МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
19 травня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія моделювання індикаторів прогресу цифрової трансформації економіки»
здобувача групи ІН.м-11н Коваля Олексія Олександровича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.
_____ Олексій КОВАЛЬ
(підпис)

Керівник,
в.о. завідувача кафедри,
кандидат технічних наук, доцент Наталія БАРЧЕНКО _____
(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»

здобувача групи ІН.м-11н Коваля Олексія Олександровича

1. Тема роботи: «Інформаційна технологія моделювання індикаторів прогресу цифрової трансформації економіки»

затверджую наказом по СумДУ від «08» травня 2023 р. № 0475-VI

2. Термін здачі здобувачем кваліфікаційної роботи до 19 травня 2023 року

3. Вхідні дані до кваліфікаційної роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для моделювання інформаційної технології. 3) Розробка моделей інформаційної технології. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____

(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	03.04-08.04.23	
2	*Огляд технологій, що використовуються для моделювання інформаційної технології*	09.04-14.04.23	
3	*Розробка моделей інформаційної технології*	15.04-30.04.23	
4	*Аналіз отриманих результатів*	01.05-13.05.23	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	14.05-19.05.23	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 59 стор., 19 рис., 9 додатків, 25 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі моделювання індикаторів прогресу цифрової трансформації економіки шляхом розробки відповідних моделей, методів та інформаційної технології.

Об'єкт дослідження – процес моделювання індикаторів прогресу цифрової трансформації економіки.

Мета роботи – розробка моделей для моделювання індикаторів прогресу цифрової трансформації економіки на основі попередніх досліджень.

Методи дослідження – алгоритми обробки даних, інтелектуальні моделі нейронних мереж, нечіткої логіки, регресивні моделі машинного навчання та кластеризація даних.

Результати – розроблено алгоритм обробки даних, інформаційні моделі, які використовують оброблені дані, робить з їх допомогою розрахунки, надає змогу користувачу робити прогнози індексів за новими даними. Проведено тестування розробки на реальних даних.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ПРОГНОЗУВАННЯ, ПРИЙНЯТТЯ РІШЕНЯ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, НЕЧІТКА ЛОГІКА, РЕГРЕСІЯ, PYTON, PANDAS, SKFUZZY, NUMPY, TENSORFLOW, MATPLOTLIB, KERAS, TORCH

ЗМІСТ

ВСТУП

Актуальність. Цифрова трансформація вже давно стала необхідною умовою для ефективного розвитку економіки та конкурентоспроможності країни в сучасних умовах. Зараз цифрові технології охоплюють все більшу кількість галузей і секторів економіки, забезпечуючи їм нові можливості та перспективи розвитку. Але, не дивлячись на значний прорив у цьому напрямку, до цифрової трансформації залишаються дещо складні проблеми, зокрема, необхідність в оцінці рівня та ефективності впровадження цифрових технологій. Таким чином розробка інформаційної технології моделювання індикаторів прогресу цифрової трансформації економіки буде мати позитивний вплив на розвиток технологій наукового прогнозування та економічного росту країн.

Об’єкт дослідження. Процес моделювання індикаторів прогресу цифрової трансформації економіки.

Гіпотеза. Прогнозування індикаторів прогресу цифрової трансформації економіки можна досягнути шляхом застосування інформаційної технології, що реалізує модель машинного навчання.

Наукова новизна. Вперше розроблена інформаційна технологія, яка дозволить спрогнозувати індикатори прогресу цифрової трансформації економіки, що реалізують технології машинного навчання.

Структура. Дана робота складається зі вступу, аналізу публікацій, постановки задачі дослідження, вибір методики та інструментів для рішення поставленої проблеми, опису програмної реалізації інформаційної технології, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОДЕЛЮВАННЯ ІНДИКАТОРІВ ПРОГРЕСУ ЦИФРОВОЇ ТРАНСФОРМАЦІЇ ЕКОНОМІКИ

1.1 Літературний огляд

Інформаційна технологія моделювання індикаторів прогресу цифрової трансформації економіки має велике значення для досягнення цілей сталого розвитку економіки України. Завдяки точному моделюванню та розумінню складнощів сталого розвитку політики можуть приймати більш обґрунтовані рішення та вживати ефективних заходів для досягнення цілей. В рамках даного дослідження будемо використовувати глобальний інноваційний індекс.

Глобальний інноваційний індекс (The Global Innovation Index) [1,2] це світовий рейтинг, що оцінює інноваційні можливості та економічний потенціал країн. Оцінка проводиться на основі різних показників, як-от наукова дослідницька діяльність, здатність до комерціалізації інновацій, ступінь розвитку інформаційних технологій та інше. Ці дані дозволяють проводити порівняльний аналіз між країнами та визначати лідерів у різних сферах інноваційної діяльності. Таким чином, використовуючи дані The Global Innovation Index, ми зможемо розробити інформаційну технологію моделювання індикаторів прогресу цифрової трансформації країни та впроваджувати її в практику. Це дозволить експертам та державним органам приймати обґрунтовані рішення щодо підтримки інноваційного розвитку та сприятиме відновленню економіки. Окрім того, це може стати стимулом для розвитку національної інноваційної технології та залучення інвестицій у цю сферу.

Глобальний інноваційний індекс використовує 80 показників, щоб оцінити інноваційність країни в таких областях, як інтелектуальна власність, науково-технічний потенціал, рівень освіти, ефективність бізнесу та інші. Крім того, індекс враховує рейтинги або показники, які вказують на економічний розвиток.

Дані мають ієрархічну структуру, верхній рівень це – Global Innovation Index, а до нього належать Innovation Input Sub-Index (IISI) та Innovation Output Sub-Index

(IOSI) які в свою чергу мають також ієрархічну структуру, діаграма якої знаходиться у додатку Д. На найнижчому рівні 54 елемента на другому рівні 15 на першому 5 та 1 на найвищому рівні, IOSI має аналогічну структуру до IISI, але іншу кількість елементів відповідно: 27, 6, 2 та 1.

Глобальний інноваційний індекс дозволяє порівнювати країни у світі за їхнім рівнем інноваційності, що може бути корисним для урядів, дослідницьких установ, бізнесу та інших зацікавлених сторін.

Формульний опис структури Глобального інноваційного індексу має наступний вигляд:

Global Innovation Index = k1× (Innovation Input Sub-Index) + k2 × (Innovation Output Sub-Index)

Де:

- Innovation Input Sub-Index - це підіндекс, який оцінює елементи, що визначають потенціал країни для інновацій, такі як інституція середовища, людський капітал та дослідницькі можливості.
- Innovation Output Sub-Index - це підіндекс, який оцінює результати інноваційної діяльності країни, такі як результати досліджень та розробок, використання нових технологій та інновацій в бізнесі.

Коефіцієнти k1 та k2 вказують на відносну вагомість внеску кожного з підіндексів в загальний індекс.

Формульний опис структури підіндексу Innovation Input Sub-Index (IISI) має наступний вигляд:

IISI = k1 × (Institutions) + k2 × (Human Capital and Research) + k3 × (Infrastructure) + k4 × (Market Sophistication) + k5 × (Business Sophistication)

Де:

- Institutions - це показник, який оцінює ефективність правової системи, захист інтелектуальної власності, корупцію та інші інституційні фактори, що підтримують інноваційну діяльність.
- Human Capital and Research - це показник, який оцінює якість освіти та науково-дослідний потенціал країни, включаючи кількість випускників з вищою освітою, кількість наукових публікацій та інші показники.
- Infrastructure - це показник, який оцінює якість технічної інфраструктури країни, включаючи доступність технологій, електронні комунікації та інші показники.
- Market Sophistication - це показник, який оцінює рівень розвитку ринку країни, включаючи доступність фінансових ресурсів, ринок праці та інші показники.
- Business Sophistication - це показник, який оцінює якість підприємницького середовища країни, включаючи культуру підприємництва, рівень інновацій в бізнесі та інші показники.

Формульний опис структури підіндексу Innovation Output Sub-Index (IOSI) має наступний вигляд:

IOSI = k1 × (Knowledge and technology outputs) + k2 × (Creative outputs)

Де:

- Knowledge and technology outputs - це показник, що оцінює якість наукових досліджень та їхній вплив на економіку, включаючи кількість публікацій, цитувань та патентів на мільйон жителів.
- Creative outputs - це показник, що оцінює кількість нових продуктів, послуг та технологій, що були випущені на ринок в країні.

Однак, однією з головних проблем розробка такої інформаційної технології, зокрема із-за складності роботи з вхідними даними. Якість і кількість вхідних даних

є критично важливими для точного моделювання, але в багатьох випадках дані можуть бути обмеженими або недоступними. Це є значною перешкодою для розробки надійної та стійкої інформаційної технології моделювання індикаторів прогресу цифрової трансформації економіки.

1.2 Визначення проблеми дослідження

Хоча створення інформаційної технології для моделювання індикаторів сталого розвитку може запропонувати багато потенційних переваг, існує також кілька викликів та обмежень, які, можливо, доведеться вирішувати при застосуванні цього підходу в Україні.

Одна з головних проблем пов'язана з доступністю та якістю даних. Інформаційна технологія вимагає великої кількості даних для точного моделювання складних систем, і ці дані повинні бути надійними та актуальними. В Україні можуть існувати прогалини в даних, доступних для індикаторів сталого розвитку, а також занепокоєння щодо якості та надійності даних. Це може обмежити точність і надійність будь-якої інформації нечіткого виведення, розробленої для моделювання сталого розвитку в Україні.

Інший виклик пов'язаний зі складністю індикаторів сталого розвитку та необхідністю врахування взаємозалежностей і зворотніх зв'язків між різними змінними. Це вимагає глибокого розуміння місцевого контексту, а також здатності інтегрувати якісні та кількісні дані. В Україні може виникнути потреба у розбудові спроможності моделювання сталого розвитку та розвитку партнерства між технічними експертами та зацікавленими сторонами, які володіють місцевими знаннями та досвідом.

Крім того, створення інформаційної технології може потребувати додаткових ресурсів і технічної експертизи порівняно з традиційними підходами до моделювання. Це може стати бар'єром для впровадження, особливо в умовах, коли ресурси для сталого розвитку вже обмежені. Також може виникнути потреба у підвищенні

обізнаності та розумінні переваг інформаційної технології та моделювання сталого розвитку серед політиків та зацікавлених сторін в Україні.

Наостанок, можуть існувати політичні та інституційні бар'єри на шляху впровадження моделювання індикаторів прогресу цифрової трансформації економіки Україні. Це може включати відсутність політичної волі або підтримки ініціатив сталого розвитку, а також опір новим підходам і технологіям.

Незважаючи на ці виклики, існують також можливості для створення інформаційної технології моделювання індикаторів прогресу цифрової трансформації економіки Україні. Вирішуючи питання якості та доступності даних, розбудовуючи потенціал та партнерства, а також сприяючи підвищенню обізнаності та розумінню переваг моделювання індикаторів прогресу цифрової трансформації економіки, можливо, вдасться розробити більш точні та всеохоплюючі моделі результатів моделювання індикаторів прогресу цифрової трансформації економіки України. Це може сприяти поточним зусиллям, спрямованим на просування сталого розвитку та побудову кращого майбутнього для всіх українців.

1.3 Постановка завдання дослідження

Для розробки інформаційної технології моделювання індикаторів прогресу цифрової трансформації економіки треба:

Ідентифікувати та визначити конкретну проблемну область і змінні. Це вимагає глибокого розуміння основних концепцій і взаємозалежностей у системі сталого розвитку, що моделюється.

Вибрати дані для аналізу та провести їх попередню обробку.

Провести кластерний аналіз для розбиття заданої вибірки об’єктів на кластери, які визначають різні рівні значень індикаторів прогресу цифрової трансформації економіки.

Провести аналітичний огляд математичних апаратів, які можуть бути застосовані для вирішення поставленої проблеми.

Зробити висновки щодо доцільності використання розглянутих математичних апаратів для моделювання індикаторів прогресу цифрової трансформації економіки.

2 ВИБІР МЕТОДІВ ДОСЛІДЖЕННЯ МОДЕЛЮВАННЯ ІНДИКАТОРІВ ПРОГРЕСУ ЦИФРОВОЇ ТРАНСФОРМАЦІЇ ЕКОНОМІКИ

2.1 Кластеризація даних

Кластеризації даних [3-5] - це процес розбиття масиву даних на групи або кластери таким чином, що дані в одному кластері подібні між собою, а дані в різних кластерах відрізняються один від одного. Кластеризація даних використовується в багатьох галузях, як-от машинне навчання, обробка сигналів, біоінформатика та інші.

Основні етапи методології кластеризації даних:

1. Визначення цілей і задач кластеризації - перед початком роботи необхідно з'ясувати, для чого потрібна кластеризація даних і які результати очікувані.
2. Підготовка даних - необхідно визначити набір даних, з якими працюємо, і провести їх попередню обробку, таку як нормалізація, стандартизація, видалення випадкових аномалій і т. д.
3. Вибір методу кластеризації - існує багато методів кластеризації, такі як кластеризація k-середніх, ієрархічна кластеризація, спектральна кластеризація та інші. Вибір методу залежить від специфіки даних та цілей кластеризації.
4. Визначення кількості кластерів - необхідно визначити, скільки кластерів буде сформовано. Це може бути визначено експертно або з використанням алгоритмів, як-от "ліктьовий метод".
5. Застосування алгоритму кластеризації - застосування вибраного методу кластеризації до даних.
6. Оцінка результатів - оцінка результатів кластеризації, зокрема, аналіз отриманих кластерів, визначення оцінки якості кластеризації та вибір метрик для оцінки, такі як індекс силуета, внутрішня і міжкластерна варіація, індекс Данна та інші.

7. Використання результатів кластеризації - після кластеризації можна застосовувати отримані результати для вирішення різноманітних задач, як-от прогнозування, класифікація, аналіз та візуалізація даних.

Важливо зазначити, що методологія кластеризації даних є досить складним процесом і залежить від багатьох факторів, такі як тип даних, розмір даних, рівень шуму в даних, кількість кластерів та інші. Тому, перед застосуванням кластеризації до даних, необхідно провести детальний аналіз та вибрати найбільш оптимальний метод та параметри для конкретного випадку.

2.2 Регресивні моделі машинного навчання

Регресивні моделі машинного навчання [6-9] використовуються для прогнозування числових значень, як-от ціни на акції, кількість продажів, час до відновлення експлуатації після збою і т. д. Ці моделі зазвичай використовуються для розв'язання задач прогнозування в області економіки, фінансів, маркетингу, логістики та інших.

Одні з найбільш поширених регресивних моделей машинного навчання включають:

1. Лінійна регресія: модель, яка шукає лінійну залежність між залежною змінною і незалежними змінними. Лінійна регресія може бути використана для прогнозування значень залежної змінної на основі значень незалежних змінних.
2. Регресія дерев рішень: модель, яка використовує дерево рішень для прогнозування значень залежної змінної на основі значень незалежних змінних. Кожний вузол дерева рішень представляє рішення про те, який шлях слід обрати для знаходження правильної відповіді.
3. Випадковий ліс: модель, яка використовує багато дерев рішень для прогнозування значень залежної змінної на основі значень незалежних змінних. Кожне дерево рішень випадкового лісу навчається на вибірці даних і дає свій внесок у прогнозування.

4. Нейронні мережі: модель, яка складається з багатьох з'єднаних між собою шарів нейронів. Нейронні мережі можуть використовуватися для прогнозування значень залежної змінної на основі значень незалежних змінних.

Кожна з цих моделей має свої переваги та недоліки, і кращий вибір залежить від конкретної задачі та властивостей даних.

Наприклад, лінійна регресія може бути корисна, коли ви досліджуєте лінійну залежність між залежною змінною та незалежними змінними. Регресія дерев рішень може бути корисною, коли є багато незалежних змінних і необхідно відібрати ті, які найбільше впливають на залежну змінну. Випадковий ліс може бути корисним, коли потрібно обробляти велику кількість даних, оскільки він може бути дуже ефективним у роботі з великими вибірками даних. Нейронні мережі можуть бути корисними, коли є складна залежність між залежною змінною та незалежними змінними, але вони можуть бути складними у використанні та налаштуванні.

Важливим етапом у використанні регресивних моделей машинного навчання є підготовка даних, така як вибір та обробка незалежних змінних, розбиття даних на навчальний та тестовий набори, а також валідація моделі. Ці етапи допоможуть забезпечити якість та точність прогнозування моделі.

У загальному, регресивні моделі машинного навчання є потужним інструментом для прогнозування числових значень і їх використання може допомогти вирішити багато задач в різних областях діяльності.

2.3 Нечітка логіка

Апарат нечіткої логіки (Fuzzy Logic) [10,11] - це метод математичного аналізу та розуміння нечіткої (невизначеної) інформації, що дозволяє прогнозувати результати в умовах нечіткості та невизначеності даних.

У методі нечіткого логічного виведення [12-14] використовуються нечіткі множини, які дозволяють робити висновки, які базуються на нечітких умовах. Наприклад, у звичайній булевій логіці значення змінної може бути тільки істинним

або хибним. А в нечіткій логіці, значення змінної може бути частково істинним і частково хибним.

Одним із застосувань методу нечіткого логічного виведення є прогнозування. Наприклад, якщо ми хочемо прогнозувати попит на продукти в магазині, ми можемо використовувати нечіткі змінні, такі як "високий попит", "середній попит" та "низький попит". Потім ми можемо використовувати правила нечіткої логіки, щоб зробити висновок про те, який рівень попиту буде на певний продукт.

Алгоритм нечіткого логічного виведення може бути представлений у вигляді послідовності кроків, що включають в себе:

1. Визначення вхідних змінних та їх значень. Це можуть бути параметри, які необхідно прогнозувати, або відомі параметри, на основі яких потрібно зробити прогноз.
2. Визначення нечітких змінних та їх значень. Нечіткі змінні відображають нечітку природу даних та можуть бути виражені за допомогою нечітких множин.
3. Визначення правил нечіткої логіки. Правила нечіткої логіки визначають, які дії потрібно виконати, якщо вхідні параметри мають певні значення. Наприклад, правило може виглядати так: "Якщо температура повітря висока і вологість повітря висока, то ймовірність дощу висока".
4. Обчислення ступенів приналежності для кожної нечіткої змінної. Ступінь приналежності відображає, наскільки вхідні дані відповідають певному значенню нечіткої змінної. Цей крок можна виконати за допомогою функцій приналежності, які описують форму та величину нечітких множин.
5. Обчислення вагових коефіцієнтів для кожного правила. Ваговий коефіцієнт відображає важливість правила в контексті задачі прогнозування.
6. Обчислення відповідності для кожного правила. Відповідність відображає, наскільки правило відповідає вхідним параметрам. Цей крок можна виконати, використовуючи ступені приналежності та вагові коефіцієнти.

7. Обчислення вихідного значення для кожної нечіткої змінної. Вихідне значення відображає прогнозоване значення параметра. Цей крок можна виконати, використовуючи відповідності та ступені приналежності.
8. Агрегування вихідних значень для отримання остаточного прогнозу. Агрегування може бути виконане за допомогою різних методів, як-от середнє значення, максимум або ваговане середнє значення.

Цей алгоритм може бути модифікований для вирішення різних задач прогнозування, включаючи прогнозування цін, продажів, кліків та інших параметрів. Він може бути застосований в багатьох галузях, включаючи економіку, маркетинг, фінанси та інженерію.

Будова моделі нечіткого логічного виведення (рис.2.1) може бути представлена у вигляді системи, що складається з трьох основних блоків: блоку вхідних даних, блоку правил та блоку вихідних даних.

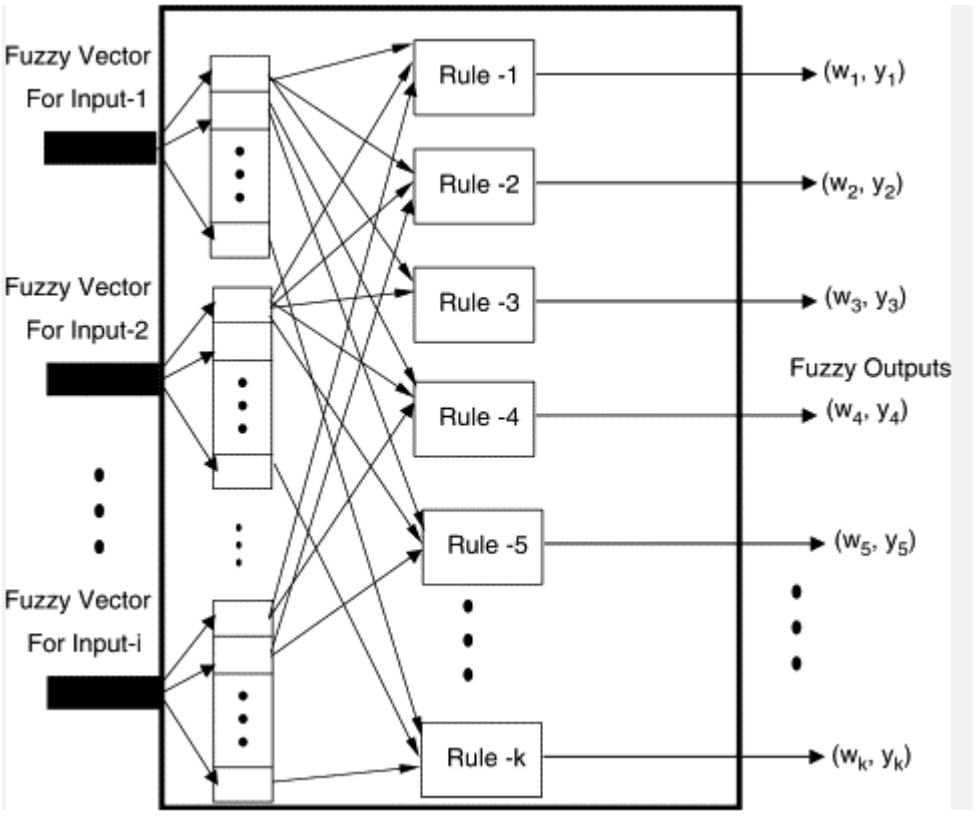

Рисунок 2.1 – Приклад будови моделі нечіткої логіки

Блок вхідних даних включає вхідні параметри, які визначаються задачею прогнозування. Ці параметри можуть бути числовими значеннями або категоріями, які відображають стани об'єктів.

Блок правил включає в себе базу правил, яка визначає, які дії потрібно виконати, якщо вхідні параметри мають певні значення. База правил може бути представлена у вигляді матриці, де стовпці відповідають вхідним параметрам, а рядки - правилам. Кожне правило містить умову та висновок.

Блок вихідних даних включає в себе вихідні параметри, які визначають прогнозовані значення. Ці параметри можуть бути числовими значеннями або категоріями.

У моделі нечіткого логічного виведення кожен блок може мати декілька підблоків, що дозволяє покращити точність прогнозування та знизити кількість помилок. Наприклад, в блоку вхідних даних можуть бути різні підблоки, що відображають різні типи даних (текстові, числові, категоріальні тощо). В блоку правил можуть бути використані різні методи виведення, такі як агрегація, дедукція або апроксимація.

Зазвичай модель нечіткого логічного виведення має також блок управління, який дозволяє визначити режим роботи моделі, включаючи в себе налаштування параметрів, оптимізацію та відлагодження.

Основною перевагою моделі нечіткого логічного виведення є те, що вона дозволяє працювати з нечіткими та невизначеними даними. Наприклад, якщо для прогнозування використовуються показники, які можуть мати різний рівень точності, то ці показники можуть бути виражені у вигляді нечітких чисел, що дозволяє додатково врахувати невизначеність даних та знизити ризик зробити помилку.

Окрім того, модель нечіткого логічного виведення дозволяє працювати з великою кількістю вхідних параметрів та використовувати різні методи виведення, що дозволяє отримати більш точні результати прогнозування.

Недоліком моделі може бути складність її розробки та налаштування, а також необхідність мати достатню кількість даних для побудови моделі та перевірки її ефективності.

2.4 Нейронні мережі

Нейронні мережі [15-18] - це клас алгоритмів машинного навчання, які моделюють роботу людського мозку. Вони складаються зі з'єднаних між собою вузлів, які називають нейронами і зв'язків між ними. Нейронні мережі використовуються для розв'язання різноманітних завдань, як-от класифікація зображень, розпізнавання мовлення, прогнозування та багато інших.

Нейрони в мережі отримують вхідні сигнали, які обробляються та передаються на виходи нейронів. Зв'язки між нейронами мають ваги, які визначають важливість кожного зв'язку. Під час навчання мережі ваги змінюються таким чином, щоб зменшити помилки прогнозування на тренувальному наборі даних.

Є декілька типів нейронних мереж, включаючи перцептрон, звичайну нейронну мережу (feedforward neural network [19]), рекурентну нейронну мережу (recurrent neural network [20]), згорткову нейронну мережу (convolutional neural network [21]) та глибоку нейронну мережу (deep neural network [22]).

Перцептрон [23] (рис. 2.2) - це проста нейронна мережа з одним шаром, яка складається з входу, виходу та блоку обчислень, який приймає вхідні сигнали та віддає вихідні сигнали. Звичайна нейронна мережа складається з кількох шарів, включаючи вхідний шар, приховані шари та вихідний шар. Рекурентна нейронна мережа має зв'язки, які дозволяють передавати інформацію між часовими кроками, що дозволяє їй працювати з послідовностями даних, такими як мовлення або часові ряди. Згорткова нейронна мережа використовується для обробки зображень та інших форм вхідних даних, які мають геометричну структуру. Вона використовує згорткові шари, які визначають локальні функції для кожної області вхідного зображення, та пулінгові шари (шари підвибірки), які зменшують розмір оброблюваного

зображення та підсумовують результуючі значення. Глибока нейронна мережа - це мережа з багатьма шарами, яка може вирішувати більш складні завдання, такі як розпізнавання мови або обробка відео.

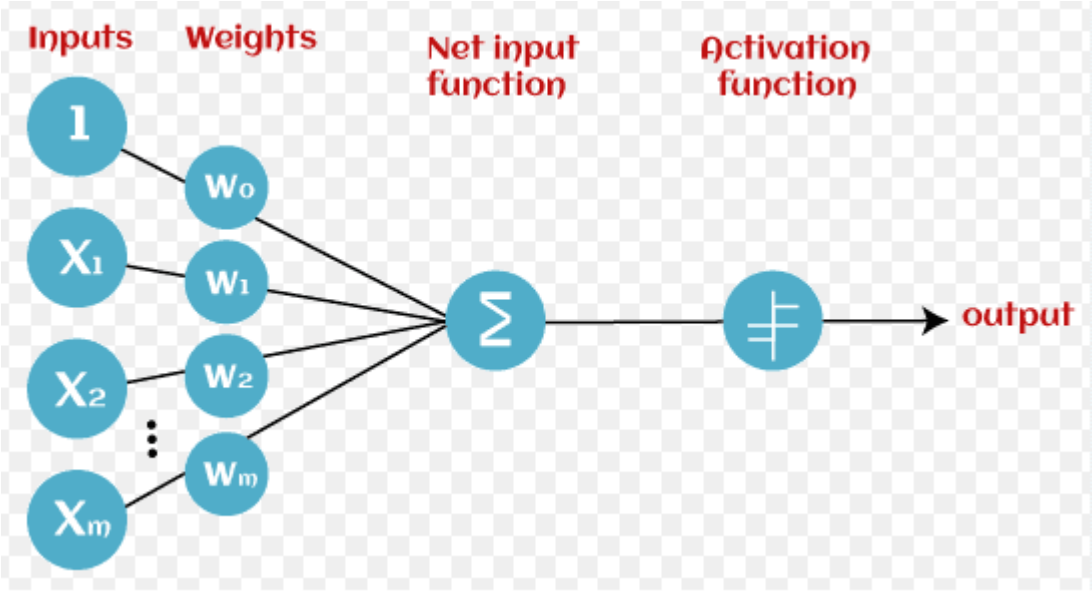

Рисунок 2.2 – Схема перцептрона

Нейронні мережі можуть бути навчені за допомогою методів навчання з вчителем, такі як зворотне поширення помилок (backpropagation), або без вчителя, як-от навчання з підкріпленням (reinforcement learning) або генетичні алгоритми (genetic algorithms). Під час навчання мережі з вчителем мережа отримує вхідні дані та відповідні вихідні дані і змінює свої ваги таким чином, щоб зменшити помилки прогнозування на тренувальному наборі даних. Під час навчання без вчителя мережа вивчає складну структуру вхідних даних, щоб знайти корисні патерни або представлення.

Нейронні мережі зараз широко використовуються в багатьох галузях, такі як комп'ютерне зору, обробка мови, рекомендації, автопілоти в автомобілях та багато інших. Вони дозволяють отримати більш точні прогнози та вирішувати більш складні завдання, що не можуть бути вирішені за допомогою класичних алгоритмів.

Прогнозування за допомогою нейронних мереж - це процес використання нейронної мережі для передбачення майбутніх значень на основі попередніх даних.

Перш за все, для прогнозування за допомогою нейронних мереж потрібно мати набір даних, який містить інформацію про певний процес або явище, що ми хочемо передбачити. Цей набір даних зазвичай розділяється на навчальний та тестовий набори.

Наступним кроком є побудова моделі нейронної мережі. Для прогнозування можна використовувати різні типи нейронних мереж, наприклад, рекурентні нейронні мережі (RNN) [8] або звичайні нейронні мережі зі зворотним поширенням (backpropagation neural networks) [24].

Після побудови моделі мережі вона навчається за допомогою навчального набору даних. Під час навчання мережа змінює свої ваги та параметри, щоб зменшити помилки прогнозування на навчальному наборі.

Коли модель мережі навчилася на достатньо великій кількості даних, її можна використовувати для прогнозування майбутніх значень на тестовому наборі даних.

2.5 Змішана технологія ANFIS

Змішана технологія ANFIS (Adaptive Neuro-Fuzzy Inference System) [25] є комбінованою моделлю, що поєднує у собі нейронні мережі та нечітку логіку. Вона використовується для прогнозування в різних галузях, включаючи фінанси, економіку, медицину, транспорт, енергетику та інші.

Основна ідея застосування змішаної технології ANFIS полягає в тому, щоб поєднати переваги нейронних мереж та нечіткої логіки. Нейронні мережі можуть виявляти складні зв'язки між великою кількістю вхідних даних, тоді як нечітка логіка дозволяє використовувати експертні знання та визначати правила на основі нечітких змінних.

Змішана технологія ANFIS складається з чотирьох основних компонентів: вхідного шару, шару на основі нечіткої логіки, шару внутрішнього перетворення та

вихідного шару. Вхідний шар відповідає за приймання вхідних даних. Шар на основі нечіткої логіки використовує нечітку логіку для створення правил на основі вхідних даних. Шар внутрішнього перетворення використовується для перетворення нечітких змінних в числові значення. Вихідний шар відповідає за вивід результату.

Змішана технологія ANFIS має декілька переваг, таких як:

- здатність до навчання на основі даних;
- здатність до врахування нечітких змінних та експертних знань;
- здатність до роботи зі складними даними та знаходження складних залежностей;
- здатність до роботи з неструктурованими даними, як-от тексти та зображення;
- можливість використання, як класифікаційних, так і регресійних моделей;
- можливість досягнення високої точності прогнозування;
- можливість досягнення швидкої обробки даних та розрахунку прогнозів.

Процес роботи зі змішаною технологією ANFIS зазвичай включає наступні кроки:

1. Збір даних та підготовка їх для використання в моделі.
2. Визначення структури моделі та побудова вхідного шару.
3. Визначення нечітких змінних та правил на основі нечіткої логіки.
4. Побудова шару на основі нечіткої логіки.
5. Побудова шару внутрішнього перетворення та вихідного шару.
6. Навчання моделі за допомогою навчального набору даних та перевірка її на тестовому наборі даних.
7. Оцінка точності прогнозів та їх використання для подальшої роботи.

Змішана технологія ANFIS є потужним інструментом для прогнозування в різних галузях. Вона може бути особливо корисною для прогнозування умовних процесів, де наявність нечітких змінних та експертних знань може допомогти покращити точність прогнозування.

3 ОПИС ІНФОРМАЦІЇНОЇ ТЕХНОЛОГІЇ МОДЕЛЮВАННЯ ІНДИКАТОРІВ ПРОГРЕСУ ЦИФРОВОЇ ТРАНСФОРМАЦІЇ ЕКОНОМІКИ

3.1 Аналіз структури даних

В якості джерела даних використовуємо веб-ресурс під назвою The Global Innovation Index.

Надані дані містяться у форматі csv файлу, який був завантажений з відповідного джерела інформації. Для зчитування цього формату використовується бібліотека pandas.

Перш за все, необхідно імпортувати бібліотеку pandas за допомогою команди: `import pandas as pd`. Потім можна використати метод `pd.read_csv(data_path_to_file)` для завантаження даних з файлу. Змінна `data_path_to_file` містить шлях до файлу. Результат буде представлений у вигляді таблиці в середовищі виконання (рис. 3.1).

	.1	Rank Indicators	Albania	Algeria	Angola	Argentina	Armenia	Australia	Austria	.
0	Indicator	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	Global Innovation Index			84.0	115.0	127.0	69.0	80.0	25.0	17.0	
2	Innovation Input Sub-index			80.0	110.0	129.0	77.0	82.0	19.0	17.0	
3	Innovation Output Sub-index			89.0	118.0	117.0	62.0	73.0	32.0	21.0	
4	Index	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 668 columns

Рисунок 3.1 — Таблиця з необробленими даними

Проаналізувавши дані з таблиці (рис. 3.1) зроблені висновки, що дані мають недоліки: пусті комірки або значення NaN та строкові значення, ці проблеми поступово усунуті.

Перше це – заміна індексів таблиці на значення з першого стовпчика за допомогою простого методу DataFrame: `df.set_index('  ', inplace = True)`. Далі залишаємо лише числові дані: `df = df.select_dtypes(include='number')`. Після чого видаляємо рядків у яких всі значення NaN: `df = df.dropna(how='all')`, та стовпців: `df = df.dropna(axis=1)`. Також у таблиці є аномальні великі значення (рис. 3.2) які видалені наступним чином:

```
std = df.std()  # обчислення стандартного відхилення
index_std = std[std > 150].index  # індекси стовпців зі стандартним відхиленням більше 150
df = df.drop(index_std, axis=1)  # видалення стовпців з великим стандартним відхиленням
max = df.max().mean() + df.max().std() # розрахунок максимального значення для комірки
df = df[df.columns[df.max() < max]]  # видалення стовпців із значеннями вищими за максимальне
```

Тепер дані готові (рис. 3.3) для подальшої роботи з ними.

```
                Indicator
Albania             126.0
Algeria             130.0
Angola              132.0
Argentina           128.0
                      ...
Viet Nam.2       186402.3
Yemen.2            5054.7
Zambia.2          12606.9
Zimbabwe.2        16010.9
Unnamed: 667          NaN
Length: 398, dtype: object
```

Рисунок 3.2 — Результат перевірки таблиці на максимальні значення

	Global Innovation Index	Innovation Input Sub-index	Innovation Output Sub-index	1.	1.1.	1.1.1.	1.1.2.	1.2.
Albania	84.0	80.0	89.0	84.0	70.0	63.0	77.0	81.0
Algeria	115.0	110.0	118.0	99.0	103.0	108.0	99.0	105.0
Angola	127.0	129.0	117.0	116.0	122.0	87.0	128.0	103.0
Argentina	69.0	77.0	62.0	96.0	83.0	81.0	83.0	119.0
Armenia	80.0	82.0	73.0	55.0	82.0	87.0	76.0	54.0
...	...	...	...	...	...	...	...	...
Uzbekistan.1	25.3	37.8	12.8	57.3	52.1	65.5	38.8	50.7
Viet Nam.1	34.3	40.1	28.4	60.6	65.2	76.4	54.0	54.6
Yemen.1	13.8	19.2	8.3	17.5	0.0	0.0	0.0	30.8
Zambia.1	15.8	24.5	7.1	37.5	45.7	58.2	33.2	22.7
Zimbabwe.1	18.1	23.9	12.3	33.3	34.1	45.5	22.8	38.5

Рисунок 3.3 — Результат алгоритму обробки даних

3.2 Кластерний аналіз даних

Після підготовки даних йде етап їх аналізу. Його можливо швидко провести за допомогою кластерного аналізу . Його реалізація програмного коду є у бібліотеці sklearn, тому імпортуємо її, а також бібліотеку numpy для роботи з даними:

```
from sklearn.cluster import KMeans
import numpy as np
```

Тепер створення функції для кластеризації. Першою частиною буде перевірка на правильність розмірності даних на вході:

```
x, y = df_points.shape
if x < y:
    df_points = df_points.T
X = df_points.values
df_points – дані на вході.
kmeans = KMeans(n_clusters=3, random_state=10)
```

n_clusters – кількість кластерів для розбиття.

random_state – початковий набір даних.

Наступним кроком є навчання на наших даних за допомогою методу fit:

```
kmeans.fit(X)
Кінцевим кроком буде виведення графічного результату та повернення відфільтро-
ваної вибірки індексів:
labels = kmeans.labels_  # відфільтрований масив індексів
colors = ['r', 'g', 'b']  # Масив з кольорами
# Створення діаграми розсіювання точок даних, розфарбованих відповідно до мітки
кластера.
```

```
plt.scatter(X[:, 0], X[:, 1], c=[colors[label] for label in labels])
# Створення легенд для діаграми
legend_elements = [plt.Line2D([0], [0],
    marker='o',
    color='w',
    label='Cluster {}'.format(i),
    markerfacecolor=colors[i], markersize=10) for i in range(len(np.unique(la-
bels)))]
Додавання легенд на діаграму
plt.legend(handles=legend_elements)
plt.show()  # Відображення діаграми
return labels  # Повернення масиву фільтрованих індексів
```

Далі для прикладу приведений аналіз відношення індексу 1.1. та його підрівня. У результаті діаграми розсіювання точок даних(ДРТД) та відфільтровані індекси даних згідно кластеризації (рис. 3.4-3.6).

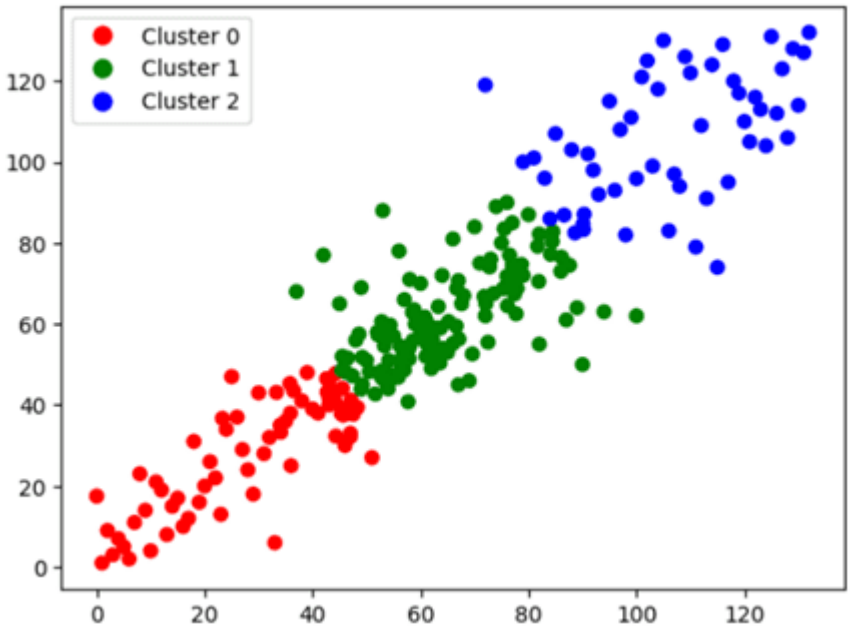

Рисунок 3.4 — ДРТД відношення індексу 1.1.1. до 1.1.

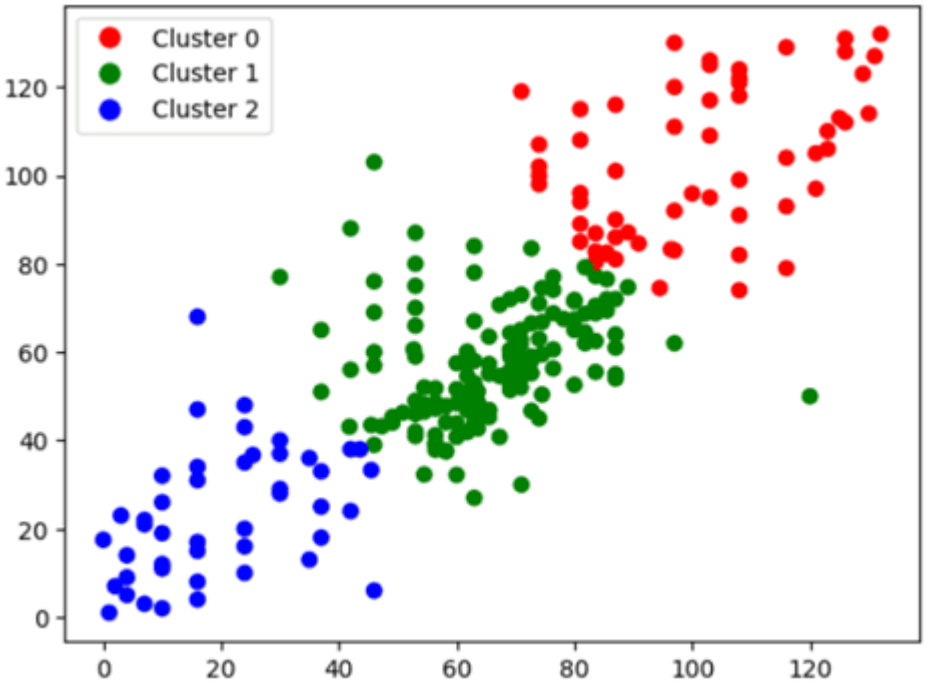

Рисунок 3.5 — ДРТД відношення індексу 1.1.2. до 1.1.

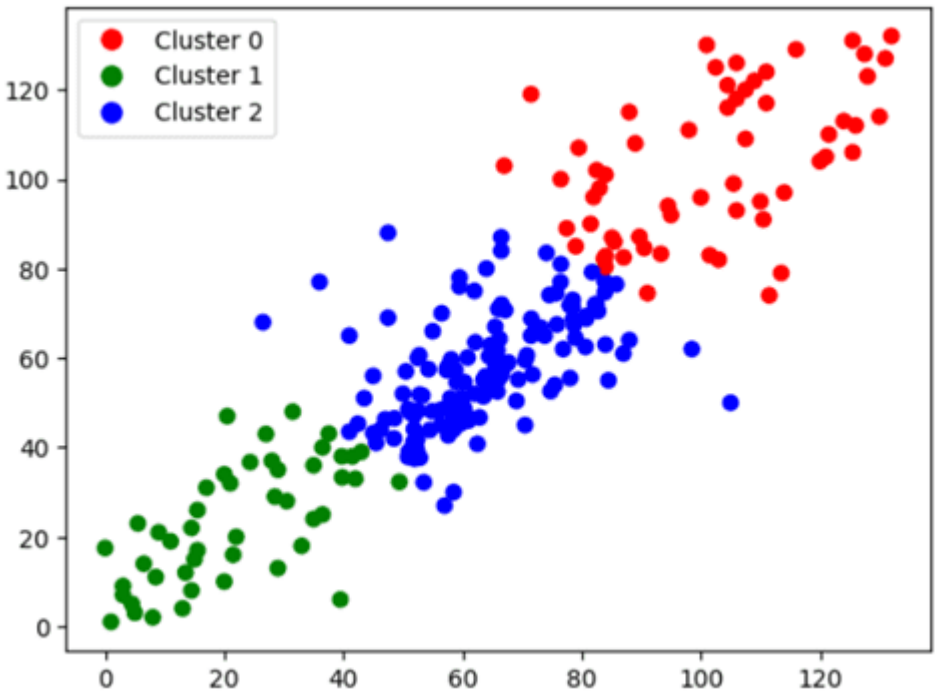

Рисунок 3.6 — ДРТД відношення індексів 1.1.1. та 1.1.2. до 1.1.

Згідно результатам кластеризації (рис. 3.4-3.6) видно, що дані, або мають велику похибку, або не пряму залежність, це потрібно уточнювати у експертів. Але також явно видно лінійну залежність, яку можна взяти за основу дослідження.

3.3 Інформаційна технологія нечіткого логічного виведення

Інформаційну технологію логічного виведення створено за допомогою бібліотеки skfuzzy.

3.3.1 Розробка моделі

Спершу імпортування бібліотек:

```
# Імпорт необхідних бібліотек
import numpy as np
import pandas as pd
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

Далі ініціалізація вхідних та вихідних даних, їх нижня та верхня границі допустимих значень. Кількість вхідних змінних дорівнює кількості елементів нижнього шару вихідної змінної, вихідна змінна в нашому випадку завжди одна.

```
# Визначення вхідних даних
inputs1 = ctrl.Antecedent(np.arange(0, 132, 1), 'inputs1')
inputs2 = ctrl.Antecedent(np.arange(0, 132, 1), 'inputs2')
# Визначення вихідних даних
outputs = ctrl.Consequent(np.arange(0, 132, 1), 'outputs')
```

Далі створення функцій залежності для наших вхідних та вихідних змінних. Їх кількість буде залежати від кількості вхідних змінних плюс змінної на виході та помножених на три. Множимо їх на три, бо нам потрібно створити терми дня значень, які будуть відповідати за низький, середній та високий показник.

```
# Визначення функцій залежності
inputs1['low'] = fuzz.trimf(inputs1.universe, [0, 0, 30])
inputs1['medium'] = fuzz.trimf(inputs1.universe, [24, 87, 87])
inputs1['high'] = fuzz.trimf(inputs1.universe, [46, 132, 132])

inputs2['low'] = fuzz.trimf(inputs2.universe, [-2.3, -2.3, 43])
inputs2['medium'] = fuzz.trimf(inputs2.universe, [19, 88, 88])
inputs2['high'] = fuzz.trimf(inputs2.universe, [60, 132, 132])
```

```
outputs['low'] = fuzz.trimf(outputs.universe, [0, 0, 37])
outputs['medium'] = fuzz.trimf(outputs.universe, [23, 75, 75])
outputs['high'] = fuzz.trimf(outputs.universe, [75, 132, 132.0])
```

Після створення функцій залежності, створено правила для них. Кількість цих правил буде дорівнювати встановлюється експертами, бо саме вони знають залежність входів та виходів. У цьому випадку правила встановлені, як лінійна залежність вхідних змінних до вихідної.

```
# Визначення правил
rule1 = ctrl.Rule(inputs1['low'] & inputs2['low'], outputs['low'])
rule2 = ctrl.Rule(inputs1['low'] & inputs2['medium'], outputs['medium'])
rule3 = ctrl.Rule(inputs1['low'] & inputs2['high'], outputs['high'])
rule4 = ctrl.Rule(inputs1['medium'] & inputs2['low'], outputs['low'])
rule5 = ctrl.Rule(inputs1['medium'] & inputs2['medium'], outputs['medium'])
rule6 = ctrl.Rule(inputs1['medium'] & inputs2['high'], outputs['high'])
rule7 = ctrl.Rule(inputs1['high'] & inputs2['low'], outputs['low'])
rule8 = ctrl.Rule(inputs1['high'] & inputs2['medium'], outputs['medium'])
rule9 = ctrl.Rule(inputs1['high'] & inputs2['high'], outputs['high'])
```

Далі створюється система:

```
# Визначення системи нечіткого виведення
fis = ctrl.ControlSystem(
[rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])
```

Після чого вона проходить симулювання:

```
# Обчислення результатів
simulation = ctrl.ControlSystemSimulation(fis)
```

Вже після чого можемо тестувати систему. Приклад вводу тестових даних:

```
# Введення тестових даних
simulation.input['inputs1'] = 16
simulation.input['inputs2'] = 12
```

Після введення тестових даних потрібно запустити їх розрахунок:

```
# Обчислення результату для тестових даних
simulation.compute()
```

Результат обчислень подані у графічному вигляді:

```
# Вивід результатів
print(simulation.output['outputs'])
# Перегляд результату на графіку залежності
outputs.view(sim=simulation)
```

Створення систем на прикладі малого сегменту 1.1. та результату тестування деяких з них наведені нижче (рис. 3.7, 3.8).

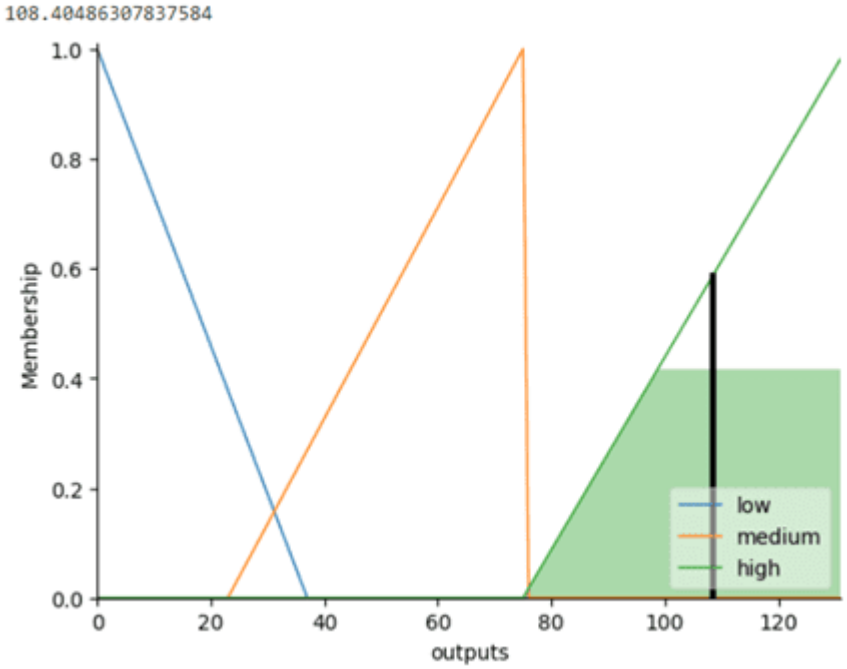

Рисунок 3.7 — Діаграма тестування на даних з України

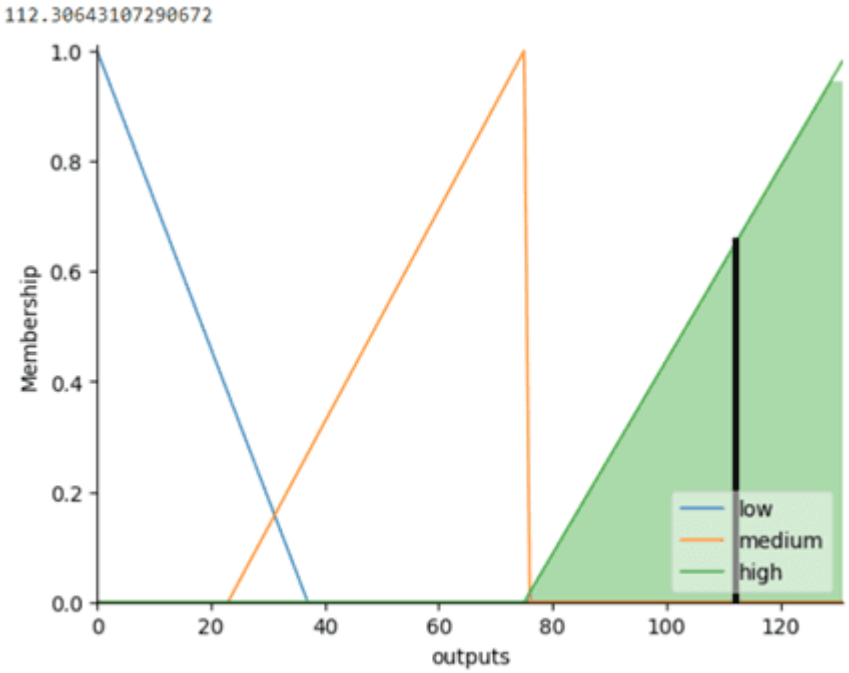

Рисунок 3.8 — Тестування на даних з Анголи

```
fact: 54.81802023516545 expect: [56.4] divine: [1.58197976]
fact: 55.734823515661525 expect: [67.1] divine: [11.36517648]
fact: 69.31750476726373 expect: [77.7] divine: [8.38249523]
fact: 67.32915177668139 expect: [72.8] divine: [5.47084822]
fact: 74.44745452100145 expect: [81.9] divine: [7.45254548]
fact: 53.932582815734996 expect: [57.9] divine: [3.96741718]
fact: 54.7324795388422 expect: [57.] divine: [2.26752046]
fact: 55.64521477999736 expect: [66.7] divine: [11.05478522]
fact: 54.737250199840126 expect: [52.8] divine: [-1.9372502]
fact: 54.66066252587998 expect: [61.3] divine: [6.63933747]
fact: 54.7324795388422 expect: [59.8] divine: [5.06752046]
fact: 109.87644075610874 expect: [100.] divine: [-9.87644076]
fact: 58.322071517707556 expect: [72.5] divine: [14.17792848]
fact: 71.37959514667826 expect: [77.4] divine: [6.02040485]
fact: 67.07057423364557 expect: [71.8] divine: [4.72942577]
fact: 54.41295289855067 expect: [57.7] divine: [3.2870471]
fact: 77.3852551551814 expect: [86.1] divine: [8.71474484]
fact: 39.46576801576414 expect: [45.4] divine: [5.93423198]
fact: 55.52831262939958 expect: [62.6] divine: [7.07168737]
fact: 40.116914196370736 expect: [49.2] divine: [9.0830858]
fact: 55.18034367729094 expect: [62.2] divine: [7.01965632]
fact: 54.50099386535935 expect: [55.3] divine: [0.79900613]
fact: 53.978367262206 expect: [52.7] divine: [-1.27836726]
fact: 44.28058699986871 expect: [48.6] divine: [4.319413]
fact: 73.51858013416259 expect: [75.5] divine: [1.98141987]
fact: 74.03430282455518 expect: [77.] divine: [2.96569718]
fact: 64.71341142286742 expect: [76.] divine: [11.28658858]
fact: 46.79402756075666 expect: [52.1] divine: [5.30597244]
fact: 55.234838891234034 expect: [65.2] divine: [9.96516111]
fact: 12.363590690139189 expect: [0.] divine: [-12.36359069]
```

Рисунок 3.9 — Часткова вибірка з тестування

Функції залежності побудовані на основі усіх даних з таблиці. Тому, на мій погляд, наявна дисперсія між очікуваним результатом та фактичним, а саме 7 одиниць, кількість розпаду, як у додатному напрямку так і у від’ємному приблизно однакова, тому, на мій погляд ,потрібно, ще фільтрувати дані, робити зміни у функціях залежності.

Але це була система для 2-х вхідних змінних, тепер щодо поставленої задачі, нам потрібно зробити 2 системи. У першій системі на вході будуть подаватись 54

вхідних змінних у другій 26, не кажучи про те, що нам потрібен експерт для встановлення правил, можливо приблизно розрахувати кількість правил:

- для першої системи: (54+15+5)*3=222
- для другої системи: (27+6+2)*3=105

Числа у дужках по черзі мають значення: кількість елементів найнижчого рівня (вхідних шар), кількість елементів наступного рівня за ієрархією 2, кількість елементів 1-го рівня. Ця сума домножиться на відношення елементів нижнього шару до верхнього. У результаті ми маємо 327 правил, у нашому випадку, для яких потрібно дослідити таку саму кількість взаємозв'язків даних. Це дуже об'ємна та клопітка праця, яка скоріш за все не дасть точного результату, тому прийнято рішення вважати, що цей метод не підходить для рішення нашої проблеми за відсутності експерта, який допоможе уникнути дослідження даних та надасть точні взаємозв'язки у даних.

3.4 Регресійне моделювання індикаторів прогресу цифрової трансформації економіки

У цьому розділі будуть створені та протестовані три моделі регресії за допомогою бібліотеки sklearn, яка надає вже готові моделі за все відомими алгоритмами машинного навчання.

Алгоритм має наступний вигляд:

1. Імпортування необхідних бібліотек:

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

2. Створення об'єкту моделі з бібліотек

```
lr_model = linear_model.LinearRegression()
```

3. Його тренування на наших даних

```
lr_model.fit(X_train_scaled, y_train)
```

4. Тестування на тестових даних

```
y_pred_lr = lr_model.predict(X_test_scaled)
```

5. Та аналізування результатів

```
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
print('mse_neural: ', mse_lr)
print('mae_neural: ', mae_lr)
```

У результаті отримаємо два числа:

- MSE - одна з найпоширеніших функцій втрат регресії. У середньоквадратичній похибці, також відомій, як втрата L2, ми обчислюємо похибку, підносячи різницю між прогнозованим і фактичним значенням до квадрата і усереднюючи її по всьому набору даних. MSE також відома, як квадратична втрата, оскільки штраф не пропорційний помилці, а квадрату помилки. Піднесення помилки до квадрату дає більшу вагу викидам, що призводить до плавного градієнта для малих помилок. Алгоритми оптимізації виграють від такого покарання за великі помилки, оскільки воно допомагає знайти оптимальні значення параметрів. MSE ніколи не буде від'ємним, оскільки помилки зведено до квадрату. Значення похибки коливається від нуля до нескінченності. MSE зростає експоненціально зі збільшенням похибки. Хороша модель має значення MSE ближче до нуля.

Наприклад, у регресії середньоквадратична похибка являє собою середньоквадратичний залишок (рис. 3.10).

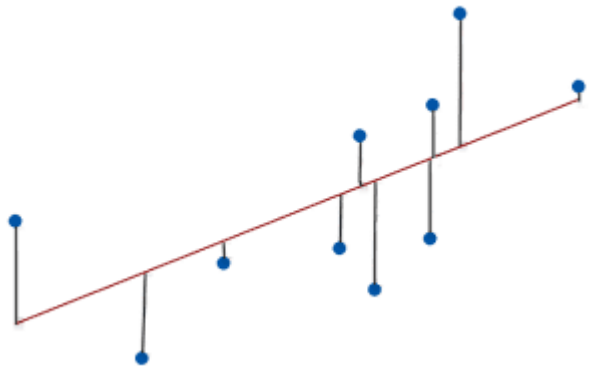

Рисунок 3.10 — Графічний приклад середньоквадратичної похибки

Чим ближче точки даних до лінії регресії, тим меншу похибку має модель, зменшуючи MSE. Модель з меншою похибкою дає точніші прогнози.

- MAE – Середня абсолютна похибка, також відома, як втрата L1, є однією з найпростіших функцій втрат і легкою для розуміння метрикою оцінки. Вона обчислюється шляхом взяття абсолютної різниці між прогнозованими і фактичними значеннями та усереднення її по всьому набору даних. Математично кажучи, це середнє арифметичне абсолютних похибок. MAE вимірює лише величину помилок і не враховує їхній напрямок. Чим нижчий MAE, тим вища точність моделі.

Першою моделлю є звичайна лінійна регресія найменших квадратів.

Linear Regression підбирає лінійну модель з коефіцієнтами w = (w1, ..., wp) для мінімізації залишкової суми квадратів між спостережуваними цілями в наборі даних і цілями, передбаченими лінійною апроксимацією.

Оцінки помилок після тестування: mse = 586.5, mae = ~16.

Друга модель – дерева рішень.

Код алгоритму аналогічний за винятку імпорту іншої моделі.

Результат має наступні значення: mse = 538.4, mae = ~ 16.3.

Остання модель - Випадковий ліс (рис. 3.11).

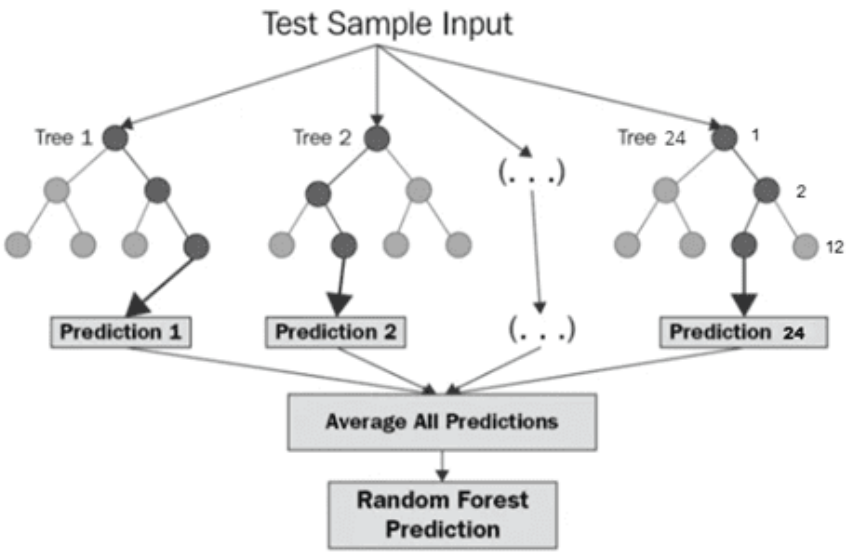

Рисунок 3.11 — Структура моделі випадковий ліс

На відміну від перших двох моделей у цієї ми використовуємо додаткові параметри:

- `n_estimators` – Кількість дерев у лісі.
- `max_depth` – Максимальна глибина дерева. Якщо None, то вершини розширюються до тих пір, поки всі листки не стануть чистими або поки всі листки не будуть містити менше ніж `min_samples_split` зразків.
- `min_samples_split` – Мінімальна кількість зразків (за замовчуванням = 2), необхідних для розділення внутрішнього вузла:
  - Якщо `int`, то вважати `min_samples_split` мінімальним числом.
  - Якщо `float`, то `min_samples_split` - це дріб, а `ceil(min_samples_split * n_samples)` - мінімальна кількість зразків для кожного розбиття.
- `random_state` – Керує, як випадковістю бутстрапування вибірок, що використовуються при побудові дерев (якщо `bootstrap=True`), так і вибіркою ознак, що враховуються при пошуку найкращого розбиття у кожному вузлі (якщо `max_features < n_features`).

Результати після тестування: mse = 345.08, mae = ~ 12.3.

Після початкових тестування видно, що точність останньої моделі вища також після аналізу декількох ресурсів більшість віддають перевагу моделі випадкових лісів для схожих задач до нашої. Тому цю модель рекомендовано взяти за основу подальших досліджень.

Після додавання більшої вибірки та тестування з різними параметрами була побудована модель:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error


model = RandomForestRegressor(n_estimators = 28, random_state=81,
                              max_depth=12, n_jobs=-1)
model.fit(X_train_scaled, y_train.values.reshape((551,)))
```

```
y_pred_RF = model.predict(X_test_scaled)

mse_RF = mean_squared_error(y_test, y_pred_RF)
mae_RF = mean_absolute_error(y_test, y_pred_RF)
print(f'{mse_RF} {mae_RF}')
```

У моделі 28 дерев з максимальною глибиною 12 та початковим набором ваг 81. Результат на тестовій вибірці: mse = 34.62, mae = ~ 3.9.

3.5 Нейромережеве моделювання

3.5.1 Алгоритм створення нейронної мережі

Перше, що потрібно зробити після підготовки даних – це з'ясувати, яку задачу буде виконувати нейромережа. У нашому випадку досліджуватиметься регресія наявних даних для передбачення нових даних.

Далі головне завдання побудувати модель нейронної мережі. Після аналізування джерел інформації встановлена область дослідження прихованих шарів. Модель (рис. 3.12) буде складатися з таких шарів:

1. вхідного, який має 54 нейрона;
2. вихідного, який має 1 нейрон;
3. прихованих, кількість яких буде від 1 до 3.

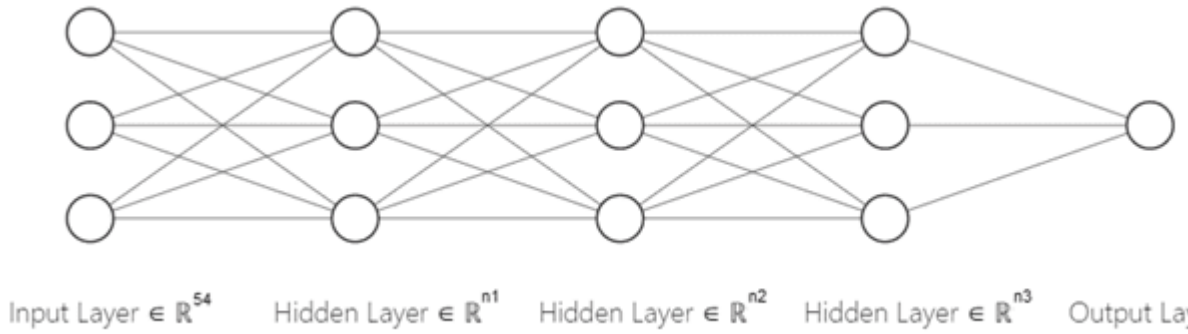

Рисунок 3.12 — Структура повністю згорткової нейронної мережі в вигляді графа

На практиці за допомогою бібліотеки tensorflow створити модель можливо наступним чином:

В об’єкт `tf.keras.Sequential` передати масив наших шарів, які розташовані в ньому по черзі. Створення шару виконується викликом об’єкту `tf.keras.layers.Dense` та передачі до нього параметрів:

- `units` – Додатне ціле число, розмірність вихідного простору (кількість нейронів).
- `activation` – Функція активації, яку потрібно використати. Якщо ви нічого не вказуєте, активація не застосовується (тобто "лінійна" активація: `a(x) = x`).
- `input_shape` – Розмірність вхідних даних.
- `kernel_regularizer` – Функція регуляризатора, застосована до матриці ваг ядра.

Приклад:

```
tf.keras.layers.Dense(64,
                                    input_shape=(n_inputs,),
                                    kernel_regularizer=regularizers.l2(0.01),
                                    activation='relu')
```

Далі виконуємо компіляцію моделі:

```
self.model.compile(optimizer=self.optimizer,  # Компілюємо модель
                           loss='mse',
                           metrics=['mae'])
```

Вона має такі параметри:

- `optimizer` – Рядок (ім'я оптимізатора) або екземпляр оптимізатора.
- `loss` – Функція втрат. Може бути рядком (назва функції втрат), або `tf.keras.losses`.
- `metrics` – Перелік метрик, які будуть оцінюватися моделлю під час навчання та тестування.

Тепер модель готова до навчання:

```
self.model.fit(self.X_train,
               self.y_train,
               validation_data=(self.X_val, self.y_val),
               epochs=180,
               batch_size=32,
               verbose=0,
               workers=16,
               use_multiprocessing=True,
               validation_split=0.2)
```

Параметри:

- `x` – Вхідні дані.
- `y` – Дані цілі.
- `batch_size` – Ціле число або Ні. Кількість відліків для оновлення градієнта. Якщо не вказано, `batch_size` за замовчуванням дорівнює 32.
- `epochs` – Ціле число. Кількість епох для навчання моделі.
- `verbose` – 'auto', 0, 1 або 2. Режим багатослівності. 0 = без звуку, 1 = індикатор виконання, 2 = один рядок за епоху. 'auto' за замовчуванням дорівнює 1 для більшості випадків, але 2 при використанні з `ParameterServerStrategy`.
- `validation_data` – Дані, на основі яких можна оцінити втрати та будь-які метрики моделі на кінець кожної епохи. Модель не буде навчатися на цих даних.
- `workers` – Ціле число. Використовується лише для введення генератором або `keras.utils.Sequence`. Максимальна кількість процесів для запуску при використанні потоків на основі процесів. Якщо не вказано, за замовчуванням буде встановлено значення 1.
- `use_multiprocessing` – Boolean. Використовується лише для введення генератора або `keras.utils.Sequence`. Якщо значення True, використовувати багатопроцесорність на основі процесів. Якщо не вказано, `use_multiprocessing` буде за замовчуванням дорівнювати `False`. Зауважте, що оскільки ця реалізація покладається на багатопроцесорність, вам не слід передавати генератору нерозбірливі аргументи, оскільки вони не можуть бути легко передані дочірнім процесам.

Після навчання тестуємо модель:

```
self.model.evaluate(self.X_test,
                    self.y_test,
                    batch_size=32,
                    verbose=0,
                    workers=16,
                    use_multiprocessing=True)
```

Параметри аналогічні до тренувальних.

3.5.2 Створення меделі нейронної мережі

Для створення моделі нам потрібно встановити потрібну кількість шарів та їх нейронів, тому було проведено тестування меделей на їх різній кількості.

3.5.2.1 Модель IISI з Dense + ReLU

Тестування моделі з одним прихованим шаром та підбір кількості нейронів. Після тестування (рис. 3.13), з кроком кількості нейронів 32, було отримано діапазон нейронів для одного шару з найркащим результатом – 1500-3000.

	n1	loss	mae
51	1664	122.689751	6.820673
52	1696	122.056343	6.862416
57	1856	118.844925	6.659274
58	1888	122.264809	6.687940
59	1920	121.900864	6.811425
65	2112	120.114655	6.712281
70	2272	122.488899	6.766075
78	2528	121.216187	6.760608
93	3008	122.699829	6.704841

Рисунок 3.13 – Найкращі результати після тестування одного шару

Тестування моделі для двох шарів. Час на навчання в рази збільшиться, тому досліджуємо лише зміну нейронів для другого шару. У результаті (рис. 3.14) якість моделі збільшилась приблизно на 32%.

	n1	n2	loss	mae
63	1024	1504.0	83.516167	5.524597
7	1856	1152.0	82.758064	5.428835
20	1856	1568.0	82.983551	5.407173
5	1856	2976.0	85.128380	5.176396
6	1856	3232.0	85.935432	5.495911

Рисунок 3.14 – Найкращі результати після тестування двох шарів

Тестування 3-х шарів ще збільшило час навчання, але результати (рис. 3.15) не відрізняються від 2-х шарової моделі.

	n1	n2	n3	loss	mae
26	1856	1500	1696	85.223228	5.435703
27	1856	1500	1760	85.854782	5.412891
29	1856	1500	1888	83.966270	5.547966

Рисунок 3.15 – Найкращі результати після тестування 3-х шарів

У результаті тестувань була визначена модель з 2 шарами (рис. 3.16).

```
model = tf.keras.Sequential([  # Define a sequential model
    tf.keras.layers.Dense(256,
                          input_shape=(n_inputs,),
                          activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(1, activation='linear')])
```

```
<class 'keras.engine.sequential.Sequential'>
Model: "sequential_1668"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5006 (Dense)          (None, 256)               14080

 dropout_2622 (Dropout)      (None, 256)               0

 dense_5007 (Dense)          (None, 256)               65792

 dropout_2623 (Dropout)      (None, 256)               0

 dense_5008 (Dense)          (None, 1)                 257

=================================================================
Total params: 80,129
Trainable params: 80,129
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.16 – Модель IISI із щільних шарів та функцією активації ReLU

Точність моделі приблизно 87% по всій вибірці при допустимій похибці 10.

3.5.2.1 Модель IOSI з Dense + ReLU

Після безлічі тестувань була створена модель (рис. 3.17) з двома прихованими шарами.

```
model = tf.keras.Sequential([  # Define a sequential model
    tf.keras.layers.Dense(512,
                          input_shape=(n_inputs,),
                          activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(1, activation='linear')])
```

```
<class 'keras.engine.sequential.Sequential'>
Model: "sequential_1551"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4655 (Dense)          (None, 512)               14336

 dropout_2388 (Dropout)      (None, 512)               0

 dense_4656 (Dense)          (None, 256)               131328

 dropout_2389 (Dropout)      (None, 256)               0

 dense_4657 (Dense)          (None, 1)                 257

=================================================================
Total params: 145,921
Trainable params: 145,921
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.17 – Модель IOSI із щільних шарів та функцією активації ReLU

Точність моделі приблизно 92% по всій вибірці при допустимій похибці 10.

3.5.2.1 Модель IISI з LSTM

Також створена модель з більш глибокою архітектурою LSTM.

```
import torch.nn as nn

class AirModel(nn.Module):
    def __init__(self):
```

```
        super().__init__()
        h_n = 256
        self.lstm = nn.LSTM(input_size=54, hidden_size=h_n, num_layers=2,
batch_first=True)
        self.linear = nn.Linear(h_n, 1)

    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x
```

Модель складається з 2-х прихованих шарів та 256 прихованих шарів в них. Після тренування та тестування така модель моє точність 96.5% по всій вибірці при допустимій похибці 10.

3.5.2.1 Модель IOSI з LSTM

Код моделі IOSI нижче:

```
import torch.nn as nn

class AirModel(nn.Module):
    def __init__(self):
        super().__init__()
        h_n = 1024
        self.lstm = nn.LSTM(input_size=27,
                            hidden_size=h_n,
                            num_layers=1,
                            batch_first=True)
        self.linear = nn.Linear(h_n, 1)

    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x
```

У моделі 1 прихований шар з 1024 нейронами. У результаті тренування та тестування точність 95%.

ВИСНОВКИ

Розроблена інформаційна технологія моделювання індикаторів прогресу цифрової трансформації економіки.

Ідентифікована та визначена конкретна проблемна область і змінні. Моделювання проводилось для індикаторів прогресу цифрової трансформації економіки The Global Innovation Index. Визначена структура індексів та субіндексів.

Вибрані дані для аналізу та проведена їх попередня обробка. Це надало можливість очисти дані та провести їх нормалізацію.

Проведено кластерний аналіз для розбиття заданої вибірки об’єктів на кластери, які визначають різні рівні значень індикаторів прогресу цифрової трансформації економіки.

Проведено аналітичний огляд математичних апаратів, які можуть бути застосовані для вирішення поставленої проблеми. Розглянуті регресійні моделі, апарат нечіткої логіки та штучні нейронні мережі.

Отримані результати кваліфікаційної роботи можуть бути використані для подальших досліджень моделювання індикаторів прогресу цифрової трансформації економіки на світовому рівні та на рівні регіонів України.

Результати дослідження підтвердили гіпотезу, щодо прогнозування індикаторів прогресу цифрової трансформації економіки можна досягнути шляхом застосування інформаційної технології, що реалізує модель машинного навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Soumitra D. Global innovation index 2020: who will finance innovation? / D. Soumitra, L. Bruno, W. Sacha // WIPO, 2020.

2. Barbara J. Efficiency of national innovation systems: Poland and Bulgaria in the context of the Global Innovation Index / J. Barbara, A. Matysek-Jędrych, M. Katarzyna // Comparative Economic Research. Central and Eastern Europe. – Warsaw: De Gruyter, 2017. – Vol. 30, № 3. – P. 77-94.

3. JAIN Data clustering: 50 years beyond K-means / JAIN, K. Anil // Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, Proceedings, Part I 19. – P. 3-4.

4. Gan G. Data clustering: theory, algorithms, and applications. Society for Industrial and Applied Mathematics / G. Gan, M. Chaoqun, W. Jianhong // SIAM, 2020.

5. Налапко О. Л. Метод кластеризації даних на основі еволюційної оптимізації котячих зграй / В. Г. Козлов, О. О. Звєрєв // Publishing House “Baltija Publishing”, 2022.

6. Щербаков Є. Ю. Порівняння апроксимаційних методів, регресії та ітераційних методів машинного навчання при аналізі часових рядів економічних показників / Є. Ю. Щербаков // ББК 65.9 (4УКР) М 77, 2018. – С.266.

7. Maulud D. A review on linear regression comprehensive in machine learning / D. Maulud, A. M. Abdulazeez // Journal of Applied Science and Technology Trends, 2020. – Vol. 1, № 4. – P. 140-147.

8. Wenpeng Y. Comparative study of CNN and RNN for natural language processing / Y. Wenpeng, K. Kann, Y. Mo., H. Schütze // arXiv preprint arXiv:1702.01923, 2017.

9. Yong. A review of recurrent neural networks: LSTM cells and network architectures / Y. Yong, et al // Neural computation, 2019. – Vol.31, № 7. – P. 1235-1270.

10. ZADEH "Fuzzy logic." Granular, Fuzzy, and Soft Computing. / ZADEH, A. Lotfi // Granular, Fuzzy, and Soft Computing. New York, NY: Springer US, 2023. – P. 19-49.

11. MENDEL Fuzzy logic systems for engineering: a tutorial. / MENDEL, M. Jerry // Proceedings of the IEEE, Ieee, 1995. – Vol. 83, № 3. – P. 345-377.

12. Yan J. Using fuzzy logic: Towards intelligent systems / Michael Ryan, James Power. / J. Yan, M. Ryan, J. Power // Prentice-Hall, Inc, 1995.

13. Karaboga D. Adaptive network based fuzzy inference system (ANFIS) training approaches: a comprehensive survey / D. Karaboga, E. Kaya. // Artificial Intelligence Review, Springer, 2019. – Vol. 52. – P. 2263-2293.

14. Jang J. S. ANFIS: adaptive-network-based fuzzy inference system. / J. S. Jang // IEEE transactions on systems, man, and cybernetics, IEEE, 1993. – Vol. 23, № 3. – P. 665-685.

15. Lawrence J. Introduction to neural networks. / J. Lawrence // California Scientific Software, 1993.

16. Abadi M. TensorFlow: learning functions at scale. / M. Abadi // Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, 2016. – P. 1-1.

17. Reed J. torch. fx: Practical Program Capture and Transformation for Deep Learning in Python / J. Reed, et al // Proceedings of Machine Learning and Systems, 2022. – Vol. 4. – P. 638-651.

18. Panchal G. Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers / G. Panchal, et al // International Journal of Computer Theory and Engineering, 2011. – Vol. 3, № 2. – P.332-337.

19. Bebis G. Feed-forward neural networks / G. Bebis, M. Georgiopoulos // Ieee Potentials, IEEE, 1994. – Vol.13, № 4. – P. 27-31.

20. Zaremba W. Recurrent neural network regularization / W. Zaremba, I. Sutskever, O. Vinyals // arXiv preprint arXiv:1409.2329, 2014.

21. O'Shea K. An introduction to convolutional neural networks / K. O'Shea, R. Nash // arXiv preprint arXiv:1511.08458, 2015.

22. Samek W. Evaluating the visualization of what a deep neural network has learned / W. Samek, et al // IEEE transactions on neural networks and learning systems, IEEE, 2016. – Vol. 28, № 11. – P. 2660-2673.

23. Григорак І. А. Побудова та навчання штучних нейронних мереж: перцептрон. / І. А. Григорак // Збірник тез доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції (випуск 71) 18-19 жовтня 2022 р. на сайті www.konferenciaonline.org.ua, 2022. – С. 10.

24. Hecht-Nielsen R. Theory of the backpropagation neural network. / R. Hecht-Nielsen // Neural networks for perception, Elsevier, 1992. – P. 65-93.

25. Akkoç S. An empirical comparison of conventional techniques, neural networks and the three stage hybrid Adaptive Neuro Fuzzy Inference System (ANFIS) model for credit scoring analysis: The case of Turkish credit card data. / S. Akkoç // European Journal of Operational Research, Elsevier, 2012. – Vol. 222, № 1. – P. 168-178.

ДОДАТОК А

ПОПЕРЕДНЯ ОБРОБКА ДАНИХ

```
import pandas as pd  # імпорт бібліотеки pandas


def cleaning_data(file_path) -> pd.DataFrame:
    """Функція для очищення даних"""
    df = pd.read_csv(file_path)  # читання даних з файлу
    df.set_index(' ', inplace=True)
    # вибір стовпців з числовими значеннями
    df = df.select_dtypes(include='number')
    df = df.dropna(how='all')  # видалення стовпців з пропущеними значенням
    df = df.dropna(axis=1)
    std = df.std()  # обчислення стандартного відхилення
    # індекси стовпців зі стандартним відхиленням більше 150
    index_std = std[std > 150].index
    # видалення стовпців з великим стандартним відхиленням
    df = df.drop(index_std, axis=1)
    return df
```

ДОДАТОК Б

ОБ’ЄДНАННЯ МАСИВІВ ДАНИХ

```
year_list = ['2022', '2021', '2020']  # Список років для відбору
df = [0]*len(year_list)  # Створення масиву для зберігання
data = 0
for i, obj in enumerate(year_list):  # Цикл по масиву років
  path = 'Analysis_' + obj + '.csv'  # Створення шляху до файлу
  df[i] = cleaning_data(path)        # Очистка даних від сміття
  if type(data) is int:              # Перевірка на першу ітерацію
    data = df[i].copy()              # Повне копіювання даних
  else:
# Конкатенація масивів по одній осі
    data = pd.concat([data, df[i]], axis=1)
```

ДОДАТОК В

НОРМАЛІЗАЦІЯ ДАНИХ

```
tmp = data.loc[:,'6.1.1.':] # [:,:'5.3.5.'] для Innovation Input Sub-index
list_columns_name_3dot = tmp.columns[tmp.columns.str.count(r'\.') ==
3].tolist() # Список індексів с потрібними колонками для вхідних даних
input_index_data = list_columns_name_3dot
# / [' Innovation Input Sub-index']
output_index_data = [' Innovation Output Sub-index']

# Масиви даних для вхідних та вихідних змінних
X = data[input_index_data]
y = data[output_index_data]

# Розбиття масивів на навчальну та тестову вибірки
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=20)

# Створення об’єкту моделі нормалізації даних
scaler=StandardScaler()
scaler.fit(X_train) # навчання моделі на навчальній вибірці

# Нормалізація вибірок
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Підготовка даних для навчання та тренування моделей
input_data = {'X_train':X_train,
              'X_test':X_test,
              'X_train_scaled':X_train_scaled,
              'X_test_scaled':X_test_scaled}

output_data = {'y_train':y_train,
               'y_test':y_test}
```

ДОДАТОК Г

ФУНКЦІЯ КЛАСТЕРИЗАЦІЇ ДАНИХ

```
def Clustering(df_points):
  # Перевірка даних на правильність їх розмірності.
  x, y = df_points.shape
  if x < y:
    df_points = df_points.T
  X = df_points.values
  # Створення об'єкту KMeans з 3 кластерами.
  kmeans = KMeans(n_clusters=3, random_state=10)
  # Тренування моделі KMeans на даних.
  kmeans.fit(X)
  # Отримання мітки кластерів для кожної точки даних
  labels = kmeans.labels_
  # Визначення списку кольорів для діаграми розсіювання
  colors = ['r', 'g', 'b']
  # Створення діаграми розсіювання точок даних, розфарбованих за міткою кластера
  plt.scatter(X[:, 0], X[:, 1], c=[colors[label] for label in labels])
  # Створення легенд для ділянок
  legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label='Cluster
{}'.format(i), markerfacecolor=colors[i], markersize=10) for i in
range(len(np.unique(labels)))]
  plt.legend(handles=legend_elements)
  # Відображення plot
  plt.show()
  return labels
```

ДОДАТОК Ґ

СИСТЕМА НЕЧІТКОГО ЛОГІЧНОГО ВИВЕДЕННЯ

```
"""
Система нечіткого виводу для моделювання індикаторів сталого розвитку.
"""

# Імпорт необхідних бібліотек
import numpy as np
import pandas as pd
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Визначення вхідних даних
inputs1 = ctrl.Antecedent(np.arange(0, 132, 1), 'inputs1')
inpurt2 = ctrl.Antecedent(np.arange(-3, 132, 1), 'inputs2')


# Визначення вихідних даних
outputs = ctrl.Consequent(np.arange(0, 132, 1), 'outputs')

# Визначення функцій залежності
inputs1['low'] = fuzz.trimf(inputs1.universe, [0, 0, 30])
inputs1['medium'] = fuzz.trimf(inputs1.universe, [24, 87, 87])
inputs1['high'] = fuzz.trimf(inputs1.universe, [46, 132, 132])

inpurt2['low'] = fuzz.trimf(inpurt2.universe, [-2.3, -2.3, 43])
inpurt2['medium'] = fuzz.trimf(inpurt2.universe, [19, 88, 88])
inpurt2['high'] = fuzz.trimf(inpurt2.universe, [60, 132, 132])

outputs['low'] = fuzz.trimf(outputs.universe, [0, 0, 37])
outputs['medium'] = fuzz.trimf(outputs.universe, [23, 75, 75])
outputs['high'] = fuzz.trimf(outputs.universe, [75, 132, 132.0])

# Визначення правил
rule1 = ctrl.Rule(inputs1['low'] & inpurt2['low'], outputs['low'])
rule2 = ctrl.Rule(inputs1['low'] & inpurt2['medium'],
outputs['medium'])
rule3 = ctrl.Rule(inputs1['low'] & inpurt2['high'],
outputs['high'])
rule4 = ctrl.Rule(inputs1['medium'] & inpurt2['low'],
outputs['low'])
rule5 = ctrl.Rule(inputs1['medium'] & inpurt2['medium'],
outputs['medium'])
rule6 = ctrl.Rule(inputs1['medium'] & inpurt2['high'],
```

```
outputs['high'])
rule7 = ctrl.Rule(inputs1['high'] & inpurt2['low'], outputs['low'])
rule8 = ctrl.Rule(inputs1['high'] & inpurt2['medium'],
outputs['medium'])
rule9 = ctrl.Rule(inputs1['high'] & inpurt2['high'],
outputs['high'])

# Визначення системи нечіткого виведення
fis = ctrl.ControlSystem(
[rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])

# Обчислення результатів
simulation = ctrl.ControlSystemSimulation(fis)

# Введення тестових даних
simulation.input['inputs1'] = 16
simulation.input['inputs2'] = 12

# Обчислення результату для тестових даних
simulation.compute()

# Вивід результатів
print(simulation.output['outputs'])

# Перегляд результату на графіку залежності
outputs.view(sim=simulation)
```

ДОДАТОК Д

МОДУЛЬ З КЛАСОМ ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ

```
""" Модуль для нейронної системи."""
import os.path as f

import matplotlib.pyplot as plt
import tensorflow as tf


class NeuronSystem:
    """Neuron System"""

    def __init__(self, name, input_data, output_data, alp = 5):
        """
        Ініціалізація об'єкту класу.

        :param name: назва моделі
        :param input_data: набір вхідних даних
        :param output_data: набір вихідних даних
        """
        self.shape_input = input_data['X_train_scaled'].shape
        # Задаємо вхідні та вихідні дані
        self.name = name
        # кількість вхідних параметрів
        self.file_path = './model/' + f'model_{name}.h5'  # шлях до файлу

        self.X_train_scaled = input_data['X_train_scaled']
        self.y_train = output_data['y_train']

        self.x_test_scaled = input_data['X_test_scaled']
        self.y_test = output_data['y_test'].values

        self.history = -1

        self.callback = tf.keras.callbacks.EarlyStopping(monitor='loss',
                                                         patience=10)
        # Цей зворотний виклик зупинить тренування, якщо протягом трьох епох
        # поспіль програш не покращиться.
        if f.exists(self.file_path):  # перевіряємо наявність файлу
            self.model = tf.keras.models.load_model(
                self.file_path)  # завантажуємо модель
            # з файлу
        else:
            self.build_model(alp)  # створюємо модель
            self.train_model()  # навчаємо модель

    def build_model(self, alp = 5):
        """
        Створюємо модель нейромережі.

        """
        self.model = None
        samples = self.shape_input[0]
        n_inputs = self.shape_input[1]
        nh = round(samples/(alp*(n_inputs+1)))
```

```
        self.model = tf.keras.Sequential([  # Визначення простої послідовної моделі
            tf.keras.layers.Dense(512,
                                  input_shape=(n_inputs,),
                                  activation='relu'),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.Dense(1, activation='linear')
        ])
        # ініціалізація алгоритму оптимізації .RMSprop(0.001)
        self.optimizer = tf.keras.optimizers.Adam()

        self.model.compile(optimizer=self.optimizer,  # Компілюємо модель
                           loss='mean_squared_error',
                           metrics=['mae'])

    def retrain_model(self):
        """
        Перенавчаємо модель
        """
        self.build_model()
        self.history = self.model.fit(self.X_train_scaled,
                                      self.y_train,
                                      epochs=80,
                                      batch_size=32,
                                      validation_split=0.2)
        self.acc = self.history.history['mae']
        self.val_acc = self.history.history['val_mae']
        self.print_loss_graph()
        self.print_acc_graph()
        _ = self.model.evaluate(self.x_test_scaled, self.y_test)
        self.model.save(self.file_path)  # Зберігаємо модель у файл

    def train_model(self):
        """
        Навчаємо модель.

        :param input_data: набір вхідних даних
        :param output_data: набір вихідних даних
        """

        self.build_model()  # перезапускаємо модель
        self.history = self.model.fit(self.X_train_scaled,
                                      self.y_train,
                                      epochs=80,
                                      batch_size=32,
                                      validation_split=0.2,)
        self.acc = self.history.history['mae']
        self.val_acc = self.history.history['val_mae']
        self.print_loss_graph()
        self.print_acc_graph()
        self.get_pred_for_test_data()
        # Оцінюємо точність моделі на тестовому наборі
        _ = self.model.evaluate(self.x_test_scaled, self.y_test)
        self.model.save(self.file_path)  # Зберігаємо модель у файл

    def get_pred_for_test_data(self):
        """Отримання прогнозів для тестових даних """
        predictions = self.model.predict(self.x_test_scaled)
        for i, obj in enumerate(predictions):
```

```
            # print("Дані вхідного змінна: " + str(self.x_test_scaled[i]))
            print("Очікувана вихідна змінна: " + str(self.y_test[i]))
            print("Передбачена вихідна змінна: " + str(obj))

    def print_loss_graph(self):
        """функції друкування графіку втрат"""
        plt.plot(self.history.history['loss'])
        plt.plot(self.history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()

    def print_acc_graph(self):
        """Функція виводить графік з точністю"""
        plt.plot(self.history.history['mae'])
        plt.plot(self.history.history['val_mae'])
        plt.title('model acc')
        plt.ylabel('acc')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()

    def get_prediction(self, input_data):
        """Створити передбачення для новий даних"""
        return self.model.predict([input_data], verbose = None)
```

ДОДАТОК Е

LSTM НЕЙРОННА СИСТЕМА НА БІБЛІОТЕЦІ TORCH

```
import torch.nn as nn
import numpy as np
import torch.optim as optim
import torch.utils.data as data

class AirModel(nn.Module):
    def __init__(self):
        super().__init__()
        h_n = 1024
        self.lstm = nn.LSTM(input_size=27, hidden_size=h_n, num_layers=1, batch_first=True)
        self.linear = nn.Linear(h_n, 1)

    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x

n_epochs = 700
for epoch in range(n_epochs):
    model.train()
    for X_batch, y_batch in loader:
        y_pred = model(X_batch)
        loss = loss_fn(y_pred, y_batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    # Validation
    if epoch % 100 != 0:
        continue
    model.eval()
    with torch.no_grad():
        y_pred = model(X_train)
        train_rmse = np.sqrt(loss_fn(y_pred, y_train))
        y_pred = model(X_test)
        test_rmse = np.sqrt(loss_fn(y_pred, y_test))
    print("Epoch %d: train RMSE %.4f, test RMSE %.4f" % (epoch, train_rmse, test_rmse))
```

ДОДАТОК Є

АЛГОРИТМ ОЦІНКИ ТОЧНОСТІ

```
fuct_test = model(X_test).detach().numpy()
fuct_train = model(X_train).detach().numpy()
all_fuct_data = np.append(fuct_train, fuct_test, axis=0)
expect_rez_test = y_test
expect_rez_train = y_train
expect_rez = np.append(expect_rez_train, expect_rez_test, axis=0)

expect_rez = expect_rez.reshape((689,))
all_fuct_data = all_fuct_data.reshape((689,))
rez_df_object = {
   'Fuct': all_fuct_data,
   'Expect': expect_rez
}

rez_df = pd.DataFrame(rez_df_object)
rez_df['div'] = rez_df['Expect'] - rez_df['Fuct']
rez_df['div_abs'] = rez_df['div'].abs()
(rez_df['div_abs'].shape[0] - rez_df['div_abs'][rez_df['div_abs'] > 10].shape[0])/rez_df['div_abs'].shape[0]*100
```

ДОДАТОК Ж

СТРУКТУРА ДАНИХ