=Y.map({0: 'blue', 1: 'orange'}));\n",
   "plt.scatter(transformX[1], transformX[2], \n",
   "            c=Y.map({0: 'blue', 1: 'orange'}));\n",
   "plt.scatter(transformX[1], transformX[3], \n",
   "            c=Y.map({0: 'blue', 1: 'orange'}));\n",
   "plt.scatter(transformX[2], transformX[3], \n",
   "            c=Y.map({0: 'blue', 1: 'orange'}));"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "np.sum(Y)"
  ]
 },
 {
```

```
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "Y.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "seq = [int(x) for x in
transformX.shape[0]*np.random.rand(round(0.6*transformX.shape[0]),1)]\
n",
    "transformX['target'] = Y\n",
    "train_fold = transformX.iloc[seq]\n",
    "newX = transformX.drop('target', axis=1)\n",
    " \n",
    "trainY = train_fold['target']\n",
    "trainX = train_fold.drop('target', axis=1)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#point = 1\n",
    "#while point in seq:\n",
    "#    point = int(transformX.shape[0]*np.random.rand(1,1))\n",
    "testX=newX\n",
    "testY=Y"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from IPython.display import clear_output\n",
    "\n",
    "def method(trainX, trainY, testX):\n",
    "    outs = []\n",
    "    for (idx1, testRow) in testX.iterrows():\n",
    "        temp = trainX\n",
    "        temp = temp.sub(testRow, axis=1)**2\n",
    "        d = temp.sum(axis=1)\n",
    "        d = [np.sum(d) if x==0 else x for x in d]\n",
    "        w=[1/(x**2) for x in d]\n",
    "        sumW=np.sum(w)\n",
```

```
   "        if np.sum(trainY*w)/sumW > 0.5:\n",
   "            out = 1\n",
   "        else:\n",
   "            out = 0\n",
   "        outs.append(out)\n",
   "        clear_output()\n",
   "        print(str(idx1/(testX.shape[0]-1)*100)+'%')\n",
   "    return outs"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "outs = method(trainX, trainY, testX)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "from sklearn.metrics import mean_squared_error"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "error = mean_squared_error(outs, testY)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "meth = mse[path]\n",
   "meth"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": [
   "meth['kNN']=[error]\n",
   "meth"
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "mse[path] = meth\n",
    "mse"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "with open('mse.txt', 'w') as outfile:  \n",
    "    json.dump(mse, outfile)"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.7.0"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```