Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Поєднання нейронних мереж з моделями штучного інтелекту, що мають пояснення знань
(тема)

Виконав:
студент 2 курсу, групи СШМ-20-1
Доценко К.О.
(прізвище, ініціали)

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

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник доц. Вітько О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

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

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Доценку Костянтину Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Поєднання нейронних мереж з моделями штучного інтелекту, що мають пояснення знань

затверджена наказом університету від 08 листопада 20 21 р. № 1695 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 08 грудня 20 21 р.

3. Вихідні дані до роботи наукова література, науково-технічні публікації, дані інтернет-джерел, методи пояснення роботи нейронної мережі, інструменти розробки на мові python, бібліотеки машинного навчання, мова OWL, редактор онтологій Protege.

4. Перелік питань, що потрібно опрацювати в роботі 1 Аналіз предметної області та постановка задачі, 2 Зрозумілий штучний інтелект, 3 Огляд використаних підходів для представлення знань, 4 Розробка системи пояснення роботи нейронної мережі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Рисунок 1.1 – Схематичне уявлення зрозумілого штучного інтелекту, Рисунок 1.2 – Класифікація літератури XAI з розподілом опублікованих наукових статей з плином часу, Рисунок 1.3 – Приклад можливої неправильної класифікації військових танків в залежності від фону, Рисунок 2.1 – Розподіл наукових прац, що визначають концепцію зрозумілості, Рисунок 2.2 – Діаграма основних факторів, що формують структуру машинно-генерованого пояснення, Рисунок 2.3 – Класифікація підходів до оцінки методів пояснення та розподіл відносних наукових досліджень за категоріями, Рисунок 2.4 – Сучасний стан і передбачувані рамки для зрозумілого штучного інтелекту, Рисунок 2.5 – Приклад класифікатора дерева рішень, Рисунок 2.6 – Приклад лінійної моделі та значення повернених ознак, Рисунок 2.7 – Приклад глобально та локально інтерпретованої моделі, Рисунок 2.8 – Узагальнюваний підхід до зворотного проектування, Рисунок 2.9 – Одна з можливих класифікацій методів пояснення, Рисунок 2.10 – Приклад пояснення класифікації зображень, Рисунок 2.11 – Приклад позитивної і негативної активації нейрона, а також генерації зображення з шуму.

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
Основна частина	доц. Вітько О.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Пошук, підготовка, аналіз матеріалів	01.09-31.10	виконано
2	Аналіз предметної області та постановка задачі	20.09-15.10	виконано
3	Аналіз підходів для пояснення роботи ШНМ	10.10-05.11	виконано
4	Опис використаних методів представлення знань	01.10-15.10	виконано
5	Підготовка даних для проведення експериментів	05.10-20.10	виконано
6	Розробка системи пояснення роботи ШНМ	10.10-10.11	виконано
7	Написання пояснювальної записки	01.11-25.11	виконано
8	Перевірка на антиплагіат та нормоконтроль	25.11-03.12	виконано
9	Отримання відгуку та попередній захист	03.12-07.12	виконано
10	Захист перед ЕК	08.12	
11			
12			

Дата видачі завдання 01 вересня 20 21 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Записка пояснювальна: 93 с., 48 рис., 3 табл., 2 дод., 20 джерел.

ДОСЛІДЖЕННЯ, ЗРОЗУМІЛИЙ ІНТЕЛЕКТ, МОДЕЛІ ШТУЧНОГО ІНТЕЛЕКТУ, НАБІР ДАНИХ, НЕЙРОННА МЕРЕЖА, ПРАВИЛА, PYTHON

Об'єкт дослідження – зрозумілий штучний інтелект.

Предмет дослідження – пояснення знань в нейронних мережах.

Мета роботи – дослідити можливість пояснення прийняття рішень нейронними мережами, використовуючи різні підходи.

Методи дослідження – аналіз літературних та інтернет джерел, порівняння існуючих методів, практична реалізація.

Практично реалізоване пояснення роботи нейронної мережі за допомогою правил програмою. В якості програмної платформи для проведення практичних досліджень обрано середу Python. Пропонована розробка є корисною для вирішення задачі пояснення знань, що містяться у нейронних мережах.

РЕФЕРАТ

Пояснительная записка: 93 с., 48 рис., 3 табл., 2 прил., 20 источников.

ИССЛЕДОВАНИЯ, МОДЕЛИ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА, НАБОР ДАННЫХ, НЕЙРОННАЯ СЕТЬ, ОБЪЯСНИМЫЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, ПРАВИЛА, PYTHON

Объект исследования – объяснимый искусственный интеллект.

Предмет исследования – объяснение знаний в нейронных сетях.

Цель работы – исследовать возможность объяснения принятия решений нейронными сетями, используя различные подходы.

Методы исследования – анализ литературных и интернет источников, сравнение существующих методов, практическая реализация.

Практически реализовано объяснение работы нейронной сети с помощью правил программой. В качестве программной платформы для проведения практических исследований выбрана среда Python. Предлагаемая разработка полезна для решения задачи объяснения знаний, содержащихся в нейронных сетях.

ABSTRACT

Master thesis: 93 p., 48 fig., 3 tabl., 2 ann., 20 references.

DATA SET, EXPLAINABLE ARTIFICIAL INTELLIGENCE, MODELS OF ARTIFICIAL INTELLIGENCE, NEURAL NETWORK, PYTHON, RESEARCH, RULES

The object of research is explainable artificial intelligence.

The subject of research is the explanation of knowledge in neural networks.

The purpose of this work is to investigate the possibility of explaining decision-making by neural networks using different approaches.

Research methods – analysis of literary and Internet sources, comparison of existing methods, practical implementation.

The explanation of the operation of the neural network using the rules of the program has been practically implemented. The Python environment was chosen as a software platform for practical research. The proposed development is useful for solving the problem of explaining the knowledge contained in neural networks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШНМ – штучна нейронна мережа;

ANN – artificial neural network – штучна нейронна мережа;

DNN – deep neural network – глибока нейронна мережа;

DT – decision tree – дерево рішень;

GDPR – general data protection regulation – загальний регламент про захист даних;

ML – machine learning – машинне навчання;

OWL – ontology web language – мова веб онтологій;

RE – rule exctraction – видобування правил;

SWRL – semantic web rule language – мова правил семантичної мережі;

XAI – explainable artificial intelligence – зрозумілий штучний інтелект.

ВСТУП

Штучний інтелект (AI), зокрема його підполе машинного навчання (ML), відіграє все більш помітну роль у науці, техніці, промисловості та суспільному житті. Успіх ML в значній мірі обумовлений досягненнями в дослідженнях глибоких нейронних мереж (DNNS). Спочатку натхненні структурою та функціональністю мозку, DNNS складаються з декількох шарів нелінійних блоків, відомих як штучні нейрони. Хоча складні архітектури сучасних моделей DNN лише віддалено нагадують біологічні нейронні системи, останнім часом вони продемонстрували чудові результати у вирішенні різних завдань, які раніше сприймалися як піддаються виключно людському інтелекту, від розпізнавання мови і візуальних об'єктів з високою точністю до гри в шахи та го і водіння автономних транспортних засобів. На жаль, багато з найбільш ефективних методів ML також, як правило, є найменш прозорими.; це особливо вірно для DNN, які працюють як чорні ящики, внутрішня робота яких все ще погано вивчена. Як наслідок, системам штучного інтелекту на основі DNN поки не можна повністю довіряти при прийнятті рішень в критично важливих для безпеки або важливих додатках, таких як медицина, інтелектуальна допомога літнім людям та інвалідам, фінанси, кримінальне правосуддя або навігація безпілотних транспортних засобів в складних умовах.

Досі не було знайдено надійного вирішення проблем підробленості і упередженості для візуального розпізнавання і обробки природної мови. Потенційні етичні пастки слід усунути якомога швидше, оскільки бездіяльність може призвести до непередбачених розколів і відмінностей в майбутньому суспільстві. Європейський союз ввів право на роз'яснення в загальні правила захисту даних (GDPR) в якості спроби усунути потенційні проблеми, пов'язані зі зростаючою важливістю алгоритмів.

У відповідь на нагальну необхідність зробити системи AI більш

прозорими і, отже, більш надійними, почала з'являтися нова область досліджень, найчастіше звана зрозумілим ШІ (XAI) або інтерпретованим ШІ.

XAI може бути визначений як проблема взаємодії людини і агента, коли агент розкриває основні причини процесу прийняття рішень своїм або іншим агентом. Іншими словами, XAI вважається підмножиною сфери взаємодії людини та агента, яку можна визначити як перетин AI, соціальних наук та HCI.

Технічно, не існує стандартного і загальноприйнятого визначення зрозумілого ШІ. Насправді термін XAI, як правило, відноситься до руху, ініціатив і зусиль, що вживаються у відповідь на проблеми прозорості та довіри ШІ, а не до формальної технічної концепції. XAI прагне «створювати більш зрозумілі моделі, зберігаючи при цьому високий рівень ефективності навчання (точність прогнозування) і дозволяючи користувачам-людям правильно розуміти, довіряти і ефективно управляти новим поколінням партнерів зі штучним інтелектом» [1]. Мета забезпечення пояснюваності в ML полягає в тому, щоб гарантувати, що алгоритмічні рішення, а також будь-які дані, що визначають ці рішення, можуть бути пояснені кінцевим користувачам та іншим зацікавленим сторонам в нетехнічних термінах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

За останнє десятиліття кількість наукових статей, конференцій і симпозіумів по всьому світу в області штучного інтелекту (XAI) значно зросла. Це призвело до розробки безлічі предметно-залежних і контекстно-залежних методів для роботи з інтерпретацією моделей машинного навчання (ML) і формування пояснень для людей. За цим послідували дослідження, присвячені вивченню підходів до оцінки якості автоматично генеруються пояснень. Зростання результатів досліджень XAI за останнє десятиліття в значній мірі обумовлений швидким зростанням популярності ML і, зокрема, глибокого навчання, з великою кількістю додатків в декількох областях бізнесу, від електронної комерції до ігор, включаючи додатки в області кримінального правосуддя, охорони здоров'я, комп'ютерного зору і моделювання бойових дій, і це лише деякі з них. На жаль, більшість моделей, які були побудовані з використанням ML та глибокого навчання, були позначені вченими як «чорний ящик», оскільки їх основні структури складні, нелінійні та надзвичайно складні для інтерпретації та пояснення непрофесіоналам [1]. Ця непрозорість створила потребу в архітектурах XAI, яка мотивується головним чином трьома причинами: потреба в створенні більш прозорих моделей, необхідність в методах, що дозволяють людям взаємодіяти з ними, вимога достовірності їх висновків [1]. Крім того, як справедливо відзначали багато вчених, моделі, створені на основі даних, повинні нести відповідальність, оскільки відповідальність, ймовірно, скоро стане юридичною вимогою. Стаття 22 загального регламенту щодо захисту даних (GDPR) встановлює права та обов'язки, пов'язані з використанням автоматизованого прийняття рішень. Примітно, що він вводить право на пояснення, надаючи окремим особам

право отримати пояснення висновків, автоматично зроблених моделлю, протистояти і оскаржувати відповідну рекомендацію, особливо коли це може негативно вплинути на людину юридично, фінансово, психічно або фізично. Схваливши цю статтю GDPR, Європейський парламент спробував вирішити проблему, пов'язану з поширенням в суспільстві потенційно упереджених висновків, які обчислювальна модель могла отримати з упереджених і незбалансованих даних.

Більшість робіт в літературі належать спільнотам машинного навчання та інтелектуального аналізу даних. Перша в основному зосереджена на описі того, як працюють чорні ящики, тоді як друга більше зацікавлена у поясненні рішень, навіть не розуміючи деталей того, як працюють непрозорі системи прийняття рішень в цілому [1].

Незважаючи на те, що інтерпретоване машинне навчання було темою протягом досить довгого часу і останнім часом отримало велику увагу, сьогодні існує безліч розрізнених результатів, і систематична організація і класифікація цих методологій відсутні [2]. У літературі міститься безліч питань, в яких пропонуються методології інтерпретації систем «чорного ящика»: Що означає, що модель піддається інтерпретації або прозора? Що таке пояснення? Коли модель або пояснення зрозумілі? Який найкращий спосіб дати пояснення? Які проблеми вимагають інтерпретованих моделей прогнозів? Які дані про прийняття рішень зачіпаються? Який тип записів даних більш зрозумілий? Скільки ми готові втратити в точності прогнозування, щоб отримати будь-яку форму інтерпретованості?

Багато авторів вивчали наукові статті, що стосуються пояснюваності в рамках штучного інтелекту в конкретних піддоменах, мотивуючи необхідність організації літератури. Наприклад, деякі автори відповідно розглянули методи пояснення за допомогою нейронних і байєсівських мереж, в той час як інші об'єднали наукові матеріали, присвячені витяганню правил з машин опорних векторів (SVM). Мета полягала і в цілому полягає в тому, щоб створити правила, легко інтерпретовані людьми, зберігаючи

при цьому ступінь точності, пропоновану навченими моделями. Лише деякі вчені спробували провести більш повний огляд і систематизувати методи пояснення в цілому. Інші вчені визначили широкий набір понять і вимог, яким має відповідати пояснення, щоб його було легко зрозуміти кінцевим користувачам, зокрема непрофесіоналам, і щоб надати ефективну практичну інформацію для підтримки процесів прийняття рішень. Концептуальна основа, що лежить в основі XAI, представлена на рисунку 1.1, відповідно до якого методи пояснення будуються на основі автоматично індукованих моделей з використанням пояснювачів, і їх можна оцінити за допомогою понять і показників. На рисунку 1.2 представлена класифікація літератури та її розподіл з плином часу.

Навчання класифікатора історичним наборам даних, що повідомляють про людські рішення, може призвести до виявлення повсюдних упереджень. Більш того, оскільки ці правила можуть бути глибоко приховані в навченому класифікаторі, ми ризикуємо розглядати, можливо, несвідомо, такі практики в якості загальних правил [2].

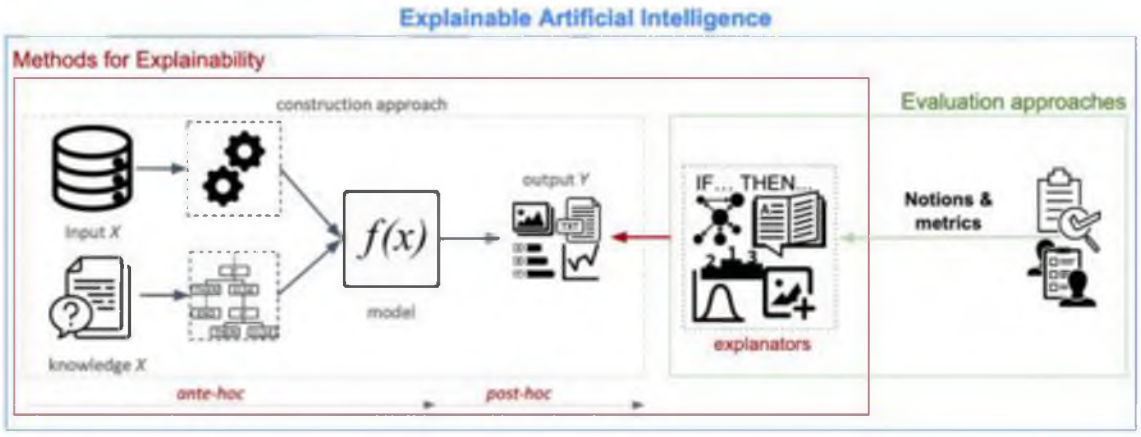


Рисунок 1.1 – Схематичне уявлення зрозумілого штучного інтелекту із взаємодією методів пояснення і підходів до їх оцінки.

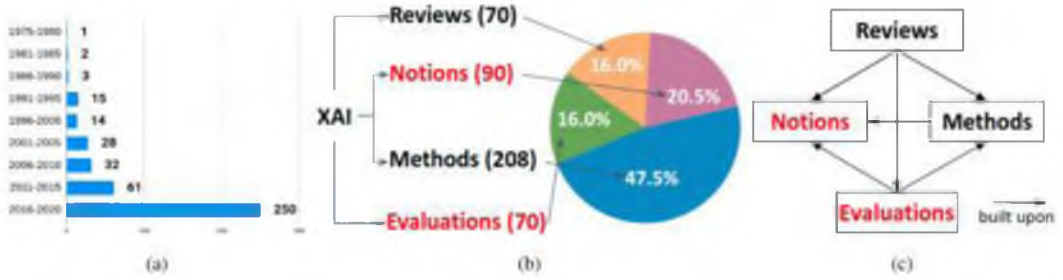


Рисунок 1.2 – Класифікація літератури XAI з розподілом опублікованих наукових статей з плином часу (a), (b) основні чотири категорії і відсоток статей в кожній, (c) характерні відносини між цими категоріями, які виникли

У [2] повідомляється про випадок, коли військові навчили класифікатор відрізняти ворожі танки від дружніх танків. Класифікатор забезпечував високу точність на тестовому наборі, але при його використанні в польових умовах мав дуже низьку продуктивність. Пізніше було виявлено, що фотографії дружніх танків були зроблені в сонячні дні, в той час як фотографії ворожих танків – в похмурі дні (рисунок 1.3).

Рисунок 1.3 – Приклад можливої неправильної класифікації військових танків в залежності від фону: сонячний (зліва) і похмурий (праворуч)

Відсутність детального розуміння того, якими знаннями володіє DNN, ускладнює повне використання їх у багатьох областях, критично важливих для безпеки, таких як навігація автономних автомобілів і медицина, де в іншому випадку вони могли б надати велику допомогу. Одним з перших і найбільш відомих прикладів упередженого ставлення до людей був

алгоритм ML Google Photo, що позначає фотографії чорношкірих людей як фотографії горил.

Автори [3] зазначають, що ще однією перешкодою для розгортання DNN у відмовостійких завдання є те, що вони можуть бути схильні не тільки до ненавмисних помилок, але і до шкідливих атак: таким чином, згорткові нейронні мережі, які повсюдно використовуються в різних задачах комп'ютерного зору, можна «обдурити», щоб неправильно класифікувати об'єкт, змінивши вхідне зображення настільки тонко, що зміни в ньому, здавалося б, не мають великого значення для спостерігача-людини. Такі ворожі атаки становлять загрозу самі по собі, а також підривають довіру осіб, які приймають рішення, до систем на основі DNN в цілому: дійсно, якщо нанесення декількох невинно виглядаючих чорно-білих наклейок на фізичний знак зупинки руху може «обдурити» сучасну DNN, класифікуючи його як знак обмеження швидкості 45, може бути досить проблематично переконати законодавців довірити безпілотний транспортний засіб, керований аналогічними моделями DNN, з повною автономією на вулицях.

Навіть у самих ранніх системах штучного інтелекту 1960-х і 1970-х років генерація пояснень була визначена як ключова проблема. Спочатку основна увага приділялася наданню користувачам механізмів для отримання слідів міркувань, виконуваних системою. Прикладом такого підходу є трасування правил, що генеруються компонентом пояснення експертної системи MYCIN [4]. Навіть на цьому ранньому етапі було усвідомлено, що існує два різних типи зацікавлених сторін, які вимагають пояснень:

– розробники системи штучного інтелекту, які потребують допомоги в налагодженні програмного забезпечення, маючи можливість перевірити правильність чи ні послідовностей спрацьовування правил, що призводять до висновку;

– користувачі системи, які прагнуть переконатися в тому, що вони можуть довіряти висновкам програмного забезпечення, перевіряючи ланцюжок міркувань, що підтверджують конкретний висновок. Обидва види зацікавлених сторін, по суті, вимагали, щоб система ШІ мала певний ступінь прозорості у своїй роботі, тобто протилежність непрозорості.

1.2 Постановка задачі

Метою даної роботи є дослідження можливості пояснення прийняття рішень нейронними мережами, використовуючи методи машинного навчання. Отримані результати повинні бути зрозумілими експерту певною предметної області, результати повинні мати можливість спільного використання та редагування при необхідності. Вхідною інформацією є дані, вихідною – придатні для сприйняття людиною знання. Для побудови подібного необхідно:

– дослідити область зрозумілого штучного інтелекту;

– дослідити можливі методи пояснення роботи чорного ящика;

– обґрунтувати та вибрати технології для реалізації системи, що дозволяла б отримувати пояснення роботи чорного ящика;

– розробити систему, що дозволяла б отримувати поясннення роботи чорного ящика;

– зробити висновки щодо виконаної роботи.

2 ЗРОЗУМІЛИЙ ШТУЧНИЙ ІНТЕЛЕКТ

2.1 Поняття зрозумілого штучного інтелекту

Пояснення моделі, створеної на основі даних з використанням певної методики навчання, не є тривіальною метою. Безліч літератури зосереджено на досягненні такої мети шляхом дослідження і спроби визначити концепцію зрозумілості, що призводить до багатьох типів пояснення і формування декількох атрибутів і структур [1]. Їх можна поділити на декілька кластерів:

– атрибути зрозумілості – критерії та характеристики, які використовуються вченими, щоб спробувати визначити конструкцію «пояснення»;

– теоретичні підходи до структурування пояснень – включає в себе різні способи, якими вчені повідомляють про пояснення для своїх спеціальних додатків, які частини інформації включені або опущені, і різні компоненти, на яких може бути побудовано пояснення, такі як причини, контекст і наслідки передбачення моделі, а також їх упорядкування.

На рисунку 2.1 зображений розподіл наукових праць за даними категоріями.

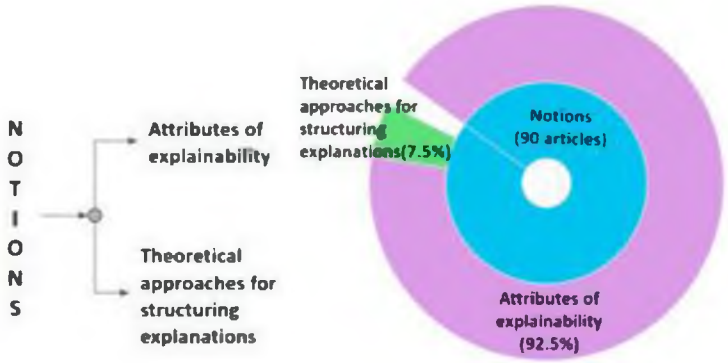


Рисунок 2.1 – Розподіл наукових прац, що визначають концепцію зрозумілості

Одна з головних причин для створення пояснення полягає в тому, щоб завоювати довіру користувачів. У [1] сказано, що довіра – це основний спосіб підвищити довіру користувачів до системи і зробити так, щоб вони відчували себе комфортно, контролюючи і використовуючи її. Крім довіри, дослідники визначили інші позитивні ефекти, викликані поясненням. Це частина людської природи – привласнювати подіям причинно-наслідкову атрибуцію. Система, яка дає причинно-наслідкове пояснення своєму процесу виведення, сприймається кінцевими користувачами більш по-людськи, як наслідок вродженої схильності людської психології до антропоморфізму. Таким чином, кілька вчених детально говорили про причинність, яка вважається фундаментальним атрибутом зрозумілості. Пояснення повинні робити причинно-наслідкові зв'язки між вхідними даними і прогнозами моделі явними, особливо коли ці зв'язки не очевидні для кінцевих користувачів. Як сказано у [1], моделі, засновані на даних, призначені для виявлення та використання асоціацій у даних, але вони не можуть гарантувати, що в цих асоціаціях існує причинно-наслідковий зв'язок. Були запропоновані чотири причини, що підтверджують необхідність пояснення логіки системи виведення або алгоритму навчання:

– пояснення для виправдовування – рішення, прийняті з використанням базової моделі, повинні бути пояснені, щоб підвищити їх обгрунтованість.;

– пояснення для контролю – пояснення повинні підвищити прозорість моделі та її функціонування, дозволяючи налагоджувати її і виявляти потенційні недоліки;

– пояснювати, щоб поліпшити – пояснення повинні допомогти вченим підвищити точність і ефективність своїх моделей;

– пояснювати, щоб виявити – пояснення повинні сприяти отриманню нових знань і вивченню взаємозв'язків і закономірностей.

Зрозумілість як концепція використовується в декількох областях, починаючи від математики, фізики, інформатики і закінчуючи інженерією, психологією, медициною і соціальними науками. Пояснимість часто замінюється поняттям інтерпретованості, що розглядається як синоніми в загальному співтоваристві ШІ, і зокрема тими вченими, які займаються автоматизованим навчанням і виводом, в той час як спільнота розробників програмного забезпечення віддає перевагу терміну зрозумілість [1]. Інтерпретованість часто визначається як здатність надавати або розкривати значення абстрактної концепції, а зрозумілість – як здатність зробити модель зрозумілою кінцевим користувачам. Однак в літературі пропонуються й інші визначення. Зрозумілість або інтерпретованість визначається як «ступінь, в якій спостерігач-людина може зрозуміти причину рішення (або прогнозу), прийнятого моделлю» [1]. Існує широкий набір характеристик, якими повинно володіти пояснення: алгоритмічна прозорість, дієвість, причинність, повнота, зрозумілість, когнітивне полегшення, коректованість, ефективність, ясність, достовірність, інтерактивність, цікавість, інтерпретованість, інформативність, обґрунтованість, психічна відповідність, монотонність, переконливість, передбачуваність, уточнення, оборотність, надійність, задоволеність, ретельність, діагностика, безпека, простота, чутливість, спрощення, обґрунтованість, стабільність, прозорість, переносимість.

Метод пояснення повинен відповідати на кілька запитань, щоб сформувати вичерпне пояснення. Два найбільш поширених питання полягають у тому, чому і як досліджувана модель виробляє свої прогнози або висновки. Різні моделі поведінки, різні проблеми і різні типи користувачів вимагають різних пояснень, як показано в діаграмі на рисунку 2.2 Це призвело до появи безлічі спеціальних класифікацій, які залежать від предметної області і які важко об'єднати в одну.

Рисунок 2.2 – Діаграма основних факторів, що формують структуру машинно-генерованого пояснення

2.2 Оцінка методів пояснення

Розвиток багатьох методів пояснення призвів вчених до того, що вони також зосередилися на їх оцінці. Були запропоновані різні показники оцінки, а також проведено різні види оцінювання, як показано на рисунку 2.3. Ці показники не обмежуються оцінкою зрозумілості, але вони також враховують інші аспекти пояснень, такі як доступність.

Рисунок 2.3 – Класифікація підходів до оцінки методів пояснення та розподіл відносних наукових досліджень за категоріями

Два основних способи оцінки методів пояснення:

– об'єктивні оцінки – вона включає в себе дослідження, в яких використовувалися об'єктивні показники і автоматизовані підходи для оцінки методів пояснення;

– оцінки, орієнтовані на людину – в ньому містяться ті дослідження, в яких оцінювалися методи пояснення з використанням підходу «людина в циклі» із залученням кінцевих користувачів і використанням їх відгуків або обґрунтованих суджень.

У науковій літературі існує консенсус щодо того, що простіші методи навчання, такі як лінійна регресія та дерева рішень, можуть призвести до більш прозорих висновків, ніж більш складні методи, такі як нейронна мережа, оскільки вони за своєю суттю самоінтерпретовані. Однак ці простіші методи зазвичай не призводять до побудови моделей з тим же рівнем точності, що і моделі, створені за допомогою більш складних методів навчання [1]. Інтерпретованість цих моделей залежить від багатьох факторів, таких як алгоритм навчання, архітектура навчання та її конфігурація (гіперпараметри). Але також існує думка, що прозора модель може бути здатна виявити ті ж самі закономірності. Якщо шаблон у даних був достатньо важливим, щоб модель чорного ящика могла використовувати його для отримання кращих прогнозів, інтерпретована модель також могла б знайти той же шаблон і використовувати його [5].

У цій галузі були представлені деякі кількісні показники оцінки для оцінки інтерпретованості методів, що генерують візуальні пояснення нейронної мережі, навченої класифікації зображень. Одна з метрик, нестабільність розташування, перевіряє, чи CNN знаходить відповідні частини одного і того ж об'єкта, показані на різних зображеннях, на майже постійній відносній відстані, оскільки відстані між частинами об'єкта повинні бути майже незмінними [1]. Ще одна метрика – чутливість, яка вимірює ступінь, в якій на візуальне пояснення впливають незначні обурення у вхідних примірниках. Ступінь кореляції між картами значущості

може бути виміряна шляхом обчислення коефіцієнтів рангової кореляції Спірмена [1].

Існують показники розрідженості, які пов'язані із загальною кількістю використовуваних ознак. Для дерев рішень розрідженість вимірює загальну кількість об'єктів, що використовуються в якості об'єктів поділу [6]. Для моделей, заснованих на правилах, розмір вимірює загальну кількість правил в наборі рішень або списку рішень. Довжина правила вимірює загальну кількість предикатів, що використовуються в умові набору рішень або списку рішень. Довжина кожного окремого правила може бути накопичена для вимірювання загальної кількості використовуваних предикатів. Крім того, для вимірювання складності можна використовувати показники для загального числа примірників даних, що задовольняють правилу (покриття), і показники для загального числа примірників даних, що задовольняють декільком правилам (перекриття) [6].

Ряд дослідників провели формальні порівняння, засновані на евристичних показниках, між різними методами пояснення, щоб оцінити їх сильні і слабкі сторони. Методології, що використовуються для порівнянь:

– чутливість до вхідних збурень – деякі функції вхідного набору даних видаляються, маскуються або змінюються, і порівнюються пояснення, згенеровані методом пояснення з моделі, навченої як на вихідних, так і на змінених вхідних даних;

– чутливість до рандомізації параметрів моделі – порівнюються результати методу пояснення, Згенерованого на основі навченої моделі, та іншої моделі тієї ж архітектури з деякими або всіма параметрами, замінениими випадковими значеннями;

– повнота пояснення – ці підходи перевіряють, який метод генерує пояснення, які найбільшою мірою описують процес виведення базової моделі. Це полягає в тому, щоб захопити найбільшу кількість функцій вхідних даних, які впливають на процес прийняття рішень в моделі.

У переважній більшості наукових статей порівнювалися методи пояснення, призначені для створення візуальних пояснень логіки, за якою слідують нейронні мережі для класифікації зображень або текстів [1]. Всі методи створюють карти, такі як теплові карти або карти об'єктів, і порівняння виконується шляхом вимірювання відмінностей в цих картах, створених до і після зміни вхідних даних або параметрів моделі. Карти значущості, створені різними методами для візуальної зрозумілості або випадково ініціалізованої непідготовленої мережі, або з копії набору даних, в якому мітки були випадковим чином переставлені.

Пояснення ефективні, коли вони допомагають кінцевим користувачам побудувати повне і правильне уявне уявлення процесу виведення даної моделі. Багато наукових статей були присвячені перевірці ступеня зрозумілості одного або декількох методів з використанням підходу «людина в циклі» [1]. У цих експериментах брали участь люди двох видів. З одного боку, люди, випадково вибрані з непрофесіоналів і без будь-яких попередніх технічних або предметних знань, яких попросили взаємодіяти з одним або декількома пояснювальними інструментами і давати відгуки, заповнюючи анкети. З іншого боку, експерти з предметної області, яких попросили дати обґрунтовані думки про пояснення, отримані за допомогою цих методів, і перевірити відповідність пояснень знанням предметної області. Наукові статті можна розділити на дві категорії, в залежності від характеру питань, що задаються людям [1]. Якісні дослідження засновані на відкритих питаннях, спрямованих на досягнення більш глибокого розуміння, в той час як кількісні дослідження використовують закриті питання, які можна легко проаналізувати статистично. Сучасний стан і передбачувані рамки для зрозумілого штучного інтелекту зображені на рисунку 2.4.

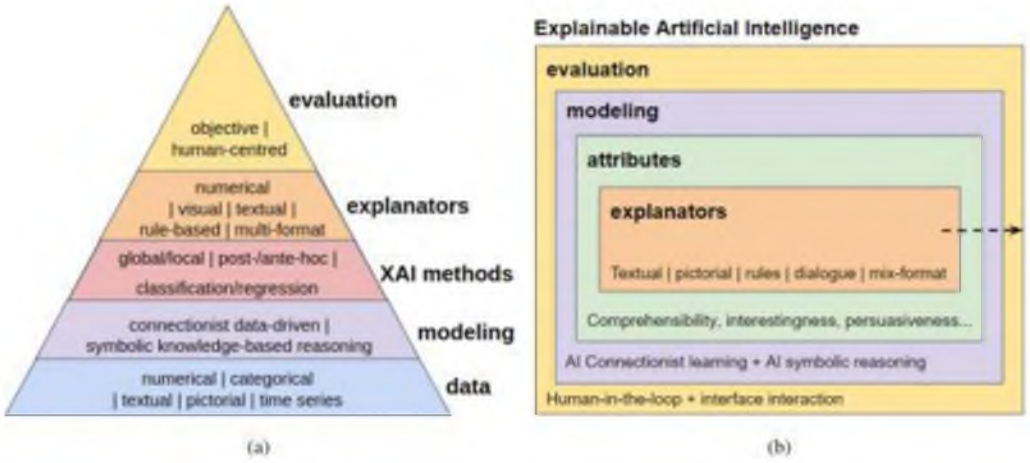


Рисунок 2.4 – Сучасний стан і передбачувані рамки для зрозумілого штучного інтелекту

2.3 Визнані інтерпретовані моделі

На даний момент визнається невеликий набір існуючих інтерпретованих моделей: дерево рішень, правила, лінійні моделі, як зазначається у [2]. Дані моделі є легко зрозумілими та інтерпретованими для людей. Заснована на дереві рішень система прийняття рішень використовує графік, структурований як дерево і складається з внутрішніх вузлів, що представляють тести на функції або атрибути (наприклад, чи має змінна значення менше, дорівнює або більше порогового значення) і кінцевих вузлів, що представляють мітку класу (рисунок 2.5). Кожна гілка представляє собою можливий результат. Правила класифікації представляють собою шляхи від кореня до листя. Дійсно, дерево рішень може бути перетворене в набір правил прийняття рішень у формі «якщо – то»: якщо умова 1 ∧ умова 2 ∧ умова 3, то результат. Тут результат відповідає мітці класу кінцевого вузла, в той час як з'єднання умов в реченні if відповідають різним умовам на шляху від кореневого вузла до цього кінцевого вузла.

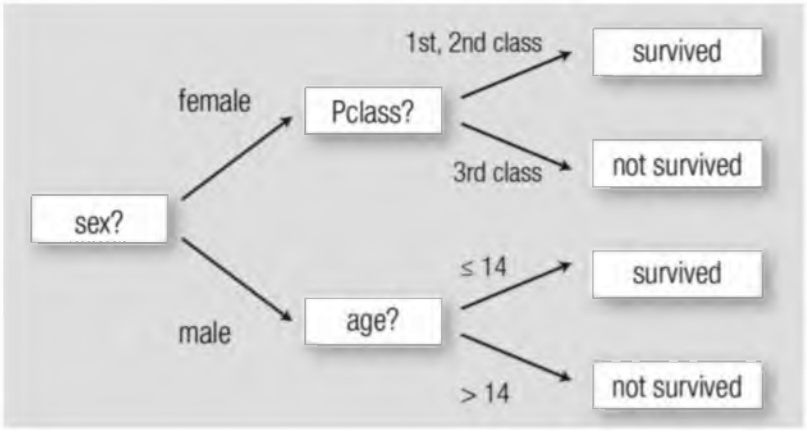


Рисунок 2.5 – Приклад класифікатора дерева рішень

У більш загальному плані правило прийняття рішень – це функція, яка зіставляє спостереження з відповідною дією. Шляхом генерації так званих правил класифікації або асоціації правила прийняття рішень можуть бути витягнуті, тобто правил, які в результаті мають мітку класу [2]. Найбільш поширеними правилами є правила if-then, де пропозиція if представляє собою комбінацію умов для вхідних змінних. Зокрема, воно може бути утворене кон'юнкціями, запереченнями і диз'юнкціями. Однак методи видобування правил зазвичай враховують тільки правила з кон'юнкціями. Іншими типами правил є: m-з-n правил, де заданий набір m з n умов, якщо m з них підтверджені, то наслідок правила вважається істинним; список правил, де заданий упорядкований набір правил вважається істинним, наслідком першого покритого правила; випадаючі списки правил складаються зі списку правил, упорядкованих за ймовірністю певного результату, і порядок визначає приклад, який повинен бути класифікований за цим правилом; набори рішень, в яких передбачено невпорядкований набір правил класифікації, так що правила не пов'язані твердженнями else, але кожне правило є незалежним класифікатором, який може привласнювати свою мітку без урахування будь-яких інших правил [2].

Інтерпретація правил і рішень різна щодо різних аспектів. Для їх графічного представлення широко використовуються дерева рішень, в той

час як правила мають текстове представлення. Основна відмінність полягає в тому, що текстове представлення не надає розуміння про більш важливі атрибути правила. Однак такого роду ключ до розгадки дає ієрархічне розташування об'єктів в дереві. Відносна важливість атрибутів може бути додана до правил за допомогою позиційної інформації. Зокрема, умови правила відображаються в порядку, в якому алгоритм вилучення правила додав їх в правило. Незважаючи на те, що представлення правил викликає деякі труднощі в розумінні моделі цілком, воно дозволяє вивчати окремі правила, що представляють частини всього знання («локальні шаблони»), які є складовими [2]. Аналіз кожного шляху окремо від кінцевого вузла до кореневого у дереві рішень дозволяє користувачам зосередитися на таких локальних шаблонах. Однак, якщо дерево дуже глибоке, в цьому випадку це набагато більш складне завдання. Ще одна важлива відмінність між правилами і деревами рішень полягає в тому, що в дереві рішень кожен запис класифікується тільки одним кінцевим вузлом, тобто передбачений клас представлений взаємовиключним і вичерпним чином набором листя і їх шляхів до кореневого вузла. Однак певний запис може задовольняти попереднім правилам, що мають в якості слідства інший клас для цього запису. Дійсно, класифікатори, засновані на правилах, мають той недолік, що вимагають додаткового підходу для вирішення таких ситуацій з суперечливим результатом [2]. Повертаючи упорядкований список правил замість невпорядкованого набору правил деякі класифікаторів на основі правил вирішують цю проблему. Отже, повертається результат, відповідний першому правилу, відповідного тестового запису і ігнорує інші правила в списку. Упорядковані списки правил може бути складніше інтерпретувати, ніж класичні правила. Фактично, в цій моделі одне правило не може розглядатися незалежно від попередніх правил у списку. Один з широко використовуваних підходів полягає в розгляді кращих k правил, що задовольняють тестовий запис, де порядок задається певною вагою

(наприклад, точністю Лапласа). Потім як передбачений клас повертається результат правил із середньою найбільшою вагою серед кращих k.

Іншим набором підходів, прийнятих для пояснення, є лінійні моделі. Оцінити важливість певних ознак можна, розглянувши і візуалізувавши важливість ознак, тобто як знака, так і величину вкладу атрибутів для даного прогнозу (рисунок 2.6).

Рисунок 2.6 – Приклад лінійної моделі та значення повернених ознак

Лінійні моделі складаються з вхідних об'єктів і ваг для кожного з них. Моделі будуються, наприклад, за допомогою логістичної регресії таким чином, щоб привласнити вагу кожній ознаці, використовуваної моделлю. Присвоєна вага вказує на внесок функції в прогноз. Лінійні моделі також легко приймаються для отримання безперервного значення замість дискретної мітки класу; отже, вони корисні для регресії.

2.4 Відкриття чорного ящика

Ми можемо розрізняти зворотній інжиніринг та прозорий дизайн на дуже високому рівні. У першому випадку, враховуючи записи рішень, створені чорним ящиком, який приймає рішення, проблема полягає у відновленні пояснення для них [2]. Як правило вихідний набір даних, на основі якого навчається чорний ящик, невідомий в реальному житті. Для

побудови пояснень у більшості наукових робіт використовується зворотне проектування. У другому випадку, при наявності набору даних записів для навчання чорного ящика, завдання полягає в розробці інтерпретованої моделі предиктора разом з її поясненнями. У [2] першу категорію також розділяють на три різні проблеми: пояснення моделі, пояснення результату і перевірка моделі.

Під час навчання під наглядом навчальний набір даних D_{train} використовується для навчання предиктора b, а тестовий набір даних D_{test} використовується для оцінки продуктивності b. У проблемах пояснень чорного ящика, D_{train} зазвичай невідомий. Враховуючи $D_{test} = \{X, \hat{Y}\}$, оцінка полягає в спостереженні для кожної пари записів даних і цільового значення $\{x, \hat{y}\} \in D_{test}$ збігів між $\hat{y}$ і $b(x) = y$. Точність – це відсоток збігів за розміром тестового набору даних. Прогностичні характеристики як чорного ящика b, так і інтерпретованого предиктора c можуть бути оцінені за допомогою вимірювання точності по тестовому набору даних [2]. Для інтерпретованого предиктора c другою мірою є вірність (fidelity). Вона оцінює, наскільки добре c імітує предиктор чорного ящика b в тестовому наборі даних. Формально вірність – це відсоток збігів $c(x) = b(x)$ для $\{x, \hat{y}\} \in D_{test}$ за розміром тестового набору даних. Показник вірності може бути інтерпретований як точність інтерпретованого предиктора c по відношенню до прогнозів чорного ящика b [2]. Точність і вірність вимірювань можуть бути легко розширені до більш досконалих, таких як precision, recall, F1-score.

У [2] сказано, що проблема пояснення чорного ящика полягає в наданні глобального пояснення моделі чорного ящика за допомогою інтерпретованої і прозорої моделі. Дана модель повинна бути одночасно здатна імітувати поведінку чорного ящика. Вона також повинна бути зрозуміла людям. Іншими словами, інтерпретована модель, що наближається до чорного ящика, повинна бути глобально інтерпретованою.

Проблема пояснення результату полягає в тому, щоб надати пояснення результату чорного ящика для екземпляру, враховуючи чорний ящик і вхідний екземпляр. Не потрібно пояснювати всю логіку, що лежить в основі чорного ящика, а тільки причину передбачення для конкретного вхідного примірника. Різниця між цими термінами показана на рисунку 2.7. Простим прикладом глобальної моделі є навчання дерева рішень на передбаченнях моделі чорного ящика.

Для розуміння деяких специфічних властивостей моделі чорного ящика або її передбачень існує проблема перевірки моделі, вона полягає в наданні представлення (візуального або текстового). Приклади властивостей, що представляють інтерес, включають чутливість до змін атрибутів та ідентифікацію компонентів чорного ящика (наприклад, нейронів у DNN), відповідальних за конкретні рішення.

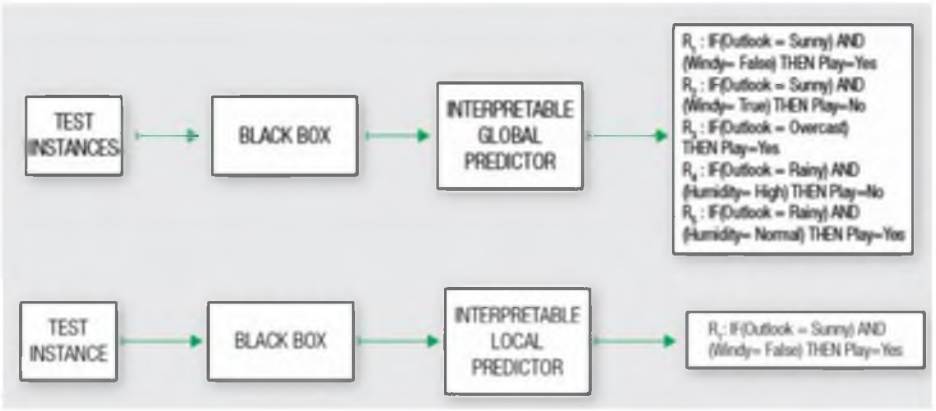


Рисунок 2.7 – Приклад глобально та локально інтерпретованої моделі

У [6] підхід зворотного інжинірингу класифікується як узагальнюваний чи ні (або педагогічний або декомпозиційний). Підхід є узагальнюваним, коли виконується чисто процедура зворотного проектування, тобто чорний ящик запитується тільки з різними вхідними записами щоб отримати оракул, який використовується для вивчення зрозумілого предиктора (рисунок 2.8). Іншими словами, внутрішні

особливості чорного ящика не використовуються для побудови зрозумілого предиктора. Таким чином, якщо підхід є узагальненим, навіть якщо він представлений для пояснення конкретного типу чорного ящика, насправді його можна використовувати для інтерпретації будь-якого типу предиктора чорного ящика. Тобто це агностичний підхід до інтерпретації чорних ящиків. Якщо підхід можна використовувати для відкриття тільки конкретного типу чорного ящика, для якого він був розроблений, то він не може бути узагальнений. Наприклад, якщо підхід призначений для інтерпретації random forest і внутрішньо використовує концепцію відстані між деревами, то такий підхід не можна використовувати для пояснення прогнозів NN [2]. Не узагальнюваний підхід не може бути агностичним щодо чорного ящика.

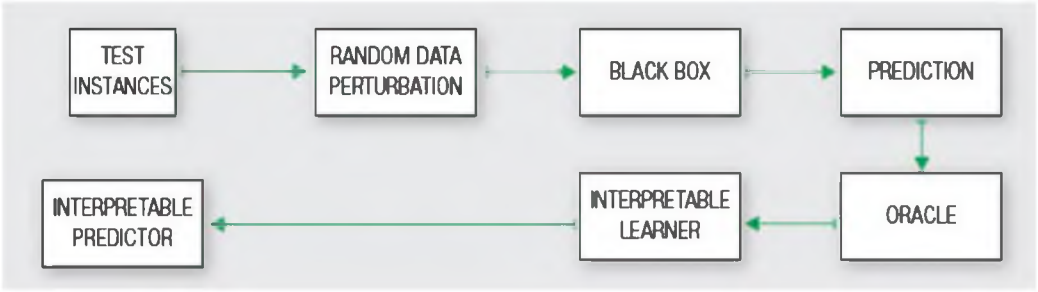


Рисунок 2.8 – Узагальнюваний підхід до зворотного проектування

Загалом усі методи пояснення можуть бути класифіковані за деякими ознаками:

– тип проблеми, що виникла;

– тип пояснювача, який використовується для відкриття чорного ящика;

– тип моделі чорного ящика, яку може відкрити пояснювач;

– тип даних, що використовуються в якості вхідних даних моделлю чорного ящика.

Типи даних, що використовуються в якості вхідних даних, такі: табличні, зображення, текст. Згідно до [7] можлива класифікація методів пояснення зображена на рисунку 2.9.

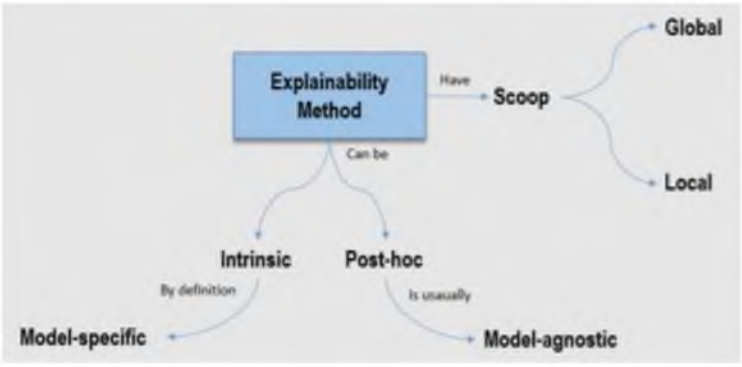


Рисунок 2.9 – Одна з можливих класифікацій методів пояснення

2.5 Основні підходи до пояснення роботи чорного ящика

2.5.1 Пояснення для зображень

Методи атрибуції пікселів виділяють пікселі, які були релевантні для певної класифікації зображень за допомогою нейронної мережі. Зображення на рисунку 2.10 є прикладом такого пояснення.

Методи атрибуції пікселів можна знайти під різними назвами: карта чутливості, карта значущості, карта атрибуції пікселів, методи атрибуції на основі градієнта, релевантність об'єктів, атрибуція об'єктів і внесок об'єктів [8].

Існує два різних типи методів:

– засновані на збуреннях: такі методи, як SHAP і LIME, маніпулюють частинами зображення для створення пояснень;

– засновані на градієнті: багато методів обчислюють градієнт прогнозу (або оцінки класифікації) по відношенню до вхідних об'єктів.

Методи, засновані на градієнті (яких існує безліч), в основному відрізняються тим, як обчислюється градієнт.

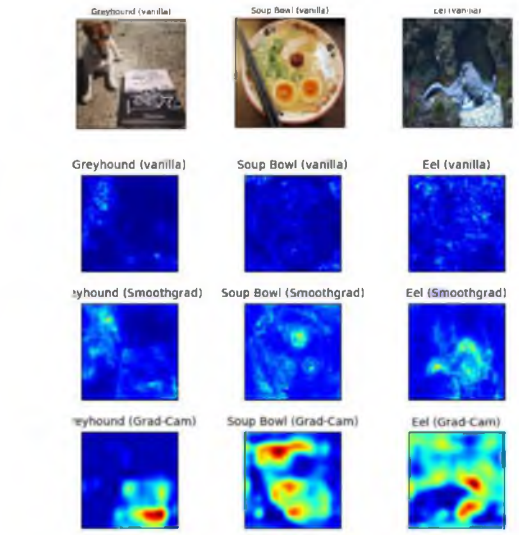


Рисунок 2.10 – Приклад пояснення класифікації зображень

Чим більше абсолютне значення градієнта, тим сильніше ефект зміни в цьому пікселі. Ідея градієнта полягає в тому, щоб обчислити градієнт функції втрат для класу, що нас цікавить, по відношенню до вхідних пікселів. Це дає нам карту розмірів вхідних об'єктів з негативними і позитивними значеннями. Класичний градієнт має проблему насичення при використанні ReLU, і коли активація опускається нижче нуля, активація обмежується нулем і більше не змінюється [8].

Пояснення наочні, і ми швидко розпізнаємо зображення. Зокрема, коли методи виділяють тільки важливі пікселі, легко відразу розпізнати важливі області зображення. Є багато методів на вибір. Швидко обчислюється.

Як і у випадку більшості методів інтерпретації, важко визначити, чи є пояснення правильним. Методи атрибуції пікселів можуть бути дуже крихкими. Методи атрибуції пікселів можуть бути вкрай

ненадійними.Методи значущості можуть бути нечутливі до моделі та даних [8].

Згорткові нейронні мережі вивчають абстрактні властивості та концепції з пікселів необробленого зображення. Feature Visualization візуалізує вивчені ознаки шляхом максимізації активації. Feature Visualization для одиниці нейронної мережі виконується шляхом пошуку вхідних даних, які максимізують активацію цієї одиниці [8]. Канали (іноді називають картами активації) як одиниці вимірювання є гарним вибором для візуалізації об'єктів. Приклад активації нейрона зображений на рисунку 2.11. У математичних термінах – це завдання оптимізації. Нове зображення, яке максимізує (середню) активацію одиниці, тут одного нейрона:

$$img *= argmax\, h_{n,x,y,z}(img). \tag{2.1}$$

Функція h – це активація нейрона, img – вхід мережі (зображення), x і y описують просторове положення нейрона, n визначає шар, а z – індекс каналу.

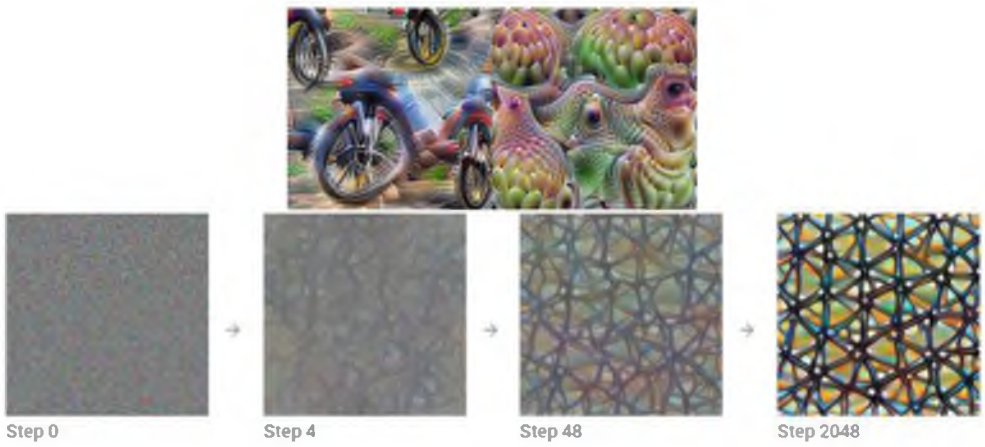


Рисунок 2.11 – Приклад позитивної і негативної активації нейрона, а також генерації зображення з шуму

Ще один підхід полягає у створенні нових зображень, починаючи з випадкового шуму. Автори [8] зазначають, що для отримання значущих візуалізацій зазвичай існують обмеження на зображення, наприклад, допускаються тільки невеликі зміни.

Feature Visualization дає унікальне уявлення про роботу нейронних мереж. Це чудовий інструмент для нетехнічного спілкування про те, як працюють нейронні мережі. Багато отриманиї зображень взагалі не піддаються інтерпретації, але містять деякі абстрактні функції, для яких у нас немає слів або ментальної концепції [8]. Занадто багато об'єктів, на які потрібно дивитися, навіть коли тільки візуалізується активація каналу. Цей підхід можливо лише створює ілюзію того, що ми розуміємо, що робить нейронна мережа.

2.5.2 Графік часткової залежності

Часткова залежність показує, як конкретна ознака впливає на прогноз. Роблячи всі інші ознаки постійними, ми хочемо з'ясувати, як ця функція впливає на наш результат. Графік часткової залежності (PDP) може показати, чи є зв'язок між ціллю та об'єктом лінійним, монотонним або складнішим [9]. Наприклад, при застосуванні до моделі лінійної регресії графіки часткової залежності завжди показують лінійну залежність. Щоб створити PDP, ми починаємо зі зміни значення однієї ознаки, зберігаючи інші постійними. Потім ми будуємо результуючі прогнози для кожного значення ознаки. Цей процес повторюється для кожного рядка в наборі даних. Для створення PDP розраховується середнє прогнозоване значення. Приклад PDP зображений на рисунку 2.12.

Обчислення графіків часткової залежності інтуїтивно зрозуміло. Функція часткової залежності при певному значенні ознаки являє собою середнє передбачення [10]. Графіки часткової залежності прості в реалізації. У некорельованому випадку інтерпретація зрозуміла: графік часткової

залежності показує, як змінюється середнє передбачення у наборі даних при зміні j-го об'єкта.

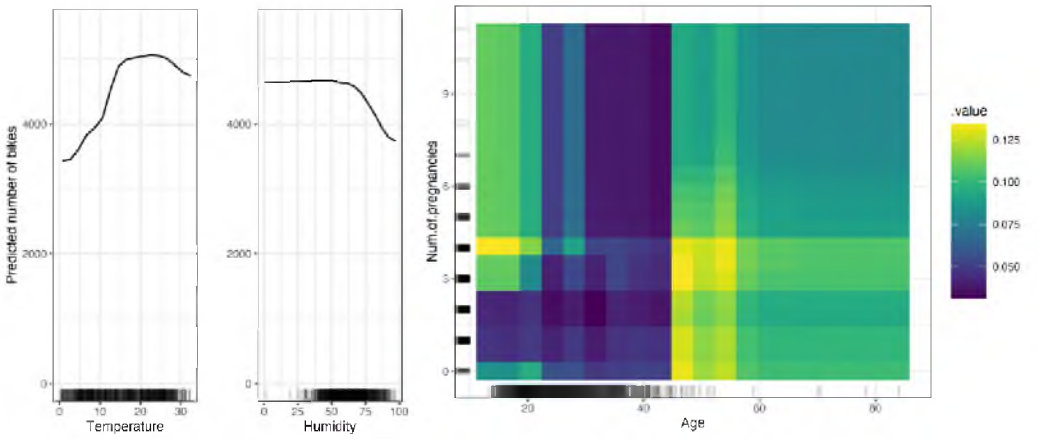


Рисунок 2.12 – Приклад графіків часткової залежності

Розрахунок для графіків часткової залежності має причинно-наслідкову інтерпретацію. У [9] стверджується, що зв'язок причинно-наслідковий для моделі, але не обов'язково для реального світу. Припущення про незалежність є найбільшою проблемою PDP. Передбачається, що ознаки, для яких обчислюється часткова залежність, не корелюють з іншими ознаками. Неоднорідні ефекти можуть бути приховані, оскільки графіки PD показують лише середні граничні ефекти.

2.5.3 Локальні сурогатні моделі

Локальні сурогатні моделі – це інтерпретовані моделі, які використовуються для пояснення окремих прогнозів моделей машинного навчання чорного ящика. Сурогатні моделі навчаються для апроксимації прогнозів базової моделі чорного ящика. Замість навчання глобальної сурогатної моделі LIME фокусується на навчанні локальних сурогатних моделей для пояснення індивідуальних прогнозів.

LIME перевіряє, що відбувається з прогнозами, коли в модель машинного навчання вводяться зміни даних. LIME генерує новий набір даних, що складається зі збурених вибірок і відповідних прогнозів моделі чорного ящика. LIME потім навчає на цьому новому наборі даних інтерпретовану модель, яка зважується по близькості обраних екземплярів до екземпляру інтересу [8]. Інтерпретована модель може бути чим завгодно з інтерпретованих моделей, наприклад ласо або дерево рішень. Вивчена модель не обов'язково повинна бути хорошою глобальною апроксимацією, вона повинна бути хорошою апроксимацією прогнозів моделі машинного навчання локально. Цей вид точності також називається локальною точністю.

Математично локальні сурогатні моделі з обмеженням інтерпретованості можуть бути виражені наступним чином:

$$explanation(x) = arg \min_{g \in G} L(f, g, \pi) + \Omega(g). \quad (2.2)$$

Модель пояснення, наприклад, x – це модель g (наприклад, модель лінійної регресії), яка мінімізує втрати L (наприклад, середньоквадратичну помилку), яка вимірює, наскільки близько пояснення до прогнозу вихідної моделі f (наприклад, модель xgboost), в той час як складність моделі Ω(g) залишається низькою (наприклад віддавати перевагу меншій кількості ознак). G – це сімейство можливих пояснень, наприклад, всі можливі моделі лінійної регресії. Міра близькості визначає, наскільки велика околиця навколо екземпляра x, яку ми розглядаємо для пояснення. На практиці LIME оптимізує тільки частину, що стосується функції втрат. Користувач повинен визначити складність, наприклад, вибравши максимальну кількість функцій, які може використовувати модель лінійної регресії.

Згідно до [8] кроки, необхідні для навчання місцевих сурогатних моделей: обрати екземпляр, для якого необхідно отримати пояснення його

передбачення чорним ящиком, змінити свій набір даних і отримати прогнози чорного ящика для цих нових точок, зважити нові зразки відповідно до їх близькості до екземпляру інтересу. Навчити зважену, інтерпретовану модель набору даних з урахуванням змін. Пояснити прогноз, інтерпретуючи локальну модель.

LIME – один з небагатьох методів, який працює з табличними даними, текстом і зображеннями, є незалежним від чорного ящика методом. При використанні ласо або коротких дерев отримані пояснення будуть короткими і, можливо, контрастними. Тому вони дають зрозумілі для людини пояснення. Визначення околиці, в якій потрібно розглядати екземпляри – дуже велика, невирішена проблема при використанні LIME з табличними даними. Складність моделі пояснення повинна бути визначена заздалегідь. В кінцевому підсумку користувачеві завжди доводиться визначати компроміс між точністю та розрідженістю [8]. Ще одна дійсно велика проблема – нестабільність пояснень. Пояснення двох дуже близьких точок можуть сильно відрізнятися в модельованій обстановці.

2.5.4 Важливість ознак

Важливість ознак відноситься до методів, які присвоюють оцінку вхідним ознакам на основі того, наскільки вони корисні для прогнозування цільової змінної.

Існує безліч типів і джерел оцінок важливості ознак, хоча популярні приклади включають оцінки статистичної кореляції, коефіцієнти, розраховані як частина лінійних моделей, дерева рішень і оцінки важливості перестановок [11].

Оцінки важливості ознак відіграють важливу роль у проекті прогностичного моделювання, включаючи уявлення про дані, розуміння моделі та основу для зменшення розмірності та вибору функцій, які можуть

підвищити ефективність та результативність прогностичної моделі для вирішення проблеми.

Оцінки корисні і можуть бути використані в ряді ситуацій в задачі прогностичного моделювання, таких як:

- краще розуміння даних;
- краще розуміння моделі;
- скорочення кількості вхідних функцій.

Алгоритми лінійного машинного навчання відповідають моделі, в якій прогноз являє собою зважену суму вхідних значень. Приклади включають лінійну регресію, логістичну регресію та розширення, які додають регуляризацію, такі як ridge regression та elastic net. Всі ці алгоритми знаходять набір коефіцієнтів для використання у зваженій сумі, щоб зробити прогноз. Ці коефіцієнти можуть бути використані безпосередньо в якості грубого типу оцінки важливості ознак.

Алгоритми дерева рішень, такі як дерева класифікації та регресії (CART), пропонують оцінки важливості, засновані на зменшенні критерію, використовуваного для вибору точок поділу, таких як Gini або ентропія [11]. Цей же підхід може бути використаний для ансамблів дерев рішень, таких як алгоритми випадкового лісу і стохастичного градієнтного бустингу.

Важливість ознак перестановки – це метод розрахунку показників відносної важливості, який не залежить від використовуваної моделі. Спочатку модель тренується на наборі даних, наприклад модель, яка не підтримує власні оцінки важливості об'єктів. Потім модель використовується для прогнозування набору даних, хоча значення об'єкта (стовпця) в наборі даних переставлені. Це повторюється для кожного об'єкта в наборі даних. Потім весь цей процес повторюється 3, 5, 10 або більше разів. Результатом є середній бал важливості для кожної вхідної ознаки (і розподіл балів з урахуванням повторень).

Цей підхід може бути використаний для регресії або класифікації і вимагає, щоб в якості основи оцінки важливості був обраний показник

ефективності, такий як середньоквадратична помилка для регресії і точність для класифікації [11].

2.5.5 Підходи до видобування правил

Підходи до видобування правил вивчають правила прийняття рішень або дерева рішень на основі прогнозів, зроблених чорними ящиками. Згідно до [2], вони діляться на педагогічний, декомпозиційний і еклектичний підходи. Педагогічний підхід (агностичний) сприймає базову модель прогнозування як чорний ящик і використовує наданий зв'язок між входом і виходом. Декомпозиційний підхід (специфічний для конкретної моделі) використовує внутрішню структуру базової моделі прогнозування. Еклектичний або гібридний підхід поєднує в собі декомпозиційний і педагогічний.

Загальновизнано, що дерево рішень (DT) є однією з найбільш інтерпретованих і легко зрозумілих моделей, в першу чергу для глобальних, але також і для локальних пояснень. Дійсно, дуже поширеним методом відкриття чорного ящика є так зване «наближення єдиним деревом».

Апроксимації одним деревом для NNs були вперше представлені в 1996 році. Зрозумілі уявлення нейронної мережі повертаються методом Trepan, який є реалізацією функції пояснення. Trepan запитує нейронну мережу b, щоб створити дерево рішень, що апроксимує концепції, представлені мережами, шляхом максимізації коефіцієнта gain разом з оцінкою точності поточної моделі. Ще одна перевага Trepan щодо звичайних класифікаторів дерев, таких як ID3 або C4.5, полягає в тому, що завдяки чорному ящику b він може використовувати стільки екземплярів, скільки потрібно для кожного поділу, так що також поділ вузлів поблизу нижньої частини дерева реалізується з використанням значного обсягу даних [2]. Крім того, він використовує критерій коефіцієнта gain для пошуку найкращих m-з-n розбиттів для кожного вузла. Всякий раз, коли для певного

поділу недостатньо навчальних примірників, створюються додаткові екземпляри даних для навчання [6]. Однак Trepan обмежений бінарною класифікацією.

Алгоритм ANN-DT був спочатку запропонований для індукції дерева з ANN. Однак двійкові дерева рішень можуть бути витягнуті з будь-якого чорного ящику за допомогою ANN-DT, який являє собою алгоритм, аналогічний CART. Він використовує стратегію вибірки для створення штучних навчальних даних, які отримують свої мітки з базового чорного ящика. DecText був спочатку запропонований для ANN, але він може бути застосований до будь-якої моделі чорного ящика. Загальна процедура нагадує Trepan з інновацією у вигляді чотирьох методів розбиття, спрямованих на пошук найбільш важливих ознак при побудові дерева [2]. Більш того, оскільки однією з основних цілей дерева є максимізація fidelity при збереженні «простоти» моделі, визначена стратегія обрізки, заснована на fidelity, для зменшення розміру дерева. Генерується набір випадкових екземплярів. Потім, починаючи з нижньої частини дерева, для кожного внутрішнього вузла створюється лист з міткою більшості, використовуючи маркування випадкових екземплярів. Якщо точність нового дерева перевершує старе, то максимальна точність і дерево оновлюються.

Існують методи, які використовують ANN або глибокі нейронні мережі (DNN) як основну модель прогнозування та витягують правила з ANN, але згідно [12] не всі вони можуть бути застосовані до довільних DNN. Метод Subset знаходить підмножини об'єктів, які повністю активують приховані і вихідні шари. Ці підмножини потім використовуються для вилучення правил з ANN. Алгоритм швидкого вилучення правил з нейронних мереж FERNN спрямований на ідентифікацію релевантних прихованих і вхідних нейронів. Він використовує C4.5 для побудови дерева рішень поверх ANN. Це дерево рішень є основою для етапу вилучення правил. Алгоритм Knowledgetron (KT) витягує правила для кожного

нейрона на основі вхідних нейронів, які відповідають за активацію іншого нейрона. Це основа для заключного етапу вилучення правила.

Деревоподібна структура ідеально підходить для захоплення взаємодій між об'єктами в даних. Дані в кінцевому підсумку об'єднуються в окремі групи, які часто легше зрозуміти, ніж точки на багатовимірній гіперплощині, як у лінійній регресії [8]. Деревоподібна структура також має природну візуалізацію зі своїми вузлами і ребрами. Дерева створюють хороші пояснення. Будь-яка лінійна залежність між вхідним об'єктом і результатом повинна бути апроксимована поділами, створюючи ступінчасту функцію. Це неефективно. Це йде рука об руку з відсутністю плавності. Невеликі зміни у вхідній функції можуть мати великий вплив на прогнозований результат, що зазвичай небажано. Дерева також досить нестійкі. Дерева рішень дуже легко піддаються інтерпретації – до тих пір, поки вони короткі.

Правила прийняття рішень (DR) відносяться до числа найбільш зрозумілих для людини методів. Існують різні типи правил. Вони використовуються для пояснення моделі, результату, а також для прозорого дизайну. Приклад правил рішень зображений на рисунку 2.13.

BRIANNE алгоритм було спочатку запропоновано для штучної нейронної мережі (ANN). Алгоритм вимірює релевантність певної ознаки для деякого виходу. Цей вимір потім використовується для побудови умов правила. Міра визначає ознаки та значення ознак умови. Алгоритм аналізу інтервалів достовірності (VIA) спрямований на пошук доказово правильних правил. Це робиться шляхом застосування підходу, аналогічного аналізу чутливості.

BIO-RE застосовує підхід, заснований на вибірці, який генерує всі можливі комбінації вхідних даних і запитує у чорного ящика їх прогнози. Виходячи з цього, будується таблиця істинності, поверх якої для вилучення правил може бути застосований довільний алгоритм, заснований на правилах [6]. Існує клас підходів до вилучення правил, що використовують

генетичне програмування, яке мотивоване теорією Дарвіна про виживання найбільш пристосованих. Алгоритм вилучення генетичних правил (G-Rex), наприклад, вибирає кращі правила з пулу правил, згенерованих чорним ящиком, і об'єднує їх з генетичним оператором.

```
IF (humidity = high) and (outlook = sunny)
THEN play=no
IF (outlook = rainy) and (windy = TRUE)
THEN play=no
OTHERWISE play=yes
```

Рисунок 2.13 – Приклад правил рішень

Алгоритм вилучення правил на основі статистики (STARE) заснований на пошуку в ширину. Крім того, вхідні дані змінюються, щоб створити таблицю істинності, яка використовується для вилучення правил. Алгоритм вилучення правил із ансамблю нейронних мереж (REFNE) використовує ансамблі ANN для створення екземплярів, які потім використовуються для побудови правил прийняття рішень. Iter – це алгоритм послідовного покриття, який вивчає одне правило за раз. Він випадковим чином генерує додаткові екземпляри даних і використовує чорний ящик як оракул для позначення цих екземплярів даних [6]. Minerva – це також алгоритм послідовного покриття, який використовує ітераційне зростання для вилучення правил прийняття рішень із чорного ящика. Алгоритм вилучення правил методом зворотного проектування (RXREN) використовує зворотне проектування для обрізки вхідних об'єктів. Вхідні об'єкти видаляються, коли їх тимчасове упущення істотно не змінює вихідні дані класифікації.

Правила легко інтерпретувати. Правила прийняття рішень можуть бути такими ж виразними, як дерева рішень, але при цьому більш компактними. Прогнозування за допомогою правил «якщо – то»

виконується швидко, так як необхідно перевірити тільки кілька операторів, щоб визначити, які правила застосовуються. Вони також стійкі до викидів, оскільки має значення тільки те, чи застосовується умова чи ні. Вони вибирають тільки релевантні ознаки для моделі.

Дослідження та література за правилами «якщо – то» фокусуються на класифікації і майже повністю ігнорують регресію. Часто атрибути також повинні бути категоричними. Багато зі старих алгоритмів навчання правилам схильні до перенавчання [8]. Правила прийняття рішень погано описують лінійні взаємозв'язки між функціями і висновком. Це проблема, яку вони поділяють з деревами рішень. Дерева рішень і правила можуть створювати тільки покрокові функції прогнозування, де зміни в прогнозуванні завжди є дискретними кроками і ніколи не кривими [8].

2.6 Глибокі нейронні мережі

Моделі DNN по суті є складними нелінійними функціями f: X→Y, що відображають з вхідного простору X у вихідний простір Y. вони точно характеризуються як чорні ящики, тому що відповідна функція відображення в їхньому випадку отримується за допомогою непрозорого процесу навчання і, отже, сама по собі непрозора. Термін «чорний ящик» протилежний терміну «білий ящик» (або «скляний ящик»), який відноситься до повністю прозорої системи. У той час як прозорість моделей «білого ящика» робить їх зрозумілими (принаймні для експертів) і, отже, надійними, схожий на чорний ящик характер DNNS створює проблему (відсутність) довіри. Зокрема, не можна бути впевненим, чи робить модель DNN правильний прогноз для раніше невидимого набору даних, тому що вона вивчила основні характеристики даних або, навпаки, тому що вона просто запам'ятала деякі несуттєві особливості в навчальному наборі даних, які також зустрічаються в тестовому наборі даних [3]. У першому випадку модель може достовірно узагальнювати в реальному світі, тоді як у другому

випадку вона спирається на помилкову кореляцію і тому не може бути корисною. Як зазначають деякі вчені, «якщо користувачі не довіряють моделі або прогнозу, вони не будуть його використовувати»; щоб подолати їх нерішучість, може бути корисним правдоподібне пояснення моделі.

В даній роботі будемо розглядати класичний варіант DNN – багатошаровий персептрон (MLP) в якості чорного ящика. Вона часто застосовується до задач класифікації, досягає високої точності, є багато готових інструментів, які дозволяють тренувати такий тип ШНМ. Модель MLP являє собою штучну нейронну мережу, що відображає деякі приклади вхідних даних у цільові значення. Мережа формується шляхом складання декількох простих шарів (не менше трьох шарів). Ми можемо розглядати застосування кожного шару як надання нового представлення кожної точки даних.

Основна мета моделі MLP – апроксимувати деяку функцію $f*$. Наприклад, в моделі класифікатора $y = f*(x)$ зіставляє вхідні дані x з міткою y. MLP визначає відображення $y = f(x;\theta)$ і знаходить правильні значення параметрів θ, які призводять до найбільш близького наближення функції. У глибоких мережах прямої передачі визначення прямої передачі відноситься до ідеї про те, що вхідні дані проходять через функцію, обчислюється з x, потім проходять через проміжні обчислювальні блоки, використовувані для визначення f, і, нарешті, надходять на вихід y. Слід зазначити, що в моделі MLP немає з'єднань зворотного зв'язку для зворотного зв'язку з виходами моделі в себе.

MLP має щонайменше три шари, в яких обчислювальні блоки (або нейрони) щільно з'єднані з блоками наступного шару.

Наприклад, MLP з двома прихованими шарами може бути записаний як:

$$f(x) = b_3 + W_3 tanh(b_2 + W_2 tanh(b_1 + W_1 x)). \qquad (2.3)$$

Вагові матриці W_1, W_2, W_3 і векторнозначні зміщення b_1, b_2, b_3 параметризують MLP. Архітектура MLP визначається кількістю прихованих шарів і розмірами шарів. Схематичне зображення багатошарового персептрону наведене на рисунку 2.14.

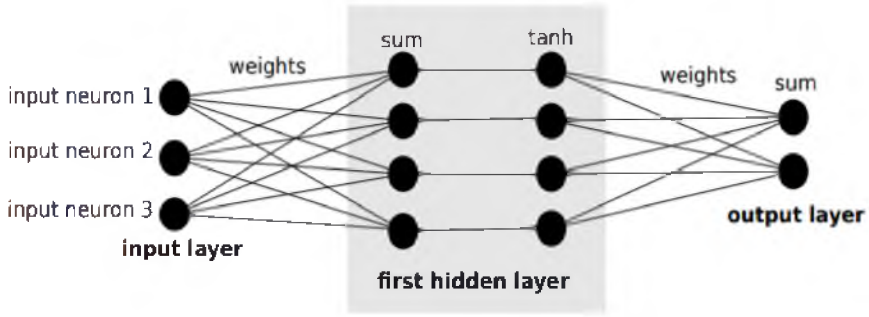


Рисунок 2.14 – Графічне представлення (3,4,2) – MLP

В данному розділі були розглянуті основні поняття зрозумілого штучного інтелекту, підходи та методи пояснення роботи штучної нейронної мережі, їх оцінка, архітектура багатошарового персептрону. Оглянуті підходи є найбільш поширеними та застосовуються для отримання уявлення про процесс прийняття рішень нейронною мережею.

3 ОГЛЯД ВИКОРИСТАНИХ ПІДХОДІВ ДЛЯ ПРЕДСТАВЛЕННЯ ЗНАНЬ

3.1 Правила та методи виведення правил

Будемо використовувати правила для поясннення роботи чорного ящика, тому що правила є одним із найзрозуміліших видів інформації для людини, а також існує безліч методів та підходів до отримання правил з даних. Будемо розглядати задачу класифікації, тому що більшість алгоритмів роботи з правилами призначені для цього. Класифікація на основі правил включає два ключові компоненти, а саме:

– індукцію правила, яка вивчає правила з певної навчальної бази даних автоматично;

– класифікацію, яка використовує набір правил для класифікації.

Процес вивчення правил з даних безпосередньо називається індукцією правил або вивченням правил. Більшість систем індукції правил використовують стратегію вивчення, яка описується як послідовне покриття [13].

Стратегія послідовного покриття (також відома як separate and conquer), безумовно, є найбільш вивченою і найбільш використовуваною сьогодні стратегією для виведення правил з даних. Вона була вперше використана алгоритмами сімейства AQ в кінці 1960-х років, і протягом багатьох років застосовувалась в якості основного алгоритму в системах індукції правил. Література з індукції правил і особливо послідовним алгоритмам покриття дуже обширна [13]. Існує багато досліджень, і робіт, що порівнюють різні алгоритми.

По суті, послідовна стратегія покриття працює наступним чином. Вона вивчає правило з навчального набору (крок conquer), видаляє з навчального набору приклади, що покриваються правилом (крок separate), і

рекурсивно вивчає інше правило, яке покриває інші приклади. Кажуть, що правило покриває приклад e, коли всі умови в антецеденті правила задовольняються прикладом e. Наприклад, правило «if (Salary > 150,000 euros) then Rich» покриває всі приклади в наборі навчання, в якому значення зарплати більше 150 000 євро, незалежно від поточного значення атрибута класу прикладу. Процес навчання триває до тих пір, поки не буде задоволений критерій зупинки. Цей критерій, як приклад, може вимагати, щоб всі або майже всі приклади в навчальному наборі покривалися правилами в заключному списку правил. Покриття всіх навчальних даних просто означає, що кожен навчальний екземпляр у навчальних даних задовольняє умовам принаймні одного правила у списку рішень – можливо, що один навчальний або тестовий екземпляр задовольняє декілька правил і, як правило, кожне правило може покривати кілька екземплярів [13]. Зупинка може відбуватися коли решта тренувального набору стає порожньою або не можна дізнатися нове правило з навчальних даних.

Існує дві основні стратегії об'єднання декількох правил: списки рішень (упорядковані) і набори рішень (невпорядковані). Обидві стратегії припускають різні рішення проблеми дублювання правил.

Список рішень вводить порядок в правила прийняття рішень. Якщо умова першого правила істинна для екземпляра, ми використовуємо передбачення першого правила. Якщо ні, переходимо до наступного правила і перевіряємо, чи застосовується воно і так далі. Списки рішень вирішують проблему правил, що перекриваються, повертаючи тільки прогноз першого правила в застосовуваному списку.

Набір рішень нагадує демократію правил, за винятком того, що деякі правила можуть мати більш високе право голосу. У наборі правила або взаємовиключні, або є стратегія вирішення конфліктів, така як голосування більшістю голосів, яка може бути зважена по точності окремих правил або іншим показникам якості.

І списки рішень, і набори можуть страждати від проблеми, що ніяке правило не застосовується до екземпляру. Це можна вирішити, ввівши правило за замовчуванням. Правило за замовчуванням – це правило, яке застосовується, коли ніяке інше правило не застосовується. Прогноз правила за замовчуванням часто є найчастішим класом серед точок даних, які не покриваються іншими правилами. Якщо набір або список правил охоплює весь простір об'єктів, ми називаємо його вичерпним. При додаванні правила за замовчуванням набір або список автоматично стають вичерпними.

Алгоритм 1.1 (рисунок 3.1) показує основний псевдокод для алгоритмів послідовного покриття, що виробляють невпорядковані набори правил, де виявлені правила застосовуються для класифікації нових тестових прикладів незалежно від порядку, в якому були виявлені правила. Алгоритм 1.2 (рисунок 3.2) описує процедуру LearnOneRule, використовувану алгоритмом. 1.1. В алгоритмі 1.1 третім кроком в циклі викликається алгоритм 1.2, передаючи в якості параметра початкове правило R, що має порожній антецедент правила (без умов в його if частині) і консеквент, що прогнозує клас C_i, пов'язаний із поточною ітерацією циклу for. Алгоритм 1.2 отримує це початкове правило і ітеративно розширює його, створюючи все кращі і кращі правила, поки не буде задоволено критерій зупинки. Найкраще правило, побудоване алгоритмом 1.2, потім повертається до алгоритму 1.1, де воно додається в набір виявлених правил. Після цього приклади класу C_i, охоплені щойно виявленим правилом, видаляються з навчального набору і так далі, поки не будуть виявлені правила для всіх класів.

В алгоритмах 1.1 та 1.2 елементи курсивом являють собою набір будівельних блоків, які можуть бути замінені для створення різних типів послідовних алгоритмів покриття [13]. Таким же чином блок «Створити початкове правило R» в алгоритмі 1.1 може бути замінено на «Створити порожнє правило R» (тобто правило без умов в його if частині) або

«Створити правило R, використовуючи випадковий приклад з навчального набору». Блок «оцінка CR» в алгоритмі 1.2 можна було б замінити на «оцінку CR, використовуючи точність» або «оцінку CR, використовуючи інформаційний приріст».

Algorithm 1.1 : CreateRuleSet (produces unordered rules)

```
RuleSet = ∅
for each class C_i do
    Set training set T as the entire set of training examples
    while Stopping Criterion is not satisfied do
        Create an Initial Rule R
        Set class predicted by new rule R as C_i
        R'= LearnOneRule(R)
        Add R'to RuleSet
        Remove from T all class C_i examples covered by R'
Post-process RuleSet
return RuleSet
```

Рисунок 3.1 – Загальний алгоритм послідовного покриття, що використовує невпорядкований набір правил

Algorithm 1.2 : LearnOneRule(R)

```
bestRule = R
candidateRules = ∅
candidateRules = candidateRules ∪ bestRule
while candidateRules ≠ ∅ do
    newCandidateRules= ∅
    for each candidateRule CR do
        Refine CR
        Evaluate CR
        if Refine Rule Stopping Criterion not satisfied then
            newCandidateRules = newCandidateRules ∪ CR
            if CR is better than bestRule then
                bestRule = CR
    candidateRules = Select b best rules in newCandidateRules
return bestRule
```

Рисунок 3.2 – Загальний алгоритм процедури LearnOneRule

Заміна будівельних блоків у цих базових алгоритмах конкретними методами може створити більшість існуючих алгоритмів послідовного покриття правил. Це можливо тому, що алгоритми, що слідують підходу

послідовного покриття, зазвичай відрізняються один від одного чотирма основними шляхами: представлення правил кандидата, пошукові механізми, що використовуються для вивчення простору правил кандидата, спосіб оцінки кандидатів і правило методу обрізки, хоча останній може бути відсутній [13].

В даний час найбільш використовувані і точні алгоритми дотримуються простого і базового підходу, описаного псевдокодом у алгоритмі 1.1 (виробляють невпорядковані правила) або псевдокодом в алгоритмі 1.3 (рисунок 3.3) (виробляють впорядковані правила) [4].

Algorithm 1.3: CreateDecisionList (produces ordered rules)

```
RuleList = ∅
Set training set T as the entire set of training examples
repeat
    Create an Initial Rule R
    R` = LearnOneRule(R)
    Set consequent of learned rule R' as the most frequent class found in the set of
    examples covered by R`
    Add R` to RuleList
    Remove from T all the examples covered by R`
until Rules Stopping Criterion not satisfied
return RuleList
```

Рисунок 3.3 – Загальний алгоритм послідовного покриття, що використовує впорядкований список правил

Порівнюючи алгоритми 1.1 і 1.3, зовнішній цикл for у алгоритмі 1.1 відсутній в алгоритмі 1.3 оскільки алгоритми списку рішень не вивчають правила для кожного класу по черзі. Замість цього вони намагаються вивчити на кожній ітерації найкраще з можливих правил для поточного набору навчання, щоб клас кожного правила-кандидата було обрано таким чином, щоб максимізувати якість цього правила з урахуванням його антецеденту. Зазвичай це робиться шляхом створення антецеденту правила кандидата, а потім установки консеквенту правила кандидата, відповідного найбільш частому класу серед всіх навчальних прикладів, що покриваються

цим правилом кандидата. Отже, правила, які прогнозують різні класи, можуть бути вивчені в будь-якому порядку (фактичний порядок залежить від даних навчання), і правила будуть застосовуватися для класифікації тестових прикладів у тому ж порядку, в якому вони були вивчені. Також змінюється набір прикладів, видалених з навчального набору після засвоєння правила. У той час як в алгоритмі 1.1 видаляються тільки покриті приклади, що відносяться до поточного класу C_i. У алгоритмі 1.3 всі наведені приклади (незалежно від їх класу) видаляються. Це відбувається тому, що, оскільки списки правил застосовують правила по порядку, якщо одне правило покриває тестовий приклад, ніякі інші правила не будуть мати можливості класифікувати його.

Існує три види стратегій пошуку: висхідна, спадна і двонаправлена. Стратегія знизу вгору починає пошук з дуже конкретного правила і ітеративно узагальнює його. Це конкретне правило зазвичай є прикладом з навчального набору, обраного випадковим чином (як у LERILS) або з використанням деякої евристики [13].

Стратегія зверху вниз, навпаки, починає пошук з самого загального правила і ітеративно спеціалізує його. Загальне правило – це те, яке охоплює всі приклади в навчальному наборі (тому що у нього є порожній антецедент, який завжди виконується для будь-якого прикладу). Стратегія зверху вниз частіше використовується послідовними алгоритмами покриття, але її основним недоліком є те, що по мірі індукції кількість даних, доступних для оцінки правила-кандидата, різко зменшується, зменшуючи статистичну надійність виявлених правил [13]. Це зазвичай призводить до перенавчання даних і проблеми правил, що покривають малу кількість прикладів. Тим не менш, є способи запобігти перенавчання в пошуку зверху вниз. Простий підхід з обмеженою ефективністю полягає в тому, щоб припинити розробку правил, коли число прикладів у поточному навчальному наборі опускається нижче деякого порогового рівня. Цей підхід може пропустити деякі рідкісні, але потенційно важливі правила, що представляють потенційно цікаві

винятки з більш загальних правил. Методи обрізки – більш складний підхід до спроби уникнути проблеми надмірної підгонки.

Нарешті, двонаправлений пошук дозволяє узагальнювати або спеціалізувати правила-кандидати. Це не загальний підхід, але його можна знайти в деяких алгоритмах.

Після вибору стратегії пошуку повинен бути реалізований метод пошуку. Метод пошуку є дуже важливою частиною алгоритму індукції правил, оскільки він визначає, які спеціалізації або узагальнення будуть розглядатися на кожному кроці спеціалізації або узагальнення. Занадто багато спеціалізацій або узагальнень не допускаються через вимоги до обчислювального часу, але занадто мало можуть ігнорувати хороші умови і зменшувати шанси знайти хороше правило. Жадібний і променевий пошук є найбільш популярними методами пошуку. Проблеми навчання, пов'язані з дуже складними взаємодіями атрибутів, як і раніше, є дуже складною проблемою для алгоритмів пошуку променя, а також для алгоритмів індукції правил і дерева рішень у цілому.

Також однією з проблем є проблема вибору методу оцінки правил. Області простору пошуку, що досліджуються за допомогою алгоритму індукції правил, можуть різко змінюватися відповідно до евристики оцінки правил, що використовується для оцінки правила під час його побудови [13].

Однією з важливих частин даного алгоритму є частина, де відбувається оцінка правил. В алгоритмі 1.2 це блок «Оцінка CR».

Окремі правила повинні одночасно оптимізувати два критерії [3]:

– покриття: кількість позитивних прикладів, що покриваються правилом (p), має бути максимізовано;

– узгодженість: кількість негативних прикладів, що покриваються правилом (n), має бути зведено до мінімуму.

Таким чином, більшість евристик залежать від p і n, але об'єднують ці значення по-різному. Наприклад, дві розповсюджені метрики Ентропія та оцінка Лапласа.

Ентропія має наступний вигляд:

$$\text{Entropy(S)} = -\sum_{i=1}^{c} p_i \log_2 p_i , \tag{3.1}$$

де S – зразок прикладу навчання,

p – частка прикладів одного з класів в S.

Ентропія вимірює нечистість S. У цій функції, чим менше ентропія, тим краще правило. Зростає з рістом p і зменшенням n. Але зі зрістом n і зменшенням p також зростає. Отже, дана оцінка дає однакове значення правилам, що покривають одну і ту ж кількість позитивних або негативних прикладів.

Оцінка Лапласа:

$$h_{Lap} = \frac{p+1}{p+n+2}, \tag{3.2}$$

де p – частка позитивних прикладів,

n – частка негативних прикладів.

Зростає з рістом p. Зростає із зменшенням n. Ілюстрація пошуку на вибірці погоди зображена на рисунку 3.4.

Такий підхід до реалізації LearnOneRule виконує general to specific пошук в просторі можливих правил, у пошуку правила з високою точністю, хоча, можливо, неповним покриттям даних. Під високою точністю мається на увазі, що передбачення, яке воно робить, має бути правильним. Низький рівень покриття означає, що не потрібно робити прогнози для кожного прикладу навчання.

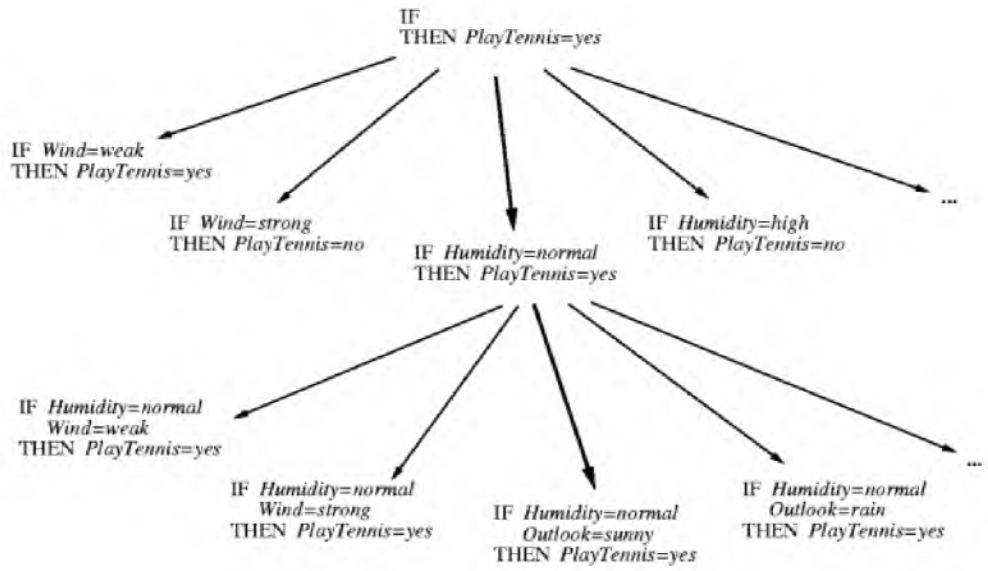


Рисунок 3.4 – Ілюстрація жадібного пошуку на вибірці погоди

3.2 Алгоритм CN2

Алгоритм CN2 – це метод класифікації, використовуючий стратегію послідовного покриття, призначений для ефективної індукції простих, зрозумілих правил форми «if cond then predict class», навіть в областях, де може бути присутній шум [14]. Вводячи описи концепцій, засновані на правилах «якщо – то», CN2 надає інструмент для надання допомоги в побудові систем, заснованих на знаннях, в яких потрібні процедури класифікації, засновані на правилах, а не на деревах рішень. CN2 демонструє, що можна успішно керувати пошуком через більший простір суперечливих правил з використанням розумно обраних евристик пошуку.

Він працює ітеративно, кожна ітерація шукає правило, яке охоплює велику кількість прикладів одного класу C і декілька інших класів. Правило повинно бути як прогностичним, так і надійним, як визначено функціями оцінки CN2. Знайшовши хороше правило, алгоритм видаляє ті приклади, які воно охоплює, з навчального набору і додає правило «if then predict C» в кінець списку правил [14]. Цей процес повторюється до тих пір, поки не

будуть знайдені більш задовільні правила. Система шукає правила, виконуючи скорочений пошук загального вигляду. На кожному етапі пошуку CN2 зберігає обмежений за розміром набір «кращих правил, знайдених на даний момент». Система досліджує тільки спеціалізації цього набору, здійснюючи променевий пошук простору правил. Правило спеціалізується або додаванням нового кон'юнктивного терміна, або видаленням диз'юнктивного елемента в одному з його селекторів. Кожне правило може бути спеціалізоване кількома способами, і CN2 генерує і оцінює всі такі спеціалізації.

Передумовами застосування данного алгоритму є:

– існує потреба у класифікації за допомогою пропозиційних правил;

– планується можливість додавання нових правил від експертів;

– у наявності є набір прикладів, який можна використовувати для навчання алгоритму, і який відповідає вимогам до вхідних даних алгоритму;

– набір даних може містити зашумлені дані;

– класифікація повинна бути швидкою;

– точність класифікації за допомогою отриманих правил повинна бути не гірше, ніж класифікація за допомогою інших існуючих методів.

Основним недоліком алгоритму є те, що це досі жадібний пошук, а отже пошук робить найкращий локальний вибір на кожному кроці, а глобальний кращий вибір може бути не знайдений. Фактично такий тип пошуку у процедурі LearnOneRule, яка в основному алгоритмі запускається у циклі, в результаті призведе до найкращого набору правил, а не кращих правил взагалі.

В даній роботі будемо використовувати даний алгоритм для пояснення роботи нейронної мережі. Він є досить простим, класичним методом, що дозволяє отримувати правила, що можуть бути представлені експертам на оцінку, розроблений для виведення якомога більш точних

правил, дозволяє проводити швидку класифікацію, та може працювати з зашумленими даними.

Через їх здатність безпосередньо вивчати правила на основі даних більшість алгоритмів виведення правил не вважаються методами видобування правил (RE) з чорних ящиків в строгому сенсі цього слова. Однак, ці алгоритми також можуть бути використані для вилучення зрозумілого для людини опису з непрозорих моделей. Згідно до [15] при використанні для цієї мети вихідні цільові значення навчальних прикладів замінюються прогнозами, зробленими моделлю чорного ящика, і потім алгоритм застосовується до цього зміненого набору даних. Якщо дерева або методи індукції правил використовуються як метод видобування правил, спостереження, які були помилково передбачені базовою моделлю, іноді не використовуються під час навчання.

Потрібно зазаначити, що RE методи вилучення правил мають перевагу перед інтерпретованими правилами прийняття рішень або деревами. Хоча комбінація ознак відсутня в навчальних даних, метод видобування правил все ще може працювати, оскільки його модель прогнозування, що лежить в основі, може присвоювати мітку класу будь-якій комбінації ознак [15]. Правила прийняття рішень залежать від доступних навчальних даних і не можуть доповнювати їх набір навчальних даних.

Звичайні алгоритми виведення правил та дерева рішень вже можна інтерпретувати, не нав'язуючи їм інтерпретованість. Це означає, що процес навчання явно не оптимізований для забезпечення інтерпретованості. Зазвичай ці алгоритми оптимізовані для забезпечення високої точності; інтерпретованість є побічним продуктом. Вона дана від природи, а отже, притаманна їм. Отримані моделі можуть бути пояснені безпосередньо. Однак для того, щоб бути легко зрозумілими людьми, моделі повинні бути розрідженими, маленькими і простими [15].

3.3 Онтології та SWRL правила

Мета пояснень систем штучного інтелекту полягає в тому, щоб зробити складні моделі зрозумілими для людей. Це включає в себе зв'язок пов'язаних фактів і їх інформаційну значимість для результату. Онтології як концепції, що представляють знання про різні взаємозв'язки даних, надають величезний потенціал для поліпшення розуміння складних структур даних [6]. Онтології сьогодні широко використовуються в області зрозумілого штучного інтелекту.

Крім того, самі онтології можуть бути використані в якості будівельних блоків для побудови ще більших відносин знань. Ці поняття, відомі як графи знань, являють собою високорівневі структури пов'язаної інформації. Використання їх для зв'язування джерел схожих тем робить їх відмінною основою для пошукових систем. Застосування онтологій може бути використано для поліпшення якості даних, а також для створення кращих пояснень. Шляхом включення знань про предметну область, схвалених експертами, наприклад, перевірка узгодженості може бути виконана безпосередньо на даних. Це може поліпшити класифікаційні характеристики моделі. Інші онтології можна використовувати для поліпшення зрозумілості, включивши їх перед навчанням моделі, наприклад, шляхом узагальнення ознак для полегшення розуміння користувачами. Крім рівномірного обміну знаннями в багатьох областях, онтології в основному використовуються для інтелектуального аналізу інформації. Основна увага приділяється не витяганню необроблених даних, а пізнанню можливих зв'язків всередині предметної області [6].

Онтологія є, у певному сенсі, скелетним каркасом знання. Онтологія – це опис речей, стосунків та їх характеристик, як правило, у досить обмеженій області, наприклад, екології чи астрономії. Частково це таксономія, яка являє собою структуру графа, що описує ієрархічний зв’язок групи речей. Речі, які знаходяться нижче в таксономії, успадковують

характеристики від одного або кількох батьків, які знаходяться вище від них. В онтології таксономія розширюється так, що виражаються логічні відносини між речами, членство в групах, відносини множинного спадкування, симетрію відносин, винятковість і різні характеристики даної речі [16]. Онтології семантичного інтернету займаються логічною узгодженістю та виразністю, щоб забезпечити можливість машинного висновку. У мовах онтологій семантичної мережі використовується логіка предикатів першого порядку. Логіка – це математична галузь, яка займається вираженням простих тверджень фактів у вигляді аксіом, а правил – як логічних виразів, які, за певних початкових умов, можна оцінювати щодо аксіом за допомогою висновку та дедукції. Оскільки онтології засновані на логіці, вони можуть полегшити процес, який імітує людські міркування. Ось як онтології можна використовувати в програмах штучного інтелекту. Варіанти використання моделей онтологій і отриманих даних екземплярів можуть включати пошук, застосування правил, які досліджують дані для визначення дій, і логічний висновок [16]. Однозначне моделювання фактів і забезпечення того, що дані, засновані на онтології, піддаються машинній обробці, були міркуваннями проектування OWL. На початку цього століття Консорціум World Wide Web (http://www.w3.org) розробив власну мову онтологій, мову веб-онтологій (OWL). Це робить OWL не просто іншою мовою онтологій, а наріжним каменем у величезному плані побудови «семантичної мережі», також званої «Web 3.0». Він має явні стосунки з іншими будівельними каменями Web 3.0: XML, RDF тощо. Автори [17] стверджують, що веб наступного покоління, тобто «семантичний веб», надає інтелектуальні сервіси, такі як інформаційні брокери, пошукові агенти та інформаційні фільтри, які забезпечують більшу функціональність, ніж поточні сервіси.

Деякі основні поняття в OWL:

– клас: група осіб, що поділяють ряд властивостей («річ» визначається як основний клас всіх);

– підклас: підрозділ класу;

– властивість: відносини між окремими особами або між окремими особами та іншими даними;

– домен: визначає, в якому класі може використовуватися властивість;

– діапазон: обмежує коло осіб, до яких може бути застосована власність;

– індивід: екземпляр класу.

Онтології фіксують поняття і відносини між поняттями в предметній області. Вони надають метаданим загальну семантику, забезпечуючи певний ступінь семантичної сумісності. Проблема полягає в тому, як представляти, створювати, управляти і використовувати онтології в якості загальних уявлень знань, а також великі обсяги записів метаданих, використовуваних для анотування ресурсів різних типів [18]. Метою побудови онтологій є обмін і повторне використання знань, а також опис семантично еквівалентних речей. Необхідно зіставляти елементи онтологій, якщо хтось хоче обробляти інформацію між додатками або доменами.

Рівень правил надає формат обміну для різних мов правил і механізмів виведення. Правила довгий час розглядалися як важлива парадигма для представлення і обґрунтування знань у семантичній мережі [19]. У 2005 році робоча група з формату обміну правилами W3C сформулювала наступні мотиви:

– самі правила являють собою цінну форму інформації, для якої ще не існує стандартного формату обміну. Правила забезпечують потужне уявлення бізнес-логіки, як бізнес-правила, в багатьох сучасних інформаційних системах;

– правила часто є технологією вибору для створення підтримуваних адаптерів між інформаційними системами;

– як частина архітектури семантичного вебу, правила можуть розширювати або доповнювати OWL, щоб більш ретельно охоплювати більш широкий набір додатків, при цьому знання кодуються в OWL або правилах або в обох.

Мова правил семантичної мережі (SWRL) є розширенням Owl DL, яке додає виразну силу правил (без заперечення) в OWL.

Основні конструкції SWRL – це правила, схожі на правила Horn. Однак, в той час як правила Horn містять комбінацію атомарної формули в антецеденті (тілі) правила і єдину атомарну формулу в консеквенті (заголовку) правила, SWRL допускає будь-який опис класу OWL, властивості або окремого інстансу як в тілі, так і в заголовку правила. Таким чином, SWRL відрізняється від традиційних систем правил, заснованих на логічному програмуванні або дедуктивних базах даних.

SWRL поєднує в собі повну виразну міць Horn логіки з виразною мовою логіки опису. У той час як деякі аксіоми OWL 2, такі як включення класів і включення властивостей, відповідають правилам, а інші класи можуть бути розкладені на правила, аксіоми ланцюжка властивостей забезпечують аксіоми, подібні правилам, існують формалізми правил, які не можуть бути виражені в OWL 2. Наприклад, заголовок правила з двома змінними не може бути представлений як аксіома підкласу, або тіло правила, що містить вираз класу, не може бути описано аксіомою підвластивості [20]. Щоб додати додаткову виразність правил в OWL 2 DL, онтології можна розширити за допомогою правил SWRL, які виражаються в термінах класів OWL, властивостей і окремих осіб. Правила SWRL – це правила в стилі datalog з унарними предикатами для опису класів і типів даних, двійковими предикатами для властивостей і деякими спеціальними вбудованими N-арними предикатами. Правила SWRL містять антецедент (тіло) і консеквент (голова), обидва з яких можуть бути атомом або з'єднанням атомів.

Голова і тіло складаються із з'єднання одного або декількох атомів. В даний час SWRL не підтримує більш складні логічні комбінації атомів. Правила SWRL підтримують висновок про індивідів OWL, в першу чергу з точки зору класів і властивостей OWL [20]. Наприклад, правило SWRL, що виражає, що у людини, що має брата чоловічої статі, є брат, зажадає врахування понять «людина», «чоловік», «брат» в OWL. Інтуїтивно, поняття людини і чоловіка може бути передано за допомогою класу OWL під назвою Person з підкласом Man; відносини між братом і сестрою можуть бути виражені за допомогою властивостей OWL hasSibling і hasBrother, які прив'язані до людини. Правило в SWRL тоді буде наступним: Person (?x1) ^ hasSibling(?x1,?x2) ^ Man(?x2) → hasBrother(?x1,?x2). Виконання цього правила призведе до встановлення властивості hasBrother на x2 у людини, яка задовольняє правилу, та відповідає змінній x1. SWRL також підтримує ряд вбудованих предикатів, які значно розширюють його виразні можливості. Вбудовані SWRL – це предикати, які приймають кілька аргументів. Вони детально описані у вбудованій специфікації SWRL. Найпростішими вбудованими функціями є операції порівняння. Наприклад, greaterThan. Правила SWRL також можуть чітко ставитися до окремих інивідів OWL. SWRL також підтримує літерали даних. На додаток до вбудованих математичних функцій, існують вбудовані функції для рядків, дат і списків. У майбутньому в цей простір імен можуть бути внесені доповнення, щоб розширити діапазон вбудованих модулів, підтримуваних SWRL [20].

В дані роботі будемо використовувати онтологію для можливості ділитися знаннями та використовувати знання повторно. Існують зручні редактори онтологій, які дозволяють працювати з ними через програмний інтерфейс. Онтології мають зручне розширення у вигляді SWRL правил, які дозволяють представляти правила у обраній предметній області, та роботи логічний висновок використовуючи правила.

4 РОЗРОБКА СИСТЕМИ ПОЯСНЕННЯ РОБОТИ НЕЙРОННОЇ МЕРЕЖІ

4.1 Опис рішення, що пропонується

В основі пояснення роботи нейронної мережі лежить описаний раніше підхід зворотнього інжинірингу. ШНМ тренується частині набору даних для тренування. Інша частина набору даних використовується для того, щоб опитати нейронну мережу та замінити справжні мітки класів на отримані мітки від нейронної мережі. На отриманому наборі даних, що складається з відповідей нейронної мережі, тренується алгоритм виведення правил. Отримані правила конвертуються у SWRL правила. Створюється онтологія, у яку додаються правила та відповідні властивості, а також класи із значень цільового атрибуту. Завантаживши онтологію у редактор онтологій Protégé, експерт у певній ПО може вносити зміни до виведених правил, застостовувати різонер для класифікації нових прикладів, доповнювати її інформацією з певної предметної області. Загальна система з точки зору робочого процесу зображена на рисунку 4.1.

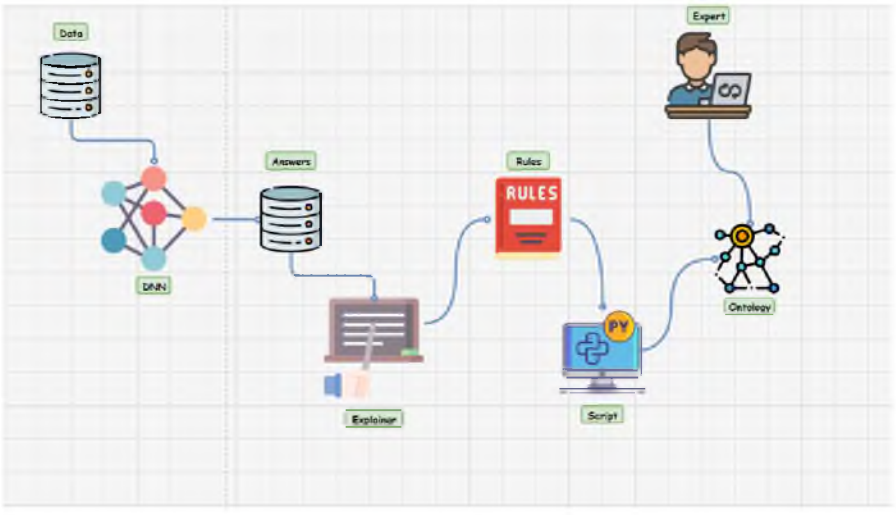


Рисунок 4.1 – Пропонований робочий процесс

4.2 Опис побудуваної системи

Для пояснення роботи нейронної мережі та подальшої можливості внесення змін до отриманих знань, була побудована система, що складається з декількох модулів. Для реалізації системи використовувалися мови програмування Python та Java. Побудована система складається з модуля, який відповідає за завантаження даних з файлу, їх передобробку та навчання нейронної мережі, навчання відбувається за допомогою бібліотек Keras та TensorFlow, в якості допоміжних бібліотек використовуються scikit та pandas. Також даний модуль отримує відповіді нейронної мережі на тестових даних, додає до частини відповідей реальні цільові класи, щоб у подальшому використовувати їх для тестування алгоритму виведення правил, та зберігає їх у файл. Також він зберігає деяку частину даних для подальшого їх використання для завантаження в онтологію за допомогою плагіну.

Ще один модуль, написаний на Python з використанням бібліотеки для машиного навчання Orange, завантажує файл з даними, який містить відповіді нейронної мережі для подальшого використання алгоритму CN2 та отримання набору правил, та тестові дані для оцінки точності отриманих правил. Даний модуль також робить деяку передобробку даних для підготовки до використання алгоритму вивчення правил на них, наприклад, категоризацію числових атрибутів з розбиттям на діапазони. Даний модуль отримує набір правил та перетворює їх у SWRL правила, створює онтологію з даними правилами та відповідними властивостями та класами для ключового атрибуту. Дана онтологія може бути завантажена до програми роботи з онтологіями Protégé.

З використанням мови Java був написаний плагін для програми роботи за онтологіями Protégé. Він дозволяє завантажити дані з файлу до онтології, у вигляді інстансів онтології, а також їх властивості.

Схема програмних компонентів та даних для наведеної системи

зображена на рисунку 4.2. Отримані правила будуть оцінені експертом, а також за допомогою спеціальних метрик одразу після їх виведення алгоритмом.

Програмна реалізація алгоритму CN2 бібліотекою Orange дозволяє вказувати деякі параметри для алгоритму, що впливає на отримані правила та їх точність. Через деякі обмеження бібліотеки Orange відповіді ШНМ та тестові дані зберігаються в один файл, а не в окремі файли.

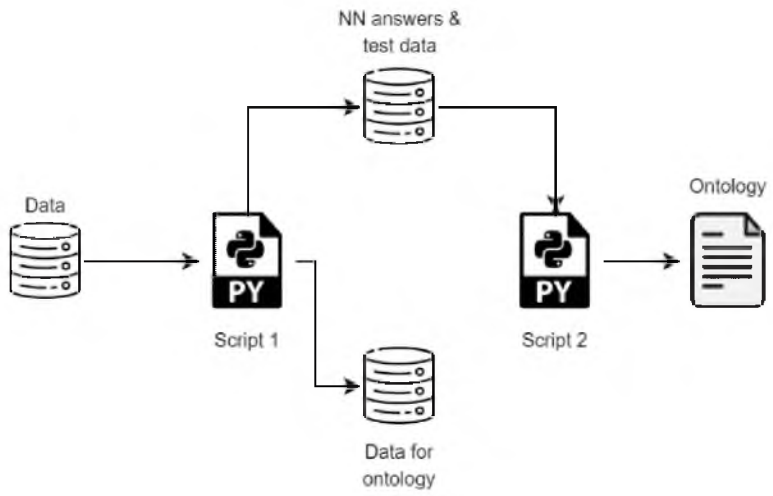


Рисунок 4.2 – Схема програмних компонентів та даних

Модуль, що отримує правила використовуючи алгоритм виведення правил, може бути запущений за допомогою командного рядка Windows. Він отримує деякі параметри для своєї роботи, такі як:

– розмір даних для навчання алгоритму, інша частина буде використовуватись для тестування;

– режим роботи алгоритму, що буде використовуватись (0 або 1);

– мінімальна кількість прикладів, які повинні покриватись правилами;

– максимальна довжина правил;

– ширина променевого пошуку;

– метрику для оцінки (1 або 2);

– кількість діапазонів для дискретизації числових атрибутів.

Кожен з них може бути опущений, в такому випадку будуть використовуватись значення за замовчуванням, у випадку опущення розміру даних для навчання усі дані з файлу будуть використовуватись. Також повинен бути вказаний шлях до файлу з даними. Приклад запуску зображений на рисунку 4.3.

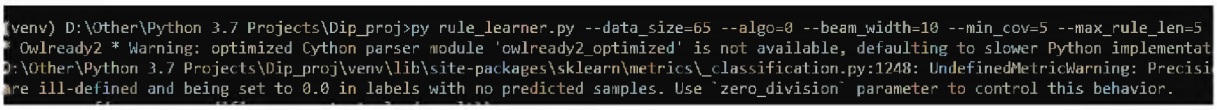


Рисунок 4.3 – Приклад запуску з командного рядку з параметрами

4.3 Використані дані

Для тренування нейронної мережі, її тестування, отримання правил та їх перевірку використовується наступне розбиття набору даних (рисунок 4.4). Деяка частина зберігається для можливості подальшого завантаження у онтологію, більша частина використовується для тренування нейронної мережі, інша частина використовується для тестування нейронної мережі, та у свою чергу поділяється на дані які ми отримуємо як передбачення нейронної мережі і будуть використані для навчання алгоритму виведення правил, та частину, яка використовується для тестування даного алгоритму.

В даній роботі в більшості випадків використовується 5 записів датасету для онтології, 60-70% даних для тренування ШНМ, 20-30% для тестування алгоритму вивдення правил.

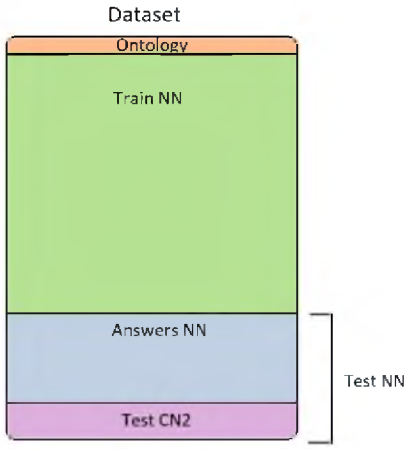


Рисунок 4.4 – Розбиття даних

В якості даних для тестів будемо використовувати набори даних перелічені у таблиці 4.1. Перші два датесети використовуються для класифікації зірок, відносячи їх до певного типу зірки. Перший набір даних має декілька категоричних атрибутів, другий – один. Останній датасет представляє собою розташування фігур у кінці гри хрестики-нолики та відповідний клас-переможець в якості цільового атрибуту.

Таблиця 4.1 – Набори даних

Назва	Розмір	Кількість атрибутів	Тип атрибутів	Цільові класи
Stars	241	6	Змішаний	7
Stars2	3643	6	Змішаний	2
TicTacToe	959	9	Символьний	2

4.4 Оцінка правил

Для оцінки отриманих правил будемо використовувати метрики accuracy, precision, recall, f1:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \quad (4.1)$$

$$precision = \frac{TP}{TP+FP}, \quad (4.2)$$

$$recall = \frac{TP}{TP+FN}, \quad (4.3)$$

де TP – true positive,

TN – true negative,

FP – false positive,

FN – false negative.

Метрика F1 розраховується за формулою:

$$F1 = 2 * \frac{precision*recall}{precision+recall}. \quad (4.4)$$

Для розрахунку даних метрик будемо використовувати функцію classification_report бібліотеки scikit.

4.5 Опис програмної реалізації

Перед початком тренування нейронної мережі, датасети були завантажені, та передоброблені. Усі числові атрибути були нормалізовані, а нечислові перетворені у числові за допомогою класу LabelEncoder бібліотеки scikit або методу get_dummies біліотеки pandas. Також було проведене розбиття даних на тестову частину та частину для тренування. Приклад передобробки та розбиття для датасету Stars на рисунку 4.5.

Наступним кроком став вибір та побудова архітектури нейронної мережі, яка могла б бути використана для класифікації нових прикладів з високою точністю. Модель складається з декількох шарів, по 32 нейрони в

кожному, на прихованих шарах використовуються активаційні функції relu, на вихідному слої softmax, який містить стільки ж нейронів скільки класів ми маємо у цільовому атрибуті.

```
dataset = pd.read_csv("data\\Stars.csv", delimiter=",")
x = dataset.iloc[:, :4]
x2 = dataset.iloc[:, 5:7]
y = dataset.iloc[:, 4:5]

x = pd.concat([x, x2], axis=1)

le_star_color = LabelEncoder()
x['Star color'] = le_star_color.fit_transform(x['Star color'])

le_spectral_class = LabelEncoder()
x['Spectral Class'] = le_spectral_class.fit_transform(x['Spectral Class'])

sc = StandardScaler()
x = sc.fit_transform(x)

y = y.iloc[:,:1].values

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, shuffle=False)
```

Рисунок 4.5 – Підготовка даних

В якості оптимізатора використовується стохастичний градієнтний спуск, з параметром навчання 0.01. В якості функції втрат SparseCategoricalCrossentropy. Код побудованої ANN зображений на рисунку 4.6.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(x_train.shape[1],)),  # input layer
    tf.keras.layers.Dense(32, activation='relu'),  # dense layer 1
    tf.keras.layers.Dense(32, activation='relu'),  # dense layer 2
    tf.keras.layers.Dense(32, activation='relu'),  # dense layer 2
    tf.keras.layers.Dense(6, activation='softmax'),  # dense layer 5
])

model.summary()

model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=["accuracy"])

print("Train using SGD")
t1_start = time.perf_counter()
history_sgd = model.fit(train_dataset, validation_data=(x_test, y_test), epochs=NUM_EPOCHS_SGD)
```

Рисунок 4.6 – Архітектура та параметри нейронної мережі

При тренуванні використовуємо 100 епох та розмір батчу, який для різних датасетів може мати розміри від 16 до 512 записів. Початок одного запуску тренування на рисунку 4.7.

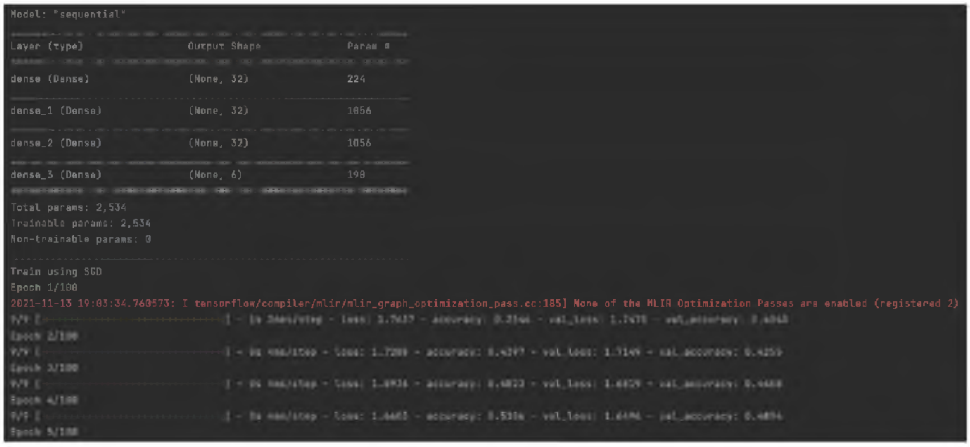


Рисунок 4.7 – Приклад тренування нейронної мережі

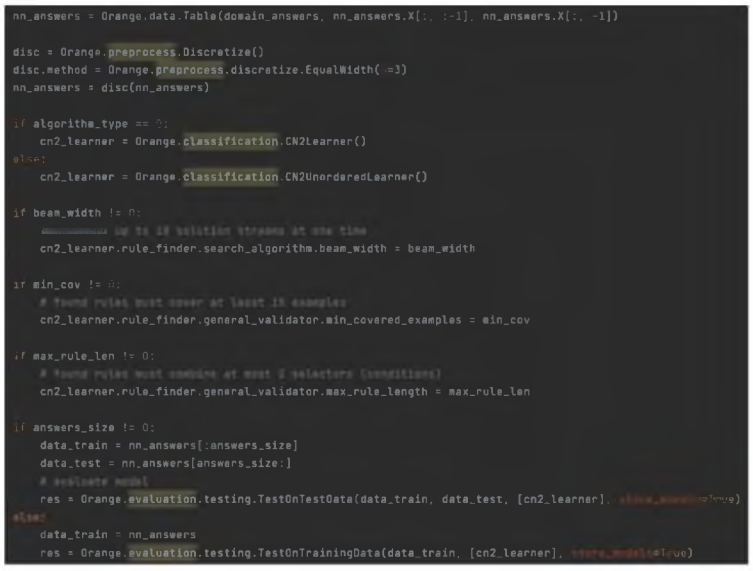
Після тренування нейронної мережі дані, що використовуються для тестування, показуються нейронній мережі для отримання відповідей. Також відбувається трансформація даних у початковий вигляд. Отримані відповіді зберігаються у файл (рисунок 4.8).

```
# Get predictions, transform the data back, save to file
y = model.predict(x_test)
y_predicted = np.array(list(map(lambda x: np.argmax(x), y)))
y_predicted = y_predicted.reshape(y_predicted.shape[0], 1)

x_test = x_test.numpy()
x_test = sc.inverse_transform(x_test)

x_test = np.hstack((x_test, y_precicted))

new_columns = dataset.columns

dataset_from_test_data_with_predicted_values = pd.DataFrame(data=x_test, index=list(range(x_test.shape[0])), columns=new_columns[:4].append(new_columns[5:7]).a

dataset_from_test_data_with_predicted_values['Star color'] = dataset_from_test_data_with_predicted_values['Star color'].astype(int)
dataset_from_test_data_with_predicted_values['Spectral Class'] = dataset_from_test_data_with_predicted_values['Spectral Class'].astype(int)

dataset_from_test_data_with_predicted_values['Star color'] = le_star_color.inverse_transform(dataset_from_test_data_with_predicted_values['Star color'])
dataset_from_test_data_with_predicted_values['Spectral Class'] = le_spectral_class.inverse_transform(dataset_from_test_data_with_predicted_values['Spectral Cla

dataset_from_test_data_with_predicted_values.to_csv("./data/Stars_answers.csv", index=False)
```

Рисунок 4.8 – Отримання відповідей та зберігання їх у файл

Для отримання правил з відповідей ШНМ побудований ще один

модуль, написаний на мові Python. Цей скрипт завантажує відповіді ШНМ, робить деяку передобробку даних, наприклад категоризацію числових атрибутів. Використовуючи алгоритм CN2 відбувається виведення правил з даних за допомогою бібліотеки Orange. Отримані правила виводяться на консоль та оцінюються на тестовій частині даних, яка завантажується з того ж файлу. Правила конвертуються у формат SWRL правил та створюється онтологія, у яку дані правила додаються. Також до онтології додаються DataProperty та класи цільового атрибуту, які отримуються із назв атрибутів та значень цільового атрибуту у вибірці. Дана онтологія зберігається в якості OWL файлу, та може бути відкрита у програмах, підтримуючих роботу із таким форматом даних. Загальний код роботи алгоритму зображений на рисунках 4.9, 4.10. Алгоритм роботи даного модулю зображений на рисунку 4.11. Інша частина коду для тренування алгоритму та створення онтології наведена у додатку А.



```
nn_answers = Orange.data.Table(domain_answers, nn_answers.X[:, :-1], nn_answers.X[:, -1])

disc = Orange.preprocess.Discretize()
disc.method = Orange.preprocess.discretize.EqualWidth(n=3)
nn_answers = disc(nn_answers)

if algorithm_type == 0:
    cn2_learner = Orange.classification.CN2Learner()
else:
    cn2_learner = Orange.classification.CN2UnorderedLearner()

if beam_width != 0:
    cn2_learner.rule_finder.search_algorithm.beam_width = beam_width

if min_cov != 0:
    cn2_learner.rule_finder.general_validator.min_covered_examples = min_cov

if max_rule_len != 0:
    cn2_learner.rule_finder.general_validator.max_rule_length = max_rule_len

if answers_size != 0:
    data_train = nn_answers[:answers_size]
    data_test = nn_answers[answers_size:]
    res = Orange.evaluation.testing.TestOnTestData(data_train, data_test, [cn2_learner])
else:
    data_train = nn_answers
    res = Orange.evaluation.testing.TestOnTrainingData(data_train, [cn2_learner])
```

Рисунок 4.9 – Завантаження даних та навчання алгоритму

```
cn2_classifier = res.models[0][0]

y_pred = cn2_classifier(data_test.X)
print(classification_report(data_test.Y, y_pred))

for rule in cn2_classifier.rule_list:
    print(rule, rule.curr_class_dist.tolist())

owl_rules = convert_rules_to_swrl_rules(cn2_classifier.rule_list)

ontology = create_ontology("http://www.semanticweb.org/nn_rule_explainer/onto.owl")

# create data properties and target classes in ontology
create_ontology_entities_from_rules(ontology, owl_rules, data_train)

add_rules_to_ontology(ontology, owl_rules)

save_ontology_to_file(ontology, "onto.owl")
```

Рисунок 4.10 – Оцінювання правил, їх конвертація та створення онтології з ними

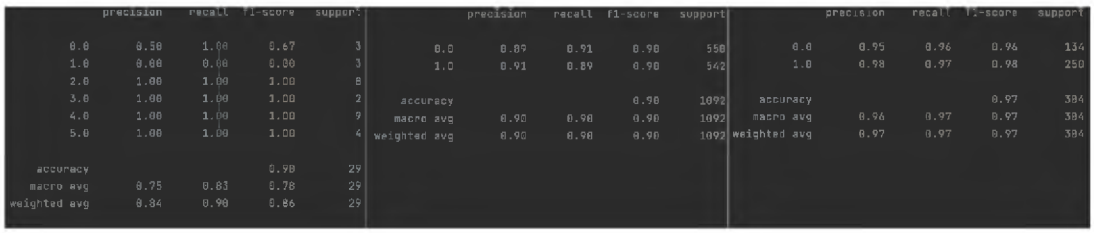
Рисунок 4.11 – Алгоритм роботи скрипту

4.6 Правила та їх оцінка

На усіх наборах даних нейронна мережа була навчена до досить високої точності. Приклади отриманих звітів класифікації на датасетах для ШНМ зображені на рисунку 4.12.

Загалом точність тримається у діапазоні 80-95% для усіх датасетів, лише для першого набору даних показники для окремих класів можуть бути низькими через те, що у першому наборі кількість даних невисока, а розподілення класів нерівномірне.

Перед кожним запуском дані перемішуються. Середні оцінки за 10 запусків для датасетів наведені у таблиці 4.2.



	precision	recall	f1-score	support
0.0	0.50	1.00	0.67	3
1.0	0.00	0.00	0.00	3
2.0	1.00	1.00	1.00	8
3.0	1.00	1.00	1.00	2
4.0	1.00	1.00	1.00	9
5.0	1.00	1.00	1.00	4
accuracy			0.90	29
macro avg	0.75	0.83	0.78	29
weighted avg	0.84	0.90	0.86	29

	precision	recall	f1-score	support
0.0	0.89	0.91	0.90	550
1.0	0.91	0.89	0.90	542
accuracy			0.90	1092
macro avg	0.90	0.90	0.90	1092
weighted avg	0.90	0.90	0.90	1092

	precision	recall	f1-score	support
0.0	0.95	0.96	0.96	134
1.0	0.98	0.97	0.98	250
accuracy			0.97	384
macro avg	0.96	0.97	0.97	384
weighted avg	0.97	0.97	0.97	384

Рисунок 4.12 – Приклади отриманих звітів класифікації ШНМ на датасетах Stars, Stars2, TicTacToe

Таблиця 4.2 – Середні оцінки для ШНМ

Назва	Accuracy	Precision	Recall	F1
Stars	94.5	95.375	94.75	94.6
Stars2	89.4	89.4	89.4	89.4
TicTacToe	94.5	94.7	94.5	94.3

Для кожного з наборів даних можуть бути отримані правила, в залежності від використовуваних параметрів правила можуть змінюватись, відповідно і їх точність також змінюється. Для отримання кращого правила на кожному кроці алгоритмом використовується метрика Лапласа. Приклади отриманих правил на датасетах Stars та Stars2 разом із розподіленням класів та оцінкою метрикою Лапласа наведені на рисунку 4.13. Так як другий датасет містить більше прикладів, то і кількість отриманих правил значно більша, для зменшення кількості використовувався параметр min_cov із значенням 20. Інші параметри встановлені за замовчуванням.

Кожен отриманий набір або список правил був оцінений за допомогою тих же метрик оцінки. Оцінки наведених правил на тестовій частині датасету зображені на рисунку 4.14. Середні оцінки для датасетів за 10 запусків наведені у таблиці 4.3.

Завдяки останньому правилу за замовчуванням, списки правил покривають усю множину вхідних прикладів. Але не всі атрибути можуть

бути присутні у правилах. Цільові класи також можуть бути відсутніми у підсумковому наборі, це залежить або від розміру набору даних, або від внутрішньої реалізації алгоритму навчання правил.

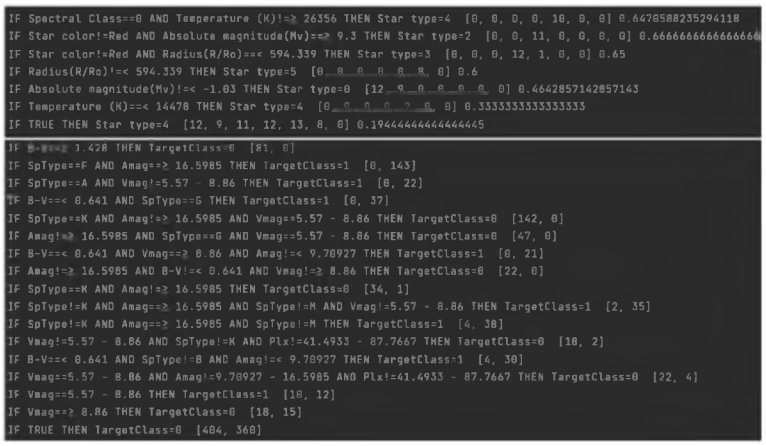


Рисунок 4.13 – Приклади отриманих правил для датасетів Stars (зверху) та Stars2 (знизу)

	precision	recall	f1-score	support
0.0	0.56	1.00	0.71	5
1.0	0.00	0.00	0.00	4
2.0	1.00	1.00	1.00	3
3.0	0.80	1.00	0.89	4
4.0	1.00	0.83	0.91	6
5.0	1.00	1.00	1.00	7
accuracy			0.83	29
macro avg	0.73	0.81	0.75	29
weighted avg	0.76	0.83	0.78	29

	precision	recall	f1-score	support
0.0	0.86	0.86	0.86	175
1.0	0.84	0.84	0.84	153
accuracy			0.85	328
macro avg	0.85	0.85	0.85	328
weighted avg	0.85	0.85	0.85	328

Рисунок 4.14 – Приклади отриманих звітів класифікації на правилах для датасетів Stars та Stars2

Таблиця 4.3 – Середні оцінки правил

Назва	Accuracy	Precision	Recall	F1
Stars	85.7	91.1	85.7	84.6
Stars2	87	87.3	87.5	86
TicTacToe	89	90.5	89.7	89

Аналізуючи декілька запусків алгоритму на різних частинах набору даних, на першому датасеті показник accuracy та середній показник precision варіюються від 60-80 відсотків, recall та f1 від 70-90 відсотків. Через те, що даних небагато не завжди приклади з усіма класами потрапляють до даних для навчання, і точність класифікації окремих класів падає. На другому датасеті усі показники тримаються в діапазоні 80-90%. Загалом можна сказати, що отримані правила мають досить непогані оцінки, але треба також зазначити, що для тренування нейронної мережі використовувалось значно більше даних. Можливо при однакові кількості даних результати були б кращі.

Цікаву тенденцію можна побачити на датасеті TicTacToe, даний датасет представляє собою записи розміщення позначок у грі хрестики-нолики, та відповідний результат гри, де позитивний клас це перемога іксів. Не треба бути експертом, що побачити на цьому датасеті деякі залженості в даних, які ведуть до певного результату, наприклад, три хрестики в певних клітинках ведуть до перемоги хрестиків. Використовуючи приблизно 600 записів для тренування нейронної мережі та приблизно 270 для тренування алгоритму CN2, використовуючи такі параметри як режим роботи – Unordered та мінімальна кількість прикладів покритих правилами – 10, були отримані оцінки точності правил зображені на рисунку 4.15. Загалом середня точність тримається в діапазоні 70-90%.

	precision	recall	f1-score	support
0.0	0.61	0.90	0.73	31
1.0	0.96	0.79	0.86	85
accuracy			0.82	116
macro avg	0.78	0.85	0.80	116
weighted avg	0.86	0.82	0.83	116

	precision	recall	f1-score	support
0.0	0.84	0.93	0.88	41
1.0	0.96	0.91	0.93	75
accuracy			0.91	116
macro avg	0.90	0.92	0.91	116
weighted avg	0.92	0.91	0.91	116

Рисунок 4.15 –Приклади отриманих звітів класифікації для датасету TicTacToe

Отримані набори правил зображені на рисунку 4.16. Можна побачити, що отримані правила відображають реальні залежності в даних, які ведуть до певного результату. Серед правил можна побачити правила, які нагадують майже повністю правильні правила, наприклад, в деяких правилах дві позначки “x” в ряд ведуть до позитивного результату. Чим більше даних для навчання алгоритму, тим більше правила відповідають оригінальним даним. Але зменшуючи кількість даних для навчання ШНМ, зменшується і точність ШНМ на тестових даних, а отже і відповіді ШНМ меньш достовірні, тому потрібно дотримуватись балансу. Так як для зрозумілого штучного інтелекту важливо, щоб отримані правила були якогомога коротшими, та їх кількість була меншою, змінюючи такі параметри як максимальна дожина правил та мінімальна кількість покритих прикладі, можна отримати компактний набір правил, з досить непоганою точністю класифікації, але звісно нижчою, ніж звичайно. Також можна зазначити, що порівнюючи отримані правила з різних запусків навчання ШНМ та тренування алгоритму, вони завжди мають деякі схожі правила.

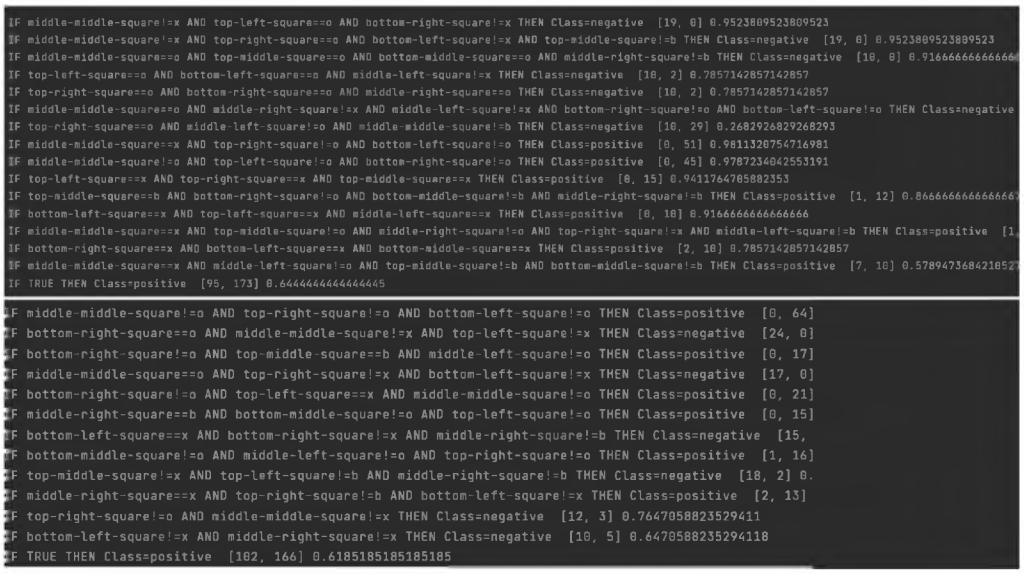


Рисунок 4.16 – Приклади отриманих правил для датасету TicTacToe при різних значеннях параметрів алгоритму

4.7 Онтологія

Згенерована другим модулем онтологія у вигляді файлу з назвою “onto.owl” з’являється після запуску скрипту у той самій директорії біля нього. Вона може бути відкрита одним з редакторів онтологій, в нашому випадку у Protégé. Отримані класи та властивості даних для датасету Stars наведені на рисунку 4.17. Отримані SWRL правила для двох датасетів наведені на рисунках 4.18, 4.19. Вони можуть бути переглянуті або у вікні “Rules” вкладки активної онтології, або у вкладці “SWRL Tab”. Також їх можна редагувати. Вони представлені майже усіма правилами, виведеними раніше алгоритмом, окрім правила за замовчуванням, дане правило відобразити в термінах SWRL не представляється можливим.

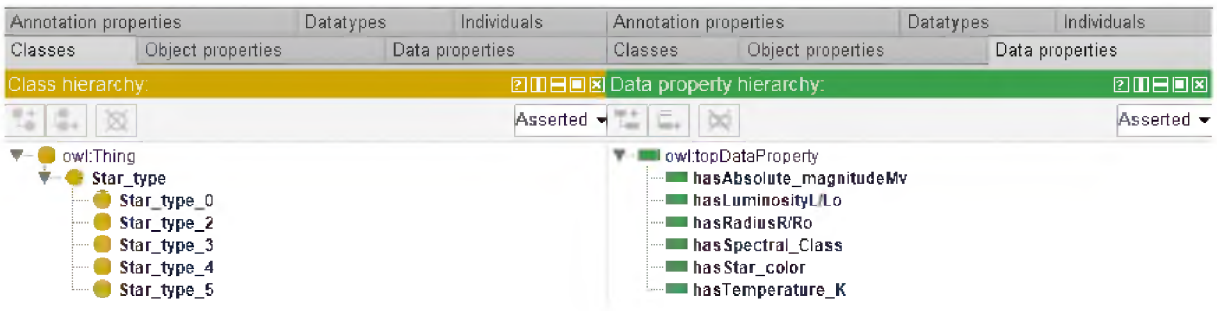


Рисунок 4.17 – Класи та DataProperty в онтології

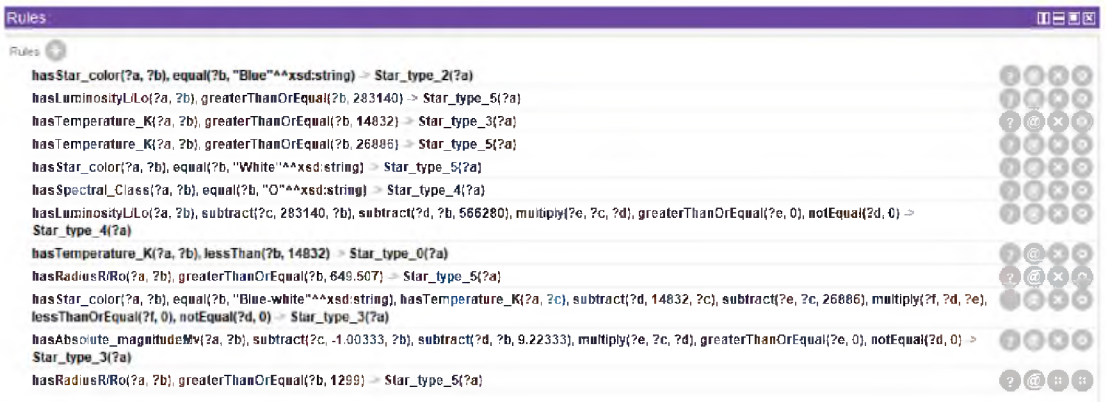


Рисунок 4.18 – Отримані з першого датасету SWRL правила у вікні Rules

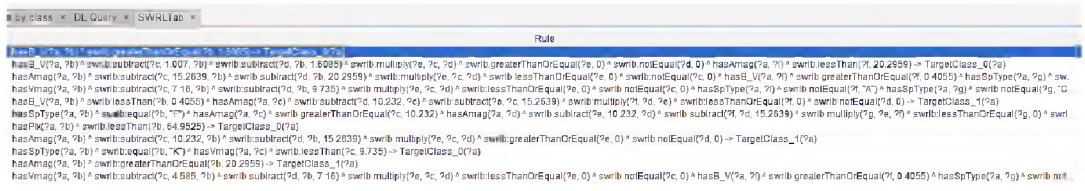


Рисунок 4.19 – Отримані з другого датасету SWRL правила у вкладці SWRL

Для того, щоб в онтології можливо було проводити класифікацію інстансів даних, був побудований плагін для програми Protégé, який дозволяє імпортувати дані з файлу до онтології у вигляді інстансів з певними властивостями. Плагін представляє собою jar файл, який додається до папки Plugins програми Protégé, він написаний на мові Java, використовуючи OWL API. Даний плагін додає новий пункт меню до одного з випадаючих меню на верхній панелі. Загалом, процес роботи із розробленим плагіном схематично зображений на рисунку 4.20. Наприклад, файл із завантажуваними даними має вигляд, наведений на рисунку 4.21.

Рисунок 4.20 – Процес роботи із плагіном

A	B	C	D	E	F	G
Temperature (K)	Luminosity(L/Lo)	Radius(R/Ro)	Absolute magnitude(Mv)	Star color	Spectral Class	Star type
3068	0.0024	0.17	44546	Red	M	0
3042	0.0005	0.1542	44363	Red	M	0
2600	0.0003	0.102	44395	Red	M	0
2800	0.0002	0.16	16.65	Red	M	0
1939	0.000138	0.103	44367	Red	M	0

Рисунок 4.21 – Файл для імпорту до онтології

Для завантаження файлу необхідно обрати пункт меню Tools, підменю Create instances from csv (рисунок 4.22). Потім вибрати файл у файловій системі (рисунок 4.23). Файл повинен бути у текстовому форматі. В процесі завантаження файлу плагін створює властивості даних з назв атрибутів, перед цим перевіряючи, чи присутні дані властивості в онтології, якщо ні, то він створює нові властивості. Інстанси створюються з кожного рядку даних, та кожному інстансу привласнюються відповідні властивості.

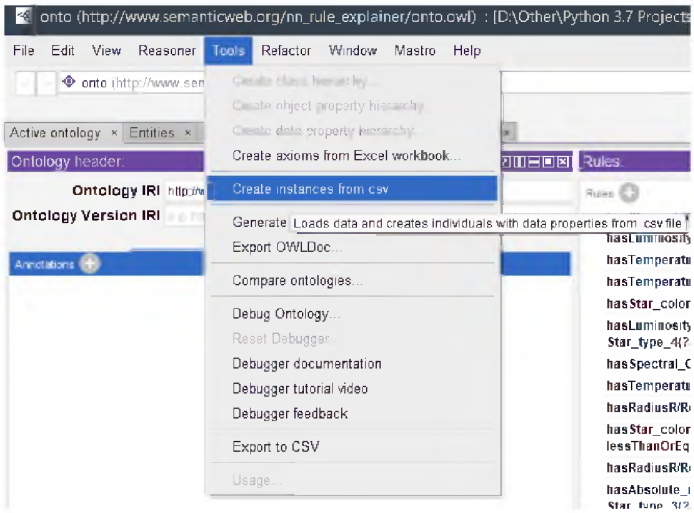


Рисунок 4.22 – Вибір пункту меню

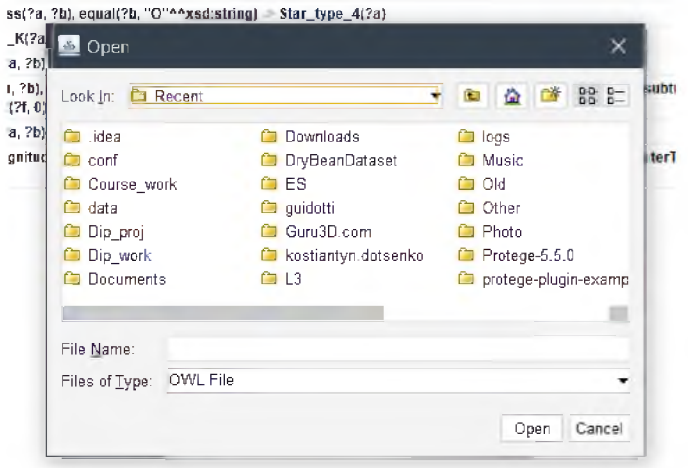


Рисунок 4.23 – Діалог вибору файлу

Перед кожним завантаженням поточні інстанси видаляються, виконується очищення онтології. Після цього в онтології з'являються нові інстанси, з властивостями та значеннями, отриманими з даних. Отримані інстанси зображені на рисунку 4.24.

Після того, як інстанси та їх властивості створені у онтології, з'являється можливість використати різонер, для того, щоб класифікувати імпортовані інстанси на основі значень їх властивостей та SWRL правил.

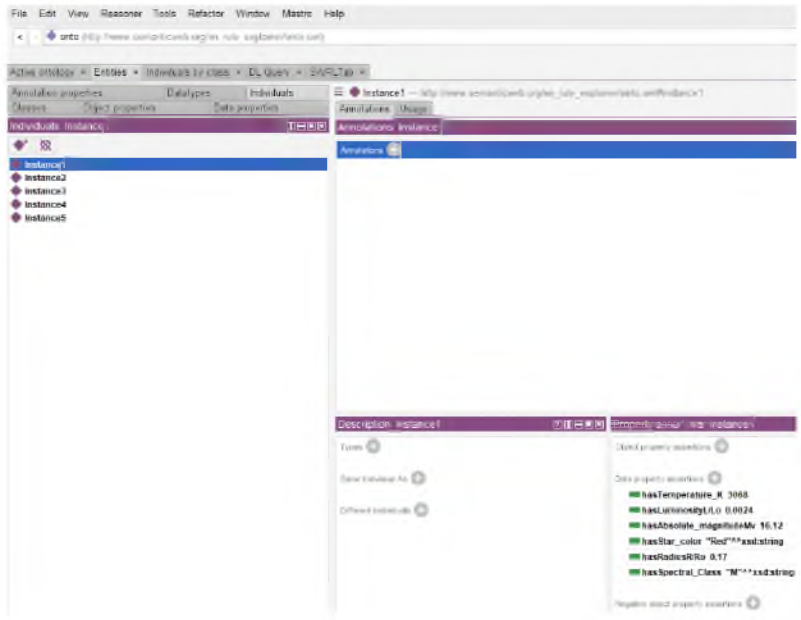


Рисунок 4.24 – Отримані інстанси та їх властивості

Для цього необхідно обрати різонер у верхній панелі меню, пункт Reasoner. В нашому випадку використовується Pellet, тому що це єдиний різонер, який повністю підтримує роботу із SWRL правилами. Також для цієї задачі можливо застосовувати рушій Drools. Для цього потрібно послідовно натиснути кнопку для імпорту даних до Drools, запустити Drools та натиснути кнопку для експорту отриманх аксіом до онтології. Після запуску різонера інстанси можуть бути віднесені до одного з цільових класів. Приклад класифікації одного з інстансів та пояснення класифікації на рисунку 4.25.

Однією з проблем, що виникає при даному підході є те, що один і той же інстанс може бути віднесений різонером до різних класів, як на рисунку 4.26. При класифікації використовуючи набори або списки правил зазвичай існують стратегії вирішення конфліктів, як наприклад при використанні версії алгоритму CN2 з бібліотеки Orange. Тому один приклад завжди виноситься лише до одного класу. На жаль, SWRL правилами не підтримується можливість вирішення конфліктів. Так само, як і можливість використання правила за замовчуванням, тому деякі інстанси можуть бути не класифіковані. Приклад класифікації інстансу на іншому датасеті зображений на рисунку 4.27.

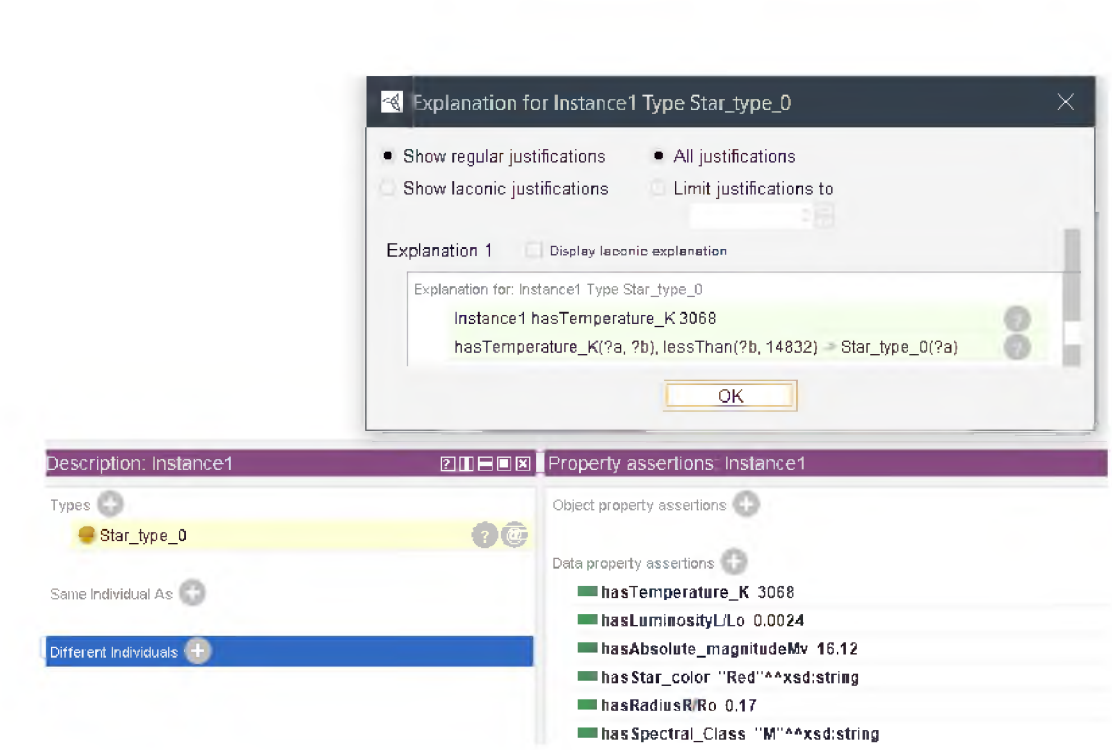


Рисунок 4.25 – Вивід та пояснення різонеру

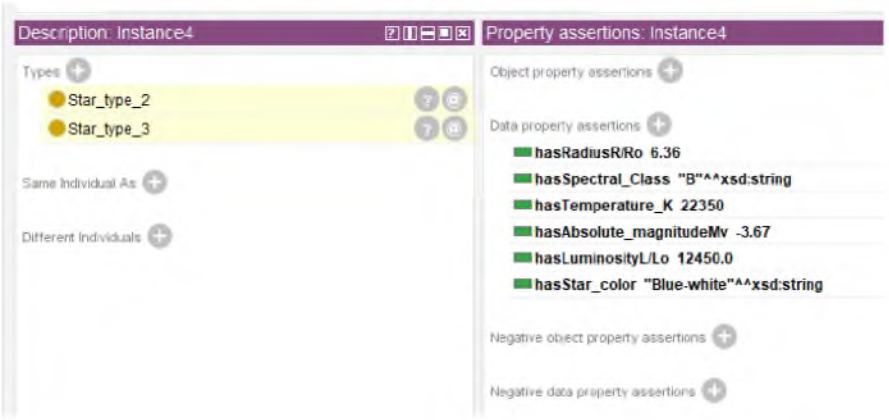


Рисунок 4.26 – Приклад конфліктної ситуації

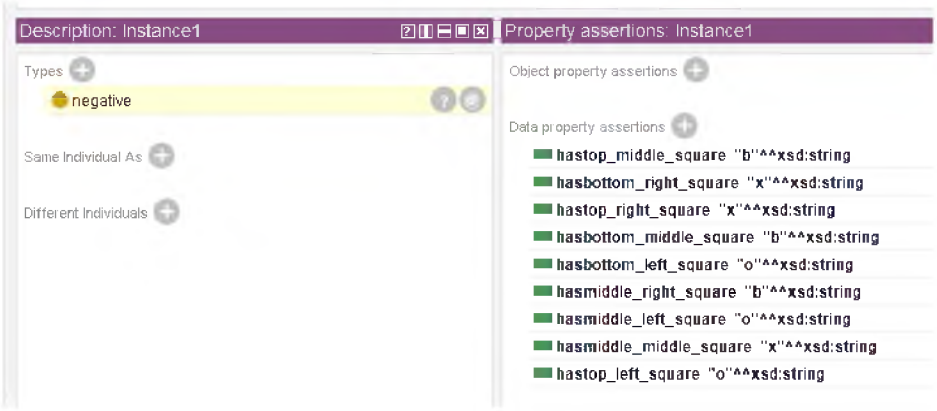


Рисунок 4.27 – Приклад класифікації на одному прикладі з датасету TicTacToe

Отже, в данному розділі була описана основна ідея пропонованого рішення проблеми, описані технічні деталі її рішення, вхідні та вихідні дані, отримані правила та їх точність. Також представлені результати запусків системи, робота з правилами та різонером в онтології.

ВИСНОВКИ

Проблема зрозумілого штучного інтелекту на сьогодні є дуже актуальною, кількість робіт та досліджень в даній області за останні роки значно виросла, в значній мірі цьому сприяло введення права на роз'яснення в загальні правила захисту даних. Для багатьох людей робота нейронної мережі є непрозорою, нейронні мережі можуть робити упереджені висновки, можуть бути обдурені чи робити висновки на основі деталей, які мало помітні для людей. В роботі був проведений аналіз предметної області, актуальність проведення робіт у даній області, її сучасний стан, сформована мета роботи, поставлені задачі, оглянуті існуючі підходи та проблеми, обрані методи для вирішення задач, практично реалізована система яка дозволяє пояснити передбачення, зроблені нейронною мережею як глобально, так і локально.

З теоритичного аналізу можна сказати, що існує багато підходів до пояснення роботи нейронних мереж, кожен з них має свої переваги та недоліки, та дозволяє отримати унікальне уявлення про роботу нейронних мереж з різних сторін, оскільки користувач, ймовірно, зацікавлений як у загальному напрямку, так і в найбільш значних внесках певних ознак у результат. Ще більша кількість методів досліджуються сьогодні, але часто вони є модифікаціями чи розширеннями розглянутих базових підходів. Так як кількість інформації, що може сприймати людина є невеликою, то бажано, щоб пояснення було якомога коротшим. Пояснення за допомогою правил є популярною технікою в області XAI на сьогодні. Відкритими проблемами залишаються питання оцінювання отриманих пояснень та порівняння різних методів, що їх надають. Одним з методів оцінювання отриманих пояснень є оцінка за допомогою експертів у певній предметній області.

Використовуючи сучасні технології та підходи, була розроблена система, що дозволяє отримувати пояснення для роботи нейронної мережі у

вигляді правил. Для цього був використаний алгоритм виведення правил та онтологія. Із переваг отриманої системи можна виділити те, що отримані правила можна застосовувати до нових даних з досить непоганою точністю, використовуючі певні параметри можливо корегувати отриманий набір правил, отримані правила дійсно можуть відображати хід міркування нейронної мережі. Завдяки онтології отримані знання можуть бути скореговані та доповнені інформацією певної предметної області експертом у ній, а також поширені та використані повторно. Завдяки мові SWRL отримані правила можливо застосовувати до нових даних та класифікувати за допомогою механізмів логічного виведення редактору онтологій Protege та розробленого плагіну для нього. Із недоліків можна відзначити, що нажаль в онтології не вдалося релізувати стратегію вирішення конфліктів та правил за замовчуванням. Для більшої довіри до результату при використанні звичайних методів, ніж методів видобування правил, потрібно мати більше даних для навчання. Отримані правила більше націлені на високу точність, ніж на стисле пояснення, не всі атрибути можуть бути представлені у поясненні, дискретизація атрибутів призводить до втрати інформації. Не намагаючись імітувати ШНМ, є велика ймовірність, що нові дані будуть класифіковані по-іншому. Усі переваги та недоліки методів виведення правил також застосовні і в даному випадку.

В якості подальшої роботи можна розглянути сучасні методи індукції правил, такі як IDS, CORELS, методи, що виробляють правила з кількома атомами у консеквенті, що може бути корисно для стислого опису, такі як MIDS та RR, сучасні методи видобування правил, для вирішення поставленої задачі та порівняти результати. Використовувати інші метрики для оцінки отриманих правил та їх зрозумілості. Розробити стратегію вирішення конфліктів в онтологіях. Потрібно більше досліджень для визначення чи можливо довіряти отриманим поясненням, сконцентруватись на дослідженні даних на основі яких надається пояснення, визначити, які залежності в даних можуть перешкодити отримати хороші пояснення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Vilone G., Longo L. Notions of explainability and evaluation approaches for explainable artificial intelligence. Information Fusion. 2021. Vol. 76, P. 89-106.

2. Guidotti R., Monreale A., Ruggieri S., Turini F., Pedreschi D., Giannotti F. A Survey of Methods for Explaining Black Box. ACM Computing Surveys. 2018. Vol. 51.

3. Ivanovs M., Kadikis R., Ozols K. Perturbation-based methods for explaining deep neural networks: A survey. Pattern Recognition Letters. 2021. Vol. 150, P. 228-234.

4. Preece A. Asking 'Why' in AI: Explainability of intelligent systems – perspectives and challenges. Intelligent Systems in Accounting, Finance & Management. 2018. Vol. 25. P. 63-72.

5. Rudin C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence. 2019. Vol. 1. P. 206-215.

6. Burkart N., Huber M. A Survey on the Explainability of Supervised Machine Learning. Journal of Artificial Intelligence Research. 2021. Vol. 70. P. 245–317.

7. Adadi A., Berrada M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). IEEE Access. 2018. Vol. 6. P. 52138-52160.

8. Molnar C. Interpretable machine learning. A Guide for Making Black Box Models Explainable. 2019. URL: https://christophm.github.io/interpretable-ml-book/ (Last accessed: 01.11.2021).

9. O'Sullivan C. Finding and Visualising Non-Linear Relationships. 2021. URL: https://towardsdatascience.com/finding-and-visualising-non-linear-relationships-4ecd63a43e7e (Last accessed: 01.11.2021).

10. Relova Z. Explaining Machine Learning Models: Partial Dependence. 2021. URL: https://towardsdatascience.com/explain-machine-learning-models-partial-dependence-ce6b9923034f (Last accessed: 01.11.2021).

11. Billiau S. From Scratch: Permutation Feature Importance for ML Interpretability. 2021. URL: https://www.kdnuggets.com/2021/06/from-scratch-permutation-feature-importance-ml-interpretability.html (Last accessed: 01.11.2021).

12. Hailesilassie T. Rule Extraction Algorithm for Deep Neural Networks: A Review. International Journal of Computer Science and Information Security. 2016. Vol. 14, No. 7. P. 371-381.

13. Pappa G., Freitas A. Automating the Design of Data Mining Algorithms. Heidelberg: Springer, 2010. 187 p.

14. Clark P., Niblett T. The CN2 induction algorithm. Machine Learning. 1989. Vol. 3. P. 261–283.

15. Huysmans J., Setiono R., Baesens B., Vanthienen J. Minerva: Sequential Covering for Rule Extraction. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). 2008. Vol. 38, No. 2. P. 299-309.

16. Powell J., Hopkins M. Ontologies. A Librarian's Guide to Graphs. Data and the Semantic Web. 2015. P. 31–43

17. Terziyan V., Vitko O. Intelligent information management in mobile electronic commerce. Artificial Intelligence News, Journal of Russian Association of Artificial Intelligence. 2002. Vol. 5.

18. V. Sugumaran Semantic technologies for enhancing knowledge management systems. 2016. P. 203–213.

19. De Keyser, P. Taxonomies and ontologies. Indexing. 2012. P. 121–142.

20. O'Connor M., Knublauch H., Tu S., Grosof B., Dean M., Grosso W., Musen M. Supporting Rule System Interoperability on the Semantic Web with SWRL. ISWC 2005. 2005. Vol. 3729. P. 974-986.