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 хвороби Паркінсона за голосовими біомаркерами»

Здобувача групи ПМ-01 Гашкодьора Єгора Володимировича

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

_____ Єгор ГАШКОДЬОР
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

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

Суми – 2024

АНОТАЦІЯ

Кваліфікаційна робота: 38 с., 24 рисунків, 1 додаток, 5 джерел.

Мета роботи: Розробка математичної та комп'ютерних моделей для виявлення хвороби Паркінсона за голосовим біомаркером. Визначення ефективності моделей та знаходження найбільш ефективної.

Об’єкт дослідження: Набір даних, що містить показники звукових записів голосу пацієнтів з хворобою Паркінсона.

Предмет дослідження: якісні характеристики методів машинного навчання для передбачення пацієнтів з хворобою Паркінсона.

Методи дослідження: метод опорних векторів (SVM), дерево рішень (Decision tree), випадковий ліс (Random forest), логістична регресія (Logistian regression), метод найближчих сусідів (KNN), метод градієнтного бустінгу (Gradient boosting).

В роботі побудова комп’ютерна модель для визначення людей на хворобу Паркінсона на основі голосових маркерів. Використовувались такі методи машинного навчання: метод опорних векторів, дерево рішень, випадковий ліс, логістична регресія, метод пошуку найближчих сусідів, метод градієнтного бустингу. Проведена попередня обробка даних, визначення найголовніших ознак. За результатами роботи виявлено, що серед всіх обраних алгоритмів, метод градієнтного бустингу є найкращим.

Ключові слова: ГРАДІЄНТНИЙ БУСТІНГ, ДЕРЕВА РІШЕНЬ, БАЛАНСУВАННЯ ДАНИХ, ХВОРОБА ПАРКІНСОНА, ГОЛОСОВІ БІОМАРКЕРИ.

ЗМІСТ

ВСТУП

Хвороба Паркінсона (ХП) — це дегенеративний неврологічний розлад, що характеризується зниженням рівня дофаміну в мозку [2]. Це проявляється через погіршення рухів, наявність тремору тощо. Зазвичай спостерігається помітний вплив на мовлення, включаючи дизартрію (труднощі з артикуляцією звуків), гіпофонію (знижена гучність) і монотонність (зменшений діапазон висоти). Крім того, можуть виникнути когнітивні порушення та зміни настрою, а також підвищується ризик деменції.

Традиційна діагностика хвороби Паркінсона передбачає, збір неврологічної історії пацієнта та спостереження за його руховими навичками в різних ситуаціях. Оскільки не існує остаточного лабораторного тесту для діагностики ХП, діагностика часто є важкою, особливо на ранніх стадіях, коли моторні ефекти ще не виражені. Моніторинг прогресування захворювання з плином часу вимагає повторного відвідування клініки пацієнтом.

Оскільки пацієнти з ХП демонструють характерні вокальні особливості, записи голосу є корисним і неінвазивним інструментом для діагностики. Якби алгоритми машинного навчання могли бути застосовані до набору даних запису голосу для точної діагностики ХП, це було б ефективним етапом скринінгу перед прийомом у лікаря.

В цій роботі буде проведений аналіз способів попередньої обробки даних та порівняння існуючих методів машинного навчання, серед яких, буде обрано найефективніші, саме для цього набору даних.

РОЗДІЛ 1
ОГЛЯД ОСНОВНИХ ПОНЯТЬ, ОГЛЯД МЕТОДІВ КЛАСИФІКАЦІЇ

1.1 Хвороба Паркінсона

Дослідження показали, що приблизно 90% пацієнтів із ПРП демонструють певну форму порушення голосу. Порушення голосу також може бути одним із найбільш ранніх показників початку захворювання, а вимірювання голосу є неінвазивним і простим у застосуванні. Таким чином, вимірювання голосу для виявлення та відстеження прогресування симптомів хвороби Паркінсона привернуло значну увагу.

Нижче наведений огляд основних понять щодо впливу хвороби Паркінсона на голос людини з точки зору досліджень голосу та мови.[5]

- Фонетичні зміни: Люди з Паркінсоном можуть демонструвати зміну вимови звуків, довготи голосних, мелодії мови та ритму. Це може бути виявлено через аналіз звуків мовлення.
- Динаміка голосу: Хворі на Паркінсона часто виявляють зміни в інтонації та ритмі мовлення. Вимірювання таких динамічних аспектів, як флуктуація амплітуди та частоти голосу, може допомогти виявити хворобу.
- Тремтіння: Тремор може впливати на голос людини. Вимірювання тремтіння в голосі може допомогти в діагностиці та оцінці ступеня хвороби.
- Артикуляційні зміни: Люди з Паркінсоном можуть демонструвати зміни у спроможності контролювати артикуляцію. Це може проявлятися у нерівномірності та неконтрольованості артикуляційних рухів, що можна виміряти з використанням акустичних даних.

- Паузи і зупинки: Люди з Паркінсоном можуть виявляти підвищену кількість пауз та зупинок у своєму мовленні. Аналіз інтервалів між фразами та словами може допомогти виявити ці зміни.
- Респіраторні зміни: Хворі на Паркінсона можуть мати зміни в дихальній функції, що може впливати на голосовий вихід. Можна аналізувати параметри, пов'язані з респіраторною функцією, такі як частота дихання та об'єм повітря.

1.2 Логістична регресія

Методи класифікації в машинному навчанні використовуються для призначення об'єктам або прикладам вхідних даних певних міток або категорій на основі їх характеристик чи ознак [4]. Нижче наведено огляд деяких популярних методів класифікації, а також їх переваги і недоліки:

Логістична регресія є одним з основних методів класифікації в машинному навчанні. Вона використовується для прогнозування ймовірності приналежності об'єкта до двох або більше категорій.

Основна ідея логістичної регресії полягає у трансформації лінійної комбінації вхідних ознак за допомогою логістичної функції (також відомої як сигмоїда), яка знаходиться в межах від 0 до 1. Логістична функція дозволяє моделі виражати ймовірність належності об'єкта до одного з класів. Чим більше значення логістичної функції, тим більше ймовірність належності до певного класу.

Одна з найпоширеніших формул логістичної регресії для двокласової задачі має вигляд:

$$\boldsymbol{P = \frac{1}{(1 + e^{(-z)})}}$$

де P - ймовірність належності об'єкта до класу 1, x - вектор вхідних ознак, z - лінійна комбінація вхідних ознак з вагами моделі.

Під час тренування логістичної регресії використовується метод максимальної правдоподібності для оцінки оптимальних значень ваг. Це включає мінімізацію функції втрат, такої як бінарна перехресна ентропія, між прогнозованими ймовірностями та фактичними значеннями класів у навчальному наборі.

Переваги: простота використання, швидкість навчання, можливість отримати ймовірнісні прогнози.

Недоліки: неефективний при складних залежностях між ознаками, не може моделювати складні рішення.

1.3 Дерева рішень:

Дерева рішень є одним із найбільш популярних та інтерпретованих методів машинного навчання. Вони використовуються для вирішення задач класифікації та регресії. Дерево рішень можна уявити як ієрархічну структуру, що складається з різних розгалужень на основі різних ознак даних. Вони допомагають робити прогнози шляхом послідовного прийняття рішень на основі характеристик об'єкта, який потрібно класифікувати або зробити прогноз.

Процес побудови дерева рішень включає наступні кроки:

- Вибір ознаки: на першому кроці дерево вибирає найбільш важливу ознаку для поділу даних на підгрупи, за допомогою ентропії

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

де p_i - це частота (або частка) класу i у наборі даних S.

- Поділ даних: дерево розгалужується на дві або більше гілки на основі значення обраної ознаки. Кожна гілка відповідає певному значенню або діапазону значень обраної ознаки. Вузол розділяється по тій ознаці, що зменшує ентропію найбільше (або збільшує чистоту підмножини, що

еквівалентно). Оптимальний поділ обирається таким чином, щоб після розбиття ентропія у кожній гілці була якомога меншою.

- Рекурсивний процес: процес вибору ознаки і поділу даних повторюється для кожної нової гілки до тих пір, поки не буде досягнута певна умова зупинки. Таким чином, утворюється ієрархічна структура рішень.
- Умова зупинки: процес побудови дерева може бути зупинений, коли досягається певна глибина дерева, коли кількість об'єктів у вузлах стає менше певного порогу або коли не вдалося поділити даних на більше підгрупи.

Переваги: інтерпретованість, легка зрозумілість, можливість обробки як числових, так і категоріальних ознак.

Недоліки: схильність до перенавчання, нестабільність до невеликих змін у даних.

1.4 Випадковий ліс (Random forest):

Випадковий ліс (Random Forest) - це ансамбль дерев рішень, що використовується для задач класифікації, регресії та інших завдань машинного навчання. Він є одним з найпоширеніших та ефективних методів згладжування дерев для покращення точності та стабільності моделі.

Процес побудови випадкового лісу можна розділити на кілька кроків:

1. Вибір підвибірок: Спочатку випадково вибираються декілька підвибірок (заміщенням) з навчального набору даних. Ці підвибірки називаються "деревами" або "деревами рішень", і кожна з них використовується для побудови окремого дерева рішень.
2. Побудова дерева: Для кожного підвибірки даних будується дерево рішень. Процес побудови дерева може включати в себе вибір оптимальних розбиттів вузлів, використання ентропії або іншої функції для визначення якості поділу та рекурсивний поділ даних на підмножини.

3. Голосування більшості: Після побудови всіх дерев у випадковому лісі, для класифікації нових даних або прогнозування в у регресії кожне дерево "голосує" за своє рішення. Клас, що отримав найбільшу кількість голосів, вважається кінцевим прогнозом моделі.

Переваги: висока точність класифікації, здатність обробляти великі набори даних з багатьма ознаками.

Недоліки: схильність до перенавчання, можливість складності при налаштуванні гіперпараметрів.

1.5 Метод опорних векторів (SVM):

Метод опорних векторів (Support Vector Machines, SVM) є одним із найпопулярніших і потужних методів машинного навчання для задач класифікації та регресії. Він використовується для знаходження гіперплощини, що найкращим чином розділяє дані у просторі ознак, забезпечуючи максимальний зазор між класами. Таким чином, SVM спробує знайти оптимальну гіперплощину, що найкращим чином розділяє дані, незалежно від їхньої розмірності.

Основні ідеї та принципи роботи методу опорних векторів:

- Роздільна гіперплощина: Основна мета SVM - знайти гіперплощину (у двовимірному випадку - лінію, у тривимірному - площину), яка розділяє дані двох класів на площині ознак, забезпечуючи максимальний проміжок між двома класами. Цей проміжок вимірюється як відстань від найближчих об'єктів (опорних векторів) до гіперплощини.
- Опорні вектори: Опорні вектори - це об'єкти навчального набору, що знаходяться найближче до роздільної гіперплощини. Вони визначають положення гіперплощини та є важливими елементами для класифікації.
- Ядро SVM: Даний метод можна застосовувати до даних, які не розділяються лінійно у просторі ознак. Це досягається за допомогою ядерної функції, яка перетворює дані у вищу розмірність, де вони можуть

бути лінійно розділені. Деякі з популярних ядерних функцій включають лінійне, поліноміальне, радіальне базисне функції (RBF) та інші.

- Вирішення задачі оптимізації: Для знаходження оптимальної гіперплощини SVM формулює задачу оптимізації, яка полягає у мінімізації функції втрат, яка враховує величину зазору та регуляризаційні члени для запобігання перенавчанню.
- М'який зазор (Soft Margin): В реальних задачах дані часто не розділяються ідеально лінійно. Тому в SVM використовується поняття "м'якого зазору", де допускається деяке перекриття класів і допустимі помилки класифікації. Це дозволяє знайти більш реалістичний розділений гіперплощини.

Переваги: ефективність в просторах високої вимірності, підтримка використання ядерних функцій для моделювання нелінійних залежностей.

Недоліки: обмежена ефективність на великих наборах даних, складність вибору підходящого ядра, чутливість до масштабування даних.

1.6 Метод k-найближчих сусідів (KNN):

Метод k-найближчих сусідів (k-Nearest Neighbors, KNN) - це простий, але ефективний алгоритм машинного навчання, що використовується для класифікації та регресії. Основна ідея полягає у тому, щоб при класифікації нового зразка призначити йому клас, який найчастіше зустрічається серед його k найближчих сусідів у просторі ознак.

Основні ідеї та принципи роботи методу k-найближчих сусідів:

• Визначення відстані: Основною операцією KNN є визначення відстані між об'єктами у просторі ознак. Це може бути здійснено за допомогою різних метрик, таких як Евклідова відстань, Манхеттенська відстань, косинусна схожість тощо.

• Вибір параметра k: Параметр k визначає кількість найближчих сусідів, які використовуються для класифікації нового зразка. Вибір оптимального значення k може впливати на результат класифікації.

• Механізм голосування: Після визначення k найближчих сусідів, застосовується механізм голосування для визначення класу нового зразка. Наприклад, у випадку бінарної класифікації може використовуватися простий механізм більшості.

• Обробка ваг: У деяких випадках можна також використовувати ваги для найближчих сусідів, де сусіди, розташовані ближче до нового зразка, мають більший вплив на його класифікацію.

• Вирішення проблеми "вузьких місцин": KNN може мати тенденцію працювати неефективно в просторах великої кількості ознак або при наявності деяких непропорційних розподілів класів, що може призводити до "вузьких місць" у просторі ознак.

Переваги: простота реалізації, ефективність для невеликих наборів даних, здатність моделювати складні залежності без потреби у великій кількості гіперпараметрів.

Недоліки: чутливість до шуму та високих вимірів, велика обчислювальна складність при великих наборах даних, неефективність у випадку нерівномірного розподілу класів.

1.7 Метод градієнтного підсилення (Gradient Boosting):

Метод градієнтного підсилення (Gradient Boosting) - це потужний алгоритм машинного навчання для задач класифікації та регресії, який базується на ідеї побудови послідовної моделі, що коригується для покращення точності прогнозів.

Основні ідеї та принципи роботи методу градієнтного підсилення:

• Послідовне побудова: Головна ідея градієнтного підсилення полягає у побудові послідовної серії слабких базових моделей (зазвичай дерев рішень), які

корегуються таким чином, щоб зменшити помилку прогнозування попередньої моделі.

• Градієнтний спуск: Для покращення моделі кожна нова модель намагається мінімізувати функцію втрат, яка визначає різницю між прогнозованими значеннями та реальними значеннями цільової змінної. Це досягається за допомогою градієнтного спуску.

• Регуляризація: Метод градієнтного підсилення підтримує можливість використання різних методів регуляризації для запобігання перенавчанню, таких як зменшення розмірності дерев, використання штрафів на параметри моделі тощо.

• Використання ансамблю: Градієнтне підсилення використовується в якості базового алгоритму для побудови ансамблю моделей, таких як градієнтний бустинг над деревами рішень (GBDT) або градієнтний бустинг над лінійними моделями.

• Збалансованість: Метод градієнтного підсилення може бути налаштований для балансу між зменшенням помилки навчання та зменшенням ризику перенавчання за рахунок використання різних параметрів моделі та стратегій навчання.

Переваги методу градієнтного підсилення:

Переваги: висока точність прогнозування, здатність враховувати різноманітні типи даних та залежності, ефективність на великих наборах даних.

Недоліки: чутливість до великої кількості гіперпараметрів, більш висока обчислювальна складність порівняно з іншими методами, можливість перенавчання при неправильному налаштуванні параметрів.

РОЗДІЛ 2
МАТЕМАТИЧНА ТА КОМП'ЮТЕРНА МОДЕЛЬ

2.1 Опис набору даних та його атрибутів

Досліджуваний набір даних складається з ряду біомедичних вимірювань голосу 31 людини, 23 з яких мають хворобу Паркінсона (ХП). Кожен стовпець у таблиці є певним показником голосу, а кожен рядок відповідає одному із 195 записів голосу цих осіб (стовпець «ім’я»). Основна мета даних полягає в тому, щоб відрізнити здорових людей від тих, хто страждає на ХП, відповідно до стовпця «статус», у якому встановлено значення 0 для здорових і 1 для ХП.

Дані представлені у форматі ASCII CSV. Рядки файлу CSV містять екземпляр, що відповідає одному запису голосу. Існує приблизно шість записів на пацієнта, ім’я пацієнта вказано в першому стовпці.2.Очистка даних

Записи (атрибути) стовпця матриці:

Name – ім’я суб’єкта ASCII та номер запису

MDVP:Fo(Hz) – середня основна частота голосу

MDVP:Fhi(Hz) – максимальна основна частота голосу

MDVP:Flo(Hz) – мінімальна основна частота голосу

MDVP: Тремтіння (%), MDVP: тремтіння (Abs), MDVP: RAP, MDVP: PPQ, Jitter: DDP - Кілька показників зміни основної частоти

MDVP:Shimmer, MDVP:Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5,MDVP:APQ, Shimmer:DDA - Кілька показників варіації амплітуди

NHR, HNR - Два показники співвідношення шуму до тональних компонентів голосу

Status - Стан здоров'я суб'єкта (одиниця) - хвороба Паркінсона, (нуль) - здоровий

RPDE, D2 – дві нелінійні динамічні міри складності

DFA - Експонента фрактального масштабування сигналу

spread1,spread2,PPE - Три нелінійні міри варіації основної частоти

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              195 non-null    object
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  status            195 non-null    int64
 18  RPDE              195 non-null    float64
 19  DFA               195 non-null    float64
...
 22  D2                195 non-null    float64
 23  PPE               195 non-null    float64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

Рисунок 2.1 Відомості про атрибути

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...

5 rows × 24 columns

Рисунок 2.2 Зовнішній вигляд даних

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	MDVP:Shimmer(dB)	...
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	...
mean	154.228641	197.104918	116.324631	0.006220	0.000044	0.003306	0.003446	0.009920	0.029709	0.282251	...
std	41.390065	91.491548	43.521413	0.004848	0.000035	0.002968	0.002759	0.008903	0.018857	0.194877	...
min	88.333000	102.145000	65.476000	0.001680	0.000007	0.000680	0.000920	0.002040	0.009540	0.085000	...
25%	117.572000	134.862500	84.291000	0.003460	0.000020	0.001660	0.001860	0.004985	0.016505	0.148500	...
50%	148.790000	175.829000	104.315000	0.004940	0.000030	0.002500	0.002690	0.007490	0.022970	0.221000	...
75%	182.769000	224.205500	140.018500	0.007365	0.000060	0.003835	0.003955	0.011505	0.037885	0.350000	...
max	260.105000	592.030000	239.170000	0.033160	0.000260	0.021440	0.019580	0.064330	0.119080	1.302000	...

8 rows × 23 columns

Рисунок 2.3 Відомості статистичні дані

```
status
1    147
0     48
Name: count, dtype: int64
(195, 24)
```

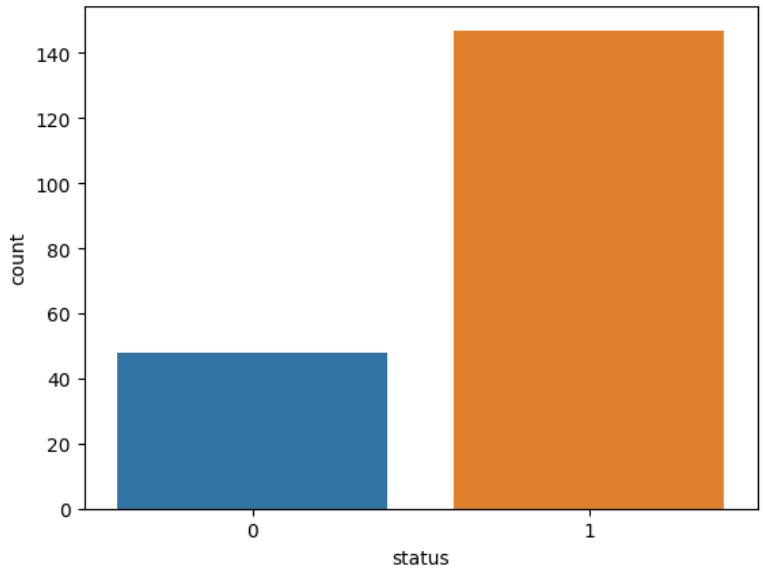

Рисунок 2.4 Кількість записів відношення кількості здорових людей до хворих, де 0 - здоровий, 1- хворий.

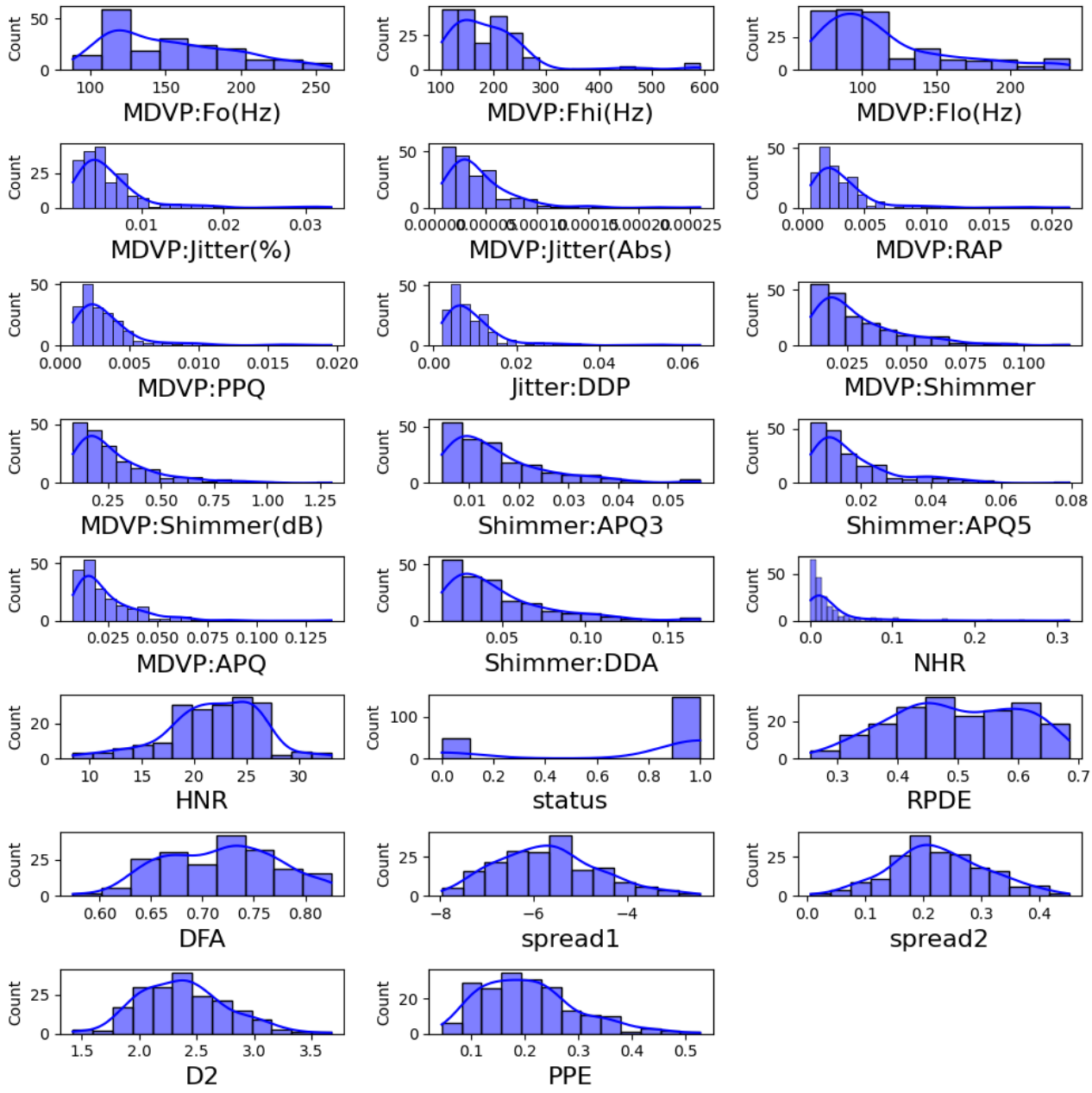

Рисунок 2.5 Розподіл атрибутів

Маємо велику кількість показників із асиметрією в лівий бік.

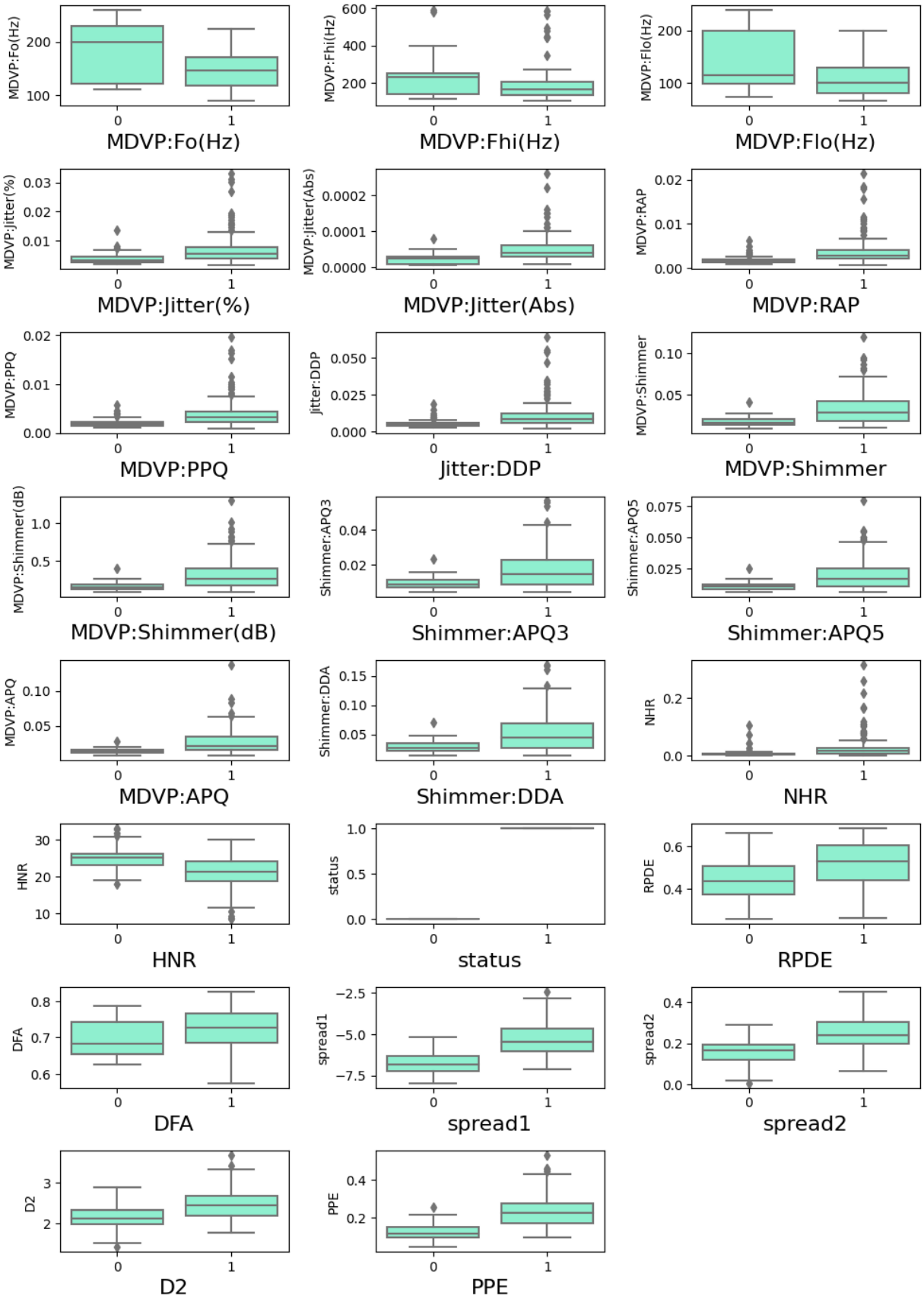

Рисунок 2.6 Коробкова діаграма по значущим атрибутам

Маємо значну кількість викидів.

З цього робимо висновок що при масштабування даних не слід використовувати стандартизацію, адже в такому випадку ми збільшимо вплив викидів та асиметрії на кінцевий результат.

Візуалізуємо кореляцію показників

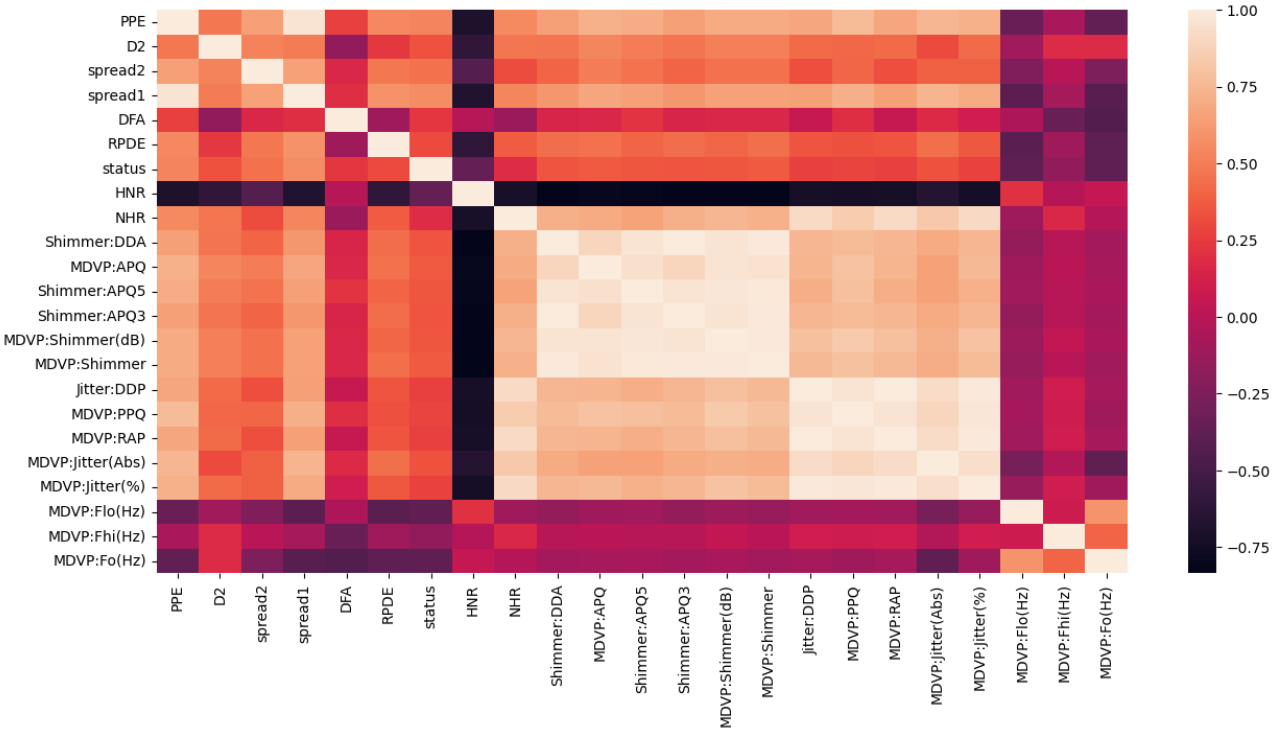

Рисунок 2.7 Кореляційна матриця атрибутів

З результатів бачимо що ознаки MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP та MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA, сильно корелюють між собою, беремо це до уваги і в подальшому впроваджуємо алгоритм відбору ознак.

2.2 Масштабування даних

Порівняння точності роботи алгоритмів у випадках стандартизації на нормалізації.

Тут ми використовуємо нормалізацію MinMax та масштабуємо дані в інтервал від 0 до 1.

	Model	Accuracy
0	Logistic Regression	0.861538
1	Decision Tree	0.907692
2	Random Forest	0.923077
3	Support Vector Machine(linear)	0.876923
4	KNN	0.923077
5	Gradient Boosting	0.969231

Рисунок 2.8 Точність роботи алгоритмів після нормалізації

	Model	Accuracy
0	Logistic Regression	0.846154
1	Decision Tree	0.907692
2	Random Forest	0.923077
3	Support Vector Machine(linear)	0.846154
4	KNN	0.907692
5	Gradient Boosting	0.969231

Рисунок 2.9 Точність роботи алгоритмів після стандартизації

Як і очікувалось, нормалізація дає кращі результати. Продемонструємо роботу кожного з методів на масштабованому наборі даних.

2.3 Демонстрація роботи алгоритмів

Логістична регресія

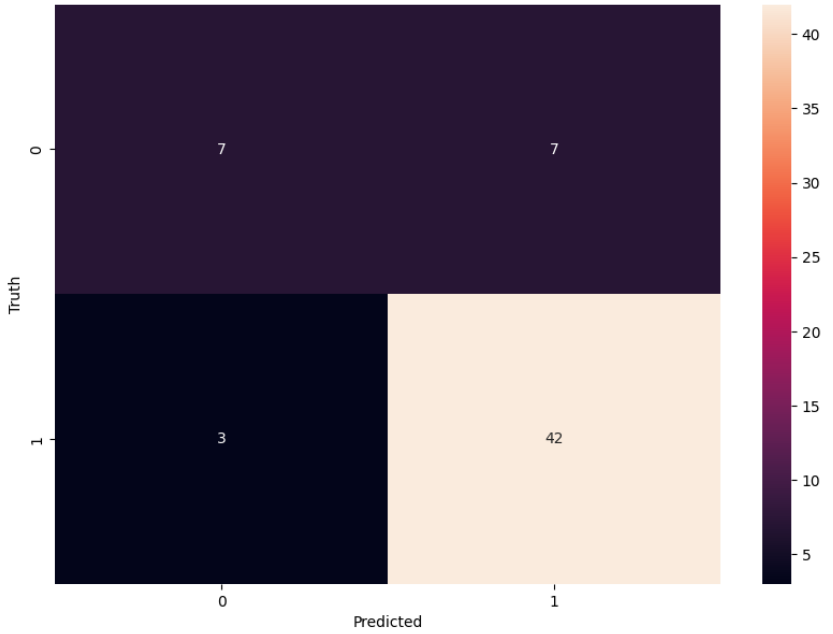

Рисунок 2.10 кореляційна матриця результатів роботи Логістичної регресії

Дерево рішень

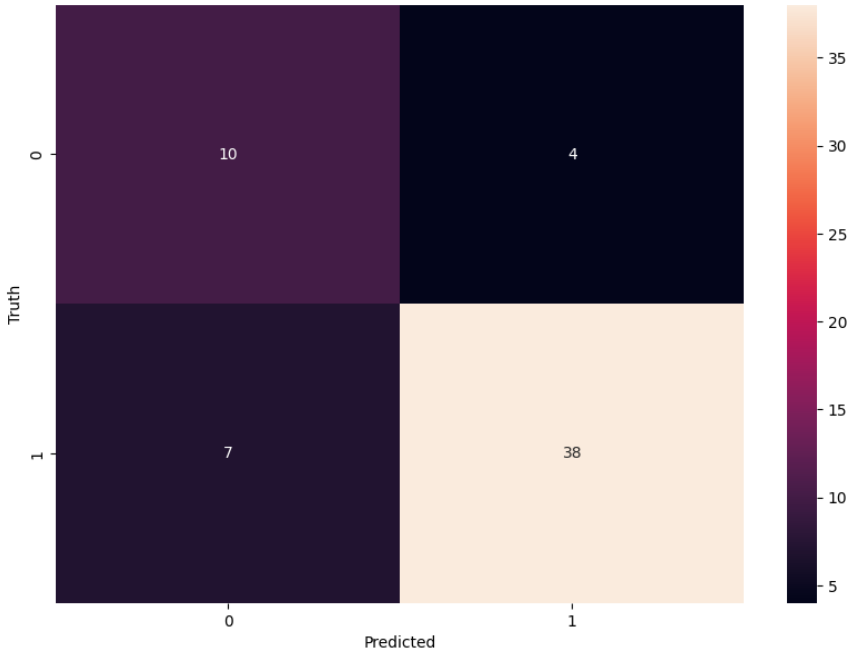

Рисунок 2.11 кореляційна матриця результатів роботи Дерева рішень

Для наочності виведемо на екран побудоване дерево рішень

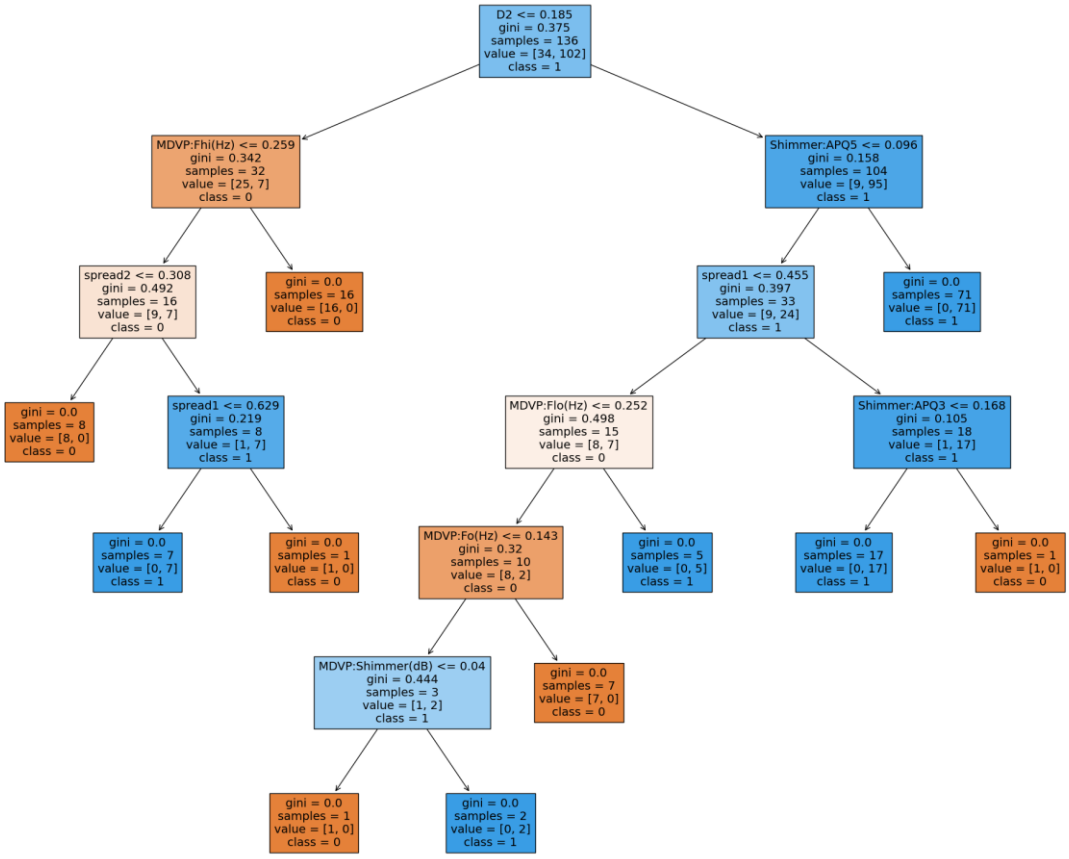

Рисунок 2.12 Дерево рішень

Випадковий ліс (Random Forest)

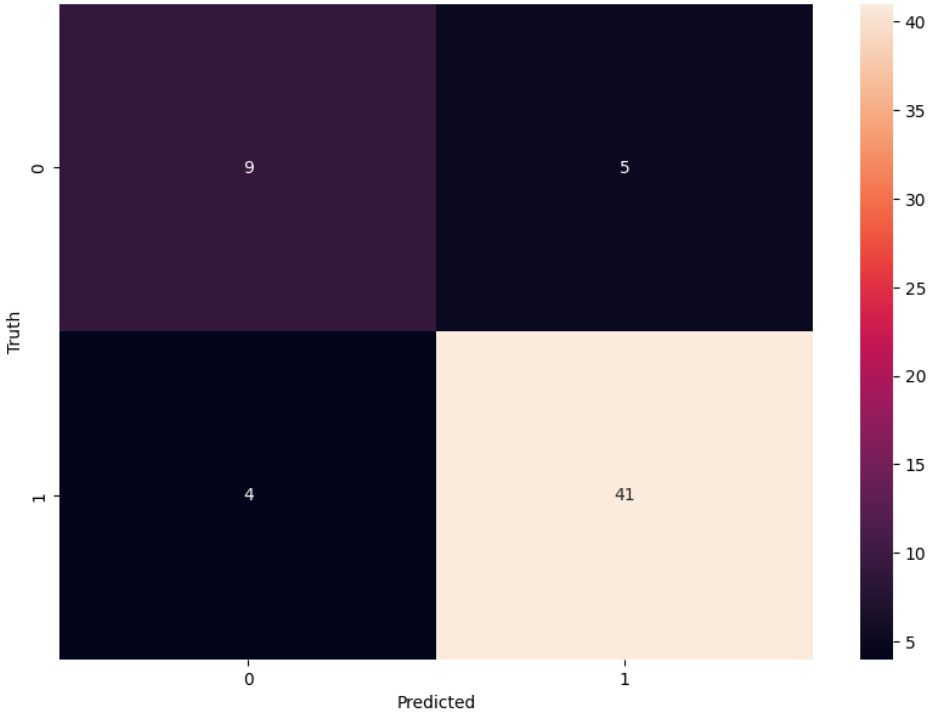

Рисунок 2.13 кореляційна матриця результатів роботи Випадкового лісу

Метод опорних векторів (SVM)

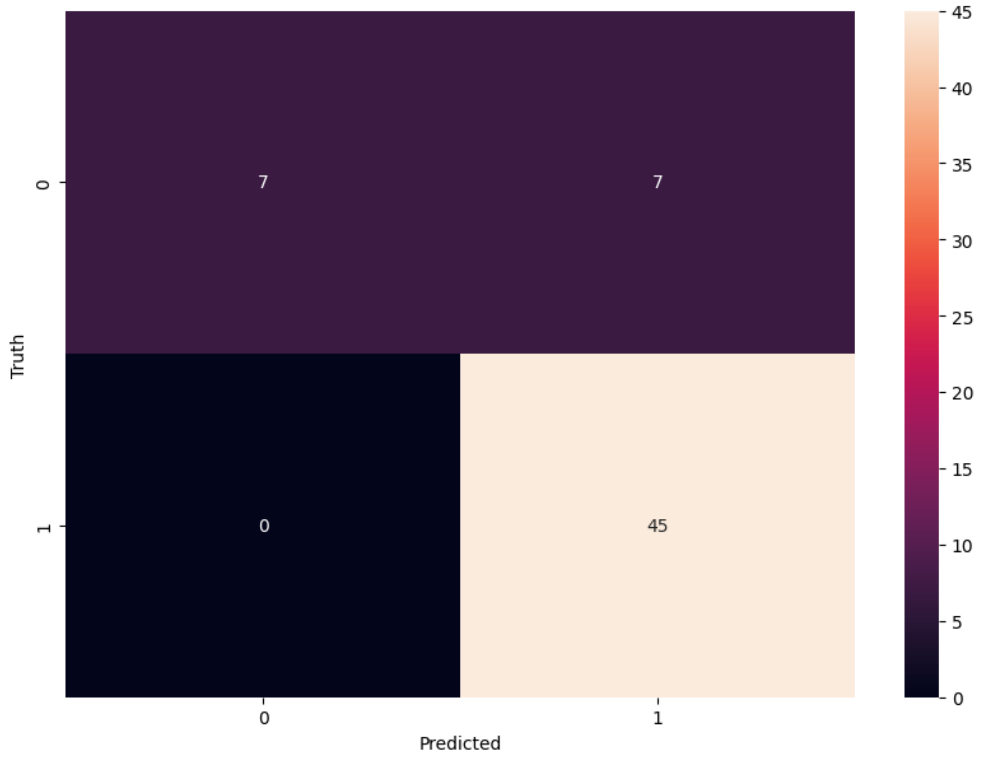

Рисунок 2.14 кореляційна матриця результатів роботи SVM

Метод найближчих сусідів(KNN)

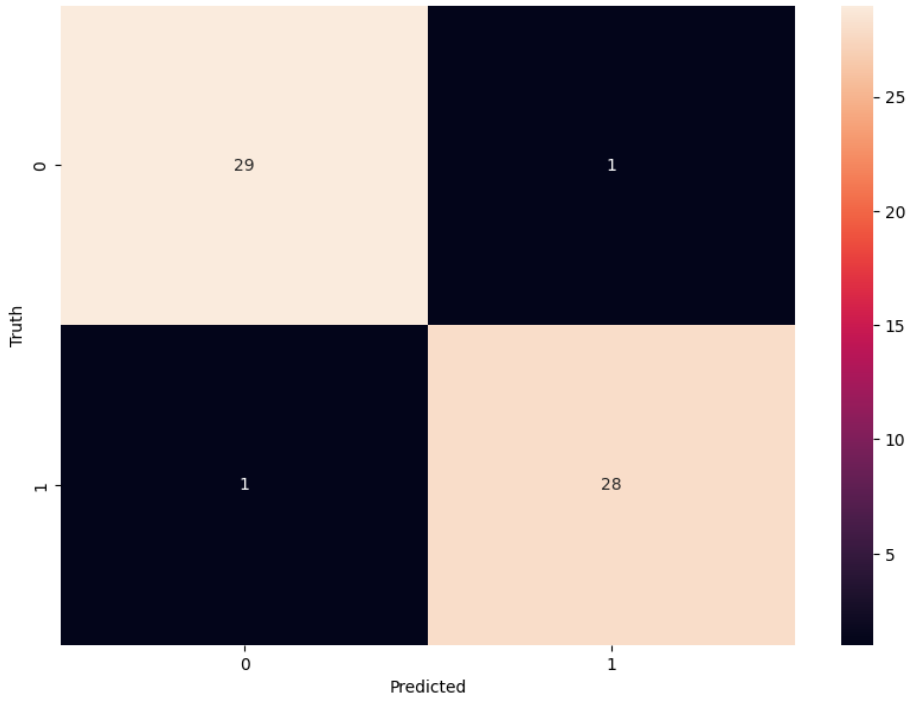

Рисунок 2.15 кореляційна матриця результатів роботи KNN

Метод градієнтного бустингу

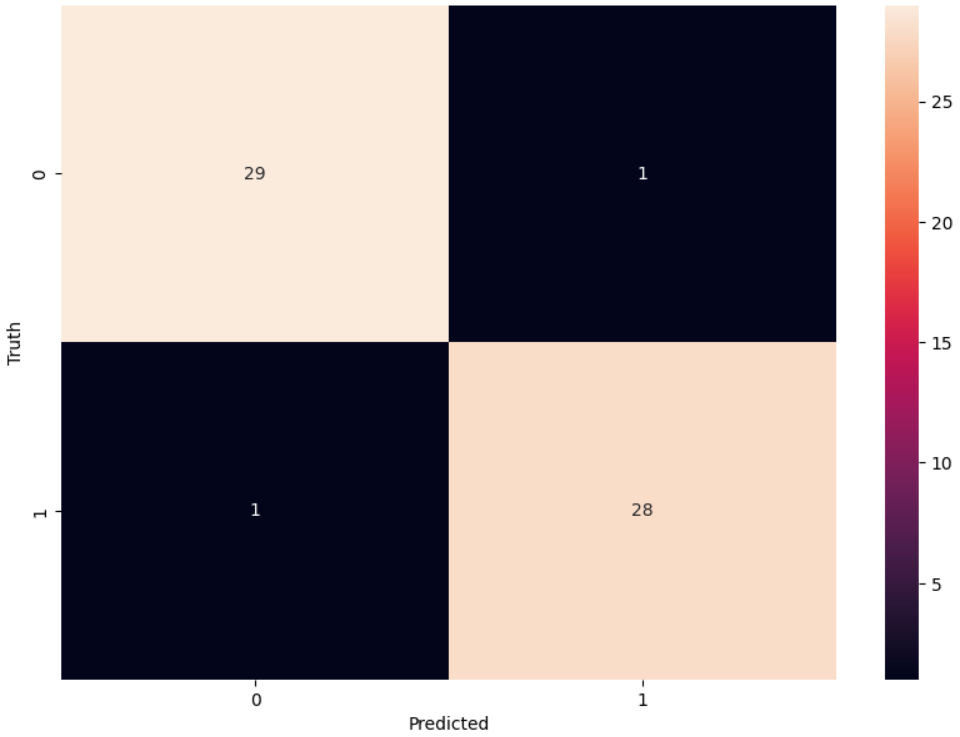

Рисунок 2.16 кореляційна матриця результатів роботи градієнтного бустингу

При аналізі набору даних ми побачили що кількість здорових та хворих людей значно відрізняється, тому проведемо штучне балансування вибірки та подивимось до яких результатів це призведе.

2.4 Штучне балансування набору даних

Проводиться в такі етапи:

1. Вибір меншого класу: Спочатку обирається клас, що має менший обсяг, тобто меншу кількість прикладів.

2. Вибір кращих сусідів: Для кожного прикладу з цього класу обчислюється відстань до всіх інших прикладів цього ж класу в просторі ознак. Потім обирається один або кілька найближчих сусідів кожного прикладу.

3. Генерація штучних зразків: Для кожного обраного прикладу збільшення штучних зразків. Це робиться шляхом створення нового прикладу, який лежить на лінії між обраним прикладом та одним з його сусідів (або декількома найближчими сусідами), але на випадковій відстані від обраного прикладу.

4. Додавання штучних зразків: Ці нові штучні зразки додаються до набору даних.

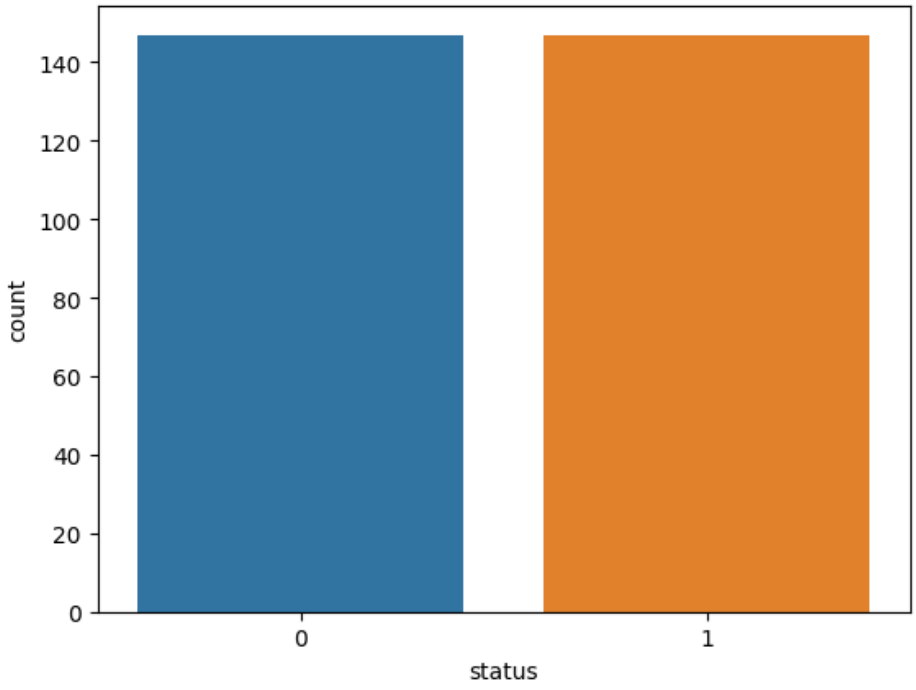

Рисунок 2.17 Відношення кількості здорових до хворих після балансування вибірки

З діаграми бачимо що ми отримали рівне співвідношення між обома класами. Тепер вибірка складає 294 рядки.

Проаналізуємо точність роботи алгоритмів на даному етапі обробки даних.

	Model	Accuracy
0	Logistic Regression	0.847458
1	Decision Tree	0.983051
2	Random Forest	0.983051
3	Support Vector Machine(linear)	0.881356
4	KNN	0.966102
5	Gradient Boosting	0.966102

Рисунок 2.18 Точність роботи алгоритмів

Бачимо значний приріст в точності роботи моделей Decision tree та Random forest.

2.5 Визначення найважливіших атрибутів, застосування методу PCA

Спробуємо спростити наші моделі шляхом відбору ознак, що містять в собі найбільшу кількість інформації, для цього використаємо PCA.

Будемо обирати ознаки які містять не менше 95% інформації

```
Number of components chosen: 8

(294, 8)

array([0.49169323, 0.20461988, 0.08976534, 0.05662747, 0.03931392,
       0.03725068, 0.02522058, 0.01619987])
```

Рисунок 2.19 Кількість компонент та відсортований масив кількості інформації, що містить в собі кожна компонента

Отримали 8 компонент які будуть приймати участь в подальшому навчанні моделей.

Виведемо точність роботи алгоритмів на даному етапі.

	Model	Accuracy
0	Logistic Regression	0.847458
1	Decision Tree	0.932203
2	Random Forest	0.983051
3	Support Vector Machine(linear)	0.864407
4	KNN	0.966102
5	Gradient Boosting	1.000000

Рисунок 2.20 Точність роботи алгоритмів

Як ми бачимо точність роботи алгоритмів Decision tree та SVM впала, але ми також отримали значний приріст в точності роботи алгоритму Gradient Boosting.

2.6 Дослідження поведінки методу Градієнтного бустингу

Побудуємо 2 графіки, на яких будуть зображення значення функції втрат, відповідно до кількості побудованих дерев.

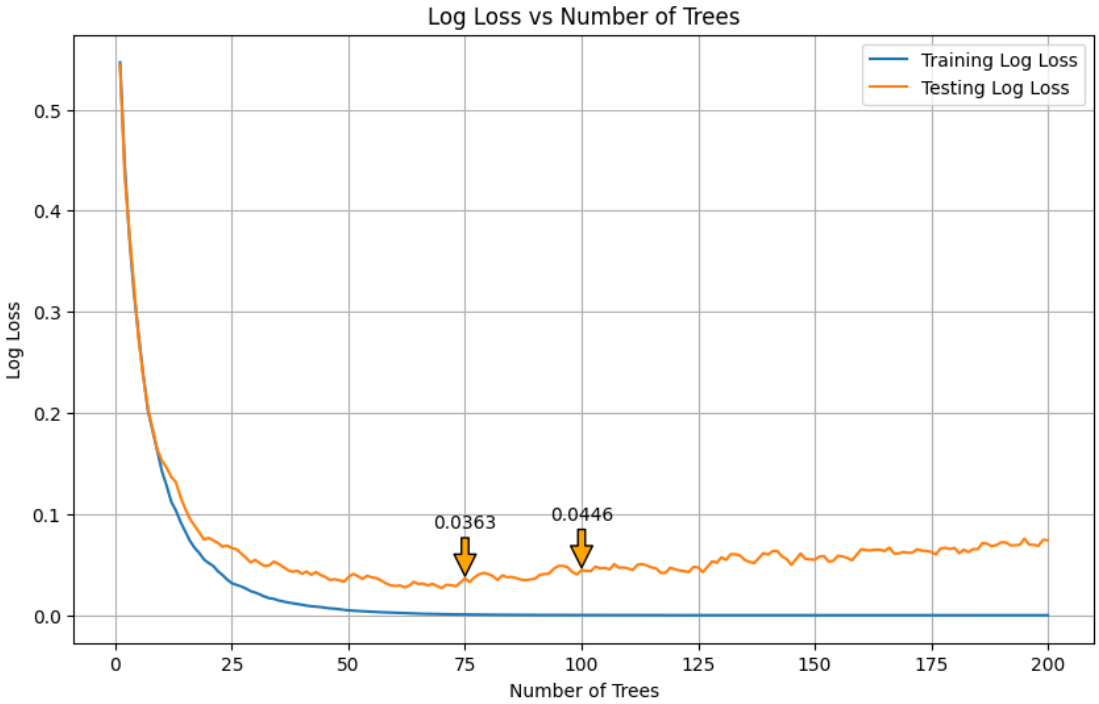

Рисунок 2.21 Значення функції втрат відповідно до кількості дерев (до застосування PCA)

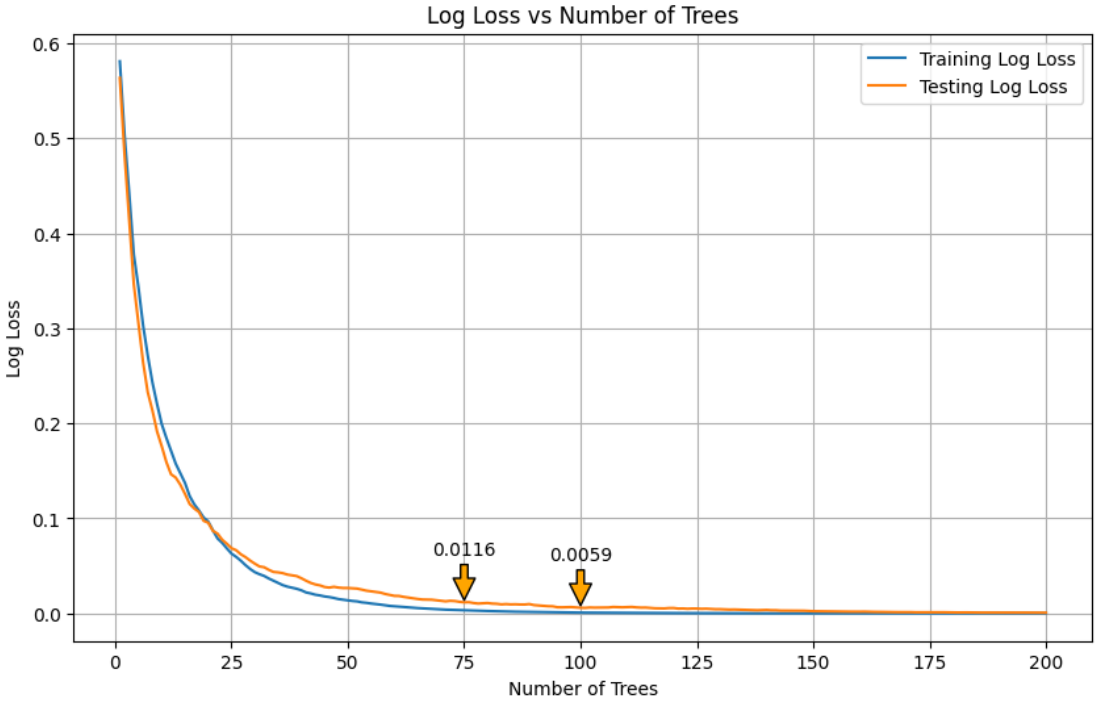

Рисунок 2.22 Значення функції втрат відповідно до кількості дерев (після застосування PCA)

Із графіку чітко бачимо, що відбувається перенавчання після появи 75 дерев,

Звідси і покращення результату після застосування PCA.

Також порівняємо значення інших стандартних метрик

	precision	recall	f1-score	support
0	1.00	0.93	0.97	30
1	0.94	1.00	0.97	29
accuracy			0.97	59
macro avg	0.97	0.97	0.97	59
weighted avg	0.97	0.97	0.97	59

Рисунок 2.23 метрики (до застосування PCA)

	precision	recall	f1-score	support
0	1.00	1.00	1.00	30
1	1.00	1.00	1.00	29
accuracy			1.00	59
macro avg	1.00	1.00	1.00	59
weighted avg	1.00	1.00	1.00	59

Рисунок 2.24 метрики (після застосування PCA)

Бачимо, що значення всіх метрик, після застосування PCA задовольняють нашим очікуванням.

ВИСНОВОК

В роботі побудова комп’ютерна модель для визначення людей на хворобу Паркінсона на основі голосових маркерів. Використовувались такі методи машинного навчання: метод опорних векторів, дерево рішень, випадковий ліс, логістична регресія, метод пошуку найближчих сусідів, метод градієнтного бустингу.

Визначено, що для наявного набору даних, серед всіх інших обраних методів машинного навчання, метод градієнтного бустингу показує накращі результати в точності класифікації, має точність 1.

Масштабування даних, в якості першого кроку, здійснювалась з урахуванням особливостей конкретних даних. Я виявив, що розподіл всіх атрибутів, окрім: HNR, RPDE, DFA, spread1, spread2, D2, PPE має значну асиметрію в лівий бік. Взявши це до уваги, було прийнято рішеннння масштабувати дані в інтервал від 0 до 1. Масштабування дозволило суиєво підвищити точність класифікації.

Вирішення проблеми незбалансованості кількості хворих людей до здорових стало другим кроком в даній роботі. Замінили відношення три до одного, на один до одного.

Також важливим було звернути увагу на висококорельовані ознаки, оскільки вони несуть в собі подібну інформації про набір даних, можуть призводити до перенавчання та збільшувати обчислювалдьну складність, такими о знаками є: MDVP: jitter (%), MDVP: jitter (Abs), MDVP: RAP, MDVP: PPQ, Jitter: DDP, - показники зміни основної частоти та Shimmer:APQ5, MDVP:APQ, Shimmer:DDA - показники варіації амплітуди. Було прийнято рішення зменшувати простір ознак.

Цікавим виявився той факт, що алгоритм градієнтного бустингу показав кращі результати після застосування методу PCA, для зниження розмірності простору ознак, з 22 до 8 компонент, точність класифікації зросла від 0.97 до 1.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Parkinson Disease Detection. URL: https://www.kaggle.com/datasets/debasisdotcom/parkinson-disease-detection (дата звернення: 01.08.2023).

2. Parkinson’s Disease Data Set. URL: https://www.kaggle.com/datasets/vikasukani/parkinsons-disease-data-set (дата звернення: 17.04.2024).

3. Dr. Deepail R Vora, Dr. Gresha S Bhatia, Python Machine Learning Projects BPB Publications (Березень 13, 2023) 85 с.

4. Clustering. URL: https://scikit-learn.org/stable/modules/clustering.html (дата звернення: 15.05.2024).

5. Parkinson’s Disease: Causes, Symptoms, and Treatments. URL: https://www.nia.nih.gov/health/parkinsons-disease (дата звернення: 01.06.2024).

Лістинг програми parkinson.ipynb

```
# завантажимо набір даних
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import *
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
df = pd.read_csv(".\Parkinsson disease.csv")
df.info()

# Виведемо короткі відомості про дані

df.isnull().sum()

df.head()

df.describe()

# Виведемо відношення кількості здорових та хворих
ax = sns.countplot(x="status", data=df)
print(df.status.value_counts())
print(df.shape)

# Виведемо розподіл атрибутів
grid_size = (8,3)
fig = plt.figure(figsize =(10, 10))
column_names = data.select_dtypes(exclude='object').columns
for i, column_name in enumerate(column_names):
    fig.add_subplot(grid_size[0], grid_size[1], i + 1)
    plot = sns.histplot(data[column_name], kde = True, color =
'blue')
    plot.set_xlabel(column_name, fontsize = 16)
plt.tight_layout()
plt.show()

# Виведемо коробкову діаграму атрибутів
grid_size = (8,3)
fig = plt.figure(figsize =(10, 14))
column_names = data.select_dtypes(exclude='object').columns
for i, column_name in enumerate(column_names):
    fig.add_subplot(grid_size[0], grid_size[1], i + 1)
    plot = sns.boxplot(y=data[column_name], x=y, color =
'aquamarine')
    plot.set_xlabel(column_name, fontsize = 16)
plt.tight_layout()
plt.show()
# Виведемо кореляційну матрицю
```

```
data_clean = data.drop(columns=['name'])
correlation = data_clean.corr()
plt.figure(figsize = (15,7))
sns.heatmap(correlation.loc[::-1,::-1])
plt.show()

# Порівняння нормалізаціїї та стандартизації
scaler = MinMaxScaler()
data2 = data.drop(['status','name'],axis='columns')
stock_df = scaler.fit_transform(data2)
X = stock_df
y = data["status"]
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.33,random_state=42)
X_t = pd.DataFrame(X_train)
X_t.describe().round(2)

results = [
    {'Model': 'Logistic Regression', 'Accuracy':
get_score(X_train, y_train, X_test, y_test,
LogisticRegression(random_state=37))},

    {'Model': 'Decision Tree', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
DecisionTreeClassifier(random_state=38))},

    {'Model': 'Random Forest', 'Accuracy': get_score(X_train,
y_train, X_test, y_test, RandomForestClassifier(n_estimators=20,
random_state=39))},

    {'Model': 'Support Vector Machine(linear)', 'Accuracy':
get_score(X_train, y_train, X_test, y_test, SVC(kernel='linear',
random_state=40))},

    {'Model': 'KNN', 'Accuracy': get_score(X_train, y_train,
X_test, y_test, KNeighborsClassifier())},

    {'Model': 'Gradient Boosting', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
GradientBoostingClassifier(random_state=41))},
]
results = pd.DataFrame(results)
results

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data2 = data.drop(['status','name'],axis='columns')
stock_df = scaler.fit_transform(data2)

X = stock_df
y = data["status"]
```

```
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.33,random_state=42)

X_t = pd.DataFrame(X_train)
X_t.describe().round(2)

results = [
    {'Model': 'Logistic Regression', 'Accuracy':
get_score(X_train, y_train, X_test, y_test,
LogisticRegression(random_state=37))},

    {'Model': 'Decision Tree', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
DecisionTreeClassifier(random_state=38))},

    {'Model': 'Random Forest', 'Accuracy': get_score(X_train,
y_train, X_test, y_test, RandomForestClassifier(n_estimators=20,
random_state=39))},

    {'Model': 'Support Vector Machine(linear)', 'Accuracy':
get_score(X_train, y_train, X_test, y_test, SVC(kernel='linear',
random_state=40))},

    {'Model': 'KNN', 'Accuracy': get_score(X_train, y_train,
X_test, y_test, KNeighborsClassifier())},

    {'Model': 'Gradient Boosting', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
GradientBoostingClassifier(random_state=41))},
]
results = pd.DataFrame(results)
results


# Демонстрація роботи методів машинного навчкання
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=16)
logreg.fit(X_train, y_train)
logreg.score(X_test, y_test)
y_pred = logreg.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sns.heatmap(cnf_matrix, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print('-'*80)
print('Logistian regression accuracy:', accuracy_score(y_test,
y_pred ))



from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```
DTC=DecisionTreeClassifier()
DTC.fit(X_train, y_train)
y_pred_DTC = DTC.predict(X_test)
cnf_dtc_matrix = metrics.confusion_matrix(y_test, y_pred_DTC)
plt.figure(figsize=(10,7))
sns.heatmap(cnf_dtc_matrix, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print('-'*80)
print('Decision tree accuracy:',accuracy_score(y_test,
y_pred_DTC))
print('-'*80)
fig = plt.figure(figsize=(25,20))
tree = plot_tree(DTC, feature_names = [ c for c in
df.drop(['name'],axis=1).columns], class_names=['0','1'],
filled=True)


from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
RM = RandomForestClassifier(n_estimators=20)
RM.fit(X_train, y_train)
RM.score(X_test, y_test)
y_pred_RM = RM.predict(X_test)
cm = confusion_matrix(y_test, y_pred_RM)
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print('-'*80)
print('Random Forest accuracy:',accuracy_score(y_test, y_pred_RM))


from sklearn.svm import SVC
SVM = SVC(probability=True, kernel = 'linear')
SVM.fit(X_train,y_train)
SVM.score(X_test, y_test)
y_pred_SVM = SVM.predict(X_test)
cm_svm = confusion_matrix(y_test, y_pred_SVM)
plt.figure(figsize=(10,7))
sns.heatmap(cm_svm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print('-'*80)
print('SVM accuracy:',accuracy_score(y_test, y_pred_SVM))
print('-'*80)


from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
knn_model.score(X_test, y_test)
```

```
y_pred = knn_model.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sns.heatmap(cnf_matrix, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print('-'*80)
print('KNN accuracy:', accuracy_score(y_test, y_pred ))
print('-'*80)


from sklearn.ensemble import GradientBoostingClassifier
grad_boost_model =  GradientBoostingClassifier()
grad_boost_model.fit(X_train, y_train)
grad_boost_model.score(X_test, y_test)
y_pred = knn_model.predict(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,7))
sns.heatmap(cnf_matrix, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
print('-'*80)
print('Gradient boosting accuracy:', accuracy_score(y_test, y_pred
))
print('-'*80)


# Балансування вибірки та аналіз роботи алгоритмів після
from imblearn.over_sampling import SMOTE
X = data.drop(['status','name'],axis='columns')
y = data["status"]

smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X, y)
smote_data = pd.concat([X_smote, y_smote], axis=1)

ax = sns.countplot(x="status", data=smote_data)
print(smote_data.status.value_counts())
print(smote_data.shape)
scaler = MinMaxScaler()

X_smote_train, X_smote_test, y_smote_train, y_smote_test =
train_test_split(X_smote,
y_smote,train_size=0.8,random_state=42)

X_smote_train = scaler.fit_transform(X_smote_train)
X_smote_test = scaler.transform(X_smote_test)
X_smote = scaler.fit_transform(X_smote)

X_train = X_smote_train
X_test = X_smote_test
y_train = y_smote_train
```

```
y_test = y_smote_test


results = [
    {'Model': 'Logistic Regression', 'Accuracy':
get_score(X_train, y_train, X_test, y_test,
LogisticRegression(random_state=37))},

    {'Model': 'Decision Tree', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
DecisionTreeClassifier(random_state=38))},

    {'Model': 'Random Forest', 'Accuracy': get_score(X_train,
y_train, X_test, y_test, RandomForestClassifier(n_estimators=20,
random_state=39))},

    {'Model': 'Support Vector Machine(linear)', 'Accuracy':
get_score(X_train, y_train, X_test, y_test, SVC(kernel='linear',
random_state=40))},

    {'Model': 'KNN', 'Accuracy': get_score(X_train, y_train,
X_test, y_test, KNeighborsClassifier())},

    {'Model': ' Gradient Boosting', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
GradientBoostingClassifier(random_state=41))},
]
results = pd.DataFrame(results)
results
```

```
# Застосування PCA та аналіз результатів

from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler

pca = PCA(n_components=0.95)
X_smote_pca = pca.fit_transform(X_smote)
print(f"Number of components chosen: {pca.n_components_}")
X_smote_pca.shape


pca.explained_variance_ratio


X_smote_train_pca, X_smote_test_pca, y_smote_train, y_smote_test =
train_test_split(X_smote_pca,
y_smote,train_size=0.8,random_state=42)
X_train = X_smote_train_pca
X_test = X_smote_test_pca
y_train = y_smote_train
y_test = y_smote_test
```

```
results = [
    {'Model': 'Logistic Regression', 'Accuracy':
get_score(X_train, y_train, X_test, y_test,
LogisticRegression(random_state=37))},

    {'Model': 'Decision Tree', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
DecisionTreeClassifier(random_state=38))},

    {'Model': 'Random Forest', 'Accuracy': get_score(X_train,
y_train, X_test, y_test, RandomForestClassifier(n_estimators=20,
random_state=39))},

    {'Model': 'Support Vector Machine(linear)', 'Accuracy':
get_score(X_train, y_train, X_test, y_test, SVC(kernel='linear',
random_state=40))},

    {'Model': 'KNN', 'Accuracy': get_score(X_train, y_train,
X_test, y_test, KNeighborsClassifier())},

    {'Model': ' Gradient Boosting', 'Accuracy': get_score(X_train,
y_train, X_test, y_test,
GradientBoostingClassifier(random_state=41))},
]
results = pd.DataFrame(results)
results
```

 # Дослідження поведінки функції втрат, при застосуванні Градієнтного бустингу.

```
from sklearn.metrics import log_loss, recall_score,
classification_report

def loss_func_visual(X_train,X_test,y_train,y_test):
    clf = GradientBoostingClassifier(n_estimators=200,
learning_rate=0.2, max_depth=3, random_state=42)
    clf.fit(X_train, y_train)
    y_pred_model = clf.predict(X_test)

    recall = recall_score(y_test, y_pred_model)
    print(f'Recall: {recall:.4f}')

    # Отримання log_loss на кожній ітерації
    train_loss = np.zeros(clf.n_estimators, dtype=np.float64)
    test_loss = np.zeros(clf.n_estimators, dtype=np.float64)
    last_loss = 0

    for i, y_pred in enumerate(clf.staged_predict_proba(X_train)):
        train_loss[i] = log_loss(y_train, y_pred)
```

```
    for i, y_pred in enumerate(clf.staged_predict_proba(X_test)):
        test_loss[i] = log_loss(y_test, y_pred)
        last_loss = test_loss[i]

    # 5. Візуалізація графіку
    plt.figure(figsize=(10, 6))
    plt.plot(np.arange(clf.n_estimators) + 1, train_loss,
label='Training Log Loss')
    plt.plot(np.arange(clf.n_estimators) + 1, test_loss,
label='Testing Log Loss')

    # Додавання анотацій для test_loss при 80 та 100 деревах
    trees = [75, 100]
    for t in trees:
        plt.annotate(f'{test_loss[t-1]:.4f}',
                     xy=(t, test_loss[t-1]),
                     xytext=(t, test_loss[t-1]+0.05),
                     arrowprops=dict(facecolor='orange',
shrink=0.05),
                     ha='center')

    plt.xlabel('Number of Trees')
    plt.ylabel('Log Loss')
    plt.title('Log Loss vs Number of Trees')
    plt.legend(loc='upper right')
    plt.grid(True)
    plt.show()
    print(last_loss)
    print(classification_report(y_test, y_pred_model))
```