Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ КЛАСТЕРИЗАЦІЇ ЧАСОВИХ РЯДІВ В ОНЛАЙН-РЕЖИМІ У ВИГЛЯДІ РЯДІВ ЕНЕРГОСПОЖИВАННЯ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-1
Скляров М.С.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник асист. Кобилін І.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Склярову Максиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільного застосунку для кластеризації часових рядів в онлайн-режимі у вигляді рядів енергоспоживання

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 29 травня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, дані інтернет-мережі, мова програмування C#, середовище розробки Microsoft Visual Studio 2022.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих методів кластеризації даних, та алгоритмів машинного навчання.
2. Моделювання структури програмного застосунку з кластеризації часових рядів.
3. Розроблення застосунку з кластеризації часових рядів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми обробки зображень, постановка задачі, методи кластеризації даних, інтерфейс застосунку, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-19.04.23	
3	Аналіз літератури з досліджуваної проблеми	19.04.23-24.04.23	
4	Аналіз технічних засобів	25.04.23-30.04.23	
5	Розробка методів	01.05.23-14.05.23	
6	Програмна реалізація	15.05.23-23.05.23	
7	Оформлення пояснювальної записки	24.05.23-28.05.23	
8	Перевірка на плагіат	29.05.23	
9	Рецензування	30.05.23	
10	Підготовка презентації та доповіді	31.05.23-03.06.23	
11	Занесення роботи в електронний архів	04.05.23	
12	Попередній захист кваліфікаційної роботи	06.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ _____асист. Кобилін І.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 66 с., 40 рис., 35 джерел.

КЛАСТЕРИЗАЦІЯ ЧАСОВИХ РЯДІВ, АНАЛІЗ ДАНИХ, РЯДИ ЕНЕРГОСПОЖИВАННЯ, K-MEANS, DYNAMIC TIME WARPING, DBSCAN, ГРАДІЄНТНИЙ СПУСК.

Об’єктом роботи є дослідження часових рядів у вигляді енергоспоживання та використання інтелектуальних технологій у вигляді штучних нейронних мереж, і їх поєднання для подальшого використання в онлайн-режимі.

Метою роботи є аналіз часових рядів, за допомогою кластеризації та використання штучних нейронних мереж. Пошук та обробка даних, які, потенційно, були отримані в ході припущення помилки збору інформації.

Використано методи обробки та аналізу даних. Проведено дослідження існуючих методів навчання та кластеризації.

У результаті роботи здійснена програмна реалізація мобільного застосунку для онлайн кластеризації часових рядів енергоспоживання в Україні.

TIME SERIES CLUSTERING, DATA ANALYSIS, ENERGY CONSUMPTION SERIES, K-MEANS, DYNAMIC TIME WARPING, DBSCAN, GRADIENT DESCENT.

The object of the work is the study of time series in the form of energy consumption and the use of intelligent technologies in the form of artificial neural networks, and their combination for further use in the online mode.

The purpose of the work is the analysis of time series using clustering and the use of artificial neural networks. Finding and processing data that is potentially obtained in the course of assuming a collection error.

Data processing and analysis methods were used. A study of existing training and clustering methods was conducted.

As a result of the work, a software implementation of a mobile application for online clustering of time series of energy consumption in Ukraine was carried out.

ЗМІСТ

ВСТУП

Часові ряди – структура даних, яку можна назвати найбільш розповсюдженою у світі. За простотою будови та розуміння приховуються безмежні напрямки розвитку інформаційних технологій та невичерпний потенціал.

Кластеризація та обробка великих масивів даних у вигляді часових рядів надає дослідникові необхідний ключ до фільтрування потоків інформації, залишаючи лише необхідне. На даний момент подібні процеси вже активно використовуються у різних сферах діяльності, наприклад в економіці.

Інтелектуальні технології станом на зараз вже набули широкого розповсюдження та активно розвиваються. Такого виду системи здатні за доволі короткий проміжок часу опанувати закономірності того чи іншого формату даних, та надалі використовувати отримані «знання» на практиці автоматично оброблюючи усю введену інформацію таким чином, який є необхідним для отримання того чи іншого результату.

Актуальність даної роботи полягає у створенні можливості та застосування системи аналізу енергоспоживання та енерговитрат у країні на основі даних у вигляді часових рядів, що представлені у формі рядів енергоспоживання попередніх років з урахуванням можливих надзвичайних ситуацій та аномалій. Аномалій на основі критичних відключень, некоректного зняття результатів з пристроїв, також воєнних дій, руйнування критичної інфраструктури, глобальні катаклізми, електромагнітні імпульси, непропорційне та незважене використання електроенергії.

1 ОГЛЯД МЕТОДІВ, МОЖЛИВИХ ДО ВИКОРИСТАННЯ В РОБОТІ

1.1 Поняття про часовий ряд

Часовий ряд – форма представлення інформації у вигляді значень набору певних параметрів, зібраних через певні проміжки часу. Таким чином часовим рядом можна назвати процес зміни стану певного вимірюваного явища у часі. Кожне вимірювання ряду має назву «відлік», і, в залежності від кількості параметрів у такому відліці, розрізняють одновимірні та багатовимірні часові ряди [1]. Проміжки при вимірюванні можуть бути рівномірними, тоді такі ряди називаємо дискретним, в іншому випадку ряд матиме характеристику «безперервний». Також поділяють шлях отримання даних: якщо виміри були зроблені під час виконання деякої заздалегідь відомої функції, то такий ряд називаємо детермінованим, якщо ж навпаки, усі дані це результат випадкового процесу то і ряд матиме назву «випадковий». На рисунку 1.1 приведено приклад одновимірного дискретного випадкового часового ряду.

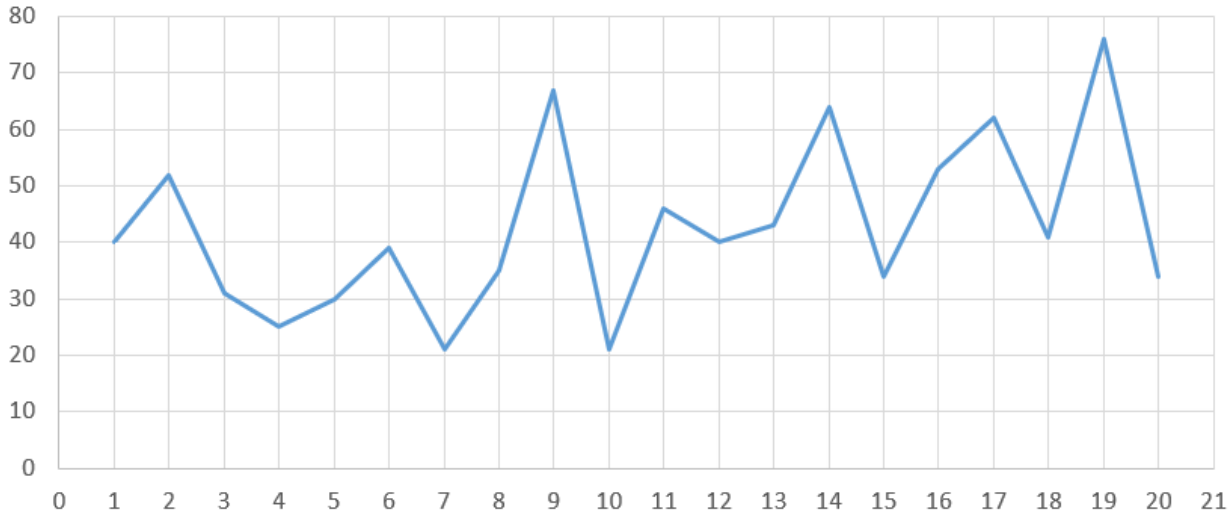


Рисунок 1.1 – Приклад часового ряду

В даній роботі буде проводитись кластеризація та аналіз часових рядів енергоживлення України, тож пропоную ввести нове поняття – короткий часовий ряд. Короткий часовий ряд [2] – такий ряд, що має кількість параметрів порівнянну з кількістю точок. Подібний підхід для розподілення часових рядів дозволяє точно візуалізувати усі оброблені дані таким чином який забезпечить недопущення втрати інформації [3].

При аналізі часових рядів основоною задачею виступає знаходження та аналіз мотивів часового ряду. Вирішення такого типу задач, дозволяє збирати інформацію, на основі якої аналітично можна пояснити дані представлені дослідником [4]. З роботою по знаходженню та опису мотивів тісно пов’язаний термін патернів часових рядів.

1.1.1 Патерни часових рядів

Патерни часових рядів описують загальну поведінку протягом майже всього ряду. Виділення патерну, або їх набору, є основою аналізу часових рядів. Завдяки цьому, можна аналітично пояснювати зміни спостережуваного процесу, а також шукати недоліки у роботі, з подальгим їх усуненням. Всього виділяють три типи патернів часових рядів: тренд, сезонний, циклічний. Кожен з них має свою особливість, та відіграє свою роль при поясненні процесів. Важливою особливістю патернів є те, що вони можуть бути комбіновані, тому процес їх виділення може бути дещо ускладненим.

Патерн «Тренд» спостерігається, коли наявний поступовий спад, або підйом значень даних. При цьому лінійність зміни не є обов’язковою умовою [5]. Приклад лінійного патерну показаний на рисунку 1.2.

Сезонний характер поведінки часового ряду виникає, коли на об’єкт спостереження безпосередньо впливають сезонні фактори. Серед таких можна виділити наприклад зміну пори року або часового проміжку в добі [5].

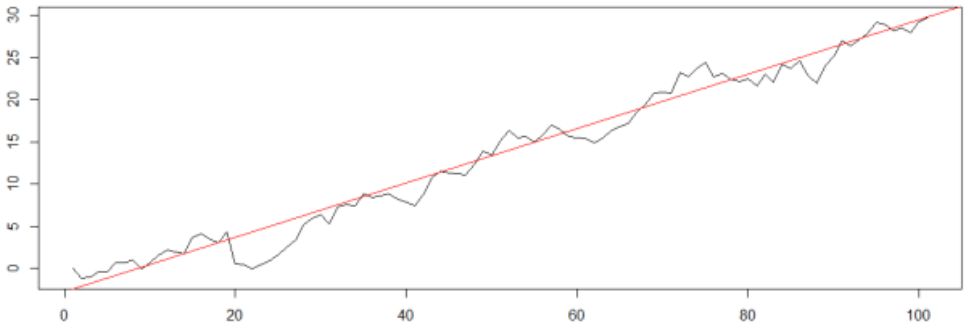


Рисунок 1.2 – Зростаючий патерн

Візуально сезонність можна відрізнити від інших патернів поведінки тим, що рівномірно розділені частини ряду як правило схожі між собою, що і візуалізує вплив сезону [6]. На рисунку 1.3 зображено приклад сезонного патерну.

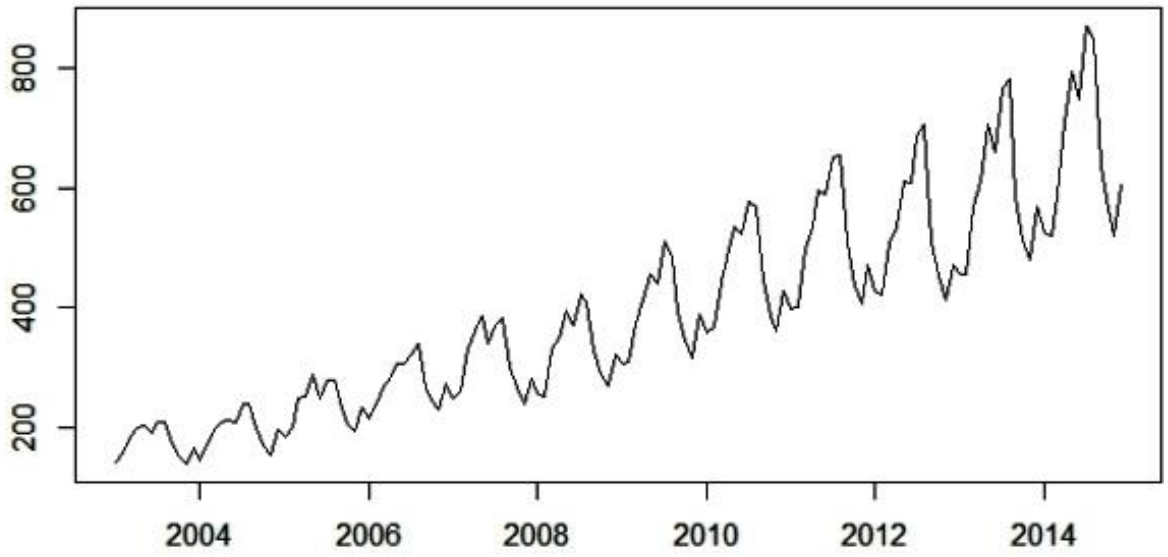


Рисунок 1.3 – Патерн «Сезонний»

Цикл у поведінці спостережуваного явища виникає тоді, коли зрости та падіння його вимірюваних значень поперемінно змінюють один одного. На відміну від сезонного патерну, цикл не має певної частоти зміни направлення [5]. Візуальний вигляд приймає форму дещо хаотичної кривої. Та насправді його поведінка обумовлена непостійними критеріями. Наприклад якщо

спостереження ведеться у сфері продажів наприклад, то повенка ряду буде залежати від економічних процесів. Також на відміну від сезонної поведінки, зважаючи на вплив інших чинників, найкраще циклічний патерн проявляє себе у спостереженнях від двох років. На рисунку 1.4 представлено приклад циклічної поведінки часового ряду.

Рисунок 1.4 – Циклічний патерн

1.1.2 Мотиви часових рядів

Мотивами часових рядів називають набір декількох рядів, або деяких більш малих частин одного ряду, котрі неозброєним оком можна прийняти за ідентичні. Виявлення мотивів становить провідну задачу в аналізі часових рядів, через те, що така інформація дає змогу робити висновки про впливи різних чинників на об’єкт, та подальшу обробку цих даних [7]. Як правило у виділених мотивах простежується однакова довжина та комбінація різних патернів поведінки часових рядів. Виявлені ж патерни в свою чергу можуть вказати на природу впливу. На рисунку 1.5 зображено часовий ряд зі знайденими на ньому мотивами, та порівняння цих мотивів між собою.

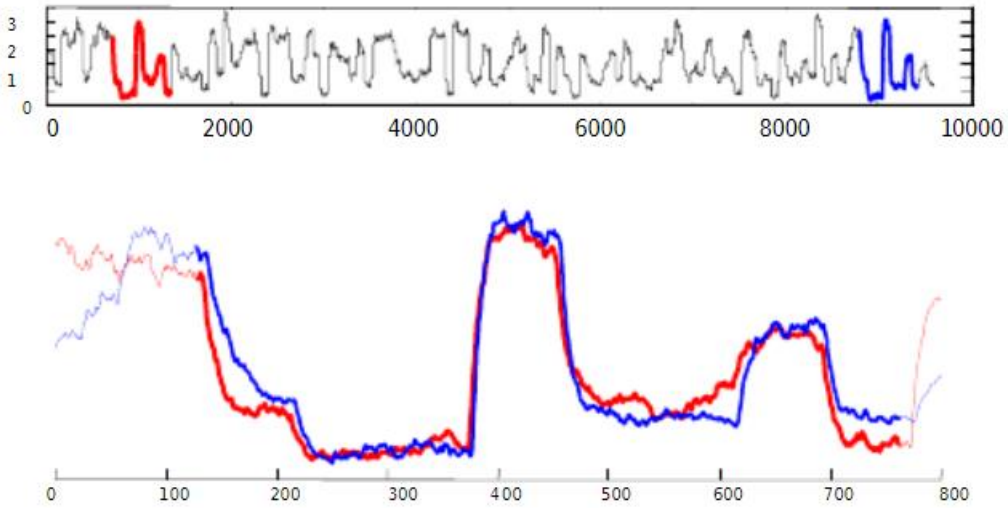


Рисунок 1.5 – Часовий ряд з його мотивом

1.2 Поняття про кластеризацію

Кластеризація – процес розподілення набору даних на підгрупи, так звані кластери, на основі деякої спільної характеристики, часто відстані від центру кластера [8]. Дана процедура так сильно закріпилась у колі аналітиків статистичних даних, що на сьогодні є обов'язковою для вивчення. А методології та використання кластеризаційних алгоритмів дедалі глибше проникають у такі науки як аналіз зображень, машинне навчання та розпізнавання патернів, тощо. Більше того, дані галузі науки сьогодні просто неможливо уявити без кластерних алгоритмів.

Як було згадано вище, після виконання процесу кластеризації ми отримуємо деякі кластери. Кластер – це, в найбільш широкому сенсі, група об'єктів яка має між собою більше спільностей, ніж з об'єктами в інших кластерах [9]. Подібність, як правило, визначається математично. Тобто це деяка штучно створена, або аналітично знайдена величина, розрахунок якої під час виконання процесу кластеризації вирішує величину подібності. Часто, за таку величину приймають деяку норму відстані – найбільш простий та зрозумілий критерій подібності. Евклідова відстань – один з прикладів найбільш вживаних.

Глобально методи кластеризації поділяються на ієрархічні та неієрархічні [10]. Неієрархічні методи додатково поділяються на два види: перший вид – жорстка кластеризація, другий – м’яка.

1.2.1 Жорстка кластеризація

Жорстка кластеризація – такий вид кластеризації, при якому, усі дані діляться групи, таким чином, щоб кожна точка, або кожен вимір належали одному і тільки одному кластеру [8]. Найбільш яскравим прикладом застосування жорсткої кластеризації це застосування методу *K*-Means. Даний метод є дуже простим у реалізації та використанні, проте має ряд суперечливих моментів. З іншого боку, жорстка направленість роботи алгоритму дозволяє обробляти значні об’єми даних, майже не втрачаючи ефективності.

1.2.2 М’яка кластеризація

М’яка кластеризація – розподілення точок даних так, що кожна з них не матиме чіткої приналежності до кластера. Умовно кажучи, кожна точка буде належати кожному кластеру, але лише частково [11]. Найбільш точно суть методу передає алгоритм Fuzzy *C*-Means. Цей метод, котрий за принципом дії дуже схожий на алгоритм *K*-Means, при роботі створює матрицю вірогідностей, за якою і відбувається розподіл на кластери. Варто зазначити, що використання м’якої кластеризації більш природнє, через те, що має більше спільного з реальним життям. Так як часто знаходяться дані, які в певних умовах неможливо віднести до тієї чи іншої групи.

1.2.3 Ієрархічні методи

Ієрархічні методи кластеризації – це такі алгоритми, які здатні з декількох маленьких кластерів зібрати один великий, такі алгоритми мають назву агломеративні. Інші ж навпаки, розбивають один великий кластер на декілька малих, це – дивизимні, або розділяючі алгоритми [12]. Візуально результат роботи обох видів методів зображують як дендрограму [13].

Але, для даної роботи жоден з ієрархічних методів не може бути використаний. Вони просто не придатні оброблювати великі потоки інформації. А ось неієрархічні будуть працювати добре, бо вони дроблять масиви даних на кластери до тих пір, доки не буде досягнуто правила зупинки. Такі правила бувають двох видів: перший – мінімізація значення відмінності між кластерами, такий підхід характерний для алгоритму Fuzzy *C*-Means, другий спосіб – визначення кластерів у місцях скупчення точок, та подальшого розподілення, тих даних що залишились. Такий підхід повністю характеризує алгоритм *K*-Means.

1.3 *K*-Means

Алгоритм *K*-Means – алгоритм жорсткої кластеризації даних. Ітераційний процес, що розподіляє введений масив даних на заздалегідь визначену кількість кластерів. За результатами роботи алгоритму кожна точка масиву матиме приналежність лише одному кластеру. Таким чином можна сказати що при накладанні один на одного такі кластери не будуть перекривати один одного [14, 15].

Інструкція роботи алгоритму *K*-Means має наступний вигляд:

– вказати необхідну кількість *K* кластерів;

– випадковим чином призначити *K* точок центроїдами кластерів;

– на основі обраних центроїдів розподілити усі інші точки по кластерам;

– перерахувати значення кластерів шляхом виведення середнього значення для кожного з вимірів;

– перевірити рівність набору центроїдів поточних та набору центроїдів попередніх. Якщо співпадіння повне, то виконати наступний крок, якщо ні – знову розподілити дані по групам на основі нових центроїдів;

– завершити виконання алгоритму.

Взявши до уваги пункт алгоритму, де ми випадково обираємо початкові центри кластерів, а також провівши ряд нескладних але наглядних тестів, можна з упевненістю сказати: з кожною ініціалізацією алгоритму кінцевий результат буде відрізнятись. Тому для найбільш точного значення виходу, рекомендується проводити декілька ініціалізацій алгоритму, та серед результатів обирати той, що при запуску надав найменшу відстань між кластерами.

1.4 Fuzzy *C*-Means

Алгоритм Fuzzy *C*-Means – алгоритм кластеризації даних, при якому кожна точка даних не належить певному класу або кластеру. Натомість така точка має власну матрицю приналежності, що має характер вірогідності [16]. Таким чином реалізується так званий підхід м’якої кластеризації.

Алгоритм має наступні кроки виконання:

– вказати необхідну кількість *C* кластерів та межу зміни;

– випадковим чином *C* точок центроїдами кластерів;

– проініціалізувати матрицю приналежності. Кожен елемент розраховується як вірогідність приналежності тому чи іншому класу (чим точка ближче до певного центроїду, тим вірогідніше вона буде йому належати). Важливо пам’ятати, що значення матриці відображають вірогідність, тому сума значень для однієї точки не може перевищувати 1;

– підрахувати нове значення центроїдів, яке буде виступати в якості середнього зваженого усіх точок даних. При цьому вагою виступає поточна вірогідність приналежності точки кластеру;

– перевірити зміну значень центроїдів. Якщо різниця значень поточного набору центроїдів та попереднього менше за вказану межу зміни то завершуємо алгоритм. Інакше перераховуємо дані по групам на основі нових центроїдів.

Таким чином можемо спостерігати одночасно схожість та відміну з алгоритмом *K*-Means. З однакового маємо випадкову ініціалізацію початкових кластерів, що при множинних ініціалізаціях буде видавати різні результати. А також досить добре виділяється ітераційна система, яка в основному має однаковий характер.

З відмінностей, варто зазначити дещо інший критерій зупинки алгоритму. Значення межі зміни, заданої на початку виконання алгоритму може докорінно змінити результат. А також появу матриці вірогідностей, яка і створює найбільшу відмінність від алгоритму *K*-Means – перехід до м’якої кластеризації. Як вже було описано у методі виконання алгоритму, точка не належить тільки до одного класу, вона має певний ступінь приналежності до всіх класів.

1.5 Алгоритм DBSCAN

DBSCAN – алгоритм призначений, як і *K*-Means та *C*-Means, для групування даних по кластерам. Даний метод дозволяє створювати різну кількість груп, яка залежить лише від вхідних параметрів. Але найважливішою перевагою DBSCAN вважається його здатність вираховувати точки, що не належать жодному кластеру. Це так звані шуми [17]. Розглянемо основний алгоритм роботи [18]:

– визначити вхідні налаштування алгоритму *minPts* та *eps,* що позначають мінімальну кількість сусідів для формування кластеру, та радіус пошуку сусідніх точок даних відповідно;

– випадково обрати точку в даних та перевірити скільки вона має сусідів. Якщо кількість сусідів менша за вказану мінімальну, то така точка позначається як шум та обирається інша. В іншому випадку можемо почати формувати новий кластер;

– проходимо кожну точку, що виявилась сусідом попередньої, та повторюємо процедуру. Якщо ж і тут вимоги налаштувань виконані, то ця точка додається до класу. Даний пункт виконується до тих пір доки знаходяться сусіди для даного набору точок;

– далі обирається випадкова точка з набору ще не пройдених, та весь алгоритм повторюється доки не буде оброблений весь набір даних.

Як бачимо, хоч даний алгоритм і базується на розрахунку відстані між точками, як це було в алгоритмах *K*-Means та *C*-Means, та все ж принцип роботи дозволяє отримати зовсім інші результати на однакових даних у порівнянні з цим методами. Саме це і дозволяє DBSCAN знаходити точки шуму поміж даних, що стане невід’ємною частиною роботи.

Та все ж, алгоритм не ідеальний. З головних проблем виділяється застосована метрика відстані. Від неї, як правило, залежить успіх роботи програми зі спеціальними видами даних. В нашому випадку, метрика евклідової відстані не дозволяє досить точно обрахувати близькість двох часових рядів. Отже необхідно застосувати, вже згаданий, алгоритм DTW, який проявить себе краще в роботі.

1.6 Dynamic Time Warping

Алгоритм DTW – алгоритм порівняння двох часових рядів на предмет виявлення між ними оптимального співпадіння [19]. Алгоритм досить

гнучкий, бо дозволяє порівнювати ряди, які відрізняються між собою як за амплітудою значень осі Y, так і за кроком та протяжністю осі X. Таким чином Алгоритм дозволяє проводити кластеризацію у часових рядах які відрізняються між собою за конфігурацією.

Алгоритм має наступну інструкцію:

– створити матрицю розмірності ($M \times N$), де M розмір першого часового ряду, а N – другого;

– далі необхідно заповнити матрицю. Для цього кожна позиція у матриці рахується як евклідова відстань між відповідними точками двох часових рядів;

– знаходимо шлях трансформації. Це значення представляє собою шлях між початковою точкою матриці D_{00} та останньою D_{mn}. Розрахунок завжди проходить через суміжні клітини;

– на основі порахованого шляху трансформації, отримуємо вартість шляху.

Даний алгоритм дозволяє досить чітко визначити схожість двох часових рядів, приймаючи до уваги їх несхожість в плані розмірностей та амплітуд значень по осях. Водночас, сам по собі алгоритм досить чутливий до довжини порівнюваних рядів, через складність розрахунку, а саме O(mn). Та, на щастя, існує ряд покращень алгоритму які можливо імплементувати до нього, аби зробити більш точним, та стійким до різного роду помилок.

1.7 Постановка задачі

Таким чином, кластеризація та класифікація даних різними способи є актуальним завданням для дослідження та обробки часових рядів. Тому ставиться задача дослідження роботи різних алгоритмів кластеризації, та застосування результатів для вирішення прикладної задачі дослідження часових рядів.

Об’єктом роботи є дослідження часових рядів у вигляді енергоспоживання та використання інтелектуальних технологій у вигляді штучних нейронних мереж, і їх поєднання для подальшого використання в онлайн-режимі.

Метою роботи є аналіз часових рядів, за допомогою кластеризації та використання штучних нейронних мереж.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати вже розроблені методи кластеризації часових рядів, а також проблеми та недоліки пов’язані з ними;

– вивчити тонкощі застосування алгоритмів *K*-Means, Fuzzy *C*-Means, DTW;

– програмно реалізувати дані методи у середовищі Visual Studio 2022;

– провести тестування розроблених методів на основі даних електроспоживання України за останні 20 років.

2 ОГЛЯД МЕТОДІВ

Специфіка роботи та формат даних показують, що для роботи можливо використовувати алгоритми описані вище. Таким чином буде забезпечуватись належна обробка інформації, а також швидкодійність роботи програми. В даному розділі ми більш детально зупинимось на кожному з алгоритмів, для того щоб розглянути принципи її роботи.

2.1 Огляд методу *K*-Means

Розглянемо алгоритм *K*-Means, а також основні особливості використання. Застосування на різних видах та формах даних, серед яких використання набору геометричних координат, набору одновимірних чисел, часових рядів. Зокрема для часових рядів також буде розглянуто можливість поєднання з алгоритмом DTW.

В основі алгоритму *K*-Means лежить принцип прорахунку відстані між вимірами даних. Як правило рахується відстань від конкретної точки до центру кластера. В залежності від даних які підлягають кластеризації центр називається «центроїдом» або «кластроїдом». Приклад кластеризації даних за допомогою даного алгоритму показано на рисунку 2.1.

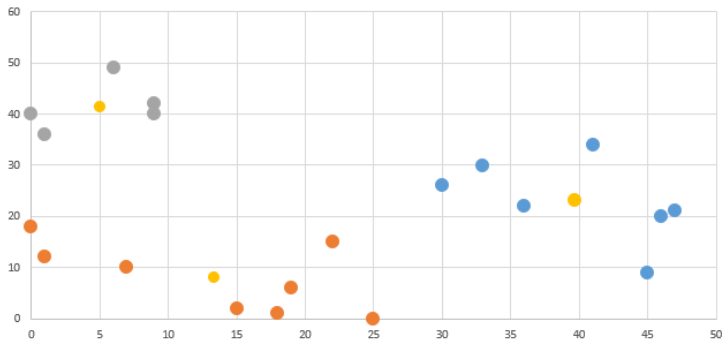


Рисунок 2.1 – Робота алгоритму *K*-Means

Найчастіше для визначення відстані за міру близькості використовують значення евклідової відстані [20], загальна формула якої виглядає так:

$$d_E(X_k, X_n) = \sqrt{\sum_{i=1}^{m} (x_{ik} - x_{in})^2},$$

де X_k X_n – часові ряди.

Насправді дана формула є універсальною та дозволяє обраховувати відстань як між часовими рядами, так і між звичайними координатами з геометрії. В такому окремому випадку єдиною різницею від часових рядів, буде наявність лише двох вимірів – координат *X* та *Y*. Також дану формулу можна використовувати для підрахунку відстані між окремими двома числами. Але в такому випадку результат зведеться до модуля різниці. Тож використання цієї формули для обрахунку окремих чисел не має раціональності.

Повернемось до алгоритму *K*-Means. Основна суть методу полягає в тому, що на першому кроці задаються значення центрів кластерів. Як правило, під час виконання даного кроку за центри кластерів видають випадкові спостереження з усієї множини. Далі, маючи початкові центроїди, усі інші спостереження групуються, використовуючи задану метрику відстані, на приклад евклідову відстань. Потім, для кожного новоутвореного кластеру спостережень вираховується новий центр. Дане значення замінює попереднє значення центроїду для цього кластеру. З новими центрами кластерів повторюємо процедуру групування та вираховування центрів. Процес повторюється до тих пір доки, змінна значень центроїдів на стане занадто малою, або не припиниться зовсім. Загалом, можна сказати що метою алгоритму вважається мінімізація суми квадратів відхилень (відстані) кожного спостереження від центроїду цього класу:

$$\min\left(\sum_{i=1}^{n}\sum_{x_j=S_i}\left[x_j-\mu_i\right]^2\right),$$

де n – число кластерів;

S_i – отримані кластери;

$x_j \in S_i$;

μ_i – центроїди кластерів;

i=1,2,3…n.

Незважаючи на свою простоту використання, алгоритм має ряд недоліків. Перш за все, використовуючи алгоритм у «чистому» вигляді, тобто без поєднання з іншими методами, необхідно самостійно задавати кількість необхідних кластерів. Така умова не завжди є доречною, особливою коли саме завдяки кластеризації необхідно дізнатись на які групи діляться дані. По-друге, необхідно відмітити те, що ініціалізація перших центроїдів відбувається довільно. Такий підхід призводить до того, що результат кластеризації буде відрізнятись від одного запуску до іншого. Тому неможна однозначно стверджувати про точність розподілу даних після декількох запусків. Також не треба забувати, що *K*-Means – алгоритм жорсткої кластеризації. Це означає, що дані, у випадку, коли створюється ситуація, що мається явна приналежність до декількох класів, будуть віднесені лише до одного з них. Таким чином втрачається значна кількість інформації, що може завадити правильній аналітиці.

Найбільш слабким місцем даного методу є використовувана метрика відстані. У прикладі наведено використання евклідової відстані за визначення міри близькості. Але такий підхід зовсім недієздатний за умови використання часових рядів як сировини для кластеризації. Часові ряди не підходять для евклідової відстані з декількох причин. По-перше, у рядах, як правило, використовують виміри різних величин, які неможливо просто порівнювати як

арифметичні середні значення. По-друге, значною проблемою постає довжина рядів. Використовуючи евклідову відстань за метрику близькості, ми ризикуємо втрачати дані, що призведе до погіршення аналітичної складової процесу. По-третє, застосування евклідової відстані не враховує можливого зміщення ряду у часі. Таким чином, якщо, на приклад ми порівняємо який-небудь процес, спостереження за яким відбувались протягом тижня, але в першому випадку відлік починався з понеділка, а в іншому з четверга, то порівнявши ці два, ідентичні в основі, але зміщенні у часі ряди, то отримаємо майже повну невідповідність один одному, хоча відомо що насправді ряди однакові.

Використання методу Dynamic Time Warping, або DTW, допоможе більш коректно порівнювати часові ряди, уникаючи проблем, описаних вище. Слід відмітити, що алгоритм в своїй основі також використовує вимірювання евклідової відстані, але оболонка алгоритму дозволяє виконувати це більш раціонально.

2.2 Огляд методу *C*-Means

Розглянемо приклад, так званої, м’якої кластеризації алгоритм *C*-Means. Його особливості використання, можливі дані. Також його відмінності від попереднього алгоритму *K*-Means.

Як можна зрозуміти з назви методу, *C*-Means значною мірою схожий на *K*-Means. Такий висновок, значною мірою, буде вірним. Хоча й існують деякі малі та великі розбіжності [21]. Як і його попередник, алгоритм спирається на прорахунок деякої міри близькості поміж набору даних. Такий прорахунок відбувається заданим методом знаходження відстані. Для прикладу, знову, будемо брати евклідову відстань.

Розпочнемо з головної різниці між алгоритмами, бо всі більш незначні відмінності виходять з основної. М’яка кластеризація, представником якої є

алгоритм *C*-Means, припускає приналежність оброблюваних даних до декількох класів. Таким чином, якщо за певним збігом обставин, ми маємо спостереження яке точно можна віднести як до одного так і до іншого класу, то даний алгоритм покаже це, та аналітика даних буде проходити більш точно. Візуально різницю такого підходу від алгоритму *K*-Means показано на рисунку 2.2 [22].

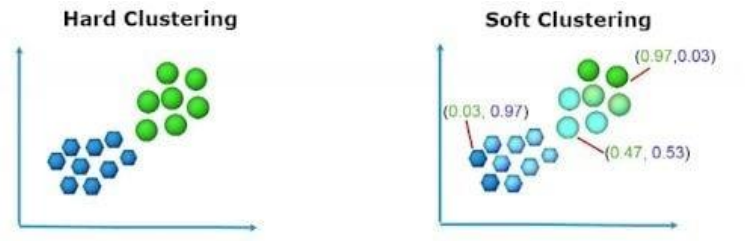


Рисунок 2.2 – Різниця результату роботи алгоритмів *K*-Means та *C*-Means

Суть підходу алгоритму полягає в тому, що під час роботи, замість додавання точки до одного з кластерів, будується матриця приналежності з якої можна побачити вірогідність приналежності даного виміру до конкретного класу [23]. Для прикладу припустимо, що ми маємо точку *A* яка в ході роботи алгоритму має 60% приналежності до першого класу та 40% до другого. В такому випадку можемо стверджувати, що точка *A* належить більше першому класу аніж другому, при цьому ми не виключаємо деяку схожість с другим класом. Форму поділу даних на класи та побудову матриць відносності можна загально представити так:

$$\min\left(\sum_{i=1}^{C}\sum_{j=1}^{N}\omega_{ij}^{m}\left[x_j-\mu_i\right]^2\right),$$

де C – число кластерів;

N – кількість спостережень;

μ_i – центроїди кластерів;

i =1,2,3…C;

j=1,2,3…N;

ω_{ij}^{m} – міра приналежності до класу.

В свою чергу вона розраховується за наступною формулою:

$$\omega_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{x_j - \mu_i}{x_j - \mu_k}\right)^{\frac{2}{m-1}}},$$

де C – число кластерів;

$x_j \in S$;

μ_i, μ_k – центроїди кластерів;

$m \in (1, 2, \dots \infty)$.

В даній формулі з’являється важливий для всього алгоритму параметр m. Він представляє собою міру нечіткості роботи алгоритму, тобто визначає наскільки сильно кожен елемент даних може належати декільком класам. Найчастіше даному параметру задають значення 2, це дозволяє проводити кластеризацію в комфортних умовах виключаючи зайву розмитість поділу. Також варто зауважити, що у випадку, коли значення m встановлюється на значення 1, то алгоритм перетворюється на стандартний *K*-Means. Це означає, що матриця приналежності, створена як результат роботи алгоритму, буде мати лише значення 0 та 1, показуючи приналежність елемента лише одному класу.

Окремої уваги заслуговує метод обрахунку центрів кластерів. Він також потребує обрахованої міри близькості ω. В загальному вигляді формула центроїдів алгоритму *C*-Means має наступний вигляд:

$$c_i = \frac{\sum_{j=1}^{N} \omega_{ij}^{m} x_j}{\sum_{j=1}^{N} \omega_{ij}^{m}},$$

де C – число кластерів;

N – кількість даних;

μ_i, μ_k – центроїди кластерів;

$m \in (1, 2, \ldots \infty)$.

З недоліків алгоритму залишається така ж проблема як і в алгоритму *K*-Means. А саме випадковість у визначенні початкових центроїдів кластерів. Через це, результат роботи алгоритму на однакових даних буде відрізнятись при кожному запуску.

Також простежується така ж необхідність спеціального підходу до роботи з часовими рядами як і в алгоритму *K*-Means, тому доречним буде імплементація методу DTW, як покращену міру близькості, та забезпечення більш точного порівняння рядів між собою.

2.3 Огляд алгоритму DBSCAN

DBSCAN – використовується для просторової кластеризації даних на основі щільності [24]. Це означає, що програма під час поділу даних по групам, обраховує кількість точок що знаходяться поруч на задану відстань. Також необхідно, що коло поточної точки знаходилось не менш ніж *minPts* сусідів, бо інакше точка буде вважатись шумом та не потрапить до кластеру. Візуалізація роботи алгоритму на певному наборі даних продемонстрована на рисунку 2.3.

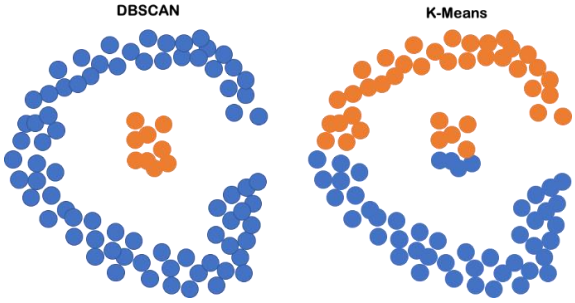


Рисунок 2.3 – робота алгоритму DBSCAN у порівнянні з *K*-Means

Суть алгоритму полягає в ітеративному розрахунку та акумулюванні кількості сусідів для однієї випадкової точки, для того щоб визначити, чи може ця точка, теоретично, бути частиною якогось кластеру. Для того щоб зрозуміти приналежність точки якійсь групі, нам необхідно вказати два параметри пошуку: по-перше, це кількість сусідів, по-друге, це радіус пошуку. Кількість сусідів *minPts* необхідна для того, або алгоритм зрозумів, чи достатньо щільна ділянка інформації на зараз проходить обробку. Щільність, в свою чергу, і розраховується з кількості найближчих сусідів. Радіус пошуку задає діапазон, в якому теоретично повинні знаходитись сусіди. Усі точки, що не підпадають у таку область сусідами не вважаються. Інші ж підраховуються і потім їх кількість порівнюється з заданим мінімальним числом сусідів.

Нажаль, алгоритм, попри свою важливість та простоту має недоліки [25], що можуть вплинути на процес побудови програми, розширивши її. По-перше, використовувана метрика відстані. Хоча і було прийнято рішення замінити використання стандартного способу з розрахунком евклідової відстані алгоритмом DTW, що допомагає у роботі з часовими рядами, все ж подібний крок помітно ускладнить процес проходження наборів даних, та сповільнить роботу. По-друге, DBSCAN не здатен працювати з великими значеннями даних. В такому випадку його досягає так зване прокляття розмірності. В такому випадку, в якості подолання проблеми пропонується дещо нормалізувати дані шляхом взяття десяткового логарифму від початкового значення. Це дозволить зменшити масштаб нашої інформації та вберегти алгоритм від можливих помилок.

2.4 Огляд алгоритму DTW

Dynamic Time Warping або DTW – алгоритм призначений для порівнювання та вирівнювання часових рядів між собою. Використання

методу виправдовується тим, що як правило необхідні для порівняння ряди мають різну довжину та конфігурацію. Алгоритм в свою чергу вирішує дану проблему шляхом будування оптимального способу вирівнювання, а також прорахунку «вартості» такого вирівнювання [26].

Сам по собі алгоритм неможливо використовувати для кластеризації даних, хоча він і передбачає спосіб порівняння. Тому його активно використовують у зв'язках з іншими алгоритмами кластеризації, для того щоб покращити результати роботи з часовими рядами для останніх.

Суть алгоритму полягає у покроковому порівнянні точок двох часових рядів та будуванні шляху вирівнювання. Наглядно результат роботи алгоритму проілюстровано на рисунку 2.4.

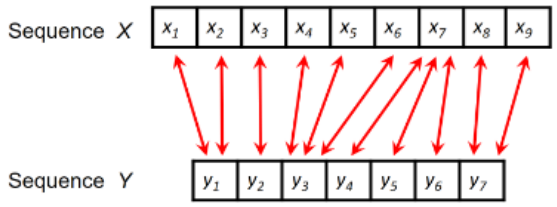


Рисунок 2.4 – Робота алгоритму DTW

Таким чином спостерігаємо, що в результаті роботи алгоритму усі точки співставляються з токами другої, при цьому очевидним є те що одна точка не обов'язково може мати лише одну відповідність у іншому ряду. Таким чином уникається втрата даних коли за різної довжини рядів більш довший обрізався.

Такий результат роботи формується завдяки побудові шляху вирівнювання. Для того щоб, створити такий шлях необхідно дотримуватись наступних кроків:

– створити матрицю розміром $M+1$ x $N+1$, де M довжина першого ряду, а N – другого;

– перший рядок та перший стовбчик матриці ініціалізуємо значенням ∞. А у клітину з індексом (0, 0) ставимо значення 0;

– заповнюємо усі клітини таблиці, що залишились, за наступною формулою.

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} \\ D_{i-1,j} \\ D_{i,j-1} \end{cases};$$

– зворотно обрахувати ціну вирівнювання двох часових рядів. Це означає, що необхідно почати з елементу *D(m+1,n+1),* та в зворотному порядку до цього значення додавати найменше суміжне. Отримане значення буде ціною вирівнювання, а ті клітини, що були використані для прорахунку цього значення, будуть входити до шляху вирівнювання.

За результатами роботи алгоритму, ми отримаємо всю необхідну для подальшої кластеризації інформацію. Приклад отриманої матриці, що відповідає двом рядам представленим вище, наведено на рисунку 2.5.

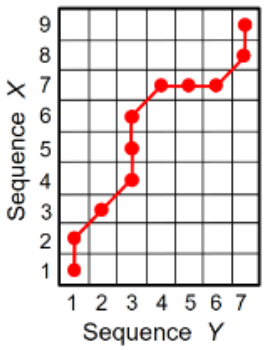


Рисунок 2.5 – Матриця вирівнювання

Як бачимо, в даному прикладі не враховувались додаткові лінія та стовбець для максимальних значень. Даний крок необхідний для правильної програмної реалізація.

Загалом, при побудові шляху приведення, слід дотримуватись трьох правил, що забезпечують, коректність роботи методу [27, 28]:

– умова границі: вимагає щоб перший елемент шляху вирівнювання був *D(0,0),* а останній – *D(M,N);*

– умова монотонності: вимагає, щоб кожен елемент шляху був більший або дорівнював попередньому;

– умова розміру кроку: вимагає, щоб кожен крок у матриці відбувався лише у напрямках (0,1), (1,0), (1,1).

Нажаль, незважаючи на свою простоту та корисність, алгоритм має недоліки. Серед таких присутня чутливість до різного роду шумів та викидів, через які матриця вирівнювання може бути побудована невірно. Також, як було згадано вище, сам по собі алгоритм DTW, не здатен зробити точний висновок щодо подібності рядів. Можливо отримати лише ціну вирівнювання, яка майже повністю буде залежати від величини вимірів у порівнюваних рядах. А за умови значної розбіжності у даних, процес приведення до адекватної оцінки стає майже неможливим.

2.5 Огляд тренованих класифікаторів

Треновані класифікатори представляють собою алгоритми машинного навчання, здатні використовувати набори даних для навчання, та застосовувати акумульовані «знання» для подальшої обробки даних. При проходженні процесу навчання програма бачить приклади даних та помітки приналежності до класів, та на їх основі навчається виділяти важливі ознаки, щоб в подальшому використовувати їх [29, 30]. Такий підхід називається навчанням з учителем.

Перед початком використання алгоритму необхідно провести деякі маніпуляції з даними, щоб програма мала змогу вірно їх обробити. Перш за все, текстові ознаки необхідно перетворити в числові, через те що машина не розуміє мови людей. Далі необхідно нормалізувати дані, для уникнення занадто великої розбіжності, що в свою чергу може призвести до

неправильного навчання алгоритму. На останок треба розділити дані на дві групи: навчальні та тестові. Навчальні дані будуть використовуватись для безпосередньо навчання алгоритму, та виділення важливих ознак у даних. Тестові ж дані будуть використані для перевірки якості тренування, це необхідно для оцінки роботи, та прийняття рішень щодо зміни налаштувань алгоритму, або тотальну зміну алгоритму навчання.

В нашому випадку необхідним алгоритмом машинного навчання був обраний алгоритм градієнтного спуску. Даний метод підходить значною мірою для тих даних, які нараховують велику кількість спостережень. Основною метою алгоритму градієнтного спуску є ітераційне проходження по тренуючих даних та послідовне переналаштування показників функції з метою мінімізації функції вартості [31, 32]. На рисунку 2.6 [33] візуалізовано принцип роботи алгоритму в трьох можливих варіаціях проходження даних.

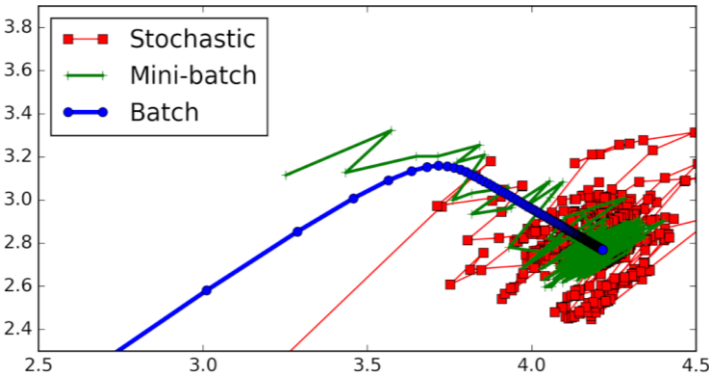


Рисунок 2.6 – Принцип роботи методу градієнтного спуску

На рисунку бачимо три графіки різного роду, що представляють собою три різні підходи використання тренуючих даних. Розглянемо їх детальніше.

2.5.1 Пакетний метод

Пакетний метод обробки даних, який на рисунку 2.6 позначено синім кольором та підписано назвою «Batch», є найочевиднішим підходом до

навчання. Згідно назви алгоритм градієнтного спуску навчається на «партії» даних, які були надані. Тобто на практиці, програма щоразу буде оброблювати повний набір даних тренування, послідовно знаходячи рішення. Такий підхід не є раціональним для використання з величезними масивами даних, через те, що обробка та навчання займатимуть значну кількість часу. Хоча як видно з графіку, шлях мінімізації функції є найстабільнішим з усіх. Таким чином, можна зробити висновок, що для використання з невеликою кількість тренуючих даних, метод має переваги.

2.5.2 Міні-пакетний метод

Міні-пакетний метод – метод обробки даних у алгоритмі градієнтного спуску, що був створений для покращення роботи попередника з великими наборами даних тренування. Як можна судити з назви, алгоритм оброблює малі партії даних при кожній ітерації спуску. Ці партії обираються випадковим чином з повного ряду даних що були додані для навчання. Такий підхід дозволяє скорити час навчання програми, звичайно, в залежності від заданого розміру міні-пакетів. З рисунку 2.6, де метод міні-пакетної обробки даних позначено назвою «Mini-batch», та відмічено зеленим кольором, ми можемо спостерігати деяку хаотичність проміжних результатів роботи градієнтного спуску. Таке явище і пояснюється випадковістю вибору даних для підпакетів. Та незважаючи на це, метод показує доволі непогані результати роботи порівняно з попереднім підходом.

2.5.3 Стохастичний підхід

Стохастичний метод деякою мірою є окремим випадком використання методу міні-пакетів. Виникає тоді коли розмір створюваних підпакетів стає

рівним 1. Тобто, загально описуючи стохастичний метод обробки даних можемо говорити, що на кожній ітерації обирається окреме спостереження з усього набору даних тренування. Такий підхід вимагає великої кількості повторів, для виявлення більш-менш прийнятного рішення, та подальшого використання на тестових даних. Насправді, необхідність на одній ітерації оброблювати лише один відлік настільки скорочує час навчання, що даний підхід можна назвати найшвидшим з усіх трьох, що у випадку використання великих наборів даних стає майже єдиним варіантом застосування.

На рисунку 2.6 метод показаний червоним кольором та має назву «Stochastic», а з його поведінки під час процесу навчання, спостерігаємо значні стрибки у проміжних значеннях функції. Проблема такої поведінки така ж, як і в методі міні-пакетів, але поглиблюється через більшу випадковість даних.

2.6 Методи відмірювання близькості

В даному розділі була показана та описана евклідова відстань, як міра подібності, що використовувалась при кластеризації даних, та алгоритм DTW, що в своїй основі також має реалізацію евклідової відстані. Насправді існує дуже багато методів знаходження відстані або різниці між даними. Всі вони мають різну мету, та призначення. Серед найуживаніших метрик можна зустріти такі: квадрат евклідової відстані, Мангеттенська відстань, відсоток незгоди та відстань Чебишова [34, 35]. Розглянемо їх детальніше.

2.6.1 Квадрат евклідової відстані

Як зрозуміло з назви, дана метрика представляє собою звичайну евклідову відстань, але приведену в квадрат. Тож вигляд даний метод має наступний:

$$d_E(X_k, X_n) = \sum_{i=1}^{m} (x_{ik} - x_{in})^2,$$

де X_k, X_n – точки, вектори, або часові ряди;

m – кількість вимірів або координат.

Даний підхід використовується в тих випадках коли необхідно надати більшої ваги при порівнянні більш віддалених точок.

2.6.2 Мангеттенська відстань

Метрика відстані, що пов’язана з міським плануванням Мангеттена. В літературі зустрічається також під назвами «сіті-блок відстань», «відстань прямокутного міста», «відстань прямих кварталів». Суть алгоритму полягає в тому, що відстань між двома точками буде дорівнювати сумі модулів різниць їх координат. Метод репрезентується наступним чином:

$$d_M(X_k, X_n) = \sum_{i=1}^{m} |x_{ik} - x_{in}|,$$

де X_k, X_n – точки, вектори, або часові ряди;

m – кількість вимірів або координат.

Хоча дана метрика і буде видавати результати, в більшості випадків схожі на результати відстані Евкліда, даний спосіб менше підлягає впливу стрибків та шумів у даних, через те що вони не приводяться у квадрат.

2.6.3 Відсоток незгоди

Дана міра схожості використовується в тих випадках коли порівнювані дані мають категоріальний характер. Обмежена кількість станів кожного виміру точки, вектору, або часового ряду що в своїй основі мають категоріальність, дозволяє швидко та просто вирахувати відсоток відмінності між даними. В загальному вигляді підхід має форму:

$$d_{pb}(X_k, X_n) = \frac{(\sum_{i=1}^{m} |x_{ik} \,!= x_{in}|)}{m},$$

де X_k, X_n – точки, вектори, або часові ряди;

m – кількість вимірів або координат.

2.6.4 Відстань Чебишова

Метрика Чебишова – це метрика що має в своїй основі дещо інший підхід до розуміння поняття «відстань». Так, згідно алгоритму, відстанню між двома векторами називається максимальний модуль різниці його координат. Даний метод задається наступною формулою:

$$l_{\infty}(X_k, X_n) = \max_{i=1,\dots,m} |x_{ik} - x_{in}|,$$

де X_k, X_n – точки, вектори, або часові ряди;

m – кількість вимірів або координат.

3 КОМП’ЮТЕРНА МОДЕЛЬ КЛАСТЕРИЗАЦІЇ ЧАСОВИХ РЯДІВ

3.1 Обґрунтування вибору середовища програмної реалізації

У рамках кваліфікаційної роботи були розроблені алгоритми для необхідної фільтрації, кластеризації та порівняння часових рядів. Для реалізації біло використано середовище розробки Visual Studio 2022. Це зручна та популярна програма для розробки, що дозволяє створювати код з великої кількості готових шаблонів на різних мовах програмування.

Visual Studio містить в собі низку інструментів, що дозволяють розробникам ефективно працювати зі всіма компонентами програми в одному місці. Це забезпечує зручність та ефективність розробки. До таких інструментів відносять:

– вбудований відлагоджувач, що дозволяє розробникам відстежувати та виправляти помилки у програмі;

– інтеграція з іншими інструментами розробки, такими як Git, Jira, Azure та інші;

– редактор коду з підсвічуванням синтаксису, автодоповненням та іншими функціями, що допомагають в написанні коду;

– підтримка систем керування версіями;

– вбудовані інструменти для створення графічного інтерфейсу користувача.

Однією з важливих переваг Visual Studio є підтримка різних мов програмування, серед яких є мова C#, C++, Visual Basic тощо. Це дозволяє розробникам працювати з різними мовами програмування в одному середовищі.

Недоліком Visual Studio може бути досить складний та об'ємний інтерфейс, що може стати перешкодою для початківців. Проте, після того, як розробники зрозуміють основні функції та інструменти Visual Studio, вони

можуть значно підвищити свою продуктивність та швидкість розробки програм. Приклад інтерфейсу платформи Visual Studio показано на рисунку 3.1.

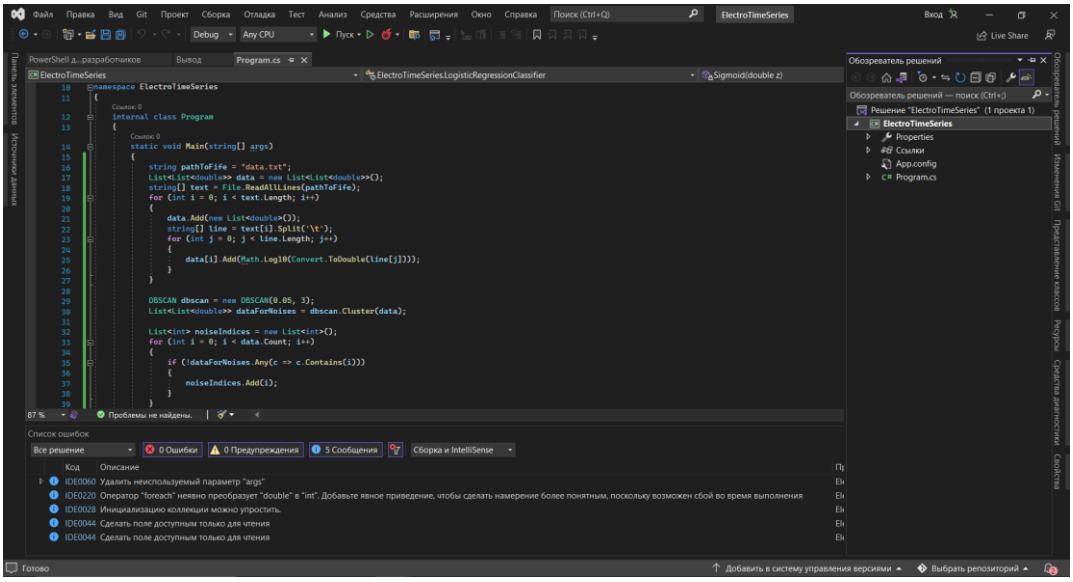


Рисунок 3.1 – Приклад інтерфейсу Visual Studio

C# була створена як частина ініціативи Microsoft .NET Framework з метою створення єдиної платформи для розробки програмного забезпечення. Ця мова програмування є досить популярною в індустрії програмування, оскільки її легко вивчити та використовувати, вона дозволяє розробникам ефективно створювати програми та має багатий функціонал.

Однією з основних особливостей мови C# є її об'єктно-орієнтована природа. Це означає, що мова програмування дозволяє розробляти програми, які складаються з об'єктів, які можуть взаємодіяти один з одним. Крім того, C# підтримує інші парадигми програмування, такі як функційне програмування та паралельне програмування.

При описі мови C# чітко виділяється інша її характеристика – висока продуктивність. Вона може бути виконана як в компіляторі, так і в інтерпретаторі, що дозволяє їй працювати з більшістю операційних систем та платформ. Мова C# використовує сміттєзбірник, що дозволяє автоматично

вивільняти пам’ять від непотрібних об’єктів, що зменшує можливість виникнення помилок у коді.

Попри високу продуктивність мова C# має досить великий рівень використання ресурсів. Цей чинник вносить негативну роль у швидкодію програми створеної за допомогою цієї мови. Але на практиці різниця швидкості роботи у порівнянні з іншими мовами програмування залишається незначною, тому C# досі користується високою популярністю.

C# має багатий функціонал та набір можливостей, через це виникає певна складність у вивченні для початківців. Проте компанія Microsoft подбала про забезпечення користувачів якісною та детальною документацією, щоб полегшити процес інтеграції у використання мови.

3.2 Програмна реалізація

Програмна реалізація методів відбувалась без використання будь-яких додаткових бібліотек. Основною метою даного підходу до роботи була можливість детальніше ознайомитись з тонкощами створення, застосування та поєднання описаних алгоритмів. Окрім ознайомлення це дозволяє більш точно налаштувати кожну деталь. З мінусів – відмова від використання сторонніх бібліотек значно ускладнює розробку коду, що потенційно може призвести до неправильної роботи програми.

3.2.1 Реалізація методу *K*-Means

Даний метод дозволяє кластеризувати великі набори даних по заданій кількості класів. Для розробки застосунку було вирішено розбивати дані на 4 кластери. Дані кластери відображають рівень використання електроенергії: 0 – низький, 1 – середній, 2 – високий, 3 – критичний.

Реалізація цього алгоритму включає в себе створення трьох основних методів. Перша функція – ініціалізація центроїдів. Функція *InitializeCentroids* допомагає почати процес кластеризації для алгоритму *K*-Means, забезпечуючи початкове значення для центроїдів кластерів. Реалізація функції представлена у лістингу 3.1.

Лістинг 3.1 Функція ініціалізації центроїді:

```
private static List<List<double>> InitializeCentroids(List<List<double>> dataPoints,
int numClusters)
        {
            List<List<double>> centroids = new List<List<double>>();
            Random rand = new Random();

            for (int i = 0; i < numClusters; i++)
            {
                int index = rand.Next(dataPoints.Count);
                centroids.Add(dataPoints[index]);
            }

            return centroids;
```

Функція *AssignToClusters* призначена для розподілу даних по клатерах, використовуючи відстань між кожною точкою та центроїдом кластера за допомогою алгоритму DTW. На вхід подаються список даних *dataPoints* та список центроїдів *centroids*. Після ініціалізації додаткових списків та змінних, проходимось по всім точкам, та розраховуємо відстань до всіх центроїдів. Знаходимо мінімальну відстань та додаємо точку до відповідного класу. На виході маємо список кластерів. Приклад реалізації показано в лістингу 3.2.

Лістинг 3.2 Функція розподілення по класам:

```
private static List<List<List<double>>> AssignToClusters(List<List<double>>
dataPoints, List<List<double>> centroids)
        {
            List<List<List<double>>> clusters = new List<List<List<double>>>();

            for (int i = 0; i < centroids.Count; i++)
            {
                clusters.Add(new List<List<double>>());
            }

            foreach (List<double> point in dataPoints)
            {
                int clusterIndex = 0;
                double minDistance = double.MaxValue;

                for (int i = 0; i < centroids.Count; i++)
                {
                    double distance = DTW.Distance(point, centroids[i]);

                    if (distance < minDistance)
                    {
                        clusterIndex = i;
                        minDistance = distance;
                    }
                }

                clusters[clusterIndex].Add(point);
            }

            return clusters;

              }
```

Третя та остання важлива функція для даного алгоритму – це функція перерахунку ценроїдів. Дана функція виконує перерахунок значень центроїдів на основі створених на попередньому кроці кластерів. Функція приймає в себе список створених кластерів *clusters* та повертає список нових центроїдів. Для кожного кластеру обчислюється сума координат або вимірів, і потім виводиться середнє арифметичне. Таким чином отримуємо нову координату або вимір нового кластеру. Приклад реалізації показано у лістингу 3.3.

Лістинг 3.3 Функція перерахунку центроїдів:

```
private static List<List<double>> RecalculateCentroids(List<List<List<double>>>
clusters)
        {
            List<List<double>> centroids = new List<List<double>>();

            foreach (List<List<double>> cluster in clusters)
            {
                List<double> sum = new List<double>();

                foreach (List<double> point in cluster)
                {
                    for (int j = 0; j < point.Count; ++j)
                    {
                        if (sum.Count < point.Count)
                        {
                            sum.Add(point[j]);
                        }
                        else
                        {
                            sum[j] += point[j];
                        }
                    }
                }
                for (int j = 0; j < sum.Count; ++j)
                {
                    sum[j] /= cluster.Count;
                }

                centroids.Add(sum);
            }

            return centroids;

              }
```

3.2.2 Реалізація алгоритму DTW

Dynamic Time Warping – алгоритм що дозволяє всьому проєкту працювати з часовими рядами. Він має просту реалізацію та легку імплементація до інших алгоритмів. Загалом метод має лише дві функції. Перша – реалізація розрахунку евклідової відстані. Дана функція є звичайним повторенням формули та не заслуговує на увагу. Друга функція – це, власне сам алгоритм DTW. Реалізація методу розрахунку показана в лістингу 3.4.

Лістинг 3.4 Метод порівняння часових рядів:

```
public static double Distance(List<double> first_series, List<double> second_series)
        {
            double[,] distance_matrix = new double[first_series.Count + 1,
second_series.Count + 1];
            for (int i = 0; i < first_series.Count + 1; ++i)
            {
                distance_matrix[i, 0] = Int32.MaxValue;
            }
            for (int i = 0; i < second_series.Count + 1; ++i)
            {
                distance_matrix[0, i] = Int32.MaxValue;
            }
            distance_matrix[0, 0] = 0;

            for (int i = 0; i < first_series.Count; ++i)
                for (int j = 0; j < second_series.Count; ++j)
                {
                    distance_matrix[i + 1, j + 1] = Euclidian(first_series[i],
second_series[j]) + Math.Min(distance_matrix[i, j], Math.Min(distance_matrix[i, j +
1], distance_matrix[i + 1, j]));
                }

            int n = first_series.Count, m = second_series.Count, numberOfSteps = 0;
            double distance = 0;
            while (n != 1 || m != 1)
            {
                distance += distance_matrix[n, m];
                double min = Math.Min(distance_matrix[n - 1, m - 1],
Math.Min(distance_matrix[n - 1, m], distance_matrix[n, m - 1]));
                if (min == distance_matrix[n - 1, m - 1])
                {
                    n--;
                    m--;
                }
                else if (min == distance_matrix[n - 1, m])
                {
                    n--;
                }
                else if (min == distance_matrix[n, m - 1])
                {
                    m--;
                }
                numberOfSteps++;
            }

            return distance;
              }
```

Дана функція приймає два часові ряди в якості вхідних даних та обчислює їх відстань. Спочатку ініціалізується матриця розміром $(n+1)\times(m+1)$ де n і m – довжини двох рядів. Далі вся матриця заповнюється евклідовою відстанню між відповідними елементами двох рядів разом із мінімальною відстанню, необхідною для перетворення одного ряду в інший. Далі ітеративно функція вираховує мінімальну відстань необхідну для перетворення рядів. Алгоритм не дозволяє з точністю порівняти дані, але натомість дає представлення про те яка «вартість» того щоб одні дані перетворити на інші. Але, в умовах використання, наприклад з алгоритмом *K*-Means, коли потребується знайти мінімальну відстань, алгоритм блискучо показує себе за роботою.

3.2.3 Реалізація алгоритму DBSCAN

Алгоритм DBSCAN має гарну основу для кластеризації даних, проте в контексті інтеграції у наш проєкт, від нього потребується дещо інший функціонал. Метод, при правильно підібраних параметрах не буде створювати кластери з введених даних, натомість зможе знаходити такі дані які в цілому не вписуються в загальний формат даних. Мова йде про так звані шуми. Знаходження таких даних є безцінним для подальшої аналітики, бо таким чином можливо виявити широкий спектр помилок збору даних.

Реалізація алгоритму займає три функції. Але фундаментальними можна назвати лише дві, через те що третя – лише поєднання попередніх двох для повноцінної роботи. Розглянемо їх детальніше.

Перша функція дозволяє визначити, які точки є «ядрами» кластерів, а які «шумом». Вона перевіряє, які точки знаходяться неподалік від заданої на визначену відстань *eps*, і якщо кількість таких сусідів більше або дорівнює *minPts*, то така точка вважається ядром та додається до кластеру, в іншому випадку точка вважається шумом. У даній реалізації використовується алгоритм DTW для визначення відстані між точками, оскільки він дозволяє працювати з часовими рядами, які можуть мати різну довжину та форму.

Лістинг 3.5 Функція пошуку сусідів для точки:

```
private List<int> GetNeighbors(List<List<double>> data, int pointIndex)
        {
            List<int> neighbors = new List<int>();

            for (int i = 0; i < data.Count; i++)
            {
                if (i == pointIndex) continue;

                double distance = DTW.Distance(data[i], data[pointIndex]);

                if (distance <= eps)
                {
                    neighbors.Add(i);
                }
            }

            return neighbors;
              }
```

Наступна важлива функція для роботи алгоритму – це функція розширення кластерів. Дана дія відбувається ітеративно, та додає усі досяжні

точки до одного класу. Він оновлює мітки кожної точки та розширює кластер доки не залишиться точок для додавання. Ітераційність гарантує, що всі доступні точки будуть додаватись в один клас. Реалізація показана в лістингу 3.6.

Лістинг 3.6 Функція розширення кластерів:

```
private void ExpandCluster(List<List<double>> data, int[] labels, int pointIndex,
List<int> neighbors, int clusterIndex)
        {
            labels[pointIndex] = clusterIndex;

            for (int i = 0; i < neighbors.Count; i++)
            {
                int neighborIndex = neighbors[i];

                if (labels[neighborIndex] == -1)
                {
                    labels[neighborIndex] = clusterIndex;
                }
                else if (labels[neighborIndex] == 0)
                {
                    labels[neighborIndex] = clusterIndex;

                    List<int> newNeighbors = GetNeighbors(data, neighborIndex);

                    if (newNeighbors.Count >= minPts)
                    {
                        neighbors.AddRange(newNeighbors);
                    }
                }
            }
              }
```

3.2.4 Реалізація алгоритму градієнтного спуску

Даний алгоритм є одним з найбільш розповсюджених у сфері створення штучного інтелекту. Досить проста загальна структура даних дозволяє реалізувати просту версію алгоритму за допомогою чотирьох-п’яти функцій. В свою чергу ці функції поділяються на дві основні та дві або три допоміжні функції. Розглянемо кожну окремо.

Перша з основних функцій – це тренування моделі. В даній реалізації функції використовується бінарна класифікація. В загальному плані такий підхід використовується для розподілення даних на два класи. Але в конкретному випадку, цей метод використовується в циклі та визначає чи належить дана точка або вимір до поточного кластеру.

Лістинг 3.7 Основний цикл навчання моделі:

```
for (int c = 0; c < numClasses; c++)
            {
                List<int> binaryLabels = labels.Select(label => label == c ? 1 :
0).ToList();

                for (int epoch = 0; epoch < numEpochs; epoch++)
                {
                    double[] gradients = new double[weights[c].Length];

                    for (int i = 0; i < inputs.Count; i++)
                    {
                        double predicted = Predict(weights[c], inputs[i]);
                        double error = binaryLabels[i] - predicted;

                        for (int j = 0; j < weights[c].Length; j++)
                        {
                            gradients[j] += error * inputs[i][j];
                        }
                    }

                    for (int j = 0; j < weights[c].Length; j++)
                    {
                        weights[c][j] += learningRate * gradients[j] / inputs.Count;
                    }
                }
                  }
```

Друга основна функція – класифікація даних на основі отриманої після навчання інформації. Таким чином, відбувається той самий порядок дій, що і в першій функції, але відсутня частина де відбувається перерахунок коефіцієнтів ваги вимірювань.

Лістинг 3.8 Функція класифікації:

```
public List<double> Classify(List<List<double>> inputs)
        {
            List<double> result = new List<double>();
            foreach (List<double> input in inputs)
            {
                input.Insert(0, 1.0);
                double[] scores = new double[weights.Count];

                for (int c = 0; c < weights.Count; c++)
                {
                    scores[c] = Predict(weights[c], input);
                }
                result.Add(Array.IndexOf(scores, scores.Max()));
            }
            return result;
              }
```

До двох допоміжних функцій алгоритму входять функція передбачення та функція розрахунку сигмоїди. Перша з них використовується як і при тренуванні моделі на даних, так і при класифікації на тестових даних. Мета даної функції припустити до якого класу буде належати конкретна точка. Таке припущення робиться на основі вже обрахованих індексів ваги, а враховуючи, що на кожній ітерації такі індекси перераховуються, то і результат з кожним разом буде точніше. Реалізація показана у лістингу 3.9.

Лістинг 3.9 Функція припущення результату:

```
private double Predict(double[] weights, List<double> input)
        {
            double z = 0;

            for (int j = 0; j < weights.Length; j++)
            {
                z += weights[j] * input[j];
            }


            return Sigmoid(z);
              }
```

У даній функції використовується ще одна. Вона пов’язана з розрахунком сигмоїди. Сигмоїда є одним з ключових елементів роботи алгоритму, бо вона дозволяє вирівнювати сигнали. Загалом існує декілька видів сигмоїд, але в контексті даної роботи ми будемо використовувати формулу логістичної функції. Вона добре вписується в контекст використання бінарної класифікації. Програмна реалізація представлена у лістингу 3.10.

Лістинг 3.10 Функція розрахунку сигмоїди:

```
private double Sigmoid(double z)
        {
            double tmp = Math.Exp(-z);
            return 1.0 / (1.0 + tmp);
              }
```

3.3 Тестування розроблених алгоритмів

Тестування програмного забезпечення є невід’ємною частиною розробки коду. Це необхідно для того аби під час роботи мати можливість калібрувати налаштування. Усі роботи з тестування проводились на готових наборах даних, які були згенеровані випадково.

3.3.1 Тестування алгоритму *K*-Means

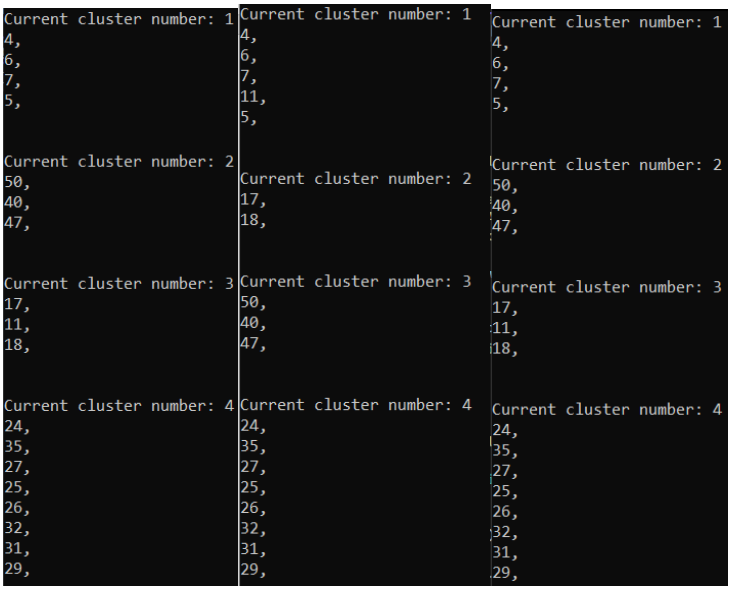
Реалізація даного алгоритму передбачає використання часових рядів, котрі мають змінну розмірність. Отже необхідно впевнитись, що програма що

використовує *K*-Means, буде здатна розбивати на класи різного роду дані. Розпочнемо з масиву чисел. Дані для тестування представлені на рисунку 3.2.

```
50
24
35
4
27
25
26
40
17
32
6
47
7
11
5
31
18
29
```

Рисунок 3.2 – Приклад тестового набору №1

Спочатку проведемо декілька пробних запусків алгоритму для перевірки роботоздатності. Отримані результати представлені на рисунку 3.3.

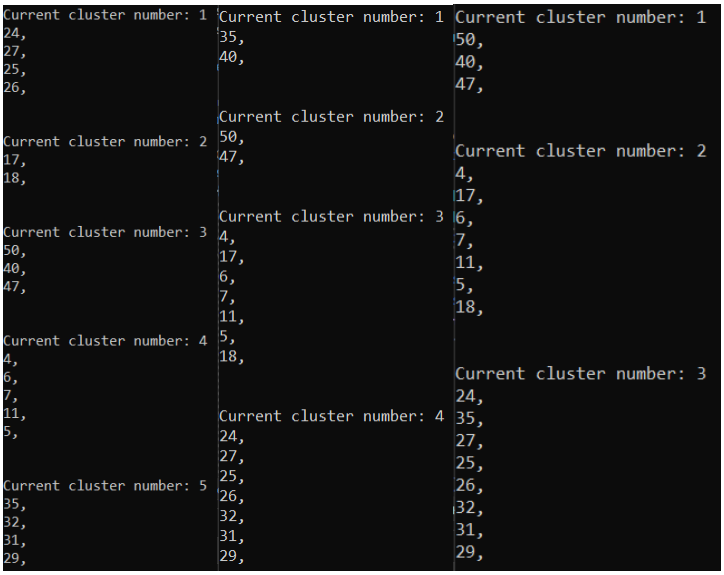


а) б) в)

Рисунок 3.3 – Результат роботи алгоритму на тестових даних:
а) перший запуск алгоритму; б) другий запуск алгоритму; в) третій запуск алгоритму

Як бачимо з результатів роботи алгоритму, принезмінних параметрах алгоритм видає схожі результати. Спостерігаємо непостійність результату

через випадкову ініціалізацію початкових центроїдів. Хоч даних небагато і їх розбіжність досить невелика, та все ж спостерігаємо наявність точки котра «плаває» між кластерами в залежності від ініціалізації. Спробуємо змінити кількість необхідних кластерів. Отриманий результат показано на рисунку 3.4.



а) б) в)

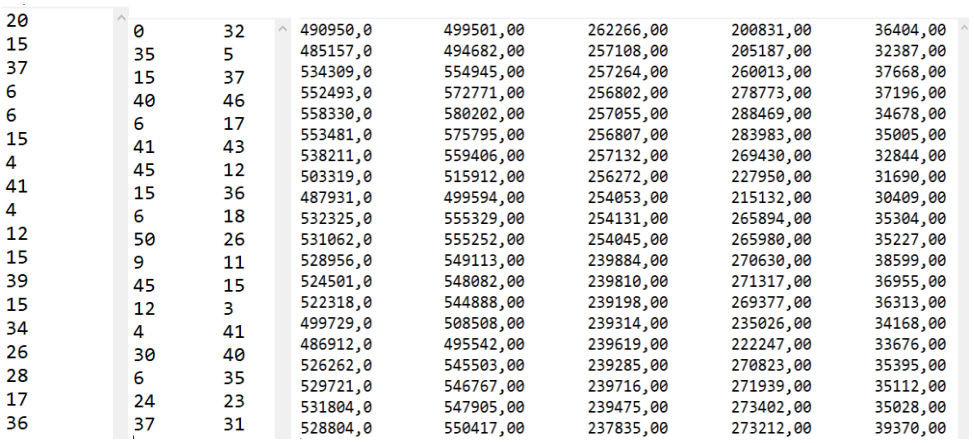
Рисунок 3.4 – Робота алгоритму за умови зміни клькості кластерів
а) кількість кластерів дорівнює 5; б) кількість кластерів дорівнює 4;
в) кількість кластерів дорівнює 3

Настав час попрацювати над зміною формату даних. Для даного тестування використаємо три набори даних: перший – масив чисел, який використовувався у попередніх тестах, другий – набір геометричних координат, третій – набір даних, які будуть використовуватись у фінальній версії проєкту. Усі дні показані на рисунку 3.5, а результати роботи алгоритму представлено на рисунку 3.6.

Дане тестування показало, що алгоритм здатен кластеризувати дані різної форми та наповнення. Результат помітний навіть візуально.

Тож підсумовуючи, дана реалізація алгоритму дозволяє кластеризувати будь-яку кількість даних, різнії розмірності та формату. Є можливість групування даних на різну кількість класів. Результат відрізняється через

випадкову ініціалізацію центроїдів, через що може відрізнятись кінцеве їх значення. Та попри все, даний алгоритм показує чудову роботоздатність, та можливість до інтеграції у більш великий проєкт.

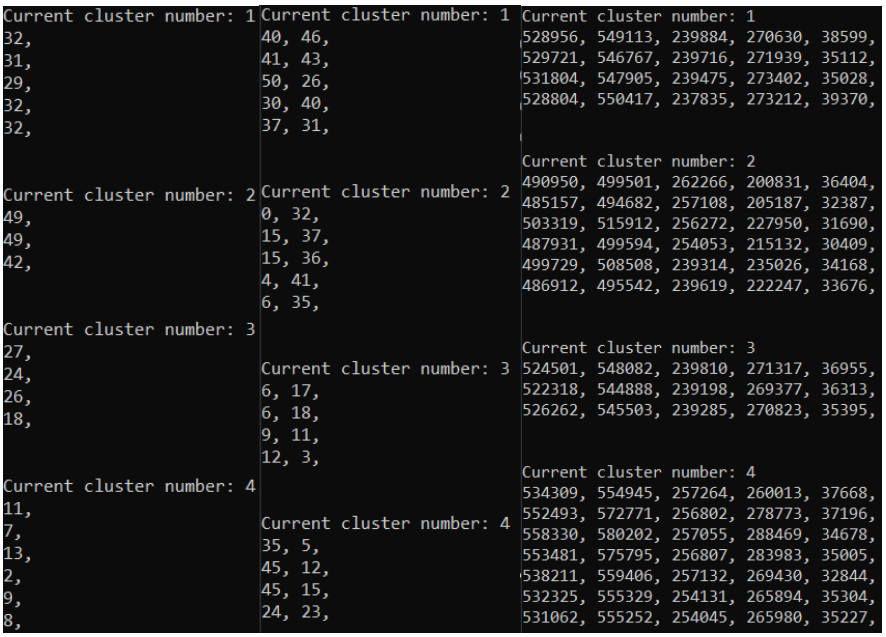


20	0	32	490950,0	499501,00	262266,00	200831,00	36404,00
15	35	5	485157,0	494682,00	257108,00	205187,00	32387,00
37	15	37	534309,0	554945,00	257264,00	260013,00	37668,00
6	40	46	552493,0	572771,00	256802,00	278773,00	37196,00
6	6	17	558330,0	580202,00	257055,00	288469,00	34678,00
15	41	43	553481,0	575795,00	256807,00	283983,00	35005,00
4	45	12	538211,0	559406,00	257132,00	269430,00	32844,00
41	15	36	503319,0	515912,00	256272,00	227950,00	31690,00
4	6	18	487931,0	499594,00	254053,00	215132,00	30409,00
12	50	26	532325,0	555329,00	254131,00	265894,00	35304,00
15	9	11	531062,0	555252,00	254045,00	265980,00	35227,00
39	45	15	528956,0	549113,00	239884,00	270630,00	38599,00
15	12	3	524501,0	548082,00	239810,00	271317,00	36955,00
34	4	41	522318,0	544888,00	239198,00	269377,00	36313,00
26	30	40	499729,0	508508,00	239314,00	235026,00	34168,00
28	6	35	486912,0	495542,00	239619,00	222247,00	33676,00
17	24	23	526262,0	545503,00	239285,00	270823,00	35395,00
36	37	31	529721,0	546767,00	239716,00	271939,00	35112,00
			531804,0	547905,00	239475,00	273402,00	35028,00
			528804,0	550417,00	237835,00	273212,00	39370,00

а) б) в)

Рисунок 3.5 – Набори тестових даних:

а) набір даних, що представляє масив чисел; б) набір даних, що представляє набір координат; в) набір даних, що представляє собою часові ряди



```
Current cluster number: 1
32,
31,
29,
32,
32,

Current cluster number: 2
49,
49,
42,

Current cluster number: 3
27,
24,
26,
18,

Current cluster number: 4
11,
7,
13,
2,
9,
8,
```

```
Current cluster number: 1
40, 46,
41, 43,
50, 26,
30, 40,
37, 31,

Current cluster number: 2
0, 32,
15, 37,
15, 36,
4, 41,
6, 35,

Current cluster number: 3
6, 17,
6, 18,
9, 11,
12, 3,

Current cluster number: 4
35, 5,
45, 12,
45, 15,
24, 23,
```

```
Current cluster number: 1
528956, 549113, 239884, 270630, 38599,
529721, 546767, 239716, 271939, 35112,
531804, 547905, 239475, 273402, 35028,
528804, 550417, 237835, 273212, 39370,

Current cluster number: 2
490950, 499501, 262266, 200831, 36404,
485157, 494682, 257108, 205187, 32387,
503319, 515912, 256272, 227950, 31690,
487931, 499594, 254053, 215132, 30409,
499729, 508508, 239314, 235026, 34168,
486912, 495542, 239619, 222247, 33676,

Current cluster number: 3
524501, 548082, 239810, 271317, 36955,
522318, 544888, 239198, 269377, 36313,
526262, 545503, 239285, 270823, 35395,

Current cluster number: 4
534309, 554945, 257264, 260013, 37668,
552493, 572771, 256802, 278773, 37196,
558330, 580202, 257055, 288469, 34678,
553481, 575795, 256807, 283983, 35005,
538211, 559406, 257132, 269430, 32844,
532325, 555329, 254131, 265894, 35304,
531062, 555252, 254045, 265980, 35227,
```

а) б) в)

Рисунок 3.6 – Результат роботи на даних різного формату:

а) результат роботи алгоритму на масиві чисел; б) результат роботи на масиві координат; в) результат роботи на наборі часових рядів

3.3.2 Тестування алгоритму DTW

Мета даного алгоритму порівняти часові ряди, використовувати його для інших видів даних немає ані потреби ані раціональності. Тож усі тести будемо проводити з варіаціями набору даних для фінальної збірки.

Перш за все перевіримо загальну роботу алгоритму. Дані для тестування показано на рисунку 3.7, а відповідний результат роботи продемонстровано на рисунку 3.8.

472858,0	484649,00	243053,00	203008,00	38588,00	472858,0	484649,00	243053,00	203008,00	38588,00
510812,0	523019,00	249002,00	232247,00	41770,00	510812,0	523019,00	249002,00	232247,00	41770,00
516263,0	527063,00	253804,00	232036,00	41223,00	516263,0	527063,00	253804,00	232036,00	41223,00
513710,0	525104,00	253337,00	228180,00	43587,00	513710,0	525104,00	253337,00	228180,00	43587,00
516905,0	527766,00	252060,00	232585,00	43121,00	516905,0	527766,00	252060,00	232585,00	43121,00
517040,0	528946,00	247772,00	236028,00	45146,00	517040,0	528946,00	247772,00	236028,00	45146,00
488407,0	499367,00	253766,00	202558,00	43043,00	488407,0	499367,00	253766,00	202558,00	43043,00
474909,0	484997,00	241575,00	202879,00	40543,00	474909,0	484997,00	241575,00	202879,00	40543,00
520581,0	532540,00	253401,00	234656,00	44483,00	520581,0	532540,00	253401,00	234656,00	44483,00
525892,0	537793,00	246532,00	246734,00	44527,00	525892,0	537793,00	246532,00	246734,00	44527,00
531746,0	543569,00	249476,00	249083,00	45010,00	531746,0	543569,00	249476,00	249083,00	45010,00
526025,0	537232,00	253125,00	241860,00	42247,00	526025,0	537232,00	253125,00	241860,00	42247,00
524937,0	537225,00	248673,00	246489,00	42063,00	524937,0	537225,00	248673,00	246489,00	42063,00
511976,0	522733,00	253461,00	227852,00	41420,00	511976,0	522733,00	253461,00	227852,00	41420,00
505555,0	517044,00	249931,00	227746,00	39367,00	505555,0	517044,00	249931,00	227746,00	39367,00
543586,0	554062,00	249142,00	262234,00	42686,00	543586,0	554062,00	249142,00	262234,00	42686,00
544330,0	554023,00	265177,00	246636,00	42210,00	544330,0	554023,00	265177,00	246636,00	42210,00
543338,0	554698,00	265100,00	248145,00	41453,00	543338,0	554698,00	265100,00	248145,00	41453,00
536538,0	547530,00	256801,00	248847,00	41882,00	536538,0	547530,00	256801,00	248847,00	41882,00
523114,0	533962,00	261688,00	230695,00	41579,00	523114,0	533962,00	261688,00	230695,00	41579,00
501830,0	509434,00	264944,00	203391,00	41099,00	501830,0	509434,00	264944,00	203391,00	41099,00
487627,0	496392,00	265191,00	193560,00	37641,00	487627,0	496392,00	265191,00	193560,00	37641,00
526341,0	537840,00	255974,00	238358,00	43508,00	526341,0	537840,00	255974,00	238358,00	43508,00
527532,0	538709,00	255706,00	242688,00	40315,00	527532,0	538709,00	255706,00	242688,00	40315,00
523468,0	534679,00	254425,00	239390,00	40864,00	523468,0	534679,00	254425,00	239390,00	40864,00
524599,0	535655,00	256123,00	238154,00	41378,00	524599,0	535655,00	256123,00	238154,00	41378,00
522753,0	533869,00	252904,00	239894,00	41071,00	522753,0	533869,00	252904,00	239894,00	41071,00
495381,0	504104,00	255699,00	219101,00	29304,00	495381,0	504104,00	255699,00	219101,00	29304,00
460760,0	470657,00	256218,00	190584,00	23855,00	460760,0	470657,00	256218,00	190584,00	23855,00
470981,0	477399,00	251412,00	194924,00	31063,00	470981,0	477399,00	251412,00	194924,00	31063,00

Рисунок 3.7 – Приклади часових рядів

```
Alignment cost: 20299274,7446282
```

Рисунок 3.8 – Результат роботи алгоритму

Даний тест підтвердив здатність алгоритму до роботи. Без контексту, результат роботи не має ніякого сенсу. Проте використовуючи даний алгоритм у зв’язках з іншими, значення ціни приведення грає роль відстані між даними, а отже набуває важливості.

Однією з основних проблем порівняння часових рядів є момент коли два ряди мають різну довжину. При використанні порівняння на основі звичайної метрики, відбувається прямий перебір координат часових рядів, і в разі різності довжин, деякі точки залишаться без порівняння, а отже результат

роботи буде не вірний. Алгоритм DTW має виправити ситуацію, бо через його алгоритм, динамічне порівняння можливе. Для тесту використаємо попередні дані, але в одному з часових рядів видалимо один запис. Результат показано на рисунку 3.9.

Рисунок 3.9 – Результат роботи алгоритму з рядами різної довжини

Як бачимо результат став трохи меншим. Це можна пояснити звичайною логікою, якщо порівнювані ряди менші то і ціна їх приведення один до одного менша. Тож алгоритм працює з різними по розміру рядами, і має можливість бути імплементованим до інших алгоритмів для визначення дистанції.

3.3.3 Тестування алгоритму DBSCAN

Алгоритм DBSCAN дозволяє кластеризувати дані, а також виявляти точки що не підлягають групуванню, так звані шуми. Для фінальної збірки даний алгоритм використовується саме для пошуку та відокремлення шумів, але перевірити точність роботи з кластеризації даних є важливою задачею. Це буде означати, що правильний пошук шумів залежить лише від правильності налаштування.

Використання алгоритму потребує ініціалізація двох параметрів. Перший це радіус відстані пошуку сусідів. Другий – мінімальна кількість сусідів. Отже тестування необхідно проводити спираючись на ці параметри.

Набір даних для дослідження продемонстровано на рисунку 3.10, та результат роботи алгоритму на рисунку 3.11.

Як показав тест, алгоритм справді працює та знаходить точки шуму. Даний приклад показав лише один кластер на всі дані. Отже, для демонстрації розбивання на групи необхідно або збільшити набір даних, або змінити

налаштування самого алгоритму. Змінемо набір даних на рисунку 3.12, та проведемо тестування. Результат роботи на рисунку 3.13.

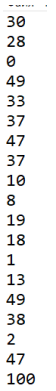


Рисунок 3.10 – Набір тестових даних

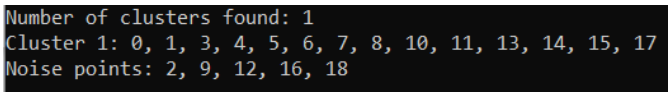


Рисунок 3.11 – Результат роботи алгоритму

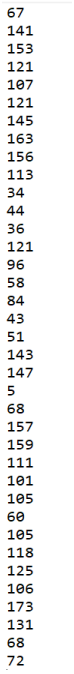


Рисунок 3.12 – Новий набір даних

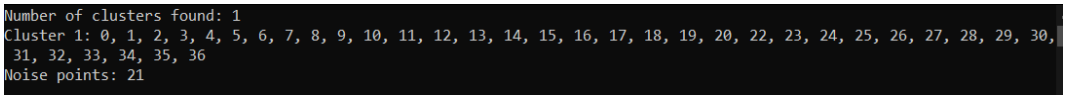
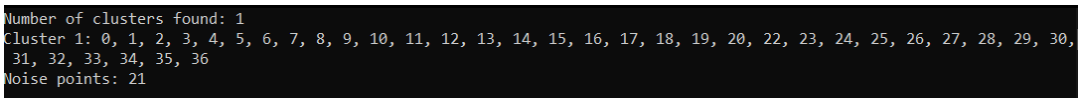


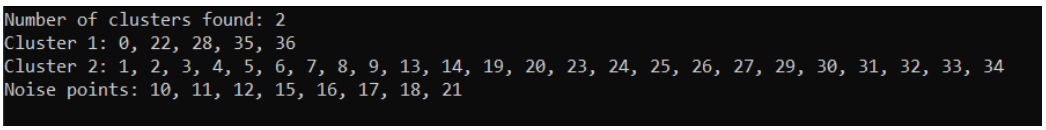
Рисунок 3.13 – Результат роботи з новими даними

Тест показав, що зміна набору даних не вплинула на фактичний результат роботи. На даних налаштуваннях досі маємо один кластер. Отже необхідно переходити до зміни параметрів.

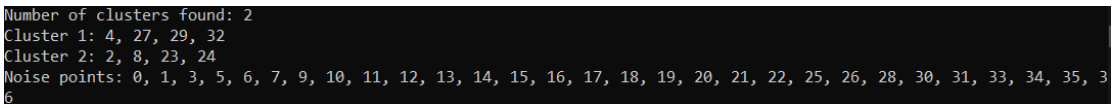
Першим параметром на зміну буде радіус пошуку. При стандартних налаштуваннях даний параметр має значення 0,2. Для тестів поставимо значення 0,1; 0,05; 0,01.Результат показано на рисунку 3.14.



а)



б)



в)

Рисунок 3.14 – Результати роботи алгоритму:
а) значення радіусу пошуку дорівнює 0,1; б) значення радіусу пошуку дорівнює 0,05; в) значення радіусу пошуку дорівнює 0,01

Зміна радіусу дії позитивно проявила себе. Можемо спостерігати таку тенденцію: при зменшенні радіусу пошуку, спочатку збільшується кількість кластерів на які розбиваються дані, а потом все більше точок відносяться до класу шумів.

Проведемо тест зі зміною кількості сусідів. Стандартно даний показник має значення 3. Для даного набору даних змінемо його на 5, 10, 15. Результат роботи алгоритму з даними значеннями показано на рисунку 3.15.

Тест показав, що зміна кількості сусідів незначно вплинула на результат роботи, збільшилась лише кількість виявлених шумів.

```
Number of clusters found: 1
Cluster 1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30,
 31, 32, 33, 34, 35, 36
Noise points: 21
```

а)

```
Number of clusters found: 1
Cluster 1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30,
 31, 32, 33, 34, 35, 36
Noise points: 21
```

б)

```
Number of clusters found: 1
Cluster 1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
 33, 34, 35, 36
Noise points: 10, 12, 21
```

в)

Рисунок 3.15 – Результати роботи з різною кількістю сусідів:
а) кількість сусідів дорівнює 5; б) кількість сусідів дорівнює 10; в) кількість сусідів дорівнює 15

Дані енергорядів мають іншу форму, а отже необхідно перевірити роботоздатність алгоритму у поєднанні з іншими формами даних. Перевіримо роботу на масиві чисел, як у попередніх випадках, масиві координат, фінальних даних. Усі набори продемонстровано на рисунку 3.16, а результати роботи алгоритму на рисунку 3.17.

14	14	105	484644,0	499731,00	242713,00	222980,00	34038,00
130	130	163	498728,0	516081,00	252484,00	226989,00	36608,00
2	2	194	504714,0	520475,00	253598,00	229660,00	37217,00
67	67	162	509114,0	530210,00	254128,00	238661,00	37421,00
69	69	133	512006,0	533194,00	254494,00	241007,00	37693,00
131	131	78	490950,0	499501,00	262266,00	200831,00	36404,00
133	133	58	485157,0	494682,00	257108,00	205187,00	32387,00
92	92	200	534309,0	554945,00	257264,00	260013,00	37668,00
186	186	149	552493,0	572771,00	256802,00	278773,00	37196,00
134	134	120	558330,0	580202,00	257055,00	288469,00	34678,00
59	59	180	553481,0	575795,00	256807,00	283983,00	35005,00
1	1	108	538211,0	559406,00	257132,00	269430,00	32844,00
101	101	118	503319,0	515912,00	256272,00	227950,00	31690,00
132	132	93	487931,0	499594,00	254053,00	215132,00	30409,00
183	183	113	532325,0	555329,00	254131,00	265894,00	35304,00
127	127	50	531062,0	555252,00	254045,00	265980,00	35227,00
166	166	16	528956,0	549113,00	239884,00	270630,00	38599,00
94	94	164	524501,0	548082,00	239810,00	271317,00	36955,00
3	3	140	522318,0	544888,00	239198,00	269377,00	36313,00
90	90	180	499729,0	508508,00	239314,00	235026,00	34168,00
88							

а) б) в)

Рисунок 3.16 – Набори даних:
а) набір даних, що представляє масив чисел; б) набір даних, що представляє набір координат; в) набір даних, що представляє числові ряди

```
Number of clusters found: 1
Cluster 1: 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 19, 20
Noise points: 0, 2, 11, 18
```

а)

```
Number of clusters found: 1
Cluster 1: 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 17, 19
Noise points: 0, 2, 11, 16, 18
```

б)

```
Number of clusters found: 2
Cluster 1: 1, 2, 3, 4
Cluster 2: 7, 8, 9, 10, 11, 14, 15, 16, 17, 18
Noise points: 0, 5, 6, 12, 13, 19
```

в)

Рисунок 3.17 – Результати роботи алгоритму:
а) з набором чисел; б) з набором координат; в) з набором часових рядів

Зміна параметрів показала гнучкість у налаштуванні алгоритму. Алгоритм здатен працювати з потрібним форматом даних. Його робота з кінцевою збіркою буде заключатись саме в пошуку шумів, а отже алгоритм працює добре, та готовий до імплементації.

3.3.4 Тестування алгоритму логістичної регресії

Даний алгоритм потрібен для фінального навчання збірки для подальшої класифікації на основі навчання. Створена реалізація дозволяє налаштовувати деякі параметри.

Першим параметром налаштування буде параметр *learningRate*. Він відповідає за рівень навчання алгоритму. Стандартно має значення 0,05, але для тестування поставимо показники на 0,1; 0,01. Набір даних береться з реальних даних, отже представляє собою готові енергоряди. Результат роботи алгоритму на тренувальних даних (рис. 3.18), тренувальних мітках (рис. 3.19), та тестувальних даних (рис. 3.20) продемонстровано на рисунку 3.21.

499641,00	520141,00	296361,78	223296,53	48269,08
496859,00	543114,00	276462,95	266125,86	52519,12
494904,00	505268,00	234578,25	270318,38	37137,20
482206,00	528215,00	261752,72	266326,00	13627,95
479331,00	502979,00	240217,24	262404,14	35761,81
477974,00	504700,00	223220,23	280865,55	61421,99
497832,00	528761,00	244410,78	283838,90	51131,19
489201,00	520231,00	337960,79	181560,62	70959,51
483989,00	490546,00	292561,63	197444,77	53960,06
492375,00	528337,00	317015,94	210542,29	77876,87
489870,00	500253,00	257645,80	242022,40	58479,58
478243,00	485574,00	233168,26	252158,58	24715,72
486240,00	510862,00	235797,04	274435,07	62989,28
482902,00	514531,00	319387,91	194698,53	44455,48
472274,00	483136,00	278346,24	204221,59	56816,79
485133,00	515278,00	235534,60	279589,84	15355,28
496664,00	541595,00	328187,07	212630,20	77773,04

Рисунок 3.18 – Тренувальні дані

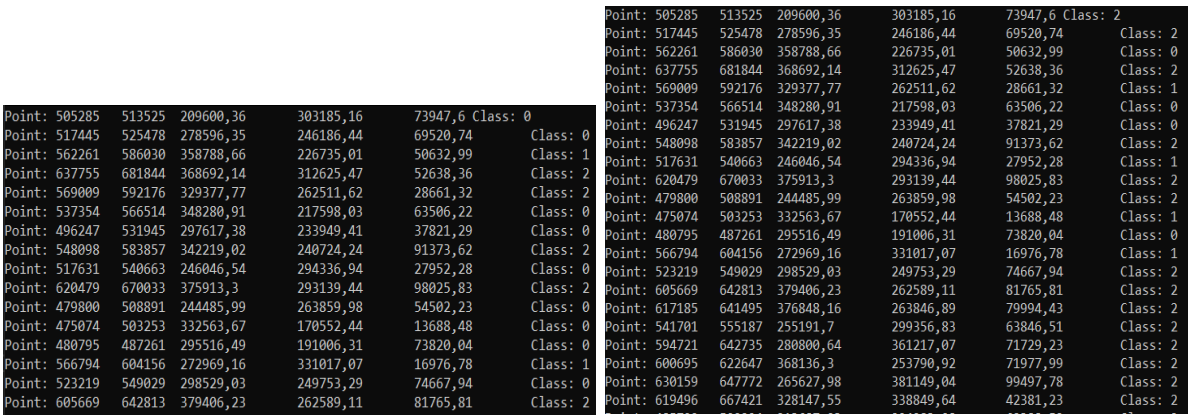
Рисунок 3.19 – Тренувальні мітки класів

505285	513525,00	209600,36	303185,16	73947,60
517445	525478,00	278596,35	246186,44	69520,74
562261	586030,00	358788,66	226735,01	50632,99
637755	681844,00	368692,14	312625,47	52638,36
569009	592176,00	329377,77	262511,62	28661,32
537354	566514,00	348280,91	217598,03	63506,22
496247	531945,00	297617,38	233949,41	37821,29
548098	583857,00	342219,02	240724,24	91373,62
517631	540663,00	246046,54	294336,94	27952,28
620479	670033,00	375913,30	293139,44	98025,83
479800	508891,00	244485,99	263859,98	54502,23
475074	503253,00	332563,67	170552,44	13688,48
480795	487261,00	295516,49	191006,31	73820,04
566794	604156,00	272969,16	331017,07	16976,78
523219	549029,00	298529,03	249753,29	74667,94
605669	642813,00	379406,23	262589,11	81765,81
617185	641495,00	376848,16	263846,89	79994,43
541701	555187,00	255191,70	299356,83	63846,51
594721	642735,00	280800,64	361217,07	71729,23

Рисунок 3.20 – Тестувальні дані

Тест показав, що зміна коефіцієнту навчання впливає на результат навчання, але не зовсім прогнозованим чином. Отже для конкретного набору даних необхідно підбирати свій коефіцієнт.

Наступним показником, якому варто приділяти уваги при тестуванні алгоритму – кількість епох. Епохи – це ітерації навчання алгоритму. Загалом, чим більше алгоритм навчається, тим точніше буде його наступна робота.

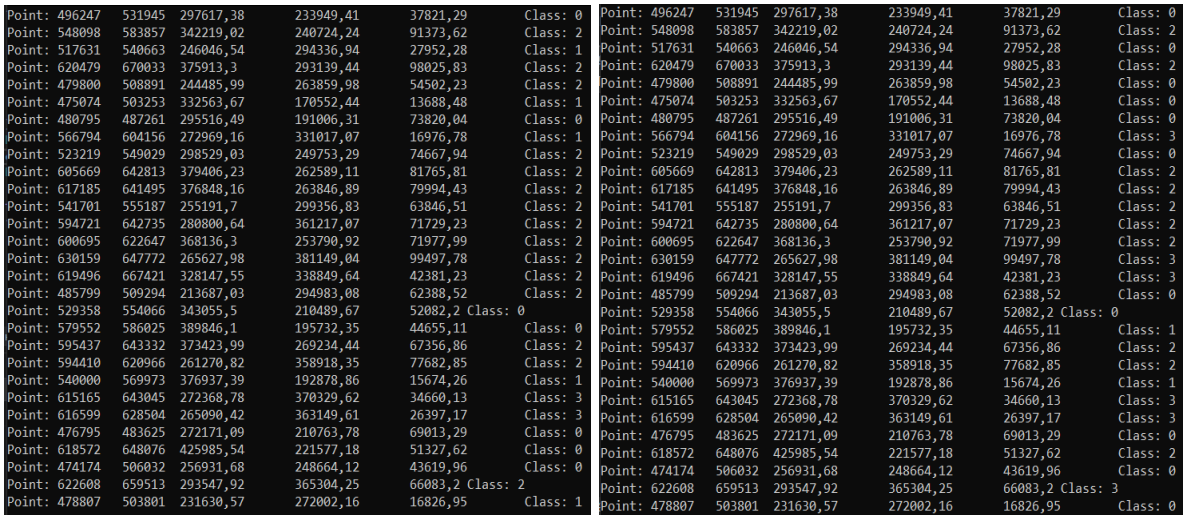


а) б)

Рисунок 3.21 – Результати роботи алгоритму:

а) *learningRate* = 0,1; б) *learningRate* = 0,01

Але необхідно не допускати моменту перенавчання, бо в такому випадку, алгоритм зможе правильно помічати лише ті дані, які були присутні в навчальному наборі, а усі інші матимуть невірний результат. Стандартно алгоритм виконує 100000 епох, але для умов тестування поставимо 50000 та 300000 епох. Результат можемо спостерігати на рисунку 3.22.



а) б)

Рисунок 3.22 – Результати тестування:

а) кількість епох дорівнює 50000; б) кількість епох дорівнює 300000

Як бачимо, зі збільшенням кількості епох якість навчання покращувалась, тож, теоретично, ми можемо і надалі збільшувати кількість ітерацій навчання. Проте необхідно пам’ятати, що збільшуючи кількість навчання, ми збільшуємо час обробки інформації, що погіршує загальну швидкодійність.

3.4 Огляд розробленого застосунку

В ході даної роботи, був розроблений додаток, що має за мету аналізувати дані про енергоспоживання та енерговиробництво України у вигляді часових рядів. Даний застосунок має мобільну форму на платформі Android, що дозволяє з легкістю встановити його на смартфон особи, яка уповноважена аналізувати такі дані, як споживання електроенергії всієї країни. В загальному вигляді структура додатку має вигляд показаний на рисунку 3.23.

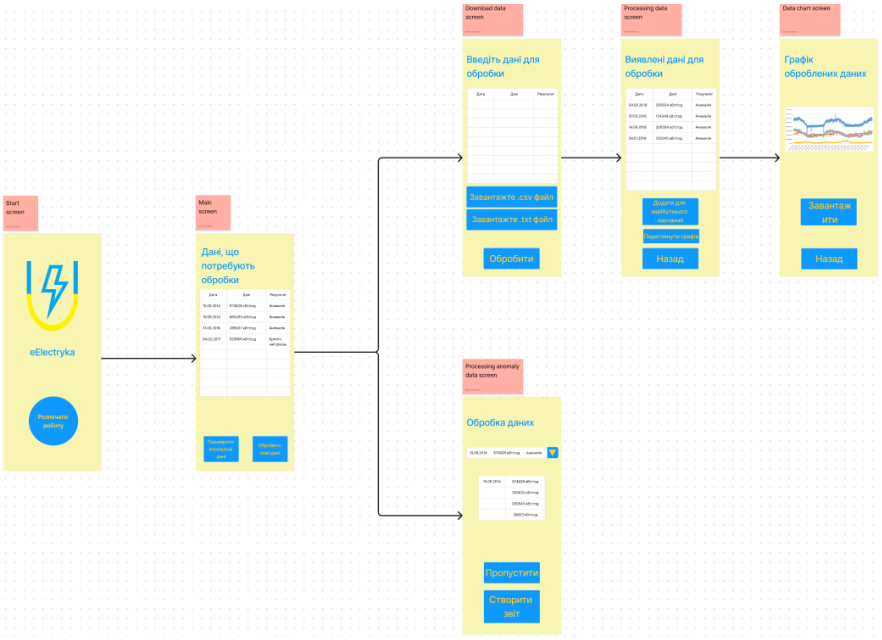


Рисунок 3.23 – Структура застосунку

Розглянемо детальніше кожен пункт. Перш за все початковий екран (рис. 3.24). Відкриваючи застосунок, користувач побачить логотип, назву програми та кнопку що запускає застосунок.

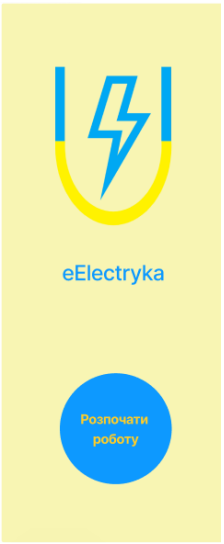


Рисунок 3.24 – Початковий екран

Після натискання кнопки запуску, користувач побачить основне робоче вікно. В ньому він зможе спостерігати всі дані, що були класифіковані як аномальні, і тому потребуються детального розгляду компетентної особи. Також в даному вікні користувач зможе використати дві функції. Перша це обробити аномальні дані, для того щоб вони зникли на основній вкладці. Хоча в теорії, алгоритм здатен автоматично виявляти аномальні дані та формувати звіти щодо них, дана дія потребує ручного контролю з боку користувача. Це обумовлено тим, що перша версія програми не може бути відкалібрована належним чином, код потребує вдосконалення. Приклад помилки можемо спостерігати на рисунку 3.25, де серед виявлених аномальних даних алгоритм також вивів запис з критичною напругою, що в контексті задачі, аномалією не є. Інша функція – обробити нові дані. Вона використовується найчастіше, та саме вона є джерелом живлення алгоритму, бо таким чином можна поповнювати кількість даних для подальшого алгоритму.

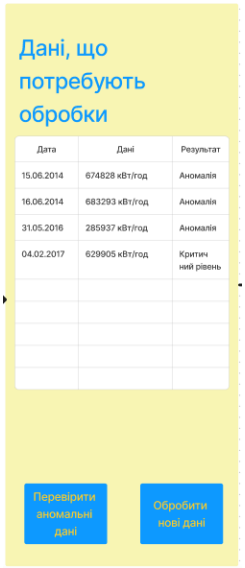


Рисунок 3.25 – Основне вікно програми.

Розглянемо функцію обробки даних детальніше. Натиснувши кнопку «Обробити нові дані» користувач побачить вікно продемонстроване на рисунку 3.26.

Рисунок 3.26 – Обробка аномальних даних

Тут користувач зможе обрати один з виявлених записів та матиме два шляхи для його обробки. Перший шлях – формування звіту. Ця функція створює простий але формального вигляду Word документ, який потів автоматично пересилається до уповноваженої особи на аналітичну обробку.

Інший шлях обробки – функція «Пропустити». Ця функція передбачає виникнення помилки як в попередньому вікні та дозволяє видалити неправильні дані зі списку аномальних. В такому разі вони знову повертаються до пулу на навчання.

Іншою функцією застосунку є обробка нових даних. Натиснувши відповідну кнопку на головному вікні, користувач побачить вікно представлене на рисунку 3.27.

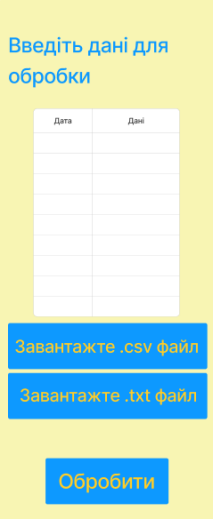


Рисунок 3.27 – Форма заповнення даними

В даному вікні користувач може ввести дані вручну використовуючи надану таблицю. Або в разі, коли користувач заздалегідь підготував файл з зібраними даними, то можна завантажити файл. Далі натискається кнопка «Обробити» і ми отримуємо результат.

Результат представляє собою ті самі дані але з вирахуваним класом як на рисунку 3.28.

В даному випадку маємо три шляхи розвитку. Перший – натиснути кнопку «Назад», в такому випадку ми втрачаємо прогрес класифікації, і у випадку якщо, ця інформація знадобиться знову, треба знову збирати інформацію та повторно запускати алгоритм. Другий шлях – це додавання даних для майбутнього навчання, що згодом підвищить точність вираховування даних.

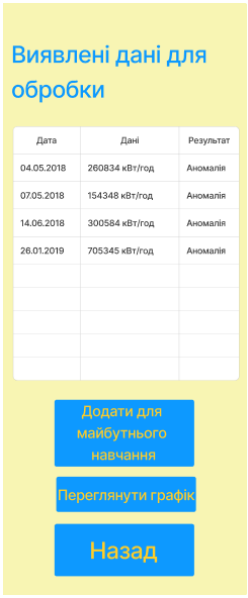


Рисунок 3.28 – Оброблені дані

Третій шлях, це перегляд візуальної інтерпретації даних (рис. 3.29). Така функція необхідна, бо іноді аномальні дані можливо відрізнити лише на око. Також маємо можливість завантажити графік, аби не зупинятись на одному наборі даних та чекати їх повної обробки.

Рисунок 3.29 – Візуалізація даних

ВИСНОВКИ

У рамках кваліфікаційної роботи були розглянуті методи класифікації, кластеризації та групування даних у вигляді часових рядів у формі рядів енергоспоживання, для подальшого аналізу та виявлення помилок збору та обробки даних. Даний процес включає в себе застосування алгоритму машинного навчання, що також був розглянутий та застосований у роботі.

Для практичного застосування було обрано, розроблено та поєднано алгоритми *K*-Means, DBSCAN, DTW, Logistic Regression. Дослідження та експериментальні роботи були проведені у середовищі Visual Studio 2022. Мовою програмування для створення алгоритмів була обрана мова C#.

Специфіка роботи та експериментальні дослідження довели відсутність єдиного, універсального алгоритму вирішення задачі. Тож в основу було покладено використання декількох методів обробки та кластеризації даних. Вибір кожного з представлених методів базується на виявлених перевагах, що підсилюють показники дієвості кінцевої збірки, та недоліках, що перекриваються завдяки іншим алгоритмам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Aghabozorgi, S., Shirkhorshidi, A. S., & Wah, T. Y. (2015). Time-series clustering–a decade review. *Information Systems*, 53, 16-38.

2. Bodyanskiy, Y., Vynokurova, O., Kobylin, I., & Kobylin, O. (2016). Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks. *Information Technology and Management Science*, *19*(1), 23-28.

3. Bodyanskiy Ye., Kobylin I., Rashkevych Yu., Peleshko D., Vynokurova O Hybrid Fuzzy Clustering Algorithm of Time Series with Unevenly and Asynchronously Distributed Observations in Computer Engineering Tasks. 14th IEEE International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering.

4. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., & Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, Advances in Electrical and Electronic Engineering, 21(1), pp. 19-27.

5. Time series patterns | Forecasting: Principles and Practice (2nd ed). URL: https://otexts.com/fpp2/tspatterns.html (дата звернення 28.04.2023).

6. What is a Time Series?. A time series is a sequence of… URL: https://medium.com/analytics-vidhya/what-is-a-time-series-fab8ebc4451b (дата звернення 17.04.2023).

7. Mueen A., Keogh E., Zhu Q., Cash S. & Westover M.B. (2009) Exact discovery of time series motifs. *Proceedings of the SIAM International Conference on Data Mining.*

8. Clustering | Introduction, Different Methods and Applications. URL: https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/ (дата звернення 17.04.2023).

9. Cluster Definition & Meaning - Merriam-Webster. URL: https://www.merriam-webster.com/dictionary/cluster (дата звернення 15.04.2023).

10. Kobylin, O., Vyskrebentseva, S., & Petrova, R. (2019). Обробка даних, що містять пропуски в задачах кластеризації. *Системи управління, навігації та зв'язку. Збірник наукових праць*, *5*(57).

11. Fuzzy Clustering. URL: https://reference.wolfram.com/legacy/applications/fuzzylogic/Manual/12.html (дата звернення 16.04.2023).

12. Frank N. (2016). 8. Hierarchical Clustering. *Introduction to HPC with MPI for Data Science*. Springer. pp. 195–211.

13. Методи кластерного аналізу. Ієрархічні методи. URL: https://moodle.znu.edu.ua/pluginfile.php/486140/mod_resource/content/1/%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%8F%2010.pdf (дата звернення 15.04.2023).

14. Rodrigues, P. P., Gama, J., & Pedroso, J. (2008). Hierarchical clustering of time-series data streams. IEEE transactions on knowledge and data engineering, 20(5), 615-627.

15. You, S. Y., Wang, Y. D., Luo, L. K., & Peng, H. (2016, August). Finding the clusters with potential value in financial time series based on agglomerative hierarchical clustering. In 2016 11th International Conference on Computer Science & Education (ICCSE) (pp. 77-81). IEEE

16. Fuzzy C-Means Clustering (FCM) Algorithm. URL: https://medium.com/geekculture/fuzzy-c-means-clustering-fcm-algorithm-in-machine-learning-c2e51e586fff (дата звернення 14.04.2023).

17. How Does DBSCAN Clustering Work? | DBSCAN Clustering for ML. URL: https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/ (дата звернення 29.04.2023).

18. DBSCAN Clustering — Explained. Detailed theorotical explanation and… URL: https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556 (дата звернення 29.04.2023).

19. An Introduction to Dynamic Time Warping. URL: https://builtin.com/data-science/dynamic-time-warping (дата звернення 16.04.2023)

20. Euclidean Distance - Definition, Formula, Derivation. URL: https://byjus.com/maths/euclidean-distance/ (дата звернення 17.04.2023).

21. Performance Comparison between k-Means and Fuzzy C-Means Algorithms using Arbitrary Data Points. URL: https://www.researchgate.net/publication/233986635_Performance_Comparison_between_k-Means_and_Fuzzy_C-Means_Algorithms_using_Arbitrary_Data_Points (дата звернення 19.04.2023).

22. C-Means Clustering Explained. URL: https://builtin.com/data-science/c-means (дата звернення 20.04.2023).

23. Data Clustering Algorithms - Fuzzy c-means clustering algorithm. URL: https://sites.google.com/site/dataclusteringalgorithms/fuzzy-c-means-clustering-algorithm (дата звернення 22.04.2023).

24. Understanding DBSCAN and Implementation with Python. URL: https://towardsdatascience.com/understanding-dbscan-and-implementation-with-python-5de75a786f9f (дата звернення 29.04.2023).

25. When to use DBSCAN; Crunching the Data. URL: https://crunchingthedata.com/when-to-use-dbscan/ (дата звернення 29.04.2023).

26. Dynamic Time Warping. URL: https://www.audiolabs-erlangen.de/content/05-fau/professor/00-mueller/03-publications/2007_Mueller_DTW-Chapter04-IR_Springer.pdf (дата звернення 22.04.2023).

27. Dynamic Time Warping. Explanation and Code Implementation. URL: https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd (дата звернення 15.04.2023).

28. C3S2_DTWbasic. URL: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C3/C3S2_DTWbasic.html (дата звернення 15.04.2023).

29. Machine learning classifiers – the algorithms & how they work. URL: https://monkeylearn.com/blog/what-is-a-classifier/ (28.04.2023).

30. Training the Classifier (Search Developer's Guide). URL: https://docs.marklogic.com/guide/search-dev/classifier (дата звернення 28.04.2023).

31. МЕТОД ГРАДІЄНТНОГО СПУСКУ ДЛЯ НАВЧАННЯ МОДЕЛІ ЛІНІЙНОЇ РЕГРЕСІЇ. URL: https://stud.com.ua/139980/informatika/metod_gradiyentnogo_spusku_navchannya_modeli_liniynoyi_regresiyi#srcannot_1 (дата звернення 23.04.2023).

32. ОГЛЯД МЕТОДІВ ОПТИМІЗАЦІЇ В МАШИННОМУ НАВЧАННІ: ГРАДІЄНТНИЙ СПУСК ТА СТОХАСТИЧНИЙ ГРАДІЄНТНИЙ СПУСК. URL: https://elartu.tntu.edu.ua/bitstream/lib/34693/2/AZST_2020v2_Martsenyuk_V_P-Review_of_optimization_44-45.pdf (дата звернення 26.04.2023).

33. Gradient Descent, clearly explained in Python. URL: https://towardsdatascience.com/gradient-descent-clearly-explained-in-python-part-2-the-compelling-code-c21ee26fbc28 (дата звернення 27.04.2023).

34. Основні заходи відстаней. URL: https://stud.com.ua/75021/statistika/osnovni_zahodi_vidstaney (дата звернення 27.04.2023).

35. Відстані між об’єктами та кластерами. URL: https://stud.com.ua/63647/marketing/klasterniy_analiz (дата звернення 27.04.2023).