НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:
Завідувач кафедри
_____ Олена ЧУМАЧЕНКО
« ____ »_____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою
«Системи і методи штучного інтелекту»
зі спеціальності 122 «Комп'ютерні науки»

на тему: **«Автоматизація класифікації стадій сну методами машинного навчання»**

Виконав:
студентка IV курсу, групи КІ-93
Деньгуб Дар'я Сергіївна _____

Керівник:
доцент, к.т.н. Жиров О.Л. _____

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

Консультант з нормоконтролю:
фахівець першої категорії Гончарук М.М. _____

Рецензент:
_____

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

Київ – 2023

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

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки»
Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олена ЧУМАЧЕНКО
« ____ »_____ 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту
Деньгуб Дар'ї Сергіївни

1. Тема роботи «Автоматизація класифікації стадій сну методами машинного навчання», керівник роботи Жиров Олександр Леонідович, доцент, к.т.н., затверджені наказом по університету від «30» травня 2023 р. № 2065-с.

2. Термін подання студентом роботи: 16.06.2023

3. Вихідні дані до роботи: набір даних полісомнографії.

4. Зміст роботи: аналіз існуючих методів застосування алгоритмів машинного навчання для вирішення задачі автоматизації оцінки стадій сну. Практична реалізація методів градієнтного бустингу та згорткової нейронної мережі. Проведення порівняльного аналізу та інтерпретація отриманих результатів.

5. Перелік ілюстративного матеріалу (із зазначення плакатів, презентацій тощо): графічне представлення схем алгоритмів, зображення вхідних даних та результатів прогнозування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к.е.н.		

7. Дата видачі завдання: 21 лютого 2023 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи	17.04.2023	Виконано
2.	Підготовка першого розділу	01.05.2023	Виконано
3.	Підготовка другого розділу	14.05.2023	Виконано
4.	Розробка програмного продукту	17.05.2023	Виконано
5.	Підготовка третього розділу	21.05.2023	Виконано
6.	Підготовка економічної частини	23.05.2023	Виконано
7.	Оформлення розділів відповідно до нормоконтролю	26.05.2023	Виконано
8.	Підготовка презентації доповіді	28.05.2023	Виконано
9.	Оформлення дипломної роботи	30.05.2023	Виконано

Студент Дар’я ДЕНЬГУБ

Керівник Олександр ЖИРОВ

РЕФЕРАТ

Дипломна робота: 102 ст., 10 табл., 28 рис., 1 додаток, 26 джерел.

КЛАСИФІКАЦІЯ СТАДІЙ СНУ, ПОЛІСОМНОГРАФІЯ, ГРАДІЄНТНИЙ БУСТИНГ, ДЕРЕВА РІШЕНЬ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, МЕТОДИ БАЛАНСУВАННЯ ДАНИХ

Об’єкт дослідження – задача оцінювання стадій сну на основі даних полісомнографії.

Предмет дослідження – застосування методів машинного навчання для класифікації даних.

Метою роботи є реалізація моделі машинного навчання для класифікації стадій сну відповідно до даних полісомнографії.

У роботі розглядаються два основні підходи застосування методів машинного навчання для класифікації стадій сну. Перший полягає у використанні апріорних знань про сигнал для вилучення ознак і подальше застосування класифікатора для оцінки фаз сну. Другий базується на використанні згорткових мереж для автоматичного вилучення ознак та класифікації стадій за допомогою повнозв’язних шарів.

Результатом роботи є найкраща побудована модель, яка обрана на основі порівняння метрик якості, для вирішення задачі класифікації стадій сну на основі даних полісомнографії.

ABSTRACT

Thesis: 102 pages, 10 tables, 28 figures, 1 appendix, 26 sources.

CLASSIFICATION OF SLEEP STAGES, POLYSOMNOGRAPHY, GRADIENT BOOSTING, DECISION TREES, CONVOLUTIONAL NEURAL NETWORK, DATA BALANCING METHODS

Object of study - the task of estimating stages based on polysomnography data.

The subject of research is the application of machine learning methods for data classification.

The purpose of the study is to implement a machine learning model for classifying sleep stages according to polysomnography data.

The paper considers two main approaches of applying machine learning methods to classify sleep stages. The first involves using a priori knowledge of the signal to extract features and then applying a classifier to estimate sleep phases. The second is based on the use of convolutional networks to automatically extract features and classify stages using fully connected layers.

The result of the work is the best built model, which is selected based on the comparison of quality metrics, to solve the problem of classifying sleep stages based on polysomnography data.

ЗМІСТ

ВСТУП

Сон – невід’ємна частина життя кожної людини; активний та закономірний процес, який виконує важливу відновлюючу функцію для фізичного та психічного здоров’я.

За даними Всесвітньої організації охорони здоров'я (ВООЗ), близько 30% дорослого населення у світі мають симптоми порушення сну, що можуть призвести як до незначних наслідків, таких як втома та знижена продуктивність, так і критичних – зниження імунітету, вищий шанс деменції. Тому вивчення фізіології сну є актуальною темою для науковців та медиків. Одним з найважливіших аспектів дослідження сну є класифікація його стадій. Відповідно до Американської академії медицини сну сон поділяють на 4 стадії:

1. N1 – поверхневий сон.

2. N2 – сон помірної глибини.

3. N3 – глибокий сон. Разом з N1 та N2 формує повільний сон (NREM), який характеризується уривчастими неемоційними думками.

4. Швидкий сон (REM) – характеризується швидким рухом очей та розслабленням м’язів.

AASM рекомендує проводити оцінку стадій на основі даних полісомнографії, а саме сигналів електроенцефалограми (активність головного мозку), електроокулограми (активність очей) та електроміограми (активність м’язів). Тобто для кожних 30-ти секунд записів сигналів лікар ставить у відповідність стадію сну. Очевидно, аналітична оцінка фаз сну є дуже трудомістким та суб’єктивним завданням для людини.

До того ж, така задача є цікавою для сфери штучного інтелекту, адже:

- Записи ЕЕГ містять викиди, отже потребують попередньої фільтрації;
- на стадію N2 припадає 45–55% від загальної тривалості сну, тоді як на N3 - лише 10-15%, тому потрібно зважати на незбалансованість даних;
- стадії - послідовні, тож потрібно враховувати темпоральний контекст.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність роботи

Сон здорової людини має певний закономірний цикл, і кожна стадія сну має свою певну тривалість. Таким чином, порівнюючи сон здорової людини та хворої, можна спостерігати, наприклад, прогресування хвороби Альцгеймеру (рисунок 1.1).

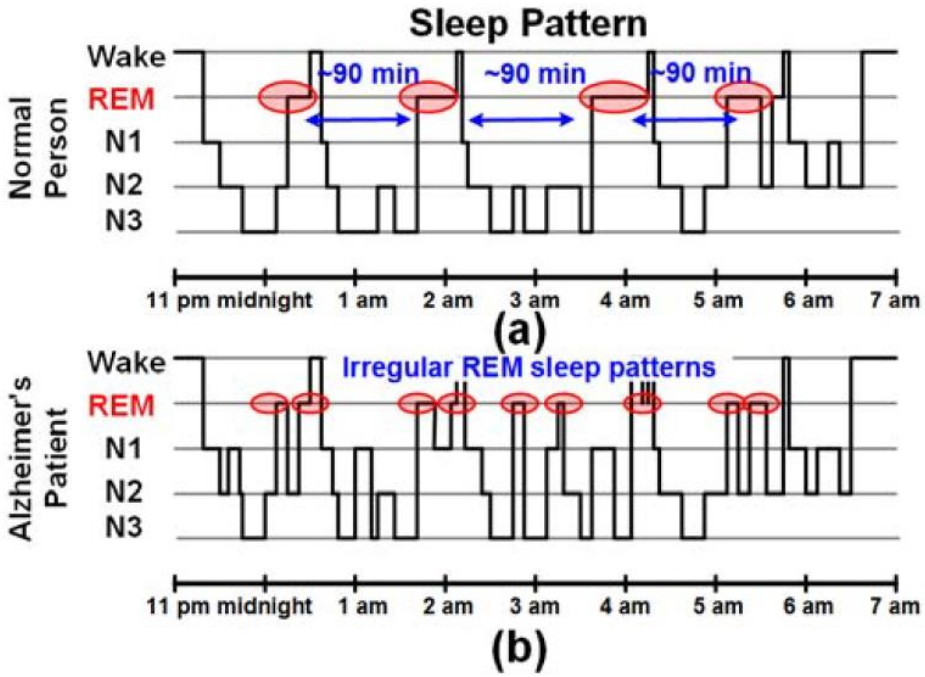

Рисунок 1.1 – (a) – паттерн сну здорової людини, (b) – паттерн сну людини хворої Альцгеймером [1]

Або спостерігати наскільки ефективним є лікування (рисунок 1.2). Як видно з гіпнограми без лікування, пацієнту потребувалося близько години, щоб заснути, і після цього відбувалося багато пробуджень. Ефективність сну, визначена як відсоток часу часу, проведеного у ліжку, становила 79% до та 96% після зопіклону.

Саме тому класифікація стадій сну є важливою задачею. В звичайних умовах лікар визначає стадію сну мануально.

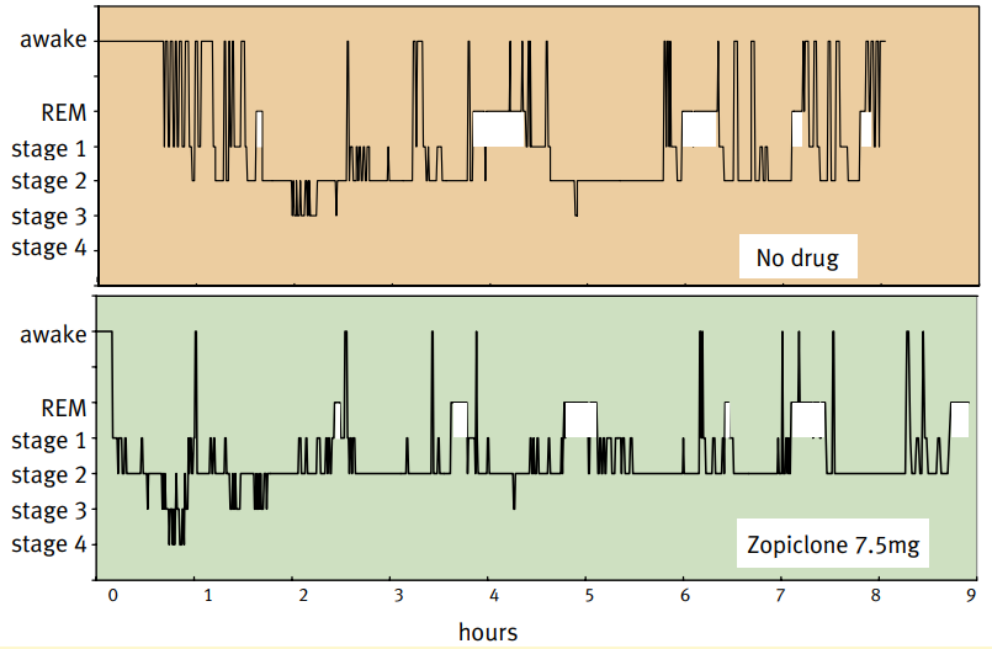

Рисунок 1.1 - Гіпнограми пацієнта з безсонням без лікування (верхній графік) та після зопіклону 7,5 мг [2]

Мануальна класифікація стадій сну має наступні проблеми та обмеження:

• Неоднозначність оцінювання сну: Немає чітких критеріїв для визначення кожної стадії. Стадії визначаються емпірично спеціалістами, і деякі стадії сну (Глибокий сон, швидкий сон) мають особливості, завдяки яким можна ідентифікувати стадію, але різниця між неспанням, N1 і N2 менш чітка. Тому існує багато розбіжностей в оцінці N1 (23-74% згоди для N1) [3].

• Вартість оцінки сну: Полісомнографія займає багато часу для аналізу біосигналів, записаних під час сну, оскільки експерти досліджують результати вручну. Тому важко вільно призначати полісомнографію пацієнтам, оскільки це дороге медичне дослідження.

Тому автоматична класифікація стадій сну мала б наступні переваги:

• Оцінка стадій сну може бути надійною. Система щоразу видаватиме однакові та стабільні результати, в той час як люди схильні до суб’єктивності та помилок.

• Сомнологи можуть заощадити час на маркування записаних сигналів під час полісомнографії та витратити його на діагностування хвороби.

1.2 Стадії сну та їх характеристики

1.2.1 Полісомнографія та системи класифікації стадій сну

Сон складається з послідовності різних фізіологічних стадій, які можна відрізнити за характеристиками, отриманими з ЕЕГ, ЕОГ, ЕМГ. Нейрофізіологи ідентифікують різні типи сигналів мозку, такі як альфа, бета, дельта та тета, які пов'язані з різними стадіями сну [4].

Електроенцефалографія - метод реєстрації сумарної електричної активності головного мозку, відведеної з поверхні шкіри голови. Для зняття сигналів використовують металеві електроди [5]. Приклад розташування електродів зображено на рисунку 1.1.

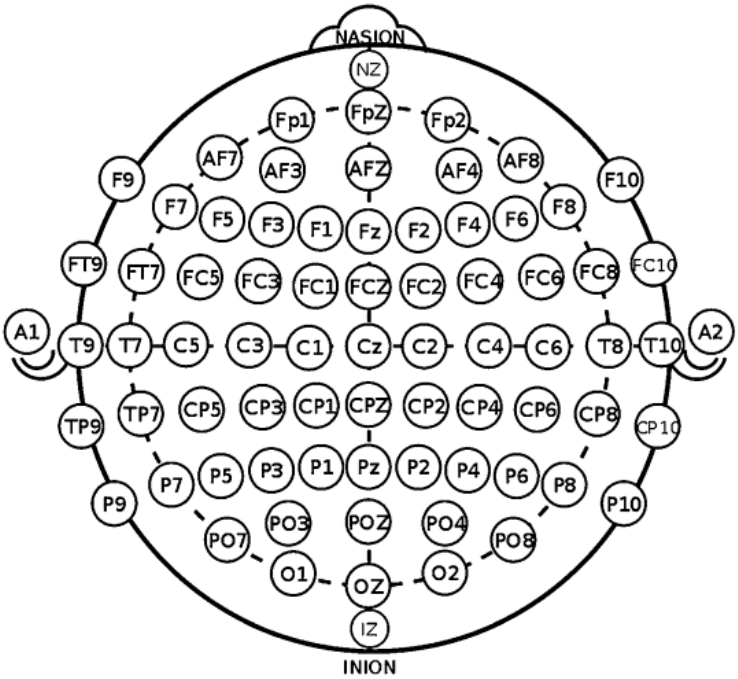

Рисунок 1.1 – Приклад розташування електродів на поверхні голови [6]

Електроокулографія – метод, що полягає у вимірюванні електричної активності м'язів навколо очей, що дозволяє фіксувати рухи очей.

Рекомендовано розташовувати один електрод над правим оком, інший – під лівим.

Електроміографія – метод, що полягає у фіксуванні електричної активності м’язів. Для задачі розпізнавання стадій сну електроди розташовують на підборідді.

Всі ці сигнали об’єднуються у єдиному дослідженні, яке називається полісомнографія (ПСГ). Далі, для кожного періоду тривалістю 20-30 с на основі даних ПСГ визначають стадію сну.

Також до ПСГ можуть включати дані електрокардіограми, додаткові канали електроміографії, для яких електроди розташовують на ногах, параметри дихання, ступінь насиченості киснем, положення тіла.

У 1968 році була запропонована перша система класифікації стадій сну - Rechtschaffen and Kales, названа на честь головних авторів. Вони виділили дві головні стадії: REM (Rapid Eyes Movement, Швидкий сон) та NREM (Non-Rapid Eyes Movement, Повільний сон), з останньої додатково визначили стадії N1, N2, N3 та N4 відповідно до міцності сну [7]. Ця система була фактично замінена роботою «The AASM Manual for the Scoring of Sleep and Associated Events», яка побачила світ у 2007 році. Головна відмінність полягала в тому, що згідно з AASM стадії N3 та N4 були об’єднані в одну – N3, також відому від назвою SWS (Slow Wave Sleep) [6]. Варто зауважити, що, оскільки під час сну людина може прокидатися, то необхідно виокремити моменти цього неспання в окрему стадію – Wake (W), яка також має свої особливості, які можна помітити на результатах ЕЕГ, ЕОГ, ЕМГ.

1.2.2 Особливості стадій сну

Кожна стадія сну має свої особливості, які можна побачити на даних ПСГ. Зупинимося на кожній стадії детальніше:

• N1 – поверхневий сон. З боку ЕЕГ характеризується наявністю тета хвиль частотою 4-7 Гц, з боку ЕОГ – майже відсутністю рухів очей, з боку ЕМГ – присутністю м’язового тонусу. Стадія N1 не завжди відчувається як сон, але це важлива фаза між неспанням і більш глибоким сном. Під час цієї стадії багато людей може зазнати відчуття падіння, що можна зафіксувати на ЕМГ.

• N2 – сон помірної глибини. Виділяється на ЕЕГ наявністю сигма-ритму (sleep spindle) частотою 12-14 Гц, який відіграє важливу роль як у сенсорній обробці, так і в консолідації довготривалої пам'яті; а також наявний так званий К-комплекс: швидкі хвилі з високою амплітудою. М’язовий тонус поступово зменшується [8].

• N3 – глибокий сон. Дельта хвилі частотою 0-3 Гц з’являються на ЕЕГ. Як і для попередніх стадій NREM сну, на ЕОГ спостерігається мінімальна присутність повільних рухів очей, на ЕОМ – глибоке розслаблення і зниженням м’язового тонусу. Ця стадія, як і загалом увесь повільний сон характеризується як час відпочинку, оскільки мозок використовує набагато менше енергії, присутні процеси консолідації пам’яті.

• REM – швидкий сон. Швидкий сон визначається наявністю ЕЕГ-паттерну низької напруги, на ЕОГ з'являються швидкі рухи очей, що відбуваються ізольовано або сплесками, а також на ЕОМ спостерігається повне розслаблення м'язів, тобто м'язова атонія. Вона переривається сплесками м'язового тонусу, що іноді призводить до поштовхів тіла [9]. Під час цієї стадії активується мигдалеподібне тіло і починається процес обробки емоцій, також спостерігаються процеси консолідації пам’яті.

• W – стадія неспання. Присутні альфа хвилі частотою 8-12 Гц, на початку цієї стадії на ЕОГ можна помітити блимання очима частотою приблизно 0.5-2 Гц, яке з часом поступово сповільнюється та може замінитися повільними рухами очей. Відповідно до ЕМГ амплітуда рухів хоч може бути і невисокою, але порівняно більшою, ніж під час сну [6].

На рисунку 1.2 зображена візуалізація сигналів під час різних стадій сну відповідно до вище згаданих пунктів.

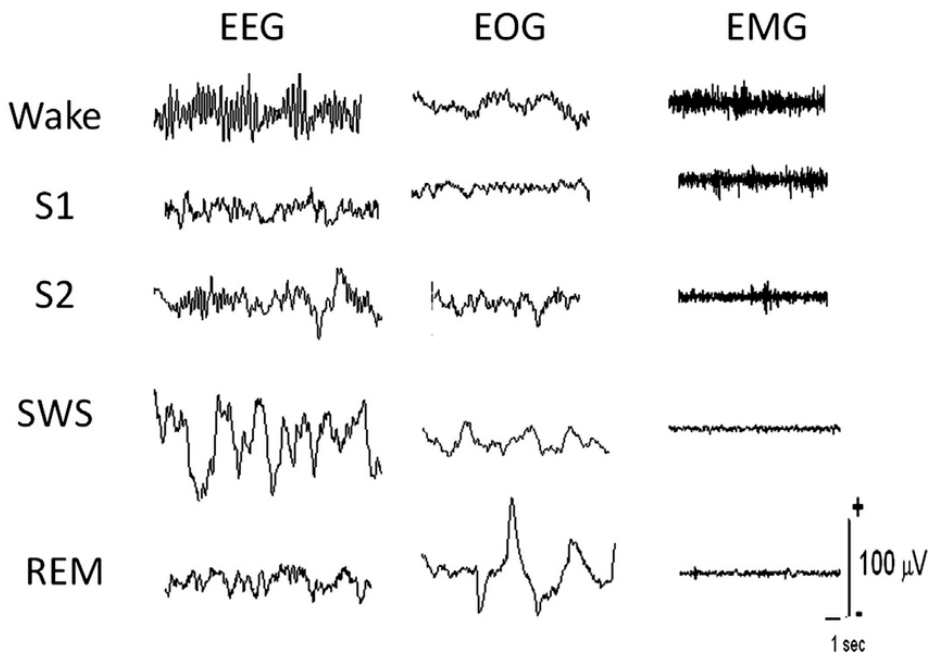

Рисунок 1.2 – Типовий вигляд ПСГ під час стадій W (Wake), N1 (S1), N2 (S2), N3 (SWS), REM. Кожен стовпець містить сигнали ЕЕГ, ЕОГ, ЕОМ, які були записані під час відповідної стадії сну [10]

1.2.3 Цикл сну здорової людини

Молода здорова людина має певний цикл зміни стадій сну. Спочатку починається стадія N1, яка триває від 1 до 7 хвилин. Ця стадія може перериватися неспанням. Далі здійснюється перехід до стадії N2 і вона триває від 10 до 25 хвилин. Поступово з'являються повільні хвилі високої напруги, що сигналізує про початок стадії N3, що триває від 20 до 40 хвилин. Є можливість короткочасного переходу до стадії N2, після чого наступає стадія REM. Перший період швидкого сну відбувається 4-8 хвилин і після його закінчення починається новий цикл сну. Швидка і повільна фази продовжують чергуватися протягом ночі і з кожним наступним циклом, його тривалість збільшується від 90 хвилин на початку до 120 хвилин в кінці. Зазвичай, буває не більше чотирьох циклів. Пропорції REM і NREM також змінюються з кожним циклом. У перших двох циклах сну переважає швидка фаза, а в останніх двох циклах - повільна

фаза. В останньому циклі стадія REM зустрічається рідко або взагалі відсутня, тоді як фаза NREM становить найдовшу частину [6].

Тож стадії під час сну йдуть у певному порядку, що потрібно враховувати при розробці моделі. Не менш важливим є те, що розподіл стадій під час сну не є рівномірним, оскільки частина, що припадає на стадію N2 може досягати 50%, тоді як частина, що припадає на стадію N3 – до 15% [11].

Моделі сну змінюються з часом. У дітей віком до 1 місяця швидкий сон, як правило, становить понад 50% від загального часу сну. Цей відсоток швидко знижується до 25% у препубертатному періоді, а потім стабілізується на рівні 20% до старості. Як правило, фаза швидкого сну має найбільшу тривалість у дитинстві і поступово знижується в дорослому віці, досягаючи найнижчого рівня у віці від 50 до 55 років. Це зниження може залежати від статі, оскільки чоловіки демонструють більш помітне зниження. Тому при класифікації стадій сну важливо враховувати не лише дані ПСГ, а й такі характеристики людини, як її вік [6].

1.2.4 Гіпнограма

Після того як вхідні дані було повністю класифіковано будують гіпнограму – графік, де зображено стадії сну як функцію від часу (рисунок 1.3).

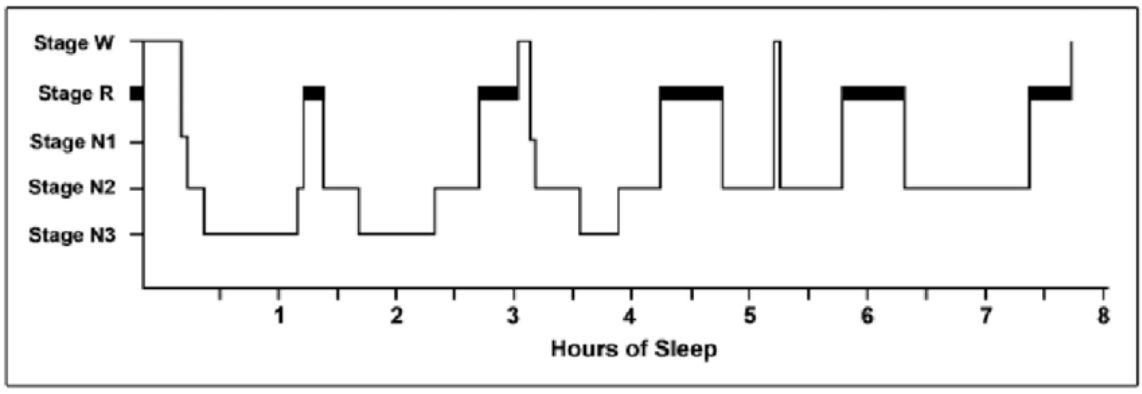

Рисунок 1.3 – Приклад гіпнограми [12]

Гіпнограма корисна тим, що на ній можна зафіксувати прояви таких хвороб як безсоння, депресія, Альцгеймер.

1.3 Фільтрація сигналів

1.3.1 Артефакти

ЕЕГ-сигнали не завжди відображають реальну активність мозку, оскільки дані містять шум та так звані артефакти – електрична активність, яка була зафіксована електродами ЕЕГ, але відображає немозкову активність, наприклад: рухи тілом або блимання очима. Тому важливо здійснити попередню обробку сигналів перед навчанням моделі, для того щоб зменшити вплив артефактів.

Виділяють два типи артефактів:

• Фізіологічні. Джерелом таких сигналів є тіло людини. Це може бути блимання та рухи очима, рухи головою, серцебиття та активність м’язів. Ідентифікувати фізіологічні артефакти можливо, якщо доступні інші біометричні дані, такі як ЕОГ та ЕМГ. В загальному, для видалення фізіологічних артефактів із записів ЕЕГ було запропоновано кілька методів попередньої обробки, що дозволяють поліпшити співвідношення сигнал/шум. Існують методи, відповідно яких цілі часові сегменти ЕЕГ, в яких було ідентифіковано артефакти, відкидаються. Для методів, що базуються на аналізі незалежних компонент (ICA), з сигналів ЕЕГ видаляються компоненти, пов'язані з артефактами. Тим не менш, таким чином, видаляється також велика кількість корисної церебральної інформації в ЕЕГ [13]. Артефакти, пов'язані з активністю очей, не знищують ЕЕГ-сигнал, що генерується мозком, а лінійно додаються до нього, й були розроблені методи для послаблення таких артефактів [14-15].

• Нефізіологічні артефакти. Вони, як правило, спричинені через втручання ззовні. Основне джерел нефізіологічних артефактів - електричні перешкоди. Це змінна напруга в електромережі частотою 50 Гц. Цей артефакт можна усунити

шляхом налаштування середи, де здійснюється запис ЕЕГ або застосувавши режекторний (notch) фільтр на 50 або 60 Гц [16]. Іншою частою причиною появи нефізіологічного артефакту є погане розміщення електродів на шкірі голови. Коли електрод рухається, він створює потенціал постійного струму, і виникає характерний електродний "хлопок": високовольтний дуже крутий спад напруги, за яким слідує експоненціальний спад. Найкращий спосіб зменшити електродний артефакт - це правильне розташування та обслуговування електродів. Найпростіший спосіб мінімізувати вплив цих нефізіологічних артефактів - відрегулювати навколишнє середовище (наприклад, екранувати приміщення, належним чином закріпити електроди) [13].

1.3.2 Типові цифрові фільтри

До сирих записів ЕЕГ можна застосувати цифрові фільтри, це може значно покращити інтерпретацію ЕЕГ та знизити вплив шумів та артефактів. Найпростіші види цифрових фільтрів є видалення певних частот. Відповідно до того, які частоти видаляються або залишаються, можна виділити чотири типи фільтрів: фільтр нижніх частот зберігає сигнали з частотами нижче певного значення, послабляючи або видаляючи сигнали з високими частотами; фільтр високих частот навпаки зберігає сигнали з частотами вище певного значення, послаблюючи низькочастотні сигнали; смуговий фільтр зберігає сигнали з частотами між нижньою і верхньою межею, а для режекторного фільтра сигнали з частотами між нижньою і верхньою межею навпаки послаблюються або видаляються, а сигнали, частоти яких виходять за ці межі - залишаються. На рисунку 1.4 можна знайти схематичне зображення дії кожного фільтру. При виборі фільтрів слід враховувати частотні діапазони артефактів, що містяться в ЕЕГ-записах. Наприклад, для усунення низькочастотних дрейфів застосовуються високочастотні фільтри з обмеженням 0,1 Гц, а для усунення

високочастотних шумів (наприклад, перешкод від м'язової активності) - низькочастотні фільтри, наприклад, з обмеженням 30 Гц. Крім цього, при виборі фільтрів слід також враховувати, сигнали якого частотного діапазону вас цікавлять.

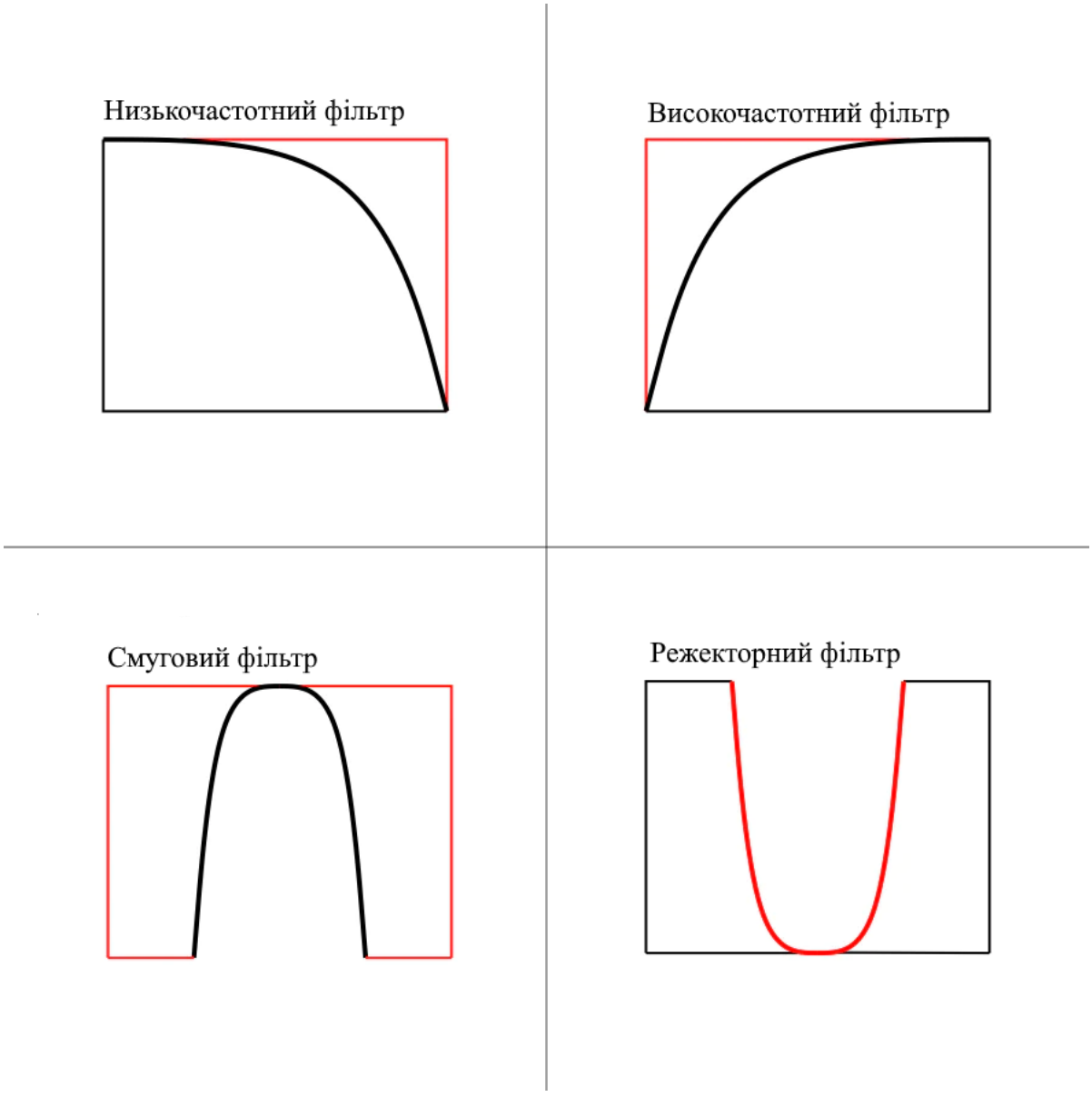

Рисунок 1.4 – Чотири типи найпростіших фільтрів: низькочастотний, високочастотний, смуговий та режекторний

Існує також метод аналізу незалежних компонент, але зазвичай він потребує великої кількості каналів ЕЕГ, що не завжди є доступним.

1.4 Методи балансування набору даних

1.4.1 Незбалансованість стадій сну

Як вже зазначалося, розподіл стадій під час сну є нерівномірним, на рисунку 1.5 зображено кругову діаграму відсотків типової тривалості кожної стадії сну. Незбалансованість даних зазвичай спричинює перенавчання моделі на одному класі і недонавчання на іншому, оскільки прагне мінімізувати загальну помилку класифікації. Такий недолік значно погіршує якість моделі.

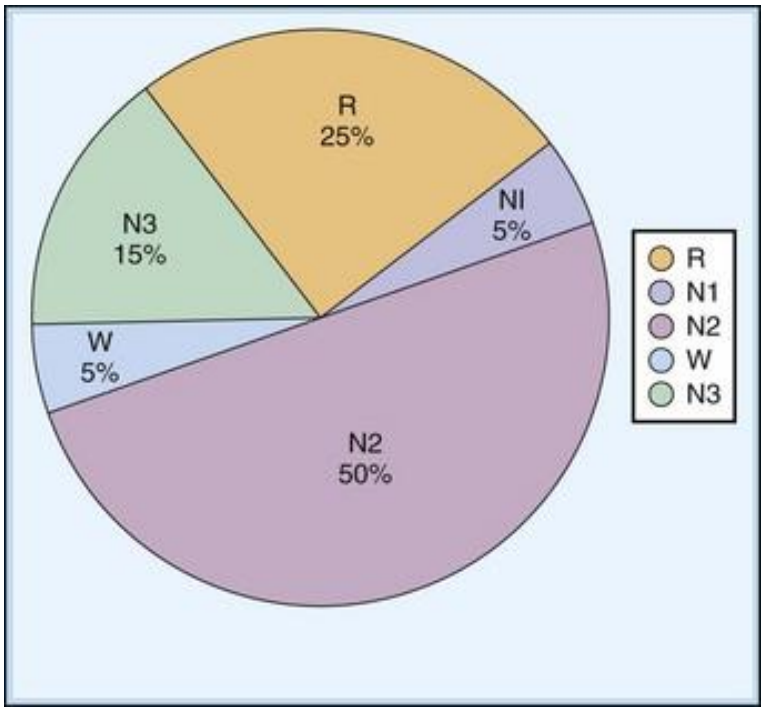

Рисунок 1.5 – Кругова діаграма тривалості стадій сну [11].

Для вирішення такої проблеми існують два способи:

1. Модифікація набору даних. Включає в себе копіювання прикладів недостатньо представлених класів або видалення прикладів переважаючого класу, генерація нових прикладів на основі існуючих.

2. Модифікація алгоритму навчання. Зазвичай, використовують зважену функцію втрат, причому для недостатньо представлених класів ваги повинні бути більшими, ніж ваги переважаючих класів.

1.4.2 Методи oversampling і undersampling

Метод oversampling, відповідно до назви полягає в тому, що випадковим чином обирають певну кількість прикладів недостатньо представленого класу і дублюють їх. Це забезпечує механізм для зміни ступеня балансу розподілу класів до будь-якого бажаного рівня. Головний недолік цього методу полягає в тому, що копіювання існуючих прикладів не збільшує інформації про недостатньо представлений клас, тому це може призвести до того, що класифікатор запам’ятає шуми і не зможе узагальнити дані [17].

Відповідно до методу undersampling навпаки випадковим чином обирають певну кількість прикладів переважаючого класу та видаляють їх. Це також дає простий метод коригування балансу вихідного набору даних. Головний недолік цього методу відносно очевидний: вилучення прикладів з переважаючого класу може призвести до того, що класифікатор може пропустити важливі особливості, що належать до цього класу [17].

1.4.3 SMOTE

Доповнення даних (Data Augmentation) – процес створення нових прикладів на основі існуючих даних. Доповнення даних використовують як техніку зменшення перенавчання класифікатора, так і для вирішення проблеми незбалансованого набору даних. В залежності від природи задачі, яка вирішуються (розпізнавання зображень, NLP (natural language processing)) існують різні методи.

SMOTE розшифровується як Synthetic Minority Oversampling Technique, і в якомусь сенсі є модифікацією методу oversampling, але замість простого дублювання існуючих прикладів, створюють нові приклади на основі існуючих,

що може дати недостатньо представленому класу більше простору для узагальнення та уникнути проблеми запам’ятовування шумів.

Для двомірних даних спочатку знаходять N найближчих сусідів у класі меншин для кожної з вибірок у класі, де N – це параметр. Потім проводять лінію між сусідами і генерують випадкові точки на цих лініях [18]. В модифікаціях SMOTE до отриманої точки також додають шум, для того щоб зробити нові приклади більш реалістичними.

1.5 Аналіз існуючих підходів

Автоматизація оцінки стадій сну досліджується вже не перший рік та досі викликає багато інтересу. У більшості робіт використовують лише сигнали ЕЕГ та ЕОГ, іноді включають дані ЕМГ. Сучасні підходи можна розділити на дві категорії в залежності від того, чи використовуються для класифікації ознаки, отримані на основі експертних знань [19-20], чи вони витягуються з необроблених сигналів [21-22].

Методи першої категорії ґрунтуються на апріорних знаннях про сигнали, що дає змогу власноруч виділяти ознаки. Кожну таку ознаку можна віднести до однієї з категорій: часові ознаки, частотні ознаки, ентропія та нелінійні ознаки. У таблиці 1 наведено перелік основних ознак, що стосуються ЕЕГ сигналу. Для ЕОГ сигналу можна обчислювати часові статистичні ознаки, ентропію та нелінійні ознаки.

Наступним кроком є визначення важливості ознак для формування найоптимальнішого набору даних. Використовують як традиційні критерій хі-квадрат та тест Фішера, що базуються на розподілі даних, так і більш специфічні методи, наприклад, RelieF. RelieF використовується для оцінювання важливості ознак шляхом вимірювання їх внеску у класифікацію кожного зразка даних. Він

працює на основі ідеї "розбалансованості" - важливі ознаки мають різницю у значеннях між найближчими сусідами, що належать до різних класів.

У якості класифікатора раніше поширеним було використання алгоритму найближчого сусіда, багатошарового персептрона (MLP), або модифікацію методу опорних векторів – Dendrogram based SVM. Більш сучасним є використання градієнтного бустинга з деревами рішень як базовою моделлю, перевагою цього методу є можливість використання дерев рішень для визначення важливості ознак.

Таблиця 1.1 – Перелік ознак, які містить сигнал ЕЕГ

Часові ознаки	Частотні	Ентропія	Нелінійні ознаки
Статистичні ознаки (мін. значення, макс. значення, середнє значення, медіана, середньоквадратичне відхилення, коефіцієнти асиметрії та ексцесу), Zero-crossing Rate, параметри Хьорта	Потужність ЕЕГ сигналу в $\alpha, \beta, \theta, \delta, \sigma$ діапазонах	Наближена ентропія, спектральна ентропія, ентропія вибірки	Фрактальні розмірності Хігучі, Петросяна та Катца

Перевагою методів цієї категорії є можливість вибору ознак, на основі яких класифікатор буде вивчати залежність між сигналами ПСГ та стадіями сну, оскільки це особливо корисно у медицини, де лікарі хочуть мати можливість пояснити та обґрунтувати результати класифікації. Тоді ж як головним недоліком є необхідність наявності знань у сферах обробки сигналів та медицини сну. Вибір правильних ознак може бути складною задачею, адже існує велика кількість характеристик сигналу.

Методи другої категорії полягають у навчанні відповідних представлень ознак з перетворених або безпосередньо з необроблених даних за допомогою згорткових нейронних мереж (CNN). Не дивлячись на те, що найбільш широкого

поширення CNN знайшли у задачах аналізу зображень, проте їх нерідко використовують для обробки сигналів. Одна з головних переваг CNN полягає у їх здатності автоматично виявляти локальні шаблони або ознаки у просторових даних без необхідності вручну визначати ці шаблони. Вони також можуть враховувати локальні контексти та ієрархічні залежності між ознаками, що є важливими у багатьох задачах обробки сигналів. Наприклад, згорткові шари можуть виявляти важливі частотні характеристики ЕЕГ, такі як альфа, бета, тета, дельта-ритми. Недоліком є складність інтерпретації рішень моделі, а також чутливість CNN до якості даних, особливо до шуму, артефактів та інших спотворень в сигналі ЕЕГ.

1.5 Постановка задачі

Задача класифікації стадій сну є комплексною та потребує декількох основних кроків для її вирішення. Основною проблемою є те, що це задача незбалансованої багатокласової класифікації, тобто до кожної стадії сну відноситься непропорційна кількість прикладів. Тому необхідно знайти залежність між вхідними даними, які є записами сигналів ЕЕГ та ЕОГ протягом сну.

Отже, першим кроком є пошук та первинний огляд набору даних. Побудова візуальних представлень розподілу даних, а саме співвідношення прикладів кожної стадії сну N1, N2, N3, R та W.

Наступний крок це огляд типових підходів, що застосовуються для вирішення даної задачі. В результаті роботи маємо порівняти кожен з них та виділити їхні переваги та недоліки.

Третій крок – це програмна реалізація обраних методів, що включає в себе побудову базових моделей, підбір гіперпараметрів та перевірку якості роботи моделей на тестовому піднаборі даних. Для ефективної оцінки моделей

необхідно підібрати метрики, що гарно відображають рівень розпізнавання кожного з класів, оскільки це є важливим аспектом в задачах незбалансованої багатокласової класифікації.

Останнім кроком необхідно дослідити способи збалансування навчальної вибірки даних із використанням основних підходів – oversampling та undersampling.

1.6 Висновки до розділу 1

В першому розділі було доведено актуальності теми роботи, детально розглянуто предметну сферу: визначено структуру вхідних даних, особливості кожної стадії сну, цикл сну здорової людини, дано означення гіпнограми.

Було розглянуто проблеми задачі: наявність артефактів в записах сигналів ЕЕГ, незбалансованість кількості стадій сну та розглянуто методи боротьби з цими проблемами.

Також розглянуто існуючі підходи для вирішення задачі. Загалом можна виділити два підходи, перший базується на апріорних знаннях про сигнали та вилученні ознак, тоді як другий полягає в автоматизації пошуку ознак із застосуванням CNN. Описані основні ознаки, що включають в себе як ознаки, на які орієнтуються експерти при оцінці фаз сну, так і ознаки, що стосуються природи біологічних сигналів, наприклад, фрактальна розмірність, яка вивчається такою наукою як синергетика.

В кінці було описано постановку задачі та описано кроки, які необхідно виконати для досягнення мети.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Деякі ознаки сигналів ПСГ

Існує безліч характеристик, що можна виділити з ЕЕГ сигналу, але найбільш вживані перелічено в таблиці 1.1. Вони стосуються часових, частотних, ентропійних та нелінійних областей природи сигналу.

Частота перетину нуля (ZCR) - це показник частоти, з якою сигнал змінюється з позитивного на нульовий або з негативного на нульовий або з нульового на позитивний. Іноді враховують лише кількість разів, коли сигнал змінив знак.

Цей показник широко використовується як у розпізнаванні мови, так і є характеристикою в музичній інформації, будучи ключовою ознакою для класифікації ударних звуків. Його часто застосовують і до ЕЕГ сигналів.

2.1.1 Ознаки Хьорта

Параметри Хьорта (Hjorth) – це показники статистичних властивостей, що використовуються в обробці сигналів у часовій області, введені Бо Хьортом у 1970 році. Параметрами є активність, рухливість і складність.

Активність обчислюється як дисперсія (середньоквадратичне відхилення) ЕЕГ сигналу.

Параметр рухливості являє собою середню частоту або частку середньоквадратичного відхилення спектра потужності. Він визначається як квадратний корінь з дисперсії першої похідної від x, поділений на дисперсію x.

Складність дає оцінку смуги пропускання сигналу, яка вказує на подібність форми сигналу до чистої синусоїди (де значення сходиться до 1). Складність визначається як відношення рухливості першої похідної від x до рухливості x.

2.1.2 Потужність сигналу ЕЕГ у визначених діапазонах

Як вже було зазначено, вчені виділили різні частотні діапазони сигналів мозку, такі як альфа, бета, дельта, тета та сігма, які пов'язані з різними стадіями сну. В таблиці 2.1 вказано відповідність частотного діапазону та стадії сну. Як бачимо, стадіям N1 та REM відповідають схожі частотні діапазони, що може призвести до складнощів під час побудови моделі. В деяких роботах ці дві стадії об’єднують в одну.

Таблиця 2.1 Зв’язок між стадіями сну та сигналами мозку

Частотний діапазон / Стадія сну	δ (0.5 − 4.5 Гц)	θ (4.5 − 8.5 Гц)	α (8.5 − 11.5 Гц)	σ (11.5 − 15.5 Гц)	β (15.5 − 30 Гц)
N1		x			x
N2				x	
N3	x				
REM		x			x
W			x		

Тому для задачі класифікації стадій сну потужність ЕЕГ сигналу в кожному з частотних діапазонів є важливою характеристикою.

Першим кроком для її обчислення є пошук спектральної густини потужності (PSD) сигналу, наприклад, за допомогою метода Уелча.

Нехай $X(j), j = 0, \dots, N-1$ – вхідний сигнал довжиною N. Розіб’ємо його на відрізки довжиною L з можливістю їх перетину, початки цих відрізків лежать на відстані D одиниць один від одного. Нехай $X_1(j), j = 0, \dots, L-1$ – перший такий відрізок. Тоді

$$X_K(j) = X(j + (K-1)D)\ j = 0, \dots, L-1 \qquad (2.1)$$

Припустимо, що існує K таких сегментів і вони покривають весь вхідний сигнал, тоді виконуватиметься рівняння $(K-1)D+L=N$. Метод оцінки полягає в наступному. Для кожного сегмента обчислюється модифікована періодограма. Спочатку обирається вікно даних $W(j), j=0,\dots,L-1$ і формуються послідовності $X_1(j)W(j),\dots,X_k(j)W(j)$. Після чого формуються скінченні перетворення Фур'є $A_1(n),\dots,A_k(n)$ цих послідовностей:

$$A_k(n)=\frac{1}{L}\sum_{j=0}^{L-1}X_k(j)W(j)e^{-2kijn/L}, \tag{2.2}$$

де $i=\sqrt{-1}$.

В результаті знаходимоK модифікованих періодограм:

$$I_k(f_n)=\frac{L}{U}|A_k(n)|^2, \tag{2.3}$$

де $f_n=\frac{n}{L}, n=0,\dots,\frac{L}{2}$ i $U=\frac{1}{L}\sum_{j=0}^{L-1}W^2(j)$.

Спектральна оцінка є середнім значенням цих періодограм (формула 2.4).

$$\hat{P}(f_n)=\frac{L}{K}\sum_{k=0}^{K}I_k(f_n), \tag{2.4}$$

За вікно можна обрати вікно Ганна (формула 2.5).

$$w(n)=0.5\left(1-\cos\left(2\pi\frac{n}{L-1}\right)\right), 0\le n\le L-1 \tag{2.5}$$

Отже, в результаті було отримано значення PSD в залежності від частоти (рисунок 2.1).

Рисунок 2.1 – Приклад графіку залежності PSD від частоти

Тобто середня потужність сигналу у певному частотному діапазоні буде дорівнювати площі під кривою графіка. Для того, щоб знайти середню потужність у певному частотному діапазоні потрібно проінтегрувати PSD в цьому діапазоні. Для цього можна використати формулу Сімпсона (формула 2.6).

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right), \tag{2.6}$$

де $f(x)$ – певна функція, a, b – початок і кінець інтервалу інтегрування відповідно.

Потужність діапазону можна нормувати, поділивши її на загальну потужність вхідного сигналу.

2.1.3 Ентропія

Ентропія є мірою ступеня невизначеності сигналу. Як поняття вона вперше виникла в галузі термодинаміки, і типова фізична інтерпретація ентропії – це невпорядкованість системи, що описується ймовірностями розподілу молекул газоподібних або рідких систем. Шеннон ввів це поняття у сферу теорії інформації і визначив те, що зазвичай називають статистичною ентропією (формула 2.1).

$$H = -\sum_{m=1}^{N} P(x)\log_2 P(x), \tag{2.7}$$

де $P(x)$ – ймовірнісний розподіл величини x.

Поняття ентропії набуло поширення і в оцінці нерегулярності або невизначеності в біологічних сигналах, таких як ЕЕГ, ЕОГ.

Існує декілька підходів для вимірювання ентропії сигналу. Це може бути зроблено або в часовій, або в частотній області.

Якщо казати про часову область, то прикладом можуть слугувати наближена ентропія та ентропія вибірки. Вони кількісно оцінюють повторюваність форми сигналу ЕЕГ.

Нехай маємо вхідний сигнал x довжиною N, його необхідно розбити на множину векторів довжиною m і аналогічно $m+1$. Введемо пороговий параметр $r = 0.2 * std(x)$, який контролює чутливість до різниці між шаблонами. Підрахуємо $C(m,r)$ – усереднена кількість векторів, що мають довжину m та відстань до найближчого сусіда менша за r, аналогічно підраховуємо $C(m+1,r)$ для векторів довжиною $m+1$. Тоді ентропія вибірки обчислюється за формулою 2.3

$$H(x,m,r) = \frac{1}{N-m+1}\log C(m,r) - \frac{1}{N-m}\log C(m+1,r), \quad (2.8)$$

$$H(x,m,r) = -\log\frac{C(m,r)}{C(m+1,r)}. \quad (2.9)$$

Менші їх значення вказують на повторюваність сигналу, а більші - на нерегулярність.

Спектральна ентропія є характеристикою частоти сигналу. Вона визначається як ентропія Шеннона (формула 2.7) спектральної густини потужності (PSD) вхідного сигналу.

2.1.4 Фрактальна розмірність

Фрактали - це математичні множини з високим ступенем геометричної складності, які можуть моделювати багато природних явищ. Численні природні процеси можна описати за допомогою часових рядів вимірювань, які є фракталами. Наприклад, економетричні та демографічні дані, варіації висоти тону в музичних сигналах, сигнали ЕЕГ та ЕКГ. Дуже важливою характеристикою фракталів, корисною для їх опису та класифікації, є їх фрактальна розмірність (FD), яка вимірює ступінь фрагментації або нерегулярності їхніх границь за кількома масштаби

Використання FD дозволяє оцінити складність або наповненість простору сигналу на різних масштабах і виявити внутрішню структуру сигналу.

Існує не один спосіб обчислення FD, у роботах, що стосуються біологічних сигналів найбільш вживаними є формули Хігучі, Каца (Katz) і Петросяна.

Обчислення FD методом Хігучі було розроблено спеціально для застосування в задачах, що стосуються часових рядів. Нехай дано сигнал X

довжиною N та параметр $k_{max} \geq 2$. Для кожного $k \in \{1, \dots, k_{max}\}$ і $m \in \{1, \dots, k\}$ визначається довжина $L_m(k)$:

$$L_m(k) = \frac{N-1}{\left\lfloor \frac{N-m}{k} \right\rfloor k^2} \sum_{i=1}^{\left\lfloor \frac{N-m}{k} \right\rfloor} |X_N(m+ik) - X_N(m+(i-1)k)|. \quad (2.10)$$

Довжина $L(k)$ визначається як середнє значення $L_1(k), \dots, L_k(k)$. Нахил найкращої лінійної функції, що проходить через точки даних $\left\{\left(\log \frac{1}{k}, \log L(k)\right)\right\}$ визначено як FD Хігучі часового ряду X.

Згідно методу Каца FD розраховується наступним чином: обчислюється сума та середнє значення евклідових відстаней між послідовними точками вибірки (позначимо їх через L та a відповідно), а також максимальна відстань між першою точкою та будь-якою іншою точкою вибірки (d). Тоді FD вибірки дорівнює:

$$FD_k = \frac{\log_{10}(L/a)}{\log_{10}(d/a)} = \frac{\log_{10} n}{\log_{10}(d/L) + \log_{10} n}. \quad (2.11)$$

Формулу Петросяна було запропоновано спеціально для застосування до ЕЕГ сигналів. Для заданого сигналу x довжиною N, FD обчислюється за формулою:

$$FD_k = \frac{\log_{10} N}{\log_{10}\left(\frac{N}{N + 0.4N_\delta}\right) + \log_{10} N}, \quad (2.12)$$

де N_δ – кількість змін знаків у похідній сигналу.

FD може бути корисною у задачі класифікації стадій сну. Основна ідея полягає в тому, що розмірність фракталу може відрізнятися для різних стадій сну

або неспання, що дозволяє використовувати цю метрику як ознаку для класифікації.

Дослідження показують, що FD може бути вищою під час стадії REM і нижчою під час стадії N3. Це пов'язано зі змінами в структурі мозкової активності під час різних стадій сну.

2.2 Нейрон

Базовими будівними блоками нейронних мереж є нейрони. Вони імітують функціонування біологічних нейронів і грають важливу роль у обробці інформації та прийнятті рішень моделями машинного навчання. Найпоширенішою моделлю штучного нейрона є модель МакКаллока-Піттса. Відповідно неї штучний нейрон складається з кількох основних компонентів:

1. Вхідні з'єднання: Кожен штучний нейрон має вхідні з'єднання, через які отримує вхідні сигнали або значення з попередніх штучних нейронів.

2. Ваги: Кожне вхідне з'єднання має вагу, яка відображає його значущість або вплив на вихідний сигнал штучного нейрона.

3. Функція активації: Функція активації приймає зважену суму вхідних сигналів і визначає вихідний сигнал штучного нейрона. Функція активації має бути нелінійною для того, щоб нейронна мережа була здатна узагальнювати нелінійні дані.

4. Вихідний сигнал: Вихідний сигнал штучного нейрона є результатом обчислень, здійснених функцією активації.

Нехай X – вектор вхідних даних довжиною m, W – вектор ваг довжиною m, $f(x)$ – функція активації. Формула 2.13 відповідає матричному обчисленню вихідного сигналу y.

$$y = f(w_0 + X^T W) \tag{2.13}$$

де w_0 – вага з’єднання що відповідає нейрону зсуву (bias).

Нейрон зсуву - це додатковий компонент у штучних нейронних мережах. Це особливий тип нейрона, який завжди активний і забезпечує постійний вхід для інших нейронів у мережі. На відміну від звичайних вхідних нейронів, які отримують вхідні дані з даних, нейрон зміщення не має вхідних зв'язків і лише видає постійне значення.

Мета нейрона зсуву – ввести константне значення, яке можна вивчити і яке може змістити функцію активації інших нейронів у мережі. Завдяки додаванню терму зміщення нейронна мережа стає здатною навчатися і представляти більш складні закономірності та взаємозв'язки в даних.

Нейрон зміщення допомагає коригувати межі прийняття рішень нейронною мережею, забезпечуючи певний рівень гнучкості моделі.

Існують багато видів функцій активації.

Одна з найперших функцій активації - порогова функція активації, вона визначається за формулою 2.14.

$$f(z) = \begin{cases} 1, z \geq 0 \\ -1, z < 0 \end{cases}, \tag{2.14}$$

де z – зважена сума вхідних сигналів.

Вона має ряд недоліків. По-перше, порогова функція активації є ступеневою функцією, що означає, що вихідний сигнал нейрона різко змінюється при переході через порогове значення. Це може створювати проблеми при градієнтному спуску та навчанні моделі, оскільки немає гладкої перехідної зони, де градієнт може змінюватись плавно. Це може призводити до проблем зі швидкістю збіжності та якістю навчання моделі. По-друге, порогова функція активації є неперервною лише на певних значеннях вхідного сигналу, в той час як вона має розриви в інших точках. Це може бути проблемою при використанні алгоритмів оптимізації, які вимагають похідних функцій для

оновлення ваг. У точках розривів порогової функції похідна невизначена або нульова, що ускладнює оптимізацію і може призвести до стійкості або проблем з навчанням моделі. По-третє, порогова функція активації просто визначає, чи перевищує вихідний сигнал нейрона порігове значення. Вона не ураховує інтенсивність або силу вхідного сигналу, що може бути корисною інформацією для моделі.

Протягом довго часу використовувалася сигмоїдальна функція активації [23], яка визначається за формулою 2.15.

$$f(z) = \frac{1}{1 + e^{-z}}, \tag{2.15}$$

де z – зважена сума вхідних сигналів.

Порівняно з пороговою функцією активації, сигмоїдальна функція активації є гладкою та неперервною функцією, що забезпечує плавні переходи між значеннями. Це дозволяє градієнту змінюватись плавно, що полегшує процес навчання штучної нейронної мережі. Гладка природа сигмоїдальної функції дозволяє моделі більш точно апроксимувати нелінійні залежності між вхідними та вихідними сигналами. Також сигмоїдальна функція приймає значення в діапазоні від 0 до 1, що відповідає ймовірностям або ймовірнісним вагам. Це корисно для моделей, де потрібно враховувати ступінь упевненості або ймовірність відповідей. Сигмоїдальна функція дозволяє моделі виражати вихідні значення від "повністю активованого" (1) до "повністю неактивованого" (0) стану, що дає більш гнучкий інтерпретаційний простір для моделі.

Серед недоліків цієї функції активації можна виділити схильність до насичення: у сигмоїдальній функції активації, коли вхідний сигнал наближається до межі 0 або 1, похідна функції стає дуже мала. Це призводить до проблеми відсутності градієнту для ваг, що знаходяться у насичених областях. В результаті, навчання може сповільнюватись або зупинятись, особливо в глибоких нейронних мережах. Це явище відоме як проблема зникновення

градієнту. Ще один недолік: відсутність центру навколо нуля. Сигмоїдальна функція активації не є симетричною відносно нуля, оскільки вона переходить з значення 0 до 1. Це може призводити до невідповідностей у функціонуванні нейронної мережі, особливо при використанні алгоритмів, які вимагають симетрії, таких як градієнтний спуск. Це також може вплинути на здатність моделі до навчання швидких та ефективних представлень.

Tanh (гіперболічний тангенс) визначається за формулою 2.16 [23].

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.16}$$

де z – зважена сума вхідних сигналів.

Головна перевага полягає в тому, що tanh знаходиться в діапазоні від -1 до 1, що дозволяє моделювати швидкі зміни в активації нейрона. Це особливо корисно в нейронних мережах, де потрібно враховувати як позитивні, так і негативні ваги. Головним недоліком функції, як і у випадку сигмоїдальної функції, є її насиченість. У межах великих вхідних значень (як позитивних, так і негативних), функція має насичену виведену активацію, що означає, що похідна функції стає дуже мала. Це може призводити до проблеми відсутності зміни градієнту під час навчання нейронної мережі, особливо в глибоких архітектурах. Цей недолік може сповільнювати процес навчання і ускладнювати оптимізацію ваг нейронної мережі.

ReLU (Rectified Linear Unit) визначається за формулою 2.17.

$$f(z) = \max(0, z) \tag{2.17}$$

де z – зважена сума вхідних сигналів.

Ця функція активації є однією з найуживаніших в сучасному світі. По-перше, вона проста та легка для обчислень. По-друге, оскільки ReLU не має насиченості, вона призводить до проблем зі зникненням градієнта. Однак,

головним недоліком функції активації ReLU є проблема "мертвих нейронів". Якщо ваги нейрона під час навчання вирівнюються так, що активація нейрона завжди буде від'ємною, то градієнт ReLU стає нульовим, і нейрон стає неактивним для всіх вхідних даних. Для подолання цієї проблеми були розроблені для подолання цієї проблеми, наприклад, Leaky ReLU (формула 2.18).

$$f(z) = \max(0.1z, z) \tag{2.18}$$

де z – зважена сума вхідних сигналів.

Функція активацій Softmax визначається за формулою 2.19.

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{2.19}$$

де z – зважена сума вхідних сигналів для i −го нейрону, N – кількість нейронів в поточному шарі.

Функція softmax перетворює вихідні значення нейронів в ймовірності, які відображають вірогідність належності вхідного сигналу до кожного класу. Це дозволяє використовувати softmax для вихідного шару нейронної мережі у задачах класифікації, де потрібно визначити ймовірність належності вхідного зразка до кожного класу.

2.3 Багатошаровий повнозв’язний персептрон

Багатошаровий повнозв'язний персептрон (MLP) є одним з основних типів штучних нейронних мереж, який використовується для розв'язання різноманітних завдань машинного навчання. Його архітектура складається з кількох шарів нейронів, які взаємодіють один з одним. Кожен шар містить набір

нейронів, а нейрони між шарами пов'язані ваговими коефіцієнтами. Основні складові архітектури MLP включають (рисунок 2.1):

• Вхідний шар. Це перший шар MLP, який отримує вхідні дані. Кількість нейронів у вхідному шарі залежить від розміру вхідних даних або ознак, що використовуються для навчання.

• Приховані шари MLP може мати один або більше прихованих шарів, розташованих між вхідним і вихідним шарами. Кількість шарів і кількість нейронів у кожному шарі можуть бути налаштовані в залежності від завдання та складності проблеми.

• Вихідний шар MLP відповідає результуючими значеннями нейронів після проходження вхідних даних через приховані шари. Кількість нейронів у вихідному шарі залежить від типу завдання: у випадку класифікації стадій сну ця цифра становить 5 нейронів. До нього зазвичай застосовують функцію активації softmax, адже вона природньо нормує вихідні сигнали таким чином, що кожен з них відображає ймовірність того, що поточний елемент належить до цього класу.

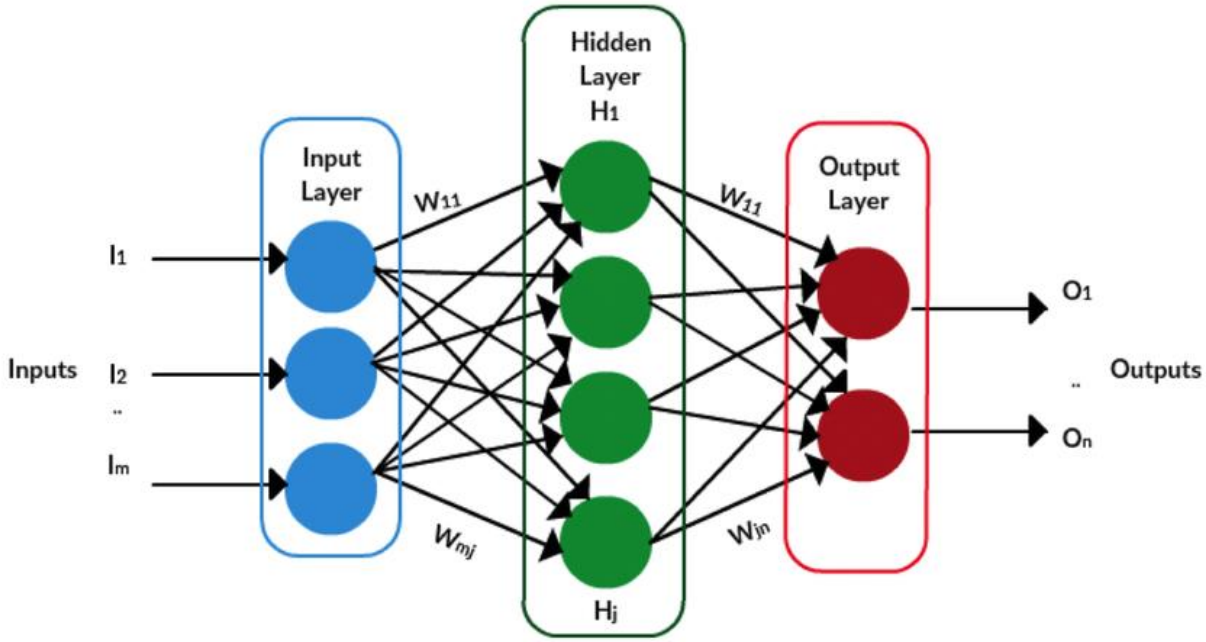

Рисунок 2.1 – Типова структура повнозв’язного персептрону [24]

2.4 Згорткова нейронна мережа

Автори алгоритму згорткової нейронної мережі (CNN) надихнулися біологічною структурою зорової системи ссавців, зокрема кори мозку, яка відповідає за обробку зображень.

Одним із ключових джерел натхнення була відкриття групою Девіда Габеля і Торстена Візеля в 1959 році, що в зоровій корі мозку існують так звані рецептивні поля, які сприймають і відповідають на різні структури вхідних зображень. Це дослідження показало, що зорова кора мозку має ієрархічну структуру, де нейрони першого рівня реагують на прості шаблони, а нейрони вищих рівнів сприймають більш складні структури (рисунок 2.2).

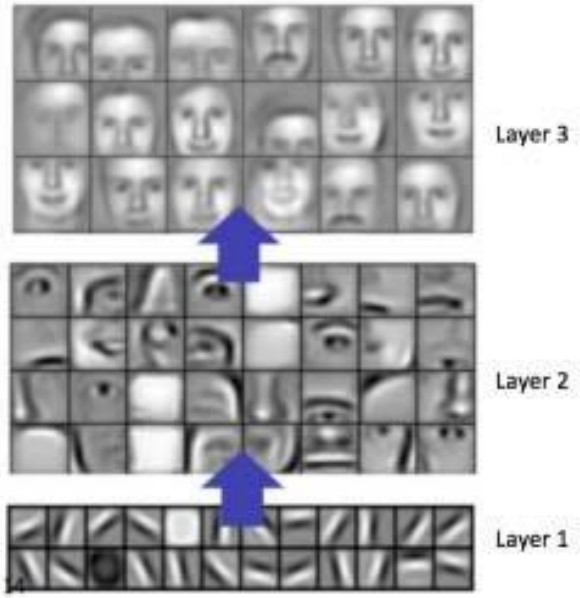

Рисунок 2.2 – Приклад складності ознак, які вивчає CNN на різних шарах [25]

В основі CNN лежить операція згортки. Оскільки вхідні дані задачі, що розглядається, є одновимірними, то наведемо одновимірну операцію згортки. Нехай дано вектор a довжиною n і вектор b довжиною m. Згортка $a * b$ визначається за формулою 2.20.

$$(a * b)(i) = \sum_{j=1}^{m} g_j \cdot f_{i-j+m/2} \quad (2.20)$$

де i – початок вікна операції згортки.

CNN працюють на основі фільтрів, або як їх ще називають - ядер, які займаються розпізнаванням ознак. Якщо у повнозв'язному персептроні ваги асоціювалися зі зв'язками між нейронами, то в конволюційних мережах за це відповідають фільтри. Тобто фільтр – це матриця ваг, у випадку одновимірних даних – вектор, її розмірність є параметром для налаштування. Фільтри здійснюють операцію згортки, переміщаючись по всьому вхідному простору. Таким чином, нейрони в будь-якому згортковому шарі не пов'язані з кожним окремим нейроном попереднього шару, як було у багатошаровому персептроні, а пов'язані тільки з нейронами у власних рецепторних полях. Це дає властивість розрідженості. Також завдяки цьому CNN зберігає структуру вхідних даних, порядок елементів, що є важливим для обробки сигналів.

Якщо деяка потрібна характеристика присутня у фрагменті даних, операція згортки на виході видаватиме число з відносно великим значенням. Якщо ж характеристика відсутня, вихідне число буде невеликим. Потім до цього числа застосовується нелінійна функція активації, як і у випадку MLP.

На кожному згортковому шарі може бути будь-яка кількість фільтрів.

Операція згортки зменшує розмір вхідні дані. Також елементи, які знаходяться на межі початку і кінця, беруть участь у меншій кількості операцій, ніж внутрішні. У зв'язку з цим у згорткових шарах може використовується доповнення даних (padding). Виходи з попереднього шару доповнюються додатковими елементами, значення яких зазвичай беруть рівним 0, так, щоб після згортки зберігся розмір вхідних даних.

Параметр крок (strides) в згорткових мережах визначає, з наскільки великим кроком фільтри будуть переміщатись по вхідних даних під час операції згортки. Коли крок дорівнює 1, фільтри переміщуються по одному елементу вліво, тобто згортка виконується на кожному елементі вхідних даних. Якщо ж

збільшити розмір кроку, фільтри переміщуються на кілька елементів. З одного боку, це зменшує розмір вихідного вектору у випадку одномірних даних та кількість згорткових операцій, зменшуючи обчислювальну складність. З іншого боку, це може призвести до втрати деякої локальної інформації.

В залежності від використання доповнення та параметру кроку буде змінюватися розмірність вихідних даних. Кількість параметрів також буде змінюватися від шару до шару. Для побудови архітектури CNN важливо розуміти ці розміри. За допомогою формули 2.21 можна обчислити розмірність вихідного вектору, за допомогою формули 2.22 – кількість параметрів у поточному шарі.

$$n_{out}^{i} = \frac{n_{in}^{i} - K_i + 2P_i}{S_i} + 1 \tag{2.21}$$

$$w_i = (K_i \cdot L_{i-1} + 1) \cdot L_i \tag{2.22}$$

де i – номер поточного шару, n_{out}^{i}, n_{in}^{i} – розміри вихідного та вхідного шару відповідно, K_i – розмір фільтру, P_i – доповнення, якщо воно використовується, то параметр приймає значення 1, якщо ні – 0, S_i- розмір кроку, w_i- кількість параметрів у поточному шарі, L_i, L_{i-1} – кількість фільтрів у поточному да попередньому шарі відповідно.

З метою прискорення процесу навчання та зменшення обчислювальних операцій використовують шари субдискретизації. Найуживаніший з них - максимальне об'єднання (max pooling).

Операція максимального об'єднання аналогічна операції згортки, тільки замість зваженої суми обирається просто максимальне число з локального рецепторного поля. Окрім оптимізації за кількістю обчислень, оскільки ваг цей шар не має, також він збільшує стійкість моделі, дає можливість концентруватися на дійсно вагомих ознаках зображення, відкидаючи несуттєві деталі.

Інші шари субдискретизації – середнє об’єднання, мінімальне об’єднання, відповідно до назви вони обирають або середнє, або мінімальне значення локального рецептурного поля.

Типова архітектура CNN складається з набору згорткових шарів, які "розбавляють" шарами субдискретизації, причому зі збільшенням глибини, збільшується кількість фільтрів, що використовують. Тобто, з одного боку, поступово зменшується розмір виходу у ширину, але він стає глибшим. У якийсь момент отриманий тензор перетворюють на вектор, після чого додають кілька повнозв'язних шарів.

2.5 Навчання штучних нейронних мереж

Навчити нейронну мережу у задачі класифікації означає підібрати значення ваг таким чином, щоб вони найкраще визначали клас для кожного прикладу вхідних даних. Для того, щоб зрозуміти наскільки гарно допасована модель до даних обчислюють значення функції втрат. Для задачі класифікації найуживанішою є категоріальна кросс-ентропія (формула 2.23). Вона вказує на відстань між тим, яким, на думку моделі, має бути вихідний розподіл, і тим, яким насправді є вихідний розподіл.

$$L(Y, W) = -\sum_{i=1}^{N}\sum_{j=1}^{C} t_i^{(j)} \log\left(p_i^{(j)}\right) \tag{2.23}$$

де N – кількість прикладів вхідного набору даних, C – кількість класів, $t_i^{(j)}$ – цільове значення класу j для прикладу i (приймає значення нуль або один), $p_i^{(j)}$ - значення імовірності, того що приклад i відноситься до класу j.

Навчання мережі можна поділити на два загальні етапи. Перший – прямий прохід, який полягає в обчислені поточних прогнозів мережі. Другий – зворотне поширення помилки.

На початку другого етапу обчислюють значення функції втрат. Далі, на основі отриманого значення функції витрат застосовують метод градієнтного спуску або його модифікації для обчислень внесків ваг в помилку на кожному шарі. Вся складність полягає в тому, що градієнт *i*-го шару залежить від значення градієнта *i+1*-го шару. Саме тому цей метод і отримав назву зворотного поширення помилки [26] .

Для обчислення часткових похідних використовують ланцюгове правило. Наприклад, для одношарового повнозв'язного персептрону часткова похідна ваг, що з'єднують вхідний шар з прихованим, буде обчислюватися за формулою 2.24.

$$\frac{\partial L(Y,W)}{\partial w_1} = \frac{\partial L(Y,W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}, \tag{2.24}$$

де $\hat{y}$ – вихідний сигнал нейронів 1-го шару, z_1 – зважена сума вхідних нейронів, w_i – ваги, що з'єднують вхідний і прихований шари мережі.

Після цього виконують етап градієнтного спуску.

Основна ідея градієнтного спуску полягає в ітеративному оновленні параметрів моделі в напрямку, протилежному до градієнта функції втрати. Градієнт функції втрати показує напрям, в якому функція найшвидше зростає. Таким чином, рухаючись в напрямку протилежному до градієнта, можна зменшити значення функції втрати та наблизитись до оптимальних параметрів моделі. Опис алгоритму градієнтного спуску:

1. Ініціалізувати початкові значення параметрів моделі.

2. Обчислити значення функції втрати відповідно до поточних параметрів моделі та тренувальних даних.

3. Обчислити градієнт функції втрати за кожним параметром моделі, використовуючи метод зворотнього поширення помилки.

4. Оновити значення параметрів моделі, перемістившись в напрямку, протилежному до градієнта, з використанням швидкості навчання (формула

2.25). Головні параметри, що впливають на роботу алгоритму градієнтного спуску, включають швидкість навчання, яка визначає крок оновлення параметрів, і критерій зупинки, який визначає умову зупинки процесу навчання.

$$W^{i+1} = W^i - \eta \cdot \frac{\partial L(Y, W)}{\partial W_i}, \tag{2.25}$$

де W^i, W^{i+1} – поточні та оновлені ваги відповідно, η – коефіцієнт швидкості навчання.

5. Повторювати кроки 2-4 до досягнення критерію зупинки, наприклад, заданої кількості ітерацій або досягнення необхідної точності.

Деякі модифікації градієнтного спуску включають стохастичний градієнтний спуск, міні-пакетний градієнтний спуск, адаптивні методи (Adagrad, Adam) та метод моментів та метод Нестерова.

Особливістю зворотнього поширення помилки в CNN полягає в тому, що для розповсюдження градієнтів через шари субдискретизації використовуються методи, такі як "розгортання" (unpooling) або "заповнення" (padding). Ці методи допомагають відновити просторову інформацію та передати градієнти до попереднього шару.

2.6 Методи регуляризації

Недонавчання виникає, коли модель машинного навчання недостатньо вивчає залежності в тренувальних даних і не може правильно узагальнювати їх на нові дані. В такому випадку модель недостатньо складна або недостатньо тренована для вирішення задачі. Для подолання цієї проблеми, зазвичай, достатньо ускладнити архітектуру моделі.

Перенавчання – явище, коли модель машинного навчання надмірно "запам'ятовує" тренувальні дані і недостатньо узагальнює отримані знання на нові невідомі дані. У такому випадку модель стає дуже специфічною для тренувальних даних, але мало придатною для нових даних, тобто значення міри якості на тренувальному наборі є набагато більше, ніж на валідаційному або тестовому.

Способи боротьби з перенавчанням:

• Dropout полягає в тому, що з деякою імовірністю p вихід кожного нейрону може замінитися на нуль, тобто він не буде брати участі в навчанні на поточній ітерації.

• Регуляризація за l1 (формула 2.26), l2 (формула 2.27) нормами. До значення функції втрат додатково додають штрафний термін, для того щоб змусити мережу узагальнити дані, а не завчити їх.

$$\hat{L}(Y,W) = L(Y,W) + \frac{1}{N}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}|w_{k,j}^{(l)}|, \tag{2.26}$$

$$\hat{L}(Y,W) = L(Y,W) + \frac{1}{N}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}(w_{k,j}^{(l)})^2 \tag{2.27}$$

• Рання зупинка полягає в тому, що ми перериваємо навчання після того, як деякий час наша модель не зменшувала значення функції витрат, або не покращувала точніть на валідаційному наборі.

2.7 Дерева рішень

Дерева рішень представляють собою модель, яка використовує ієрархічну структуру у вигляді дерева для прийняття рішень.

Кожне дерево рішень складається з вузлів та ребер. Корінь дерева знаходиться в самому верху і представляє початковий набір даних. Кожен вузол

в дереві відповідає певному розбиттю даних на підгрупи в залежності від значень певної ознаки. У внутрішніх вузлах дерева виконуються розбиття даних на основі різних ознак, а листки представляють кінцеві рішення або прогнози. Кожен листок може представляти певний клас у випадку задачі класифікації.

Процес побудови дерева рішень полягає у виборі найбільш інформативної ознаки для розбиття даних на кожному вузлі. Цей вибір зазвичай здійснюється на основі критеріїв, таких як індекс Джині або ентропія Шеннона. Але існують і певні модифікації. Наприклад, в популярному пакеті XGBoost використовується критерій розбиття, який називається "gradient-based split finding" або "loss-based split finding". Цей критерій базується на мінімізації функції втрат, який дозволяє ефективно визначати оптимальні поділи при побудові дерева.

У XGBoost, при побудові кожного вузла дерева, обчислюється градієнт функції втрат для кожного навчального прикладу в цьому вузлі. Потім виконується пошук найкращого поділу, шляхом порівняння градієнтів для різних можливих розділів. Для багатокласової класифікації в XGBoost використовується натуральний логарифм від функції softmax (формула 2.19).

Основні параметри дерев рішень включають в себе:

- Максимальна глибина – кількість рівнів у дереві рішень. Більша глибина дерева дозволяє моделі навчатись більш складним закономірностям в даних, але може призвести до перенавчання.
- Максимальна кількість листів – визначає мінімальну кількість прикладів, яка повинна бути у кожному листі дерева. Великі значення можуть запобігти перенавчанню, але можуть призвести до менш точних моделей.
- Мінімальна кількість прикладів у листі.
- Визначає мінімальну кількість прикладів, які повинні бути у кожному листі дерева. Великі значення можуть запобігти перенавчанню, але можуть призвести до менш точних моделей.
- Коефіцієнт гамма – оскільки вузол розбивається тільки тоді, коли результат розбиття дає позитивне зменшення функції втрат. Гамма задає мінімальне зменшення втрат, необхідне для розділення.

Перевага дерев рішень полягає в тому, що вони дозволяють обчислити важливість ознак без застосування додаткових алгоритмів. Важливість розраховується для одного дерева рішень за величиною, на яку кожен поділ певного атрибуту покращує показник ефективності (тобто наскільки зменшує значення функції втрат), зважений на кількість спостережень, за які відповідає вершина.

2.8 Градієнтний бустинг

Градієнтний бустинг (GB) – це потужний ансамблевий метод машинного навчання. Його концепція базується на понятті слабких учнів, тобто моделей якість яких трохи вища за 50%, але, якщо їх об’єднати, можна сформувати сильного учня. В даному алгоритмі слабкі учні поєднуються послідовно, таким чином, що кожна наступна модель намагається скоригувати помилки попередніх моделей.

Частина назви алгоритму «градієнтний» походить від того, що цільові результати для кожного випадку встановлюються на основі градієнта помилки відносно прогнозу. Кожна нова модель робить крок у напрямку, який мінімізує помилку передбачення.

Отже загальний алгоритм методу наступний:

1. Побудова початкової моделі.

2. Обчислення помилки (residual) між прогнозами початкової моделі і фактичними мітками класів. Ця помилка використовується як вихідний пункт для наступного кроку.

3. Побудова наступної моделі таким чином, щоб мінімізувати цю помилку. Для визначення напрямку і величини корекції використовують градієнтний спуск.

4. Нова модель додається до ансамблю попередніх моделей з певним коефіцієнтом, що контролює вагу моделі в ансамблі.

5. Процес повторюється, будуючи наступну модель з урахуванням помилок, залишених попередніми моделями.

6. Кінцевий прогноз обчислюється із застосуванням концепції логістичної регресії. GB використовує логарифм відношення шансів (log-odds) для прогнозування,

Ключовими параметрами для налаштування градієнтного бустингу є кількість учнів, коефіцієнт навчання та мінімальна вага учня.

2.9 Висновки до розділу 2

В другому розділі спочатку було розглянуто ключові ознаки, які можна вилучити з сигналів ПСГ

Описано структуру нейрону, розглянуто різні функції активації, визначено їх переваги та недоліки. Далі було описано повнозв’язні шари. Також дано визначення операції згортки, описано CNN та їх ключові параметри: розмір фільтру, кількість фільтрів, розмір кроку та доповнення. Наведено приклади шарів субдискретизації та їх роль у згорткових нейронних мережах. Було наведено алгоритм навчання мережі та визначено функцію втрат, яка буде використовуватися у роботі. Окремо було визначено проблему перенавчання мережі та способи її вирішення.

Остаточно, розглянуто алгоритм дерев рішень, в особливості – модифікації, які реалізовані в пакеті XGBoost, а також ансамблевий метод машинного навчання – градієнтний бустинг.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Середовище розробки

За мову програмування було обрано Python, адже він є потужним і гнучким інструментом для розробки проектів у сфері машинного навчання та забезпечує простоту використання, широкий вибір бібліотек та фреймворків, та велику підтримку спільноти розробників.

Середовище розробки – Google Colab. Його переваги включають в себе інтерактивність, що є особливо зручним при розробці моделі та аналізі результатів, а також доступність ресурсів – більший об’єм оперативної пам’ять та можливість використання графічного процесора під час навчання моделей.

Використані бібліотеки:

- NumPy та Pandas – для маніпуляцій з даними;
- matplotlib, seaborn – для створення візуалізацій;
- pyedflib – для роботи з файлами у розширенні .edf;
- scipy, antropy – для обробки сигналів та вилучення ознак;
- scikit-learn – для оцінки результатів роботи моделей;
- XGBoost, tensorflow – для побудови моделей градієнтного бустингу та нейронних мереж відповідно;
- imbalanced-learn – для балансування набору даних.

3.2 Огляд набору даних

Відсутність якісних відкритих наборів записів ПСГ довгий час стояло на заваді дослідження задачі автоматизації класифікації стадій сну методами машинного навчання. Навіть сьогодні їх кількість залишається дуже малою.

Одним з популярних наборів є Sleep-EDF Database Expanded. Він містить 153 записи ПСГ здорових людей, частота записів – 100 Гц. Наявні ЕЕГ сигнали з каналів Fpz-Cz та Pz-Oz, а також дані ЕОГ з горизонтального каналу. На рисунку 3.1 зображено розподіл записів за статтю та за віком. Отже присутні записи дорослих людей різних вікових категорій, кількість чоловіків і жінок є майже однаковою.

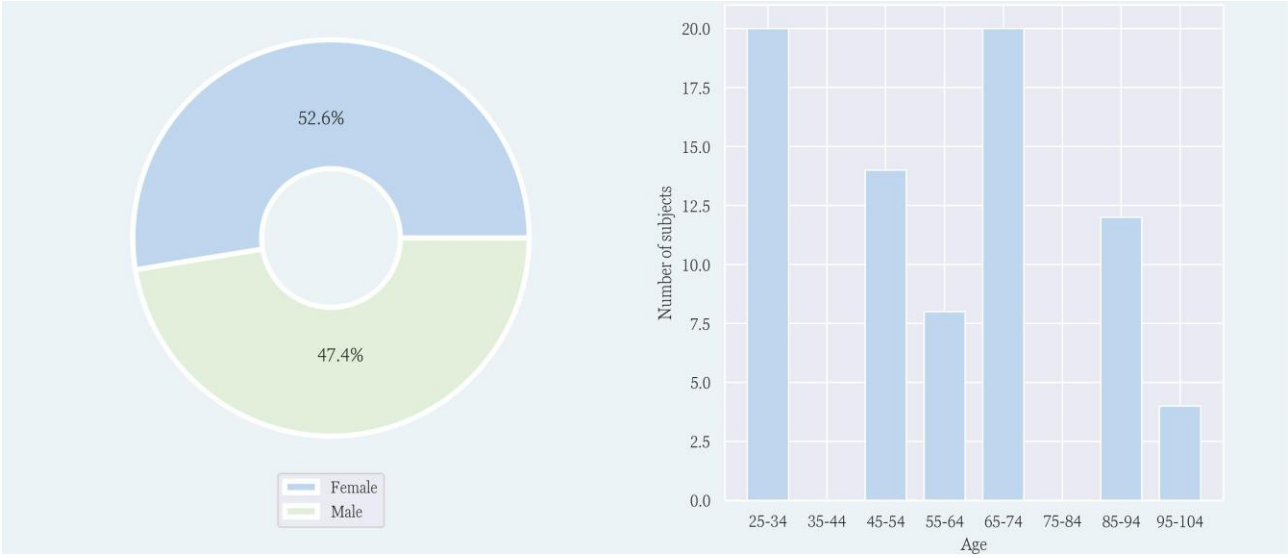

Рисунок 3.1 – Розподіл записів за статтю (зліва) та за віком (зправа)

Анотації до ПСГ файлів містили, окрім визначених 5 стадій, також невизначені стадії, які було позначено «?», і стадії позначені як рух. Такі анотації та відповідні їм часові проміжки ПСГ даних було видалено.

Особливістю цього набору даних є те, що ПСГ девайси були закріплені на пацієнтах протягом цілого дня, тому більшість часу займає стадія W. Було вирішено обрізати записи таким чином, щоб перед початком сну і після його кінця залишалася рівно 1 період стадії W.

Далі дані ПСГ було розбито на періоди довжиною 30 секунд, для кожного було визначено відповідну стадію сну. В результаті було отримано набір даних розмірністю $177010 \times 3000 \times 2$, тобто 177010 прикладів довжиною 3000 елементів для сигналів ЕЕГ з каналу Fpz-Cz та сигналів ЕОГ. На рисунку 3.2 зображено розподіл стадії цього набору даних.

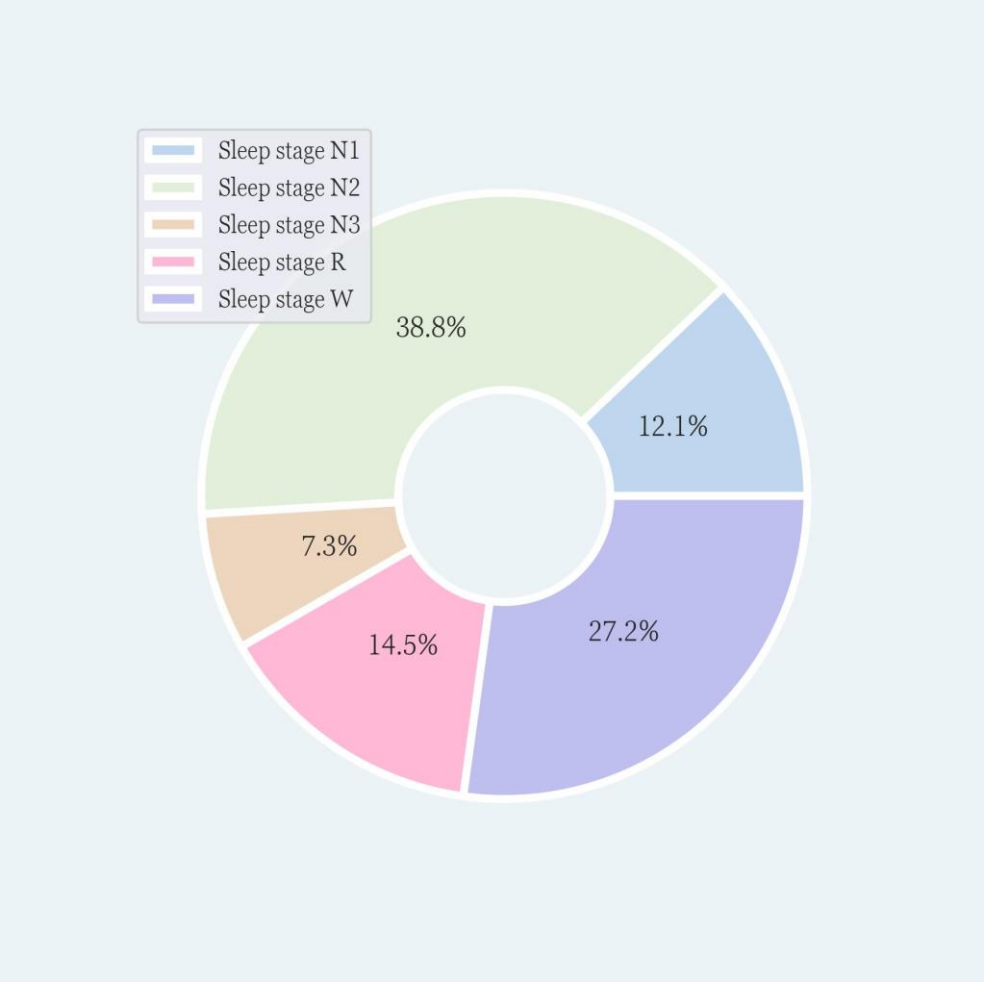

Рисунок 3.2 – розподіл набору даних за стадіями сну.

Файли з записами ПСГ було розбито на навчальну та тренувальну вибірки у пропорціях 0.85:0.15.

3.3 Обрані алгоритми

Відповідно до поставленої задачі – порівняння на практиці двох основних підходів для автоматизації оцінки фаз сну, необхідно визначити, які саме алгоритми та архітектури будуть використовуватися для побудови моделей.

Для підходу, що базується на вилученні ознак у якості класифікатора було обрано алгоритм градієнтного бустингу з деревами рішень як базовою моделлю,

а саме його реалізацію у пакеті XGBoost. XGBoost показує гарні результати в багатьох задачах машинного навчання, включаючи класифікацію. Однією з переваг GB є його здатність автоматично розпізнавати складні взаємозв'язки та нелінійність у даних. XGBoost підтримує роботу з числовими та категоріальними ознаками, включаючи розріджені дані. XGBoost також має вбудовані функції для регуляризації моделей, що допомагають уникнути перенавчання.

Другий підхід базується на використанні CNN для автоматичного вилучення ознак. На рисунку 3.3 зображено обрану архітектуру CNN [19]. Вона є достатньо типовою і простою за будовою і гарантує відносно швидке навчання. Шар виключення допоможе контролювати перенавчання моделі.

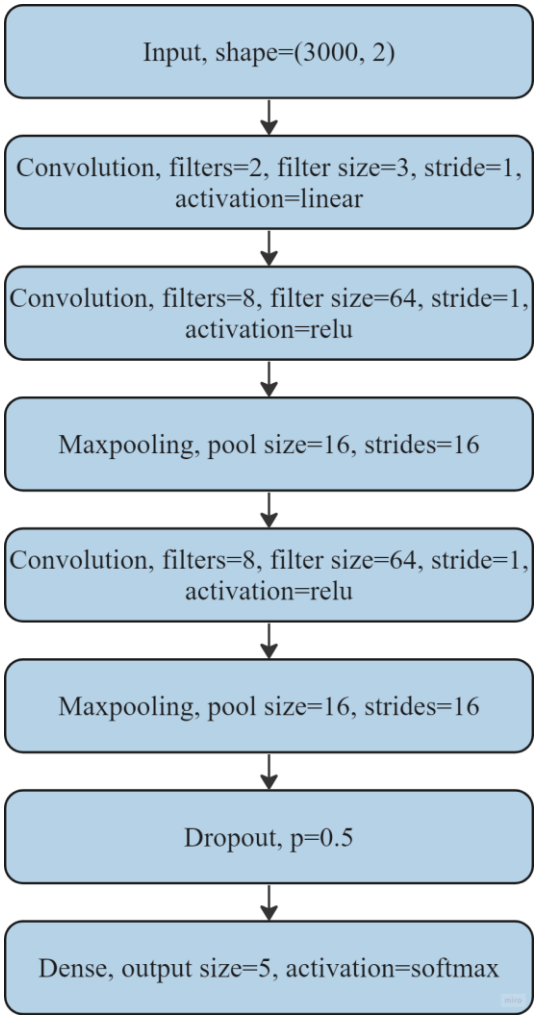

Рисунок 3.3 – Обрана архітектура CNN.

3.4 Міри якості

В задачах незбалансованої багатокласової класифікації важливим є правильне обрання метрик для оцінки якості моделей, адже поганий вибір може призвести до того, що значення міри якості не буде відповідати реальним результатам роботи моделі. В результаті класифікації прикладу можливий один з чотирьох варіантів:

- істинно позитивний (TP). Приклад було віднесено до потрібного класу;
- хибно позитивний (FP). Приклад було віднесено до певного класу, до якого він насправді не відноситься;
- істинно негативний (TN). Приклад не було віднесено до певного класу, і він до нього не належить;
- хибно негативний (FN). Приклад не було віднесено до певного класу, хоча він йому належить.

Для контролю рівня розпізнавання кожного класу буде використовуватися нормована матриця неточностей. Вона має форму $N \times N$, де N – кількість унікальних класів у задачі класифікації, в даному випадку $N = 5$. Рядки відповідають справжнім класам, стовпці – прогнозованим. В загальному випадку по діагоналі знаходяться значення TP, під діагоналлю FN, а над – FP. У випадку нормованої матриці неточностей виконується нормування по рядкам. Тоді діагональні значення цієї матриці буде відповідати точності (accuracy).

Як агреговану міру якості можна використовувати середню влучність (формула 3.1), середню повноту (формула 3.2) та середнє значення F1 (формула 3.3). Влучність вимірює, яка частка об'єктів, визначених моделлю як TP для певного класу, насправді належить до цього класу. Вона визначає, наскільки модель "точна" у визначенні позитивних випадків. Повнота вимірює, яка частка об’єктів класу була визначена правильно. Вона визначає, наскільки модель "повна" у визначенні позитивних випадків. F1-score – це гармонічне середнє між влучністю і повнотою. і використовується для збалансованої оцінки моделі.

$$Precision = \frac{TP}{TP + FP} \quad (3.1)$$

$$Recall = \frac{TP}{TP + FN} \quad (3.2)$$

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (3.3)$$

3.5 Аналіз отриманих результатів

3.5.1 Модель GB

Спочатку було побудовано базову модель GB з параметрами за замовчуванням, що пропонує пакет XGBoost. Агреговані метрики для цієї моделі наведені у таблиці 3.1. На рисунку 3.4 зображено нормовану матрицю неточностей. Різниця метрик на тренувальному та тестовому наборах становить більше ніж 10%, тому можна казати про суттєве перенавчання моделі. З матриці неточностей можна побачити, що точність розпізнавання класу N1 є дуже низькою.

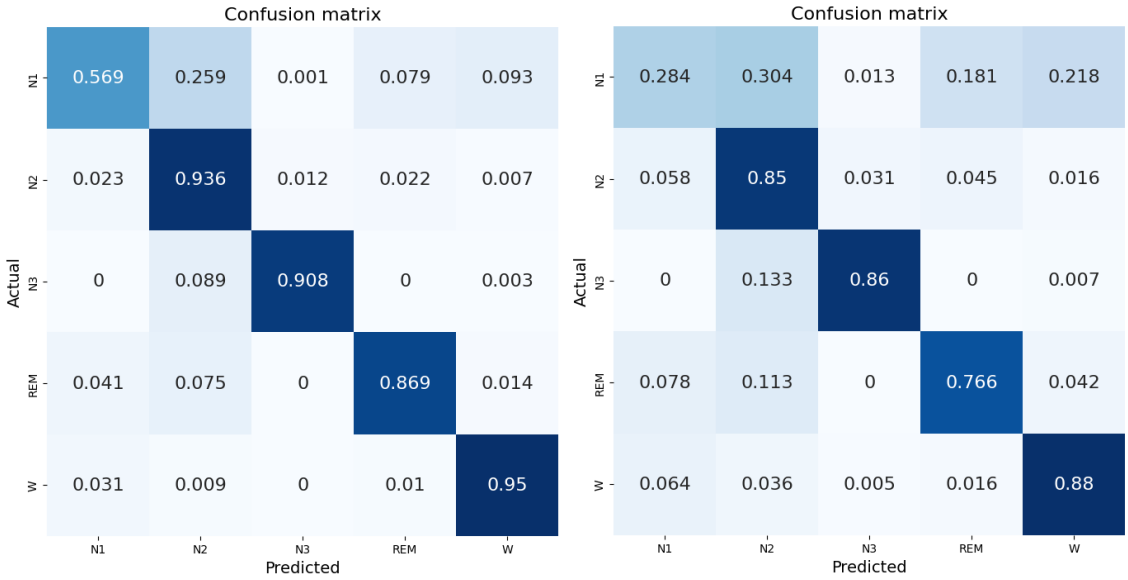

Рисунок 3.4 – Матриця неточностей базової моделі на тренувальному (зліва) та тестовому (справа) наборах

Для того, щоб зменшити перенавчання моделі необхідно підібрати гіперпараметри моделі, а саме її глибину дерев рішень і їх максимальну кількість листків, кількість учнів у ансамблі, мінімальну вагу нащадка та коефіцієнт гамма.

Використовувався решітчастий пошук, якість моделі перевірялася за допомогою 6-кратного перехресного затверджування (6-fold cross-validation). Матриця неточностей для найкращої моделі зображена на рисунку 3.5. Дивлячись на агреговані метрики (таблиця 3.1) можна зробити висновок, що тепер модель є менш перенавченою, адже на тестовому наборі результати майже ті ж самі, а різниця між значеннями мір якості на тренувальному наборі та тестовому впала майже на 5%.

Таблиця 3.1 – Результати навчання моделей GB

Модель	**Тренувальний набір**			**Тестовий набір**		
	Влучність	**Повнота**	**F1**	**Влучність**	**Повнота**	**F1**
Базова	87.16%	84.67%	85.89%	72.25%	72.79%	72.52%
З підібраними гіперпараметрами	80.62%	77.83%	79.20%	71.89%	72.00%	71.95%
З відібраними ознаками	79.13%	76.27%	77.67%	71.58%	71.72%	71.65%
Із застосуванням технік семплінгу	77.34%	76.73%	77.02%	72.24%	72.27%	72.25%

Наступний крок – перевірка важливості ознак. На рисунку 3.6 зображено 5 найбільш важливих та 5 найменш важливих ознак. Необхідно визначити при використанні якої кількості ознак якість моделі при перехресній перевірці буде найкраща. За початковий набір даних обираємо перші 10 найважливіших ознак. Далі поступово додаємо по одній ознаці відповідно до її рівня важливості. На рисунку 3.7 зображено графік зміни якості моделі під час додавання нових ознак.

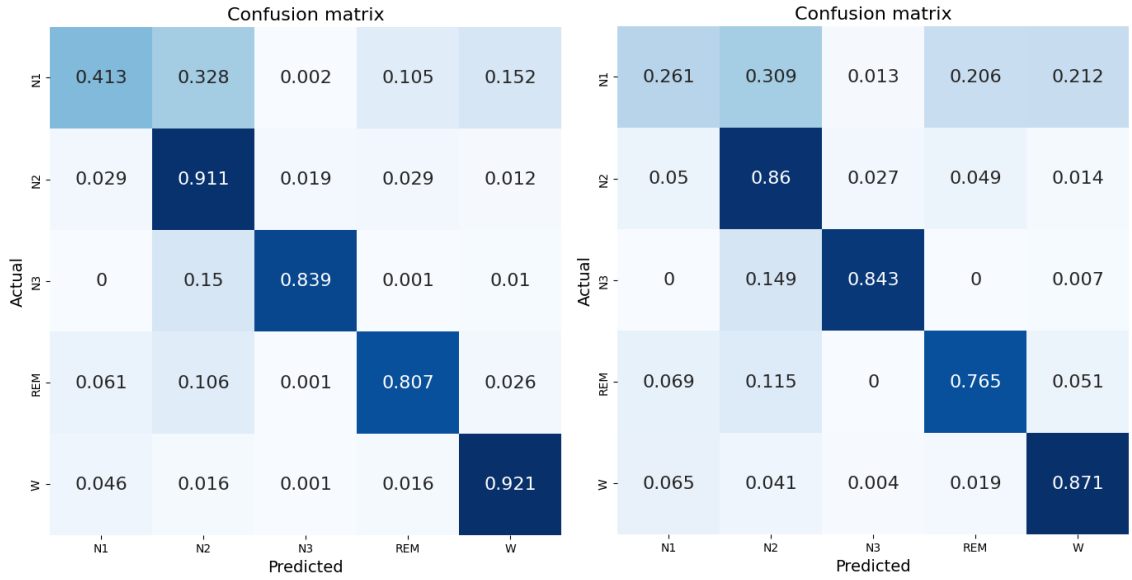

Рисунок 3.5 – Матриця неточностей моделі з підібраними гіперпараметрами на тренувальному (зліва) та тестовому (справа) наборах

	feature_name	importance
35	eog_katz_fd	0.187312
27	eog_max	0.139222
4	beta_bandpower	0.091646
29	eog_std	0.076947
1	theta_bandpower	0.036566

	feature_name	importance
12	eeg_mean	0.005051
23	eeg_num_zerocross	0.004946
19	eeg_katz_fd	0.004412
8	delta_ratio_2	0.004246
30	eog_skew	0.002788

Рисунок 3.6 – П’ять найважливіших (зліва) ознак та п’ять найменш важливих ознак (справа)

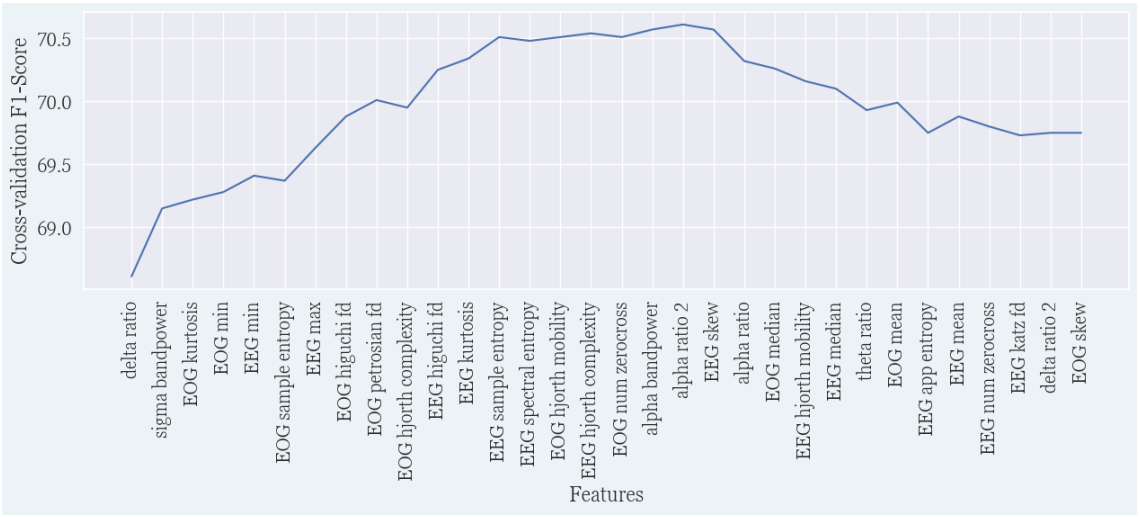

Рисунок 3.7 – Графік зміни якості моделей з додаванням нових ознак

Найоптимальніший набір даних складається з 30 найважливіших ознак. Матриця неточностей моделі, навченої на такому наборі даних зображена на рисунку 3.8., агреговані метрики наведені у таблиці 3.1. Відкидання зайвих ознак дало можливість зменшити перенавчання моделі.

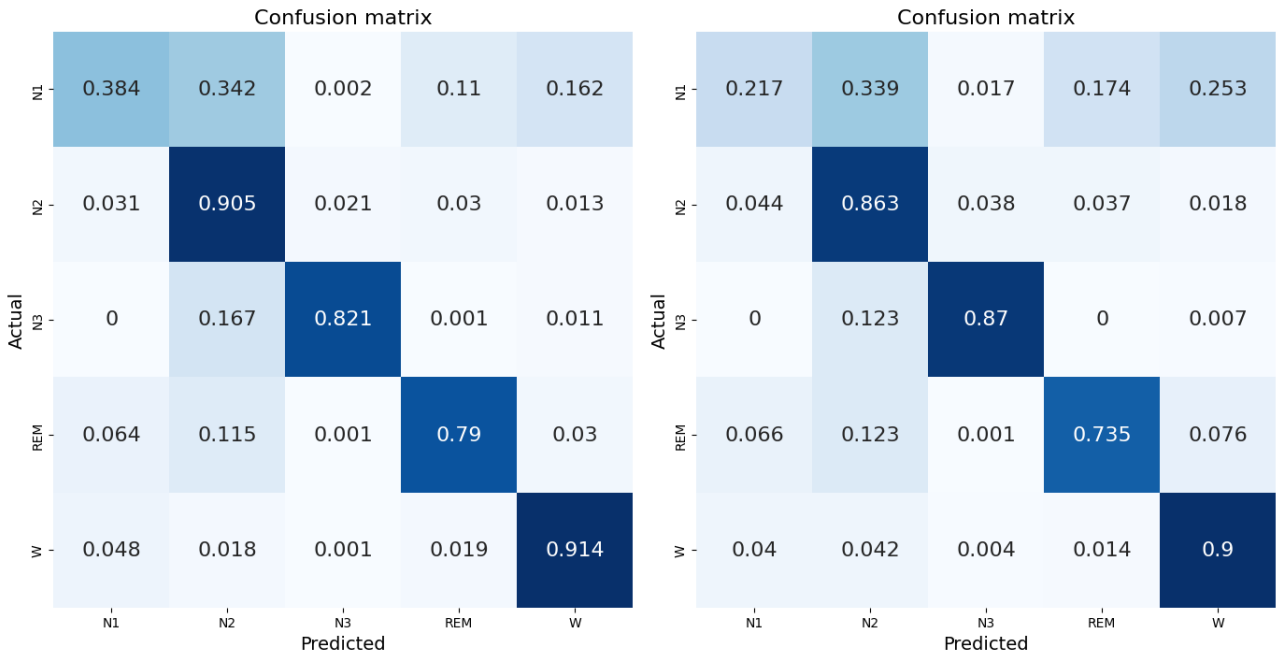

Рисунок 3.8 – Матриця неточностей для моделі навченої на відібраних ознаках

Рівень розпізнавання класу N1 залишається занадто низьким. Тому далі було застосовано техніки семплінгу: метод Edited Nearest-Neighbor у якості undersampling для переважаючих класів N2 та W, і метод SVMSMOTE у якості oversampling для класу N1.

Не дивлячись на те, що значення агрегованих метрик майже не змінилося (таблиця 3.1), на матриці неточностей (рисунок 3.9) бачимо, що точність для класу N1 зросла з 21.7% до 55%, проте точність класу R впала з 73.5% до 57%.

Отже, у порівнянні з базовою моделлю вдалося зменшити рівень перенавчання моделі, а також підвищити точність для класу N1. Варто розглядати і базову, і фінальну модель, адже перша пропонує достатньо гарну

точність на всіх класах, окрім N1, тоді як друга має рівень розпізнавання класу N1 ≥ 50%.

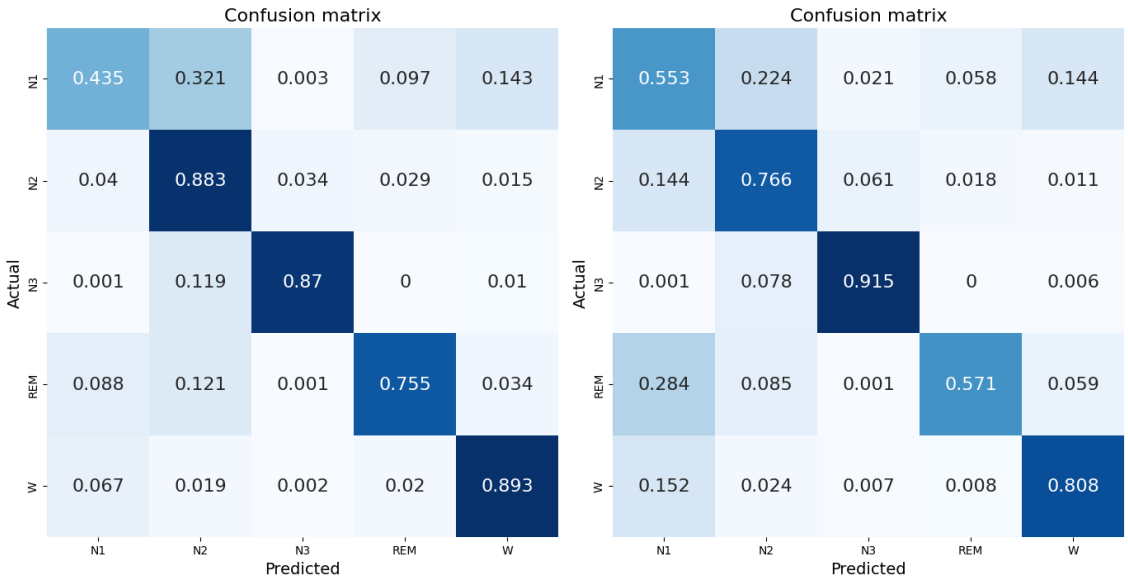

Рисунок 3.9 – Матриця неточностей фінальної моделі на тренувальному (зліва) та тестовому (справа) наборах даних

3.5.2 Модель CNN

Зафіксуємо деякі параметри. Нехай використовується модифікація градієнтного спуску – Adam, коефіцієнт навчання становить 0.0001. Розмір міні-батчу рівний 64., а кількість епох навчання – 300.

CNN є достатньо чутливою до незбалансованості даних і має тенденцію перевчатися на більш поширених класах. Найпростіший спосіб це компенсувати – встановити ваги для кожного класу для функції втрат. Емпіричним шляхом були підібрані наступні ваги: для класу N1 – 20, для N2 – 0.01, для N3 – 20, для R – 5, для W – 0.01.

Перша модель була навчена на даних одного каналу ЕЕГ без попередньої обробки. Аналогічно до базової моделі GB, згорткова модель має погану точність на класі N1 (рисунок 3.10), проте перенавчання відсутнє (таблиця 3.2).

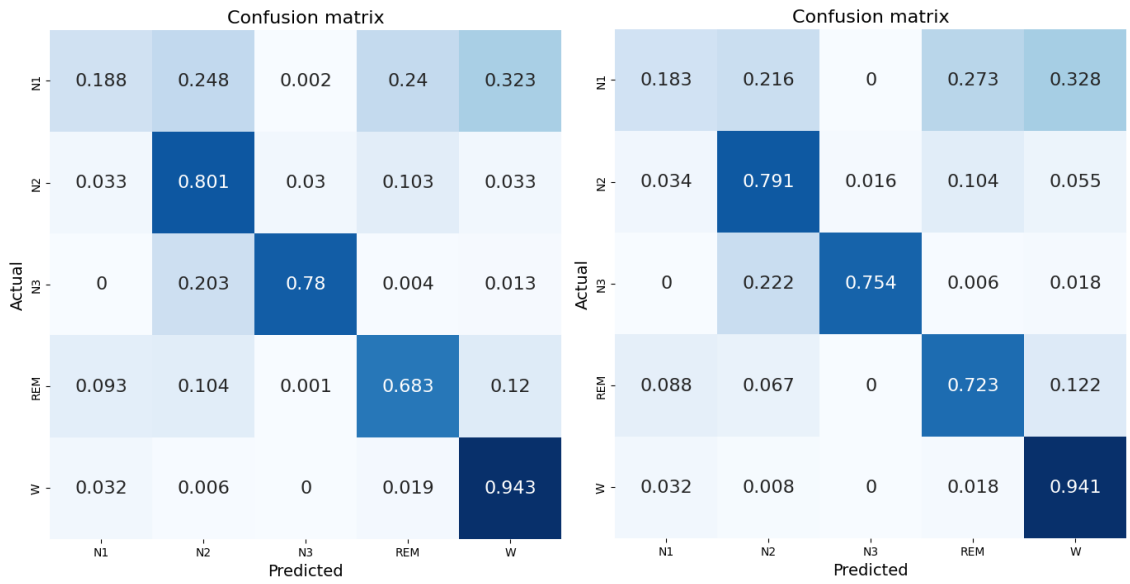

Рисунок 3.10 – Матриця неточностей моделі, навченої на 1 каналі ЕЕГ на тренувальному (зліва) та тестовому (справа) наборах даних.

Таблиця 3.2 – Результати навчання моделей CNN

Модель	**Тренувальний набір**			**Тестовий набір**		
	Влучність	**Повнота**	**F1**	**Влучність**	**Повнота**	**F1**
Один канал ЕЕГ	68.13%	67.89%	68.01%	69.71%	67.85%	68.76%
Застосування смугового фільтру	69.30%	68.34%	68.82%	72.36%	68.98%	70.63%
Один канал ЕЕГ та канал ЕОГ	69.93%	72.27%	71.08%	71.09%	71.69%	71.39%
Із застосуванням технік семплінгу	71.81%	74.15%	72.96%	72.32%	72.93%	72.63%

З метою поліпшення якості сигналу та зменшення впливу шуму до ЕЕГ сигналу було застосовано фільтр Баттерворта, який пригнічував усі частоти поза діапазону (0.3Гц – 35Гц). Модель навчена на таких даних демонструє трохи кращі результати, ніж попередня (таблиця 3.2, рисунок 3.11). Рівень розпізнавання класу N1 зріс на 7%.

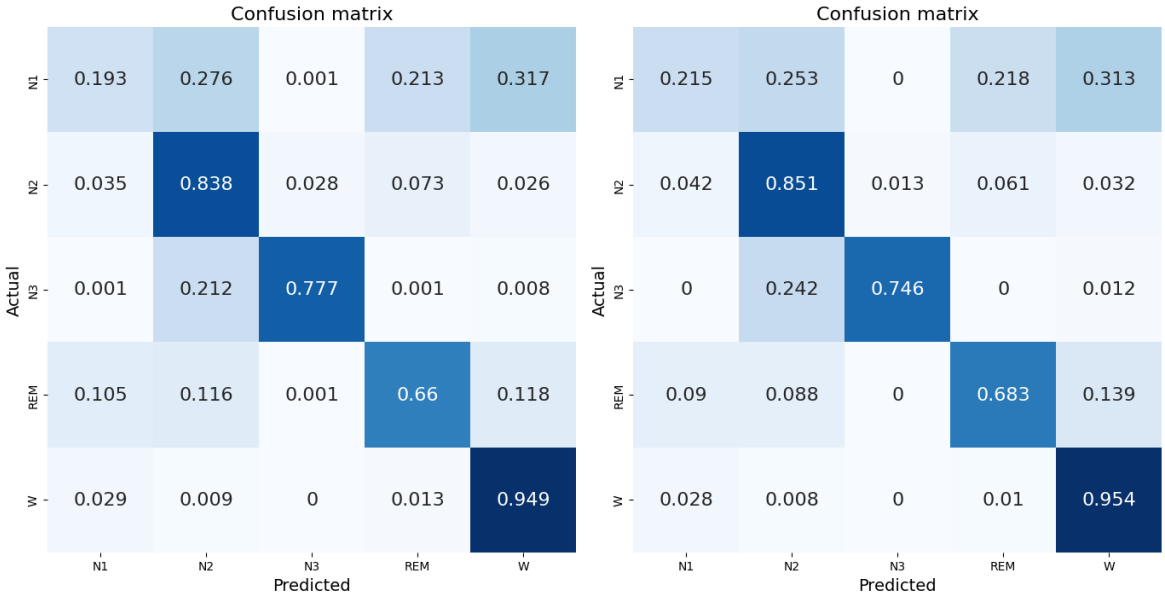

Рисунок 3.11 – Матриця неточностей моделі, навченої на оброблених даних ЕЕГ на тренувальному (зліва) та тестовому (справа) наборах даних.

Наступна модель була навчена на даних одного каналу ЕЕГ та каналу ЕОГ, що підвищило рівень розпізнавання класів N1, N3, R.

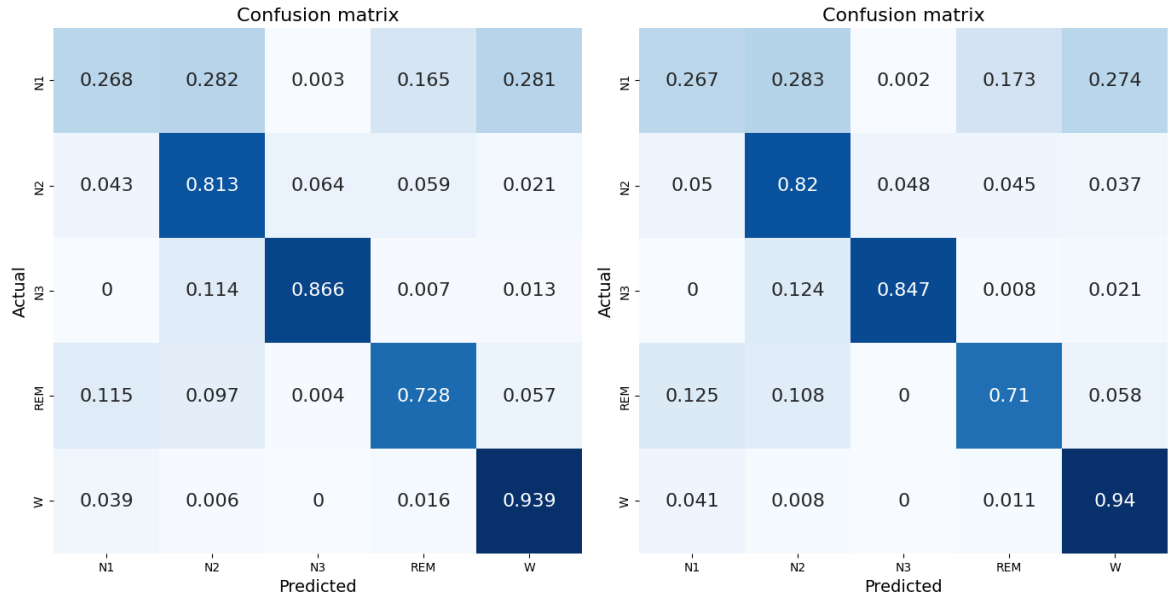

Рисунок 3.12 – Матриця неточностей моделі, навченої на даних ЕЕГ та ЕОГ на тренувальному (зліва) та тестовому (справа) наборах даних.

Для того, щоб покращити точність на класі N1 було застосовано техніки семплінгу. Оскільки алгоритми SMOTE до необроблених даних застосовувати не можна, було використано звичайний випадковий oversampling до класу N1, збільшивши кількість прикладів цього класу на 7500, і використано випадковий undersampling, щоб зменшити кількість прикладів класу N2 до 40000 прикладів. В результаті вдалося збільшити точність розпізнавання класу N1 до 39% на перевірочному наборі, але це все ще нижче 50%.

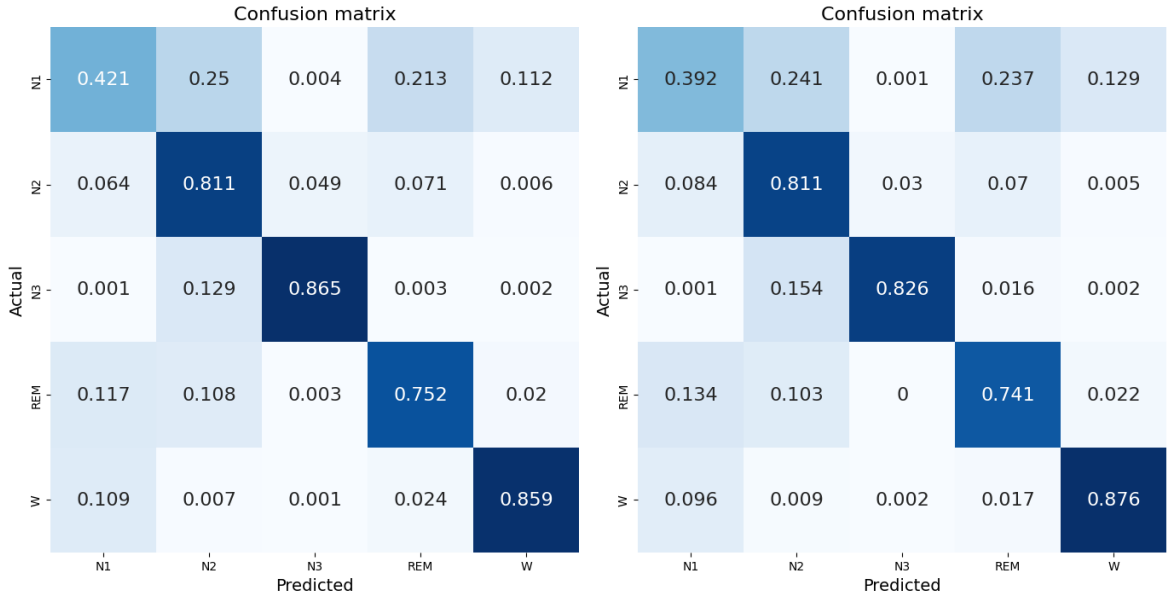

Рисунок 3.13 – Матриця неточностей моделі, навченої на даних ЕЕГ та ЕОГ на тренувальному (зліва) та тестовому (справа) наборах даних.

3.6 Висновки до розділу 3

В третьому розділі було обрано та оглянуто набір даних, який застосовувався для навчання моделей. Він, як і будь-який інший набір даних, що стосується стадій сну, є незбалансованим. Найменш представленими класами є N1 та N3, найбільш представленим виявився клас N3.

Далі, на основі проаналізованої літератури, у якості класифікаторів було обрано алгоритм градієнтного бустингу, який далі навчався на вилучених з ПСГ

ознаках, та архітектуру згорткової нейронної мережі, для навчання якої використовувалися необроблені дані.

Для моделі градієнтного бустингу було підібрано гіперпараметри, проаналізовано важливість ознак. У випадку CNN було розглянуто ефективність моделі без та з використанням смугового фільтру, на одному каналі ЕЕГ та з додатковим використанням каналу ЕОГ. Для обох підходів було в кінці застосовано техніки семплінгу для покращення точності на класі N1.

Значення агрегованих метрик на тестовому наборі даних для найкращої моделі GB наступні: влучність – 72.24%, повнота – 72.27%, F1 – 72.25%, що на 20% краще, ніж модель GB запропонована у [19]. Для найкращої моделі CNN: влучність – 72.32%, повнота – 72.93%, F1 – 72.63%. Отже, обидва підходи вирішення задачі автоматизації оцінки фаз сну демонструють майже однакові результати.

Моделі градієнтного бустингу варто надавати перевагу у випадку, якщо необхідно розуміти яким чином була визначена стадія сну, оскільки дерева рішень можна легко інтерпретувати.

Перевагою моделі згорткової мережі є те, що збільшення точності на класі N1 не призводить до значного погіршення рівня розпізнавання інших класів. Також, в подальшому можливе масштабування моделі за допомогою додавання рекурентних шарів, наприклад LSTM, що може покращити ефективність моделі.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні стадій сну. Дана реалізація буде розпізнавати стадію сну за даними сигналів ЕЕГ та ЕОГ, що дозволятиме аналізувати тривалість сну та його якість. А також можливі проблеми зі сном у людини, що актуальним питанням в наші дні.

Також в даному дослідженні будуть розглянуті різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. В сучасному світі існує багато різних можливостей для розробки програмних продуктів. Всі варіанти реалізації мають свої переваги та недоліки, тому дуже важливо обрати з-поміж усіх найбільш оптимальних варіант реалізації. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що 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 годин, що потрібні для реалізації програмного продукту, 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 прогнозу стійкості фінансових показників. 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 компанії.

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 можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

- F_1 – вибір мови програмування;
- F_2 – вибір архітектури обробки даних;
- F_3 – вибір типу набору даних.

Кожна з цих функцій має декілька варіантів реалізації.

Функція F_1:

а) Python;

б) Java.

Функція F_2:

а) застосування класичних цифрових фільтрів;

б) використання ICA (independent component analysis).

Функція F_3:

а) використання набору даних зі здоровими людьми;

б) використання набору даних, де наявні хворі люди.

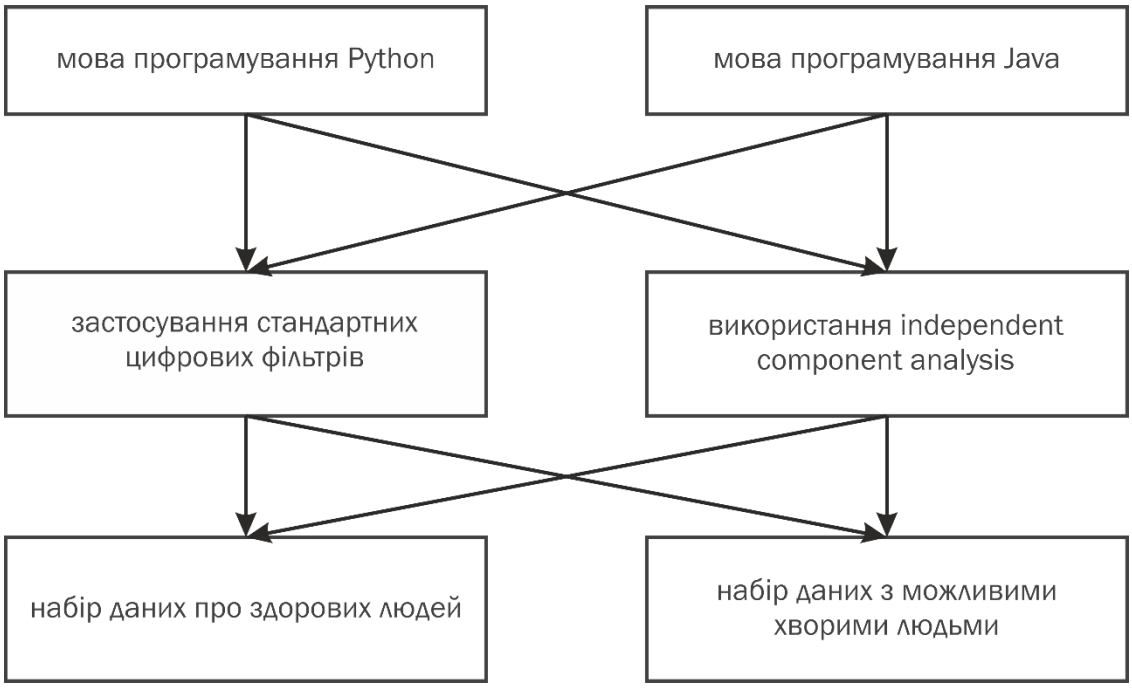

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Легка у використанні мова програмування, наявність задокументованих	Відсутня оптимізація використання оперативної пам’яті ЕОМ

		бібліотек машинного навчання	
	Б	Кросплатформна мова програмування, чітко структурована та майже не має вбудованих помилок	Потребує більш висококваліфікованих програмістів
F_2	А	Алгоритми давно відомі та можуть бути легко застосовані до будь-яких даних	Занадто прості і можуть не спрацювати на складних даних
	Б	Демонструє високу ефективність порівняно зі стандартними підходами	Потребує даних ЕЕГ з декількох каналів
F_3	А	Такий набір даних не містить аномальних викидів	Обмежує сферу застосування моделі
	Б	Навчена в результаті модель буде більш універсальною	Більш складно реалізувати через медичну специфіку задачі

На основі аналізу позитивно-негативної матриці 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 надаємо варіанту А, так як це потребуватиме менше ресурсів та буде більш просто реалізувати.

Функція F_2 допускає реалізацію обох варіантів, тому можливо використати варіант А чи Б.

Функція F_3 перевага надається варіанту реалізації А, так як ця задача буде простіша і при цьому втрачається досить незначна частина універсальності програмного продукту.

Таким чином, розглядатимемо такі варіанти реалізації ПП: F_1а $-$ F_2а $-$ F_3 та F_1а $-$ F_2б $-$ F_3.

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

4.3 Обґрунтування системи параметрів програмного продукту

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 програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	Умовне позначення	Од. виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	500	800	1500

Об’єм пам’яті	X2	Гб	20	15	10
Час обробки даних	X3	с	140	60	20
Об’єм програмного коду	X4	к-сть рядків	2300	1500	1000

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

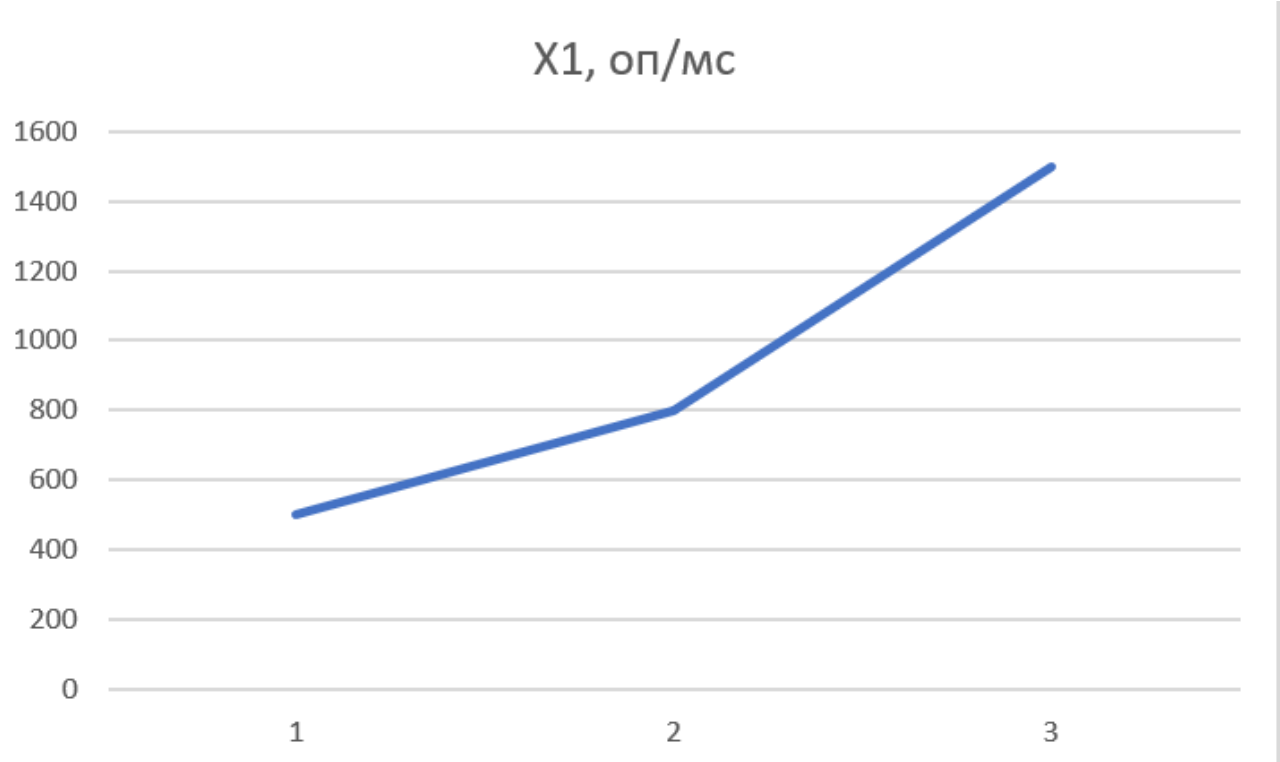

Рисунок 4.2 – X1, швидкодія мови програмування

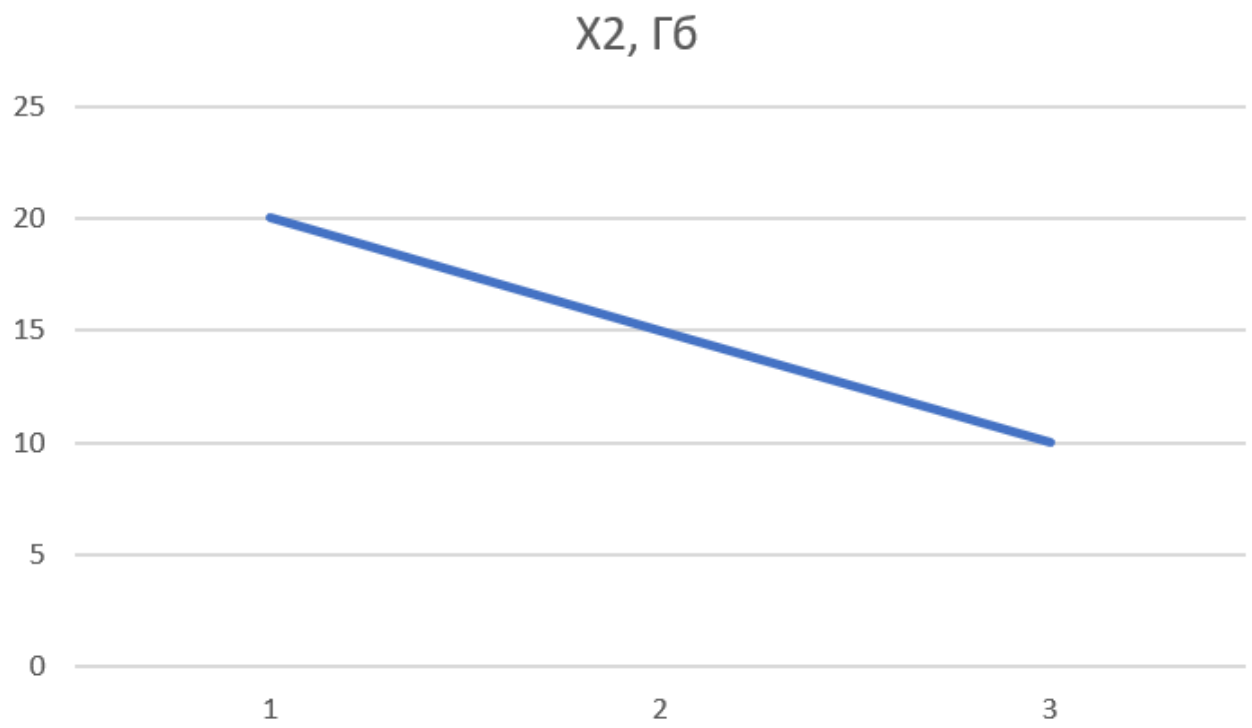

Рисунок 4.3 – X2, об'єм пам'яті

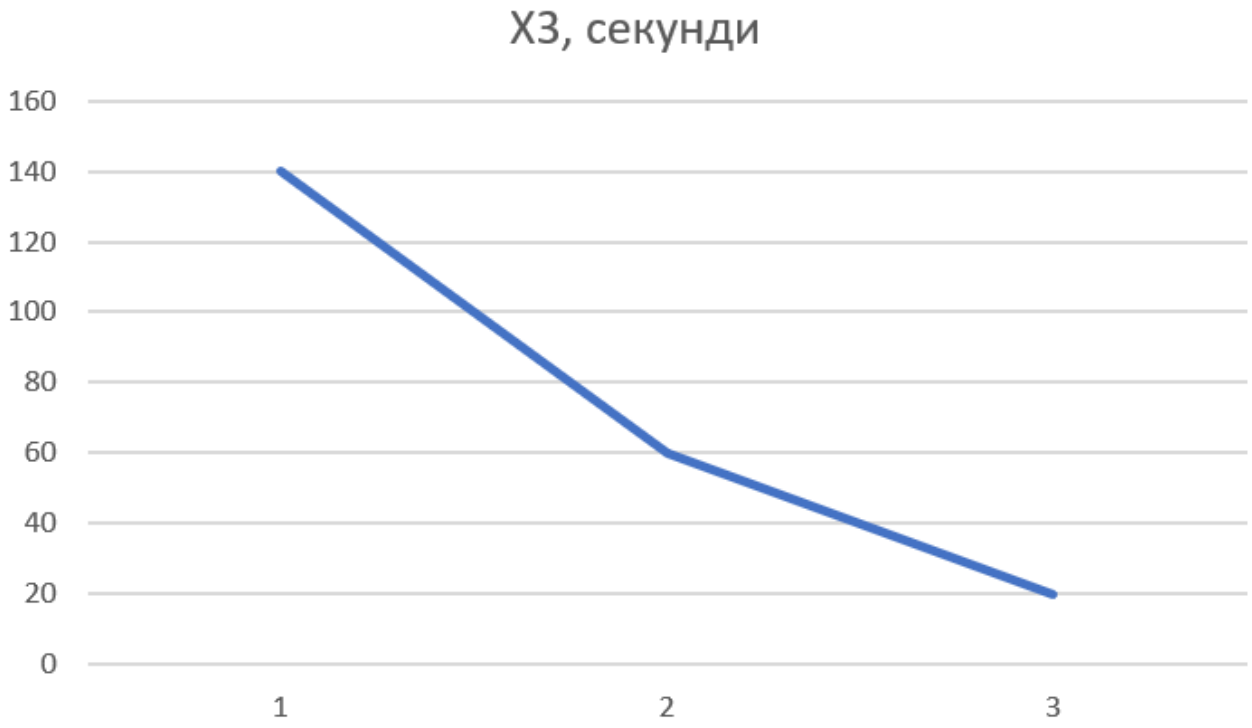

Рисунок 4.4 – X3, час обробки даних

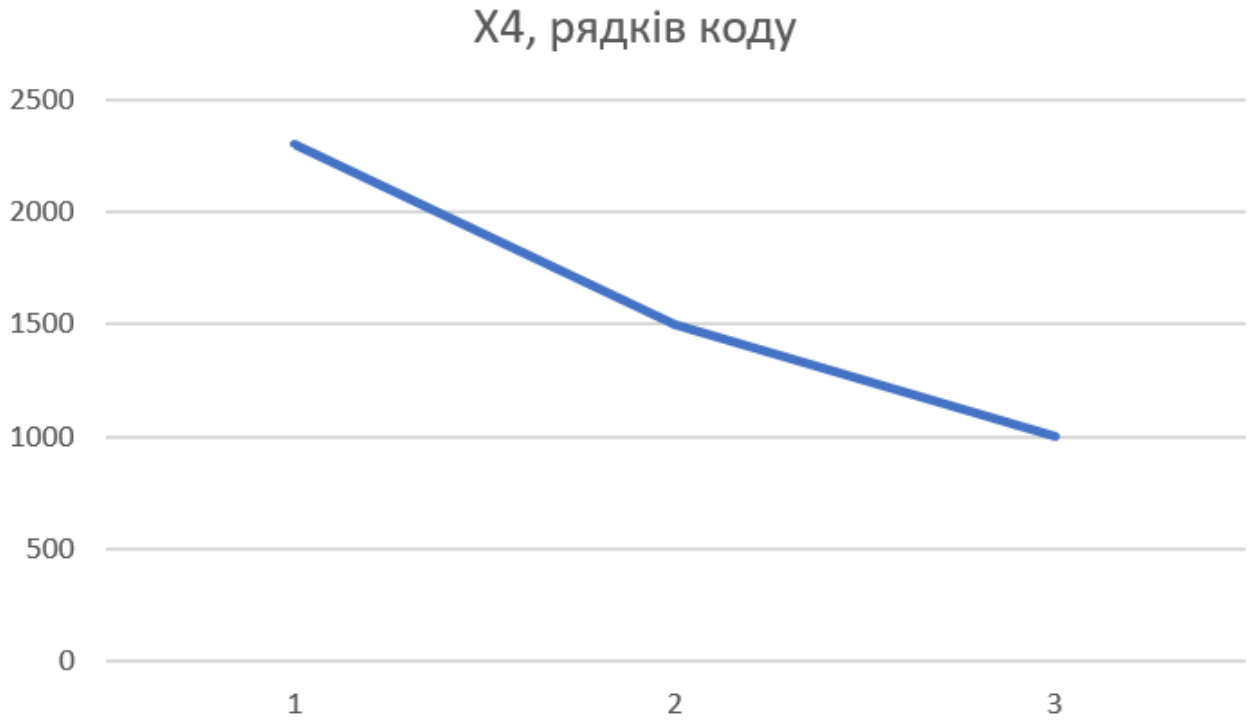

Рисунок 4.5 – X4, потенційний об’єм програмного коду

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 класифікації стадій сну.

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

Параметр	Назва параметра	Од. 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	2	3	2	1	1	2	2	13	-4,5	20,25
X2	Об’єм пам’яті	Гб	1	1	1	2	2	1	1	9	-8,5	72,25
X3	Час обробки даних	с	4	2	4	4	3	3	3	23	5,5	30,25
X4	Об’єм програмного коду	к-сть рядків	3	4	3	3	4	4	4	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	179

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

$$R_i = \sum_{j=1}^{N} r_{uj} R_{ij} = \frac{Nn(n+1)}{2} = 70, \tag{4.1}$$

де N – число експертів;

n – кількість параметрів.

б) середня сума рангів:

$$T = \frac{1}{n} \sum R_{ij} = 17{,}5. \tag{4.2}$$

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

$$\Delta_i = R_i - T. \tag{4.3}$$

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 179 \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 179}{7^2(4^3 - 4)} = 0{,}731 > W_k = 0{,}67. \tag{4.5}$$

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

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

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

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

Для кожного параметра зробимо розрахунок вагомості $K_{\text{ві}}$ за наступними формулами:

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

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

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:

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

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

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

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	1,5	0,5	0,5	3,5	0,22	12,25	0,21	41,88	0,2
X2	0,5	1	0,5	0,5	2,5	0,16	9,25	0,16	34,13	0,16
X3	1,5	1,5	1	0,5	4,5	0,28	16,25	0,28	59,13	0,28
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,88	0,36
Всього:					16	1	59	1	213	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):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \tag{4.11}$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості *i*-го параметра;

B_i – оцінка *i*-го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізацій основних функцій ПП

Основні функції	Варіант реалізації	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	800	6	0,2	1,2
F2	А	X2	10	10	0,16	1,6
	Б	X3	20	3	0,28	0,84
F3	А	X4	1000	9	0,36	3,24

За даними з таблиці 4.6 за формулою:

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

Визначаємо рівень якості кожного з варіантів:

- $K_{K1} = 1{,}2 + 1{,}6 + 3{,}24 = 6{,}04$;
- $K_{K2} = 1{,}2 + 0{,}84 + 3{,}24 = 5{,}28$.

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:

$$\text{Т}_\text{О} = \text{Т}_\text{Р} \cdot \text{К}_\text{П} \cdot \text{К}_\text{СК} \cdot \text{К}_\text{М} \cdot \text{К}_\text{СТ} \cdot \text{К}_\text{СТ.М}, \quad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт за складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

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

$К_{СТ.М}$ – 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: $Т_Р = 40$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П = 1.5$. 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 $К_{СТ} = 0.75$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 40 \cdot 1.5 \cdot 0.75 = 45 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань. Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р = 23$ людино-днів, $К_П = 0.84$, $К_{СК} = 1$ та $К_{СТ} = 0.7$:

$$Т_2 = 23 \cdot 0.84 \cdot 0.7 = 13{,}52 \text{ людино-днів.}$$

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

$$Т_I = (45 + 13.52 + 6.6 + 18.4) \cdot 8 = 668{,}16 \text{ людино-годин,}$$
$$Т_{II} = (45 + 13.52 + 8.3 + 18.4) \cdot 8 = 681{,}76 \text{ людино-годин.}$$

Найбільш високу трудомісткість має другий варіант.

В розробці беруть участь один полісомнограф з окладом 20000 грн., один програміст з окладом 17000. Визначимо середню зарплату за годину за формулою:

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

де M – місячний оклад працівників;

T_m – кількість робочих днів за місяць всіх працівників;

t – кількість робочих годин в день.

$$С_ч = \frac{20000 + 17000}{3 \cdot 21 \cdot 8} = 73{,}41 \text{ грн.} \quad (4.15)$$

Тоді розрахуємо заробітну плату за формулою:

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

де $С_ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

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

Заробітна плата робітників за варіантами становить:

$$\text{I. } С_{ЗП} = 73{,}41 \cdot 668{,}16 \cdot 1{,}2 = 58859{,}55 \text{ грн.}$$

$$\text{II. } С_{ЗП} = 73{,}41 \cdot 681{,}76 \cdot 1{,}2 = 60057{,}60 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } С_{ВІД} = С_{ЗП} \cdot 0{,}22 = 58859{,}55 \cdot 0{,}22 = 12949{,}10 \text{ грн.}$$

$$\text{II. } С_{ВІД} = С_{ЗП} \cdot 0{,}22 = 60057{,}60 \cdot 0{,}22 = 13212{,}67 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. Так як одна ЕОМ обслуговує одного програміста з окладом 17000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot M \cdot K_З = 12 \cdot 17000 \cdot 0.2 = 40800 \text{ грн.}$$

З урахування додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + K_З) = 40800 \cdot (1 + 0.2) = 48960 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 48960 \cdot 0.22 = 10771{,}2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 50000 грн:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.3 \cdot 0.25 \cdot 50000 = 16250 \text{ грн.},$$

де $К_{ТМ}$ - коефіцієнт витрат на транспортування та монтаж приладу;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot К_Р \cdot Ц_{ПР} = 1.3 \cdot 0.13 \cdot 50000 = 8450 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\text{Т}_{\text{ЕФ}} = (\text{Д}_{\text{К}} - \text{Д}_{\text{В}} - \text{Д}_{\text{С}} - \text{Д}_{\text{Р}}) \cdot t \cdot \text{К}_{\text{В}} = \\ = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.8 = 1491{,}2 \text{ годин}, \quad (0.1)$$

де $\text{Д}_{\text{К}}$ – календарна кількість днів у році;

$\text{Д}_{\text{В}}$, $\text{Д}_{\text{С}}$ – кількість вихідних та святкових днів відповідно;

$\text{Д}_{\text{Р}}$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$\text{К}_{\text{В}}$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$\text{С}_{\text{ЕЛ}} = \text{Т}_{\text{ЕФ}} \cdot N_C \cdot \text{К}_{\text{З}} \cdot \text{Ц}_{\text{ЕН}} = 1491.2 \cdot 1.2 \cdot 0.6 \cdot 4.79 = 5142.85 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу;

$\text{К}_{\text{З}}$ – коефіцієнт зайнятості приладу;

$\text{Ц}_{\text{ЕН}}$ – тариф за 1 кВт-годину електроенергії.

Накладні витрати розраховуємо за формулою:

$$\text{С}_{\text{Н}} = \text{Ц}_{\text{ПР}} \cdot 0.67 = 50000 \cdot 0.67 = 33500 \text{ грн.}$$

Тоді річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}} = \\ = 48960 + 10771.2 + 16250 + 8450 + 5142.85 + 33500 = \\ = 123074.05 \text{ грн.} \quad (0.2)$$

Собівартість одної машино-години ЕОМ дорівнюватиме:

$$C_{\text{М}-\text{Г}} = C_{\text{ЕКС}}/\text{Т}_{\text{ЕФ}} = 123074.05/1491.2 = 82.53$$

$$С_М = С_{М-Г} \cdot Т, \quad (0.3)$$

I. $С_М = 82.53 \cdot 668{,}16 = 55143{,}24$ грн.,

II. $С_М = 82.53 \cdot 681{,}76 = 56265{,}65$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0.67, \quad (4.20)$$

I. $С_Н = 58859.55 \cdot 0.67 = 39435.90$ грн.,

II. $С_Н = 60057.60 \cdot 0.67 = 40238.59$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н, \quad (4.21)$$

I. $С_{ПП} = 58859.55 + 12949.10 + 55143.24 + 39435.90 = 166387.79,$

II. $С_{ПП} = 60057.60 + 13212.67 + 56265.65 + 40238.59 = 169774.51.$

4.7 Вибір кращого варіант ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{К j} / С_{Ф j}, \quad (4.21)$$

$$К_{ТЕР1} = 6.04/166387.79 = 3.630 \cdot 10^{-5},$$

$$К_{ТЕР2} = 5.28/169774.51 = 3.110 \cdot 10^{-5}$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 3.630 \cdot 10^{-5}$.

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 $К_{ТЕР1} = 3.630 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

1. вибір мови програмування – Python;
2. вибір архітектури обробки даних – стандартні алгоритми фільтрації;
3. вибір типу набору даних – дані зі здоровими людьми.

Даний варіант програмного комплексу дає користувачу швидкий та надійний інструмент для відслідковування якості сну та здебільшого орієнтований на людей, що не мають вад здоров’я.

4.8 Висновки до четвертого розділу

В даному розділі був проведений функціонально-вартісний аналіз програмного продукту. Він складається з двох етапів – розрахунку коефіцієнту технічного рівня та вартості розробки програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного продукту, що розробляється для аналізу сигналів ЕЕГ та ЕОГ для класифікації стадій сну, було визначено та проведено оцінку основний функцій програмного продукту, а також знайдено параметри, що його характеризують.

Остаточно були розраховані коефіцієнти техніко-економічного рівня для кожного з можливих варіантів реалізації програмного продукту та був обраний найкращий варіант з огляду на це.

ВИСНОВОК

У даній роботі було розглянуто задачу автоматичної класифікації стадій сну.

У першому розділі було детально описано предметну сферу: визначено стадії сну та їх особливості, наведено цикл сну здорової людини, визначено поняття шумів та артефактів записів ЕЕГ, розглянуто проблему незбалансованості тривалості кожної стадії сну. Також було проаналізовано існуючі способи вирішення задачі, і здійснено постановку задачі.

У другому розділі була приділена увага математичним основам роботи. Було розглянуто ознаки, які можна вилучати з даних ПСГ, структуру штучного нейрону, наведено функції активації, їх переваги та недоліки, описано типову структуру багатошарового персептрону та згорткової нейронної мережі, типовий алгоритм навчання штучних нейронних мереж. Також описано дерева рішень та градієнтний бустинг.

В третьому розділі було обрано набір даних для навчання моделей та метрики, які будуть використовуватися для перевірки якості моделей. Визначено які алгоритми машинного навчання та архітектури згорткових мереж будуть використовуватися для вирішення задачі. Проаналізовано ефективність побудованих моделей та порівняно їх результати.

У четвертому розділі описано результати функціонально-вартісного аналізу програмного продукту , що розробляється для аналізу сигналів ЕЕГ та ЕОГ для класифікації стадій сну. В результаті було визначено та проведено оцінку основний функцій програмного продукту, а також знайдено параметри, що його характеризують.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Altaf, M. A. B. A 0.21 μJ patient-specific REM/Non-REM sleep classifier for Alzheimer patients / Altaf, M. A. B., Saadeh, W.. // IEEE Biomedical Circuits and Systems Conference. – 2017.

2. Wilson, S. Treatment of insomnia / Wilson, S., Nutt, D.. // Psychiatry. – 2007. – №6. – С. 301–304.

3. Rosenberg, R.S. The American Academy of Sleep Medicine inter-scorer reliability program: Respiratory events. / Rosenberg, R.S., Hout, S.V.. // J. Clin. Sleep Med.. – 2014. – №10. – С. 447–454.

4. Qureshi, Shahnawaz. Human Sleep Scoring Based on K-Nearest Neighbors. / Qureshi, Shahnawaz, Karrila, Seppo, Vanichayobon, Sirirut. // TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES. – 2018. – №26. – С. 2803–2819.

5. Чернінський А. О. Електрофізіологія головного мозку: методичні рекомендації до практикуму / А. О. Чернінський, С. А. Крижановський, І. Г. Зима. – Київ, 2011.

6. The AASM Manual for the Scoring of Sleep and Associated Events: Rules, Terminology and Technical Specifications / Iber, Conrad, Ancoli-Israel, Sonia, Chesson, A.L., Quan, Stuart. // Westchester, IL: American Academy of Sleep Medicine. – 2007.

7. Rechtschaffen A. A manual for standardized terminology, techniques and scoring system for sleep stages in human subjects / Rechtschaffen A.. // Clin Neurophysiol. – 1968.

8. Silber, M. H. The Visual Scoring of Sleep in Adults / Silber, M. H., Ancoli-Israel, S., Bonnet, M. H.. // Journal of Clinical Sleep Medicine. – 2007. – С. 121–131.

9. Billiard M. Sleep: physiology, investigation, and medicine / Michel Billiard., 2003.

10. Lan, Kun-Chan. Using Off-the-Shelf Lossy Compression for Wireless Home Sleep Staging / Lan, Kun-Chan, Chang, Da-Wei, Kuo, Chih-En. // Journal of neuroscience methods. – 2015.

11. Bradley's Neurology in Clinical Practice / Robert B. Daroff, Gerald M. Fenichel, Joseph Jankovic, John C. Mazziotta.

12. Bahammam A. POLYSOMNOGRAPHY I: PROCEDURE AND TECHNOLOGY / A. Bahammam, D. Gacuan, S. George. – 2016.

13. Li Hu. EEG Signal Processing and Feature Extraction / Li Hu, Zhiguo Zhang., 2019.

14. Jung, T.-P. Removing electroencephalographic artifacts by blind source separation / Jung, T.-P., Makeig, S., Humphries, C.. // Psychophysiology. – 2000. – С. 163–178.

15. Gratton, G. A new method for off-line removal of ocular artifact / Gratton, G., Coles, M., Donchin, E.. // Electroencephalography and Clinical Neurophysiology. – 1983. – С. 468–484.

16. Devuyst, S. Automatic Processing of EEG-EOG-EMG Artifacts in Sleep Stage Classification / Devuyst, S., Dutoit, T., Ravet, T.. // 13th International Conference on Biomedical Engineering. – 2009. – С. 146–150.

17. Haibo He. Learning from Imbalanced Data / Haibo He, Garcia, E. A.. // IEEE Transactions on Knowledge and Data Engineering. – 2009. – С. 1263–1284.

18. Chawla N. V. SMOTE: synthetic minority over-sampling technique / N. V. Chawla, L. O. Hall, K. W. Bowyer. // 2002. – С. 321–357.

19. Stephansen J. B. The use of neural networks in the analysis of sleep stages and the diagnosis of narcolepsy / J. B. Stephansen, A. Ambati, E. B. Leary. // CoRR. – 2017.

20. Lajnef T. Learning Machines and Sleeping Brains: Automatic Sleep Stage Classification using Decision-Tree Multi-Class Support Vector Machines / T. Lajnef, S. Chaibi, P. Ruby. // Journal of Neuroscience Methods. – 2015. – С. 94–105.

21. Chambon S. A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series / S. Chambon, M. Galtier, A. Pierrick. // IEEE Transactions on Neural Systems and Rehabilitation Engineering. – 2018. – С. 758–769.

22. Mousavi S. SleepEEGNet: Automated sleep stage scoring with sequence to sequence deep learning approach / S. Mousavi, F. Afghah, U. Acharya. // PLOS ONE. – 2019.

23. Dubey S. Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark / S. Dubey, S. Singh, B. Chaudhuri. // arXiv. – 2021.

24. Alboaneen D. Glowworm Swarm Optimisation for Training Multi-Layer Perceptrons / D. Alboaneen, H. Tianfield, Y. Zhang. – 2017. – С. 131–138.

25. Albawi S. Understanding of a Convolutional Neural Network / S. Albawi, T. Abed Mohammed. – 2017.

26. Rumelhart D. Learning representations by back-propagating errors / D. Rumelhart, G. Hinton, R. Williams. // Nature. – 1986. – №323. – С. 533–536.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
# -*- coding: utf-8 -*-
"""
# Imports
"""

# Commented out IPython magic to ensure Python compatibility.
import os
os.environ['PYTHONHASHSEED']=str(225)
import gc
import numpy as np
import random
import tensorflow as tf2
from sklearn.model_selection import train_test_split
import glob
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
# %matplotlib inline

def reset_random_seeds():
   os.environ['PYTHONHASHSEED']=str(225)
   tf2.random.set_seed(225)
   np.random.seed(225)
   random.seed(225)

"""# Scoring"""

import seaborn as sns
from sklearn.metrics import confusion_matrix, f1_score, precision_score,
recall_score, balanced_accuracy_score, matthews_corrcoef

def scoring(y_true, y_pred, labels=None, average=[None], print_res=False):

    conf_mat = confusion_matrix(y_true, y_pred, normalize='true').round(decimals=3)
    con_mat = pd.DataFrame(conf_mat, index=labels, columns=labels)

    score = dict()
    for avg in average:
        score[avg] = {"Precision" : precision_score(y_true, y_pred, average=avg),
                      "Recall" : recall_score(y_true, y_pred, average=avg),
                      "F1-score" : f1_score(y_true, y_pred, average=avg)}

    if print_res:
        score_df = pd.DataFrame(score)
        plt.figure(figsize=(8,8))
```

```
        plt.title("Confusion matrix", fontsize=16)
        sns.heatmap(con_mat, annot=True, annot_kws={"size": 16}, fmt='g',
cmap='Blues', cbar=False)
        plt.ylabel("Actual", fontsize=14)
        plt.xlabel("Predicted", fontsize=14)

        display(score_df)

    return conf_mat, score

"""# Load Data"""

def get_data(file_names):
  all_data = []
  all_labels = []
  for file in file_names:
    with np.load(file, allow_pickle=True) as f:
      data = f['x'].item()['EEG Fpz-Cz']
      labels = f['y']

    data = data.astype(np.float32)
    labels = labels.astype(np.int32)

    all_data.append(data)
    all_labels.append(labels)

  all_data = np.vstack(all_data)
  all_labels = np.hstack(all_labels)
  return all_data, all_labels

def get_data_plus_EOG(file_names):
  all_data = []
  all_labels = []
  for file in file_names:
    with np.load(file, allow_pickle=True) as f:
      data = np.array([butter_bandpass_filter(f['x'].item()['EEG Fpz-Cz'], 0.3, 35,
100).T, f['x'].item()['EOG horizontal'].T]).T
      labels = f['y']

    data = data.astype(np.float32)
    labels = labels.astype(np.int32)
    all_data.append(data)
    all_labels.append(labels)

  all_data = np.vstack(all_data)
  all_labels = np.hstack(all_labels)
  return all_data, all_labels

"""# Build Models"""

def first_model():
```

```
  model = tf2.keras.Sequential()
  model.add(tf2.keras.layers.Conv1D(filters=1, kernel_size=3, strides=1,
activation='linear', kernel_initializer='glorot_normal',input_shape=(3000, 2)))
  model.add(tf2.keras.layers.Conv1D(filters=10, kernel_size=64, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=16, strides=16, padding='valid'))
  model.add(tf2.keras.layers.Conv1D(filters=10, kernel_size=64, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=16, strides=16, padding='valid'))
  model.add(tf2.keras.layers.Flatten())
  model.add(tf2.keras.layers.Dropout(0.5))
  #model.add(tf2.keras.layers.Dense(100, activation='relu',
kernel_initializer="he_normal"))
  model.add(tf2.keras.layers.Dense(5, activation='softmax',
kernel_initializer='glorot_normal'))
  return model

def second_model():
  model = tf2.keras.Sequential()
  model.add(tf2.keras.layers.Conv1D(filters=64, kernel_size=5, strides=3,
activation='relu', kernel_initializer='he_normal',input_shape=(3000, 1)))
  model.add(tf2.keras.layers.Conv1D(filters=128, kernel_size=5, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=2, strides=2, padding='valid'))
  model.add(tf2.keras.layers.Dropout(0.2))

  model.add(tf2.keras.layers.Conv1D(filters=128, kernel_size=13, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.Conv1D(filters=256, kernel_size=7, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=2, strides=2, padding='valid'))

  model.add(tf2.keras.layers.Conv1D(filters=128, kernel_size=13, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.Conv1D(filters=256, kernel_size=7, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=2, strides=2, padding='valid'))

  model.add(tf2.keras.layers.Conv1D(filters=32, kernel_size=3, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.Conv1D(filters=64, kernel_size=6, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=2, strides=2, padding='valid'))

  model.add(tf2.keras.layers.Conv1D(filters=8, kernel_size=5, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.Conv1D(filters=8, kernel_size=2, strides=1,
padding='same', activation='relu', kernel_initializer='he_normal'))
  model.add(tf2.keras.layers.MaxPool1D(pool_size=2, strides=2, padding='valid'))

  model.add(tf2.keras.layers.Flatten())
```

```
  #model.add(tf2.keras.layers.Dropout(0.4))
  model.add(tf2.keras.layers.Dense(64, activation='relu',
kernel_initializer="he_normal"))
  model.add(tf2.keras.layers.Dropout(0.2))
  model.add(tf2.keras.layers.Dense(5, activation='softmax',
kernel_initializer='glorot_normal'))
  return model

"""# Filters"""

from scipy.signal import butter, lfilter, iirnotch, filtfilt

def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a


def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

def notch_filter(data, fs, f0, Q):
  b_notch, a_notch = iirnotch(f0, Q, fs)
  return filtfilt(b_notch, a_notch, data)

"""# Data loading and preprocessing"""

npz_files_pattern = '/content/drive/MyDrive/diploma_data_edf/*.npz'
npz_files_names = glob.glob(npz_files_pattern)

train_files, test_files = train_test_split(npz_files_names, test_size=0.15,
random_state=100)

X_train, y_train = get_data_plus_EOG(train_files)
X_test, y_test = get_data_plus_EOG(test_files)

label_to_name ={1: 'N1', 2: 'N2', 3: 'N3', 4: 'REM', 5: 'W'}
train_stages, train_counts = np.unique(y_train, return_counts=True)
test_stages, test_counts = np.unique(y_test, return_counts=True)

print(f'Train set shape: {X_train.shape}')
print(f'Test set shape: {X_test.shape}')
print(f'Train stages distributions: {[f"{label_to_name[stage]} - {count}" for
stage, count in zip(train_stages, train_counts)]}')
print(f'Test stages distributions: {[f"{label_to_name[stage]} - {count}" for stage,
count in zip(test_stages, test_counts)]}')
```

```
onehotencoder = OneHotEncoder()

y_train = onehotencoder.fit_transform(y_train.reshape(-1,1)).toarray()
y_test = onehotencoder.fit_transform(y_test.reshape(-1,1)).toarray()

reset_random_seeds()

"""# First model

## Without any transformation
"""

base_model = first_model()

base_model.compile(optimizer=tf2.keras.optimizers.Adam(learning_rate=0.0001,
weight_decay=0.003),
              metrics=['accuracy'],
              loss='categorical_crossentropy',
              loss_weights=[20, 0.01, 20, 5, 0.01])

base_model.fit(
    X_train, y_train, batch_size=64,validation_split=0.15, epochs=300,
shuffle=False
)

gc.collect()

_, _ = scoring(np.argmax(y_train, axis=1), np.argmax(base_model.predict(X_train),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])

_, _ = scoring(np.argmax(y_test, axis=1), np.argmax(base_model.predict(X_test),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])

base_model.save('/content/drive/MyDrive/models/first_base_model')

"""## Min-Max Scaler"""

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

scaled_model = first_model()

scaled_model.compile(optimizer=tf2.keras.optimizers.Adam(learning_rate=0.0001,
weight_decay=0.003),
              metrics=['accuracy',
                       tf2.keras.metrics.Precision(name='precision'),
```

```
                        tf2.keras.metrics.Recall(name='recall'),
                        tf2.keras.metrics.AUC(name='auc')],
              loss='categorical_crossentropy',
              loss_weights=[20, 0.01, 20, 5, 0.01])

scaled_model.fit(
    X_train, y_train, batch_size=64,validation_split=0.15, epochs=300,
shuffle=False
)

gc.collect()

_, _ = scoring(np.argmax(y_train, axis=1), np.argmax(scaled_model.predict(X_train),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])

_, _ = scoring(np.argmax(y_test, axis=1), np.argmax(scaled_model.predict(X_test),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])

scaled_model.save('/content/drive/MyDrive/models/first_scaled_model')

"""## Notch filter"""

X_train = notch_filter(X_train, 100, 50, 30)
X_test = notch_filter(X_test, 100, 50, 30)

notch_model = first_model()

notch_model.compile(optimizer=tf2.keras.optimizers.Adam(learning_rate=0.0001,
weight_decay=0.003),
              metrics=['accuracy',
                        tf2.keras.metrics.Precision(name='precision'),
                        tf2.keras.metrics.Recall(name='recall'),
                        tf2.keras.metrics.AUC(name='auc')],
              loss='categorical_crossentropy',
              loss_weights=[20, 0.01, 20, 5, 0.01])

notch_model.fit(
    X_train, y_train, batch_size=64,validation_split=0.15, epochs=300,
shuffle=False
)

gc.collect()

_, _ = scoring(np.argmax(y_train, axis=1), np.argmax(notch_model.predict(X_train),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])
```

```
_, _ = scoring(np.argmax(y_test, axis=1), np.argmax(notch_model.predict(X_test),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])

notch_model.save('/content/drive/MyDrive/models/first_notch_model')

"""## Bandpass filter"""

X_train = butter_bandpass_filter(X_train, 0.3, 35, 100)
X_test = butter_bandpass_filter(X_test, 0.3, 35, 100)

bandpass_model = first_model()

bandpass_model.compile(optimizer=tf2.keras.optimizers.Adam(learning_rate=0.0001,
weight_decay=0.003),
              metrics=['accuracy',
                       tf2.keras.metrics.Precision(name='precision'),
                       tf2.keras.metrics.Recall(name='recall'),
                       tf2.keras.metrics.AUC(name='auc')],
              loss='categorical_crossentropy',
              loss_weights=[20, 0.01, 20, 5, 0.01])

bandpass_model.fit(
    X_train, y_train, batch_size=64,validation_split=0.15, epochs=300,
shuffle=False
)

gc.collect()

_, _ = scoring(np.argmax(y_train, axis=1),
np.argmax(bandpass_model.predict(X_train), axis=1),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(np.argmax(y_test, axis=1), np.argmax(bandpass_model.predict(X_test),
axis=1), print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3',
'REM', 'W'])

bandpass_model.save('/content/drive/MyDrive/models/first_banpass_epochs_model')

input_placeholder = second_bandpass_model.input

np.where(np.argmax(y_train, axis=1) == 2)

input_placeholder =
second_bandpass_model.input                                    # input
placeholder
outputs = [layer.output for layer in second_bandpass_model.layers]          # all
layer outputs
functors = [tf2.keras.backend.function([input_placeholder], [out]) for out in
outputs]    # evaluation functions
```

```
# Testing
test = X_train[74].reshape(1, -1, 1)
layer_outs = [func([test]) for func in functors]

plt.plot(range(0, layer_outs[0][0][0].shape[0]), layer_outs[0][0][0])

plt.plot(range(0, layer_outs[1][0][0][:, 1].shape[0]), layer_outs[1][0][0][:, 1:4])

plt.plot(range(0, layer_outs[2][0][0][:, 1:4].shape[0]), layer_outs[2][0][0][:,
1:4])

plt.plot(range(0, layer_outs[3][0][0].shape[0]), layer_outs[3][0][0])

plt.plot(range(0, layer_outs[4][0][0].shape[0]), layer_outs[4][0][0])

"""## EEG + EOG

### Same input different chanels
"""

two_channels_model_final = first_model()

two_channels_model_final.compile(optimizer=tf2.keras.optimizers.Adam(learning_rate=
0.0001, weight_decay=0.003),
              metrics=['accuracy',
                       tf2.keras.metrics.Precision(name='precision'),
                       tf2.keras.metrics.Recall(name='recall'),
                       tf2.keras.metrics.AUC(name='auc')],
              loss='categorical_crossentropy',
              loss_weights=[20, 0.01, 20, 5, 0.01])

history = two_channels_model_final.fit(
    X_train, y_train, batch_size=64,validation_split=0.15, epochs=300,
shuffle=False
)

gc.collect()

two_channels_model_final.save('/content/drive/MyDrive/models/first_two_channels_mod
el_final')

history.history

_, _ = scoring(np.argmax(y_train, axis=1),
np.argmax(two_channels_model_final.predict(X_train), axis=1),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(np.argmax(y_test, axis=1),
np.argmax(two_channels_model_final.predict(X_test), axis=1),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])
```

```
"""## Resampling"""

n1_indx = np.where(np.argmax(y_train, axis=1) == 0)[0]
n2_indx = np.where(np.argmax(y_train, axis=1) == 1)[0]
n3_indx = np.where(np.argmax(y_train, axis=1) == 2)[0]
r_indx = np.where(np.argmax(y_train, axis=1) == 3)[0]
w_indx = np.where(np.argmax(y_train, axis=1) == 4)[0]
selected_n2_indx = np.random.choice(n2_indx, size=33000, replace=False)
additional_n1_indx = np.random.choice(n1_indx, size=7500, replace=False)
additional_n3_indx = np.random.choice(n3_indx, size=4000, replace=False)
final_indx = np.concatenate([n1_indx, additional_n1_indx, selected_n2_indx,
n3_indx, additional_n3_indx, r_indx, w_indx])
np.random.shuffle(final_indx)
X_train_resampled, y_train_resampled = X_train[final_indx], y_train[final_indx]

two_channels_model_resampled = first_model()

two_channels_model_resampled.compile(optimizer=tf2.keras.optimizers.Adam(learning_r
ate=0.0001, weight_decay=0.003),
              metrics=['accuracy',
                       tf2.keras.metrics.Precision(name='precision'),
                       tf2.keras.metrics.Recall(name='recall'),
                       tf2.keras.metrics.AUC(name='auc')],
              loss='categorical_crossentropy',
              loss_weights=[20, 0.01, 20, 5, 0.01])

history = two_channels_model_resampled.fit(
    X_train_resampled, y_train_resampled, batch_size=64,validation_split=0.15,
epochs=300, shuffle=False
)

gc.collect()

_, _ = scoring(np.argmax(y_train, axis=1),
np.argmax(two_channels_model_resampled.predict(X_train), axis=1),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(np.argmax(y_test, axis=1),
np.argmax(two_channels_model_resampled.predict(X_test), axis=1),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

# -*- coding: utf-8 -*-

!pip install antropy

import os
os.environ['PYTHONHASHSEED']=str(225)
import numpy as np
import pandas
import glob
import scipy
```

```
#import antropy
import xgboost
import random

from imblearn.over_sampling import RandomOverSampler, SMOTE, SVMSMOTE
from imblearn.under_sampling import RandomUnderSampler, AllKNN

def reset_random_seeds():
   os.environ['PYTHONHASHSEED']=str(225)
   #tf2.random.set_seed(225)
   np.random.seed(225)
   random.seed(225)

from sklearn.model_selection import train_test_split

def get_data_from_channel(file_names, chname='EEG Fpz-Cz', return_y=True):
  all_data = []
  if return_y:
    all_labels = []

  for file in file_names:
    with np.load(file, allow_pickle=True) as f:
      data = f['x'].item()[chname]
      if return_y:
        labels = f['y']

    data = data.astype(np.float32)
    all_data.append(data)
    if return_y:
      labels = labels.astype(np.int32)
      all_labels.append(labels)

  all_data = np.vstack(all_data)
  if return_y:
    all_labels = np.hstack(all_labels)
    return all_data, all_labels
  return all_data


def compute_bandpower(data, fs=100, win=600, kwargs_welch=dict(average="median",
window="hamming"),
                      bands=[
                        (0.5, 4.5, "Delta"),
                        (4.5, 8.5, "Theta"),
                        (8.5, 11.5, "Alpha"),
                        (11.5, 15.5, "Sigma"),
                        (15.5, 30, "Beta")
                      ], relative=True):
  freqs, psd = scipy.signal.welch(data, fs, nperseg=win, **kwargs_welch)

  all_freqs = np.hstack([[b[0], b[1]] for b in bands])
```

```
  fmin, fmax = min(all_freqs), max(all_freqs)
  idx_good_freq = np.logical_and(freqs >= fmin, freqs <= fmax)
  freqs = freqs[idx_good_freq]
  res = freqs[1] - freqs[0]

  nsamples = psd.shape[0]
  bp = np.zeros((nsamples, len(bands)), dtype=np.float32)
  psd = psd[:, idx_good_freq]
  total_power = scipy.integrate.simpson(psd, dx=res)
  total_power = total_power[..., np.newaxis]

  labels = []
  for i, band in enumerate(bands):
      b0, b1, la = band
      labels.append(la)
      idx_band = np.logical_and(freqs >= b0, freqs <= b1)
      bp[:, i] = scipy.integrate.simpson(psd[:, idx_band], dx=res)
  if relative:
    bp /= total_power

  return bp

def compute_power_ratios(data):
  alpha_ratio = data[:, 2] / (data[:, 0] + data[:, 1])
  delta_ratio = data[:, 0] / (data[:, 2] + data[:, 1])
  theta_ratio = data[:, 1] / (data[:, 0] + data[:, 2])
  delta_ratio_2 = data[:, 0] / data[:, 1]
  alpha_ratio_2 = data[:, 2] / data[:, 1]
  return np.array([alpha_ratio, delta_ratio, theta_ratio, delta_ratio_2,
alpha_ratio_2]).T

# Commented out IPython magic to ensure Python compatibility.
import seaborn as sns
from sklearn.metrics import confusion_matrix, f1_score, precision_score,
recall_score, balanced_accuracy_score, matthews_corrcoef
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline

from IPython.display import clear_output

def compute_entropy_nonlinear_features(data):
  features = []

  for i, sample in enumerate(data):
    clear_output()
    print(f'Processing {i + 1}th sample...')
    features.append([
        antropy.higuchi_fd(sample),
        antropy.petrosian_fd(sample),
        antropy.katz_fd(sample),
```

```
        antropy.app_entropy(sample),
        *antropy.hjorth_params(sample),
        antropy.num_zerocross(sample),
        antropy.sample_entropy(sample),
        antropy.spectral_entropy(sample, sf=100)
    ])
  return np.array(features)

def compute_statistical_features(data):
  min_ = data.min(axis=1)
  max_ = data.max(axis=1)
  mean = data.mean(axis=1)
  std = data.std(axis=1)
  skewness = scipy.stats.skew(data, axis=1)
  kurtosis = scipy.stats.kurtosis(data, axis=1)
  median = np.median(data, axis=1)

  return np.array([min_, max_, mean, std, skewness, kurtosis, median]).T

from sklearn.preprocessing import StandardScaler

def get_features_set(eeg_data, eog_data, y, set='train'):
  bandpowers = compute_bandpower(eeg_data)
  ratios = compute_power_ratios(bandpowers)
  eeg_statistical = compute_statistical_features(eeg_data)
  eog_statistical = compute_statistical_features(eog_data)
  if set == 'train':
    eeg_other_features_file =
'/content/drive/MyDrive/features/entropy_train_features.npz'
    eog_other_features_file =
'/content/drive/MyDrive/features/entropy_train_features_eog.npz'
  else:
    eeg_other_features_file =
'/content/drive/MyDrive/features/entropy_test_features.npz'
    eog_other_features_file =
'/content/drive/MyDrive/features/entropy_test_features_eog.npz'
  with np.load(eeg_other_features_file, allow_pickle=True) as f:
        eeg_other_features = f['arr_0']
  with np.load(eog_other_features_file, allow_pickle=True) as f:
        eog_other_features = f['arr_0']

  features_set = np.concatenate([bandpowers, ratios, eeg_statistical,
eeg_other_features, eog_statistical, eog_other_features], axis=1)
  #features_set = StandardScaler().fit_transform(features_set)

  y = np.where(y == 1, 0, y)
  y = np.where(y == 2, 1, y)
  y = np.where(y == 3, 2, y)
  y = np.where(y == 4, 3, y)
  y = np.where(y == 5, 4, y)
```

```
  return features_set, y

reset_random_seeds()

npz_files_pattern = '/content/drive/MyDrive/diploma_data_edf/*.npz'
npz_files_names = glob.glob(npz_files_pattern)

train_files, test_files = train_test_split(npz_files_names, test_size=0.15,
random_state=100)

eeg_train, y_train = get_data_from_channel(train_files)
eog_train = get_data_from_channel(train_files, chname='EOG horizontal',
return_y=False)

features_names = np.array(['delta_bandpower', 'theta_bandpower', 'alpha_bandpower',
'sigma_bandpower', 'beta_bandpower', 'alpha_ratio', 'delta_ratio', 'theta_ratio',
'delta_ratio_2', 'alpha_ratio_2', 'eeg_min', 'eeg_max', 'eeg_mean', 'eeg_std',
'eeg_skew', 'eeg_kurtosis', 'eeg_median', 'eeg_higuchi_fd', 'eeg_petrosian_fd',
'eeg_katz_fd', 'eeg_app_entropy', 'eeg_hjorth_mobility', 'eeg_hjorth_complexity',
'eeg_num_zerocross', 'eeg_sample_entropy', 'eeg_spectral_entropy', 'eog_min',
'eog_max', 'eog_mean', 'eog_std', 'eog_skew', 'eog_kurtosis', 'eog_median',
'eog_higuchi_fd', 'eog_petrosian_fd', 'eog_katz_fd', 'eog_app_entropy',
'eog_hjorth_mobility', 'eog_hjorth_complexity', 'eog_num_zerocross',
'eog_sample_entropy', 'eog_spectral_entropy'])
X_train, y_train = get_features_set(eeg_train, eog_train, y_train)

#X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,
test_size=0.15, random_state=100)

X_train = StandardScaler().fit_transform(X_train)
#X_valid = StandardScaler().fit_transform(X_valid)

eeg_test, y_test = get_data_from_channel(test_files)
eog_test = get_data_from_channel(test_files, chname='EOG horizontal',
return_y=False)

X_test, y_test = get_features_set(eeg_test, eog_test, y_test, set='test')
X_test = StandardScaler().fit_transform(X_test)

data_df = pd.DataFrame(data=X_train,
                       columns=features_names)
plt.figure(figsize=(30,30))
sns.heatmap(data_df.corr(), annot=True, fmt='.2f')

data_df = pd.DataFrame(data=np.concatenate([X_test, y_test.reshape(-1, 1)],
axis=1),
                       columns=np.concatenate([features_names, ['target']]))
features_to_plot = ['delta_bandpower', 'theta_bandpower', 'alpha_bandpower',
'sigma_bandpower', 'beta_bandpower', 'eog_max', 'eog_katz_fd', 'target']
sns.pairplot(data_df[features_to_plot].sample(n=10000, random_state=1),
hue='target')
```

```
import gc
gc.collect()

from sklearn.model_selection import ParameterGrid, cross_val_score
grid_params = {
                'max_depth': [6, 7, 8],
                'gamma': [1, 2.5, 5, 7.5, 10],
                'min_child_weight': [1, 2.5, 5, 7.5, 10],
                'max_leaves': [0, 5, 10, 25, 50, 75]}

grid = ParameterGrid(grid_params)

for i, p in enumerate(grid):
  print(f'{i + 1}. Parameters: {p}')
  xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **p)
  scores = cross_val_score(xgb_cl, X_train, y_train, cv=6, scoring='f1_macro',
n_jobs=-1, verbose=3)

  print('\t' + f'Cross-validation f1: {scores.mean():.4f}')

d = {'feature_name': features_names, 'importance': xgb_cl.feature_importances_}
df = pd.DataFrame(data=d)

df.sort_values(by=['importance'], ascending=False)

sorted_indx = np.array(df.sort_values(by=['importance'], ascending=False).index)

best_params = {'gamma': 5, 'max_depth': 6, 'max_leaves': 30, 'min_child_weight': 5}
shift = 10
n_iters = sorted_indx.shape[0] - shift
for i in range(n_iters):
  print(f'{i + 1}. Selected features: {sorted_indx[0: i + 1 + shift]}')
  xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
  scores = cross_val_score(xgb_cl, X_train[:, sorted_indx[0: i + 2 + shift]],
y_train, cv=6, scoring='f1_macro', n_jobs=-1, verbose=3)

  print('\t' + f'Cross-validation f1: {scores.mean():.4f}')

"""## Базова модель"""

xgb_cl = xgboost.XGBClassifier()
xgb_cl.fit(X_train, y_train)

_, _ = scoring(y_train, xgb_cl.predict(X_train), print_res=True,  average=["micro",
"macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(y_test, xgb_cl.predict(X_test), print_res=True,  average=["micro",
"macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])
```

```
"""## Модель з підібраними гіперпараметрами"""

#{'gamma': 7.5, 'max_depth': 6, 'max_leaves': 0, 'min_child_weight': 7.5}
best_params = {'gamma': 5, 'max_depth': 6, 'max_leaves': 30, 'min_child_weight': 5}
#best_params = {'gamma': 7.5, 'max_depth': 6, 'max_leaves': 0, 'min_child_weight':
7.5}
xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
xgb_cl.fit(X_train, y_train)

_, _ = scoring(y_train, xgb_cl.predict(X_train), print_res=True,  average=["micro",
"macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(y_test, xgb_cl.predict(X_test), print_res=True,  average=["micro",
"macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

"""## Feature selection"""

selected_features = sorted_indx[0: 18 + 2 + 10]

#{'gamma': 7.5, 'max_depth': 6, 'max_leaves': 0, 'min_child_weight': 7.5}
best_params = {'gamma': 5, 'max_depth': 6, 'max_leaves': 30, 'min_child_weight': 5}
#best_params = {'gamma': 7.5, 'max_depth': 6, 'max_leaves': 0, 'min_child_weight':
7.5}
xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
xgb_cl.fit(X_train[:, selected_features], y_train)

_, _ = scoring(y_train, xgb_cl.predict(X_train[:, selected_features]),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(y_test, xgb_cl.predict(X_test[:, selected_features]),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

"""## Undersampling"""

label_to_name = {0: 'N1', 1: 'N2', 2: 'N3', 3: 'REM', 4: 'W'}
train_stages, train_counts = np.unique(y_train, return_counts=True)
test_stages, test_counts = np.unique(y_test, return_counts=True)

train_stat = [f"{label_to_name[stage]} - {count}" for stage, count in
zip(train_stages, train_counts)]
print(f'Train stages distributions: {train_stat}')
print(f'Test stages distributions: {[f"{label_to_name[stage]} - {count}" for stage,
count in zip(test_stages, test_counts)]}')

X_train_selected, X_test_selected = X_train[:, selected_features], X_test[:,
selected_features]

cc = AllKNN()
```

```
X_resampled, y_resampled = cc.fit_resample(X_train_selected, y_train)

train_stages, train_counts = np.unique(y_resampled, return_counts=True)

train_stat = [f"{label_to_name[stage]} - {count}" for stage, count in
zip(train_stages, train_counts)]
print(f'Train stages distributions: {train_stat}')

xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
xgb_cl.fit(X_resampled, y_resampled)

_, _ = scoring(y_train, xgb_cl.predict(X_train_selected),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(y_test, xgb_cl.predict(X_test_selected),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

"""## Oversampling"""

ros = SVMSMOTE(random_state=0, sampling_strategy={0: 15000})
X_resampled_2, y_resampled_2 = ros.fit_resample(X_resampled, y_resampled)

xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
xgb_cl.fit(X_resampled_2, y_resampled_2)

_, _ = scoring(y_train, xgb_cl.predict(X_train_selected),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

_, _ = scoring(y_test, xgb_cl.predict(X_test_selected),
print_res=True,  average=["micro", "macro"], labels=['N1', 'N2', 'N3', 'REM', 'W'])

"""## Three-class problem"""

three_class_y = np.where(y_train == 1, 0, y_train)
three_class_y = np.where(three_class_y == 2, 0, three_class_y)
three_class_y = np.where(three_class_y == 3, 1, three_class_y)
three_class_y = np.where(three_class_y == 4, 2, three_class_y)

three_class_y_test = np.where(y_test == 1, 0, y_test)
three_class_y_test = np.where(three_class_y_test == 2, 0, three_class_y_test)
three_class_y_test = np.where(three_class_y_test == 3, 1, three_class_y_test)
three_class_y_test = np.where(three_class_y_test == 4, 2, three_class_y_test)

xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
xgb_cl.fit(X_train_selected, three_class_y)

_, _ = scoring(three_class_y, xgb_cl.predict(X_train_selected),
print_res=True,  average=["micro", "macro"], labels=['NREM', 'REM', 'W'])
```

```
_, _ = scoring(three_class_y_test, xgb_cl.predict(X_test_selected),
print_res=True,  average=["micro", "macro"], labels=['N1', 'REM', 'W'])

two_class_y = np.where(y_train == 1, 0, y_train)
two_class_y = np.where(two_class_y == 2, 0, two_class_y)
two_class_y = np.where(two_class_y == 3, 0, two_class_y)
two_class_y = np.where(two_class_y == 4, 1, two_class_y)

two_class_y_test = np.where(y_test == 1, 0, y_test)
two_class_y_test = np.where(two_class_y_test == 2, 0, two_class_y_test)
two_class_y_test = np.where(two_class_y_test == 3, 0, two_class_y_test)
two_class_y_test = np.where(two_class_y_test == 4, 1, two_class_y_test)

xgb_cl = xgboost.XGBClassifier(n_estimators=100, grow_policy='depthwise',
tree_method='hist', **best_params)
xgb_cl.fit(X_train_selected, two_class_y)

_, _ = scoring(two_class_y, xgb_cl.predict(X_train_selected),
print_res=True,  average=["micro", "macro"], labels=['Sleep', 'Awake'])

_, _ = scoring(two_class_y_test, xgb_cl.predict(X_test_selected),
print_res=True,  average=["micro", "macro"], labels=['Sleep', 'Awake'])
```