НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Система розпізнавання серцевої аритмії за сигналами ЕКГ на основі методів машинного навчання»

Виконав:
Студент IV курсу, групи КА-71
Гульчук Всеволод Павлович _____

Керівник:
доцент, к.т.н., доцент
Дідковська М. В. _____

Консультант з економічного розділу:
доцент, к.е.н., доцент
Рощіна Н. В., доцент _____

Консультант з нормоконтролю:
доцент, к.т.н., доцент
Коваленко Анатолій Єпіфанович _____

Рецензент:
к.т.н., доцент кафедри ФПМ
Заболотня Т. М. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Оксана ТИМОЩУК
«25» травня 2021 р.

ЗАВДАННЯ

на дипломну роботу студенту

Гульчуку Всеволоду Павловичу

1. Тема роботи «Система розпізнавання серцевої аритмії за сигналами ЕКГ на основі методів машинного навчання», керівник роботи Дідковська Марина Віталіївна, доцент кафедри ММСА, затверджені наказом по університету від «25» травня 2020 р. №1143-с

2. Термін подання студентом роботи 08.06.2020

3. Вихідні дані до роботи:

4. Зміст роботи:

5. Перелік ілюстративного матеріалу: використані метрики, опис набору даних, графік захворювань, побудова моделі, значення метрик.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент, к.е.н., Рощина Н. В.		

7. Дата видачі завдання 06.03.2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Приміт ка
1	Затвердження теми ДР	06.03.2021-20.03.2021	
2	Аналіз актуальності задачі дослідження	10.03.2021-25.03.2021	
3	Збір та аналіз даних	27.03.2021-14.04.2021	
4	Формулювання задачі дослідження	30.03.2021-05.04.2020	
5	Розробка програмного продукту	15.04.2021-30.04.2021	
6	Аналіз та впорядкування результатів	30.04.2021-06.05.2021	
7	Оформлення пояснювальної записки	07.05.2021-16.05.2021	
8	Оформлення презентації для демонстрації	18.05.2021-22.05.2021	
9	Передзахист диплому	01.06.2021-02.06.2021	
10	Аналіз уточнень	02.06.2021-04.06.2021	
11	Захист дипломної роботи	15.06.2021-18.06.2021	

Студент Всеволод ГУЛЬЧУК

Керівник роботи Марина ДІДКОВСЬКА

РЕФЕРАТ

Дипломна робота 136 с., 29 рис., 14 табл., 2 додатки, 18 джерел.

КЛАСИФІКАЦІЯ, МАШИННЕ НАВЧАННЯ, ДІАГНОСТИКА, ГЛИБИННЕ НАВЧАННЯ ,ПРОГНОЗ, СТЕКІНГ.

У роботі було поставлено та вирішено проблему розпізнавання серцевої аритмії за сигналами ЕКГ методами машинного навчання. Було реалізовано всі етапи створення та покращення моделі машинного навчання а також проаналізовано отримані результати, порівняно альтернативи.

Об’єктом дослідження стали уривки електрокардіограми фіксованої довжини/фіксованої кількості піків, їх статистичні показники.

Предметом дослідження стали методи машинного навчання, в тому числі глибинного навчання, їх ансамблювання та методи їх оптимізації.

ABSTRACT

Thesis 136 pages, 29 figures, 14 tables, 2 appendices, 18 sources.

CLASSIFICATION, MACHINE LEARNING, DIAGNOSTICS, DEEP LEARNING, FORECAST, STACKING.

In the work the problem of recognition of Atrial Fibrillation on ECG signals by machine learning methods was stated and solved. It was transformed by all sorts of machine learning methods and the initial results generated by the alternatives were analysed.

The objects of the study were fragments of the electrocardiogram of fixed length / fixed number of peaks, their statistical indicators.

The subject of the study were methods of machine learning, including deep learning, their assembly and methods of their optimization.

ЗМІСТ

ВСТУП

Останніми роками все більше гостро стає питання автоматизації діагнозів та медичних порад. Тисячі людей просто не мають можливості та часу своєчасно сходити на прийом до лікаря, тим часом як сучасні технології та методи машинного навчання роблять кілком можливим створення систем що могли б автоматизувати або напівавтоматизувати ці процеси.

Однією з найбільш гострих проблем медицини є серцеві хвороби. Серцева аритмія посідає лідуючі місця у причинах смертей. Проте виявлення цієї хвороби на ранніх етапах допомогло б суттєво зменшити негативні впливи цієї хвороби та збільшило б показники здоров'я в цілому.

З іншої сторони, стрімкий розвиток кількості відкритих баз даних та методів обробки, аналізу та машинного навчання дозволяє отримати інструмент до створення таких систем. Ні для кого не секрет що в наші дні алгоритми машинного навчання використовуються і в керуванні автомобілями і іншими роботами, і при розпізнаванні зображень, а також майже в кожній сфері людського функціонування. Через те що вони себе там себе дуже добре зарекомендували люди почали впроваджувати медичні системи що ґрунтувалися б на алгоритмах машинного навчання.

Гостро також постає етичне питання впровадження подібних систем, адже не є зрозумілим хто стає відповідальним за помилку. Так ось, є дві точки зору вирішення цього питання. По-перше, кожна використовувати покази подібних систем лише як консультативні системи. Наприклад, не цілком покладатися на покази, а при сигналі що щось пішло не так звернутися до лікаря. В такому випадку остаточне рішення все ж прийматиме лікар, а системи штучного інтелекту лише перейматимуть на себе функції первинного аналізу та діагностики. З іншої сторони, нещодавні дослідження Ілона Маска ефективності автопілотів в автомобілях показали, що при використанні автопілоту ймовірність аварії зменшується у десятки разів, що означає що

сучасним методам машинного навчання є підстава довіряти навіть більше ніж конкретним експертам своєї галузі.

Саме дослідженню питанню застосування методів машинного навчання до діагностування серцевої аритмії і буде присвячена моя робота.

Тематика першого розділу присвячена огляду медичної проблеми та існуючим шляхам її дослідження за допомогою машинного навчання, а також огляду існуючої літератури з цього питання.

У другому розділі присутній огляд теоретичної частини обраного шляху вирішення частини.

Третій розділ присвячений результатам застосування обраних методів та їх порівняльному аналізу.

У четвертому розділі був проведений функціонально-вартісний аналіз програмного продукту.

РОЗДІЛ 1 ОГЛЯД ПРОБЛЕМИ РОЗПІЗНАВАННЯ СЕРЦЕВОЇ АРИТМІЇ ЗА ДОПОМОГОЮ МЕТОДІВ МАШИННОГО НАВЧАННЯ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз актуальності задачі розпізнавання серцевої аритмії за допомогою методів машинного навчання

Проблема розпізнавання серцевої аритмії за допомогою методів машинного навчання є окремим випадком проблеми автоматичної діагностики. Тож у цьому випадку розглянемо проблему автоматичної діагностики як таку.

За даними “The Institute of Medicine at the National Academies of Science” 10 відсотків смертей відбуваються через хибний діагноз. Дослідники цього університету виокремлюють три основні причини таких помилок:

- Неефективне застосування інформаційних технологій в медицині.
- Неінформативний обмін інформацією між пацієнтами а лікарями, а також лікарями між собою (помилки в розмові, некомпетентність, суб’єктивність).
- Система охорони здоров’я що не зацікавлена в глибокій діагностиці причини захворювань.

Задля вирішення цих проблем багато медичних працівників та науковців використовують системи штучного інтелекту та машинного навчання з метою покращити та удосконалити медичну діагностику.

У моїй роботі було взято до уваги одну конкретну проблему: проблеми розпізнавання серцевої аритмії. Актуальність даної проблеми є дуже вагомою: один із сорока українців страждає на цю хворобу. Сама по собі серцева аритмія

– це хвороба порушення частоти чи ритму серцевих скорочень. Такий розлад значно підвищує ризик інсульту і є дуже частою причиною смертей та настання інсультів. Близько 20-25% інсультів розвиваються на фоні порушення ритму серця.

Проблема діагностування даного захворювання ускладнюється тим, що воно може протікати безсимптомно. Серце пацієнта може входити і виходити з аритмії. Через це задля встановлення діагнозу доводиться моніторити ЕКГ серця та спостерігати за його поведінкою протягом дуже тривалого часу.

ЕКГ (Електрокардіограма) – це графік зміни напруги активності серця з часом. Ще до нещодавнього часу такі виміри можна було зробити лише за допомогою великого приладу у лікарні. Пацієнтам доводилося перебувати у лікарні під час вимірів, одягати багато катодів на різні частини тіла. Але сьогодні, з розвитком портативної медично техніки, функція виміру ЕКГ стала доступною у деяких розумних годинниках.

У зв’язку з цим постала можливість накопичувати цю інформацію та встановлювати захворювання ще на ранніх стадіях. Таке діагностування раніше було вкрай проблематичним6 оскільки потребувало від пацієнта довготривалого знаходження у лікарні, До того ж, у лікарні пацієнт знаходиться у штучних умовах, що не відповідають реальному життю, через що аналіз ЕКГ з такого портативного медичного приладу може дати набагато більше інформації.

Але з такою можливістю постає відповідна проблема: якщо отримувати безперервний сигнал ЕКГ з пристрою, то необхідно мати можливість цю всю велику кількість інформації обробити. Старий спосіб коли лікарі вручну переглядали увесь знімок стає вже недоступним оскільки затрати часу, необхідні лікарю для аналізу ЕКГ людини протягом такого великого інтервалу роблять просто неможливим масштабування подібної технології. При масовому використані часу лікарів просто не вистачатиме для того, щоб проаналізувати усі записи ЕКГ. До того ж такий аналіз був би дуже дорогим оскільки потребував би великої кількості часу лікаря.

Вирішуючи цю проблему логічним чином можна підійти до теми цієї роботи, розробка системи автоматичного розпізнавання серцевої дасть можливим вчасному та недорогому діагностуванні розглянутої хвороби, Це задача є на сьогодні є актуальною як ніколи, оскільки портативні пристрої вже існують, а відповідних систем існує мало, тож їх розробка і покращення має для сьогоднішньої та майбутньої сфери охорони здоров’я неабиякий сенс.

1.2 Аналіз існуючих підходів до вирішення задачі розпізнавання серцевої аритмії за допомогою методів машинного навчання

1.2.1 Структура аналізу

Вирішення задачі розпізнавання серцевої аритмії методами машинного навчання складається з кількох підзадач. Кожен конкретний метод вирішення даної задачі є комбінацією методів вирішення цих підзадач. Отже, основними підзадачами є:

- Збір та підготовка даних
    - Збір даних
    - Розмітка даних
    - Первинна обробка даних
    - Передобробка даних
- Розробка моделі розпізнавання
    - Обрання моделі розпізнавання
    - Налаштування параметрів моделі розпізнавання
- Післяобробка результатів моделі
    - Валідація результатів розпізнавання
    - Обґрунтування побудованої моделі

Кожну підзадачу часто можна вирішувати різними способами. Іноді метод вирішення однієї підзадачі не впливає на вирішення інших підзадач.

Отже, для того щоб розглянути існуючі підходи до вирішення задачі в цілому спочатку розберемо існуючі підходи до розв’язку кожної підзадачі, а потім їх комбінації, що є ефективними рішеннями задачі в цілому.

1.2.2 Збір та підготовка даних

Збір даних – це процес отримання даних для подальшої їх обробки та побудови моделі на основі цих даних. Отримувати якісні дані є необхідним оскільки базуючись на них побудована система розпізнавання буде діяти у реальному світі. Якщо обрати неякісні дані, це може призвести до неприємних наслідків таких як хибне діагностування, незастосовність у реальному світі, тощо. Через це вже на першому етапі слід уважно вивчити усі доступні опції і обрати джерело майбутніх даних, Обираючи таке джерело потрібно пам’ятати про те, що зібрані дані треба буде також якісно розмітити, що є теж дуже важливим етапом. Тож обираючи підхід для розв’язання підзадачі збору даних потрібно розуміти що цей етап тісно пов’язаний із наступним етапом: розміткою даних.

Розмітка даних – це процес присвоєння кожній одиниці зібраних даних (кожному запису, прикладу, сигналові, тощо) мітки. У рамках задачі розпізнавання серцевої аритмії це є процес встановлення, які сигнали або їх частини є проявами серцевої аритмії. Неякісне або неправильне виконання цього етапу аналізу може призвести до того, що побудована модель буде не повністю або повністю не відповідати поставленій задачі. По суті розмічаючи дані на цьому етапі, ми встановлюємо верхню межу того, як адекватно може функціонувати побудована модель через те, що будь-яка модель буде формалізувати та відновлювати функціональну залежність між даними та їх розміткою. Це може супроводжуватись побудовою складних структур та технік, але отримати кращу точність ніж точність розмітки даних не можна (у межах випадкових відхилень).

Тепер розглянемо найпопулярніші підходи до цих двох тісно пов’язаних етапів.

Основними підходами до збору даних є:

- Збір даних з пацієнтів певної лікарні за допомогою старих приладів
- Збір даних з пацієнтів за допомогою портативних медичних приладів
- Збір даних з відкритих джерел у мережі інтернет
- Купівля відповідних баз даних у приватних компаній

Порівняння між підходами до збору даних показано у таблиці 1.1.

Таблиця 1.1 - Порівняння підходів до збору даних

Підхід	Не потребує багато часу	Не потребує інвестицій	Надає можливість обирати свій формат даних	Надає упевненість у якості даних
Збір даних з пацієнтів певної лікарні за допомогою старих приладів	-	-	+	+
Збір даних з пацієнтів за допомогою портативних медичних приладів	-	-	+	+
Збір даних з відкритих джерел у мережі інтернет	+	+	-	-
Купівля відповідних баз даних у приватних компаній	+	-	+	-

Основними підходами до розмітки даних є:

- Використання експертів, що вручну розмічають кожен екземпляр даних.

- Використання автоматичних систем розмітки, точність та надійність яких вже підтверджена.
- Комбінування автоматичних систем розмітки та експертних оцінок.

Порівняння даних підходів показано у таблиці 1.2.

Таблиця 1.2 - Порівняння підходів до розмітки даних

Підхід	Не потребує багато часу	Не потребує інвестицій	Надає упевненість у якості даних
Використання експертів, що вручну розмічають кожен екземпляр даних	-	-	+
Використання автоматичних систем розмітки, точність та надійність яких вже підтверджена	+	+	-
Комбінування автоматичних систем розмітки та експертних оцінок	+	+	+

Первинна обробка та передобробка даних є наступними двома пов’язаними підзадачами.

Задача первинної обробки полягає в тому щоб з існуючих даних виокремити таку структуру як приклад, і чітко визначити його параметри. Наприклад, у випадку іншої задачі, якщо отримані дані – це сигнали

невизначеної довжини, то після первинної обробки можна отримати фрагменти довжиною по 100 піків ЕКГ. Цей етап іноді міняється місцями з попереднім етапом: розміткою даних.

Задача передобробки даних полягає у тому, щоб з існуючих розмічених прикладів виокремити характеристики, які потім допоможуть моделі відновити структуру зв’язку між прикладом та розміткою. Наприклад, при використанні багатошарової нейронної мережі на наступних етапах можна залишити «сирий» сигнал ЕКГ розраховуючи що система сама знайде необхідні властивості прикладів, а при використанні логістичної регресії кращим підходом є отримання статистичних характеристик кожного прикладу, оскільки за ними часто можна з непоганою точністю відновити структуру навіть нескладною функцією логістичної регресії.

Найпопулярнішими підходами первинної обробки сигналів ЕКГ для даної задачі обробки є:

- Розділення сигналу на підсигнали фіксованої довжини
- Розділення сигналу на підсигнали з фіксованою кількістю HRV піків

Порівняння даних підходів показано у таблиці 1.3.

Таблиця 1.3 - Порівняння підходів до первинної обробки сигналів ЕКГ

Підхід	Фіксована кількість точок на вхід	Фіксована статистична значимість для статистичних властивостей	Фіксований час заміру у пацієнта
Розділення сигналу на підсигнали фіксованої довжини інтервалу часу	-	-	+
Розділення сигналу на підсигнали з фіксованою кількістю HRV-піків	+	+	-

Найпоширенішими підходами передобробки сигналів у даній задачі є:

- Використання виокремлених статистичних ознак
- Використання лише сирого сигналу
- Використання комбінації сирого сигналу і статистичних ознак

Порівняння даних підходів показано у таблиці 1.4.

Таблиця 1.4 - Порівняння підходів до передобробки сигналів ЕКГ

Метод	Можливість ефективного застосування глибинного навчання	Можливість застосування простих моделей	Врахування помилок класифікації різних типів моделей
Використання виокремлених статистичних ознак	-	+	-
Використання лише сирого сигналу	+	-	-
Використання комбінації сирого сигналу і статистичних ознак	+	+	+

1.2.3 Розробка моделі розпізнавання

Розробка моделі розпізнавання є ключовим елементом даної роботи. Це є власне процес відновлення функціональної залежності між даними та розміткою даних, в нашому випадку, побудова функції розпізнавання.

В залежності від обраної моделі буде залежати, наскільки стійка система до викидів, на яких даних вона краще себе показує, наскільки вона стійка з часом, наскільки може автоматично адаптуватися, наскільки пристосована до обраних даних та найважливіше: наскільки адекватно вона виконує задачу розпізнавання.

Отже, процес розробки моделі складається з дох підзадач: обирання моделі та налаштування її параметрів. Часто ці підзадачі виконуються не послідовно, а циклічноб якщо сама модель передбачає зміну своєї структури при навчанні і підбиранні параметрів.

Підзадача обирання моделі є важливим етапом розв’язку задачі. Модель має бути компромісом складності, адекватності результатів розпізнавання та інтерпретовності. Іноді має сенс будувати кілька моделей, та поєднувати їх у ноу модель: ансамбль. Іноді якусь модель можна будувати тільки як допоміжну, щоб, наприклад, встановити певні прояви відновлювальної структури як, наприклад, важливість характеристик.

Основними підходами до обирання моделі є:

- Лінійні моделі (лінійна, логістична регресія)
- Нелінійні моделі (Дерева прийняття рішень, SVM, тощо)
- Нейронні мережі
- Однорідні ансамблі (Random forest, XGBoost, тощо)
- Неоднорідні ансамблі (stacking, bagging, тощо)

Порівняння даних підходів показано у таблиці 1.5.

Таблиця 1.5 - Порівняння підходів до обирання моделі

Підхід	Можливість використання сирих даних (без передобробки)	Швидкість навчання	Перспектива якості моделі	Швидкість розробки	Небезпека перенавчання
Лінійні моделі	-	+	-	+	+
Нелінійні моделі	+	-	+	-	+
Нейронні мережі	+	-	+	-	+
Однорідні ансамблі	+	-	+	-	-
Неоднорідні ансамблі	+	-	+	-	-

Майже всі моделі (не всі) мають у собі гіперпараметри. В залежності від цих гіперпараметрів моделі можна отримати кардинально різні результати. Такими параметрами, наприклад, є крок градієнтного спуску деяких моделей, кількість прихованих шарів нейронної мережі, метод оптимізації цільової функції, тощо.

Основними підходами до налаштування параметрів моделі є:

- Ітеративний метод
- Автоматичні методи підбору
- Використання алгоритмів що не потребують підбору параметрів
- Використання параметрів за замовчуванням

Порівняння даних підходів показано у таблиці 1.6.

Таблиця 1.6 - Порівняння підходів до налаштування параметрів моделі

Підхід	Не потребує часу розробника	Забезпечує максиміза цію метрики	Надає місце для інтуїції розробника	Не потребує багато ресурсів комп’ютера
Ітеративний метод	-	+	+	-
Автоматичні методи підбору	+	+	-	-
Використання алгоритмів що не потребують підбору параметрів	+	-	-	+
Використання параметрів за замовчуванням	+	-	-	+

1.2.4 Післяобробка результатів моделі

Післяобробка результатів моделі є заключним етапом розробки системи розпізнавання серцевої аритмії за допомогою методів машинного навчання. Післяобробка складається з двох підзадач: валідація результатів та їх обґрунтування.

Валідація результатів це перевірка адекватності моделі. Дуже важливо ретельно продумати адекватність валідації, щоб не ввести себе в оманливе

враження що модель працює гарно у час коли це не так. Суть полягає в тому, щоб продумати, як грамотно поділити дані на тестові і дані для тренування, а потім обрати критерій, за яким ми вимірюємо степінь адекватності розпізнавання: метрику. У даній роботі доволі цікавою у своєму підході стала саме ця частина, оскільки потребувала нестандартного підходу до вирішення, що пов’язане з тим, що в обраних даних є дуже довгі записи ЕКГ невеликої кількості людей.

Найпоширенішими підходами до валідації є:

- Випадкове розділення прикладів на множину для тренування та множину для тестування;
- Розділення на множину для тренування та тестування таким чином, щоб у множинах лежали незалежні приклади.
- Тестування на наперед заданих даних, у яких невідома розмітка на сервісі.

Порівняння даних підходів показано у таблиці 1.7.

Таблиця 1.7 - Порівняння підходів до валідації результатів

Підхід	Відсутність можливості перенавчання	Відсутність затрат часу дослідника на реалізацію розділення	Простота використання
Випадкове розділення прикладів	-	+	+
У множинах лежать незалежні приклади.	+	-	-
Тестування на наперед заданих даних, у яких невідома розмітка знаходиться на сервісі.	+	+	+

Підзадача обґрунтування побудованої моделі

Обґрунтування побудованої моделі полягає в тому, щоб візуалізувати отримані результати певним чином, а також пояснити отримані результати і проінтерпретувати, чому той чи інший алгоритм повів себе так чи інакше.

У даній задачі немає сенсу порівнювати ті чи інші підходи, оскільки їх дуже багато і кожен з них може візуалізувати чи обґрунтувати свою вузьку частину побудованої системи. Для прикладу можна навести такі конкретні методи: Матриця похибок, tsne, аналіз помилок. Також цей етап залишає дослідникові багато місця для творчості оскільки основний підхід на даному етапі це використання логіки та заснованих на логіці підходів, оскільки загальноприйняті тут розробити доволі складно.

1.3 Аналіз існуючих рішень, бібліотек та програмних продуктів

Розділимо існуючі рішення на дві категорії: повні та часткові.

Повні рішення є комбінаціями рішень усіх вищенаведених підзадач та є повним розв'язком наведеної проблеми

Часткові рішення є рішеннями окремих підзадач або групи підзадач і може слугувати лише частино вирішення проблеми в цілому.

1.3.1 Повні рішення.

Оскільки такі повні рішення є комерційними, вона не надають доступу до використаних підходів. Вони лише розповідають про те які результати валідації даних, тож просто опишемо дані продукти.

- Apple Watch

Відносно нещодавно вийшов Apple watch 4 – вони використали не нову технологію, взяли ринок тим, що до них є довіра і велика кількість користувачів у фірмі, тож вони внесли великий вклад до довіри до даного продукту. Завдяки ним, іншим подібним проєктам тепер легше пояснювати, чому і як працює такий програмний та апаратний продукт, оскільки більшість людей вже чула про Apple Watch 4.

- Mawi Band

Це український проєкт, який розпочав свою діяльність ще задовго до того як Apple Watch запровадили функцію зчитування та аналізу ЕКГ. Суть проєкту в тому, щоб розробити мобільний пристрій, що може постійно зчитувати ЕКГ людини: а потім аналізувати його за допомогою методів штучного інтелекту на машинного навчання. Проєкт почав бути успішним і займати свою нішу вже давно.

Алгоритми наведених проєктів не можуть бути наведеними оскільки є закритими, але певні їх частини відомі. По-перше, частину сигналів ЕКГ вони

могли зібрати безпосередньо за допомогою розроблених приладів, що є дуже зручним. Такий спосіб вимагає велику кількість ресурсів, але компанії можуть собі таке дозволити.

По-друге, оскільки такі продукти мають виходити на ринок, компанії дуже обережно ставляться до валідації результатів, використовуючи розвинені статистичні методи перевірки. У своїй науковій роботі через відсутність такої кількості аналітичних і фінансових ресурсів розроблю свою власну систему валідації на даних що в мене є, яка буде відповідати певним вимогам та є цікавою за своєю суттю (описана нижче).

1.3.2 Часткові рішення.

Існують бібліотеки для мови програмування python, що узяли на себе функції деяких часткових рішень цієї проблеми. Опишемо їх.

- Hrvanalysis

Це бібліотека, яка має велику кількість функцій, що допомагають з аналізом ЕКГ-сигналів. Якщо точніше, ця бібліотека має наступні реалізовані функції:

- get_time_domain_features – отримання часових характеристик ЕКГ-сигналу.
- get_geometrical_features – отримання геометричних характеристик ЕКГ-сигналу.
- get_frequency_domain_features – отримання частотних характеристик ЕКГ-сигналу.
- get_csi_cvi_features – отрминня csi та cvi індексів, отриманих з графіку Лоренца.
- get_poincare_plot_features – отримання характеристик, отриманих із графіку Пуанкаре.
- sklearn.preprocessing – бібліотека для передобробки даних, розділення множини для навчання і подібних задач.

- sklearn.metrics – бібліотека для отримання існуючих метрик, які надалі можна використовувати при валідації, налаштуванні гіпепараметрів.
- Seaborn – бібліотека для візуалізації отриманих даних на всіх етапах розв’язання задачі, що суттєво допомагає досліднику отримати інтуїтивне бачення проблеми та отриманих результатів.
- Keras – фреймворк для побудови усіх можливих типів нейромереж. Цей фрейморк побудований на базі ядра tensorflow, що дуже оптимізований під задачу створення та використання нейромереж. Також цей фреймворк дуже добре підходить для новачків та побудови прототипів через простоту у використанні, хоча за це і доводиться платати меншою гнучкістю у порівняні з такими фреймворками, як, наприклад, PyTorch.
- sklearn.manifold.TSNE – програмний модуль для візуалізації розділення початкових даних на групи. Це є дуже хорошим інструментом для візуалізації даних, який допоможе зрозуміти степінь структурованості даних.
- Hyperopt – бібліотека, яка надає багато можливостей для автоматичного підбору гіперпараметрів, Вона має уже реалізовані методи перебору та створення сітки гіперпараметрів, що суттєво зменшує необхідну кількість затрачених зусиль дослідника.
- XGBoost – бібліотека створена спеціально для побудови одного типу моделей: XGBoost. Частинами цієї бібліотеки та властивостями моделі є вбудоване ранжування важливостей властивостей, що дозволяє виділити, які властивості прикладів є дійсно важливими, а які можна відкинути при навчанні.
- sklearn.linear_model – бібліотека для побудови простих лінійних моделей.
- h2o – бібліотека для автоматичного підбору моделі з уже налаштованими гіперпараметрами.

1.4 Формалізація постановки задачі дослідження

Формалізувати задачу дослідження можна наступним чином:

Задача полягає у знаходженні та реалізації ефективного методу розпізнавання серцевої аритмії за ЕКГ, що є масштабованим на нові дані без суттєвої втрати ефективності.

Задачу розпізнавання формалізуємо наступним чином:

Існує множина складних об’єктів A, і множина міток B={0, 1}, а також відображення f: A -> B. A1 – підмножина A . Задача полягає в тому, щоб маючи інформацію про відображення f1: A1->B, f1==f на A1, побудувати функцію f*: A->B таким чином, щоб quality(f*) досягло певного адекватного значення. Quality(f*, A2) - метрика, що визначає якість або адекватність побудованої моделі, A2 - тестова множина і визначається як Quality(f*(xi), xi), xi є A2, A2 – підмножина A, що не перетинається з A1.

Формалізувати дану задачу можна по-різному, але вищенаведений спосіб, запропонований мною на мою думку по-новому розкриває сутність задачі розпізнавання.

1.5 Висновки за розділом

У результаті виконання першого розділу можна зробити висновки по актуальності обраної задачі, існуючим підходам до її вирішення а також існуючих рішень та формалізації постановки даної задачі.

У підсумку, результаті дослідження електронних джерел було встановлено, що обрана задача має дуже високий рівень актуальності. Вона не тільки має перспективу допомагати людям зі здоров'ям, а й розвантажити роботу лікарям, яких і так не вистачає у сучасному світі.

Також можна зробити висновок, що на сьогодні уже існує дуже багато підходів до розв'язання цієї комплексної задачі. Кожен такий підхід має свої переваги та недоліки. Обираючи один підхід над іншим потрібно уважно зважити усі наслідки такого рішення, також розрахувати, чи достатньо у дослідника часу та ресурсів на обрання більш точного але сильно більш ємного за кількістю роботи підходу. Дослідження цих підходів є дуже важливою частиною наукової роботи, оскільки надалі під час виконання усіх інших частин, дослідник не раз звертатиметься до таблиць порівняння задля зваженого рішення обрати той чи інший підхід.

Іншим висновком з даного розділу є те, що станом на сьогодні існує багато повних і часткових рішень обраної проблеми. З них можна узяти сильні сторони та намагатися розробити розв'язок, який би відрізнявся від цих готових рішень у кращу сторону по мірі можливості, мав свої переваги. Але використання готових часткових рішень це те що суттєво послабить процес отримання готового продукту, оскільки ці часткові рішення було розроблено тисячами розробників саме з цією метою.

Постановка задачі

1. Розробити модель розпізнавання серцевої аритмії за сигналом ЕКГ
2. Проаналізувати адекватність моделі та порівняти моделі між собою
3. Дослідити існуючі моделі та підходи розпізнавання
4. Сформувати перспективи подальших досліджень

РОЗДІЛ 2 ОГЛЯД ТЕОРЕТИЧНОЇ ЧАСТИНИ ОБРАНИХ МЕТОДІВ ВИРІШЕННЯ ПРОБЛЕМИ

2.1 Методи розпізнавання серцевої аритмії та їх математичні основи

Уданому розділі розглянемо основні методи машинного навчання, які при досліджені проблеми були виокремлені як дуже різні, такі що на них можна виокремити різні прояви даних та класифікувати дані враховуючи можливу різну природу відновлювальної залежності міток від прикладів.

2.1.1 Логістична регресія

Для того, щоб правильно увести поняття логістичної регресії, для початку уведемо поняття логістичної функції:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}, \tag{2.1}$$

де L, k – змінні параметри;

x_0 – параметр «центрування»

Приклад такої функції з параметрами L=1, k=1, x_0=0 показано на рисунку 2.1.

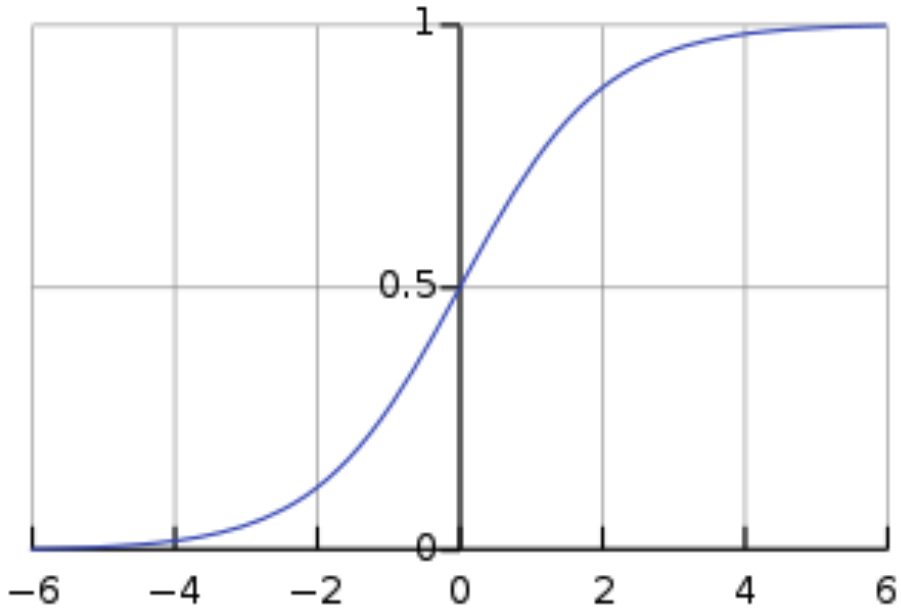


Рисунок 2.1 - Логістична функція

Ми будемо користуватися нею саме в такому вигляді.

Також важливою функцією надалі є функція крос-ентропії також відома як функція логарифмічних втрат:

$$\begin{aligned} \text{Cost}\,(p, y) &= -\log\,(p) \qquad \text{if y} = 1 \\ \text{Cost}\,(\text{p}, y) &= -\log\,(1 - \text{p}) \text{ if y} = 0\text{'} \end{aligned} \tag{2.2}$$

де p – передбачена ймовірність;

y – справжнє значення.

Або в більш загальному вигляді:

$$\text{Cost}(p, y) = -(y \log(p) + (1 - y) \log(1 - p)), \tag{2.3}$$

де p – передбачена ймовірність;

y – справжнє значення.

Ця функція дуже добре відображає зміст функції втрат. Зміст p: передбачення класу для певного прикладу, зміст y: його реальне значення класу. Виходить що ця функція має приймати великих значень коли класи відрізняються, і дорівнювати нулю коли вони співпадають.

Дійсно, як виглядає графік функції при y = 0 показано на рисунку 2.2.

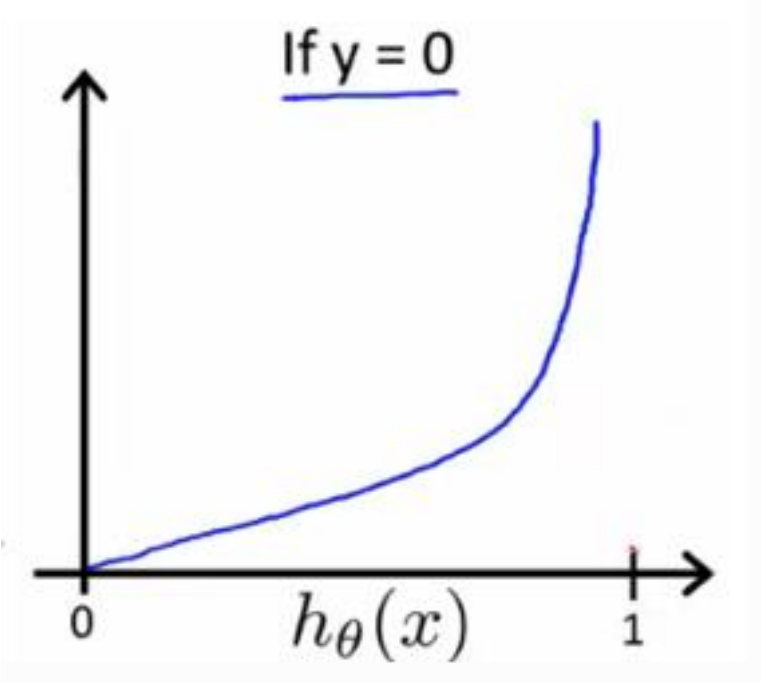


Рисунок 2.2 - Крос-ентропія при y=1

А як при y = 1 показано на рисунку 2.3.

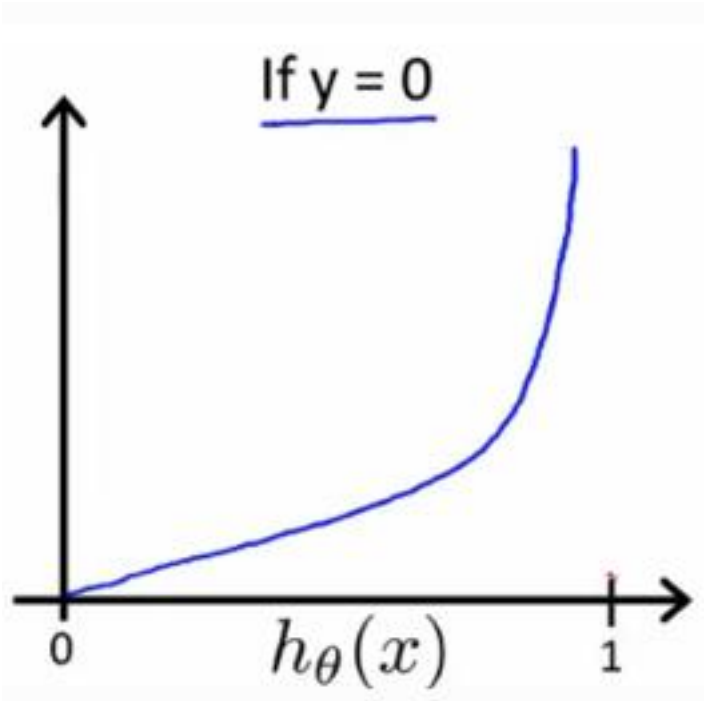


Рисунок 2.3 - Крос-ентропія при y=0

Бачимо, що ця функція як раз відповідає поставленим умовам.

Зауважимо, що оскільки в нашій задачі класифікації кожен приклад належить до одного з двох класів, то y може приймати лише значення 0 та 1.

Тоді яка постає перед нами задача. Беремо Тренувальну вибірку, що є набором $x^{(i)}$ та $y^{(i)}$, i =1, m. $x^{(i)}$- n-вимірний вектор параметрів для кожного прикладу, $y^{(i)}$- його мітка. Модель буде передбачувати ймовірність прикладу $x^{(i)}$ приналежності прикладу до класу 1 базуючитсь на значеннях параметрів прикладу за допомогою наступної формули:

$$h_\theta\left(x^{(i)}\right) = P\left(\text{ class }(x^{(i)}) \ = 1, \theta\right) = \frac{1}{1 + e^{-z\left(x^{(i)}, \theta\right)}}, \tag{2.4}$$

де $x^{(i)}$ – n-вимірний вектор параметрів для кожного прикладу;

$y^{(i)}$– його мітка;

θ – значення вектору параметрів.

z визначається як лінійна комбінація значень параметрів, або скалярний добуток параметрів прикладу з параметрами моделі, які нам і треба підібрати для максимізації адекватності моделі. Отже формула для визначення z:

$$z(x^{(i)}, \theta) = \sum_{j=1}^{n} {x^{(i)}}_j * \theta_j, \tag{2.5}$$

де ${x^{(i)}}_j$ – j-та координата n-вимірного вектору параметрів прикладу;

θ – значення вектору параметрів.

Тоді щоб враховувати, наскільки в середньому добре модель класифікувала кожен приклад з тренувальної вибірки, уведемо таку кост-функцію, або функцію ціни втрат:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right), \tag{2.6}$$

де усі внутрішні функції ми визначали раніше.

Так ось, сутність задачі навчання логістичної регресії полягає у тому, щоб підібрати такі параметри θ, щоб мінімізувати $J(\theta)$ на тренувальній вибірці.

Отже задача навчання логістичної регресії зводиться до задачі мінімізації:

$$\theta^* = \min_{\theta} J(\theta), \tag{2.7}$$

де θ – вектор параметрів моделі

Задача мінімізації може бути виконана по-різному. Наприклад часто використовуються метод градієнтного спуску (найпростіший в реалізації), його модифікації (ефективніше оптимізують).

Хоча логістична регресія і є найпростішою моделлю для класифікації об’єктів, вона також слугує основою для більш складних моделей як, наприклад, нейронна мережа (перцептрон). Також часто у неоднорідних ансамблях використовується логістична регресія для поєднання більш складних моделей саме через свою простоту.

2.1.2 Перцептрон

Елементарний перцептрон складається з елементів трьох типів: S-елементів, A-елементів та одного R-елемента. S-елементи - це шар сенсорів або рецепторів. У фізичному втіленні вони відповідають, наприклад, світлочутливим клітинам сітківки ока або фоторезисторами матриці камери. Кожен рецептор може перебувати в одному з двох станів - спокою або збудження, і тільки в останньому випадку він передає одиничний сигнал в наступний шар, асоціативним елементам.

A-елементи називаються асоціативними, тому що кожному такому елементу, як правило, відповідає цілий набір (асоціація) S-елементів. A-

елемент активізується, як тільки кількість сигналів від S-елементів на його вході перевищило деяку величину θ Таким чином, якщо набір відповідних S-елементів розташовується на сенсорному поле в формі літери «Д», А-елемент активізується, якщо достатня кількість рецепторів повідомило про появу «білої плями світла» в їх околиці, тобто А-елемент буде як би асоційований з наявністю / відсутністю літери «Д» в деякій області.

Сигнали від порушила А-елементів, в свою чергу, передаються в акумулятор R, причому сигнал від і-го асоціативного елемента передається з коефіцієнтом w_i . Цей коефіцієнт називається вагою A-R зв'язку.

Так само як і А-елементи, R-елемент підраховує суму значень вхідних сигналів, помножених на ваги (лінійну форму). R-елемент, а разом з ним і елементарний перцептрон, видає «1», якщо лінійна форма перевищує поріг θ, інакше на виході буде «-1». Математично, функцію, реалізовану R-елементом, можна записати так:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} w_i x_i - \theta\right), \qquad (2.8)$$

де w_i − відповідні ваги

Навчання елементарного перцептрона полягає в зміні вагових коефіцієнтів w_i зв'язків A-R. Ваги зв'язків S-A (які можуть набувати значень {-1; 0; +1}) і значення порогів А-елементів вибираються випадковим чином на самому початку і потім не змінюються.

Після навчання перцептрон готовий працювати в режимі розпізнавання або узагальнення . В цьому режимі перцептроном пред'являються раніше невідомі йому об'єкти, і перцептрон повинен встановити, до якого класу вони належать. Робота перцептрону полягає в наступному: при пред'явленні об'єкта порушила А-елементи передають сигнал R-елементу, який дорівнює сумі відповідних коефіцієнтів w_iЯкщо ця сума позитивна, то приймається рішення,

що даний об'єкт належить до першого класу, а якщо вона негативна - то до другого.

Логічну схему елементарного перцептрону показано на рисунку 2.4.

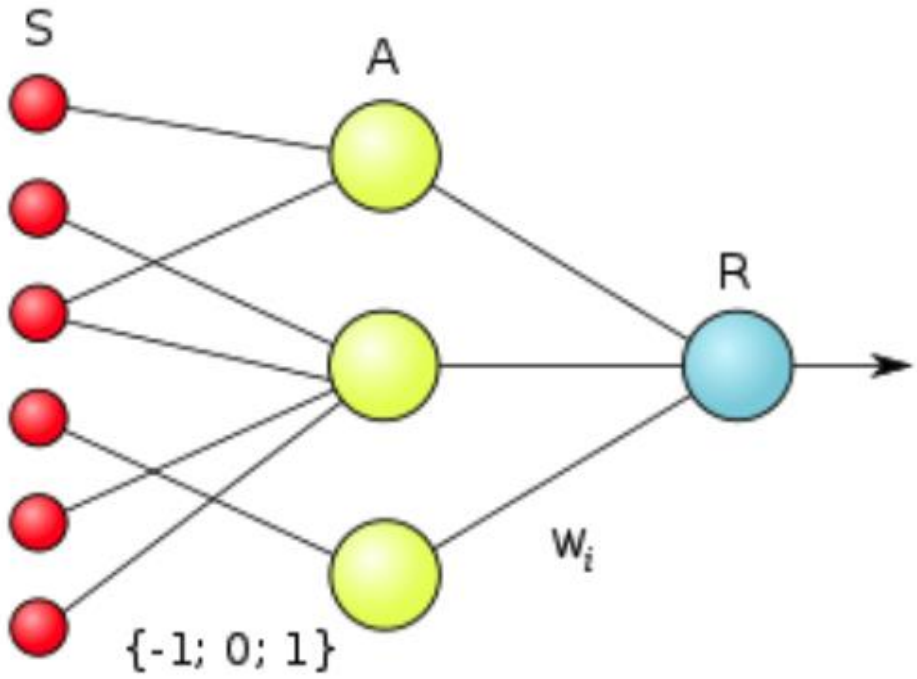


Рисунок 2.4 - Логічна схема елементарного перцептрону

2.1.3 Згорткова нейронна мережа

Згорткова нейронна мережа (англ. Convolutional neural network, CNN) - спеціальна архітектура нейронних мереж, запропонована Яном Лекуном, від самого початку націлена на ефективне розпізнавання зображень.

Згорткова нейронна мережа складається з вхідного шару, прихованих шарів та вихідного шару. У будь-якій нейронній мережі прямого передавання будь-які середні шари називаються прихованими, оскільки їх входи та виходи приховуються функцією активації та кінцевою згорткою. У згортковій нейронній мережі приховані шари включають шари, які виконують згортки. Зазвичай це включає шар, який виконує скалярний добуток ядра згортки із вхідною матрицею шару. Цей добуток, як правило, є скалярним добутком Фробеніуса, і його функція активації зазвичай є ReLU. Коли ядро згортки

ковзає вздовж вхідної матриці шару, операція згортки відображає властивості на наступний шар. Далі слідують інші шари, такі як шари об’єднання, повністю зв’язані шари та шари нормалізації.

Приклад архітектури згорткової нейронної мережі показано на рисунку 2.5.

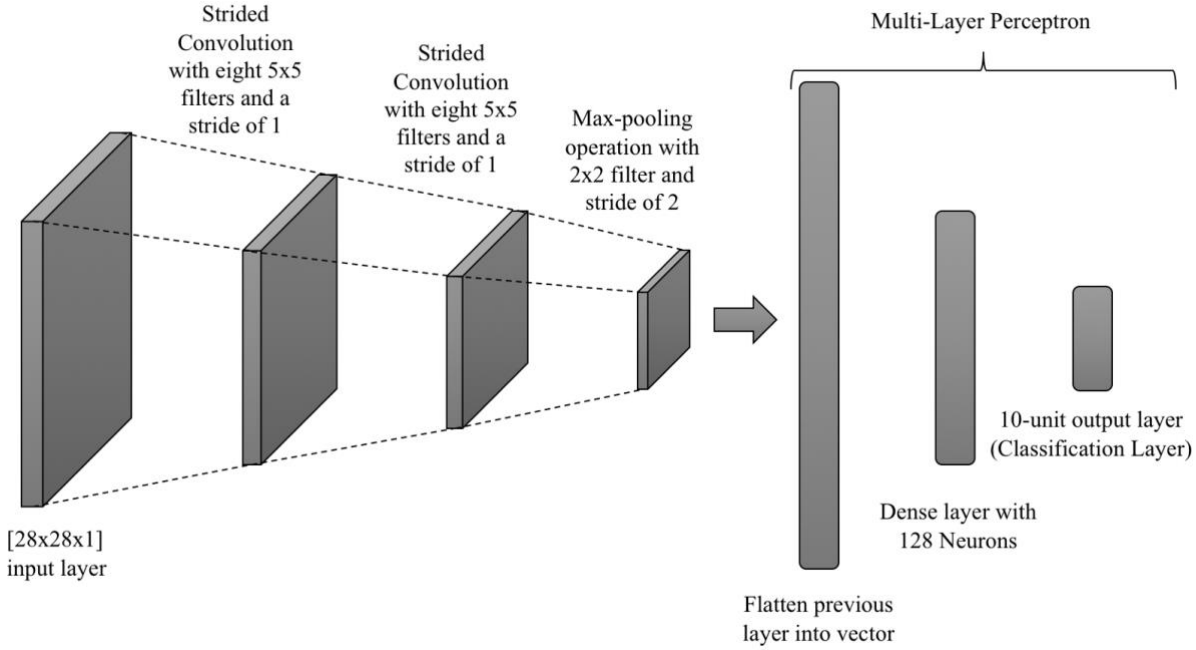


Рисунок 2.5 - Приклад архітектури згорткової нейронної мережі

2.1.4 XGBoost

XGBoost - одна з найпопулярніших і ефективних реалізацій алгоритму градієнтного бустингу на деревах прийняття рішень.

В основі XGBoost лежить алгоритм градієнтного бустингу дерев рішень. Метод найшвидшого бустингу - це техніка машинного навчання для задач класифікації і регресії, яка будує модель передбачення в формі ансамблю слабких пророкують моделей, зазвичай дерев рішень. Навчання ансамблю проводиться послідовно на відміну, наприклад від беггінгу (інший вид неоднорідних ансамблів). На кожній ітерації обчислюються відхилення прогнозів вже навченого ансамблю на навчальній вибірці. Наступна модель,

яка буде додана в ансамбль буде передбачати ці відхилення. Таким чином, додавши передбачення нового дерева до пророкувань навченого ансамблю ми можемо зменшити середнє відхилення моделі, котре є метою оптимізаційної задачі. Нові дерева додаються в ансамбль до тих пір, поки помилка зменшується, або поки не виконується одна з правил "ранньої зупинки".

Розглянемо ілюстрацію бустингу. На ній розглядається поведінка моделі на одній точці абстрактної задачі лінійної регресії. Припустимо, що перша модель ансамблю F завжди видає вибіркове середнє передбачали величини f0. Такий прогноз досить грубе, тому середньоквадратичне відхилення на обраної нами точці буде досить великим. Ми спробуємо це виправити навчивши модель Δ1, яка буде "коригувати" пророкування попереднього ансамблю F0. Таким чином ми отримаємо ансамбль F1, пророкування якого буде підсумовуватися з прогнозів моделей f0 і Δ1. Продовжуючи таку послідовність ми приходимо до ансамблю F4 пророкування якого підсумовується з прогнозів f0, Δ1, Δ2, Δ3, Δ4 і пророкує в точності значення заданої мети.

Схему бустингу проілюстровано на рисунку 2.6.

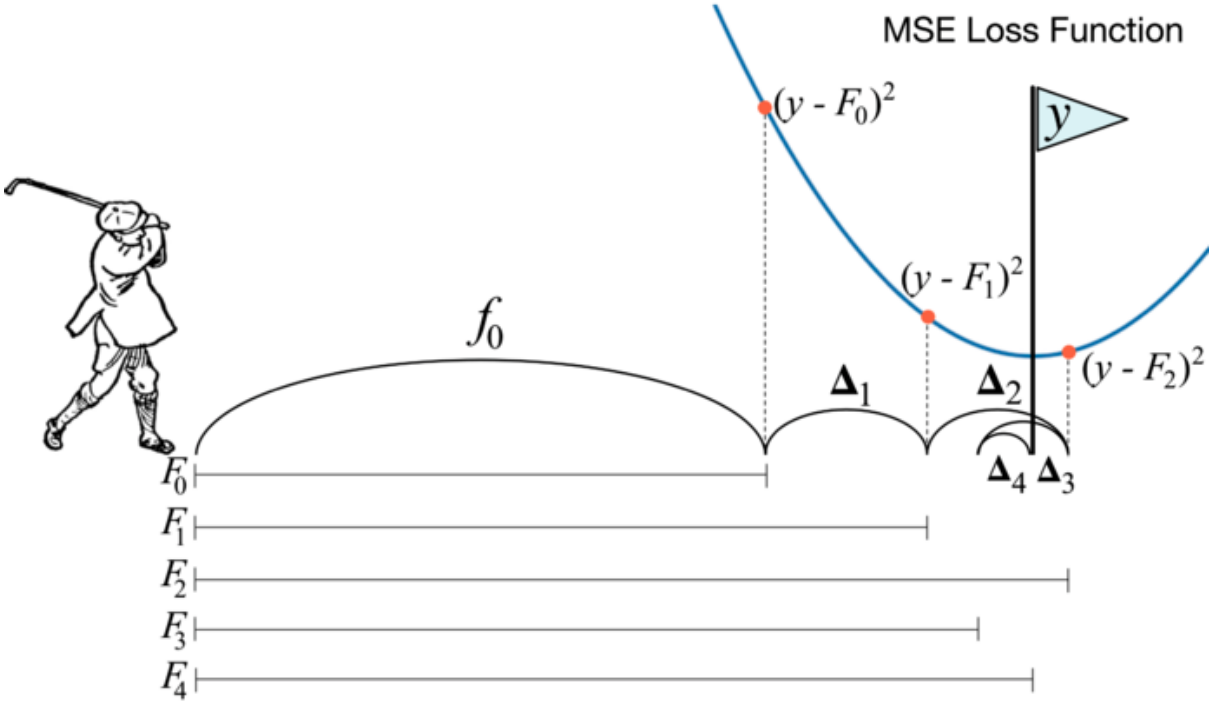


Рисунок 2.6 - Ілюстрація бустингу

2.2 Критерії адекватності моделі

Перед тим як увести метрики оцінки класифікатора введемо спочатку такі поняття, як:

- fp – false positive – це кількість прикладів тестової вибірки, які були хибно розпізнані як позитивні (як такі, у яких діагностовано аритмію)
- fn – false negative – це кількість прикладів з тестової вибірки, які було хибно класифіковано як такі, що не мають аритмії
- tp – true positive – це кількість прикладів з тестової вибірки, у яких було правильно діагностовано наявність аритмії.
- tn – true negative – це кількість прикладів з тестової вибірки, у яких було правильно діагностовано відсутність аритмії.

2.2.1 Точність(Accuracy)

Найпершою і інтуїтивно зрозумілою є метрика accuracy, що за своєю сутністю являє долю правильно класифікованих прикладів. Така метрика рідко використовується, тому що, наприклад, якщо в тестовій вибірці буде 95% позитивних прикладів, то класифікатор, що класифікує усі без винятку приклади як позитивні, отримає метрику адекватності 95%. Але вона використовується як базова для інших метрик.

Формулу accuracy можна записати наступним чином:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.9}$$

де $TP, TN -$ відповідно true positive та true negative.

2.2.2 Точність(Precision)

Для оцінки якості моделі на кожному класі окремо можна увести таку метрику як precision. Вона являє собою долю об’єктів, класифікованих позитивно з усіх позитивно класифікованих. Вона уже буде більш адекватно відображати якість алгоритму, оскільки для попереднього прикладу, для класу якого 5% з усієї тестової вибірки, така метрика отримає результат 5% . Тому в принципі, на неї можна орієнтуватись, якщо вираховувати її окремо для кожного класу. Але зручніше все-таки мати одну метрику, тому на ній теж зупинимось лише як на проміжній.

Формула для вирахування цієї метрики є наступною:

$$\text{precision} = \frac{TP}{TP + FP}, \tag{2.10}$$

де $TP, TN -$ відповідно true positive та true negative.

2.2.3 Чутливість(Recall)

Ще однією покращеною метрикою є метрика recall. Вона являє собою долю правильно позитивно класифікованих прикладів з усіх позитивних прикладів. Така метрика по суті своїй означає властивість алгоритму розпізнавати даний клас взагалі, а метрика precision – властивість відрізняти цей клас від інших класів. Формула метрики recall виглядає наступним чином:

$$\text{recall} = \frac{TP}{TP + FN}, \tag{2.11}$$

де $TP, TN -$ відповідно true positive та true negative

А те, що саме означають ці метрики наглядно проілюстровано на рисунку 2.7.

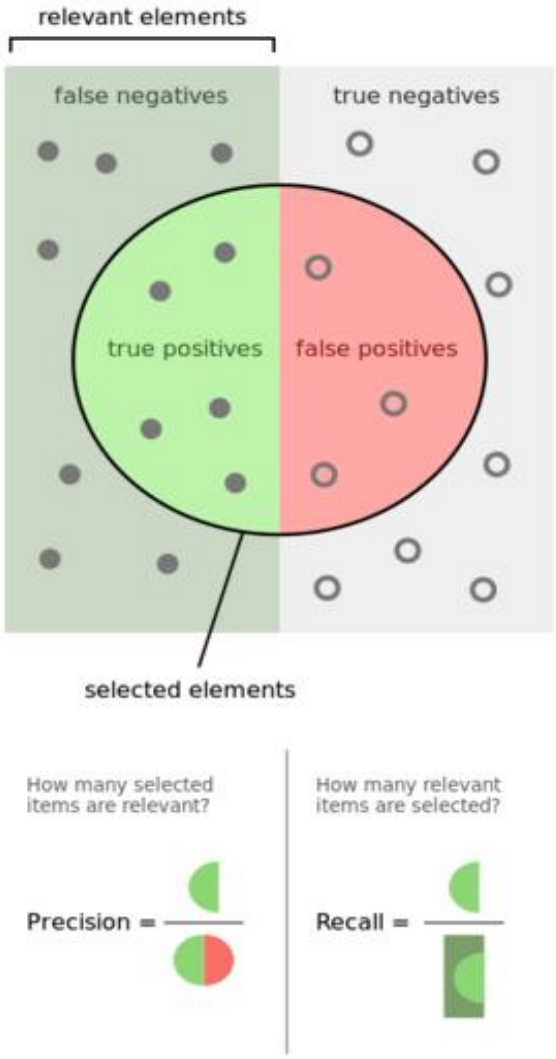


Рисунок 2.7 – Ілюстрація метрик precision та recall

Часто у практичних застосуваннях необхідно ввести певний баланс між цими двома метриками. Таким чином, наступна метрика як раз цей баланс і утримує:

2.2.4 Fβ -метрика

Як і зазначено, ця метрика є спробою привести дві попередні в одну: precision і recall. Вона створена для того, аби не намагатися збалансувати дві різні метрики, а дивитись лише на одне число і намагатися будувати модель, що буде оптимізувати саме цю метрику.

Формула цієї метрики виглядає наступним чином:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}, \quad (2.12)$$

де β в даному випадку визначає вагу точності в метриці.

При $\beta = 1$ це середнє гармонійне (з множником 2, щоб в разі precision = 1 і recall = 1 мати F1=1):

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \quad (2.13)$$

де $F_1 -$ значення метрики F_β при β=1.

F-міра досягає максимуму при повноті і точності, рівними одиниці, і близька до нуля, якщо один з аргументів близький до нуля.

Саме таку метрику і буде використано у подальших дослідженнях.

2.3 Порівняльний аналіз існуючих методів

Отже, проведемо порівняння описаних методів розпізнавання.

Почнемо з того, що порівняємо «прості алгоритми» зі «складними», а саме, порівняємо логістичну регресію з іншими алгоритмами.

Найбільшою перевагою логістичної регресії є її просто у навчанні, інтерпретації результатів та розумінні. Для навчання логістичної регресії необхідно знайти один вектор параметрів, і кожен із цих параметрів буде означати, з яким коефіцієнтом враховується та чи інша координата прикладу при прийнятті рішення класифікувати приклад до того чи іншого класу.

З цією простотою приходить і головний недолік цього алгоритму: модель логістичної регресії дуже погано показує себе на даних, у яких відновлювальна структур не є лінійною за своєю природою. Наприклад, згорткова нейронна мережа покаже себе в рази краще при розпізнаванні зображень, звуків чи іншого неструктурованого сигналу (наприклад сигналу ЕКГ для нашого випадку). Але якщо приклад це набір чітко інтерпретовних та сталих даних (наприклад набір конкретних показників, за якими вже приймають рішення часто люди, а не часовий ряд), логістична регресія може показати себе несуттєво гірше ніж інші складні алгоритми. Тож якщо за своєю природою дані є доволі простими та інтерпретовними, немає сенсу використовувати якісь складні алгоритми.

Тепер коли зрозуміло переваги і недоліки логістичної регресії перед більш складними алгоритмами, спробуємо порівняти між собою два складних алгоритма, а саме нейронну мережу та XGBoost.

Почнімо з того, що ці два алгоритми є дуже різними за своєю сутністю. В той час як XGBoost є градієнтним бустингом, або ансамблем дуже простих моделей, нейронна мережа є складною моделлю зі складеною структурою, яка навчається підбираючи одразу усі свої параметри. Градієнтний бустинг же в цей час підбирає параметри кожної наступної простої моделі і потім враховує їх передбачення у кінцевому результаті.

До чого ж призводять такі принципові відмінності у схемі побудови таких моделей?

Перша спільна деталь яку варт зазначити, що обидві ці моделі доволі непогано покажуть себе на складних даних, у яких відновлювальна структура є нелінійною за своєю суттю, адже ці моделі за своєю природою є нелінійними. У цьому можна переконатися у результатах виконання програми дипломної роботи. Там можна побачити, що хоча якщо навчати модель на попередньо виявлених статистичних характеристиках, адекватності усіх трьох моделей суттєво не відрізняються одна від одної, але якщо навчати моделі на «сирих» сигналах ЕКГ, адекватність логістичної регресії сильно падає, в той час як ці дві моделі тримаються на високому рівні.

Але є і суттєві відмінності. Однією з таких відмінностей є те, що XGBoost не має дуже багато гіперпараметрів. Це означає, що якість побудованої моделі кардинально не зміниться від того, я якими параметрами її навчати. Звісно, якщо поставити неадекватні гіперпараметри, то адекватність моделі сильно впаде, але ось значно покращити ефективність моделі за допомого гіперпараметрів не вийде.

В цей же час, для того щоб зробити хорошу модель і затрачати мінімум зусиль, XGBoost підійде якнайкраще, оскільки з параметрам за замовчуванням уже видасть хороші результати.

Нейромережа в цьому відношенні має суттєво інші властивості. Будуючи нейромережу, розробник взагалі не має деяких гіперпараметрів за замовчуванням, таких як функції активації, кількість прихованих шарів, та розмір кожного прихованого шару і тому подібне. Також існує безліч архітектур нейромереж, як, наприклад, перцептрон (найпростіша), згорткова нейронна мережа, нейронна мережа зі зворотнім зв’язком. Кожну таку архітектуру потрібно застосовувати до конкретної задачі і налаштовувати її. Тобто іншими словами, ту ми не маємо жодної універсальності.

Перевагою яку надає така особливість нейромережі є те, що адекватність такої моделі можна дуже сильно підвищити для кожної конкретної задачі.

Наприклад, якщо задача полягає у розпізнаванні неструктурованих даних (зображення, звуковий сигнал, тощо), грамотно налаштована згорткова нейронна мережа часто покаже себе краще ніж XGBoost.

Отже, якщо підсумувати порівняльний аналіз, можна зазначити, що використання логістичної регресії має сенс на простих, структурованих і зрозумілих людині даних, які є інтерпретовні. Це надасть перевагу у тому, що не потрібно буде витрачати багато часу на її реалізацію, легко буде вчити та інтерпретувати отримані результати. Більш складні алгоритми маж сенс використовувати на більш складних даних. Наприклад, при розпізнаванні аритмії за ЕКГ вони добре підійдуть. Якщо порівнювати XGBoost та нейромережу, головні відмінності у тому, що нейромережа може показати кращі результати, але її налаштування займе багато часу, а XGBoost покаж непогані результати уже з гіперпараметрами за замовчуванням, але і сильно покращити їх у майбутньому часто є доволі складною задачею.

2.4 Алгоритм побудованої системи

Алгоритм побудованої системи зображено на рисунку 2.8.

Рисунок 2.8 - Алгоритм побудованої системи

2.4.1 Якість побудованої моделі

Проблема класичного оцінювання моделі на основі критерію на тестових даних полягає у наступному:

У наших даних є дуже багато прикладі в у результаті нарізки. Але ц приклади не можна просто взяти і випадковим чином розподілити на тренувальну т тестову множину. Через це нам потрібно прослідкувати з тим, щоб тренувальна і тестова множини н включали в себе ЕКГ-уривки одного і того самого пацієнта.

Це покращення на практиці також виявилося недостатнім. Якщо створити тестову та тренувальну множину кільки разів, і навчати алгоритм за однією і тією ж схемою, метрика адекватності моделі може видавати абсолютно різні результати в залежності від того, якого саме пацієнта ми визначимо в тренувальну чи в тестову множину відповідно.

Задля того, аби адекватно оцінити ефективність моделі, була побудована така схема:

- Розподіл усієї множини на тестову та тренувальну множини n = 30 разів
- На кожному з таких поділів навчається модель та визначається метрика адекватності
- Кінцевою метрикою будемо вважати середнє арифметичне з отриманих метрик.

Приклад такого порівняння алгоритмів для n=10 зображено на рисунку 2.9.

	train_recs	test_recs	xgb__prop	log_reg__prop	xgb__rrs	log_reg__rrs	RF__rrs	RF__prop	NN__rrs	NN__prop	ensamble_nn+xgb__prop_or_rrs
0	['5091', '7910', '8455', '4936', '4746', '6426...	['5121', '7879', '4043', '4048', '8434', '6995...	0.929100	0.941405	0.941935	0.534962	0.911803	0.927713	0.803005	0.876401	0.935125
1	['5121', '7910', '8455', '7879', '4908', '5261...	['5091', '4936', '4746', '4043', '3665', '8405...	0.889359	0.881110	0.943169	0.524493	0.919889	0.886735	0.776046	0.915877	0.986866
2	['7910', '7879', '4936', '4746', '4043', '5261...	['5121', '5091', '8455', '4908', '4015', '8405...	0.894559	0.878457	0.895732	0.456580	0.850198	0.879422	0.791574	0.885211	0.974165
3	['5121', '5091', '7879', '4936', '8455', '4746...	['7910', '4908', '4048', '8434', '8215', '8378...	0.862534	0.775445	0.861762	0.416240	0.731151	0.841795	0.680072	0.841696	0.983957
4	['5121', '5091', '7879', '8455', '4936', '4746...	['7910', '4043', '4908', '8434', '8378', '8219']	0.887720	0.826805	0.871089	0.411872	0.741021	0.866393	0.744999	0.846288	0.972079
5	['5121', '8455', '4043', '4908', '5261', '6426...	['5091', '7910', '7879', '4936', '4746', '6453...	0.917916	0.911050	0.950384	0.490829	0.926066	0.915570	0.776524	0.940799	0.994422
6	['5121', '7879', '4936', '4043', '6426', '5261...	['5091', '7910', '8455', '4746', '4908', '8434...	0.873187	0.902626	0.892275	0.434892	0.843647	0.886297	0.758474	0.894867	0.921474
7	['7910', '8455', '4936', '4746', '6426', '4908...	['5121', '5091', '7879', '5261', '4015', '6995...	0.838177	0.901736	0.874595	0.556026	0.830012	0.859429	0.778761	0.901658	0.931543
8	['5121', '5091', '7910', '8455', '4746', '6426...	['7879', '4936', '4043', '4048', '8219']	0.902693	0.878554	0.934834	0.581657	0.901866	0.893264	0.552702	0.883124	0.986991
9	['5121', '5091', '7910', '4936', '4746', '4043...	['8455', '7879', '5261', '6995', '6453', '8215...	0.846753	0.884339	0.867806	0.747779	0.817694	0.861462	0.755367	0.907746	0.932203

Рисунок 2.9 - Статистичне оцінювання алгоритму

2.4.2 Отримання та підготовка даних

При виборі даних було враховано такі особливості, як:

- Легкість отримання
- Відсутність потреб у інвестиціях
- Готова розмітка даних

Отже, логічним рішенням було обрати дані “MIT-BIH Atrial Fibrillation Database”. Ці дані викладені на сайті https://physionet.org/content/afdb/1.0.0/ , вони мають таку ліцензію, що кожен охочий може узяти ці дані та використовувати їх у свої цілях.

Великою перевагою такого рішення є те, що такі дані є вже завчасно розміченими, і є змога не витрачати багато часу на розмітку, вирівнювання та інші підготовчі етапи.

Також у сумі з цих баз даних можна отримати дуже великий інтервал розміченого ЕКГ. Але недоліком є те, що ці ЕКГ записані лише у кількох десятків людей, через це навчання на один людях, та тренування на інших може дати доволі погані і немасштабовані результати. Обходження даної проблеми буде описано пізніше.

Під час розробки моделі було проведено спробу робити нарізання сигналу на підсигнали сталої довжини часового інтервалу. На практиці цей метод виявився не ефективним через те, що такі сигнали не можна ані подати як сирий сигнал на вхід складної моделі (через невизначену кількість піків), ані подати на вхід простої моделі статистичні набори властивостей, оскільки кількість піків невизначена, і від прикладу до прикладу впевненість у статистичній обґрунтованості може варіюватися. Для первинної обробки даних було використано два підходи, які потім поєднуються в ансамблі.

Перший підхід – затрачає найменше зусиль і потім продовжується другим. Зібрані дані уже мають розмітку, на які ділянки сигналу промарковані як з аритмією, а які промарковані здоровими. А також сигнали мають відмічені HRV-піки. Суть підходу у тому, щоб, прибирати неякісні зразки, пропущені піки, та іншого типу зашумлені дані. У результаті, отримуємо часовий ряд сталої довжини, де час – це час HRV-піку, а значення – це висота цього піку. Такі дані можна подати на вхід складним методам машинного навчання.

Другий підхід – це розвиток першого підходу то такого вигляду, аби навіть прості моделі як, наприклад, логістична регресія, могли отримати більш-менш хороші результати. Суть його полягає в тому, щоб за сигналами з

попереднього методу виокремити певний набір показників цього сигналу, і потім для класифікації використовувати уже ці показники. Його перевагою є те, що в класифікації алгоритм уже використовує досвід отриманий медичними працівниками у вигляді отриманих показників, і алгоритму не треба «перевинаходити» ці показники задля класифікації. Недоліком є те, що через те що алгоритм з сирих даних отримує лише визначений набір параметрів, при розпізнаванні можуть не враховуватися показники та прояви, на які люди зазвичай не звертають уваги і не виокремлюють в окремі показники через складність їх природи та взаємозв'язків, але які точно відтворюють причино-наслідковий зв'язок.

В розробленому алгоритмі використовуються обидва ці підходи та отримуються два набори даних, для входу на різні алгоритми, як зображено на рисунку 2.10.

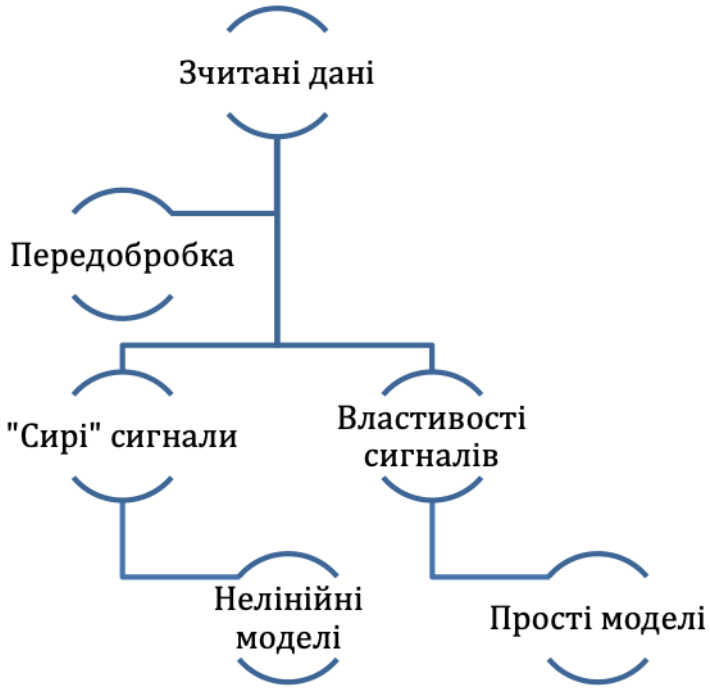


Рисунок 2.10 - Передобробка даних

2.4.3 Побудова моделі

Побудова моделі починається з вибору самої моделі. Процес розробки моделі можна зобразити за допомогою наступної схеми:

Він включає в себе:

- Вибір моделі
- Налаштування її гіперпараметрів
- Валідація результатів моделі
- Статистична оцінка результатів моделі
- Порівняння ефективності моделі з уже існуючими моделями.

Якщо підстави думати, що якась модель отримає кращі результат, є сенс повторити цей самий процес для іншої моделі.

Саме такий процес зображено на рисунку 2.11.

Рисунок 2.11 - Процес обрання моделі

В результаті такого процесу та поєднання різних моделей отримано кінцевий алгоритм з урахуванням перспективності алгоритму. Алгоритм було обрано згідно з таблицею з рисунку 2.12 порівняння розроблених алгоритмів:

	min	max	count	mean
ensamble_nn+xgb__prop_or_rrs	0.921474	0.994422	10.0	0.961883
xgb__rrs	0.861762	0.950384	10.0	0.903358
NN__prop	0.841696	0.940799	10.0	0.889367
xgb__prop	0.838177	0.929100	10.0	0.884200
RF__prop	0.841795	0.927713	10.0	0.881808
log_reg__prop	0.775445	0.941405	10.0	0.878153
RF__rrs	0.731151	0.926066	10.0	0.847335
NN__rrs	0.552702	0.803005	10.0	0.741752
log_reg__rrs	0.411872	0.747779	10.0	0.515533

Рисунок 2.12 - Порівняння моделей

Перший результат – ансамбль нейронної мережі на XGBoost – обрано як найперспективніший у подальшій розробці, Він же і має найбільш цікавий алгоритм прийому та обробки даних задля розпізнавання.

можна представити наступним чином:

- Виконання «xgboost» на сирих даних
- Виконання розпізнавання за допомогою нейромережі на статистичних даних
- Подання на кінцевий ансамблю статистичних даних, а також результати розпізнавання нейронною мережею та XGboost
- Отримання кінцевих результатів розпізнавання.

Алгоритм розробки такого алгоритму буде описаний у окремому пункті.

Схему цього алгоритму зображено на рисунку 2.13.

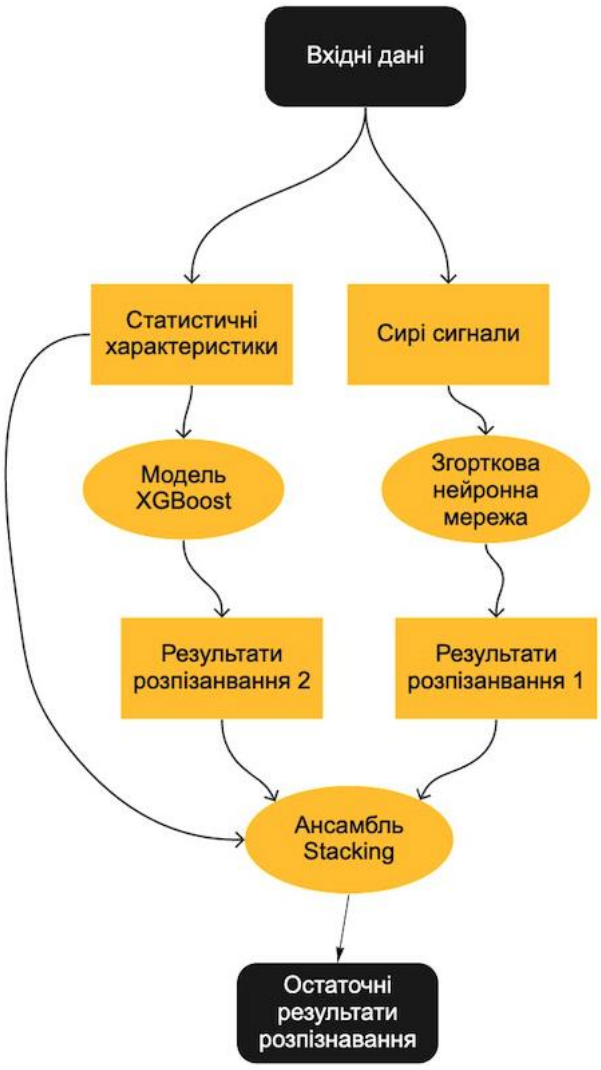


Рисунок 2.13 - Схема алгоритму розпізнавання

2.4.4 Післяобробка результатів

Післяобробка результатів є скоріше етапом розробки алгоритму розпізнавання ніж етапом самого алгоритму розпізнавання.

Отже розроблений алгоритм після обробки включає в себе:

- Аналіз матриці похибок та звіту з класифікації розробленого алгоритму, що зображено на рисунку 2.14.

	precision	recall	f1-score	support
0	0.93	0.97	0.95	7733
1	0.94	0.87	0.91	4352
accuracy			0.94	12085
macro avg	0.94	0.92	0.93	12085
weighted avg	0.94	0.94	0.93	12085

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x11b1431c0>
```

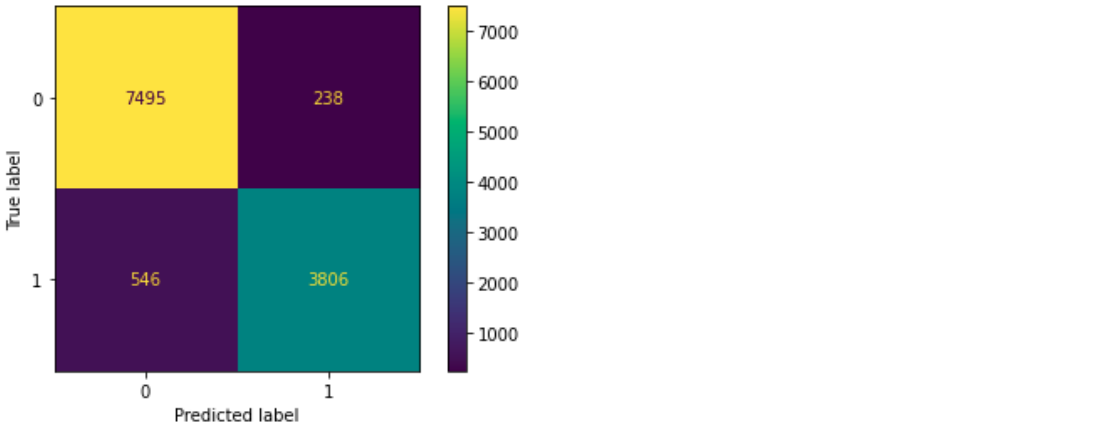


Рисунок 2.14 - Перший етап післяобробки результатів

Аналіз ранжування важливостей характеристик (для алгоритму, для якого це доступно), що зображено на рисунку 2.15.

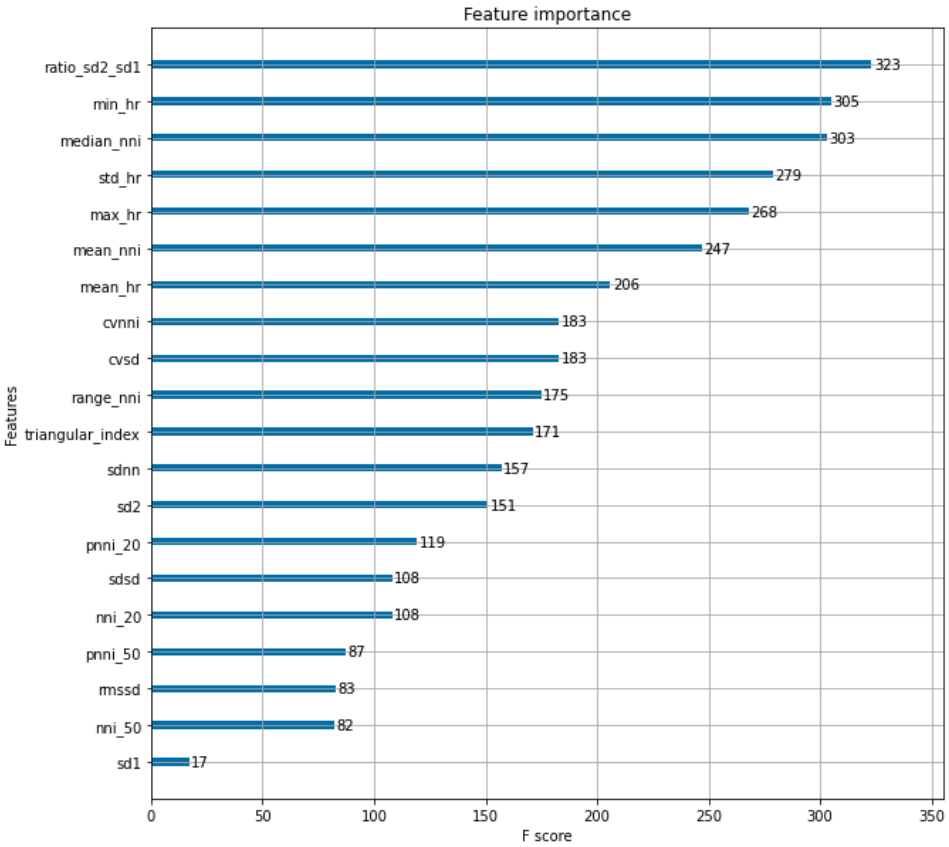


Рисунок 2.15 - розподіл важливостей характеристик

Аналіз розподілу помилок, та впевненості моделі їх прийняття, що зображено на рисунку 2.16.

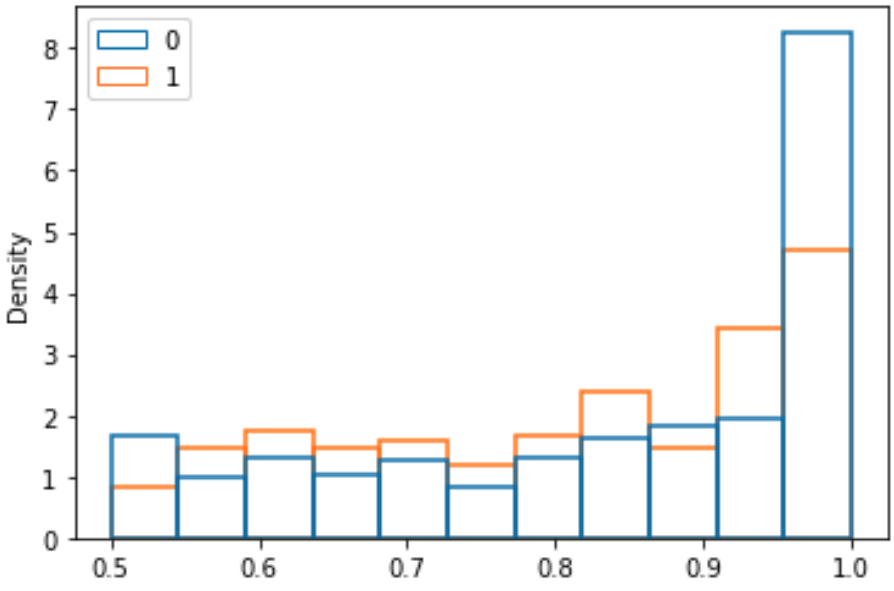


Рисунок 2.16 - Аналіз розподілу впевненості прийняття помилок

Аналіз розподілу метрики оцінку адекватності алгоритму в залежності від обраного поділу на тестовий та тренувальний приклад, що зображено на рисунку 2.17.

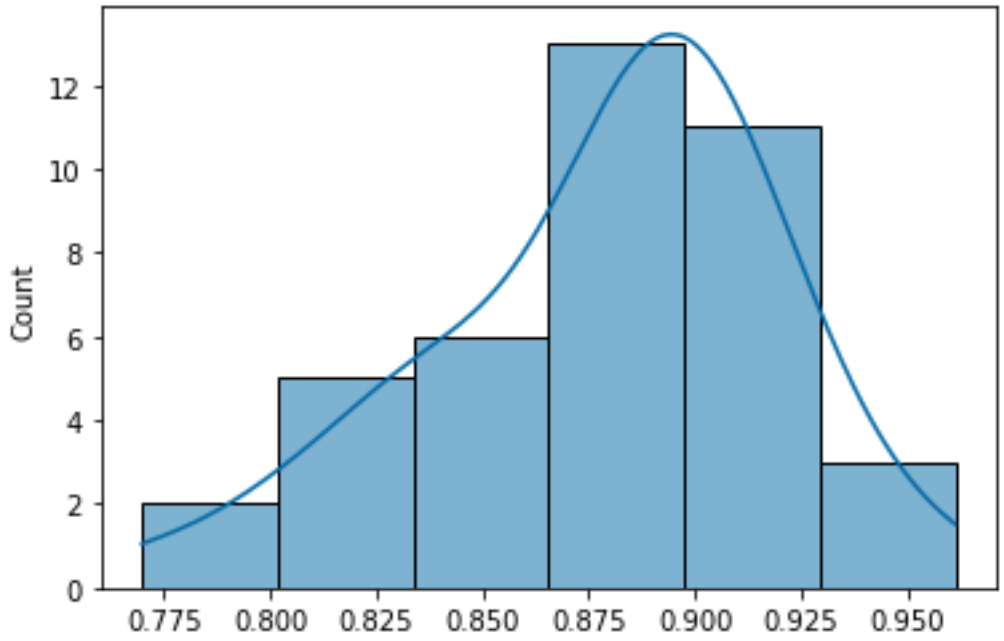


Рисунок 2.17 - Розподіл метрики алгоритму

2.5 Висновки за розділом

Згідно з цим розділом було проведено ознайомлення з основними використовуваними методами розпізнавання: логістична регресія, XGBoost, згорткова нейронна мережа. Також розглянуто алгоритм ансамблювання моделей машинного навчання, що є перспективним з точки зору одночасного врахування різної природи різних підходів передобробки вхідних даних та різних алгоритмів.

При детальному вивченні існуючих алгоритмів встановлення серцевої аритмії було встановлено, які саме методи є оптимальними для розв’язку поставленої задачі. Сформульовано основні етапи розробки моделей та розпізнавання серцевої аритмії.

Важливим результатом є розроблений алгоритм статистичного порівняння моделей між собою, що. Необхідним наслідком обраних даних, Справа в тому, що при виборі даних, обравши найдоступніший варіант, можна наштовхнутися на супроводжуючі проблеми, такі як невелика кількість пацієнтів, для яких доводиться розробляти нові методи оцінки адекватності моделі виходячи з фізичного змісту задачі.

Також було розроблено алгоритм, за яким враховано особливості даних, що можна отримати як із сирих так і з оброблених вхідних даних. Такий алгоритм має найперспективніші можливості розпізнавання.

В результаті детального вивчення етапу передобробки даних можливим став розвиток та покращення моделей, обґрунтований вибір тих чи інших моделей та інтерпретації їх поведінки.

У сумі з теоретичним обґрунтуванням це може дати потужний алгоритм розпізнавання серцевої аритмії. Такий алгоритм можна розвивати далі та обґрунтовувати його результати за допомогою післяобробки та візуалізації його результатів.

РОЗДІЛ 3 ОЦІНЮВАННЯ ЯКОСТІ РОБОТИ КЛАСИФІКАТОРІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОЗПІЗНАВАННЯ

3.1 Попередня робота з даними та аналіз вхідних даних

Як уже було зазначено раніше, дані було узято з відкритого джерела. У даних є 24 довгих електрокардіограми, на кожній з яких позначено проміжки, що відповідають аритмії, та проміжки, що відповідають нормальній роботі серця.

Кожна електрокардіограма представлена самим сигналом, а у сигналі позначені його R-піки, як зображено на рисунку 3.1.

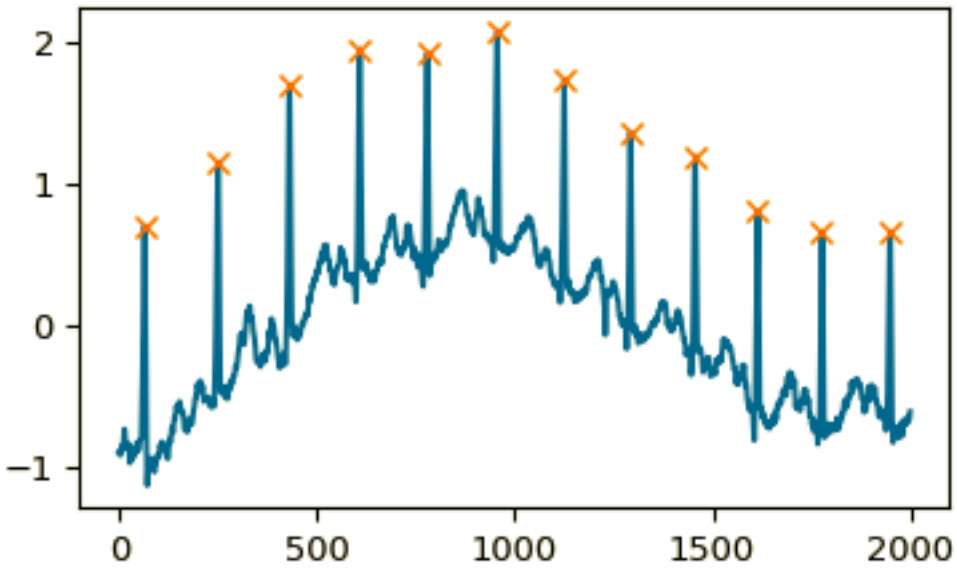


Рисунок 3.1 - Графік сигналу ЕКГ та його піки

Оскільки більшість інформації можна отримати лише з інформації про послідовності піків, то у роботі будуть використовуватись лише вони.

Кожну кардіограму поділено на відрізки по 20 піків. Такі відрізки і називаються прикладами, і кожен з них буде промарковано на наявність аритмії відповідно до маркування з вхідних даних.

Вхідні дані буде перетворено на два набори даних:

- Сирі дані (тільки R-піки), що зображено на рисунку 3.2.

record	0	1	2	3	4	5	6	7	8	...	21	22	23	24	25	26	27	28	label	beginning
00735	269.0	279.0	279.0	268.0	269.0	277.0	267.0	273.0	289.0	...	291.0	277.0	288.0	285.0	281.0	288.0	277.0	265.0	0	448.0
00735	280.0	268.0	268.0	302.0	288.0	272.0	278.0	275.0	278.0	...	281.0	295.0	280.0	289.0	283.0	276.0	282.0	292.0	0	33944.0
00735	278.0	278.0	277.0	292.0	299.0	275.0	294.0	286.0	272.0	...	273.0	283.0	292.0	278.0	291.0	285.0	272.0	276.0	0	67700.0
00735	268.0	277.0	291.0	278.0	282.0	285.0	277.0	279.0	289.0	...	286.0	272.0	283.0	281.0	271.0	282.0	293.0	280.0	0	101488.0
00735	280.0	270.0	277.0	285.0	273.0	283.0	289.0	284.0	274.0	...	270.0	288.0	288.0	272.0	271.0	270.0	266.0	267.0	0	135044.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
08455	190.0	192.0	190.0	187.0	188.0	189.0	188.0	187.0	188.0	...	188.0	190.0	188.0	187.0	186.0	188.0	189.0	188.0	0	11351768.0
08455	188.0	190.0	187.0	187.0	189.0	189.0	187.0	187.0	186.0	...	187.0	187.0	189.0	191.0	186.0	188.0	187.0	189.0	0	11374308.0
08455	166.0	187.0	202.0	197.0	141.0	224.0	125.0	174.0	143.0	...	103.0	222.0	123.0	187.0	118.0	94.0	90.0	52.0	1	11233820.0
08455	121.0	89.0	109.0	90.0	237.0	120.0	89.0	113.0	155.0	...	90.0	112.0	111.0	100.0	130.0	123.0	121.0	106.0	1	11251108.0
08455	98.0	118.0	120.0	117.0	122.0	131.0	123.0	123.0	207.0	...	276.0	125.0	90.0	110.0	110.0	93.0	117.0	138.0	1	11266512.0

Рисунок 3.2 - Вхідні дані у вигляді сирих R-піків

- Характеристики сигналу, що зображено на рисунку 3.3.

	0	1	2	3	4	5	6	7	8	9	...	502	503
record	00735	00735	00735	00735	00735	00735	00735	00735	00735	00735	...	08455	08455
mean_nni	1116.97	1127.72	1126.48	1117.93	1107.03	1127.86	1137.52	1130.62	1136.97	1131.17	...	766.897	751.448
sdnn	35.8673	37.2011	30.8436	24.8566	29.4066	29.866	37.9234	28.4108	33.5628	30.4139	...	63.362	8.66679
sdsd	46.8153	47.7664	41.7739	35.0743	38.0148	39.3132	50.8443	41.119	46.6161	43.7619	...	54.5159	11.2295
nni_50	8	8	8	5	6	5	10	7	8	10	...	4	0
pnni_50	27.5862	27.5862	27.5862	17.2414	20.6897	17.2414	34.4828	24.1379	27.5862	34.4828	...	13.7931	0
nni_20	23	20	18	17	16	17	20	19	21	18	...	4	1
pnni_20	79.3103	68.9655	62.069	58.6207	55.1724	58.6207	68.9655	65.5172	72.4138	62.069	...	13.7931	3.44828
rmssd	46.8188	47.7972	41.7749	35.1161	38.0601	39.3156	50.8443	41.1548	46.6721	43.7656	...	54.5161	11.2377
median_nni	1116	1128	1112	1116	1108	1128	1144	1132	1136	1128	...	748	752
range_nni	132	164	108	100	104	132	144	104	128	124	...	300	32
cvsd	0.0419161	0.0423838	0.0370844	0.0314117	0.0343802	0.0348585	0.0446976	0.0364001	0.0410497	0.0386905	...	0.0710866	0.0149547
cvnni	0.0321114	0.0329878	0.0273805	0.0222345	0.0265634	0.0264802	0.0333388	0.0251285	0.0295196	0.0268871	...	0.0826213	0.0115335
mean_hr	53.7703	53.2606	53.3011	53.6961	54.2357	53.2339	52.8033	53.1003	52.8165	53.0788	...	78.6529	79.8561
max_hr	56.6038	57.4713	55.1471	55.9701	56.8182	56.391	56.391	54.9451	55.7621	55.3506	...	81.9672	81.9672
min_hr	50.3356	49.6689	50.1672	51.1945	51.7241	50.1672	49.6689	50.1672	49.8339	49.6689	...	58.1395	78.534
std_hr	1.69185	1.73188	1.41209	1.16657	1.4103	1.38007	1.73516	1.29878	1.53433	1.3802	...	5.13535	0.909318
triangular_index	7.25	9.66667	4.14286	5.8	7.25	4.14286	5.8	5.8	7.25	5.8	...	2.9	2.9
tinn	0	0	0	0	0	0	0	0	0	0	...	0	0
sd1	33.7109	34.3958	30.0807	25.2564	27.3738	28.3088	36.6121	29.6091	33.5674	31.5122	...	39.2559	8.08618
sd2	37.9012	39.8092	31.5882	24.4504	31.3077	31.346	39.1909	27.1596	33.5582	29.2745	...	80.5509	9.21088
ratio_sd2_sd1	1.1243	1.15739	1.05012	0.968088	1.14371	1.10729	1.07044	0.917274	0.999725	0.928989	...	2.05194	1.13909
beginning	448	33944	67700	101488	135044	168284	202116	236212	270108	304244	...	1.11817e+07	1.12042e+07
label	0	0	0	0	0	0	0	0	0	0	...	0	0

Рисунок 3.3 - Сирі дані у вигляді характеристик сигналу

Усі сигнали розділяються на сигнали довжиною по 20 піків.

3.2 Проблема оцінювання результатів

Основна проблема оцінювання результатів моделі полягає у тому, що всього є 24 пацієнти, і від того, які пацієнти потраплять до тренувального або тестового набору метрика адекватності моделі може сильно мінятися. Тому для оцінки адекватності моделі було проведено 10 поділів та навчено 10 відповідних моделей, а за адекватність було узято середню метрику по поділам.

Також було проаналізовано, які саме пацієнти погано впливають на результати моделі використовуючи 1vs all технікою. Техніка полягає у тому, що по черзі навчити модель на всіх пацієнтах без одного, а перевіряти модель на тому одному пацієнтові. Отримані результати зображено на рисунку 3.4.

	test_record	xgb__prop	xgb__rrs
0	5121	0.930211	0.942423
1	8455	0.800000	0.857143
2	4746	0.980178	0.990177
3	5261	0.318182	0.409091
4	8215	0.000000	0.000000
5	6426	0.990947	0.992605
6	8219	0.757447	0.812670
7	4126	0.721088	0.899160
8	4908	0.879668	0.997658
9	4015	0.250000	0.206897
10	6453	0.952381	0.777778
11	8378	0.876836	0.906926
12	7879	0.997748	0.998501
13	4936	0.845983	0.947368
14	4043	0.940803	0.924037
15	4048	1.000000	0.978723
16	3665	0.888889	0.972892
17	6995	0.624444	0.637681
18	8405	0.333333	0.200000
19	735	1.000000	1.000000
20	5091	0.500000	0.571429
21	7910	0.864322	0.883495
22	8434	0.924051	0.961538

Рисунок 3.4 - Виявлення пацієнтів що ускладнюють навчання моделі

Як бачимо на цьому прикладі, існують окремі пацієнти, для яких моделі погано розпізнають аритмію.

3.3 Порівняння помилок в залежності від типу вхідних даних

Як зазначалося раніше, в залежності від типу вхідних даних моделі дають різні помилки.

У результаті, різні моделі видають різні результати. Наприклад, згорткова нейронна мережа краще працює з властивостями сигналів, а модель XGBoost краще працює з сирими сигналами.

Отже було вирішено побудувати ще одну модель: Ансамбль, який поєднав би результати нейромережі з властивостей сигналів та XGBoost з сирих сигналів.

3.4 Налаштування моделі

Для налаштування гіперпараметрів моделі було використано два методи:

- Ручний ітеративний
- Автоналаштування за допомогою бібліотеки hyperopt

Ручний ітеративний метод було використано для створення архітектури згорткової нейромережі, а автоналаштування для налаштування параметрів моделі XGBoost. Виявилось, що в нашому випадку автоналаштування XGBoost не покращило результат, а ось ручне налаштування згорткової нейромережі підняло його сильно.

3.5 Аналіз помилок моделі

Подивимось, головні три моделі: нейромережа на властивостях, XGBoost на сирих даних, та їх ансамбль. Як між собою пов’язані помилки, що роблять моделі зображено на рисунку 3.5.

			count
label	**ens**	**xgb**	
0	**0**	**0**	4082
		1	5
	1	**0**	6
		1	17
1	**0**	**0**	153
		1	302
	1	**0**	91
		1	3105

Рисунок 3.5 - Аналіз помилок моделей

Бачимо, що найбільше помилок ансамбль припустив повторюючи їх за моделлю XGBoost. Але також варто відзначити, що багато помилок XGBoost ансамбль «знешкодив» та виправив. Отже можна зробити висновок що покращення цієї моделі покращить роботу ансамблю в цілому.

3.6 Результати роботи

В результаті роботи було побудовано моделі для розпізнавання серцевої аритмії. А також порівняні результати роботи цих моделей статистичним методом. Також було побудовано довірчі інтервали для оцінок якостей моделей. Отримано результати, що зображено на рисунку 3.6.

	min	max	count	mean	std	up	down
neg	3975.000000	6682.000000	10.0	4871.400000	787.577608	5357.055120	4385.744880
pos	866.000000	4241.000000	10.0	3053.800000	937.379539	3631.829603	2475.770397
xgb_rrs	0.660790	0.962429	10.0	0.907861	0.088164	0.962227	0.853495
NN_prop	0.702058	0.949587	10.0	0.896954	0.071229	0.940877	0.853031
log_reg_prop	0.739489	0.944729	10.0	0.894583	0.056904	0.929673	0.859493
RF_prop	0.662548	0.943568	10.0	0.891900	0.082668	0.942877	0.840923
xgb_prop	0.636364	0.942002	10.0	0.887155	0.091167	0.943373	0.830937
RF_rrs	0.555878	0.940144	10.0	0.865987	0.112153	0.935146	0.796829
NN_rrs	0.000000	0.844043	10.0	0.668224	0.258204	0.827444	0.509004
ratio	0.129602	1.066918	10.0	0.662232	0.258393	0.821569	0.502896
log_reg_rrs	0.314259	0.698066	10.0	0.501889	0.129718	0.581879	0.421899

Рисунок 3.6 - Порівняння адекватностей моделей без ансамблю

Можемо побачити, що найкраще себе показала модель XGBoost, потім нейромережа. Логістична регресія на вхідних даних «властивості» впоралася із задачею на достатньому рівні, але гірше за запропоновані моделі. Бачимо що на сирих даних логістична регресія впоралася із задачею вкрай погано, тим часом нелінійні моделі показали кращий результат. Це можна пояснити тим, що нелінійні моделі змогли знайти властивості та взаємозв’язки, які не враховуються при діставанні параметрів. Для того щоб поєднати результати двох найпотужніших моделей, було вирішено отримати переваги обох моделей. Отримані результати з урахуванням ансамблю зображено на рисунку 3.7.

	min	max	count	mean	std	up	down
neg	3975.000000	6682.000000	10.0	4871.400000	787.577608	5357.055120	4385.744880
pos	866.000000	4241.000000	10.0	3053.800000	937.379539	3631.829603	2475.770397
xgb__rrs	0.660790	0.962429	10.0	0.907861	0.088164	0.962227	0.853495
NN__prop	0.702058	0.949587	10.0	0.896954	0.071229	0.940877	0.853031
ensamble_nn+xgb__prop_or_rrs	0.662777	0.949808	10.0	0.895546	0.085300	0.948146	0.842946
log_reg__prop	0.739489	0.944729	10.0	0.894583	0.056904	0.929673	0.859493
RF__prop	0.662548	0.943568	10.0	0.891900	0.082668	0.942877	0.840923
xgb__prop	0.636364	0.942002	10.0	0.887155	0.091167	0.943373	0.830937
RF__rrs	0.555878	0.940144	10.0	0.865987	0.112153	0.935146	0.796829
NN__rrs	0.000000	0.844043	10.0	0.668224	0.258204	0.827444	0.509004
ratio	0.129602	1.066918	10.0	0.662232	0.258393	0.821569	0.502896
log_reg__rrs	0.314259	0.698066	10.0	0.501889	0.129718	0.581879	0.421899

Рисунок 3.7 - Порівняння адекватностей моделей з урахуванням ансамблю

Бачимо, що в даному випадку ансамбль не покращив результати моделей. Але оскільки він приймає на вхід результати двох найкращих моделей, можна припустити, що більш детальне налаштування ансамблю покращить результат.

3.7 Висновки за розділом

У цьому розділі ми побачили, що правильно налаштовані моделі суттєво покращують адекватність у порівнянні зі звичайною логістичною регресією.

Також стали видно, що більш детально налаштований ансамбль є перспективною технологією яка може потенційно видавати покращений результат.

Було проведено аналіз помилок та встановлено, що у більшості своїй помилки ансамблю є спровокованими помилками моделей, на яких побудовано ансамбль.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для вирішення задач розпізнавання серцевої аритмії за допомогою методів машинного навчання.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи задач розпізнавання серцевої аритмії за допомогою методів машинного навчання. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по серцевій аритмії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп’ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу задач розпізнавання серцевої аритмії та будує його модель. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

- Python
- C++

Функція F_2:

- Tensorflow;
- Caffe

Функція F_3:

- Jupyter Notebook;
- Visual Studio.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

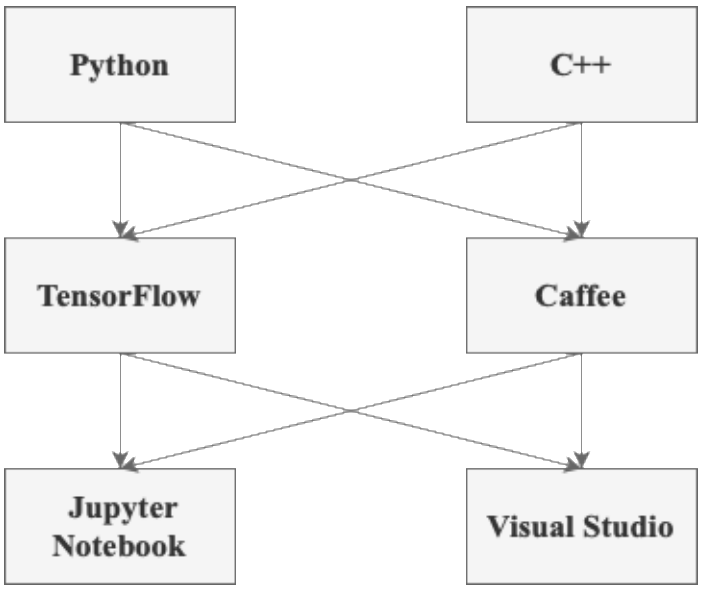


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Швидка розробка програми, доступність бібліотек, кросплатформеність	Низька швидкість роботи, особливо, якщо потрібно обробляти велику кількість даних
	Б	Код швидко виконується	Іде багато часу на розробку програми
F_2	*А*	Надійно працює з складними проектами	Не підтримується багатьма мовами
	Б	Надійність	Додатковий час на інсталяцію та вивчення
F_3	*А*	Підтримується багатьма мовами програмування,	Відсутня можливість роботи без інтернету

		легко запускається на будь-якому сервері	
	Б	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування

На основі цієї карти будуємо позитивно-негативну матрицю варіантів основних функцій (Таб.4.1). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1*:*

Перевагу віддаємо швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2*:*

Обидва варіанти можна використовувати в розробці.

Функція F_3*:*

Віддаємо перевагу варіанту А в разі вибору мови програмування Python.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

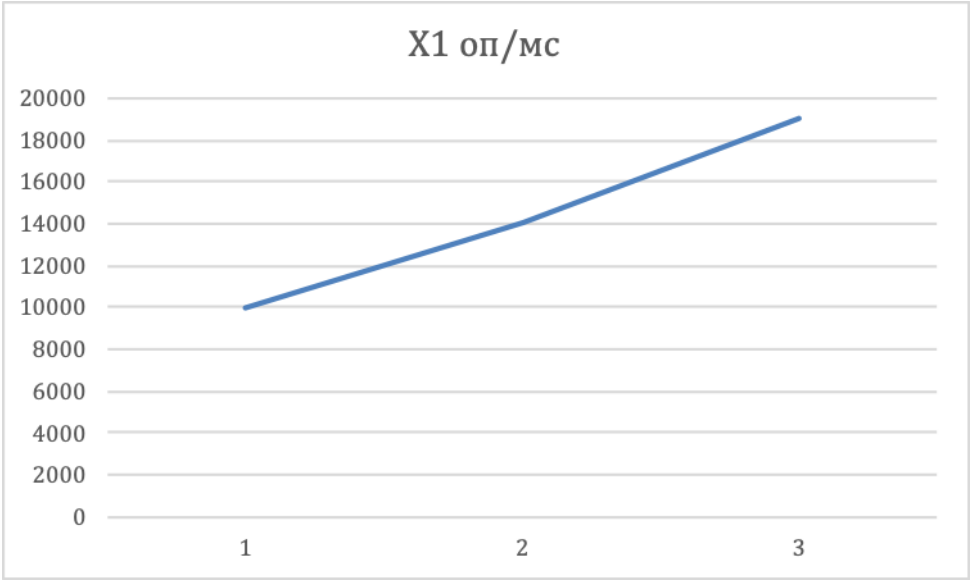


Рисунок 4.2 – X1, швидкодія мови програмування

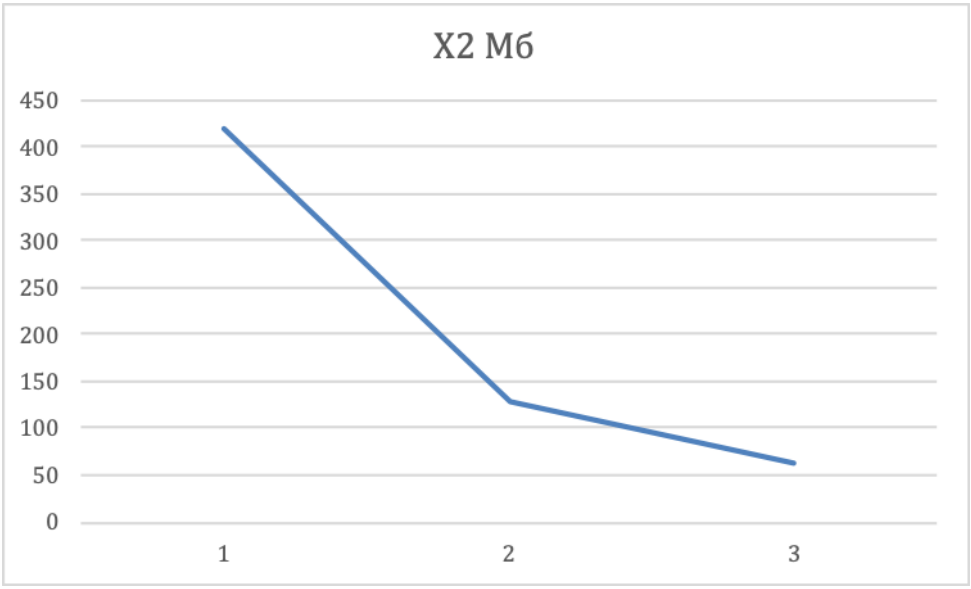


Рисунок 4.3 – X2, об’єм пам’яті

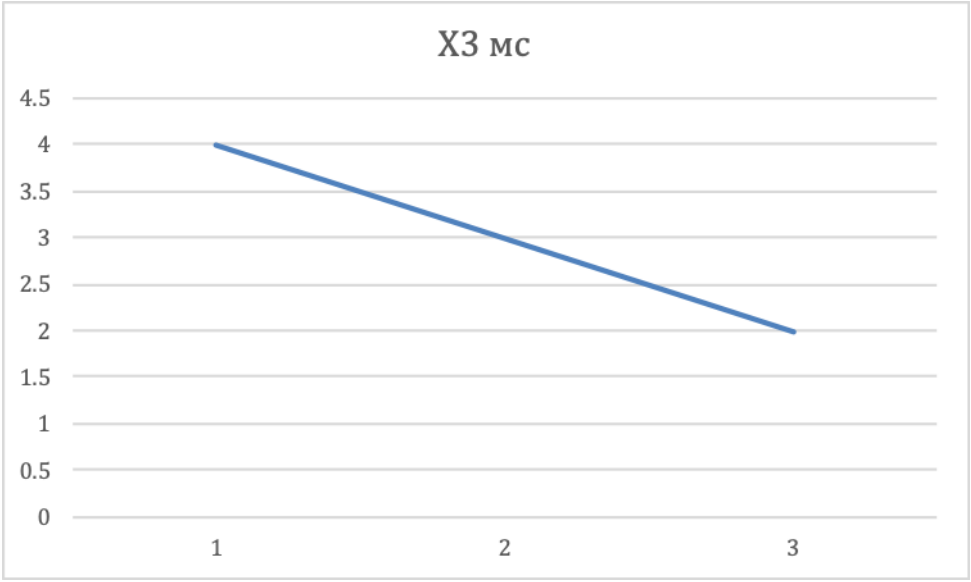


Рисунок 4.4 – X3, час попередньої обробки даних

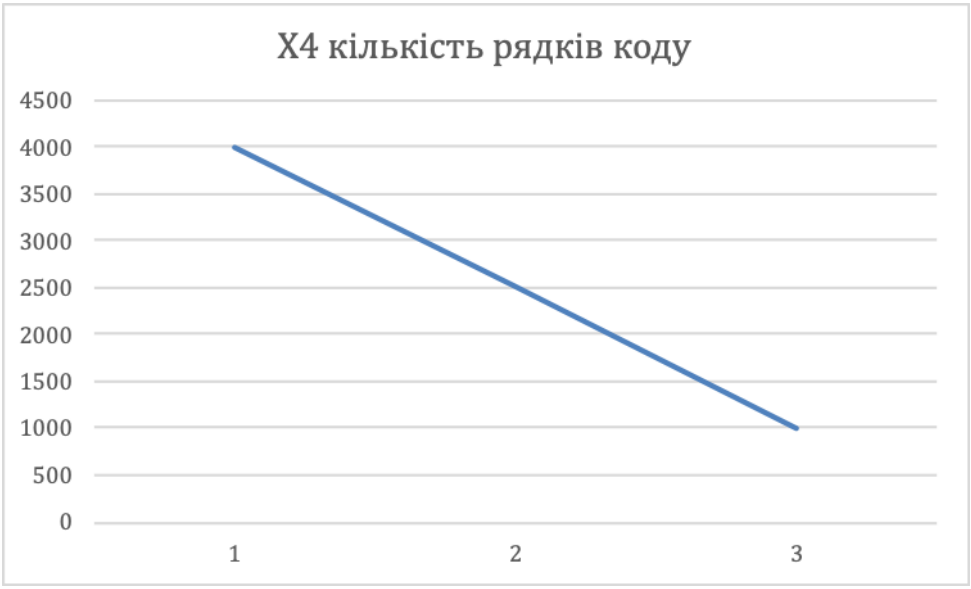


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об’єм пам’яті	Mb	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої	мс	5	3	5	5	4	5	3	30	5,5	30,25

	обробки даних											
X4	Потенційний об’єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24{,}5 \qquad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \qquad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211. \qquad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 211}{7^2(4^3 - 4)} = 0{,}86 > W_k = 0{,}67. \qquad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	>	>	>	>	1,5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	>	<	=	<	=	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	>	<	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \qquad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \tag{4.7}$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

.Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32
X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	10000	6	0,3	1,8
F2	А	X2	64	5	0,21	1,05
	Б	X2	128	2	0,22	0,44
F3	А	X3	1000	8	0,23	1,84

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}8 + 1{,}05 + 1{,}84 = 4{,}69,$$

$$K_{K2} = 1{,}8 + 0{,}44 + 1{,}84 = 4{,}08.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання:

$К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$Т_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64 \text{ людино-годин.}$$

$$Т_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 12000 грн., один аналітик в області даних з окладом 14500. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.}, \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{12000 + 12000 + 14500}{3 \cdot 21 \cdot 8} = 72{,}32 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 72.32 \cdot 1328.64 \cdot 1.2 = 115306{,}97$ грн.

II. $С_{ЗП} = 72.32 \cdot 1345.52 \cdot 1.2 = 116771{,}91$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 115306{,}97 \cdot 0.22 = 25367{,}53$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 116771{,}91 \cdot 0.22 = 25689{,}82$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 10000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 10000 \cdot 0,2 = 24000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 24000 \cdot (1 + 0.2) = 28800 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 28800 \cdot 0,22 = 6336 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 20000 = 5750 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 20000 \cdot 0.05 = 1150 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677.6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} =1677,6 \cdot 0,3 \cdot 0,8 \cdot 3,51 = 1413,95 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 20000 \cdot 0,67 =13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$C_{ЕКС}$ = 28800 + 6336 + 10589,76 + 5750 + 1150 + 1413,95 + 13400= 67439,71 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ} = 67439,71 / 1677.6 = 40,44 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} \cdot T, \quad (4.18)$$

I. $C_М = 40,44 \cdot 1328,64 = 53731,76$ грн.

II. $C_М = 40,44 \cdot 1345,52 = 54412,83$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $C_Н = 102811,43 \cdot 0,67 = 68883,66$ грн.

II. $C_Н = 104117,62 \cdot 0,67 = 69758,80$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н, \quad (4.20)$$

I. $С_{ПП}$ = 102811,43 + 22618,51 + 453731,76 + 68883,66 = 248045,36 грн.

II. $С_{ПП}$ = 104117,62 + 22905,88 + 54412,83 + 69758,80 = 251195,13 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР\,j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 4{,}25/\ 248045{,}36 = 1{,}71 \cdot 10^{-5},$$

$$К_{ТЕР2} = 3{,}74/\ 251195{,}13 = 1{,}48 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1{,}71 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1{,}711 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- Використання моделей з великою ємністю
- Використання стандартного інтерфейсу візуалізації, швидкість розробки

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

4.8 Висновки до розділу 4

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВОК

За переддипломну практику вдалося досягнути дуже значних результатів у дослідженні поставленої задачі.

По-перше6 дослідження існуючих статей та літератури допоміг переконатися у тому, що поставлена задача дійсно є актуальною на сьогодні та її розв’язання допоможе вирішити ряд гострих проблем, таких як складність довготривалого вимірювання ЕКГ, затрати часу лікарів на його аналіз (великих сигналів). Було з’ясовано, що розробка алгоритму розпізнавання аритмії за сигналом ЕКГ допомогла б врятувати багато життів.

Було проведено дослідження існуючих методів, їх порівняльний аналіз. Було виявлено сильні та слабкі сторону кожного підходу.

Також було встановлено, які частини задачі потребують розробки власного підходу, та побудовано архітектуру алгоритму враховуючи порівняльних аналіз різних алгоритмів розпізнавання. Архітектуру алгоритму було розроблено таким чином, щоб використати по максимуму сильні сторони кожного алгоритму та нівелювати . Як приклад задачі для якої необхідно реалізовувати власне рішення, постає задача валідації моделі на тестових даних. Через невелику кількість пацієнтів, оцінювати модель доводиться статистично проводячи багато поділів на тренувальну та тестову множину.

Було також встановлено метод оцінювання адекватності моделі: за допомогою метрики f1-score, що має відображати якість алгоритму розпізнавання відповідно до потреб поставленої задачі.

В ході переддипломної практики було сформульовано та розроблено план післяобробки та візуалізації результатів роботи моделі, що є вкрай важливою складової розробки моделі та вибору її архітектури.

ЛІТЕРАТУРА

1. Medical error—the third leading cause of death in the US.URL: https://www.bmj.com/content/353/bmj.i2139 (дата звернення 14.04.20).
2. Medical diagnosis.URL:https://ru.qwe.wiki/wiki/Medical_diagnosis (дата зверення 14.04.20).
3. Прогноз хвороби.URL:https://bigenc.ru/medicine/text/3178819 (дата звернення 14.04.20).
4. Искусственный интеллект в медицине:как машинное обучение ставит диагнозы.URL : https://aiconference .com.ua/ru/news/iskusstven niy-intellek t-v-meditsine-kak-mashinnoe-obuchenie-stavit-diagnozi-97639 (дата звернення 20.04.20).
5. Штучний інтелект у медицині: комп'ютер знає, що з вами не так.URL : https://www.dw.com/uk/ (дата звернення 20.04.20).
6. Процесс Data Mining. Начальные этапы.URL : https://www.intuit.ru/studies /courses/6/6/lecture/192?3 (дата звернення 27.04.20).
7. Методи предварительной обработки данных.URL : http://masters.donntu.org/2013/fknt/pettse/library/article2.htm (дата звернення 27.04.20).
8. Матеріали з Вікіпедії.Машинное обучение.URL: https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5 (дата звернення 28.04.20).
9. Classification problem.URL:https:// wiki.loginom.ru/article s/classificat ion-problem.html (дата звернення 04.05.20).
10. Логистическая регрессия и ROC-анализ — математический аппарат. URL:https://loginom.ru/blog/logistic-regression-roc-auc (дата звернення 4.05.20).

11. Машинное обучение (курс лекций, К.В.Воронцов) URL: https://bit.ly/1bCmE3Z (дата звернення 04.05.20).
12. Метод ближайших соседей URL:http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B1%D0%BB%D0%B8%D0%B6%D0%B0%D0%B9%D1%88%D0%B8%D1%85_%D1%81%D0%BE%D1%81%D0%B5%D0%B4%D0%B5%D0%B9 (дата звернення 10.05.20).
13. Метрики в задачах машинного обучения URL:https://habr.com/ru/company/ods/blog/328372/ (дата звернення 10.05.20).
14. Stochastic Gradient Boosting.URL : https://machinelearningmastery.com/start-here/#xgboost (дата звернення 14.05.20).
15. What is TensorFlow? Introduction, Architecture & Example.URL: https://www.guru99.com/what-is-tensorflow.html (дата звернення 14.05.20).
16. Scikit-learn.Classification.URL:https://scikit-learn.org/stable/ (дата звернення 17.05.20).
17. Дж.Вандер Плас.Python для сложный задач.Наука о данных и машинное обучение.Санкт-Петербург:Питер, 2018. 572 с.
18. Мюллер А.,Гвидо С. Введение в машинное обучение с помощью Python.Москва:Вильямс, 2017. 393 с.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Підготовка даних:

```
#!/usr/bin/env python
# coding: utf-8

# Вийшло навчити перший класифікатор.
# Попередні результати:
# - Наївний підхід (поділяємо трейн-тест з усіх прикладів незалежно від людей):
# середій f1: 0.922
# - Більш суверий поділ трейн-тест (одна людина або в трейні, або в тесті):
# середній f1: 0.85
#
# Чому брав середній f1:
# Перформанс на тестовій вибірці дуже сильно залежав від того, як саме поділяться люди на трейн-тест(просто від випадку до випадку),
# тому я будував по 30 моделей, і дивився на їх середній скор.
# Звідси є подивився, що балансувати дані: погана ідея.
# Для першого підходи такої волатильності, як не дивно, немає.
#
# Усі сигнали порізав на проміжки по 20 секунд. (Дуже грубо кажучи)
#
# Оскільки процес обробки доволі довго іде (до півгодини), я кидаю ноутбук, в якому закоментовано рядок обробки даних.
# Там прочитаються уже оброблені результати з csv-шника.
#
# За бажанням, можна розкоментувати той рядок.
#
# Класифікатор: XGBoost
#
# Подальші можливі плани, які я намітив:
# - Спробую встановити, які саме люди погіршують перформанц потрапляючи в трейн або тест. +
# - Подивитись на типові помилки класифікатора -
# - Подивитись на перформанс інших Класифікаторів -
# - Можливо спробувати інші алгоритми семплінгу (боротьби з Imbalanced data) *хоча з поточним алгоритмом незбалансованість не виглядає проблемою +
# - Також можна спробувати погратись з довжиною фрагменту, який ми класифікуємо -
# - Спробувати інші підходи для train-test split
```

```
# In[1]:


import numpy as np
import pandas as pd
import wfdb
from wfdb import processing
from matplotlib import pyplot as plt
from hrvanalysis import (get_time_domain_features, get_time_domain_features,
                         get_geometrical_features, get_frequency_domain_features,
                         get_csi_cvi_features, get_poincare_plot_features)
from hrvanalysis import remove_outliers, remove_ectopic_beats, interpolate_nan_values
import sys
import os
from sklearn.metrics import f1_score, classification_report, confusion_matrix, plot_confusion_matrix


# In[2]:


fs=250


# In[3]:


#Get list of Normal-interval and list of non-normal-intervals
def get_divition(marks, points):
    Afib_intervals = []
    Normal_intervals = []
    state = 'N' if marks[0] == '(N' else 'A'
    point = points[0]
    for i in range(1, len(points)):
        current_point = points[i]
        current_mark = marks[i]
        current_state = 'N' if current_mark == '(N' else 'A'
        if current_state == state:
            continue
        if state == 'N':
            Normal_intervals.append([point, current_point])
        if state == 'A':
            Afib_intervals.append([point, current_point])
```

```
            point = current_point
            state = current_state
        result = dict()
        result['afib_intervals'] = Afib_intervals
        result['normal_intervals'] = Normal_intervals
        return result


# In[4]:


#Split long arrays on the arrays of small length
def divide_on_intervals(arrays, len_needed):
    def reshape(lst, n):
        return [lst[i*n:(i+1)*n] for i in range(len(lst)//n)]
    array_of_arrays = [reshape(i, len_needed) for i in arrays]
    return [i for j in array_of_arrays for i in j]


# In[5]:



def get_qrs(record_name, seconds_on_interval = 20):
    #get annotation
    annotation = wfdb.rdann(record_name, 'atr')
    points = np.array(annotation.__dict__['sample'])
    marks = np.array(annotation.__dict__['aux_note'])
    #get qrs
    qrs_info = wfdb.rdann(record_name, 'qrs')
    qrs_points = np.array(qrs_info.__dict__['sample'])
    #get norm and not-norm intervals
    divition = get_divition(marks, points)
    afib_qrs = [qrs_points[(qrs_points >= i[0]) &
                           (qrs_points < i[1])]
                for i in divition['afib_intervals']]
    norm_qrs = [qrs_points[(qrs_points >= i[0]) &
                           (qrs_points < i[1])]
                for i in divition['normal_intervals']]
    #get sigmals of needed length from the intervals and the signal
    afib_signals = divide_on_intervals(afib_qrs, seconds_on_interval)
    norm_signals = divide_on_intervals(norm_qrs, seconds_on_interval)
    answer = dict()
    answer['afib_qrs'] = afib_signals
    answer['normal_qrs'] = norm_signals
```

```
    return answer


# In[6]:


#get nn-intervals from rr-intervals
def get_nn_intervals(rr_intervals_list):
    rr_intervals_without_outliers = remove_outliers(rr_intervals=rr_intervals_list,
low_rri=300, high_rri=2000, verbose=False)
    # This replace outliers nan values with linear interpolation
    interpolated_rr_intervals =
interpolate_nan_values(rr_intervals=rr_intervals_without_outliers,
interpolation_method="linear")
    # This remove ectopic beats from signal
    save_stdout = sys.stdout
    sys.stdout = open('trash', 'w')
    nn_intervals_list = remove_ectopic_beats(rr_intervals=interpolated_rr_intervals,
method="malik")
    sys.stdout = save_stdout
    # This replace ectopic beats nan values with linear interpolation
    interpolated_nn_intervals =
interpolate_nan_values(rr_intervals=nn_intervals_list)
    if pd.isna(interpolated_nn_intervals[-1]):
        return interpolated_nn_intervals[:-1]
    return interpolated_nn_intervals

def get_features_from_nns(nns):
    if len(nns) < 10:
        return dict()
    nns = np.array([i for i in nns if not pd.isna(i) ])
    time_domain_features = get_time_domain_features(nns)
    geometrical_features = get_geometrical_features(nns)
    try:
        frequency_domain_features = get_frequency_domain_features(nns)
    except Exception as e:
        print('Error getting frequency domain features ', nns)
        #raise e
    csi_cvi_features = get_csi_cvi_features(nns)
    poincare_plot_features = get_poincare_plot_features(nns)
    features_dicts = [time_domain_features, geometrical_features,
                #frequency_domain_features, csi_cvi_features,
                poincare_plot_features]
    def merge_dicts(dicts):
        answ = dict()
```

```
            [answ.update(i) for i in dicts]
            return answ
    features = merge_dicts(features_dicts)
    return features
#Get features from one signal
def get_features(qrs_inds):
    import warnings
    warnings.filterwarnings('ignore')
    beginning = qrs_inds[0]*1000/fs
    rr_intervals_list = np.diff(qrs_inds)*1000/fs
    nn_intervals = get_nn_intervals(rr_intervals_list)
    features = get_features_from_nns(nn_intervals)
    warnings.filterwarnings('default')
    features['beginning'] = beginning
    return features

#Get dataframe of features from the list of the signals
def get_data_from_signals(signals):
    el_list = []
    for signal in signals:
        el_list.append(get_features(signal))
    return pd.DataFrame(el_list)


# In[7]:


#Get features and labels from the certain record
def get_data_from_record(record_name_orig, peaks_per_interval = 20):
    record_name = os.path.join('files', record_name_orig)
    answ = get_qrs(record_name, peaks_per_interval)
    norm_signals = answ['normal_qrs']
    afib_signals = answ['afib_qrs']

    norm_data = get_data_from_signals(norm_signals)
    norm_data['label'] = 0
    norm_data.fillna(0, inplace=True)

    afib_data = get_data_from_signals(afib_signals)
    afib_data['label'] = 1
    afib_data.fillna(0, inplace=True)

    data = pd.concat([norm_data, afib_data], axis = 0)
    data.insert(0, 'record', record_name_orig)
    return data
```

```
# In[8]:


def get_data_qrs_from_record(record_name_orig, peaks_per_interval = 20):
    record_name = os.path.join('files', record_name_orig)
    answ = get_qrs(record_name, peaks_per_interval)
    norm_signals = answ['normal_qrs']
    afib_signals = answ['afib_qrs']

    norm_data = pd.DataFrame(np.array([ np.diff(i) for i in norm_signals if len(i) ==
peaks_per_interval]))
    afib_data =pd.DataFrame(np.array([ np.diff(i) for i in afib_signals if len(i) ==
peaks_per_interval]))
    norm_data['label'] = 0
    norm_data['beginning'] = np.array([ i[0]*1000/fs for i in norm_signals if len(i) ==
peaks_per_interval])
    norm_data.fillna(0, inplace=True) #MAYBE NEED TO REVIEW THIS

    afib_data['label'] = 1
    afib_data['beginning'] = np.array([ i[0]*1000/fs for i in afib_signals if len(i) ==
peaks_per_interval])
    afib_data.fillna(0, inplace=True) #MAYBE NEED TO REVIEW THIS

    data = pd.concat([norm_data, afib_data], axis = 0)
    data.insert(0, 'record', record_name_orig)
    return data


# In[9]:


# get list of the dataframes, each from certain record
def process_data(peaks_per_interval = 20, print_exceptions=False,
data_from_record = get_data_from_record):
    records = []
    records_names_list_file = os.path.join('files', 'RECORDS')
    with open(records_names_list_file, 'r') as f:
        text = f.read()
        records = [i for i in text.split('\n') if i != '']
    dataframes = []
    for i in records:
        try:
            dataframes.append(data_from_record(i, peaks_per_interval))
```

```
            print(f'Read the data from the record {i}')
        except Exception as e:
            print(f'Failed to read the data from the record {i}')
            if print_exceptions:
                print("Exception:", e)
                if str(e) != 'sampto must be greater than sampfrom':
                    raise e
    return dataframes


# # У наступних рядках закоментовано власне процес діставання даних

# In[10]:


dataframes = process_data(peaks_per_interval = 30, print_exceptions=False)
data = pd.concat(dataframes, axis = 0)
data.to_csv('data-processed.csv', index=False)


# In[97]:


data.T


# In[12]:


dataframes_rrs = process_data(peaks_per_interval = 30, print_exceptions=True,
data_from_record = get_data_qrs_from_record)
data_rrs = pd.concat(dataframes_rrs, axis = 0)
data_rrs.to_csv('data-processed-rrs.csv', index=False)


# In[96]:


data_rrs


# In[ ]:
```

In[ ]:

In[19]:

import neurokit2 as nk

In[ ]:

In[ ]:

Process of getting features for one signal

In[14]:

Тут показано, як дістаються фічі з одного фрагменту сигналу

In[ ]:

In[15]:

# Дістаються нарізані сигнали з запису
record_name = 'files' + '/' + '04048'

```
answ = get_qrs(record_name, seconds_on_interval=20)
norm_signals = answ['normal_qrs']
afib_signals = answ['afib_qrs']



# In[ ]:




# In[81]:


# signal = wfdb.rdann(record_name, 'atr')
record = wfdb.rdrecord(record_name, channels=[1])
record.__dict__


# In[86]:


annotation = wfdb.rdann(record_name, 'qrs')


# In[87]:


annotation.__dict__['sample']


# In[89]:


# А далі порівняються графіки nn(t)/nn(t+1) для двох випадків:


# In[91]:


# Для normal:
qrs_inds = norm_signals[0]
nns = get_nn_intervals(np.diff(qrs_inds)*1000/fs)
a = nns[:-1]
```

```
b = nns[1:]
plt.scatter(a, b)


# In[92]:


# Для non-normal (afib)
ex_afib = afib_signals[3]
qrs_inds = get_peaks(ex_afib, fs)
nns = get_nn_intervals(np.diff(qrs_inds)*1000/fs)
a = nns[:-1]
b = nns[1:]
plt.scatter(a, b)


# In[93]:


time_domain_features = get_time_domain_features(nns)
geometrical_features = get_geometrical_features(nns)
frequency_domain_features = get_frequency_domain_features(nns)
csi_cvi_features = get_csi_cvi_features(nns)
poincare_plot_features = get_poincare_plot_features(nns)
features_dicts = [time_domain_features, geometrical_features,
            frequency_domain_features, csi_cvi_features,
            poincare_plot_features]
def merge_dicts(dicts):
      answ = dict()
      [answ.update(i) for i in dicts]
      return answ
features = merge_dicts(features_dicts)


# In[94]:


features = get_features(example)


# In[95]:


features
```

```
# In[ ]:


get_frequency_domain_features([632.0, 592.0, 552.0, 604.0, 586.0, 568.0, 620.0, 734.0, 848.0, 836.0, 840.0, 828.0, 824.0, 682.0, 540.0, 488.0, 412.0, 468.0, 468.0, 666.0, 864.0, 742.0, 620.0, 590.0, 560.0, 644.0, np.nan])


# In[ ]:
```

Тренування моделі:

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:


import numpy as np
import pandas as pd
import sys
import os
from sklearn.metrics import f1_score, classification_report, confusion_matrix, plot_confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv1D, MaxPooling1D
import numpy as np


# In[6]:


#get prepared data
def read_processed_data(filename = 'data-processed.csv'):
```

```
    all_data = pd.read_csv(filename)
    dataframes = [rows for _, rows in all_data.groupby('record')]
    return dataframes


# In[7]:


def my_train_test_split(dataframe, test_size=0.3, add_records = False):
    dataframes = [rows for _, rows in dataframe.groupby('record')]
    overall_len = sum(len(i) for i in dataframes)
    from random import sample
    dataframes_shuffled = sample(dataframes, len(dataframes))
    #train-test divisionTa
    test_data_size = overall_len * test_size
    cur_test_len = 0
    test_samples = []
    train_samples = []
    for i in dataframes_shuffled:
        if cur_test_len < test_data_size:
            test_samples.append(i)
            cur_test_len += len(i)
        else:
            train_samples.append(i)
    test_frame = pd.concat(test_samples, axis=0)
    records_test = test_frame['record']
    beginning_test = test_frame['beginning']
    test_frame = test_frame.drop(['record', 'beginning'], axis = 1)
    train_frame = pd.concat(train_samples, axis=0)
    records_train = train_frame['record']
    beginning_train = train_frame['beginning']
    train_frame = train_frame.drop(['record', 'beginning'], axis = 1)
    records = {'train': records_train, "test": records_test,
               'beginning_test': beginning_test, 'beginning_train': beginning_train}
    if add_records:
        return train_frame, test_frame, records
    return train_frame, test_frame


# In[8]:


def eval_model(model, test_frame, score = f1_score):
    predicted = model.predict(test_frame.drop(['label'], axis = 1)) > 0.5
    true_labels = np.array(test_frame['label'])
```

```
    return score(true_labels, predicted)


# In[9]:


dataframe_properties = pd.read_csv('data-processed.csv')
dataframe_rrs = pd.read_csv('data-processed-rrs.csv')


# In[10]:


dataframe_rrs


# In[11]:


try:
    train_test_combs = pd.read_csv('train_test_combs.csv')
    train_test_combs['train_recs'] = train_test_combs['train_recs'].apply(lambda x:
                                                    np.vectorize(int)(x[1:-1].split(', ')))
    train_test_combs['test_recs'] = train_test_combs['test_recs'].apply(lambda x:
                                                   np.vectorize(int)(x[1:-1].split(', ')))

except:
    train_test_combs = pd.DataFrame(columns=['train_recs', 'test_recs'])
    N_combs = 10
    for i in range(N_combs):
        _, _, recs = my_train_test_split(dataframe_properties, add_records = True)
        train_recs = set(recs['train'])
        test_recs = set(recs['test'])
        train_test_combs = train_test_combs.append({'train_recs': train_recs,
'test_recs': test_recs}, ignore_index=True)


# In[12]:


def train_test_by_records(dataframe, records):
    train_frame = dataframe[dataframe['record'].isin(records['train_recs'])]
    test_frame  = dataframe[dataframe['record'].isin(records[ 'test_recs'])]
    test_frame.set_index(['record', 'beginning'], append=True, inplace=True)
    train_frame.set_index(['record', 'beginning'], append=True, inplace=True)
```

```
    return train_frame, test_frame


# In[13]:


def eval_model_stat(build_model, df, mod_name = 'new_model', df_name=""):
    def model_row(x):
        print('next_row')
        train, test = train_test_by_records(df, x)
        model = build_model(train)
        return eval_model(model, test)
    train_test_combs[mod_name+"__"+df_name] =
train_test_combs.apply(model_row, axis=1)

def eval_model_stat_ensamble(build_model, dfs, mod_name = 'new_model',
df_name=""):
    def model_row(x):
        print('next_row')
        train_1, test_1 = train_test_by_records(dfs[0], x)
        train_2, test_2 = train_test_by_records(dfs[1], x)
        model = build_model(train_1, train_2)
        predictions = model.predict(test_1, test_2)
        return f1_score(test_1['label'], predictions)
    train_test_combs[mod_name+"__"+df_name] =
train_test_combs.apply(model_row, axis=1)


# In[14]:


def build_model_basic(train_frame_imbal):
    from xgboost import XGBClassifier
    model = XGBClassifier()
    model.fit(train_frame_imbal.drop('label', axis=1), train_frame_imbal['label'])
    return model


# In[15]:


def build_model_logictic_regr(train):
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression()
    model.fit(train.drop('label', axis = 1), train['label'])
```

```
    return model


# In[16]:


def build_model_RF(train_frame_imbal):
    from sklearn.ensemble import RandomForestClassifier
    model=RandomForestClassifier(n_estimators=100)
    model.fit(train_frame_imbal.drop('label', axis=1), train_frame_imbal['label'])
    return model


# In[ ]:




# In[17]:


eval_model_stat(build_model_basic, dataframe_properties, "xgb", "prop")


# In[18]:


eval_model_stat(build_model_logictic_regr, dataframe_properties, "log_reg",
"prop")


# In[19]:


eval_model_stat(build_model_basic, dataframe_rrs, "xgb", "rrs")


# In[20]:


eval_model_stat(build_model_logictic_regr, dataframe_rrs, "log_reg", "rrs")


# In[21]:
```

```
eval_model_stat(build_model_RF, dataframe_rrs, "RF", "rrs")

# In[22]:

eval_model_stat(build_model_RF, dataframe_properties, "RF", "prop")

# In[ ]:




# In[23]:

sum(train_test_combs['xgb__prop'] < train_test_combs['xgb__rrs'])

# In[24]:

def build_model_nn(train):

    # Neural network
    input_sh = train.shape[1] -1
    model = Sequential()
    model.add(Dense(15, input_dim=input_sh, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',                    optimizer='adam',
metrics=['accuracy'] )
    model.fit(np.array(train.drop('label', axis=1)), np.array(train['label']).reshape(-1),
            epochs=15, batch_size=50, verbose=False)
    return model

# In[25]:

eval_model_stat(build_model_nn, dataframe_rrs, "NN", "rrs")
```

```
# In[26]:


eval_model_stat(build_model_nn, dataframe_properties, "NN", "prop")


# In[27]:


stats = train_test_combs.agg(['mean', 'min', 'max', 'std', 'count']).drop(columns=['train_recs', 'test_recs']).T
stats['up'] = stats['mean'] + 1.95*stats['std']/(stats['count']**(1/2))
stats['down'] = stats['mean'] - 1.95*stats['std']/(stats['count']**(1/2))
stats.sort_values("mean", ascending=False)


# In[ ]:




# In[28]:


cur_trtst = train_test_combs.iloc[5]


# In[29]:


train_prop, test_prop = train_test_by_records(dataframe=dataframe_properties, records=cur_trtst)
train_rrs, test_rrs = train_test_by_records(dataframe=dataframe_rrs, records=cur_trtst)


# In[30]:


models = {"xgb__prop": build_model_basic(train_prop),
        "xgb__rrs": build_model_basic(train_rrs),
        "log_reg__prop": build_model_logictic_regr(train_prop)}
```

```
# In[31]:


classif_df = pd.DataFrame(test_prop['label'])

for name in models:
    if 'rrs' in name:
        cur_test = test_rrs
    else:
        cur_test = test_prop
    cur_model = models[name]
    predicts = cur_model.predict(cur_test.drop(['label'], axis=1))
    classif_df[name+' correct'] = (predicts == test_prop['label']).astype(int)


# In[32]:


classif_df['all'] = 1
classif_df.agg(lambda x: sum((x+1)%2))


# In[33]:


def percent_sharing_mistakes(s1, s2):
    sharing_mistakes = sum((classif_df[s1] + classif_df[s2]) == 0)
    min_mistaken = min(sum((classif_df[s1]+1)%2), sum((classif_df[s2]+1)%2))
    return sharing_mistakes/min_mistaken


# In[34]:


percent_sharing_mistakes('xgb__prop correct', 'log_reg__prop correct')


# In[35]:


percent_sharing_mistakes('xgb__prop correct', 'xgb__rrs correct')
```

```
# In[36]:


classif_df


# In[37]:


sum(test_rrs['label'] != test_prop['label'])


# In[38]:


one_vs_all_df = pd.DataFrame(columns=['test_record'])
for rec in set(dataframe_properties.reset_index()['record']):
    one_vs_all_df = one_vs_all_df.append({'test_record': rec}, ignore_index=True)


# In[39]:


def train_test_by_test_rec(dataframe, test_record):
    train_frame = dataframe[dataframe['record'] != test_record['test_record']]
    test_frame  = dataframe[dataframe['record'] == test_record['test_record']]
    test_frame.set_index(['record', 'beginning'], append=True, inplace=True)
    train_frame.set_index(['record', 'beginning'], append=True, inplace=True)
    return train_frame, test_frame

def eval_model_stat_1vs_all(build_model, df, mod_name = 'new_model',
df_name=""):
    def model_row(x):
        print('next_row')
        train, test = train_test_by_test_rec(df, x)
        model = build_model(train)
        return eval_model(model, test)
    one_vs_all_df[mod_name+"__"+df_name] = one_vs_all_df.apply(model_row,
axis=1)


# In[40]:


eval_model_stat_1vs_all(build_model_basic, dataframe_properties, 'xgb', 'prop')
```

```
# In[41]:


dataframe_properties['record'] != '2'


# In[42]:


eval_model_stat_1vs_all(build_model_basic, dataframe_rrs, 'xgb', 'rrs')


# In[43]:


one_vs_all_df


# In[44]:


eval_model_stat_1vs_all(build_model_logictic_regr, dataframe_properties,
'log_reg', 'prop')


# In[45]:


one_vs_all_df = one_vs_all_df.sort_values(one_vs_all_df.columns[1])


# In[46]:


rec_info = dataframe_rrs.groupby('record')['label'].agg(['count','sum'
]).rename(columns={'sum': 'pos'})
rec_info['neg'] =rec_info['count'] - rec_info['pos']


# In[47]:
```

```
one_vs_all_df.merge(rec_info.reset_index(), left_on = 'test_record', right_on='record').drop('record', axis=1)


# In[48]:


def count_pos(x):
    pos = sum(dataframe_properties[dataframe_properties['record'].isin(x['test_recs'])]['label'])
    neg = sum(1 - dataframe_properties[dataframe_properties['record'].isin(x['test_recs'])]['label'])
    return [pos, neg]
train_test_combs['pos'] = train_test_combs.apply(lambda x: count_pos(x)[0], axis=1)
train_test_combs['neg'] = train_test_combs.apply(lambda x: count_pos(x)[1], axis=1)


# In[49]:


train_test_combs['ratio'] =train_test_combs['pos']/train_test_combs['neg']


# In[ ]:




# In[50]:


#eval_model_stat(build_model_nn, dataframe_properties, "NN", "prop")


# In[ ]:




# In[51]:
```

```
def eval_net(model, test):
    predicted = model.predict(test.drop(['label'], axis = 1))
    true_labels = np.array(test['label'])
    return f1_score(true_labels, predicted.reshape(-1) > 0.5)


# In[52]:


def build_model_nn(train, fit=True):

    # Neural network
    input_sh = train.shape[1] -1
    model = Sequential()
    model.add(Dense(15, input_dim=input_sh, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'] )
    if fit:
        model.fit(np.array(train.drop('label', axis=1)), np.array(train['label']).reshape(-
1),
                  epochs=60, batch_size=100, verbose=False)
    return model


# In[53]:


tr_f1_mas = []
tst_f1_mas = []
for split_numb in range(10):

    train, test = train_test_by_records(dataframe_properties,
train_test_combs.iloc[split_numb])
    model = build_model_nn(train, fit=False)
    tr_f1=[]
    tst_f1=[]
    tr_f1_mas.append(tr_f1)
    tst_f1_mas.append(tst_f1)
    for i in range(8):
        model.fit(np.array(train.drop('label', axis=1)), np.array(train['label']).reshape(-
1), epochs=10, batch_size=100)
```

```
        tr_f1.append(eval_net(model, train))
        tst_f1.append(eval_net(model, test))


# In[54]:


mean_tr_f1 = np.array(tr_f1_mas).mean(axis=0)
mean_tst_f1 = np.array(tst_f1_mas).mean(axis=0)


# In[55]:


from matplotlib import pyplot as plt
x = np.arange(len(mean_tr_f1))
plt.plot(x*10, mean_tr_f1)
plt.plot(x*10, mean_tst_f1)
plt.legend(['tr', 'tst'])
plt.show()


# In[56]:



for i in range(30):
    model.fit(np.array(train.drop('label', axis=1)), np.array(train['label']).reshape(-1),
epochs=1, batch_size=100)
    tr_f1.append(eval_net(model, train))
    tst_f1.append(eval_net(model, test))


# In[57]:


tr_f1=[]
tst_f1=[]


# In[ ]:
```

```
# In[58]:


eval_model_stat(build_model_nn, dataframe_rrs, "NN", "rrs")


# In[59]:


eval_model_stat(build_model_nn, dataframe_properties, "NN", "prop")


# In[92]:


stats = train_test_combs.agg(['mean', 'min', 'max', 'std', 'count']).drop(columns=['train_recs', 'test_recs']).T
stats['up'] = stats['mean'] + 1.95*stats['std']/(stats['count']**(1/2))
stats['down'] = stats['mean'] - 1.95*stats['std']/(stats['count']**(1/2))
stats.sort_values("mean", ascending=False)


# In[61]:


def Stacking(train_orig, build_model = build_model_basic, test_size = 0.1):
    train = train_orig.copy()

    records = list(set(list(train.reset_index()['record'])))
    n_recs = int(len(records)*test_size)
    folds = [records[x:x+n_recs] for x in range(0, len(records), n_recs)]# test_pred=np.empty((test.shape[0],1),float)
#     train_pred=np.empty((0,1),float)
    train.loc[train.index, 'predictions'] = None
    train.loc[train.index, 'train'] = None

    for i in range(len(folds)):
        print(i)
        train.loc[train.index, 'train'] = list(~train.reset_index()['record'].isin(folds[i]))
        df_train = train[train['train']].drop(['train', 'predictions'], axis=1)
        df_test = train[~train['train']].drop(['train', 'predictions'], axis=1)

        model = build_model(df_train)
        predictions = model.predict(df_test.drop('label', axis=1))
```

```
        train.loc[~train['train'], 'predictions'] = predictions
    return train['predictions']


# In[62]:


train_prop, test_prop = train_test_by_records(dataframe_properties,
train_test_combs.iloc[1])
train_rrs, test_rrs = train_test_by_records(dataframe_rrs, train_test_combs.iloc[1])


# In[63]:


model_xgb = build_model_basic(train_rrs)
test_prop.loc[test_prop.index, 'xgb_preds'] =
model_xgb.predict(test_rrs.drop('label', axis=1))


# In[64]:


model_nn = build_model_nn(train_prop)
test_prop.loc[test_prop.index, 'nn_preds'] =
(model_nn.predict(test_prop.drop(['label', 'xgb_preds'],
                                 axis=1)) > 0.5).astype(int)


# In[65]:


test_prop


# In[66]:


train_prop.loc[train_prop.index, 'xgb_preds'] = list(Stacking(train_rrs,
build_model_basic).astype(int))


# In[67]:
```

```
train_prop.loc[train_prop.index, 'nn_preds'] = Stacking(train_prop, build_model_nn).astype(int)


# In[68]:


final_model = build_model_basic(train_prop)


# In[ ]:




# In[69]:


eval_model(final_model, test_prop)


# In[70]:


eval_model(model_nn, test_prop.drop(['nn_preds', 'xgb_preds'], axis=1))


# In[71]:


eval_model(model_xgb, test_rrs)


# In[72]:


class Ensemble_xgb_nn:
    def fit(self, train_prop_orig, train_rrs_orig):
        train_prop, train_rrs = train_prop_orig.copy(), train_rrs_orig.copy()
        self.train_prop_orig = train_prop_orig.copy()
        self.train_rrs_orig = train_rrs_orig.copy()
        train_prop.loc[train_prop.index, 'xgb_preds'] = list(Stacking(train_rrs, build_model_basic).astype(int))
```

```
        train_prop.loc[train_prop.index, 'nn_preds'] = Stacking(train_prop,
build_model_nn).astype(int)
#         print(train_prop)
        self.final_model = build_model_basic(train_prop)
    def predict(self, test_prop_orig, test_rrs_orig):
        test_prop, test_rrs = test_prop_orig.copy(), test_rrs_orig.copy()
        self.model_xgb = build_model_basic(self.train_rrs_orig)
        test_prop.loc[test_prop.index,
                      'xgb_preds'] = self.model_xgb.predict(test_rrs.drop('label',
axis=1))
#         print(test_prop)
        self.model_nn = build_model_nn(self.train_prop_orig)
        test_prop.loc[test_prop.index,
                      'nn_preds'] = (self.model_nn.predict(test_prop.drop(['label',
'xgb_preds'],
                                                          axis=1)) > 0.5).astype(int)

        return self.final_model.predict(test_prop.drop('label', axis=1))


# In[73]:


def build_ensemble_model(train_prop, train_rrs):
    model = Ensemble_xgb_nn()
    model.fit(train_prop, train_rrs)
    return model


# In[74]:


train_prop, test_prop = train_test_by_records(dataframe_properties,
train_test_combs.iloc[1])
train_rrs, test_rrs = train_test_by_records(dataframe_rrs, train_test_combs.iloc[1])


# In[75]:


model = build_ensemble_model(train_prop, train_rrs)


# In[76]:
```

```
predictions = model.predict(test_prop, test_rrs)


# In[77]:


f1_score(predictions, test_prop['label'])


# In[78]:


model_nn = build_model_nn(train_prop)
eval_model(model_nn, test_prop)


# In[79]:


model_xgb = build_model_basic(train_rrs)
eval_model(model_xgb, test_rrs)


# In[80]:


predictions_compare = pd.DataFrame()


# In[81]:


predictions_compare['label'] = list(test_rrs['label'])
predictions_compare['ens'] = list(predictions)
predictions_compare['xgb'] = model_xgb.predict(test_rrs.drop('label', axis=1))
predictions_compare['nn'] = (model_nn.predict(test_prop.drop('label', axis=1) ) >
0.5).astype(int)
predictions_compare['count'] = 1


# In[82]:


model.final_model
```

```
from xgboost import plot_importance
from matplotlib import pyplot as plt
# plot_importance(model.final_model, max_num_features=100)
# plt.show()
names = model.final_model.get_booster().feature_names
imps = model.final_model.feature_importances_

names = [x for _,x in sorted(zip(imps,names))]
imps = sorted(imps)

plt.xticks(rotation=90)
plt.plot(names[:-1],
      imps[:-1], 'X')


# In[83]:


names = model_xgb.get_booster().feature_names
imps = model_xgb.feature_importances_

names = [x for _,x in sorted(zip(imps,names))]
imps = sorted(imps)

plt.xticks(rotation=90)
plt.plot(names[:-1],
      imps[:-1], 'X')


# In[ ]:




# In[84]:


model.final_model


# In[85]:
```

```
pc = predictions_compare
len(predictions_compare[(pc['ens'] == pc['label']) & (pc['ens'] != pc['xgb']) & (pc['ens'] != pc['nn'])])


# In[86]:


pd.DataFrame(predictions_compare.groupby(list(pc.columns)[:-2]).sum()['count'])


# In[87]:


pd.DataFrame(predictions_compare[pc['label'] == 1].groupby(list(pc.columns)[1:-1]).sum()['count'])


# In[88]:


list(pc.columns)


# In[89]:


eval_model_stat_ensamble(build_ensemble_model,
                         [dataframe_properties,
                          dataframe_rrs], "ensamble_nn+xgb", "prop_or_rrs")


# In[90]:


train_test_combs.to_csv('train_test_combs.csv', index=False)


# In[91]:


train_test_combs


# In[ ]:
```

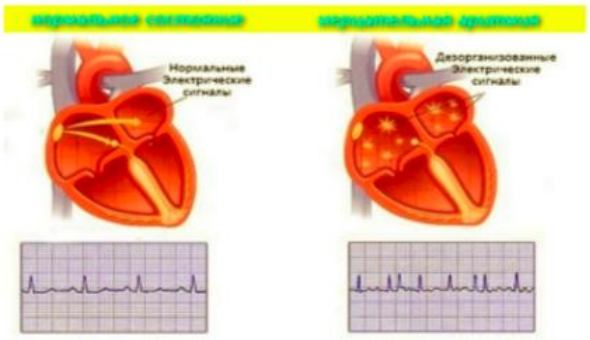
ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

Система розпізнавання серцевої аритмії за сигналами ЕКГ на основі методів машинного навчання

Виконав: Студент IV курсу, групи КА-71 Гульчук Всеволод
Керівник: доцент кафедри ММСА, к.ф.-м. наук, Дідковська Марина Віталіївна

Актуальність проблеми

- Серйозність проблеми
- Наявність інструментів



Постановка задачі

- ***Об'єкт дослідження***: ЕКГ сигнали
- ***Предмет дослідження:*** Встановлення аритмії за ЕКГ сигналами методами машинного навчання
- ***Мета дослідження:*** Розробка моделі що встановлює аритмію за ЕКГ сигналом, дослідження її властивостей та перспектив, а також порівняння методів між собою та оцінка їх адекватностей

Існуючі рішення

- Apple watch
- Mawi band
- Часткові рішення

pyHRV

Python Toolbox for Heart Rate Variability

Стратегія вирішення

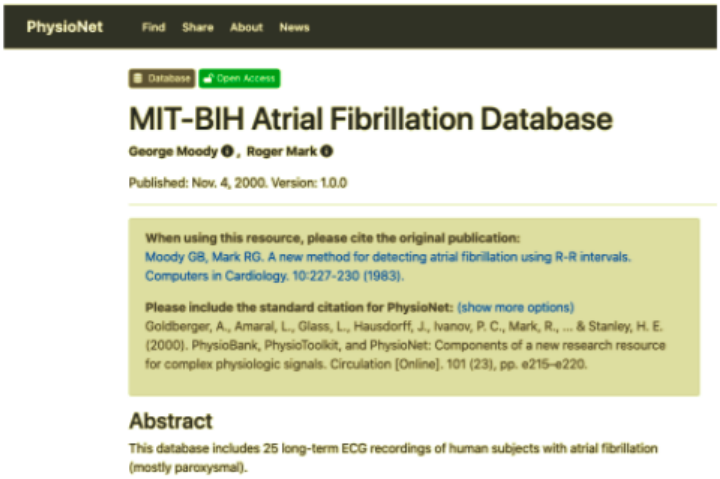
-Пошук даних

-Побудова моделі

-Аналіз моделі на адекватність

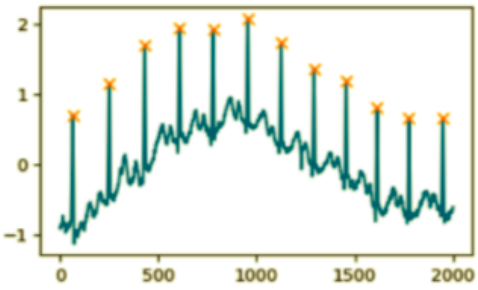
Дані

MIT-BIH Atrial Fibrillation Database



Передобробка даних

- Розділення на проміжки
- Виділення піків
- Виділення властивостей



Дані1. Властивості сигналу

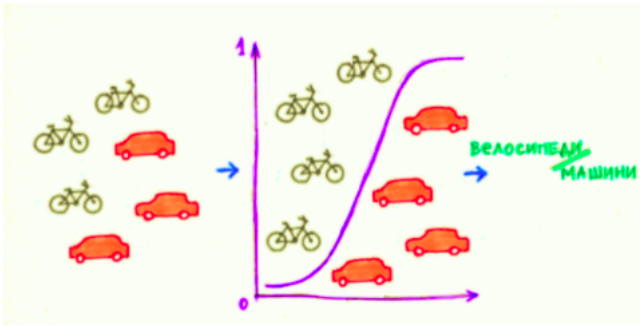
Дані1. Сирі Піки сигналу

Побудова моделей

Логістична регресія

- Простота
- Інтерпретовність
- Відсутність нелінійних переваг

$$h_\theta(x^{(i)}) = P(\text{class}(x^{(i)}) = 1, \theta) = \frac{1}{1 + e^{-z(x^{(i)}, \theta)}},$$



$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

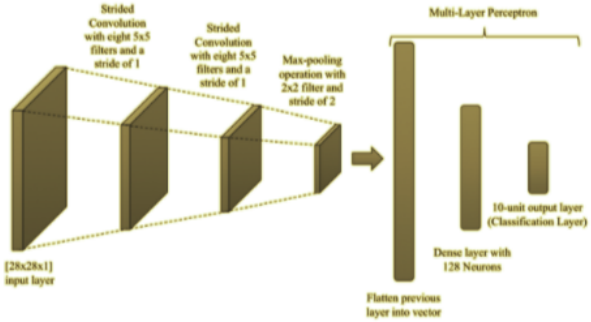
$$\text{Cost}(p, y) = -(y \log(p) + (1 - y) \log(1 - p))$$

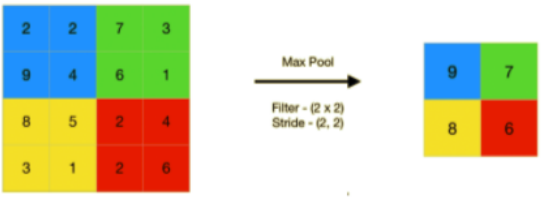
Згорткова нейромережа

- Універсальність
- Довгий час тренування
- Довгий час налаштування

Forward Propagation

$$W^{(1)^T} X = z^{(2)}$$
$$a^{(2)} = g(z^{(2)})$$
$$W^{(2)^T} a^{(2)} = z^{(3)}$$
$$a^{(3)} = g(z^{(3)})$$

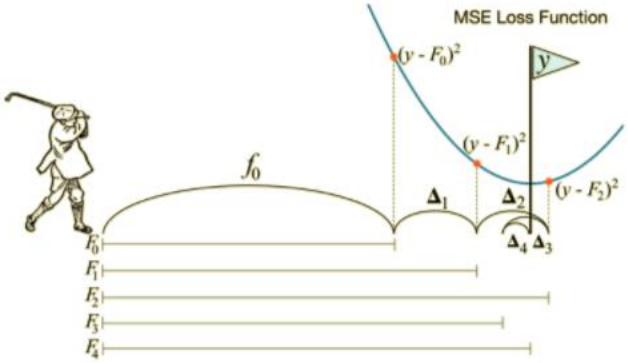




$$y_j = b_j + \sum_{c=0}^{n_c-1} \sum_{k=-p}^{p} x_{c,j-k} w_{c,k} \qquad \text{convolution}$$

XGBoost

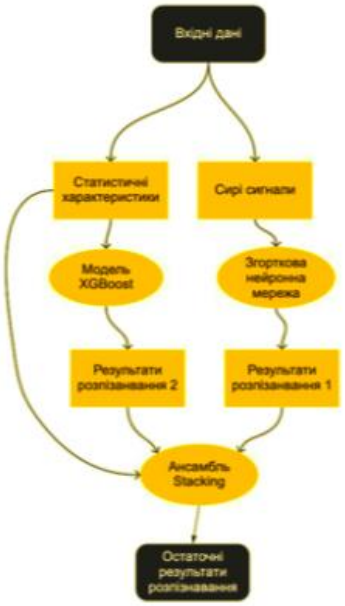
- Нелінійна модель
- Доволі швидко тренується в порівнянні з іншими
- Не потребує детального налаштування



$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$
$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Ансамбль Stacking (розроблена модифікація)

Поєднання результатів попередніх моделей



Аналіз помилок

- Аналіз кожного пацієнта
- Аналіз помилок моделей

label	ens	xgb	count
0	0	0	4082
		1	5
	1	0	6
		1	17
1	0	0	153
		1	302
	1	0	91
		1	3105

	test_record	xgb_prop	xgb_rrs
0	5121	0.930211	0.942423
1	8455	0.800000	0.857143
2	4746	0.980178	0.990177
3	5261	0.318182	0.409091
4	8215	0.000000	0.000000
5	6426	0.990947	0.992605
6	8219	0.757447	0.812670
7	4126	0.721088	0.899160
8	4908	0.879668	0.997658
9	4015	0.250000	0.206897
10	6453	0.952381	0.777778
11	8378	0.876836	0.906926
12	7879	0.997748	0.998501
13	4936	0.845983	0.947368
14	4043	0.940803	0.924037

Аналіз адекватності моделі

- Статистичний метод оцінювання;

	min	max	count	mean	std	up	down
neg	3975.000000	6682.000000	10.0	4871.400000	787.577608	5357.055120	4385.744880
pos	866.000000	4241.000000	10.0	3053.800000	937.379539	3631.829603	2475.770397
xgb_rrs	0.660790	0.962429	10.0	0.907861	0.088164	0.962227	0.853495
NN_prop	0.702058	0.949587	10.0	0.896954	0.071229	0.940877	0.853031
ensamble_nn+xgb_prop_or_rrs	0.662777	0.949808	10.0	0.895546	0.085300	0.948146	0.842946
log_reg_prop	0.739489	0.944729	10.0	0.894583	0.056904	0.929673	0.859493
RF_prop	0.662548	0.943568	10.0	0.891900	0.082668	0.942877	0.840923
xgb_prop	0.636364	0.942002	10.0	0.887155	0.091167	0.943373	0.830937
RF_rrs	0.555878	0.940144	10.0	0.865987	0.112153	0.935146	0.796829
			10.0	0.668			
			10.0	0.662			
			10.0	0.501			

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

$$\bar{x} - z_{\alpha/2}\frac{\sigma}{\sqrt{n}} < \mu < \bar{x} + z_{\alpha/2}\frac{\sigma}{\sqrt{n}}$$

Перспективи подальших досліджень

- Розробка веб-інтерфейсу для моделі
- Навчання на більш обширних даних
- Розробка нових моделей на основі інших моделей машинного навчання
- Візуалізація на кластеризація даних
- Проведення детального факторного аналізу

Перспектива застосування

- Веб-сайт
- Комерційний застосунок (закритий продукт)

Наукова новизна

- Метод статистичної валідації
- Модифікація стекінгу
- Конкретна реалізація згорткової нейронної мережі
- Дослідження зв'язків між помилками
- Ефективне поєднання двох наборів даних, узятих з одного сигналу

Висновки

- Було розроблено багато моделей та порівняно їхні адекватності: найбільш перспективною виявився розроблений стекінг, а найбільш адекватною локально XGBoost на сирих даних, де модель сама визначає правильні параметри
- Розроблена модель поводить себе надійно на більшості пацієнтів, хоча є такі, для яких необхідне подальше дослідження.
- Моделі не діють за однаковим принципом та поєднуючи їх логіку можна досягати кращих результатів.
- Проблема розпізнавання серцевої аритмії стоїть дуже гостро на сьогодні та потребує нових рішень та реалізацій.

Дякую за увагу!