МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»
Завідувач кафедри
_____ Ігор КОПЛИК
(підпис) (Ім'я та ПРІЗВИЩЕ)
___ _____20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 113 «Прикладна математика», освітньо-професійної програми «Наука про дані та моделювання складних систем» на тему:

«Моделювання виявлення діабету за медичними характеристиками людини методами машинного навчання»

Здобувача групи ПМ–01 Полуянова Андрія Олександровича

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

_____ Андрій ПОЛУЯНОВ

(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник: док. фіз.-мат. наук, професор Олександр ЛИСЕНКО _____

(підпис)

Суми – 2024

АНОТАЦІЯ

Кваліфікаційна робота: 57 с., 28 рисунків, 25 джерел.

Мета роботи: Аналіз медичних даних пацієнтів з метою виявлення діабету з використанням методів машинного навчання. Порівняння різних методів машинного навчання та визначення найбільш ефективних.

Об’єкт дослідження: Медичні дані ста тисяч пацієнтів, які складаються з 9 атрибутів: стать, вік, наявність гіпертонії, хвороб серця, історія куріння, рівень гемоглобіну, цукру у крові та результат – чи хвора людина.

Предмет дослідження: Точність, повнота та інші характеристики методів машинного навчання для передбачення наявності діабету.

Методи дослідження: методи машинного навчання, а саме: k-найближчих сусідів (kNN), наївний баєсів класифікатор (Naïve Bayes), дерево рішень (decision tree), випадковий ліс (Random forest), логістична регресія (Logistic regression), метод опорних векторів (SVM - support vector machine).

З метою виявлення діабету за медичними даними пацієнтів було створено комп'ютерні моделі на базі методів машинного навчання, проведено їх налаштування, визначена їх ефективність. Комп'ютерні моделі були створені на основі наступних алгоритмів: k-найближчих сусідів (kNN), наївний баєсів класифікатор (Naïve Bayes), дерево рішень (decision tree), випадковий ліс (Random forest), логістична регресія (Logistic regression), метод опорних векторів (SVM - support vector machine). Для кожного алгоритму проведено налаштування гіперпараметрів за допомогою методу сіток. З'ясовано, що найкращим алгоритмом є випадковий ліс (Random forest) з точністю в 93% та повнотою 68%. Продемонстровано високу якість створених моделей, їх придатність до практичного використання задачах виявлення діабету.

Ключові слова: КЛАСИФІКАЦІЯ ДАНИХ, АНАЛІЗ ДАНИХ, МАШИННЕ НАВЧАННЯ, ДІАБЕТ.

ЗМІСТ

Розділ 1. Вступ

Діабет - це хронічне захворювання, яке виникає або коли підшлункова залоза не виробляє достатньо інсуліну, або коли організм не може ефективно використовувати інсулін, який вона виробляє. Інсулін - це гормон, який регулює рівень глюкози в крові. Гіперглікемія, яку також називають підвищеним вмістом глюкози в крові або підвищеним рівнем цукру в крові, є поширеним наслідком неконтрольованого діабету і з часом призводить до серйозних пошкоджень багатьох систем організму, як нервів і кровоносних судин. [1]

Діабет зараз є однією з найпоширених хвороб у світі, що вражає понад 540 мільйонів дорослих осіб (20 – 79 років) у світі. Тобто кожна десята доросла людина у світі хвора на діабет. Цифра хворих буде збільшуватися та досягне 643 мільйонів осіб на 2030 рік та 783 мільйони на 2045 за прогнозами вчених.[2]

Розрізняють декілька типів діабету:

Діабет 1-го типу характеризується недостатнім виробленням інсуліну і вимагає щоденного введення інсуліну.

Діабет 2-го типу впливає на те, як ваш організм використовує цукор (глюкозу) для отримання енергії. Він заважає організму використовувати інсулін належним чином, що може призвести до високого рівня цукру в крові, якщо його не лікувати.

Симптоми діабету можуть виникнути раптово. При діабеті 2 типу симптоми можуть бути слабко вираженими, і може пройти багато років, перш ніж їх помітять. Симптоми діабету включають:

- відчуття сильної спраги
- частіші, ніж зазвичай, позиви до сечовипускання
- нечіткість зору
- відчуття втоми

- мимовільна втрата ваги

З часом діабет може пошкодити кровоносні судини серця, очей, нирок та нервів. Люди з діабетом мають вищий ризик виникнення проблем зі здоров'ям, включаючи серцевий напад, інсульт та ниркову недостатність.

Діабет може спричинити незворотну втрату зору через пошкодження кровоносних судин в очах.

У багатьох людей з діабетом виникають проблеми зі стопами через пошкодження нервів і поганий кровотік. Це може спричинити виразки на ногах і призвести до ампутації.[1]

Метою цього дослідження є розробка та впровадження моделей машинного навчання для виявлення діабету на основі показників тіла. Для досягнення цієї мети необхідно вирішити наступні завдання:

1. Зібрати та підготувати дані, необхідні для моделювання.
2. Провести аналіз та попередню обробку даних.
3. Вибрати відповідні алгоритми машинного навчання.
4. Навчити та оцінити моделі.
5. Зробити висновки щодо ефективності запропонованих моделей.

У цьому дослідженні будуть використані різні алгоритми машинного навчання, такі як k-найближчих сусідів (kNN), наївний баєсів класифікатор (Naïve Bayes), дерево рішень (decision tree), випадковий ліс (Random forest), логістична регресія (Logistic regression), метод опорних векторів (SVM - support vector machine). Вибір цих методів обґрунтований їхньою популярністю та ефективністю в попередніх дослідженнях.

Розділ 2.Розділ 1
Методи класифікації даних (огляд літератури)

1.1 Медичні дані пацієнтів для визначення діабету

Ми маємо набір даних для прогнозування діабету, що включає колекцію медичних і демографічних даних ста тисяч пацієнтів, а також їхнього діабетичного статусу (позитивного чи негативного).[3] Дані включають такі дев’ять характеристик, як:

- Стать (gender)
- Вік (age)
- гіпертонія (hypertension) (високий кров'яний тиск) - це коли тиск у ваших кровоносних судинах занадто високий (140/90 мм рт. ст. або вище). [4]
- хвороби серця (heart_disease)
- історія куріння (smoking_history)
- індекс маси тіла ІМТ (bmi) - це вага людини в кілограмах, поділена на квадрат зросту в метрах. Високий ІМТ може свідчити про високий рівень ожиріння. ІМТ виявляє вагові категорії, які можуть призвести до проблем зі здоров'ям, але не діагностує ожиріння або стан здоров'я людини. [5]
- глікований гемоглобін (HbA1c_level) - утворюється, коли глюкоза (цукор) у організмі прилипає до еритроцитів. Ваш організм не може використовувати цукор належним чином, тому більша його частина прилипає до еритроцитів і накопичується в крові. Еритроцити активні приблизно 2-3 місяці, тому показники вимірюються щоквартально. [6]
- рівень глюкози у крові (blood_glucose_level)
- наявність діабету

1.2 Методи класифікації даних, їх переваги і недоліки

У сучасних дослідженнях, спрямованих на виявлення діабету, застосування методів машинного навчання є надзвичайно важливим для досягнення високої точності прогнозування. Проте, ефективність моделей значною мірою залежить від якості підготовки даних та правильного вибору алгоритмів машинного навчання. У цьому розділі буде представлено огляд основних методів підготовки даних, таких як очищення, нормалізація та обробка пропущених значень, а також різних алгоритмів машинного навчання, включаючи логістичну регресію, дерева рішень, випадковий ліс та нейронні мережі. Метою цього огляду є надання глибокого розуміння підходів, що використовуються для підготовки даних та побудови моделей, а також обґрунтування вибору конкретних методів для даного дослідження.

1.2.1 Метод k-найближчих сусідів (kNN)

KNN - один з найпростіших, але важливих алгоритмів класифікації в машинному навчанні. Він належить до області керованого навчання і знаходить інтенсивне застосування в розпізнаванні образів, інтелектуальному аналізі даних і виявленні вторгнень.

Він широко використовується в реальних сценаріях, оскільки є непараметричним, тобто не робить жодних базових припущень щодо розподілу даних, на відміну від інших алгоритмів. Нам надаються деякі попередні дані (навчальні дані), які класифікують координати на групи, ідентифіковані за певною ознакою.[7]

Як приклад, розглянемо дані з двома ознаками, що розбиті на дві категорії:

Рис. 2.1 Візуалізація роботи методу k-найближчих сусідів

З рисунку 1.1 ми можемо побачити що тестова точка(New Data Point) має три сусіди категорії A (Category A) та два сусіди категорії B (Category B). Отже вона відповідає категорії A за методом k-найближчих сусідів.

Працює метод машинного навчання за таким алгоритмом:

1. Вибір числа сусідів. З нашого прикладу маємо число сусідів дорівнює п’яти.
2. Знаходимо евклідову відстань від точки до всіх членів набору даних.
3. Беремо K-найближчих сусідів по евклідовій відстані. З прикладу бачимо що вони обведені.
4. Серед цих найближчих сусідів рахуємо кількість точок кожної категорії.
5. Відносимо нові дані до тієї категорії, для якої кількість сусідів є максимальною.

Нехай $\boldsymbol{X}$ - навчальний набір даних з $\boldsymbol{n}$ точками, де кожна точка представлена d-вимірним вектором ознак $\boldsymbol{X_i}$, а $\boldsymbol{Y}$ - відповідні мітки або значення для кожної точки даних у $\boldsymbol{X}$. Отримавши нову точку даних $\boldsymbol{x}$, алгоритм обчислює відстань між $\boldsymbol{x}$ та кожною точкою даних $\boldsymbol{X_i}$ у $\boldsymbol{X}$, використовуючи метрику відстані:

Евклідова відстань – це декартова відстань між двома точками, які знаходяться в площині/гіперплощині. Евклідову відстань можна також візуалізувати як довжину прямої лінії, яка з'єднує дві точки, що розглядаються. Ця метрика допомагає нам обчислити чисте зміщення між двома станами об'єкта.

$$Euclidean\_distance(x, X_i) = \sqrt{\sum_{j=1}^{d} \left(x_j - X_{ij}\right)^2} \quad (2.1)$$

Метрика Манхеттенської відстані зазвичай використовується, коли нас цікавить загальна відстань, пройдена об'єктом, а не його переміщення. Ця метрика обчислюється шляхом підсумовування абсолютної різниці між координатами точок у n-вимірах.

$$Manhattan\_distance(x, y) = \sum_{i=1}^{n} |x_i - y_i| \quad (2.2)$$

Відстань Мінковського - це метрика в нормованому векторному просторі. Відстань Мінковського використовується для визначення відстані між векторами. За заданими двома або більше векторами знайти відстань між цими векторами.

В основному, відстань Мінковського застосовується в машинному навчанні для знаходження відстані між векторами.

$$Minkowski\_distance(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^p\right)^{\frac{1}{p}} \quad (2.3)$$

З наведеної вище формули можна сказати, що коли p = 2, то вона збігається з формулою для евклідової відстані, а коли p = 1, то ми отримуємо формулу для манхеттенської відстані.[7]

Алгоритм вибирає $\boldsymbol{K}$ точок даних з $\boldsymbol{X}$, які мають найкоротшу відстань до $\boldsymbol{x}$. Для задач класифікації алгоритм присвоює мітку $\boldsymbol{y}$, яка найчастіше зустрічається серед $\boldsymbol{K}$ найближчих сусідів $\boldsymbol{x}$. Для задач регресії алгоритм

обчислює середнє або середньозважене значення $\boldsymbol{y}$ з $\boldsymbol{K}$ найближчих сусідів і присвоює його як прогнозоване значення для $\boldsymbol{x}$.[7]

Плюси:

- Алгоритм K-NN дуже простий для розуміння і настільки ж простий для реалізації. Для класифікації нової точки даних алгоритм K-NN зчитує весь набір даних, щоб знайти K найближчих сусідів.
- K-NN є непараметричним алгоритмом, що означає, що для його реалізації не потрібно робити жодних припущень про розподіл даних. На відміну від параметричних моделей, таких як лінійна регресія, які мають багато припущень щодо даних, K-NN не вимагає дотримання цих умов.
- K-NN не будує явної моделі; замість цього він просто класифікує нові дані на основі історичних даних. Новий запис буде позначений класом, який є найпоширенішим серед його найближчих сусідів.
- Оскільки K-NN використовує навчання на основі екземплярів, він є підходом, заснованим на пам'яті. Класифікатор миттєво адаптується при отриманні нових навчальних даних. Це дозволяє алгоритму швидко реагувати на зміни у вхідних даних під час використання в режимі реального часу.
- Більшість алгоритмів класифікатора легко реалізувати для бінарних задач і досить складно для багатокласових задач, тоді як K-NN адаптується до багатокласових задач без будь-яких додаткових зусиль.
- Однією з найбільших переваг K-NN є те, що K-NN можна використовувати як для класифікації, так і для регресії.
- K-NN вимагає налаштування одного гіперпараметра. Хоча вибір цього параметра може зайняти деякий час, решта параметрів автоматично підлаштовуються відповідно до нього.

- Різноманітність критеріїв відстані на вибір: алгоритм K-NN надає користувачеві гнучкість у виборі відстані при побудові K-NN моделі.
  1. Евклідова відстань
  2. Манхеттенська відстань
  3. Відстань Мінковського

Мінуси:

- K-NN може бути дуже простим у реалізації, але зі збільшенням набору даних ефективність або швидкість алгоритму дуже швидко падає.
- KNN добре працює з невеликою кількістю вхідних змінних, але зі збільшенням кількості змінних алгоритм K-NN намагається передбачити вихід нової точки даних.
- K-NN потребує однорідних ознак, тобто якщо ви вирішили побудувати k-NN, використовуючи загальну відстань, наприклад, евклідову або манхеттенську, абсолютно необхідно, щоб ознаки мали однаковий масштаб, оскільки абсолютні відмінності в ознаках важать однаково, тобто задана відстань для ознаки 1 повинна означати те ж саме для ознаки 2.
- Однією з найбільших проблем K-NN є вибір оптимальної кількості сусідів, яку слід враховувати під час класифікації нових даних.
- k-NN погано працює на незбалансованих даних. Якщо ми розглядаємо два класи, A і B, і більшість навчальних даних позначено як A, то модель в кінцевому підсумку надасть перевагу A. Це може призвести до того, що менш поширений клас B буде неправильно класифіковано.
- Алгоритм K-NN дуже чутливий до викидів, оскільки він просто обирає сусідів на основі критерію відстані.
- K-NN за своєю суттю не має можливості вирішувати проблему пропущених значень.[8]

1.2.2 Наївний баєсівський класифікатор (Naïve bayes)

Наївні байєсівські класифікатори - це набір алгоритмів класифікації, заснованих на теоремі Байєса. Це не один алгоритм, а сімейство алгоритмів, де всі вони мають спільний принцип, тобто кожна пара ознак, що класифікуються, є незалежною одна від одної.

Один з найпростіших і найефективніших алгоритмів класифікації, класифікатор Naïve Bayes, допомагає швидко розробляти моделі машинного навчання з можливостями швидкого прогнозування.

Ця модель прогнозує ймовірність того, що екземпляр належить до класу із заданим набором значень ознак. Це імовірнісний класифікатор. Це тому, що він припускає, що одна ознака в моделі не залежить від існування іншої ознаки. Іншими словами, кожна ознака робить свій внесок у передбачення, не маючи жодного зв'язку між собою. У реальному світі ця умова виконується рідко. В алгоритмі навчання та прогнозування використовується теорема Байєса

Фундаментальне припущення наївного Байєса полягає в тому, що кожна ознака вносить свій внесок:

- Незалежність ознак: Ознаки є умовно незалежними одна від одної з урахуванням мітки класу.
- Неперервні ознаки: Якщо ознака неперервна, вона вважається нормально розподіленою в межах кожного класу.
- Дискретні ознаки: Якщо ознака дискретна, вона вважається мультиноміально розподіленою в межах кожного класу.

- Однакова важливість ознак: Вважається, що всі ознаки однаково важливі для прогнозування мітки класу.
- Відсутність пропущених даних: Дані не повинні містити пропущених значень.

Теорема Байєса знаходить ймовірність появи події, якщо відома ймовірність іншої події, яка вже відбулася. Математично теорема Байєса записується у вигляді наступного рівняння:

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)} \tag{2.4}$$

де $\boldsymbol{y}$ і $\boldsymbol{x}$ - події, а $\boldsymbol{P(x) \neq 0}$

По суті, ми намагаємося знайти ймовірність події $\boldsymbol{y}$ за умови, що подія $\boldsymbol{x}$ є істинною. Подія $\boldsymbol{x}$ також називається доказом.

$\boldsymbol{P(y)}$ - це апріорна ймовірність $\boldsymbol{y}$ (попередня ймовірність, тобто ймовірність події до того, як з'явився доказ). Доказ - це значення атрибуту невідомого екземпляра (тут це подія $\boldsymbol{x}$).

$\boldsymbol{P(x)}$ - це гранична ймовірність: Ймовірність доказу.

$\boldsymbol{P(y|x)}$ - апостеріорна ймовірність $\boldsymbol{x}$, тобто ймовірність події після того, як з'явиться доказ.

$\boldsymbol{P(x|y)}$ - ймовірність правдоподібності, тобто ймовірність того, що гіпотеза справдиться на основі доказів. [9]

Стосовно нашого набору даних, ми можемо застосувати теорему Байєса наступним чином:

$$P(y|X) = \frac{P(y)P(X|y)}{P(X)} \tag{2.5}$$

де, $\boldsymbol{y}$ - змінна класу, а $\boldsymbol{X}$ - залежний вектор ознак (розміру n), $X = (x_1, x_2, \cdots, x_n)$.

Тепер застосуємо наївне припущення до теореми Байєса, а саме: незалежність між ознаками. Отже, ми розбиваємо дані на незалежні частини.

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1|y)P(x_2|y) \dots P(x_n|y)P(y)}{P(x_1)P(x_2) \dots P(x_n)} \tag{2.6}$$

Яке може бути виражене як:

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2) \dots P(x_n)} \tag{2.7}$$

Тепер, оскільки знаменник залишається постійним для заданих вхідних даних, ми можемо вилучити його:

$$P(y|x_1, x_2, \dots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y) \tag{2.8}$$

Тепер нам потрібно створити модель класифікатора. Для цього ми обчислюємо ймовірність заданого набору входів для всіх можливих значень змінної класу $\boldsymbol{y}$ і вибираємо вихід з максимальною ймовірністю. Математично це можна виразити так:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y) \tag{2.9}$$

Отже, нарешті, нам залишається завдання обчислити $\boldsymbol{P(y)}$ та $\boldsymbol{P(x_i \mid y)}$. Зверніть увагу, що $\boldsymbol{P(y)}$ також називають ймовірністю класу, а $\boldsymbol{P(x_i \mid y)}$ називається умовною ймовірністю.

Різні наївні байєсівські класифікатори відрізняються головним чином припущеннями, які вони роблять щодо розподілу $\boldsymbol{P(x_i \mid y)}$.

У теорії, якщо всі ознаки в $\boldsymbol{x}$ дійсно умовно незалежні, наївний баєсівський класифікатор має спрацювати дуже добре.

Переваги наївного Баєса

Наївний баєсівський класифікатор є популярним алгоритмом завдяки наступним перевагам:

- Цей алгоритм працює дуже швидко і може легко передбачити клас тестового набору даних.
- Цей метод можна використовувати для вирішення багатокласових задач прогнозування, оскільки він є досить ефективним для них.
- Наївний баєсівський класифікатор працює краще, ніж інші моделі з меншою кількістю навчальних даних, якщо виконується припущення про незалежність ознак.
- Якщо у вас є категоріальні вхідні змінні, наївний алгоритм Байєса показує винятково хороші результати порівняно з числовими змінними.
- Його можна використовувати як для бінарних, так і для багатокласових класифікацій.

Недоліки наївного Байєса

- Якщо у вашому тестовому наборі даних є категоріальна змінна, якої не було в навчальному наборі даних, наївна модель Байєса присвоїть їй нульову ймовірність і не зможе зробити жодних прогнозів щодо цього. Це явище називається "нульова частота", і вам доведеться використовувати техніку згладжування, щоб вирішити цю проблему.
- Цей алгоритм також відомий як неточний оцінювач, тому не слід надто серйозно сприймати ймовірнісні результати прогнозування.
- Він припускає, що всі ознаки є незалежними. Хоча в теорії це звучить чудово, в реальному житті ви навряд чи знайдете набір незалежних ознак. [10]

1.2.3 Метод опорних векторів (SVM)

Метод опорних векторів (SVM) - це потужний алгоритм машинного навчання, який використовується для задач класифікації та регресії. Він особливо добре підходить для задач класифікації, де метою є розділення точок даних на різні категорії або класи. Метод опорних векторів працює шляхом пошуку оптимальної гіперплощини, яка найкраще розділяє дані на окремі класи, максимізуючи маржу (відстань) між гіперплощиною та найближчими точками даних кожного класу.

Кілька ключових моментів, які потрібно розуміти про метод опорних векторів:

Гіперплощина(Hyperplane): У задачі бінарної класифікації (поділ даних на два класи) гіперплощина є межею рішення. Це лінія у двовимірному, площина у тривимірному або гіперплощина у вищих вимірах, яка розділяє точки даних на різні класи. Метод опорних векторів прагне знайти гіперплощину, яка максимізує маржу - відстань між гіперплощиною і найближчими точками даних кожного класу.

Опорні вектори(Support Vectors): Це точки даних, які знаходяться найближче до гіперплощини і мають найменший запас. Опорні вектори мають вирішальне значення для визначення положення та орієнтації гіперплощини.

Відступ(Margin): Відступ - це відстань між гіперплощиною і найближчими точками даних кожного класу. Метод опорних векторів прагне максимізувати цю відстань, оскільки більша відстань зазвичай призводить до кращого узагальнення нових, ще не бачених даних.

Ядра(Kernel): Метод опорних векторів може обробляти як лінійно відокремлювані, так і нелінійно відокремлювані дані. Для нелінійних даних він використовує трюк ядра для відображення даних у простір вищої

розмірності, де можливе лінійне розділення. Найпоширеніші ядра включають лінійні, поліноміальні та ядра радіально-базисних функцій (RBF).

Параметр C: Метод опорних векторів має гіперпараметр «C», який контролює компроміс між максимізацією маржі (відстань між розділяючою гіперплощиною та найближчими точками даних) та мінімізацією помилок класифікації. Менші значення C збільшують маржу, але можуть призвести до більшої кількості помилок класифікації. Вищі значення C зменшують маржу, що надає перевагу точнішій класифікації, але може призвести до вужчої маржі. [11]

Математична складова методу опорних векторів полягає в задачі бінарної класифікації з двома класами, позначеними як **+1** і **-1**. У нас є навчальний набір даних, що складається з вхідних векторів ознак $\boldsymbol{X}$ і відповідних їм міток класів $\boldsymbol{Y}$.

Рівняння для лінійної гіперплощини можна записати як:

$$w^T x + b = 0 \tag{2.10}$$

Вектор $\boldsymbol{W}$ є вектором нормалі до гіперплощини, тобто напрямком, перпендикулярним до гіперплощини. Параметр $\boldsymbol{b}$ в рівнянні являє собою зміщення або відстань гіперплощини від початку координат вздовж вектора нормалі $\boldsymbol{w}$.

Відстань між точкою даних $\boldsymbol{x_i}$ та межею рішення можна обчислити як:

$$d_i = \frac{w^T x_i + b}{||w||} \tag{2.11}$$

де $||\boldsymbol{w}||$ - евклідова норма вагового вектора $\boldsymbol{w}$. Евклідова норма нормального вектора $\boldsymbol{W}$.

Для лінійного класифікатора методу опорних векторів:

$$\hat{y} = \begin{cases} 1: w^T x + b \geq 0 \\ 0: w^T x + b < 0 \end{cases} \quad (2.12)$$

Типи методів опорних векторів

Лінійний метод опорних векторів (Linear SVM): Лінійний метод опорних векторів використовує лінійну границю рішення для розділення точок даних різних класів. Якщо дані можуть бути точно лінійно розділені, лінійний метод опорних векторів дуже добре підходять. Це означає, що одна пряма лінія у двовимірному або гіперплощина у вищих вимірах може повністю розділити точки даних на відповідні класи. Гіперплощина, яка максимізує відстань між класами, є межею рішення.

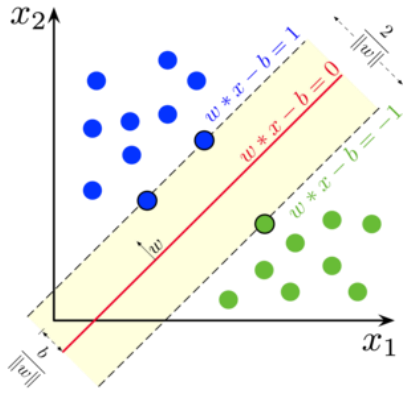

Рис. 2.2 Візуалізація роботи методу опорних векторів

Нелінійний метод опорних векторів (Non-Linear SVM): Нелінійний метод опорних векторів можна використовувати для класифікації даних, коли вони не можуть бути розділені на два класи прямою лінією у випадку двох вимірів. Використовуючи функції ядра, нелінійний метод опорних векторів може обробляти дані, що нелінійно розділяються. Вихідні вхідні дані перетворюються за допомогою цих функцій ядра у вимірний простір ознак, де точки даних можуть бути лінійно розділені. Лінійний метод опорних векторів

використовується для знаходження нелінійної межі рішення в цьому модифікованому просторі.

Ядро методу опорних векторів - це функція, яка бере низьковимірний вхідний простір і перетворює його у вимірний простір, тобто перетворює невідокремлювані задачі на відокремлювані. Здебільшого це корисно у нелінійних задачах розділення. Простіше кажучи, ядро виконує деякі надзвичайно складні перетворення даних, а потім знаходить процес розділення даних на основі визначених міток або виходів.

1. Лінійна (Linear):

$$K(\omega, b) = \omega^T x + b \tag{2.13}$$

2. Поліноміальна (Polynomial):

$$K(\omega, x) = (\gamma \omega^T x + b)^N \tag{2.14}$$

3. Гаусова радіальна базисна функція (Gaussian RBF):

$$K(\omega, x) = \exp\left(-\gamma \left|\left|x_i - x_j\right|\right|^n\right) \tag{2.15}$$

4. Сигмоїдна (Sigmoid):

$$K\left(x_i, x_j\right) = \tanh(\alpha x_i^T x_j + b) \tag{2.16}$$

[12]

Переваги методу опорних векторів:

• Метод опорних векторів працює відносно добре, коли існує чітка межа поділу між класами.

• Метод опорних векторів є більш ефективним у просторах високої розмірності.

• Метод опорних векторів ефективний у випадках, коли кількість вимірів перевищує кількість вибірок.

• Метод опорних векторів відносно ефективно використовує пам'ять.

Недоліки методу опорних векторів:

• Алгоритм метод опорних векторів не підходить для великих наборів даних.

• Метод опорних векторів не дуже добре працює, коли набір даних містить багато шуму, тобто цільові класи перекриваються.

• У випадках, коли кількість ознак для кожної точки даних перевищує кількість навчальних вибірок, метод опорних векторів працюватиме незадовільно.

• Оскільки класифікатор методу опорних векторів працює, розміщуючи точки даних вище і нижче класифікаційної гіперплощини, немає ймовірнісного пояснення для класифікації. [13]

1.2.4 Логістична регресія (Logistic regression)

Логістична регресія - це керований алгоритм машинного навчання, який виконує завдання бінарної класифікації, прогнозуючи ймовірність результату, події або спостереження. Модель дає бінарний або дихотомічний результат, обмежений двома можливими варіантами: так/ні, 0/1 або істина/хибність.

Логічна регресія аналізує зв'язок між однією або декількома незалежними змінними і класифікує дані на дискретні класи. Вона широко

використовується в прогнозному моделюванні, де модель оцінює математичну ймовірність того, чи належить екземпляр до певної категорії, чи ні.[14]

Логістична регресійна модель перетворює безперервний вихід лінійної регресійної функції у категоричний вихід за допомогою сигмоїдної функції, яка відображає будь-який реальний набір вхідних незалежних змінних у значення між 0 і 1. Розглянемо лінійну регресійну модель. [15]

Нехай є незалежні вхідні характеристики:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

і залежною змінною є Y, яка має лише бінарне значення, тобто 0 або 1.

$$Y = \begin{cases} 0\text{: Клас 1} \\ 1\text{: Клас 2} \end{cases}$$

Потім застосуємо мультилінійну функцію до вхідних змінних X.

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b \tag{2.17}$$

Тут $\boldsymbol{x_i}$ і-те спостереження $\boldsymbol{X}$, $w_i = [w_1, w_2, \cdots, w_m]$, $\boldsymbol{w}$ - це ваги або коефіцієнт, а $\boldsymbol{b}$ - зміщення (bias). просто це можна представити як точковий добуток ваги та зміщення.

$$z = w \cdot X + b \tag{2.18}$$

Тепер ми використовуємо сигмоїдну функцію(математична функція, яка використовується для відображення прогнозованих значень у ймовірності), де на вході буде $\boldsymbol{z}$, і знаходимо ймовірність між 0 і 1, тобто прогнозоване значення $\boldsymbol{y}$.

$$\sigma(z) = \frac{1}{1 - e^{-z}} \tag{2.19}$$

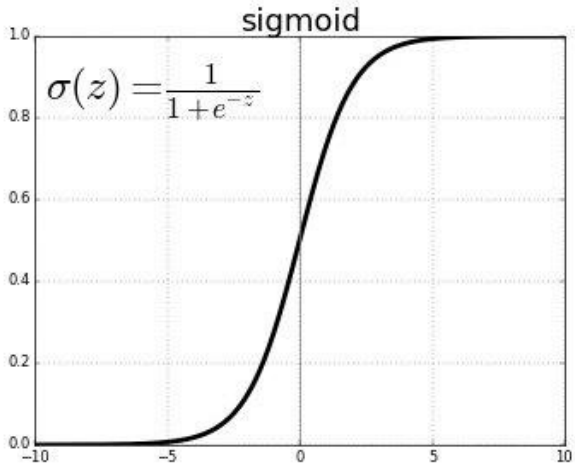

Рис. 2.3 Сігмоїдна функція

Як ми бачимо з малюнка, сигмоїдна функція перетворює дані неперервної змінної у ймовірність, тобто у значення від 0 до 1.

Маємо що ймовірність класифікатора вираховується як:

$$P(y = 1) = \sigma(z) \text{ та } P(y = 0) = 1 - \sigma(z) \tag{2.20}$$

Саме рівняння логістичної регресії буде набувати вигляду:

$$p(X; b; w) = \frac{1}{1 + e^{-w \cdot X + b}} \tag{2.21}$$

Переваги

- Навчання моделі за допомогою логістичної регресії не вимагає великих обчислювальних потужностей.
- Прогнозовані параметри (навчені ваги) дають змогу зробити висновок про важливість кожної ознаки. Також вказується напрямок зв'язку, тобто позитивний чи негативний. Таким чином, ми можемо використовувати логістичну регресію, щоб з'ясувати зв'язок між ознаками.
- Цей алгоритм дозволяє легко оновлювати моделі для врахування нових даних, на відміну від дерев рішень або методів опорних векторів. Оновлення можна здійснити за допомогою стохастичного градієнтного спуску.

- Логістична регресія виводить добре відкалібровані ймовірності разом з результатами класифікації.
- На низьковимірному наборі даних з достатньою кількістю навчальних прикладів логістична регресія менш схильна до перенавчання.

Недоліки

- На наборах даних високої розмірності це може призвести до того, що модель буде надмірно перенавчатися, а отже, модель може бути нездатною передбачити точні результати на тестовій вибірці.
- Нелінійні задачі не можуть бути вирішені за допомогою логістичної регресії, оскільки вона має лінійну поверхню рішень. Лінійно відокремлювані дані рідко зустрічаються в реальних сценаріях. Тому необхідна трансформація нелінійних ознак, яку можна зробити, збільшивши кількість ознак так, щоб дані стали лінійно відокремлюваними у вищих вимірах.
- Логістична регресія вимагає помірної або відсутньої колінеарності між незалежними змінними. Повторення інформації може призвести до неправильного навчання параметрів (ваг) під час мінімізації функції витрат.
- Для побудови моделі слід використовувати лише важливі та релевантні ознаки, інакше ймовірнісні прогнози, зроблені моделлю, можуть бути невірними, а прогностична цінність моделі може погіршитися.
- Наявність у наборі даних значень, що відхиляються від очікуваного діапазону, може призвести до неправильних результатів, оскільки цей алгоритм чутливий до викидів.
- Логістична регресія вимагає великого набору даних, а також достатньої кількості навчальних прикладів для всіх категорій, які потрібно ідентифікувати. [16]

1.2.5 Дерево рішень (Decision tree)

Дерева рішень - це непараметричний метод керованого навчання, який використовується для класифікації та регресії. Мета полягає в тому, щоб створити модель, яка передбачає значення цільової змінної шляхом вивчення простих правил прийняття рішень, виведених з особливостей даних. Дерево можна розглядати як кусково-постійну апроксимацію.[17]

Він має ієрархічну, деревоподібну структуру, яка складається з кореневого вузла (root node), гілок (branch), внутрішніх вузлів(internal node) і листових вузлів (leaf node).

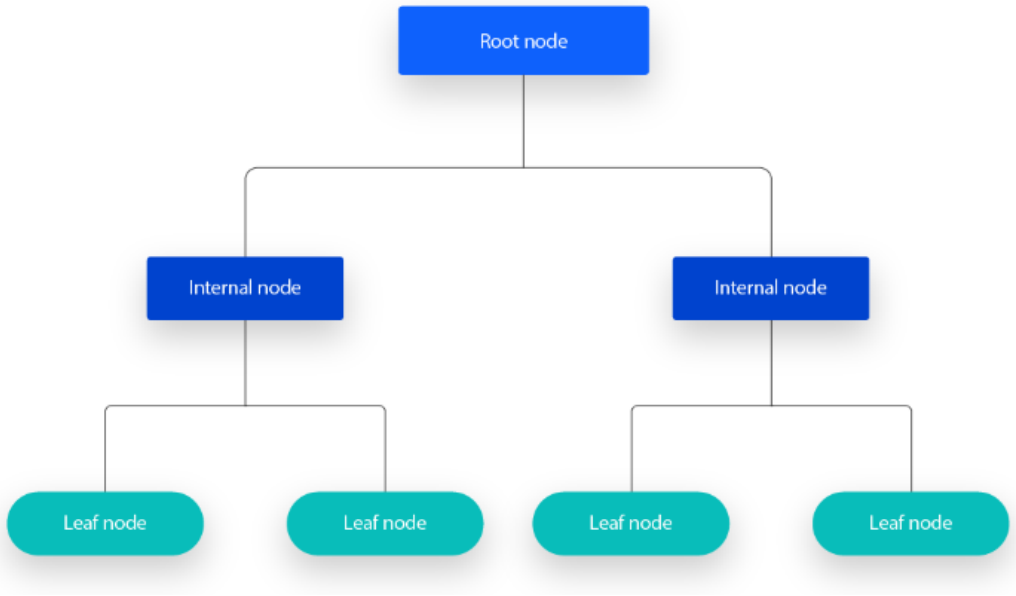

Рис. 2.4 Елементи методу машинного навчання дерева рішень

Як видно з наведеної вище схеми, дерево рішень починається з кореневого вузла, який не має жодних вхідних гілок. Вихідні гілки з кореневого вузла потім потрапляють у внутрішні вузли, також відомі як вузли прийняття рішень. На основі наявних можливостей обидва типи вузлів проводять оцінки для формування однорідних підмножин, які позначаються

листовими вузлами або термінальними вузлами. Листяні вузли представляють всі можливі результати в наборі даних.[18]

Алгоритм побудови дерева рішень включає наступні етапи:

1. Вибір найкращого атрибуту: Використовуючи метрику, таку як Gini, ентропія або інформаційний приріст, обирається найкращий атрибут для розділення даних.
2. Поділ набору даних: Набір даних розбивається на підмножини на основі обраного атрибуту.
3. Повторення процесу: Процес повторюється рекурсивно для кожної підмножини, створюючи новий внутрішній вузол або листовий вузол, поки не буде виконано критерій зупинки (наприклад, всі екземпляри у вузлі належать до одного класу або не буде досягнута задана глибина).

Подивимося на атрибути алгоритму дерева рішень:

Gini - вимірює ймовірність неправильної класифікації нового екземпляра, якщо він був випадково класифікований відповідно до розподілу класів у наборі даних.

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2 \tag{2.22}$$

Де $\boldsymbol{p}$ це ймовірність того, що екземпляр буде віднесено до певного класу.

Ентропія - вимірює кількість невизначеності або домішок у наборі даних.

$$Entropy = \sum_{i=1}^{n} p_i \log_2(p_i) \tag{2.23}$$

Приріст інформації - вимірює зменшення ентропії або домішки Джині після розділення набору даних за атрибутом.

$$Information\ Gain = Entropy_{parent} - \sum_{i=1}^{n} \left(\frac{|D_i|}{|D|} * Entropy(D_i) \right) \tag{2.24}$$

Де $\boldsymbol{D_i}$ підмножина $\boldsymbol{D}$ після розбиття за атрибутом.[19]

Переваги дерева рішень:

- Булева логіка та візуальне представлення дерев рішень роблять їх простішими для розуміння та використання.
- Порівняно з іншими алгоритмами, він потребує меншого очищення даних. Вони можуть працювати з різними типами даних - дискретними або неперервними значеннями, а неперервні значення можуть бути перетворені в категоріальні за допомогою використання порогових значень.
- Можливість перевірки моделі за допомогою статистичних тестів. Це дає можливість врахувати надійність моделі.
- Добре працює, навіть якщо її припущення дещо порушуються справжньою моделлю, з якої були отримані дані.[17][18][19]

Недоліки дерева рішень:

- Невеликі варіації в даних можуть призвести до створення дуже різних дерев рішень. Об'єднання в мішки або усереднення оцінок може бути методом зменшення дисперсії дерев рішень. Однак цей підхід є обмеженим, оскільки він може призвести до отримання висококорельованих предикторів.
- Дерева рішень можуть створювати надто складні дерева, які погано узагальнюють дані (перенавчання).
- Для більшої кількості міток класів обчислювальна складність дерева рішень може збільшитися.

1.2.6 Випадковий ліс (Random forest)

Випадковий ліс - це метод машинного навчання, який використовується для вирішення проблем регресії та класифікації. Це один з методів ансамблевого навчання - складаються з набору класифікаторів, наприклад, для випадкового лісу це метод дерев рішень. Далі їх прогнози агрегуються, щоб визначити найпопулярніший результат.[20]

Алгоритм випадкового лісу працює в декілька етапів:

Ансамбль дерев рішень: Random Forest використовує можливості ансамблевого навчання шляхом побудови армії дерев рішень. Ці дерева схожі на окремих експертів, кожен з яких спеціалізується на певному аспекті даних. Важливо, що вони працюють незалежно, мінімізуючи ризик надмірного впливу на модель нюансів одного дерева.

Випадковий вибір ознак: Щоб гарантувати, що кожне дерево рішень в ансамблі приносить унікальну перспективу, Random Forest використовує випадковий вибір ознак. Під час навчання кожного дерева вибирається випадкова підмножина ознак. Ця випадковість гарантує, що кожне дерево фокусується на різних аспектах даних, сприяючи створенню різноманітного набору предикторів в ансамблі.

Bootstrap-агрегування або об'єднання в мішки: Техніка пакування є наріжним каменем стратегії навчання Random Forest, яка передбачає створення декількох бутстрап-зразків з початкового набору даних, що дозволяє вибирати приклади із заміною. Це призводить до різних підмножин даних для кожного дерева рішень, вносячи варіативність у процес навчання і роблячи модель більш надійною.

Прийняття рішень і голосування: Коли справа доходить до прогнозування, кожне дерево рішень у випадковому лісі подає свій голос. Для задач класифікації остаточний прогноз визначається модою (найчастішим прогнозом) по всіх деревах. У задачах регресії береться середнє значення прогнозів окремих дерев. Цей внутрішній механізм голосування забезпечує

збалансований і колективний процес прийняття рішень.Особливості алгоритму випадкового лісу.[21]

Переваги випадкового лісу

- Зменшення ризику надмірної підгонки, якщо у лісі є достатня кількість дерев рішень, класифікатор не буде перенастроювати модель, оскільки усереднення некорельованих дерев знижує загальну дисперсію та похибку прогнозування.
- Пакетування ознак також робить класифікатор випадкових лісів ефективним інструментом для оцінки відсутніх значень, оскільки він зберігає точність, коли частина даних відсутня.
- Випадковий ліс дозволяє легко оцінити важливість змінної або її внесок у модель.

Недоліки випадкового лісу

- Оскільки алгоритми випадкового лісу можуть обробляти великі масиви даних, вони можуть надавати більш точні прогнози, але можуть бути повільними та вимагати більше ресурсів в обробці даних, оскільки вони обчислюють дані для кожного окремого дерева рішень.
- Прогноз одного дерева рішень легше інтерпретувати, якщо порівнювати його з лісом дерев.[20]

1.2.7 Метод головних компонент (PCA)

Метод головних компонент (PCA) - це статистична процедура, яка використовує ортогональне перетворення, що перетворює набір корельованих змінних на набір некорельованих змінних. Алгоритм навчання без нагляду,

який використовується для вивчення взаємозв'язків між набором змінних. Він також відомий як загальний факторний аналіз, де регресія визначає лінію найкращої відповідності.

Основна мета методу головних компонент (PCA) це зменшити розмірність набору даних, зберігаючи найважливіші закономірності або взаємозв'язки між змінними без будь-яких попередніх знань про цільові змінні.[22]

Алгоритм методу головних компонент:

1. По-перше, нам потрібно стандартизувати наш набір даних, щоб кожна змінна мала середнє значення 0 і стандартне відхилення 1.

$$Standardized = \frac{X - \mu}{\sigma} \tag{2.25}$$

Де $\boldsymbol{\mu}$ – середнє арифметичне значення, а $\boldsymbol{\sigma}$ - середньоквадратичне відхилення незалежних ознак.

2. Коваріація вимірює силу спільної мінливості між двома або більше змінними, вказуючи, наскільки сильно вони змінюються відносно одна одної. Може набувати позитивного, негативного або нульового значення. Щоб знайти коваріацію, ми можемо використати формулу:

$$cov(x_1; x_2) = \frac{\sum_{i=1}^{n}(x_{1i} - x_1)(x_{2i} - x_2)}{n - 1} \tag{2.26}$$

3. Обчислення власних значень та власних векторів коваріаційної матриці для визначення головних компонент

Нехай $\boldsymbol{A}$ - квадратна матриця $\boldsymbol{n} * \boldsymbol{n}$, а $\boldsymbol{X}$ - ненульовий вектор, для якого:

$$AX = \lambda X \tag{2.27}$$

для деяких скалярних значень $\boldsymbol{\lambda}$. Тоді $\boldsymbol{\lambda}$ називається власним значенням матриці $\boldsymbol{A}$, а $\boldsymbol{X}$ називається власним вектором матриці $\boldsymbol{A}$ для відповідного власного значення.

Його також можна записати як :

$$AX - \lambda X = 0$$

$$(A - \lambda I)X = 0$$

де $\boldsymbol{I}$ - матриця тотожності тієї ж форми, що і матриця $\boldsymbol{A}$. А наведені вище умови будуть справедливими лише у тому випадку, якщо $(\boldsymbol{A} - \boldsymbol{\lambda I})$ буде не оберненою (тобто сингулярною матрицею). Це означає, що

$$|A - \lambda I| = 0 \tag{2.28}$$

З наведеного вище рівняння можна знайти власні значення $\boldsymbol{\lambda}$, а отже, відповідний власний вектор можна знайти за допомогою рівняння $\boldsymbol{AX} = \boldsymbol{\lambda X}$.[22]

4. Створення вектора ознак.

Вектор ознак - це просто матриця, стовпцями якої є власні вектори компонент, які ми вирішили залишити. Це перший крок до зменшення розмірності, адже якщо ми вирішимо залишити лише $\boldsymbol{p}$ власних векторів (компонент) з $\boldsymbol{n}$, то остаточний набір даних матиме лише $\boldsymbol{p}$ вимірів.

5. Перерозподіл даних вздовж осей головних компонент.

На цьому кроці, який є останнім, метою є використання вектора ознак, сформованого за допомогою власних векторів коваріаційної матриці, для переорієнтації даних з вихідних осей на ті, що представлені головними компонентами (звідси і назва - аналіз головних компонент). Це можна зробити, помноживши транспонування вихідного набору даних на транспонування вектора ознак.

$$FinalDataSet = FeatureVector^T * StandartizedOrDataSet^T \tag{2.29}$$

1.2.8 Перехресна валідація (Cross Validation)

Перехресна валідація - це метод, який використовується в машинному навчанні для оцінки продуктивності моделі на нових даних. Вона передбачає

поділ наявних даних на кілька складових або підмножин, використання однієї з них як валідаційної вибірки і навчання моделі на решті складових. Цей процес повторюється кілька разів, щоразу використовуючи різні згортки як валідаційний набір. Нарешті, результати кожного кроку валідації усереднюються, щоб отримати більш надійну оцінку роботи моделі. Перехресна валідація є важливим кроком у процесі машинного навчання і допомагає переконатися, що обрана для розгортання модель є надійною і добре узагальнює нові дані.

Основна мета перехресної перевірки - запобігти надмірному перенавчанню, яке відбувається, коли модель занадто добре навчена на навчальних даних і погано працює на нових.[24]

1.2.9 Пошук по сітці (Grid Search)

Пошук по сітці - це техніка налаштування гіперпараметрів, яка використовується в машинному навчанні для пошуку найкращої комбінації гіперпараметрів для заданої моделі.

Пошук по сітці працює шляхом систематичного дослідження заздалегідь визначеної сітки можливих значень для кожного гіперпараметра. Він навчає та оцінює модель для кожної комбінації гіперпараметрів у сітці, зазвичай використовуючи метод перехресної перевірки для забезпечення надійності результатів. Потім порівнюються результати роботи кожної моделі, і вибирається комбінація гіперпараметрів, яка дає найкращі результати.

Розділ 3.Розділ 2
Аналіз та обробка даних

Для того щоб зробити аналіз даних за допомогою методів машинного навчання, дані треба підготувати. Тому почнемо з перегляду та обробки нашого набору даних. Для вирішення цієї задачі використовувалася мова програмування Python.

3.1. Огляд та обробка даних

Ми маємо набір даних для прогнозування діабету, що включає колекцію медичних і демографічних даних ста тисяч пацієнтів, а також їхнього діабетичного статусу (позитивного чи негативного).[3] Дані включають такі дев’ять характеристик, як:

- Стать (gender)
- Вік (age)
- гіпертонія (hypertension)(1 – так, 0 – ні)
- хвороби серця (heart_disease)(1 – так, 0 – ні)
- історія куріння (smoking_history)(нема інформації – (-1), ніколи – 0, колишній – 1, наразі – 2, не наразі – 3, колись – 4)
- індекс маси тіла ІМТ (bmi)
- глікований гемоглобін (HbA1c_level)
- рівень глюкози у крові (blood_glucose_level)
- наявність діабету(1 – так, 0 – ні)

Глянемо на наш набір даних через програмне забезпечення Python.

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
...	...	...	...	...	...	...	...	...	...
99995	Female	80.0	0	0	No Info	27.32	6.2	90	0
99996	Female	2.0	0	0	No Info	17.37	6.5	100	0
99997	Male	66.0	0	0	former	27.83	5.7	155	0
99998	Female	24.0	0	0	never	35.42	4.0	100	0
99999	Female	57.0	0	0	current	22.43	6.6	90	0

Рис. 3.1 Елементи набору даних

Маємо набір розміром сто тисяч на дев’ять ознак (100000 x 9). Маємо не цілочислові дані, які не зможуть проходити крізь наші методи машинного навчання, тому замінимо або видалимо їх.

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes	smoking_history_num
0	80.0	0	1	25.19	6.6	140	0	0
1	54.0	0	0	27.32	6.6	80	0	-1
2	28.0	0	0	27.32	5.7	158	0	0
3	36.0	0	0	23.45	5.0	155	0	2
4	76.0	1	1	20.14	4.8	155	0	2
...	...	...	...	...	...	...	...	...
99995	80.0	0	0	27.32	6.2	90	0	-1
99996	2.0	0	0	17.37	6.5	100	0	-1
99997	66.0	0	0	27.83	5.7	155	0	1
99998	24.0	0	0	35.42	4.0	100	0	0
99999	57.0	0	0	22.43	6.6	90	0	2

Рис. 3.2 Перетворені ознаки у цілочислові

Історію куріння ми замінили таким чином: нема інформації – (-1), ніколи – 0, колишній – 1, наразі – 2, не наразі – 3, колись – 4. А колонку стать(gender), я прибрав через втрачені деякі дані.

Для того щоб зрозуміти які дані в нас будуть головними у виявлені діабету, треба побудувати кореляційну матрицю. Вона покаже нам усі залежності і взаємозв’язки між нашими ознаками, наскільки вони незалежні один від одної та наскільки вони залежні від результату.

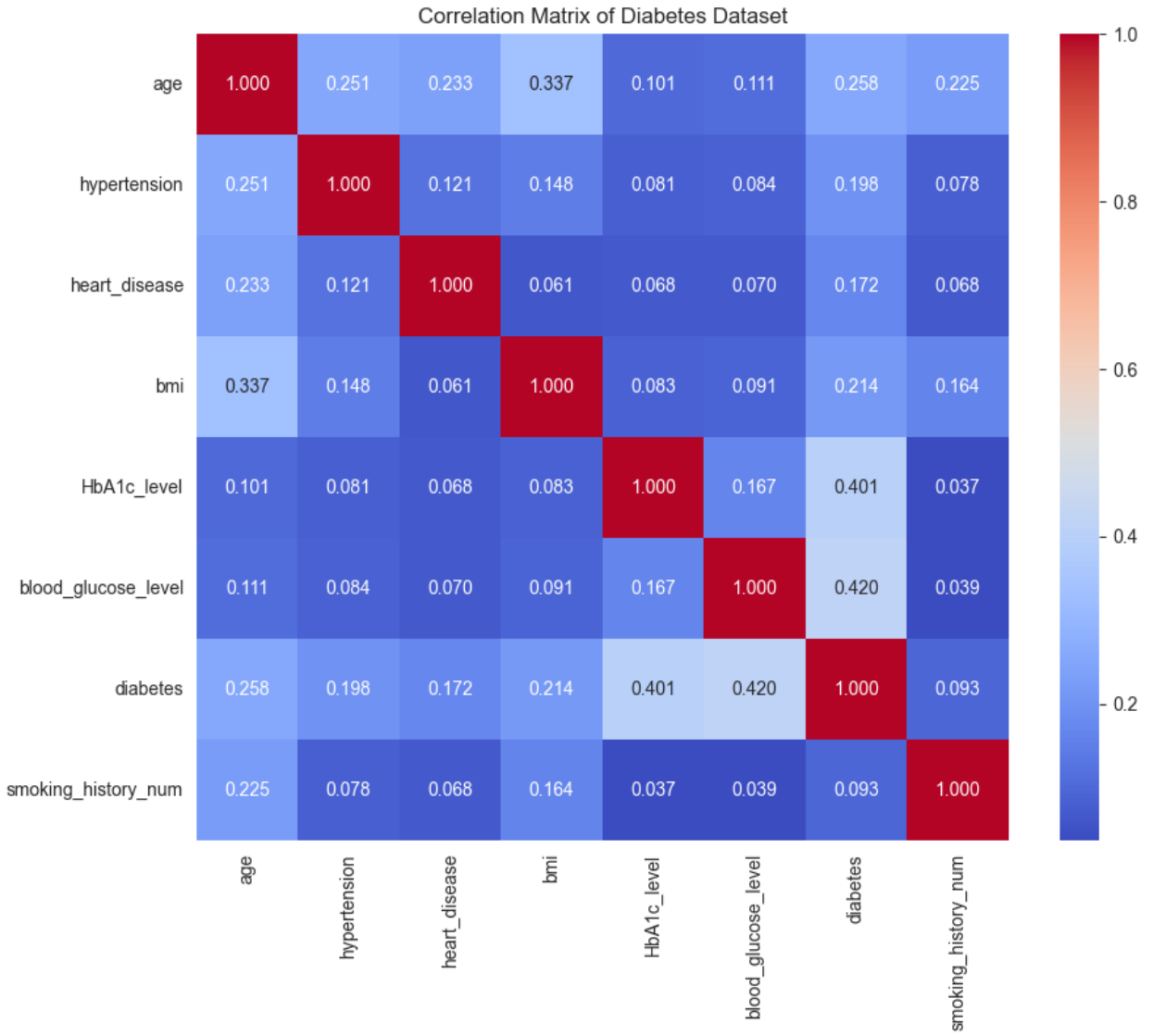

Рис. 3.3 Кореляційна матриця даного набору даних

З рисунку можемо побачити що найбільш важливі для нашого набору даних становлять показник глюкози та гемоглобіну у крові з кореляцією 0.42 та 0.4 відповідно та невисокою кореляцією між собою, що допоможе нам у подальшому для моделювання.

Далі можемо подивитися на розподіл окремих змінних та графіки залежності різних атрибутів між собою.

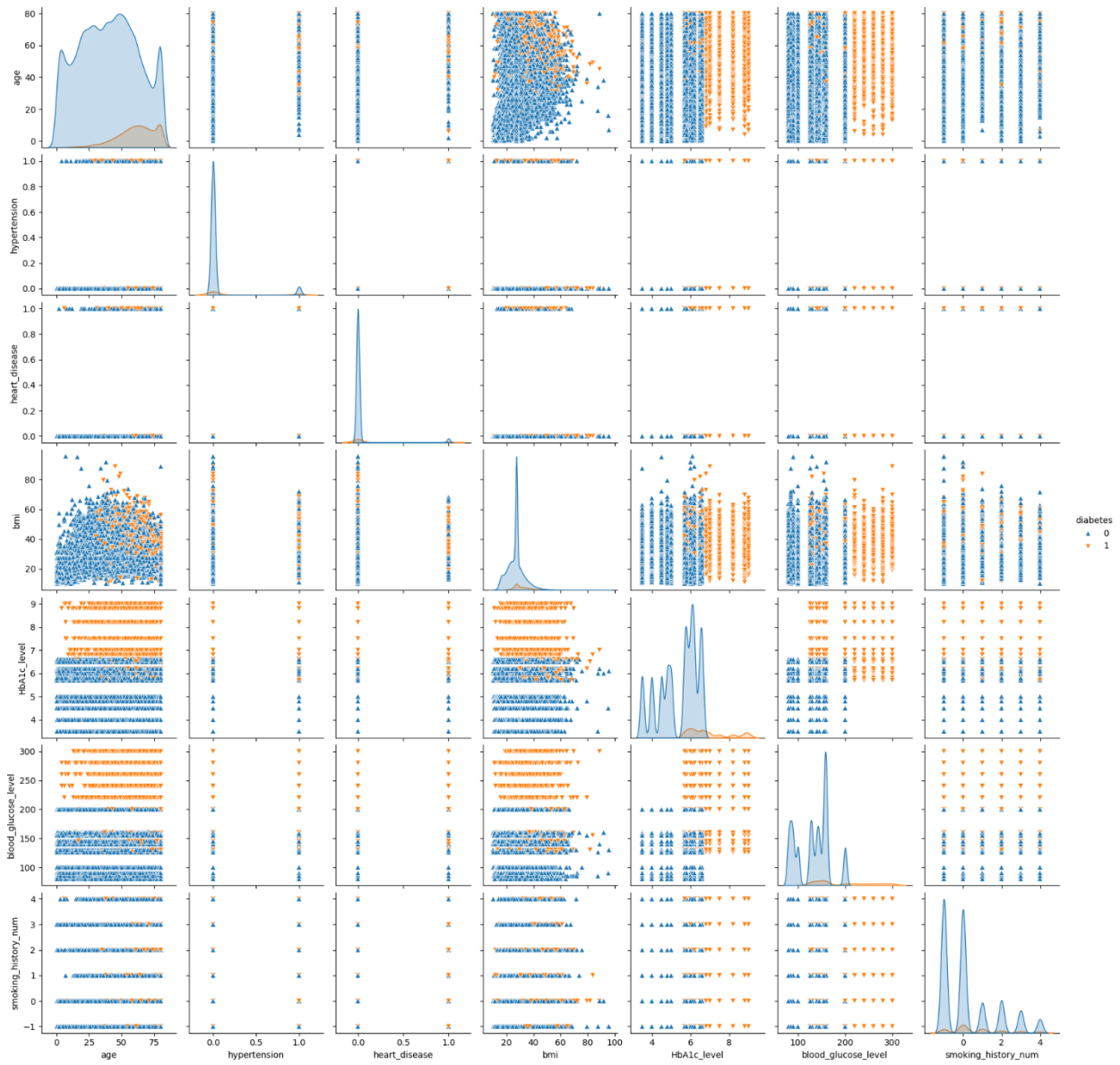

Рис. 3.4 Розподіл та графіки залежності атрибутів

Можемо подивитися на графіки, помаранчевими маркерами позначені хворі на діабет, а сині – здорові.

Для більшої наочності та того щоб зробити висновки з графіків виведемо розподіл та графіки залежності лише найбільш впливових даних з кореляцією більше за 0.2 та невисокою кореляцією між собою.

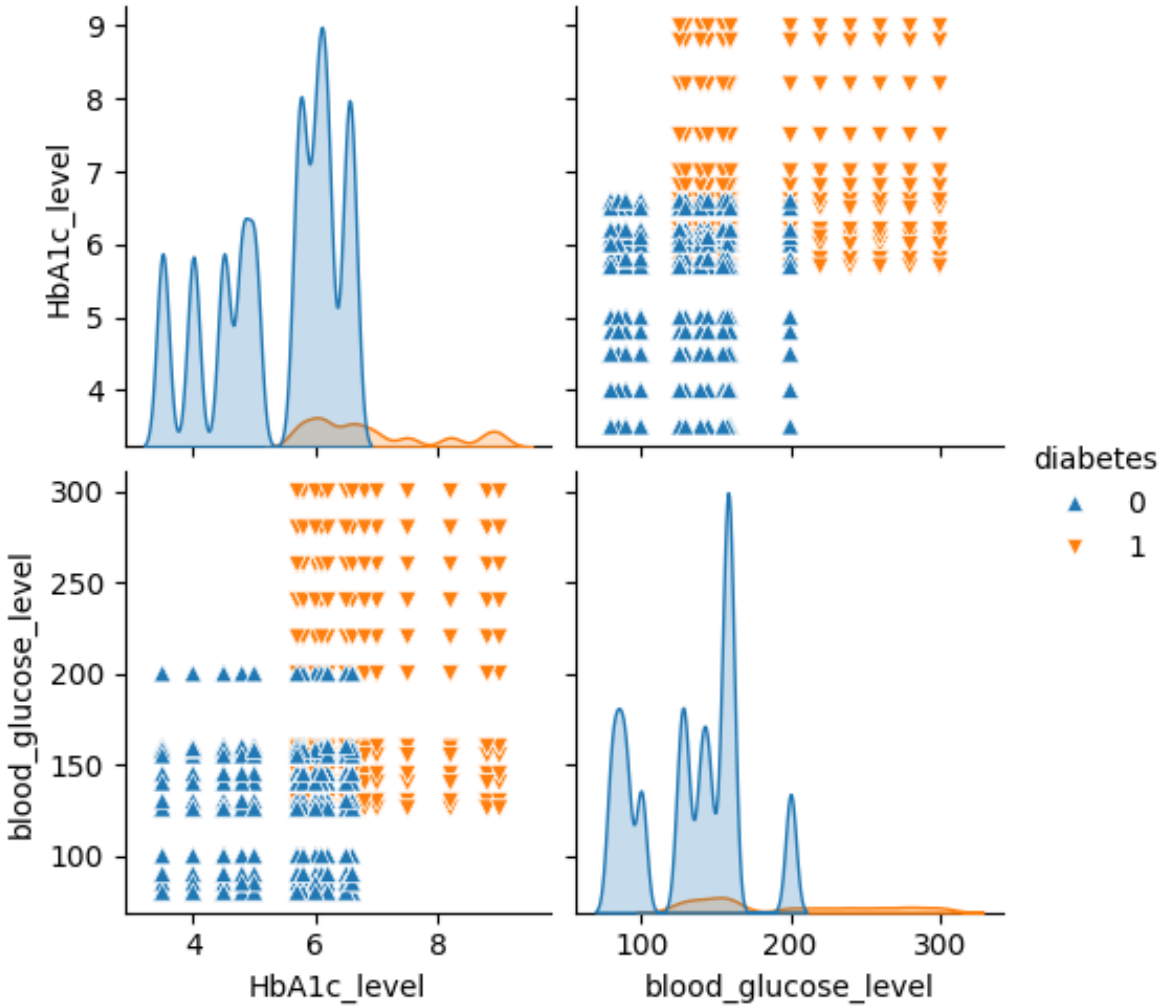

Рис. 3.5 Розподіл та графіки залежності кількості гемоглобіну та глюкози у крові

З рисунку (2.5) навіть неозброєним оком бачимо що можна класифікувати деяку частину хворих, наприклад, якщо подивимося ближче то можемо сказати що хворими будуть люди з показником глюкози у крові більше ніж 200 та показником гемоглобіну більше за 7.

Також по розподілу даних можемо помітити що пік хворих та пік здорових співпадають.

Подивимося на приклад інших даних.

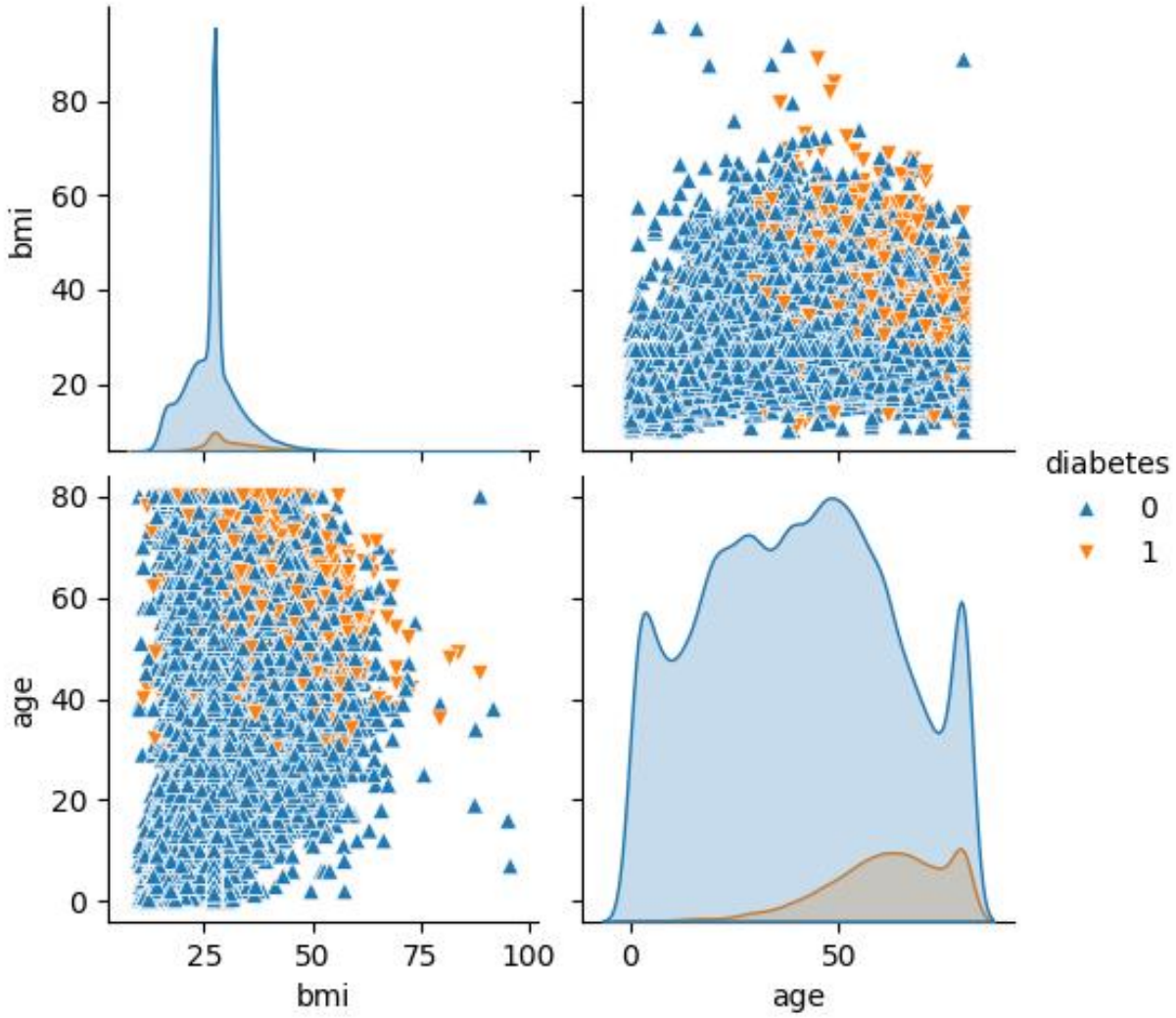

Рис. 3.6 Розподіл та графіки залежності віку та індексу маси тіла

З рисунку (2.6) ми не можемо зробити подібної класифікації як на попередньому, але можемо помітити що зі старінням людини збільшується шанс захворіти на цукровий діабет.

За розподілом можемо побачити що піки людей збігається тільки у показнику індексу маси тіла, навіть графіки досить схожі.

Зробимо стандартизацію даних. Як я писав у першому розділі, це робиться для того щоб вони мали середнє значення 0 і стандартне відхилення 1. Використовучи стандартизацію, всі ознами матимуть однаковий маштаб. Це важиливо для більшості алгоритмів машинного навчання, а особливо для методів що є чутливими до маштабування.

Майже вся підготовка даних завершилася, тепер перейдемо до навчання моделей.

3.2. Комп'ютерна модель. Методи класифікації даних.

Для початку нам треба щоб натренувати модель правильно, нам треба дізнатися гіперпараметри для кожного з методів машинного навчання, тому передемо до пошуку по сітці (Grid Search)

Найважливішою характеристикою для нашої мети є повнота, тому що нам важливо розпізнати якомога більше хворих навіть якщо ми класифікуємо у деяких пацієнтів не правильно діабет, все одно краще перевірити все точно.

```
Grid scores on development set for SVC:
Precision: 0.957 (±0.007), Recall: 0.544 (±0.015), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.581 (±0.011), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 0.001, 'classifier__kernel': 'linear'}
Precision: 0.995 (±0.004), Recall: 0.379 (±0.012), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 0.0001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.581 (±0.011), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 0.0001, 'classifier__kernel': 'linear'}
Precision: 0.953 (±0.005), Recall: 0.567 (±0.012), Params: {'classifier': SVC(), 'classifier__C': 10, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.582 (±0.010), Params: {'classifier': SVC(), 'classifier__C': 10, 'classifier__gamma': 0.001, 'classifier__kernel': 'linear'}
Precision: 0.952 (±0.006), Recall: 0.546 (±0.015), Params: {'classifier': SVC(), 'classifier__C': 10, 'classifier__gamma': 0.0001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.582 (±0.010), Params: {'classifier': SVC(), 'classifier__C': 10, 'classifier__gamma': 0.0001, 'classifier__kernel': 'linear'}
Precision: 0.967 (±0.004), Recall: 0.566 (±0.014), Params: {'classifier': SVC(), 'classifier__C': 100, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.582 (±0.010), Params: {'classifier': SVC(), 'classifier__C': 100, 'classifier__gamma': 0.001, 'classifier__kernel': 'linear'}
Precision: 0.930 (±0.003), Recall: 0.575 (±0.012), Params: {'classifier': SVC(), 'classifier__C': 100, 'classifier__gamma': 0.0001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.582 (±0.010), Params: {'classifier': SVC(), 'classifier__C': 100, 'classifier__gamma': 0.0001, 'classifier__kernel': 'linear'}
```

Рис. 3.7 Перший пошук по сітці для методу опорних векторів

```
Grid scores on development set for SVC:
Precision: 0.976 (±0.005), Recall: 0.585 (±0.015), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.581 (±0.011), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 'scale', 'classifier__kernel': 'linear'}
Precision: 0.957 (±0.007), Recall: 0.544 (±0.015), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.581 (±0.011), Params: {'classifier': SVC(), 'classifier__C': 1, 'classifier__gamma': 0.001, 'classifier__kernel': 'linear'}
Precision: 0.977 (±0.006), Recall: 0.609 (±0.010), Params: {'classifier': SVC(), 'classifier__C': 5, 'classifier__gamma': 'scale', 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.581 (±0.011), Params: {'classifier': SVC(), 'classifier__C': 5, 'classifier__gamma': 'scale', 'classifier__kernel': 'linear'}
Precision: 0.947 (±0.004), Recall: 0.566 (±0.013), Params: {'classifier': SVC(), 'classifier__C': 5, 'classifier__gamma': 0.001, 'classifier__kernel': 'rbf'}
Precision: 0.923 (±0.004), Recall: 0.581 (±0.011), Params: {'classifier': SVC(), 'classifier__C': 5, 'classifier__gamma': 0.001, 'classifier__kernel': 'linear'}
```

Рис. 3.8 Другий пошук по сітці для методу опорних векторів

```
Grid scores on development set for KNN:
Precision: 0.834 (±0.014), Recall: 0.635 (±0.010), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean', 'classifier__n_neighbors': 3, 'classifier__weights': 'uniform'}
Precision: 0.802 (±0.019), Recall: 0.644 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean', 'classifier__n_neighbors': 3, 'classifier__weights': 'distance'}
Precision: 0.884 (±0.014), Recall: 0.624 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean', 'classifier__n_neighbors': 5, 'classifier__weights': 'uniform'}
Precision: 0.853 (±0.016), Recall: 0.631 (±0.013), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean', 'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}
Precision: 0.912 (±0.010), Recall: 0.612 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean', 'classifier__n_neighbors': 7, 'classifier__weights': 'uniform'}
Precision: 0.884 (±0.012), Recall: 0.623 (±0.013), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'euclidean', 'classifier__n_neighbors': 7, 'classifier__weights': 'distance'}
Precision: 0.835 (±0.012), Recall: 0.635 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan', 'classifier__n_neighbors': 3, 'classifier__weights': 'uniform'}
Precision: 0.799 (±0.021), Recall: 0.645 (±0.014), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan', 'classifier__n_neighbors': 3, 'classifier__weights': 'distance'}
Precision: 0.893 (±0.016), Recall: 0.622 (±0.013), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan', 'classifier__n_neighbors': 5, 'classifier__weights': 'uniform'}
Precision: 0.854 (±0.020), Recall: 0.630 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan', 'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}
Precision: 0.921 (±0.012), Recall: 0.608 (±0.009), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan', 'classifier__n_neighbors': 7, 'classifier__weights': 'uniform'}
Precision: 0.885 (±0.016), Recall: 0.618 (±0.010), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'manhattan', 'classifier__n_neighbors': 7, 'classifier__weights': 'distance'}
```

Рис. 3.9 Перший пошук по сітці для методу k-найближчих сусідів

```
Grid scores on development set for KNN:
Precision: 0.884 (±0.014), Recall: 0.624 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'minkowski', 'classifier__n_neighbors': 5, 'classifier__weights': 'uniform'}
Precision: 0.853 (±0.016), Recall: 0.631 (±0.013), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'minkowski', 'classifier__n_neighbors': 5, 'classifier__weights': 'distance'}
Precision: 0.912 (±0.010), Recall: 0.612 (±0.012), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'minkowski', 'classifier__n_neighbors': 7, 'classifier__weights': 'uniform'}
Precision: 0.884 (±0.012), Recall: 0.623 (±0.013), Params: {'classifier': KNeighborsClassifier(), 'classifier__metric': 'minkowski', 'classifier__n_neighbors': 7, 'classifier__weights': 'distance'}
```

Рис. 3.10 Другий пошук по сітці для методу k-найближчих сусідів

```
Grid scores on development set for NaiveBayes:
Precision: 0.450 (±0.010), Recall: 0.647 (±0.025), Params: {'classifier': GaussianNB(), 'classifier__var_smoothing': 1e-09}
Precision: 0.450 (±0.010), Recall: 0.647 (±0.025), Params: {'classifier': GaussianNB(), 'classifier__var_smoothing': 1e-08}
Precision: 0.450 (±0.010), Recall: 0.647 (±0.025), Params: {'classifier': GaussianNB(), 'classifier__var_smoothing': 1e-07}
Precision: 0.450 (±0.010), Recall: 0.647 (±0.025), Params: {'classifier': GaussianNB(), 'classifier__var_smoothing': 1e-06}
Precision: 0.450 (±0.010), Recall: 0.647 (±0.025), Params: {'classifier': GaussianNB(), 'classifier__var_smoothing': 1e-05}
Precision: 0.450 (±0.010), Recall: 0.647 (±0.025), Params: {'classifier': GaussianNB(), 'classifier__var_smoothing': 0.0001}
```

Рис. 3.11 Пошук по сітці для методу баєсівського класифікатора

```
Grid scores on development set for RandomForest:
Precision: 0.921 (±0.008), Recall: 0.688 (±0.013), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 25}
Precision: 0.932 (±0.012), Recall: 0.684 (±0.012), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 75}
Precision: 0.937 (±0.009), Recall: 0.684 (±0.012), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 150}
Precision: 0.998 (±0.001), Recall: 0.667 (±0.010), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 10, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 25}
Precision: 0.997 (±0.001), Recall: 0.667 (±0.009), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 10, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 75}
Precision: 0.997 (±0.001), Recall: 0.667 (±0.010), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 10, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 150}
Precision: 0.947 (±0.011), Recall: 0.685 (±0.011), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 20, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 25}
Precision: 0.958 (±0.008), Recall: 0.682 (±0.010), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 20, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 75}
Precision: 0.961 (±0.008), Recall: 0.682 (±0.010), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 20, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 150}
```

Рис. 3.12 Перший пошук по сітці для методу випадковий ліс

```
Grid scores on development set for RandomForest:
Precision: 0.937 (±0.011), Recall: 0.682 (±0.012), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 100}
Precision: 0.932 (±0.012), Recall: 0.687 (±0.011), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 75}
Precision: 0.929 (±0.011), Recall: 0.684 (±0.009), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 30}
Precision: 0.998 (±0.001), Recall: 0.666 (±0.009), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 10, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 100}
Precision: 0.997 (±0.002), Recall: 0.666 (±0.010), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 10, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 75}
Precision: 0.997 (±0.002), Recall: 0.666 (±0.009), Params: {'classifier': RandomForestClassifier(), 'classifier__max_depth': 10, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 30}
```

Рис. 3.13 Другий пошук по сітці для методу випадковий ліс

```
Grid scores on development set for LogisticRegression:
Precision: 0.870 (±0.005), Recall: 0.621 (±0.012), Params: {'classifier': LogisticRegression(), 'classifier__C': 0.1, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}
Precision: 0.870 (±0.005), Recall: 0.621 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 0.1, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
Precision: 0.873 (±0.006), Recall: 0.619 (±0.012), Params: {'classifier': LogisticRegression(), 'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'liblinear'}
Precision: 0.870 (±0.005), Recall: 0.621 (±0.012), Params: {'classifier': LogisticRegression(), 'classifier__C': 0.1, 'classifier__penalty': 'l2', 'classifier__solver': 'saga'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 1, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 1, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 1, 'classifier__penalty': 'l2', 'classifier__solver': 'liblinear'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 1, 'classifier__penalty': 'l2', 'classifier__solver': 'saga'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 10, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 10, 'classifier__penalty': 'l1', 'classifier__solver': 'saga'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 10, 'classifier__penalty': 'l2', 'classifier__solver': 'liblinear'}
Precision: 0.868 (±0.005), Recall: 0.622 (±0.011), Params: {'classifier': LogisticRegression(), 'classifier__C': 10, 'classifier__penalty': 'l2', 'classifier__solver': 'saga'}
```

Рис. 3.14 Пошук по сітці для методу логістична регресія

```
Grid scores on development set for DecisionTree:
Precision: 0.701 (±0.007), Recall: 0.733 (±0.015), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': None, 'classifier__min_samples_split': 2}
Precision: 0.748 (±0.012), Recall: 0.726 (±0.013), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': None, 'classifier__min_samples_split': 5}
Precision: 0.799 (±0.014), Recall: 0.720 (±0.010), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': None, 'classifier__min_samples_split': 10}
Precision: 0.970 (±0.005), Recall: 0.678 (±0.009), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': 10, 'classifier__min_samples_split': 2}
Precision: 0.970 (±0.005), Recall: 0.678 (±0.009), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': 10, 'classifier__min_samples_split': 5}
Precision: 0.970 (±0.005), Recall: 0.678 (±0.009), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': 10, 'classifier__min_samples_split': 10}
Precision: 0.766 (±0.008), Recall: 0.722 (±0.012), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': 20, 'classifier__min_samples_split': 2}
Precision: 0.795 (±0.016), Recall: 0.718 (±0.012), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': 20, 'classifier__min_samples_split': 5}
Precision: 0.827 (±0.014), Recall: 0.714 (±0.008), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'gini', 'classifier__max_depth': 20, 'classifier__min_samples_split': 10}
Precision: 0.709 (±0.010), Recall: 0.731 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': None, 'classifier__min_samples_split': 2}
Precision: 0.743 (±0.012), Recall: 0.724 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': None, 'classifier__min_samples_split': 5}
Precision: 0.780 (±0.009), Recall: 0.720 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': None, 'classifier__min_samples_split': 10}
Precision: 0.967 (±0.012), Recall: 0.679 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': 10, 'classifier__min_samples_split': 2}
Precision: 0.967 (±0.014), Recall: 0.679 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': 10, 'classifier__min_samples_split': 5}
Precision: 0.966 (±0.012), Recall: 0.679 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': 10, 'classifier__min_samples_split': 10}
Precision: 0.790 (±0.019), Recall: 0.715 (±0.012), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': 20, 'classifier__min_samples_split': 2}
Precision: 0.809 (±0.019), Recall: 0.713 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': 20, 'classifier__min_samples_split': 5}
Precision: 0.828 (±0.026), Recall: 0.710 (±0.011), Params: {'classifier': DecisionTreeClassifier(), 'classifier__criterion': 'entropy', 'classifier__max_depth': 20, 'classifier__min_samples_split': 10}
```

Рис. 3.15 Пошук по сітці для методу дерево рішень

Я відмітив найкращі варіанти для всіх моделей, що я знайшов. Можемо подивитися але я не відмічав майже завжди максимальне значення повноти, тому що в деяких випадках від цього дуже падає точність. Тобто наша модель перенавчається і ми отримуємо кращий результат по повноті наших даних. Наприклад, ми це дуже гарно можемо побачити на рисунку (2.15), перша відмітка: точність – 70, а повнота 73 відсотки. Інший – точність – 83, а повнота 71. Бачимо що всього два відсотки повноти, але точність підвищилася на 13 відсотків, дуже гарний приклад перенавчання моделі.

Тепер ми перейдемо до методу головних компонент, перевіримо залежність точності та повноти моделей від кількості атрибутів.

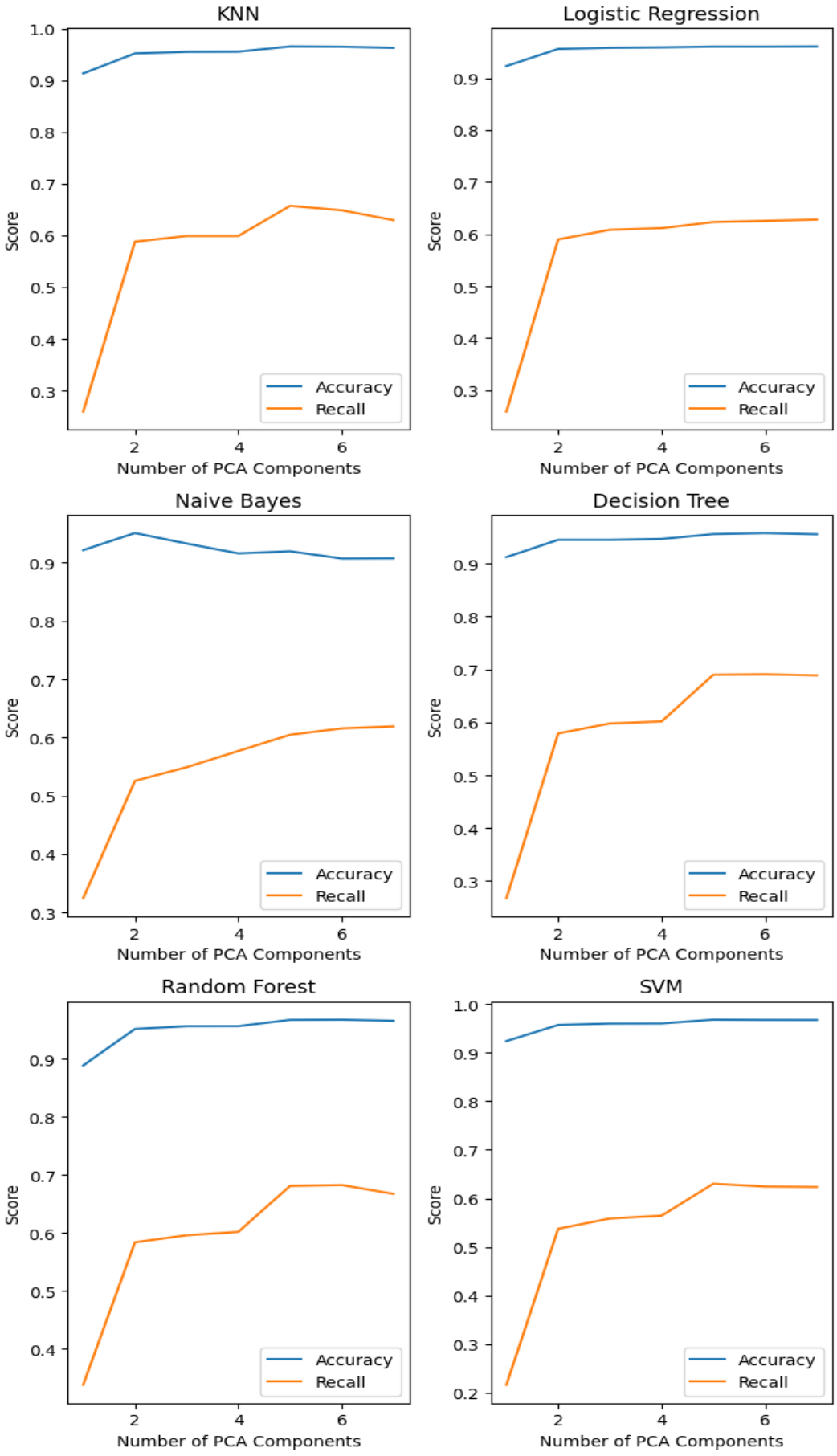

Рис. 3.16 Залежність повноти та точності від кількості головних компонент

Як ми бачимо з графіків, для більшості методів машинного навчання пік повноти та точності становить на п’ятому головному компоненті. Окрім наївного баєсівського класифікатора, у нього вона росте зі збільшенням кількості компонент, а точність навпаки падає. Пік точності для наївного баєсівського класифікатора залишився на другому головному компоненті.

Тепер перейдемо до тренування моделей з найкращими для них умовами та порівняємо їх.

KNN Accuracy:0.9655

	precision	recall	f1-score	support
0	0.97	0.99	0.98	27447
1	0.91	0.66	0.76	2553
accuracy			0.97	30000
macro avg	0.94	0.83	0.87	30000
weighted avg	0.96	0.97	0.96	30000

Рис. 3.17 Метрики методу k-найближчих сусідів

Logistic Regression Accuracy:0.9604

	precision	recall	f1-score	support
0	0.97	0.99	0.98	27447
1	0.88	0.62	0.73	2553
accuracy			0.96	30000
macro avg	0.92	0.81	0.85	30000
weighted avg	0.96	0.96	0.96	30000

Рис. 3.18 Метрики методу логістичної регресії

Naive Bayes Accuracy:0.9191

	precision	recall	f1-score	support
0	0.96	0.95	0.96	27447
1	0.52	0.61	0.56	2553
accuracy			0.92	30000
macro avg	0.74	0.78	0.76	30000
weighted avg	0.93	0.92	0.92	30000

Рис. 3.19 Метрики методу наївного баєсівського класифікатора

```
Decision Tree Accuracy:0.9546
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     27447
           1       0.76      0.69      0.72      2553

    accuracy                           0.95     30000
   macro avg       0.86      0.83      0.85     30000
weighted avg       0.95      0.95      0.95     30000
```

Рис. 3.20 Метрики методу дерева рішень

```
Random Forest Accuracy:0.9686
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     27447
           1       0.93      0.68      0.79      2553

    accuracy                           0.97     30000
   macro avg       0.95      0.84      0.89     30000
weighted avg       0.97      0.97      0.97     30000
```

Рис. 3.21 Метрики методу випадкового лісу

```
SVM Accuracy:0.9679
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     27447
           1       0.99      0.63      0.77      2553

    accuracy                           0.97     30000
   macro avg       0.98      0.81      0.88     30000
weighted avg       0.97      0.97      0.96     30000
```

Рис. 3.22 Метрики методу опорних векторів

Для оцінки якості навченої моделі необхідно порівняти результати перехресної валідації та результати, отримані на тестових даних.

```
KNN Cross-validated Accuracy: 0.9651 (+/- 0.0017)
Logistic Regression Cross-validated Accuracy: 0.9601 (+/- 0.0025)
Naive Bayes Cross-validated Accuracy: 0.9182 (+/- 0.0028)
Decision Tree Cross-validated Accuracy: 0.9570 (+/- 0.0038)
Random Forest Cross-validated Accuracy: 0.9668 (+/- 0.0029)
SVM Cross-validated Accuracy: 0.9677 (+/- 0.0017)
```

Рис. 3.23 Перехресна валідація для точності методів класифікації

```
KNN Cross-validated Recall: 0.6548 (+/- 0.0211)
Logistic Regression Cross-validated Recall: 0.6202 (+/- 0.0346)
Naive Bayes Cross-validated Recall: 0.5952 (+/- 0.0326)
Decision Tree Cross-validated Recall: 0.6764 (+/- 0.0377)
Random Forest Cross-validated Recall: 0.6720 (+/- 0.0183)
SVM Cross-validated Recall: 0.6284 (+/- 0.0212)
```

Рис. 3.24 Перехресна валідація для повноти методів класифікації

Ми маємо, що середні значення метрик якості моделі на обох наборах даних (перехресної валідації та тестових) є подібними, а це свідчить про те, що модель була навчена належним чином.

Нам найважливішою метрикою в нашому аналізі є метрика повноти(recall) і метод дерево рішень має найбільший показник 69%, але точність дуже низька 76%, якщо порівнювати з іншими методами. Тому найкращим методом для цього аналізу вийшов випадковий ліс, з повнотою 68% та точністю 93%, що на 1% менше за повнотою ніж у дерева рішень і на 17% більше точності.

Розділ 4. Висновки

У поданій роботі було створено комп'ютерні моделі на базі методів машинного навчання з метою виявлення діабету за медичними даними пацієнтів, проведено їх налаштування, визначена їх ефективність. Комп'ютерні моделі були створені на основі наступних алгоритмів: k-найближчих сусідів (kNN), наївний баєсів класифікатор (Naïve Bayes), дерево рішень (decision tree), випадковий ліс (Random forest), логістична регресія (Logistic regression), метод опорних векторів (SVM - support vector machine). Для кожного алгоритму проведено налаштування гіперпараметрів за допомогою методу сіток.

З'ясовано, що найкращим алгоритмом є випадковий ліс (Random forest) з точністю в 93% та повнотою 68%. Такі методи як k-найближчих сусідів, метод опорних векторів та логістична регресія також показали досить високу точність 91%, 99% та 88% та повноту 66%, 63%, 62% відповідно. Продемонстровано високу якість створених моделей, їх придатність до практичного використання задачах виявлення діабету.

Розділ 5.Список використаних джерел

1. Diabetes. World Health Organization (WHO). URL: https://www.who.int/news-room/fact-sheets/detail/diabetes (date of access: 29.05.2024).
2. IDF Diabetes Atlas. IDF Diabetes Atlas. URL: https://diabetesatlas.org/ (date of access: 29.05.2024).
3. MUSTAFA M. Diabetes prediction dataset. Kaggle: Your Machine Learning and Data Science Community. URL: https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset (date of access: 29.05.2024).
4. Hypertension. World Health Organization (WHO). URL: https://www.who.int/news-room/fact-sheets/detail/hypertension (date of access: 29.05.2024).
5. Body Mass Index (BMI). Centers for Disease Control and Prevention. URL: https://www.cdc.gov/healthyweight/assessing/bmi/index.html (date of access: 29.05.2024).
6. Emma W. What is HbA1c?. Diabetes UK. URL: https://www.diabetes.org.uk/guide-to-diabetes/managing-your-diabetes/hba1c (date of access: 29.05.2024).
7. K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/k-nearest-neighbours/ (date of access: 29.05.2024).
8. Pros and Cons of K-Nearest Neighbors - From The GENESIS. From The GENESIS. URL: https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/ (date of access: 05.05.2024).
9. Naive Bayes Classifiers - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/naive-bayes-classifiers/ (date of access: 31.05.2024).

10. Naive Bayes Classifier: Pros & Cons, Applications & Types Explained | upGrad blog. upGrad blog. URL: https://www.upgrad.com/blog/naive-bayes-classifier/ (date of access: 05.05.2024).
11. Tales T. &. What is SVM and How Does It Work?. Medium. URL: https://techntales.medium.com/what-is-svm-and-how-does-it-work-b20d612af9e2 (date of access: 31.05.2024).
12. Support Vector Machine (SVM) Algorithm - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (date of access: 05.05.2024).
13. K D. Top 4 advantages and disadvantages of Support Vector Machine or SVM. Medium. URL: https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107 (date of access: 05.05.2024).
14. Kanade V. Business and Industry News, Analysis and Expert Insights - Spiceworks. URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/ (date of access: 02.06.2024).
15. Logistic Regression in Machine Learning - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/understanding-logistic-regression/ (date of access: 05.05.2024).
16. Grover K. Advantages and Disadvantages of Logistic Regression. OpenGenus IQ: Computing Expertise & Legacy. URL: https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/ (date of access: 05.05.2024).
17. Decision Trees. scikit-learn. URL: https://scikit-learn.org/stable/modules/tree.html (date of access: 03.06.2024).
18. What is a Decision Tree? | IBM. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/topics/decision-trees (date of access: 03.06.2024).
19. Decision Tree - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/decision-tree/ (date of access: 05.05.2024).

20. What Is Random Forest? | IBM. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/topics/random-forest (date of access: 03.06.2024).
21. Random Forest Algorithm in Machine Learning - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/ (date of access: 03.06.2024).
22. Principal Component Analysis(PCA) - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/principal-component-analysis-pca/ (date of access: 03.06.2024).
23. A Step-by-Step Explanation of Principal Component Analysis (PCA). Built In. URL: https://builtin.com/data-science/step-step-explanation-principal-component-analysis (date of access: 05.05.2024).
24. Cross Validation in Machine Learning - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/cross-validation-machine-learning/ (date of access: 04.06.2024).
25. What is Grid Search? | Grid Search Defined | Dremio. Dremio Unified Analytics Platform for a Self-Service Lakehouse. URL: https://www.dremio.com/wiki/grid-search/ (date of access: 04.06.2024).

Розділ 6.Додаток А

Коди комп'ютерної моделі

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline

# Завантаження та підготовка даних
diabetes_data = pd.read_csv('diabetes_prediction_dataset.csv')
data = diabetes_data

# Перетворення текстових значень "smoking_history" на числові
```

```
data['smoking_history_num'] = data['smoking_history'].map({'No Info': -1, 'never': 0, 'former': 1, 'current': 2, 'not current': 3, 'ever': 4})

# Видалення колонок "smoking_history" та "gender"
data = data.drop('smoking_history', axis=1)
data = data.drop('gender', axis=1)
data

# Побудова матриці кореляцій
correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".3f")
plt.title("Correlation Matrix of Diabetes Dataset")
plt.show()

# Побудова парних графіків (pairplot)
sns.pairplot(data, hue='diabetes', markers=["^", "v"])
plt.show()

sns.pairplot(data[['bmi', 'age', 'diabetes']], hue='diabetes', markers=["^", "v"])
sns.pairplot(data[['HbA1c_level', 'blood_glucose_level', 'diabetes']], hue='diabetes', markers=["^", "v"])
plt.show()

# Перемішування даних
shuffled_data = data.sample(frac=1.0, random_state=42)

# Визначення X та y
X = shuffled_data.drop('diabetes', axis=1)
y = shuffled_data['diabetes']
```

```
# Розподіл на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Масштабування даних
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

# Функція для виведення результатів GridSearchCV
def print_dataframe(filtered_cv_results):
    """Прекрасний друк для відфільтрованого датафрейму"""
    for mean_precision, std_precision, mean_recall, std_recall, params in zip(
        filtered_cv_results["mean_test_precision"],
        filtered_cv_results["std_test_precision"],
        filtered_cv_results["mean_test_recall"],
        filtered_cv_results["std_test_recall"],
        filtered_cv_results["params"],
    ):
        print(
            f"Precision: {mean_precision:0.3f} (±{std_precision:0.03f}),"
            f" Recall: {mean_recall:0.3f} (±{std_recall:0.03f}),"
            f" Params: {params}"
        )
    print()

# Функція для вибору найкращого оцінювача на основі стратегії підбору
def refit_strategy(cv_results):
```

```
    """Вибір найкращого оцінювача на основі recall як основного критерію."""
    precision_threshold = 0.95
    cv_results_ = pd.DataFrame(cv_results)
    high_precision_cv_results = cv_results_[cv_results_["mean_test_precision"] > precision_threshold]

    if high_precision_cv_results.empty:
        print("No models met the precision threshold.")
        return cv_results_["mean_test_recall"].idxmax()

    best_recall = high_precision_cv_results["mean_test_recall"].max()
    best_recall_std = high_precision_cv_results["mean_test_recall"].std()
    best_recall_threshold = best_recall - best_recall_std
    high_recall_cv_results = high_precision_cv_results[
        high_precision_cv_results["mean_test_recall"] > best_recall_threshold
    ]

    if (high_recall_cv_results.empty):
        print("No models met the recall threshold within the high precision models.")
        return high_precision_cv_results["mean_test_recall"].idxmax()

    fastest_top_recall_high_precision_index = high_recall_cv_results["mean_score_time"].idxmin()

    return fastest_top_recall_high_precision_index

# Налаштування параметрів для різних класифікаторів
scores = ["precision", "recall"]
```

```
param_grids = {
    'SVC': {
        'classifier': [SVC()],
        'classifier__kernel': ['rbf', 'linear'],
        'classifier__gamma': ['scale', 1e-3, 1e-4],
        'classifier__C': [1, 5, 10, 100]
    },
    'LogisticRegression': {
        'classifier': [LogisticRegression()],
        'classifier__penalty': ['l1', 'l2'],
        'classifier__C': [0.1, 1, 10],
        'classifier__solver': ['liblinear', 'saga']
    },
    'RandomForest': {
        'classifier': [RandomForestClassifier()],
        'classifier__n_estimators': [25, 75, 100, 150],
        'classifier__max_depth': [None, 10, 20],
        'classifier__min_samples_split': [2]
    },
    'KNN': {
        'classifier': [KNeighborsClassifier()],
        'classifier__n_neighbors': [3, 5, 7],
        'classifier__weights': ['uniform', 'distance'],
        'classifier__metric': ['euclidean', 'manhattan', 'minkowski']
    },
    'NaiveBayes': {
        'classifier': [GaussianNB()],
        'classifier__var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4]
    },
    'DecisionTree': {
```

```
            'classifier': [DecisionTreeClassifier()],
            'classifier__criterion': ['gini', 'entropy'],
            'classifier__max_depth': [None, 5, 7, 15],
            'classifier__min_samples_split': [2]
        }
    }

    # Виконання GridSearchCV для кожного класифікатора
    for clf_name, param_grid in param_grids.items():
        print(f"\nRunning GridSearchCV for {clf_name}...")
        grid_search = GridSearchCV(
            Pipeline([('scaler', StandardScaler()), ('classifier', param_grid['classifier'][0])]),
            param_grid, scoring=scores, refit=refit_strategy, cv=5
        )
        grid_search.fit(X_train_scaled, y_train)

        print(f"Best parameters set found for {clf_name}:")
        print(grid_search.best_params_)

        print(f"Grid scores on development set for {clf_name}:")
        results = pd.DataFrame(grid_search.cv_results_)
        print_dataframe(results)

    # Визначення класифікаторів
    classifiers = {
        'KNN': KNeighborsClassifier(n_neighbors=5, weights='uniform', metric='minkowski'),
        'Logistic Regression': LogisticRegression(penalty='l1', C=0.1, solver='saga'),
        'Naive Bayes': GaussianNB(var_smoothing=1e-09),
```

```
    'Decision Tree': DecisionTreeClassifier(criterion='gini', max_depth=20, min_samples_split=10),
    'Random Forest': RandomForestClassifier(n_estimators=75, criterion='gini', max_depth=None, min_samples_split=2),
    'SVM': SVC(C=5.0, kernel='rbf', gamma='scale')
}

# Максимальна кількість компонент для PCA
max_components = X_train.shape[1]

# Ініціалізація результатів
results = {name: {'accuracy': [], 'recall': []} for name in classifiers.keys()}

# Перебір кількості компонент
components_range = range(1, max_components + 1)
for n_components in components_range:
    pca = PCA(n_components=n_components)
    X_train_pca = pca.fit_transform(X_train_scaled)
    X_test_pca = pca.transform(X_test_scaled)

    for name, clf in classifiers.items():
        clf.fit(X_train_pca, y_train)
        y_pred = clf.predict(X_test_pca)

        acc = accuracy_score(y_test, y_pred)
        rec = recall_score(y_test, y_pred)

        results[name]['accuracy'].append(acc)
        results[name]['recall'].append(rec)
```

```
    # Побудова графіків
    plt.figure(figsize=(7, 14))

    for i, (name, metrics) in enumerate(results.items(), 1):
        plt.subplot(3, 2, i)
        plt.plot(components_range, metrics['accuracy'], label='Accuracy')
        plt.plot(components_range, metrics['recall'], label='Recall')
        plt.xlabel('Number of PCA Components')
        plt.ylabel('Score')
        plt.title(f'{name}')
        plt.legend()

    plt.tight_layout()
    plt.show()

    # Виконання PCA з 5 компонентами
    pca = PCA(n_components=5)
    X_train_pca = pca.fit_transform(X_train_scaled)
    X_test_pca = pca.transform(X_test_scaled)
    n_components = 5

    # Перевірка класифікаторів за допомогою крос-валідації
    for name, clf in classifiers.items():
        pipeline = make_pipeline(StandardScaler(),
PCA(n_components=n_components), clf)
        scores = cross_val_score(pipeline, X, y, cv=5)
        print(f"{name} Cross-validated Accuracy: {scores.mean():.4f} (+/-
{scores.std() * 2:.4f})")

    # Перевірка класифікаторів за допомогою крос-валідації (Recall)
```

```
    for name, clf in classifiers.items():
        pipeline = make_pipeline(StandardScaler(), PCA(n_components=n_components), clf)
        scores = cross_val_score(pipeline, X, y, cv=4, scoring='recall')
        print(f"{name} Cross-validated Recall: {scores.mean():.4f} (+/- {scores.std() * 2:.4f})")

    # Оцінка класифікаторів на тестових даних
    for name, clf in classifiers.items():
        clf.fit(X_train_pca, y_train)
        print(f"{name} Accuracy: {accuracy_score(y_test, clf.predict(X_test_pca)):.4f}\n {classification_report(y_test, clf.predict(X_test_pca))}")
```