Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра математичного та комп'ютерного моделювання

Дипломна робота

бакалавра

на тему: «Дослідження класичних методів машинного навчання»

«Research of classical methods of machine learning»

Виконала: студентка денної форми навчання
спеціальності 113 Прикладна математика

Філістовіч Олександра Валентинівна

Керівник: к. ф.-м. н., доц. Таірова М.С.
Рецензент: к. ф.-м. н., доц. Вербіцький В.В.

Рекомендовано до захисту:
Протокол засідання кафедри
№ ___ від _____ р.

Завідувач кафедри

_____ _____
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №_____
протокол № ____ від _____ р.
Оцінка_____/___ ___/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _____
(підпис) (прізвище, ініціали)

Одеса – 2021

Зміст

1. Вступ

Інтерес до завдань машинного навчання (Machine Learning) виник при появі проблеми рішення машиною нетривіальних завдань, які не можуть бути розв’язані за допомогую алгоритмів та статистики, а також засновані на емпіричних даних, таких як зображення, звукові сигнали, письмова інформація. Іншими словами, основним завданням було побудова алгоритму здатного до навчання.

Великою частиною машинного навчання є інтелектуальний аналіз поставлених завдань і вирішуваних питань. Та одну з найважливіших задач викинує добре підібраний алгоритм. Так наприклад швидкий аналіз данного масиву даних та знання про алгоритми може значно прискорити та покращити розв’язок задачі.

У цій роботі розглянуті методи класичного машинного навчання та різниця в їх роботі на різних масивах данних.

Проаналізовано наступні класичні методи машинного навчання.

- Лінійна регрессія,
- Логістична регрессія,
- К-ближчих сусідів,
- Дерево рішень,
- Метод опорних векторів,
- Випадковий ліс
- Наївний Байесовський метод

Провести навчання на наступних масивах даних та провести аналіз робити цих методів на різних даних.

- Рак молочної залози
- Аналіз рукописних цифр
- Види вина
- Обличчя Оліветті
- Розпізнавання рослин

2. Лінійна регресія

2.1 Опис методу

Лінійна регресія (Linear regression) - одна з найважливіших і широко використовуваних технік регресії. Ця найпростіший метод регресії. Одним з його переваг є легкість інтерпретації результатів.

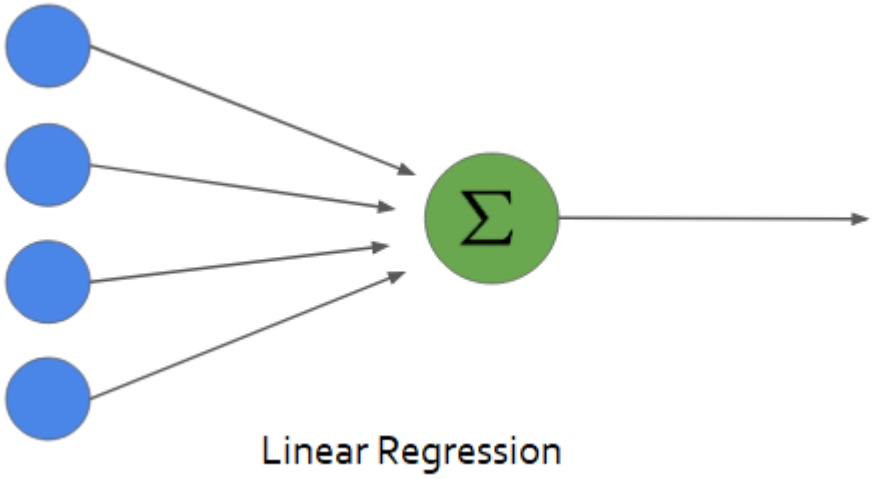

Рисунок 1: схема методу

Регресія шукає відносини між змінними. Для прикладу можна взяти співробітників якої-небудь компанії і зрозуміти, як значення зарплати залежить від інших даних, таких як досвід роботи, рівень освіти, роль, місто, в якому вони працюють, і так далі.

Регресія вирішує проблему єдиного уявлення даних аналізу для кожного працівника. Причому досвід, освіту, роль і місто - це незалежні змінні при залежною від них зарплати.

Таким же способом можна встановити математичну залежність між цінами будинків в певній галузі, кількістю кімнат, відстанню від центру і т.д.

Регресія розглядає деяке явище і ряд спостережень. Кожне спостереження має дві і більше змінних. Припускаючи, що одна змінна залежить від інших, робляться спроби побудувати відносини між ними.

Іншими словами, потрібно знайти функцію, яка відображає залежність одних змінних або даних від інших. Зовсім дані називаються залежними змінними, виходами або відповідями.

Незалежні дані називаються незалежними змінними або входами.

Зазвичай в регресії присутній одна безперервна і необмежена залежна змінна. Вхідні змінні можуть бути необмеженими, дискретними або категоричними даними.

Загальною практикою є позначення даних на виході – y, вхідних даних – X.

У випадку з двома або більш незалежними змінними, їх можна представити у вигляді вектора $X = (x_1, x_2, \dots, x_m)$, де m – кількість вхідних змінних.

2.2 Постановка задачі та її застосування

Лінійна регресія корисна для прогнозування відповіді на нові умови. Можна вгадати споживання електроенергії в житловому будинку з даних температури, часу доби і кількості мешканців.

Лінійна регресія використовується в багатьох галузях: економіка, комп'ютерні та соціальні науки, інше. Її важливість зростає з доступністю великих даних.

Лінійна регресія деякої залежної змінної y на набір незалежних змінних $x = (x_1, x_2, \dots, x_r)$, де r – це число входів, передбачає, що лінійне відношення між y та x: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$. Це рівняння регресії. $\beta_0, \beta_1, \dots, \beta_r$ – коефіцієнти регресії, та ε – випадкова помилка.

Лінійна регресія обчислює оціночні функції коефіцієнтів регресії або просто прогнозовані ваги вимірювання, що позначаються як $b_0, b_1, \dots, b_r$. Вони визначають оціночну функцію регресії $f(x) = b_0 + b_1 x_1 + \cdots + b_r x_r$. Ця функція захоплює залежності між входами і виходом досить добре.

Для кожного результату спостереження $\mathrm{i} = 1, \dots, \mathrm{n}$, оціночна або передбачена відповідь $f(x_i)$ повинна бути якомога ближче до відповідного фактичного відповіді y_i. Різниці від $y_i - f(x_i)$ для всіх результатів спостережень називаються залишками. Регресія визначає кращі прогнозовані ваги вимірювання, які відповідають мінімальним залишкам.

Для отримання кращих ваг, потрібно мінімізувати суму залишкових квадратів (SSR) для всіх результатів спостережень $SSR = \sum_i (y_i - f(x_i))^2$. Цей підхід називається методом найменших квадратів.

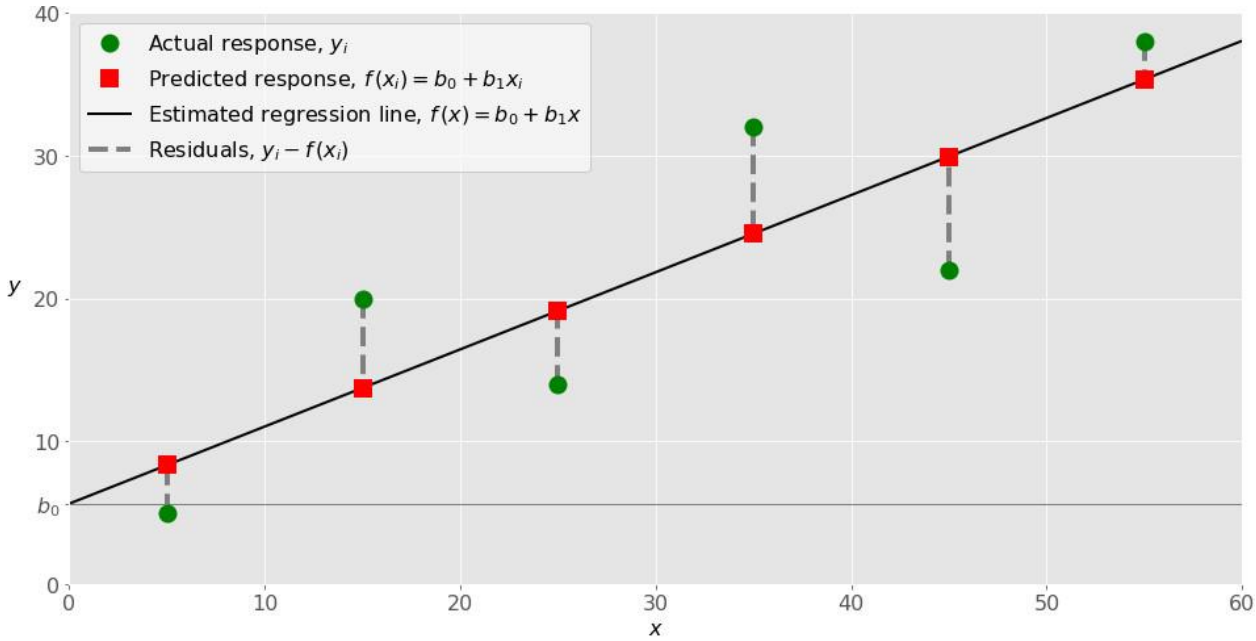

Рисунок 2: мінімізація похибки

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Реалізація простої лінійної регресії починається з заданим набором пар (зелені кола) входів-виходів $(x - y)$. Ці пари - результати спостережень. Спостереження, крайнє зліва (зелений круг) має на вході $x = 5$ і відповідний вихід (відповідь) $y = 5$. Наступне спостереження має $x = 15$ та $y = 20$, та т.д.

Оціночна функція регресії (чорна лінія) виражається рівнянням $f(x) = b_0 + b_1 x$. Потрібно розрахувати оптимальні значення прогнозованих ваг b_0 та b_1 для мінімізації SSR і визначити оцінну функцію регресії. Величина b_0, також звана відрізком, показує точку, де розрахункова лінія регресії перетинає вісь y. Це значення розрахункового відповіді $f(x)$ для $x = 0$. Величина b_1 визначає нахил розрахункової лінії регресії.

Передбачені відповіді (червоні квадрати) - точки лінії регресії, відповідні вхідним значенням. Для входу $x = 5$ передбачений відповідь дорівнює $f(5) = 8.33$ (представлений крайнім лівими квадратом).

Залишки (вертикальні пунктирні сірі лінії) можуть бути обчислені як $y_i - f(x_i) = y_i - b_0 - b_1 x_i$ для $i = 1, \ldots, n$. Вони являють собою відстані між зеленими і червоними пунктами. При реалізації лінійної регресії ви мінімізуєте ці відстані і робите червоні квадрати як можна ближче до визначеним зеленим колам.

3. Логістична регресія

3.1 Опис методу

Логістична регресія (Logistic regression) - це алгоритм класифікації машинного навчання, який використовується для прогнозування ймовірності категоріальної залежною змінною. У логістичної регресії залежна змінна є бінарною змінною, що містить дані, закодовані як 1 (так, успіх і т.п.) або 0 (ні, провал і т.п.). Іншими словами, модель логістичної регресії передбачує $P(Y = 1)$ як функцію X.

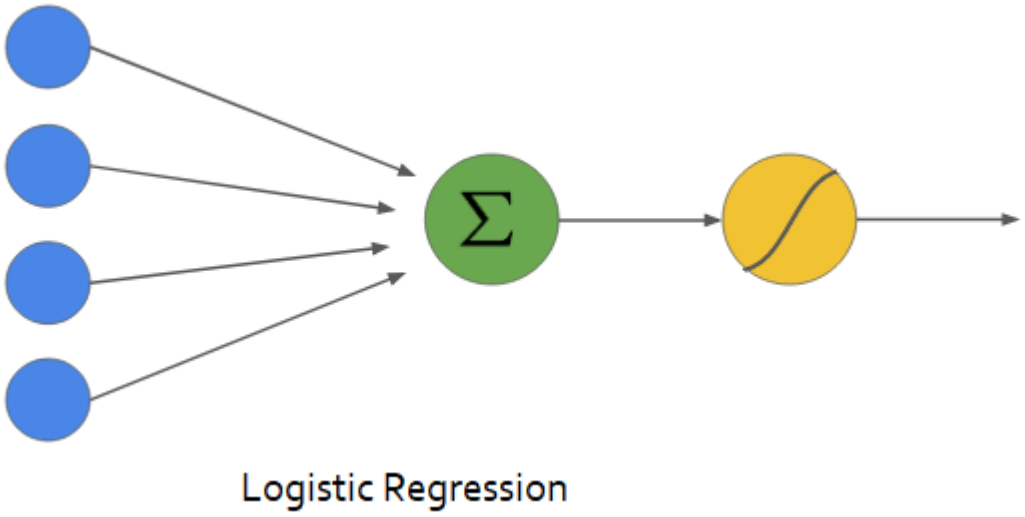

Рисунок 3: схема методу

Логістичну регресію відносять до моделей бінарного вибору. Регресійна модель бінарного вибору - це регресійна модель, в якій залежна змінна дихотомическая (бінарна). Залежна змінна може приймати лише два значення і означати, наприклад, приналежність до певної групи (надійний клієнт або ненадійний клієнт банку), що робляться дію (покупка товару), варіанти відповіді «так» або «ні» (подобається реклама або не подобається). Будувати звичайну лінійну регресійну модель з бінарними залежними змінними не можна. У цьому випадку неможливо буде інтерпретувати передбачені по регресії в безперервної кількісної шкалою значення залежної змінної.

Значення факторів в моделях бінарного вибору повинні бути виміряні в кількісної шкалою. Також в моделі бінарного вибору можна включати в якості факторів категоріальні змінні. Отже, в моделях бінарного вибору будується регресійна модель залежності ймовірності того, що результативна дихотомічна змінна прийме значення 0 або 1 при заданому значенні факторів.

Для моделювання ймовірності дихотомічної залежної змінної підбирають спеціальну монотонно зростаючу функцію, яка може приймати значення тільки від 0 до 1.

В якості спеціальної функції в моделях бінарного вибору зазвичай використовують:

- логістичну функцію;
- функцію стандартного нормального розподілу.

Моделі бінарного вибору на основі логістичної функції називаються логістичної регресією або логит-моделлю.

Моделі бінарного вибору на основі функції стандартного нормального розподілу називають пробитий-моделями.

За допомогою логістичної регресії прогнозується ймовірність відгуку для залежної змінної від включених в модель незалежних змінних. На основі прогнозних значень ймовірності можна зробити класифікацію всіх спостережень на дві групи. Окремим аналізом при побудові моделі логістичної регресії є аналіз ROC-кривих (Receiver Operator Characteristic). ROC-аналіз дозволяє вибрати оптимальне значення порогового значення ймовірності для класифікації. ROC-крива - крива, яка використовується для представлення результатів бінарної класифікації та оцінки ефективності класифікації.

3.2 Постановка задачі та її застосування

У прикладному статистичному аналізі логістична регресія використовується для вирішення двох завдань: моделювання взаємозв'язку і класифікації спостережень. Логістичну регресію застосовують при проведення клінічних досліджень в медицині, в банківському скоринг для побудови рейтингу позичальників і управління кредитними ризиками, в споживчому скоринг для моделювання поведінки покупців та інших сферах.

Лінійна регресія дає безперервний результат, а логістична регресія забезпечує постійний результат. Прикладом безперервного виведення є ціна будинку і ціна акцій. Приклад дискретних вихідних даних - це прогнозування того, хворий пацієнт раком, і прогнозування догляду клієнта. Лінійна регресія оцінюється з використанням звичайних найменших квадратів (OLS), а логістична регресія оцінюється з використанням методу оцінки максимальної правдоподібності (MLE).

Що значить, якщо необхідний якомога точніший результат краще використовувати логістичну регресію.

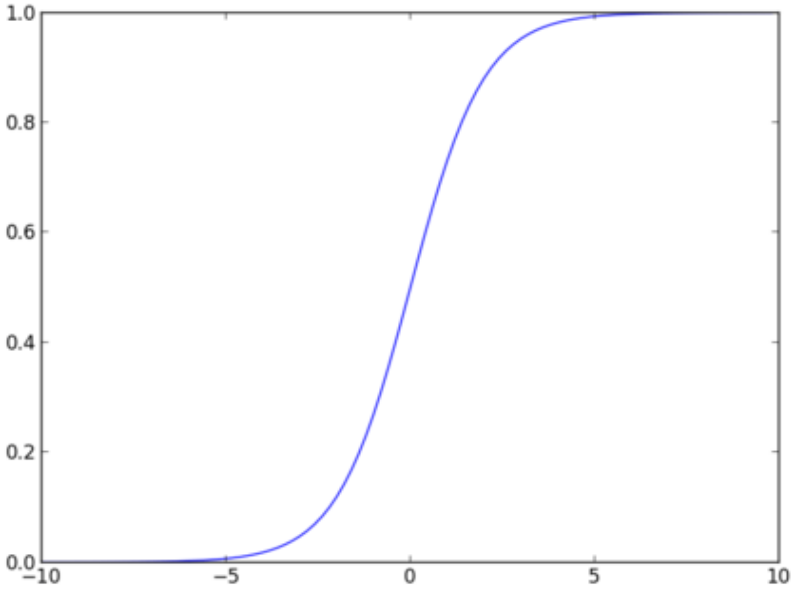

Рисунок 4: логістична функція

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

Сигмоїдальна функція, також звана логістичною функцією, дає S-подібну криву, яка може приймати будь-яке дійсне число і відображати його в значення від 0 до 1. Якщо крива переходить в позитивну нескінченність, прогноз стане 1, а якщо крива переходить в негативну нескінченність, прогноз стане 0. Якщо вихідний сигнал сігмоідної функції більше 0,5, ми можемо класифікувати результат як 1 або ТАК, а якщо він менше 0,5, ми можемо класифікувати його як 0 або НІ. Якщо вихід дорівнює 0,75, можна сказати з точки зору ймовірності наступним чином: Імовірність того, що пацієнт буде хворіти на рак, становить 75 відсотків.

Типи логістичної регресії:

- Двоичная логістична регресія: цільова змінна має тільки два можливих результату, наприклад, спам або не спам, рак або відсутність раку.
- Поліноміальна логістична регресія: цільова змінна має три або більше номінальних категорій, таких як прогноз типу вина.
- Порядкова логістична регресія: цільова змінна має три або більше порядкових категорій, наприклад рейтинг ресторану або продукту від 1 до 5.

4. K-найближчих сусідів

4.1 Опис методу

K-найближчих сусідів (K-nearest Neighbors або KNN) - це один з найпростіших алгоритмів класифікації, також іноді використовується в задачах регресії. Завдяки своїй простоті, він є хорошим прикладом, з якого можна почати знайомство з областю Machine Learning. Метод k-найближчих сусідів — метричний алгоритм для автоматичної класифікації об'єктів. Основним принципом методу найближчих сусідів є те, що об'єкт присвоюється тому класу, який є найбільш поширеним серед сусідів даного елемента.

KNN - це непараметричний алгоритм ледачого навчання. Непараметричний означає, що немає ніяких припущень для базового розподілу даних. Іншими словами, структура моделі визначається на основі набору даних. Це буде дуже корисно на практиці, коли більшість наборів даних реального світу не слід математичним теоретичним припущенням. Лінивий алгоритм означає, що для створення моделі йому не потрібні ніякі навчальні дані. Всі дані навчання використовуються на етапі тестування. Це прискорює навчання, а етап тестування - повільніше і дорожче. Дорогий етап тестування вимагає часу і пам'яті. У гіршому випадку KNN потрібно більше часу для сканування всіх точок даних, а сканування всіх точок даних зажадає більше пам'яті для зберігання даних навчання.

4.2 Постановка задачі та її застосування

KNN - дуже простий, зрозумілий, універсальний і один з кращих алгоритмів машинного навчання. KNN використовується в різних прикладних задачах, таких як фінанси, охорона здоров'я, політологія, виявлення почерку, розпізнавання зображень і розпізнавання відео. У кредитних рейтингах фінансові інститути прогнозують кредитний рейтинг клієнтів. При видачі кредиту банківські інститути прогнозують, чи буде кредит безпечний або ризикований. В політології поділ потенційних виборців на два класи буде голосувати чи не голосувати. Алгоритм KNN використовується для задач класифікації і регресії. Алгоритм KNN, заснований на підході подібності ознак.

KNN краще працює з меншою кількістю функцій, ніж з великою кількістю функцій. Можна сказати, що при збільшенні кількості функції споживають більше даних. Збільшення розмірів також призводить до проблеми перенавчання. Щоб уникнути перенавчання, необхідні дані повинні будуть рости експоненціально зі збільшенням кількості вимірювань. Ця проблема більш високого виміру відома як Curse of Dimensionality.

Щоб впоратися з проблемою розмірності, необхідно виконати аналіз основних компонентів перед застосуванням будь-якого алгоритму машинного навчання, або ви також можете використовувати підхід вибору функцій. Дослідження показали, що у великих вимірах евклідова відстань марна. Отже, можна віддати перевагу іншим заходи, такі як косинусное схожість, на які значно менше впливає висока розмірність.

У KNN K - кількість найближчих сусідів. Вирішальним фактором є кількість сусідів. K зазвичай є непарним числом, якщо кількість класів одно 2. Коли K = 1, тоді алгоритм відомий як алгоритм найближчого сусіда. Це найпростіший випадок. Припустимо, що P1 - це точка, для якої мітка повинна передбачати. Спочатку ви знаходите найближчу точку до P1, а потім мітку найближчої точки, призначену для P1.

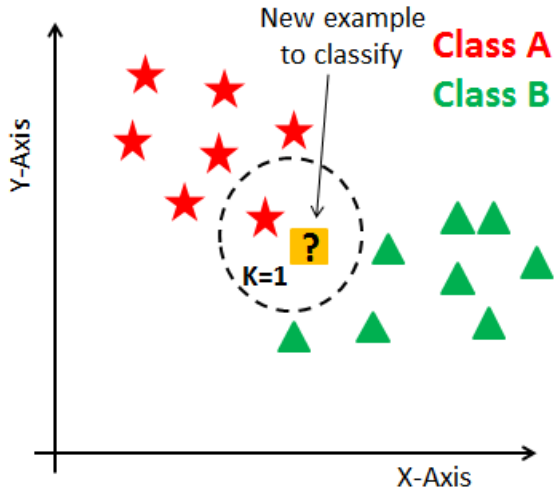

Рисунок 5: схема методу

Припустимо, що $P1$ - це точка, для якої мітка повинна передбачати. Спочатку ви знаходите k найближчу точку до $P1$, а потім класифікуєте точки більшістю голосів її k сусідів. Кожен об'єкт голосує за свій клас, і клас з найбільшою кількістю голосів приймається в якості прогнозу. Для пошуку найближчих схожих точок відстань між точками визначається за допомогою таких заходів відстані, як евклідова відстань, відстань Hamming, Manhattan і Minkowski. KNN має наступні основні кроки:

- Розрахувати відстань

- Знайдіть найближчих сусідів

- Голосування за прапори

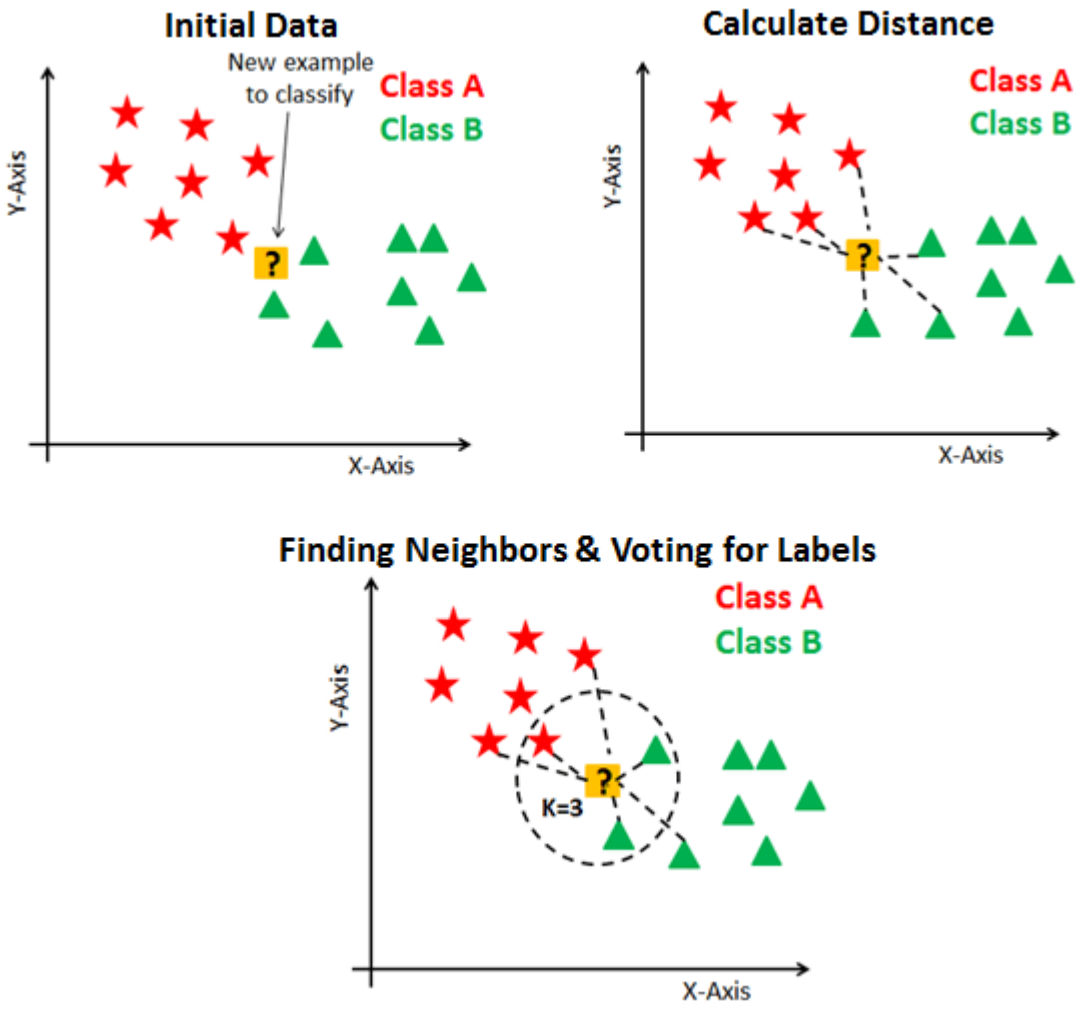

Рисунок 6: схема пошуку відповіді

5. Дерево рішень

5.1 Опис методу

Дерево рішень (Decision Tree) - рішення задачі навчання з учителем, заснований на тому, як вирішує завдання прогнозування людина. У загальному випадку - це k-ічное дерево з вирішальними правилами в нелістових вершинах (вузлах) і деякому укладанні про цільової функції в листових вершинах (прогнозом).

Дерево рішень - це деревоподібна структура, подібна блок-схемі, де внутрішній вузол являє функцію (або атрибут), гілка представляє правило прийняття рішення, а кожен кінцевий вузол являє результат. Самий верхній вузол в дереві рішень відомий як кореневий вузол. Він навчається поділу на основі значення атрибута. Він розбиває дерево рекурсивно, викликаючи рекурсивне розбиття. Ця структура, схожа на блок-схему, допомагає приймати рішення. Це візуалізація у вигляді блок-схеми, яка легко імітує мислення людського рівня. Ось чому дерева рішень легко зрозуміти і інтерпретувати.

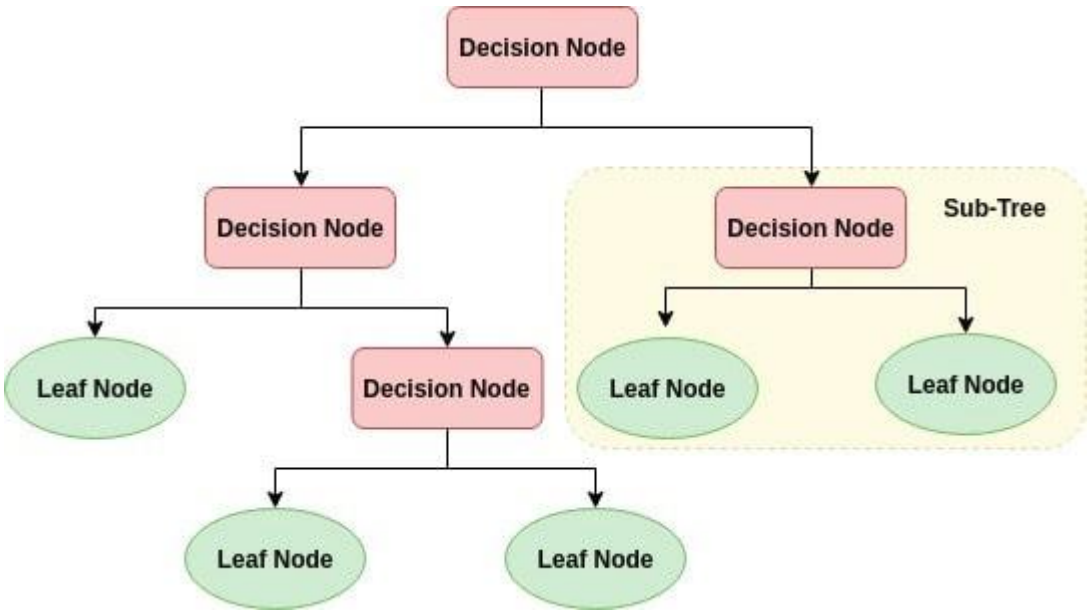

Рисунок 7: схема методу

Дерево рішень - це білий ящик алгоритму машинного навчання. Він розділяє внутрішню логіку прийняття рішень, яка недоступна в алгоритмах типу чорного ящика, таких як нейронна мережа. Його час навчання швидше в порівнянні з алгоритмом нейронної мережі. Тимчасова складність дерев рішень є функцією кількості записів і кількості атрибутів в даних. Дерево рішень - це метод без розподілу або непараметричний метод, який не залежить від припущень про розподіл ймовірностей. Дерева рішень можуть обробляти багатовимірні дані з хорошою точністю.

5.2 Постановка задачі та її застосування

Як менеджер з маркетингу, необхідний набір клієнтів, які з найбільшою ймовірністю куплять ваш продукт. Так можна заощадити свій маркетинговий бюджет, знайшовши свою аудиторію. Як кредитний менеджер, необхідно виявляти ризиковані заявки з результатом, коли справа дійде до видачі кредиту, щоб знизити відсоток невиплат кредиту. Цей процес класифікації клієнтів в групу потенційних і непотенційного клієнтів або безпечних або ризикованих заявок на отримання кредиту відомий як проблема класифікації. Класифікація - це двоетапний процес: крок навчання і крок прогнозування. На етапі навчання модель розробляється на основі заданих систему адаптації. На етапі прогнозування модель використовується для прогнозування відповіді на задані дані. Дерево рішень - один з найпростіших і популярних алгоритмів класифікації для розуміння та інтерпретації. Його можна використовувати як для задач класифікації, так і для завдань регресії.

Основна ідея будь-якого алгоритму дерева рішень полягає в наступному:

- Вибрати кращий атрибут, використовуючи заходи вибору атрибутів (ASM), щоб розділити записи.
- Зробіть цей атрибут вузлом прийняття рішень і розбийте набір даних на більш дрібні підмножини.
- Починає побудова дерева, рекурсивно повторюючи цей процес для кожного дочірнього елемента, поки не буде виконано одну з умов:
- Все кортежі належать одному і тому ж значенню атрибута.
- Більше немає зайвих атрибутів.
- Більше немає примірників.

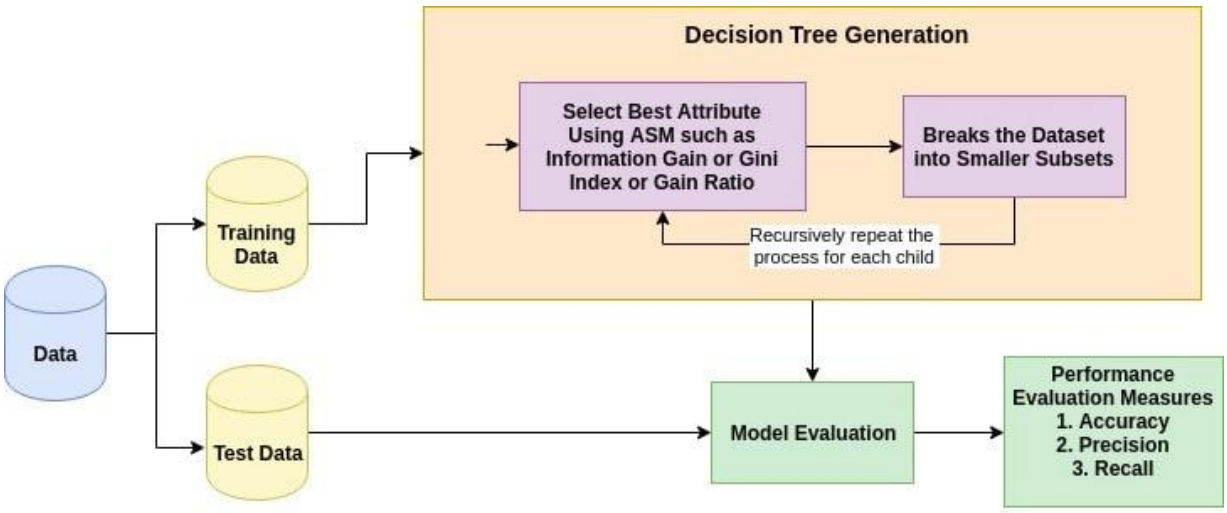

Рисунок 8: алгоритм навчання

Міра вибору атрибута - це евристика для вибору критерію розбиття, який розбиває дані найкращим чином. Це також відомо як правила поділу, тому що вони допомагають нам визначати точки зупину для кортежів на даному вузлі. ASM привласнює рейтинг кожної функції (або атрибуту), пояснюючи даний набір даних. Атрибут кращого результату буде обраний як атрибут поділу (Джерело). У разі атрибута з безперервним значенням також необхідно визначити точки поділу для гілок. Найбільш популярними критеріями відбору є Information Gain, Gain Ratio і Gini Index.

Information Gain:

$$Info(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

$$Info_A(D) = \sum_{j=1}^{V} \frac{|D_j|}{|D|} \; X \; Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

Gain Ratio:

$$Split\ Info_A(D) = \sum_{j=1}^{V} \frac{|D_j|}{|D|} \times \log_2 \left(\frac{|D_j|}{|D|} \right)$$

$$Gain\ Ratio(A) = \frac{Gain(A)}{Split\ Info_A(D)}$$

Gini Index:

$$Gini(D) = 1 - \sum_{i=1}^{m} Pi^2$$

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

6. Метод опорних векторів

6.1 Опис методу

Метод опорних векторів (Support Vector Machine) — алгоритм, і його концепції відносно прості. Даний алгоритм надає широке застосування на практиці і може переглядати як лінійні так і нелінійні завдання. Суть роботи “Машин” Опорних Векторів проста: алгоритм створює лінію або гіперплощина, яка розділяє дані на класи.

SVM пропонує дуже високу точність у порівнянні з іншими класифікаторами, такими як логістична регресія і дерева рішень. Він відомий своїм трюком з ядром для обробки нелінійних вхідних просторів. Він використовується в різних додатках, таких як виявлення осіб, виявлення вторгнень, класифікація електронних листів, новинних статей і веб-сторінок, класифікація генів і розпізнавання почерку.

Класифікатор розділяє точки даних за допомогою гіперплощини з найбільшим розміром поля. Ось чому класифікатор SVM також відомий як дискримінантний класифікатор. SVM знаходить оптимальну гіперплощину, яка допомагає в класифікації нових точок даних.

Оптимальна гіперплоскость - єдина гіперплоскость така, що сума відстаней від найближчих до неї (зверху і знизу) точок вибірки максимальна серед усіх, що поділяють S гіперплоскостей, розташованих на рівних від них відстанях.

6.2 Постановка задачі та її застосування

Як правило, метод опорних векторів вважається підходом до класифікації, але його можна використовувати як в задачах класифікації, так і в задачах регресії. Він може легко обробляти кілька безперервних і категоріальних змінних. SVM створює гіперплощину в багатовимірному просторі для поділу різних класів. SVM послідовно генерує оптимальну гіперплощину, яка використовується для мінімізації помилки. Основна ідея SVM - знайти максимальну маргінальну гіперплощину (ММН), яка найкращим чином поділяє набір даних на класи.

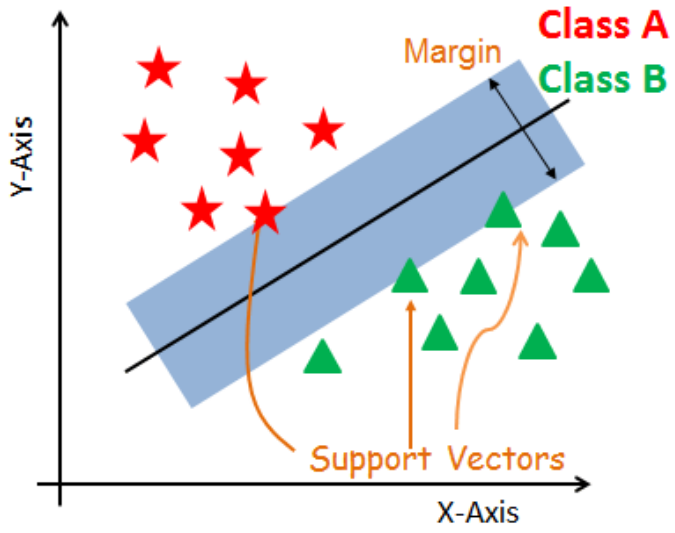

Рисунок 9: схема методу

Основна мета - найкращим чином розділити даний набір даних. Відстань між найближчими точками називається полем. Мета полягає в тому, щоб вибрати гіперплощину з максимально можливим запасом між опорними векторами в даному наборі даних.

SVM шукає максимальну маргінальну гіперплощину, виконавши наступні кроки:

- Створення гіперплощини, яка найкращим чином поділяють класи. На малюнку нижче, зліва показані три гіперплощини: чорна, синя і помаранчева. Тут синій і помаранчевий мають більш високу помилку класифікації, але чорний правильно розділяє два класи.
- Вибираємо праву гіперплощина з максимальним відділенням від найближчих точок даних, як показано на малюнку нижче, праворуч.

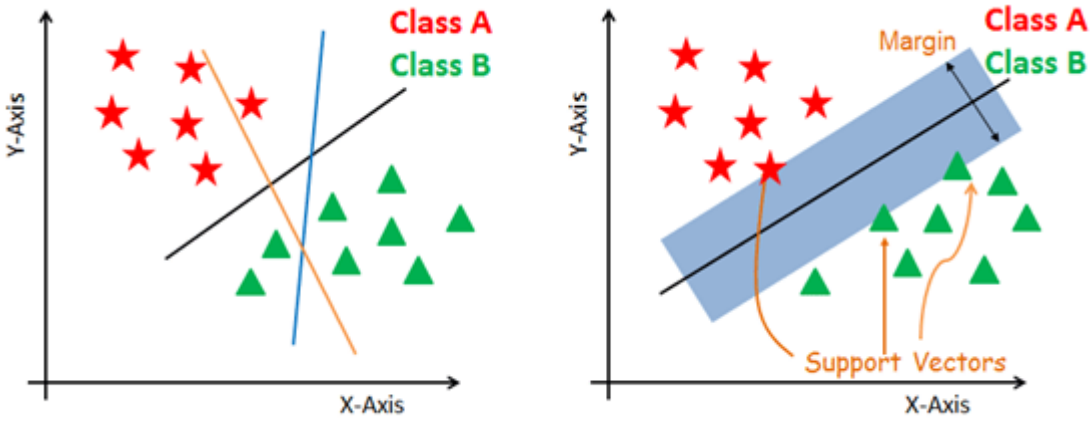

Рисунок 10: алгоритм навчання

На практиці алгоритм SVM реалізований з використанням ядра. Ядро перетворює простір вхідних даних в необхідну форму. SVM використовує техніку, звану трюком з ядром. Тут ядро бере малорозмірне вхідний простір і перетворює його в більш зверхнє простір. Іншими словами, можна сказати, що він перетворює нерозривну проблему в розривні проблеми, додаючи до неї більше вимірів. Це найбільш корисно в завданню нелінійного поділу. Виверт з ядром допоможе вам побудувати більш точний класифікатор.

7. Випадковий ліс

7.1 Опис методу

Випадковий ліс (Random Forest або RF) - це безліч дерев рішень. У задачі регресії їх відповіді усереднюють, в завданню класифікації приймається рішення голосуванням за більшістю.

Випадкові лісу - це алгоритм навчання з учителем. Його можна використовувати як для класифікації, так і для регресії. Це також самий гнучкий і простий у використанні алгоритм. Ліс складається з дерев. Кажуть, що чим більше дерев, тим міцніше ліс. Випадкові лісу створюють дерева рішень на основі випадково вибраних вибірок даних, отримують прогнози від кожного дерева і вибирають краще рішення за допомогою голосування. Це також досить хороший індикатор важливості функції.

7.2 Постановка задачі та її застосування

Випадкові лісу мають безліч застосувань, таких як механізми рекомендацій, класифікація зображень і вибір функцій. Його можна використовувати для класифікації лояльних здобувачів кредиту, виявлення шахрайства та прогнозування захворювань. Він лежить в основі алгоритму Boruta, який вибирає важливі функції в наборі даних.

Припустимо, хтось хоче відправитися в подорож і хоче вирушити в місце, яке йому сподобається.

Можна шукати в Інтернеті, читати огляди в туристичних блогах і порталах, або можна запитати своїх друзів.

Припустимо, було прийнято рішення запитати своїх друзів і поговорити з ними про їхнє минуле досвіді подорожей по різних місцях. В ході з'ясування були отримані рекомендації від кожного друга. Тепер потрібно скласти список рекомендованих місць. Потім друзі повинні проголосувати (або вибрати одне найкраще місце для поїздки) зі списку рекомендованих місць. Місце, яке набрало найбільшу кількість голосів, стане остаточним вибором для поїздки.

Вищезгаданий процес прийняття рішення складається з двох частин. По-перше, запитати друзів про їх індивідуальному досвіді подорожей і отримаєте одну рекомендацію з декількох місць, які вони відвідали. Ця частина схожа на використання алгоритму дерева рішень. Тут кожен друг вибирає місця, які він або вона вже відвідав.

Друга частина після збору всіх рекомендацій - це процедура голосування для вибору кращого місця в списку рекомендацій. Весь цей процес отримання рекомендацій від друзів і голосування за них, щоб знайти найкраще місце, відомий як алгоритм випадкових лісів.

Технічно це метод ансамблю (заснований на підході «розділяй і володарюй») дерев рішень, створених на основі випадково розділеного набору

даних. Цей набір класифікаторів дерева рішень також відомий як ліс. Окремі дерева рішень генеруються з використанням індикатора вибору атрибута, такого як Information Gain, Gain Ratio і Gini Index для кожного атрибута. Кожне дерево залежить від незалежної випадкової вибірки. У задачі класифікації кожне дерево голосує, і в якості остаточного результату вибирається найпопулярніший клас. У разі регресії кінцевим результатом вважається середнє значення всіх вихідних даних дерева. Він простіше і могутніше в порівнянні з іншими алгоритмами нелінійної класифікації.

Реалізація відбувається в 4 етапи:

- Вибір випадкових вибірок з заданого набору даних.
- Побудова дерева рішень для кожної вибірки і на виході буде результат прогнозу для кожного дерева рішень.
- Провести голосування за кожен прогнозований результат.
- Вибрати результат прогнозу з найбільшою кількістю голосів в якості остаточного.

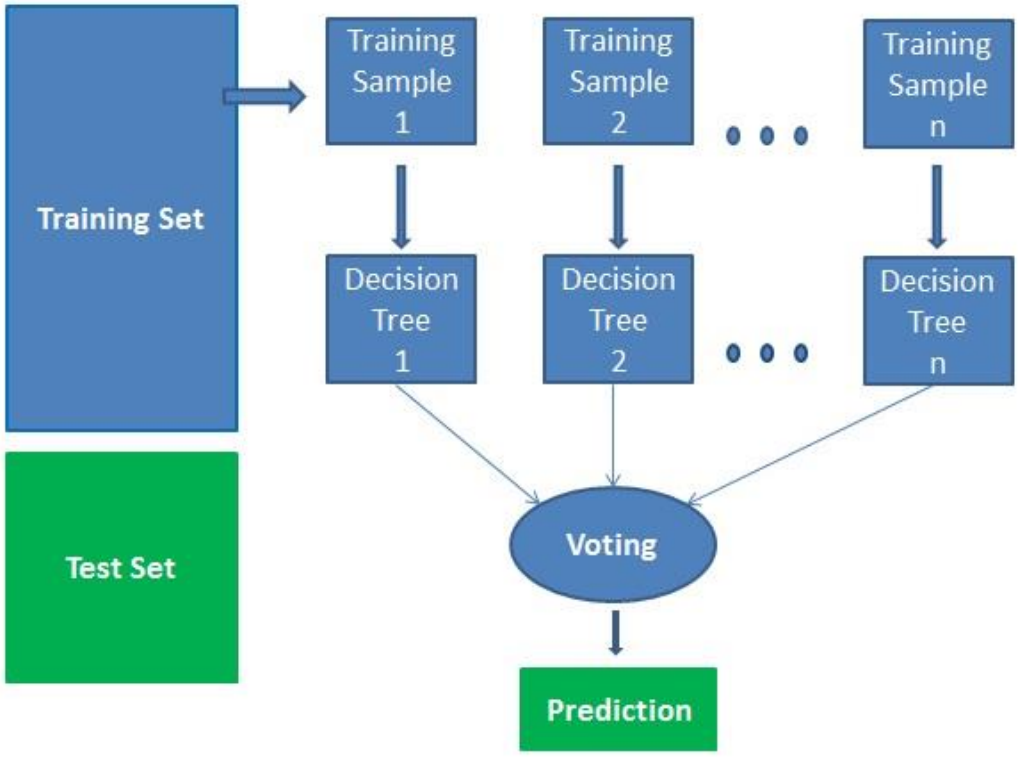

Рисунок 11: схема методу

8. Наївний байесовский класифікатор

8.1 Опис методу

Наївний байесовский класифікатор (Naive Bayes або NB) - це сімейство алгоритмів класифікації, які приймають одне припущення: Кожен параметр класифікуються даних розглядається незалежно від інших параметрів класу.

Наївний Байес - це метод статистичної класифікації, заснований на теоремі Байеса. Це один з найпростіших алгоритмів контрольованого навчання. Наївний байесовский класифікатор - це швидкий, точний і надійний алгоритм. Наївні байєсовські класифікатори мають високу точність і швидкість роботи з великими наборами даних.

Наївний байесовский класифікатор передбачає, що ефект певної функції в класі не залежить від інших функцій. Наприклад, бажаний чи ні здобувач позики в залежності від його / її доходу, попереднього кредиту і історії транзакцій, віку і місцезнаходження. Навіть якщо ці функції взаємозалежні, вони все одно розглядаються незалежно. Це припущення спрощує обчислення, і тому вважається наївним. Це припущення називається умовною незалежністю від класу.

8. 2 Постановка задачі та її застосування

Припустимо, є менеджер по продукту і він хоче розділити відгуки клієнтів на позитивні і негативні. Або як кредитний менеджер треба визначити, які здобувачі кредиту безпечні або небезпечні? Як медичний аналітик, треба передбачити, які пацієнти можуть страждати діабетом. У всіх прикладах є одна і та ж проблема для класифікації відгуків, здобувачів кредиту і пацієнтів.

Всякий раз, коли виконується класифікацію, першим кроком є розуміння проблеми і визначення потенційних функцій і мітки. Особливості - це ті характеристики або атрибути, які впливають на результати мітки. Наприклад, в разі видачі кредиту, менеджер банку визначає рід занять, дохід, вік, місцезнаходження клієнта, попередню кредитну історію, історію транзакцій і кредитний рейтинг. Ці характеристики відомі як функції, які допомагають моделі класифікувати клієнтів.

Класифікація складається з двох етапів: етапу навчання і етапи оцінки. На етапі навчання класифікатор навчає свою модель на заданому наборі даних, а на етапі оцінки він перевіряє продуктивність класифікатора. Продуктивність оцінюється на основі різних параметрів, таких як точність, помилка, точність і відгук.

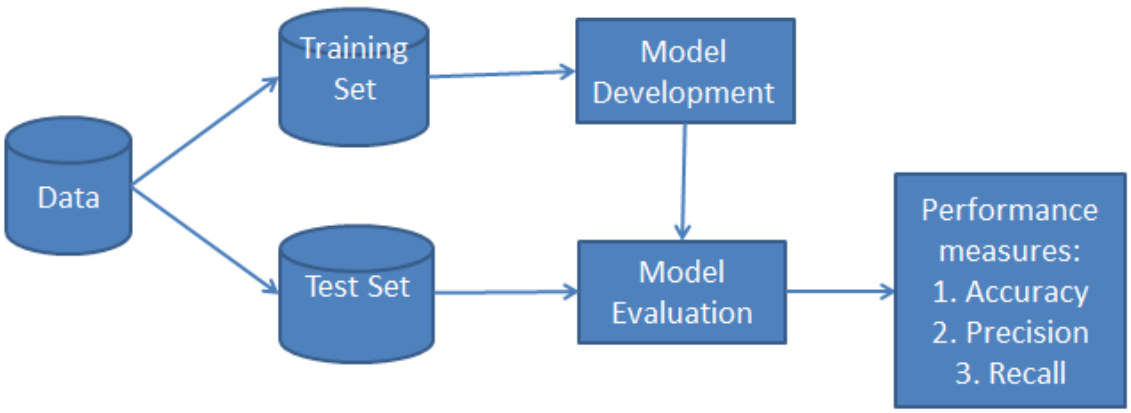

Рисунок 12: схема методу

Наївний байесовский класифікатор розраховує ймовірність події в наступних кроках:

- Розрахувати апріорну ймовірність для міток даного класу.
- Знайти ймовірність правдоподібності для кожного атрибута для кожного класу
- Помістити ці значення в формулу Байеса і обчисліть апостеріорну ймовірність.
- Подивитися, який клас має більш високу ймовірність, враховуючи, що вхідні дані належать до більш високого класу ймовірності.

Для спрощення розрахунку апріорної і апостеріорної ймовірності можна використовувати дві таблиці частоти і таблиці правдоподібності. Обидві ці таблиці допоможуть розрахувати апріорну і апостеріорну ймовірність. Таблиця періодичності містить наявність міток для всіх функцій. Є дві таблиці правдоподібності. Таблиця правдоподібності 1 показує апріорні ймовірності міток, а таблиця правдоподібності 2 показує апостеріорну ймовірність.

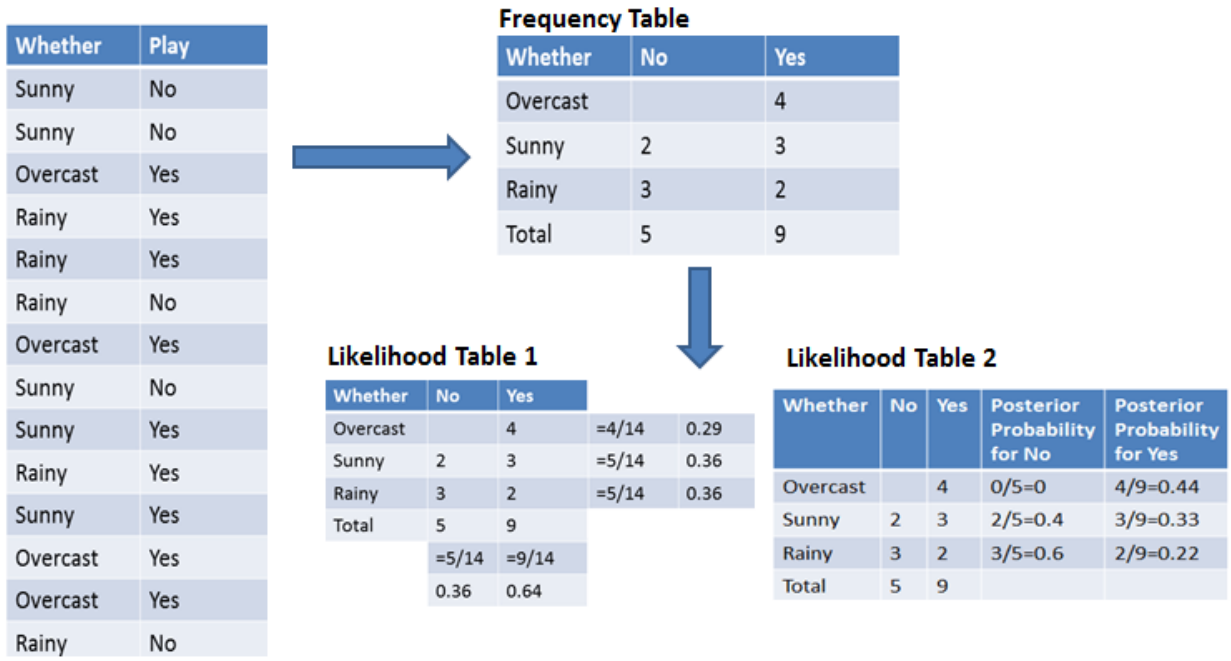

Рисунок 13: алгоритм навчання

9. Аналіз масивів даних

9.1 Діагностика раку молочної залози

Розмір: 569 x 30

Кількість досліджень: 569

Кількість атрибутів: 30 числові, атрибути та клас

Кількість відсутніх даних: немає.

Інформація про атрибути:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

Ітогові класи:

- WDBC-Malignant
- WDBC-Benign

9.2 Оптичне розпізнавання рукописних цифр

Розмір: 1797 x 64

Кількість досліджень: 1797

Кількість атрибутів: 64

Кількість відсутніх даних: немає.

Інформація про атрибути:

- 8x8 зображення цілочисельних пікселів у діапазоні 0..16

Ітогові класи:

- рукописні числа від 0 до 9.

9.3 Розпізнавання виду вина

Розмір: 178 x 13

Кількість досліджень: 178 (50 кожного класу)

Кількість атрибутів: 13 числові, атрибути та клас

Кількість відсутніх даних: немає.

Інформація про атрибути:

- Alcohol
- Malic acid
- Ash
- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

Ітогові класи:

- class_0
- class_1
- class_2

9.4 Оптичне розпізнавання осіб Оліветті

Розмір: 400 x 4096

Кількість досліджень: 400

Кількість атрибутів: 4096

Кількість відсутніх даних: немає.

Інформація про атрибути:

- дійсні числа від 0 до 1 (значення пікселів зображення 64x64)

Ітогові класи:

- 40 класів.

Зображення квантується до 256 рівнів сірого і зберігається як 8-бітове без підпису цілі числа; завантажувач перетворить їх у значення з плаваючою комою на інтервал [0, 1].

"Ціль" для цієї бази даних - це ціле число від 0 до 39, що вказує на особу людини на фотографії; однак, маючи лише 10 прикладів на клас, це відносно невеликий набір даних.

9.5 Розпізнавання виду рослини

Розмір: 150 x 4

Кількість досліджень: 150

Кількість атрибутів: 4 числові, атрибути та клас

Кількість відсутніх даних: немає.

Інформація про атрибути:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

Ітогові класи:

- Iris-Setosa
- Iris-Versicolour
- Iris-Virginica

10. Результати тестів

В excel таблиці надані результати тестів всіх, описаних в цій роботі, методів.

З результатів в таблиці можна сказати які методи справляються з навчанням краще за інших на прикладі різних за розмірами датасета. Так само результат дуже сильно залежить не тільки від кількості прикладів для навчання, але і сильно різнитися від розмірності.

Ми можемо бачити такі результати роботи алгоритмів:

- Linear Regression: достатньо швидкий на всіх датасетах, але точність найгірша в більшості випадків. Дає відносно непогану точність для малих датасетів.
- Logistic Regression: на датасетах зібраних з зображень працює повільно, крім того точність на них погана (виключення датасет Digits з малими зображеннями всього 8x8). В інших випадках дає середні результати з непоганою швидкістю.
- K-nearest Neighbors: також дає погані результати при аналізі великих зображень. Підходить для інших датасетів, особливо великих.
- Decision Tree: на всіх датасетах працює швидко, але не відрізняється великою точність.
- Support Vector Machine: швидкий на малих датасетах. Один з найточніших на всіх датасетах.
- Random forest: не підходить для малих датасетів, так як дає погану точність та швидкість. Для інших датасетів вилає середню швидкість та непогану точність.
- Naive Bayes: найшвидший алгоритм на всіх датасетах. Найкращі результати видає для великих, числових датасетів. На зображеннях дає середній результат.

Wine	Linear Regression	Accuracy	85,117%
		Time	0,001
	Logistic Regression	Accuracy	74,074%
		Time	0,011
	K-nearest Neigbours	Accuracy	68,519%
		Time	0,004
	Decision tree	Accuracy	83,333%
		Time	0,001
	Support Vector Machine	Accuracy	94,444%
		Time	0,069
	Random Forest	Accuracy	96,296%
		Time	0,132
	Naive Bayes	Accuracy	98,148%
		Time	0,001

Таблиця 1: Розпізнавання вин

Digits	Linear Regression	Accuracy	57,214%
		Time	0,016
	Logistic Regression	Accuracy	96,667%
		Time	1,063
	K-nearest Neigbours	Accuracy	98,889%
		Time	0,085
	Decision tree	Accuracy	59,444%
		Time	0,008
	Support Vector Machine	Accuracy	98,704%
		Time	0,049
	Random Forest	Accuracy	98,704%
		Time	0,272
	Naive Bayes	Accuracy	87,407%
		Time	0,006

Таблиця 2: Розпізнавання цифр

Breast Cancer	Linear Regression	Accuracy	74,786%
		Time	0,002
	Logistic Regression	Accuracy	91,813%
		Time	0,037
	K-nearest Neigbours	Accuracy	93,567%
		Time	0,029
	Decision tree	Accuracy	94,737%
		Time	0,002
	Support Vector Machine	Accuracy	95,322%
		Time	0,519
	Random Forest	Accuracy	96,491%
		Time	0,172
	Naive Bayes	Accuracy	95,322%
		Time	0,001

Таблиця 3: Діагностика раку молочної залози

Olivetti	Linear Regression	Accuracy	59,523%
		Time	0,197
	Logistic Regression	Accuracy	79,167%
		Time	70,281
	K-nearest Neigbours	Accuracy	80,000%
		Time	0,046
	Decision tree	Accuracy	6,667%
		Time	0,071
	Support Vector Machine	Accuracy	97,500%
		Time	0,319
	Random Forest	Accuracy	92,500%
		Time	1,460
	Naive Bayes	Accuracy	80,833%
		Time	0,123

Таблиця 4: Розпізнавання осіб Оліветті

Iris	Linear Regression	Accuracy	89,074%
		Time	0,001
	Logistic Regression	Accuracy	88,889%
		Time	0,005
	K-nearest Neigbours	Accuracy	95,556%
		Time	0,004
	Decision tree	Accuracy	84,444%
		Time	0,001
	Support Vector Machine	Accuracy	97,778%
		Time	0,001
	Random Forest	Accuracy	86,667%
		Time	0,127
	Naive Bayes	Accuracy	91,111%
		Time	0,001

Таблиця 5: Розпізнавання рослин

Висновки

У цій роботі розглянуті класичні методи машинного навчання, а саме:

- Linear Regression,
- Logistic Regression,
- K-nearest Neighbors,
- Decision Tree,
- Support Vector Machine,
- Random Forest, Naive Bayes.

В результаті виконання складено порівняльний аналіз у вигляді таблиці. Аналіз складено на основі наступних дата сетів:

- breast_cancer
- digits
- wine
- olivetti_faces
- iris

Методи порівнювалися за наступними критеріями: точність, час роботи і розміри використовуваних дата сетів.

Виходячи з результатів можна зробити наступні висновки: Для невеликих датасета найкраще підходять:

- Linear Regression (найшвидший)
- Support Vector Machine (точніший)
- Naive Bayes (найшвидший)

Для великих датасета з великою розмірністю підходять:

- K-nearest Neighbors (але може бути повільним)
- Support Vector Machine (точніший)
- Random forest (точніший)
- Naive Bayes

Для датасетів із зображеннями підходять:

- Logistic Regression (тільки для малих зображень)
- Random forest

Список використаної літератури

1. https://www.datacamp.com/
2. https://habr.com/
3. https://dyakonov.org/
4. https://learnmachinelearning.wikia.org/
5. https://www.machinelearningmastery.ru/
6. Вьюгін В.В. Математичні основи теорії машиного навчання та прогнозувания М.: 2013. - 387 с.

Додаток А (код програми для дослідження)

```
import time
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_digits
from sklearn.datasets import load_wine
from sklearn.datasets import fetch_olivetti_faces
from sklearn.datasets import load_iris

def linear(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = LinearRegression()
        neuralnet.fit(attributes_train, labels_train)
        accuracy = neuralnet.score(attributes_test, labels_test)

        print("== Linear Regression ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")

def logistic(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = LogisticRegression(random_state=17, penalty='elasticnet', solver='saga'
                        , l1_ratio=0.5, class_weight={0:0.4, 1:0.6})
        neuralnet.fit(attributes_train, labels_train)
```

```
        accuracy = neuralnet.score(attributes_test, labels_test)

        print("== Logistic Regression ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")


def k_neighbor(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = KNeighborsClassifier(n_neighbors=5)
        neuralnet.fit(attributes_train, labels_train)
        accuracy = neuralnet.score(attributes_test, labels_test)

        print("== K Nearest Neighbors ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")


def dec_tree(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = DecisionTreeClassifier(max_depth=4, random_state=19, splitter='random')
        neuralnet.fit(attributes_train, labels_train)
        accuracy = neuralnet.score(attributes_test, labels_test)

        print("== Decision Tree ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")


def svc(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = svm.SVC(kernel='linear')
        neuralnet.fit(attributes_train, labels_train)
        accuracy = neuralnet.score(attributes_test, labels_test)
```

```
        print("== Support Vector Machine ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")

def rand_forest(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = RandomForestClassifier(n_estimators=100, bootstrap=True, max_features='sqrt')
        neuralnet.fit(attributes_train, labels_train)
        accuracy = neuralnet.score(attributes_test, labels_test)

        print("== Random Forest ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")

def naive_bayes(attributes_train, labels_train, attributes_test, labels_test):
        start_time = time.time()
        neuralnet = GaussianNB()
        neuralnet.fit(attributes_train, labels_train)
        accuracy = neuralnet.score(attributes_test, labels_test)


        print("== Naive Bayes ==")
        print("Accuracy: " + str(round(accuracy * 100, 5)) + "%")
        print("End in " + str(round(time.time() - start_time, 5)) + " seconds \n")


def learningWithMethods(dataset):

        attributes = dataset.data
        labels = dataset.target

        print("Dataset size: ", dataset.data.shape[0], "x", dataset.data.shape[1])


        attributes_train, attributes_test, labels_train, labels_test = \
```

```
        train_test_split(attributes, labels, test_size=0.30)
        linear(attributes_train, labels_train, attributes_test, labels_test)
        logistic(attributes_train, labels_train, attributes_test, labels_test)
        k_neighbor(attributes_train, labels_train, attributes_test, labels_test)
        dec_tree(attributes_train, labels_train, attributes_test, labels_test)
        svc(attributes_train, labels_train, attributes_test, labels_test)
        rand_forest(attributes_train, labels_train, attributes_test, labels_test)
        naive_bayes(attributes_train, labels_train, attributes_test, labels_test)

if __name__ == "__main__":
        data = load_breast_cancer(); print("===== Breast canser dataset =====")
        learningWithMethods(data)

        data = load_digits(); print("===== Digits dataset =====")
        learningWithMethods(data)

        data = load_wine(); print("===== Wine dataset =====")
        learningWithMethods(data)

        data = fetch_olivetti_faces(); print("===== Olivetti dataset =====")
        learningWithMethods(data)

        data = load_iris(); print("===== Iris dataset =====")
        learningWithMethods(data)
```