МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка системи підтримки ухвалення рішень для діагностики інфекційних захворювань на основі алгоритму дерева рішень

Виконав студент 2 курсу групи МІС-22
спеціальності 122 Комп'ютерні науки
Скрипнік Михайло Сергійович

Керівник к.ф.-м.н.,
Ткач Тетяна Борисівна

Рецензент к.т.н., доцент
Перелигін Борис Вікторович

Одеса 2023

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API – інтерфейсів прикладного програмування.

GUI – graphical user interface – графічний інтерфейс користувача.

ML – Machine Learning – машинне навчання.

UML – Unified Modeling Language – уніфікована мова моделювання.

ГРВІ – гостра респіраторна вірусна інфекція.

СППР – система підтримки прийняття рішень.

СППЛР – система підтримки прийняття лікарських рішень.

ВСТУП

В даний час багато закладів охорони здоров'я різного масштабу та профілів у всьому світі широко застосовують автоматизацію медичних технологій та відповідних бізнес-процесів. І тому використовуються медичні інформаційні системи. Одним із важливих напрямів є розробка систем підтримки прийняття рішень. СППР використовують для діагностики та прогнозування захворювань. Подібні системи не можуть відповідати за прийняті з її допомогою рішення, але здатні суттєво спростити і полегшити роботу лікаря. Мета СППР полягає у здійсненні кооперації системи та людини у процесі прийняття рішень.

Інтелектуальний аналіз даних використовується для отримання корисної інформації з великих наборів даних і відображення її в простих для інтерпретації візуалізаціях. Дерева рішень, вперше представлені в середині 20 століття, є одним із найефективніших методів інтелектуального аналізу даних, вони широко використовуються у багатьох дисциплінах.

Використання дерев рішень обґрунтовано тим, що вони представлені природною мовою. Завдяки цьому така модель інтуїтивно зрозуміла та легко інтерпретується користувачем, на відміну, наприклад, від нейронних мереж.

До переваг дерев рішень можна віднести точність, яка можна порівняти з іншими методами побудови кваліфікаційних моделей. Ще одна перевага зменшення часу на побудову класифікаційних моделей за допомогою алгоритмів конструювання дерев рішень. Актуальність розробки систем підтримки прийняття рішень обумовлена виникненням лікарських помилок, які можуть бути з різних причин. У тому числі обмеженість часових ресурсів, недостатність знань, відсутність можливості залучення компетентних експертів, неповнота інформації про стан пацієнта. Зазначені фактори можуть призвести до серйозних проблем зі здоров'ям у майбутньому. У зв'язку з цим загальним завданням для всіх СППР стає прийняття «персоналізованого» рішення про траєкторію лікування кожного пацієнта.

Метою магістерської роботи є розробка системи підтримки прийняття рішень для діагностики інфекційних захворювань на основі алгоритму дерева рішень.

Об'єкт дослідження – процес розробки та дослідження системи підтримки прийняття рішень для діагностики інфекційних захворювань.

Предмет дослідження – автоматизація алгоритму генерації дерева рішень.

Задачі дослідження: провести аналіз предметної області, визначити вимоги до системи, спроектувати основні процеси системи та реалізувати конфігурацію системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Дерева рішень: загальні принципи

Дерева рішень є потужним та популярним інструментом. Вони зазвичай використовуються аналітиками даних щодо прогнозного аналізу (наприклад, розробки операційних стратегій у компаніях). Також вони є популярним інструментом для машинного навчання та штучного інтелекту, де вони використовуються як навчальні алгоритми для навчання з учителем (тобто категоризації даних на основі різних тестів, таких як класифікатори «так» або «ні»).

У широкому значенні дерева рішень використовуються в різних галузях для вирішення багатьох типів проблем. Через свою гнучкість вони використовуються у різних секторах, від технологій та охорони здоров'я до фінансового планування.

Розглянемо основні терміни, які потрібно знати при роботі з деревами рішень:

- вузли: існує три типи вузлів. Кореневий вузол, також званий вузлом прийняття рішення, представляє вибір, який призведе до підрозділу всіх записів на два або більше взаємовиключних підмножини. Внутрішні вузли, також звані випадковими вузлами, є одним із можливих варіантів вибору, доступних у цій точці деревоподібної структури. Верхній край вузла з'єднується з його батьківським вузлом, а нижній край з'єднується з дочірніми вузлами або кінцевими вузлами. Листові вузли, також звані кінцевими вузлами, є кінцевим результатом комбінації рішень або подій;
- гілки є випадковими результатами або подіями, що виходять з кореневих вузлів та внутрішніх вузлів. Модель дерева рішень формується за допомогою ієрархії гілок. Кожен шлях від кореневого вузла через внутрішні вузли до кінцевого вузла є правилом

прийняття рішення про класифікацію. Ці шляхи дерева рішень також можуть бути представлені як правила «якщо то». Наприклад, «якщо виконуються умова 1, умова 2, умова … і умова k, відбувається вихід j»;

- поділ – це процес поділу вузла на два або більше підвузлів;
- вузол прийняття рішення: коли підвузил поділяється на додаткові підвузли, він називається вузлом прийняття рішення;
- листовий (кінцевий) вузол – вузли, які не поділяються;
- скорочення: коли ми видаляємо підвузли вузла прийняття рішень, цей процес називається скороченням. Можна сказати зворотний процес розщеплення;
- гілка (піддерево) – частина всього дерева;
- батьківський та дочірній вузол – вузол, який розділений на підвузли, називається батьківським вузлом підвузлів, тоді як підвузли є дочірніми елементами батьківського вузла.

Дерева рішень класифікують приклади, сортуючи їх вниз по дереву від кореня до деякого кінцевого вузла, при цьому кінцевий вузол надає класифікацію для прикладу [1].

Кожен вузол у дереві діє як тестовий приклад для деякого атрибуту, і кожне ребро, що спускається з цього вузла, відповідає одній з можливих відповідей тестовий приклад. Цей процес має рекурсивний характер і повторюється для кожного піддерева, що укорінився в нових вузлах.

Розглянемо приклад бінарного дерева для передбачення, чи підходить людина чи ні, надаючи різну інформацію, таку як харчові звички та фізичні вправи (рис.1).

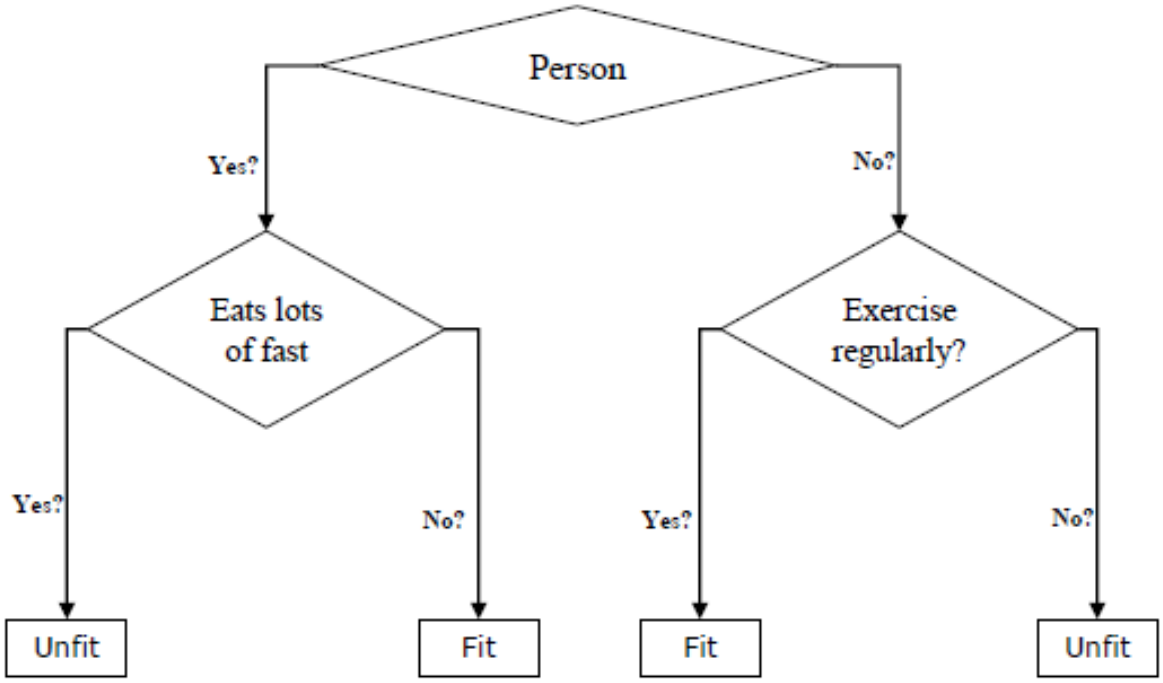

Рисунок 1 – Дерево рішень

У наведеному дереві рішень питання – це вузли рішення, а кінцеві результати – листя.

Розглянемо переваги та недоліки дерева рішень.

1. До переваг дерев рішень відносять наступні пункти.
2. Дерева рішень формують чіткі та зрозумілі правила класифікації. Наприклад, «якщо вік < 40 і немає майна для застави, то відмовити у кредиті». Тобто дерева рішень добре та швидко інтерпретуються.
3. Здатні генерувати правила в областях, де фахівцю важко формалізувати свої знання.
4. Легко візуалізуються, тобто можуть «інтерпретуватися» не лише як модель загалом, а й як прогноз для окремого тестового суб'єкта (шлях у дереві).
5. Швидко навчаються та прогнозують.
6. Не потрібно багато параметрів моделі.
7. Підтримують як числові, і категоріальні ознаки.

До недоліків можна віднести такі пункти.

1. Дерева рішень чутливі до шумів у вхідних даних. Невеликі зміни навчальної вибірки можуть призвести до глобальних коригувань моделі, що позначиться зміні правил класифікації та інтерпретованості моделі.
2. Розділяюча межа має певні обмеження, через що дерево рішень щодо якості класифікації поступається іншим методам.
3. Можливе перенавчання дерева рішень, через що доводиться вдаватися до методу «відсікання гілок», встановлення мінімальної кількості елементів у листі дерева або максимальної глибини дерева.
4. Складний пошук оптимального дерева рішень: це призводить до необхідності використання евристики типу жадібного пошуку ознаки з максимальним приростом інформації, які зрештою не дають 100% гарантії знаходження оптимального дерева.
5. Дерево рішень робить константний прогноз для об'єктів, що знаходяться в ознаковому просторі поза паралелепіпедом, який охоплює не всі об'єкти навчальної вибірки [2].

Модулі для побудови та дослідження дерев рішень входять до складу безлічі аналітичних платформ. Це зручний інструмент, який застосовується в системах підтримки прийняття рішень та інтелектуального аналізу даних.

Найуспішніше дерева застосовують у наступних областях:

1. Банківська справа. Оцінка кредитоспроможності клієнтів банку під час видачі кредитів.
2. Промисловість. Контроль якості продукції (виявлення дефектів у готових товарах), випробування без порушень (наприклад, перевірка якості зварювання) тощо.
3. Медицина. Діагностика захворювань різної складності.
4. Молекулярна біологія. Аналіз будови амінокислот.
5. Торгівля. Класифікація клієнтів та товару.

Це не вичерпний список сфер застосування дерева рішень. Коло використання постійно розширюється, а дерева рішень поступово стають важливим інструментом управління бізнес-процесами та підтримки прийняття рішень.

1.1.1 Аналіз дерева рішень

Аналіз дерева рішень – це загальний інструмент прогнозного моделювання, який має програми, що охоплюють низку різних галузей. Як правило, дерева рішень будуються за допомогою алгоритмічного підходу, який визначає способи розподілу набору даних на основі різних умов. Це один із найбільш широко використовуваних та практичних методів навчання з учителем.

Дерева рішень – це непараметричний контрольований метод навчання, що використовується як задач класифікації, так задач регресії. Мета полягає в тому, щоб створити модель, яка передбачає значення цільової змінної, вивчаючи прості правила прийняття рішень, виведені з характеристик даних. Правила прийняття рішень здебільшого мають форму операторів if-then-else. Чим глибше дерево, тим складніше правила і точніше модель.

Дерево рішень – це деревоподібний граф з вузлами, що становлять місця, де ми вибираємо атрибут і ставимо питання; ребра репрезентують відповіді на запитання; а листя представляють фактичний висновок або мітку класу. Вони використовуються при нелінійному прийнятті рішень із простою лінійною поверхнею прийняття рішень [3].

Загальні способи використання моделей дерева рішень включають:

- вибір змінних. Кількість змінних, які регулярно відстежуються у клінічних умовах, різко збільшилася із введенням електронного сховища даних. Багато з цих змінних мають незначне значення, і тому їх, ймовірно, не слід включати до вправ з інтелектуального аналізу даних. Як і покроковий вибір змінних у регресійному аналізі,

методи дерева рішень можуть використовуватися для вибору найбільш підходящих вхідних змінних, які слід використовувати для формування моделей дерева рішень, які можна використовувати для формулювання клінічних гіпотез та інформування наступних досліджень;

- оцінка відносної важливості змінних. Щойно набір релевантних змінних визначено, дослідники можуть захотіти дізнатися, які змінні грають основну роль. Як правило, значення змінної обчислюється на основі зниження точності моделі (або чистоти вузлів у дереві) при видаленні змінної. Найчастіше, що більше записів впливає змінну, то вище її важливість;
- обробка пропущених значень. Звичайний, але неправильний метод обробки відсутніх даних полягає у виключенні випадків із відсутніми значеннями. Це неефективно і може спричинити систематичну помилку в аналізі. Аналіз дерева рішень може працювати з відсутніми цими двома способами. Перший спосіб може класифікувати відсутні значення як окрему категорію, яку можна аналізувати разом з іншими категоріями. Другий спосіб може використовувати побудовану модель дерева рішень, яка встановлює змінну з великою кількістю відсутніх значень як цільової змінної, щоб зробити прогноз та замінити ці відсутні прогнозованим значенням;
- прогноз. Це одне з найважливіших застосувань моделей дерева рішень. Використовуючи деревоподібну модель, отриману з історичних даних, легко передбачити результат майбутніх записів;
- маніпуляції з даними. У медичних дослідженнях поширено дуже багато категорій однієї категоріальної змінної чи сильно спотворених безперервних даних. У цих обставинах моделі дерева рішень можуть допомогти в прийнятті рішення про те, як найкраще згорнути

категоріальні змінні в більш керовану кількість категорій або як розділити перекручені змінні на діапазони.

1.1.2 Виразність дерев рішень

Дерева рішень можуть представляти будь-яку логічну функцію вхідних атрибутів.

Скористаємось деревами рішень, щоб виконати функцію трьох логічних операторів AND, OR та XOR.

Представимо на рисунках таблицю істинності даних операторів та дерева рішень для них.

Бульова функція AND зображена на рис.1. Бачимо, що є дві концепції-кандидата створення дерева рішень, що виконує операцію AND. Так само ми можемо створити дерево рішень, яке виконує логічну операцію OR.

Логічна функція OR зображено на рис.3.

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T

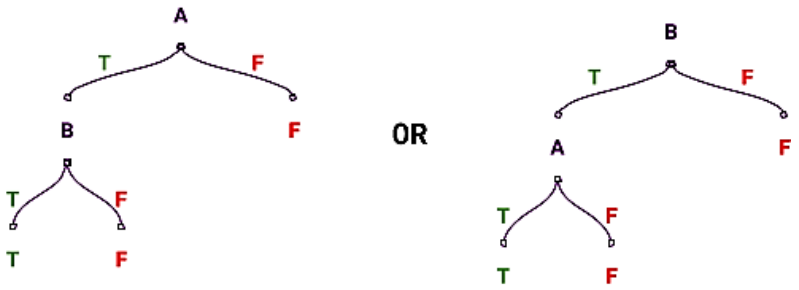

Рисунок 2 – Дерево рішень для операції AND

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

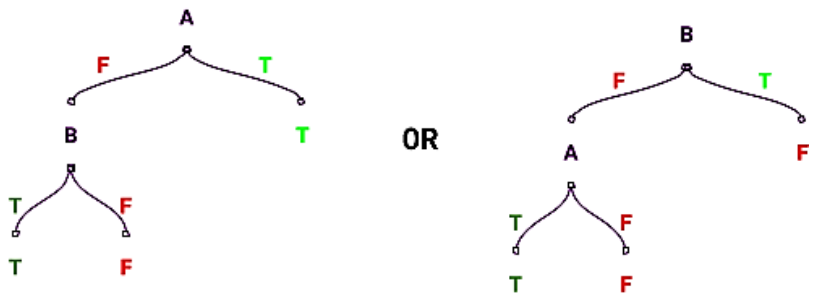

Рисунок 3 – Дерево рішень для операції OR

Далі розглянемо ще одну логічну функцію. Так само, як і за двома попередніми функціями, ми можемо створити дерево рішень, яке виконуватиме необхідну логічну операцію.

Логічна функція: XOR зображено на рис.4. Бачимо, що саме дерева рішень дуже наглядно представляють результат.

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

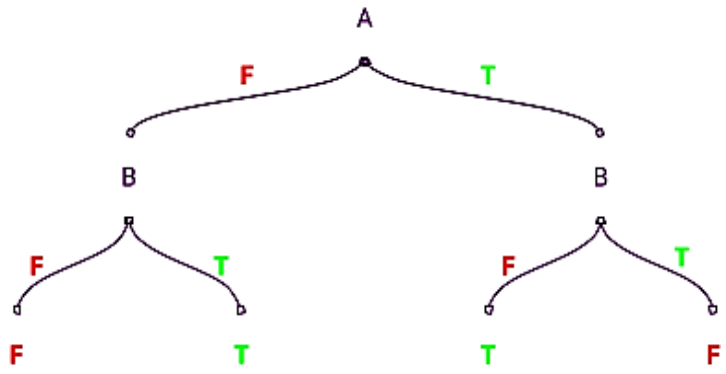

Рисунок 4 – Дерево рішень для операції XOR

Далі для більшої наочності створимо дерево рішень, що виконує функцію XOR, використовуючи 3 атрибути. Результат процесу зображено на рис.5.

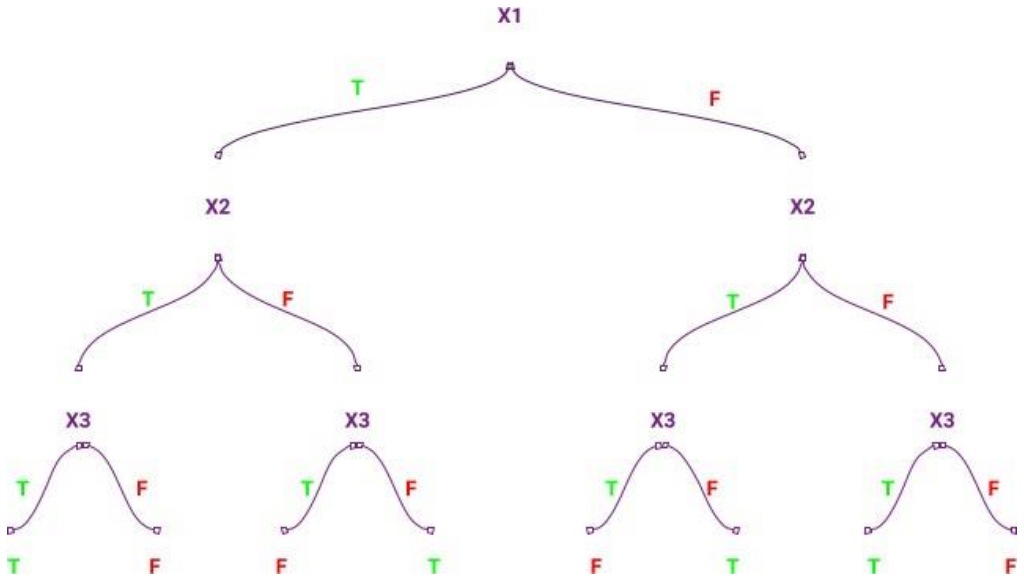

Рисунок 5 – Дерево рішень для операції XOR з трьома операндами

У дереві рішень, наведеному вище, для трьох ознак є 7 вузлів у дереві, тобто при n = 3 кількість вузлів = 2^3-1. Так само, якщо ми маємо n атрибутів, у дереві рішень буде 2^n вузлів (приблизно). Таким чином, дерево потребує експоненційної кількості вузлів у найгіршому випадку.

Ми можемо уявити логічні операції, використовуючи дерева рішень. Але які ще функції ми можемо уявити і про скільки дерев рішень нам доведеться турбуватися? Відповімо на це питання, з'ясувавши можливу кількість дерев рішень, які ми можемо згенерувати із заданими N різними атрибутами (за умови, що атрибути булеві). Оскільки таблицю істинності можна перетворити на дерево рішень, сформуємо таблицю істинності з атрибутів N в якості вхідних даних, наведених на таблиці 1.

Наведена вище таблиця 1 містить 2^n рядків (тобто кількість вузлів у дереві рішень), що представляє можливі комбінації вхідних атрибутів, і оскільки кожен вузол може містити двійкове значення, кількість способів заповнити значень у дереві рішень {2^{2 ^n}}. Отже, простір дерев рішень, т. е. простір гіпотез дерева рішень, дуже виразно, оскільки може представляти

безліч різних функцій. Але це також означає, що потрібно мати хитрий спосіб пошуку найкращого дерева серед них [4].

Таблиця 1 – Таблиця істинності з N атрибутів

X1	X2	X3	....	XN	OUTPUT
T	T	T	...	T	…
T	T	T	...	F	…
...	...	...	...	...	…
F	F	F	...	F	…

Варто відзначити різницю у вищеописаних деревах.

Так останнім описано бінарне дерево. Отже, хоч це і найкращий поділ, але на практиці найчастіше застосовується досить рідко. І для побудови використовуються інші алгоритми, ніж для побудови дерев рішення, в яких з вузла виходить, не два, а більше гілок. Також різні алгоритми застосовують у деревах рішень, у яких функції безперервні.

Розглянемо також, як відбувається вироблення рішення за умов ризику.

Умови ризику та невизначеності характеризуються так званими умовами багатозначних очікувань майбутньої ситуації у зовнішньому середовищі. У цьому випадку особа, яка приймає рішення, має зробити вибір альтернативи (не маючи точного уявлення про фактори зовнішнього середовища та їх вплив на результат).

Для представлення та аналізу результатів обраних альтернативних стратегій використовують матрицю рішень, яку називають платіжною матрицею.

1.1.3 Методи та алгоритми вирішення задач за допомогою дерева рішень

Завданнями, які вирішуються за допомогою дерева рішень, є:

- класифікація – віднесення об'єктів до одного із заздалегідь відомих класів. Цільова змінна повинна мати дискретні значення;
- регресія (чисельне передбачення) – передбачення числового значення незалежної змінної для заданого вхідного вектора;
- опис об'єктів – набір правил у дереві рішень дозволяє компактно описувати об'єкти. Тому замість складних структур, що описують об'єкти, можна зберігати дерева рішень.

Розглянемо процес побудови дерев рішень.

Процес побудови дерев рішень полягає в послідовному, рекурсивному розбиття навчальної множини на підмножини із застосуванням вирішальних правил у вузлах. Процес розбиття триває доти, доки всі вузли наприкінці всіх гілок не будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він міститиме єдиний об'єкт, або об'єкти лише одного класу), або після досягнення певної умови зупинки, що задається користувачем (наприклад, мінімально допустима кількість прикладів у вузлі або максимальна глибина дерева).

Алгоритми побудови дерев рішень відносять до категорії про жадібних алгоритмів. Згадаємо, що жадібними називаються алгоритми, які допускають, що локально-оптимальні рішення на кожному кроці (розбиття у вузлах), призводять до оптимального підсумкового рішення. У випадку дерев рішень це означає, що якщо один раз був обраний атрибут, і по ньому було розбито на підмножини, то алгоритм не може повернутися назад і вибрати інший атрибут, який дав би найкраще підсумкове розбиття. Тому на етапі побудови не можна сказати, чи забезпечить обраний атрибут, зрештою, оптимальне розбиття.

В основі більшості популярних алгоритмів навчання дерев рішень лежить принцип «поділяй і володарюй». Алгоритмічно цей принцип реалізується в такий спосіб. Нехай задано навчальну множину S, що містить n прикладів, для кожного з яких задана мітка класу Ci (i=1..k), і m атрибутів Aj (j=1..m), які, як передбачається, визначають приналежність об'єкта до того чи іншого класу. Тоді можливі три випадки:

- усі приклади множини S мають однакову мітку класу Ci (тобто всі навчальні приклади відносяться лише до одного класу). Очевидно, що навчання в цьому випадку не має сенсу, оскільки всі приклади моделі, що пред'являються, будуть одного класу, який і «навчиться» розпізнавати модель. Саме дерево рішень у цьому випадку буде листом, асоційованим з класом Ci. Практичне використання такого дерева безглуздо, оскільки будь-який новий об'єкт воно відноситиме лише до цього класу;
- більшість S взагалі не містить прикладів, тобто є порожнім безліччю. У цьому випадку для неї теж буде створено лист (застосовувати правило, щоб створити вузол, до порожньої множини безглуздо), клас якого буде обраний з іншої множини (наприклад, клас, який найчастіше зустрічається в батьківській множині);
- безліч S містить навчальні приклади всіх класів Ck. В цьому випадку потрібно розбити безліч S на підмножини, асоційовані з класами. Для цього вибирається один з атрибутів Aj множини S, який містить два і більше унікальних значень (a1,a2,...,ap), де p – число унікальних значень ознаки.Потім безліч S розбивається на p підмножин (S1, S2, ..., Sp), кожне з яких включає приклади, що містять відповідне значення атрибуту. Потім вибирається наступний атрибут і повторюється розбиття. Ця процедура буде рекурсивно повторюватиметься до тих пір, поки всі приклади в результуючих підмножинах не виявляться одного класу.

Описана вище процедура є основою багатьох сучасних алгоритмів побудови дерев рішень. Очевидно, що при використанні цієї методики, побудова дерева рішень відбуватиметься зверху донизу (від кореневого вузла до листя).

На сьогоднішній день існує чимало алгоритмів навчальних рішень, але найбільш популярними вважаються: ID3, CART, C4.5.

Опишемо коротко кожний із них.

ID3 (ітеративний дихотомайзер 3) був розроблений у 1986 році Россом Куїнланом. Алгоритм створює багатоходове дерево, знаходячи для кожного вузла (тобто жадібним способом) категоричну ознаку, яка дасть найбільший приріст інформації для категоріальних цілей. Дерева виростають до максимального розміру, а потім зазвичай застосовують етап обрізки, щоб поліпшити здатність дерева узагальнювати невидимі дані. ID3 вважається дуже простим алгоритмом дерева рішень. Він визначає класифікацію об'єктів чи записів шляхом перевірки значень їх атрибутів. Він будує дерево рішень для заданих даних у низхідній структурі, починаючи з набору записів та набору атрибутів. У кожному вузлі дерева перевіряється один атрибут на основі максимізації виміру приросту інформації та мінімізації виміру ентропії, а результат використовується для розділення записів. Цей процес рекурсивно виконується до того часу, поки записи, дані у піддереві, стануть однорідними (всі записи належать одному й тому класу). Ці однорідні записи стають кінцевими вузлами дерева рішень.

C4.5 є наступником ID3 і прибрав обмеження, згідно з яким об'єкти повинні бути категоріальними шляхом динамічного визначення дискретного атрибута (на основі числових змінних), який розбиває безперервне значення атрибута на дискретний набір інтервалів. C4.5 перетворює навчені дерева (тобто вихідні дані алгоритму ID3) на набори правил «якщось». Ця точність кожного правила потім оцінюється для визначення порядку, в якому вони повинні застосовуватися. Скорочення виконується шляхом видалення попередньої умови правила, якщо точність правила покращується без нього.

Алгоритм C4.5 використовується в інтелектуальному аналізі даних як класифікатор дерева рішень, який можна використовувати для генерації рішення на основі певної вибірки даних (одномірні або багатовимірні предиктори). Алгоритм C4.5 генерує дерево рішень для заданого набору даних шляхом рекурсивного розділення записів. Алгоритм C4.5 враховує категоріальні та числові атрибути. Для кожного категоріального атрибуту C4.5 обчислює приріст інформації та вибирає той, який має найбільше значення, та використовує атрибут для отримання множини результатів у вигляді числа різних значень цього атрибуту.

CART (дерева класифікації та регресії) дуже схожий на C4.5, але відрізняється тим, що підтримує числові цільові змінні (регресія) та не обчислює набори правил. CART будує бінарні дерева, використовуючи функцію та поріг, які дають найбільший приріст інформації у кожному вузлі. CART – корисний непараметричний метод, який можна використовувати для пояснення безперервної або категоріальної залежної змінної з погляду кількох незалежних змінних. Незалежні змінні можуть бути безперервними чи категоріальними. CART використовує підхід до поділу, широко відомий як «розділяй і володарюй».

Алгоритм CART працює, щоб знайти незалежну змінну, яка створює найкращу однорідну групу під час поділу даних. Для завдання класифікації, де змінна відповіді є категоріальною, це вирішується шляхом обчислення інформації, одержаної на основі ентропії, одержаної в результаті поділу. Для числової відповіді однорідність вимірюється статистичними даними, такими як стандартне відхилення чи дисперсія.

Двома важливими параметрами методу CART є критерій мінімального поділу та параметр складності (Cp). Критерій мінімального поділу – це мінімальна кількість записів, які мають бути присутніми у вузлі, перш ніж можна буде спробувати виконати поділ. Це має бути зазначено на початку. Cp – це параметр складності, що дозволяє уникнути поділу тих вузлів, які явно не потрібні. Інший спосіб розгляду цих параметрів полягає в тому, що

значення Ср визначається після вирощування дерева, а оптимальне значення використовується для обрізки дерева.

Алгоритм CART продовжує ділити набір даних до того часу, поки у кожному «листовому» вузлі залишиться мінімальна кількість записів, як зазначено критерієм мінімального поділу. В результаті виходить деревоподібна структура. Потім значення Ср наноситься на графік щодо різних рівнів дерева, оптимальне значення використовується для обрізки дерева.

Дані алгоритми розв'язання задач за допомогою дерева рішень є найпоширенішими, але це не означає, що кожен з них ідеально підійде для будь-якого завдання. Необхідно пам'ятати, що кожен випадок і кожна задача індивідуальні та вимагають попереднього аналізу та персонального підбору алгоритму розв'язання.

Розберемо основні етапи побудови дерева рішень.

Для того, щоб побудувати дерево рішень, необхідно вирішити кілька завдань, які пов'язані з відповідним кроком процесу навчання:

- вибір атрибуту, за яким буде проводитися розбиття в даному вузлі (атрибута розбиття);
- вибір критерію зупинення навчання;
- вибір методу відсікання гілок (спрощення);
- оцінка точності побудованого дерева.

Розглянемо представлені етапи більш докладніше.

Вибір атрибута розбиття.

Під час формування правила для розбиття в черговому вузлі дерева необхідно вибрати атрибут, за яким це буде зроблено. Загальне правило для цього можна сформулювати наступним чином: обраний атрибут повинен розбити безліч спостережень у вузлі так, щоб підмножини, що результують, містили приклади з однаковими мітками класу, або були максимально наближені до цього, тобто кількість об'єктів з інших класів (домішок) у кожному з цих множин було якнайменше. Для цього було обрано різні

критерії, найбільш популярними з яких стали теоретико-інформаційний та статистичний.

Теоретико-інформаційний критерій.

Як випливає з назви, критерій ґрунтується на поняттях теорії інформації, а саме – інформаційної ентропії.

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right), \quad (1.1)$$

де n – число класів у вихідній підмножині;

Ni – число прикладів і-го класу;

N – загальна кількість прикладів у підмножині.

Ентропія – це спосіб виміру невизначеності класу в підмножині прикладів. Припустимо, що елемент належить підмножини S, що має два класи: позитивний та негативний. Ентропія визначається як N бітів, необхідних, щоб сказати, чи є х позитивним чи негативним.

Ентропія завжди дає число від 0 до 1. Тому, якщо підмножина, сформована після поділу з використанням атрибуту, є чистою, то нам знадобляться нульові біти, щоб сказати, чи є вона позитивною чи негативною. Якщо сформоване підмножина має рівні N позитивних і негативних елементів, то необхідних бітів буде 1 як зображено рисунку 6.

Наведений графік показує відношення між ентропією та ймовірністю позитивного класу. Як бачимо, ентропія досягає 1, що є максимальним значенням, коли є рівні шанси елемента бути позитивним чи негативним. Ентропія мінімальна, коли p(+) прагне нуля (символізуючи х негативно) чи 1 (символізуючи х позитивно).

Ентропія говорить нам, наскільки чистим чи нечистим є кожне підмножина після поділу. Що нам потрібно зробити, так це агрегувати ці оцінки, щоб перевірити, чи можливий поділ чи ні. Це робиться шляхом отримання інформації.

Таким чином, ентропія може розглядатися як міра неоднорідності підмножини за представленими у ньому класами. Коли класи представлені у рівних частках та невизначеність класифікації найбільша, ентропія також максимальна. Якщо приклади у вузлі ставляться одного класу, тобто N=Ni, логарифм від одиниці перетворює ентропію в нуль.

Найкращим атрибутом розбиття Aj буде той, який забезпечить максимальне зниження ентропії результуючого підмножини щодо батьківського. На практиці, однак, говорять не про ентропію, а про величину, обернену їй, яка називається інформацією.

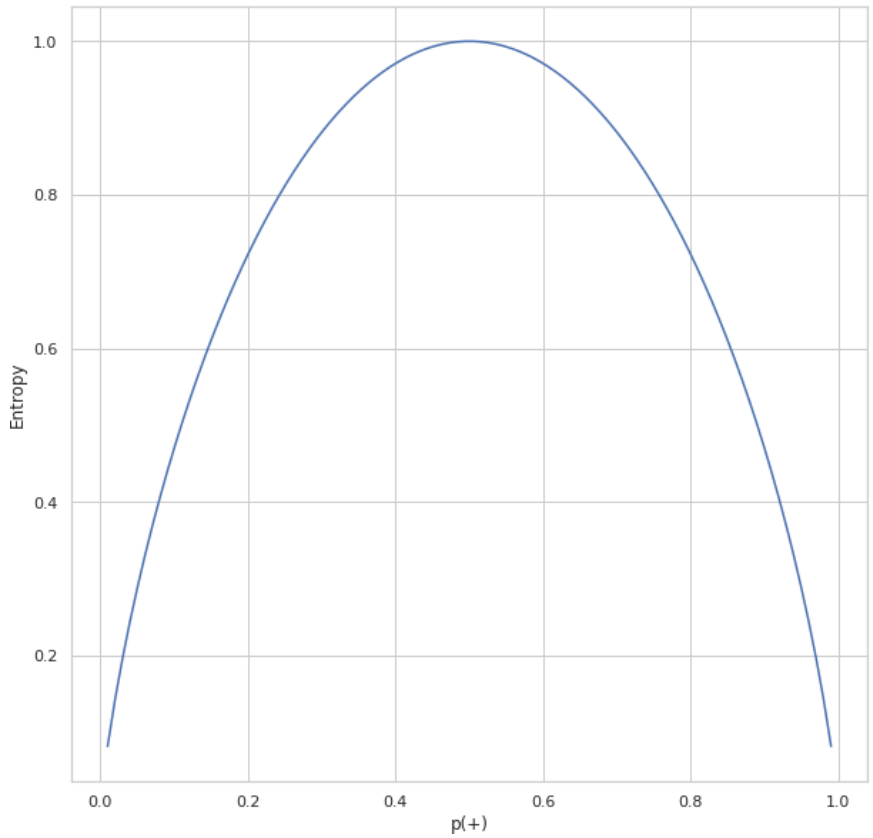

Рисунок 6 – Ентропія

Тоді найкращим атрибутом розбиття буде той, який забезпечить максимальний приріст інформації результуючого вузла щодо вихідного:

$$Gain(A) = Info(S) - Info(SA), \quad (1.2)$$

де Info(S) – інформація, пов'язана з підмножиною S до розбиття;

Info(SA) – інформація, пов'язана з підмножиною, отриманими при розбиття за атрибутом A.

Таким чином, завдання вибору атрибуту розбиття у вузлі полягає в максимізації величини Gain(A), яка називається приростом інформації (від англ. gain – приріст, збільшення). Тому сам теоретико-інформаційний підхід відомий як критерій приросту інформації. Він уперше був застосований в алгоритмі ID3, а потім у C4.5 та інших алгоритмах.

Також як критерій розщеплення використовують індекс Gini. Він був запропонований Брейманом (Breiman) і реалізований в алгоритмі CART. Індекс дозволяє вибирати атрибут, ґрунтуючись на відстані між розподілами класів.

Нехай дано безліч T, яке включає приклади з n класів, тоді індекс Gini визначається за формулою (1.3):

$$Gini(T) = 1 - \sum_{i=1}^{n} p_i^2 , \quad (1.3)$$

де T – поточний вузол;

pi – ймовірність класу i у вузлі T;

n – кількість класів.

Дерева рішень, у яких описано досить багато окремих випадків називаються «гіллястими». Вони складаються з невиправдано великої кількості внутрішніх вузлів та гілок. Такі дерева розбивають вихідну множину на велику кількість підмножин, які складаються з малої кількості об'єктів. Якщо в дереві рішень описано досить багато окремих випадків, то в такому дереві мала кількість об'єктів потраплятиме в кожен окремий випадок. У результаті «наповненість» таких дерев зменшує здатність узагальнювати, а побудовані моделі часто видаватимуть невірні рішення.

Оптимальним за розміром деревом рішення вважається таке дерево, яке буде досить складним, щоб враховувати інформацію з вихідного набору даних, але досить простим. Тобто дерево має брати до уваги лише ті вхідні атрибути, які впливають на точність моделі, а решту ігнорувати.

Критерій зупинення алгоритму.

Теоретично, алгоритм навчання дерева рішень працюватиме доти, доки в результаті не буде отримано абсолютно «чистих» підмножин, у кожному з яких будуть приклади одного класу. Правда, можливо, при цьому буде побудовано дерево, в якому для кожного прикладу буде створено окремий лист. Очевидно, що таке дерево виявиться марним, оскільки воно буде перенавченим – кожному прикладу відповідатиме свій унікальний шлях у дереві, а, отже, і набір правил, актуальний тільки для цього прикладу.

Перенавчання у випадку дерева рішень веде до тих самих наслідків, що й для нейронної мережі – точне розпізнавання прикладів, що беруть участь у навчанні, та повна неспроможність нових даних. Крім цього, перенавчені дерева мають дуже складну структуру і тому їх складно інтерпретувати.

Очевидним вирішенням проблеми є примусова зупинка побудови дерева, доки воно не стало перенавченим. Для цього розроблено наступні підходи:

- рання зупинка – алгоритм буде зупинено, як тільки буде досягнуто заданого значення деякого критерію, наприклад, відсоткової частки правильно розпізнаних прикладів. Єдиною перевагою підходу є зниження часу навчання. Головним недоліком є те, що рання зупинка завжди робиться на шкоду точності дерева, тому багато авторів рекомендують віддавати перевагу відсіканню гілок;
- обмеження глибини дерева – завдання максимальної кількості розбиття у гілках, після досягнення якого навчання зупиняється. Цей метод також веде до зниження точності дерева;
- завдання мінімально припустимого число прикладів у вузлі – заборонити алгоритму створювати вузли з числом прикладів менше

заданого (наприклад, 5). Це дозволить уникнути створення тривіальних розбиття і, відповідно, незначних правил.

Усі перелічені підходи є евристичними, тобто не гарантують кращого результату чи взагалі працюють лише у якихось окремих випадках. Тому до використання слід підходити з обережністю. Яких-небудь обґрунтованих рекомендацій щодо того, який метод краще працює, нині теж не існує. Тому аналітикам доводиться використовувати метод спроб і помилок.

Відсікання гілок.

Як було зазначено вище, якщо «зростання» дерева не обмежити, то в результаті буде побудовано складне дерево з великою кількістю вузлів та листя. Як наслідок воно буде важко інтерпретованим. У той самий час вирішальні правила таких деревах, створюють вузли, у яких потрапляють два-три приклади, виявляються малозначними з практичної погляду.

«Набагато краще мати дерево, що складається з малої кількості вузлів, яким відповідало б велику кількість прикладів з навчальної вибірки. Тому цікавий підхід, альтернативний ранній зупинці – побудувати всі можливі дерева і вибрати їх, яке за розумної глибині забезпечує прийнятний рівень помилки розпізнавання, тобто знайти найбільш вигідний баланс між складністю та точністю дерева.

Альтернативним підходом є так зване відсікання гілок (pruning). Він містить такі кроки:

- побудувати повне дерево (щоб усе листя містило приклади одного класу);
- визначити два показники: відносну точність моделі – відношення числа правильно розпізнаних прикладів до загального числа прикладів, та абсолютну помилку – число неправильно класифікованих прикладів;
- видалити з дерева листя та вузли, відсікання яких не призведе до значного зменшення точності моделі або збільшення помилки.

Відсікання гілок, очевидно, виробляється у напрямі, протилежному напрямку зростання дерева, тобто знизу-вгору, шляхом послідовного перетворення вузлів на листя. Перевагою відсікання гілок у порівнянні з ранньою зупинкою є можливість пошуку оптимального співвідношення між точністю та зрозумілістю дерева. Недоліком є більший час навчання через необхідність спочатку збудувати повне дерево.

Альтернативний метод запобігання переоснащенню полягає в тому, щоб спробувати зупинити процес побудови дерева на ранній стадії, перш ніж він створить листя з дуже маленькими вибірками. Ця евристика відома як рання зупинка, але іноді також відома як попереднє обрізання дерев рішень.

На кожному етапі розбиття дерева ми перевіряємо помилку перехресної перевірки. Якщо помилка не зменшується значно, ми зупиняємося. Рання зупинка може бути недостатньою через занадто ранню зупинку. Поточний спліт може принести мало користі, але зробивши його, наступні спліти значно зменшать помилку.

Вилучення правил.

Іноді навіть спрощене дерево рішень все ще є надто складним для візуального сприйняття та інтерпретації. У цьому випадку може виявитися корисним витягти з дерева вирішальні правила та організувати їх у набори, що описують класи.

Для отримання правил потрібно відстежити всі шляхи від кореневого вузла до листя дерева. Кожен такий шлях дасть правило, що складається з безлічі умов, що є перевіркою в кожному вузлі шляху [4].

1.1.4 Класифікація за допомогою дерева рішень

Суть завдання класифікації полягає у розподілі вихідної множини об'єктів за заздалегідь відомими класами. Приналежність об'єктів з вихідної множини до класів відома, класи інших об'єктів невідомі. Необхідно побудувати алгоритм, який класифікуватиме випадковий об'єкт.

Класифікувати об'єкт – означає вказати номер класу, до якого належить даний об'єкт.

Класифікація об'єкта – це вказівка номера чи назви класу, якого належить даний об'єкт. Клас визначається за допомогою раніше побудованого алгоритму класифікації в результаті його застосування до цього об'єкта.

У математичній статистиці завдання класифікації є завданнями дискримінантного аналізу. У машинному навчанні завдання класифікації вирішується, наприклад, методом штучних нейронних мереж.

У машинному навчанні методи прогнозування зазвичай називаються контрольованим навчанням (навчання з учителем). Такий метод протистоїть неконтрольованого навчання (навчання без вчителя), яке відноситься до моделювання розподілу екземплярів.

Для вирішення завдань класифікації проводиться навчання з учителем. Навчання без вчителя застосовується при вирішенні завдань кластеризації та таксономії. У цих завданнях об'єкти, які у навчальної вибірки не поділені на класи, і потрібно класифікувати об'єкти, враховуючи лише їх подібності друг з одним. Завдання кластеризації та завдання класифікації є близькими один до одного. Тому їх часто плутають у певних прикладних областях, наприклад, у математичній статистиці.

Алгоритм дерева рішень найпоширеніший серед методів вирішення завдання класифікації. В інтелектуальному аналізі даних дерево рішень – це прогностична модель, яка може використовуватися для подання як класифікаторів, так і регресійних моделей. З іншого боку, у дослідженнях операцій дерева рішень відносяться до ієрархічної моделі рішень та їх наслідків. Особа, яка приймає рішення, використовує дерева рішень для визначення стратегії, найбільш вірогідної досягнення її мети. Коли дерево рішень використовується для класифікаційних завдань, його більш доречно називати деревом класифікації. Коли воно використовується для завдань регресії, називається деревом регресії.

Дерева класифікації використовуються для класифікації об'єкта або екземпляра (наприклад, страхувальника) в заздалегідь визначений набір класів (наприклад, ризикований, не ризикований) на основі їх атрибутів (наприклад, вік, стать). Класифікаційні дерева часто використовуються в прикладних областях, таких як фінанси, маркетинг, інженерія та медицина. Дерево класифікації корисне як дослідницький метод.

Метод дерев рішень часто називають «наївним» підходом, але завдяки ряду переваг, даний метод є одним з найбільш популярних для вирішення завдань класифікації.

Однією з головних переваг моделі класифікації, яка представлена у вигляді дерева рішень, є інтуїтивність, що сприяє кращому розумінню розв'язуваного завдання. Дерева рішення представлені природною мовою. Ця властивість дозволяє легко інтерпретувати роботу алгоритму побудови дерева рішення, на відміну, наприклад, від нейронних мереж. Модель дерева рішення дозволяє якнайкраще пояснити причини належності об'єкти до певного класу.

З допомогою методу дерев рішень можна створювати класифікаційні моделі у досить складних, з погляду формалізації даних, областях.

Існує кілька алгоритмів, які можуть бути використані для побудови дерев рішень на надвеликих базах даних. Приклади таких алгоритмів: SLIQ, SPRINT. Ці алгоритми масштабуються. Масштабованість означає, що необхідний час навчання, тобто побудова дерева рішення, лінійно залежить від зростання кількості записів у базі даних. Також більшість алгоритмів побудови дерев рішень дозволяють обробляти пропущені значення.

Багато класичних статистичних методів вирішення задачі класифікації працюють виключно з числовими даними. Тоді як дерева рішень можуть обробляти числові та категоріальні типи даних.

Якість моделі дерева рішень для класифікації визначається точністю розпізнавання та помилкою.

Точність розпізнавання – це ставлення об'єктів, правильно класифікованих у процесі навчання, до кількості об'єктів набору даних, які брали участь у навчанні моделі.

Помилка – це відношення об'єктів, які неправильно класифіковані в процесі навчання, до загальної кількості об'єктів набору даних, які брали участь у навчанні моделі.

Відсікання гілок використовується лише в тому випадку, коли це не призводить до зростання помилки. Ця процедура є висхідною, тобто проводиться знизу нагору.

У випадку, коли дерево все ще є складним для розуміння, використовують вилучення правил, які об'єднують у набори для опису класів. Кожен шлях від кореня дерева до його вершини чи аркуша дає одне правило. Умовами правила є перевірки на внутрішніх вузлах дерева.

1.2 Побудова систем підтримки прийняття рішень у медицині на основі дерев рішень

1.2.1 Невирішені проблеми створення СППР для медицини

При спробі створення СППР для медицини ми стикаємося з низкою принципових концептуальних бар'єрів.

Перший концептуальний бар'єр пов'язаний із колосальним обсягом накопичених медичних знань.

Другий концептуальний бар'єр пов'язаний з постійним оновленням медичних знань та технологій.

Очевидно, що СППР має ґрунтуватися на актуальних медичних знаннях. Для цього необхідно подолати перші два бар'єри.

Для застосування у СППР медичні знання мають бути формалізовані. Моделі, засновані на медичних наукових знаннях, мають узагальнений та обмежений характер, а емпіричні знання можуть бути недостатньо репрезентативними загалом.

Третій концептуальний бар'єр пов'язаний із самими клінічними даними, наявністю підготовлених джерел даних, наявністю великих клінічних даних (data set, Big Data). За твердженням Microsoft від 30% до 40% світових цифрових даних нині становлять медичні дані.

Четвертий концептуальний бар'єр пов'язаний з недостатньою формалізацією та стандартизацією даних. На Заході цим питанням вже тривалий час приділяється достатньо уваги. Створено тезауруси та онтології (LOING, SNOMED та ін.), що дозволяють розмічати та кодувати медичні дані, пропонуються стандарти для формалізації медичних документів (HL7 CDA, OpenEHR).

Найчастіше доводиться мати справу не з формалізованими та розміченими даними, а з вільними медичними текстами.

Незважаючи на наявність в арсеналах методів роботи з природною та обмеженою професійною мовою (natural language processing, control language processing), аналіз текстів та виділення з них фактів все ще є досить трудомістким і проблемним процесом.

П'ятий концептуальний бар'єр є когнітивним. Він пов'язаний з недостатнім обґрунтуванням та розумінням методу отримання рекомендованого машиною рішення.

Таким чином, основні труднощі при впровадженні СППР є їхня інтеграція в сам робочий процес. Дуже довго існувала тенденція до зосередження лише на функціональному ядрі прийняття рішень у рамках СППЛР, що призводило до неефективного планування фактичного застосування препарату лікарями. Найчастіше такі системи являли собою окремі додатки, що вимагали переривання роботи лікаря в системі, де він працював, перемикання на СППР, введення необхідних даних (навіть якщо вони вже були раніше введені в іншу систему) і аналізу отриманих результатів. Тобто, такі додаткові дії порушують нормальний робочий процес лікаря та забирають час. Ще однією проблемою є те, що у деяких галузях СППР стикаються зі значними технічними проблемами. Наприклад,

біологічні системи можуть бути дуже складними і клінічне рішення може вимагати аналізу великого обсягу потенційно релевантних даних. Так, електронна система, що працює на основі принципів доказової медицини, при формулюванні рекомендацій щодо плану лікування пацієнта потенційно може враховувати симптоми, медичний анамнез, сімейний анамнез та генетику пацієнта. Ще одним проблемним моментом в СППЛР є велика кількість повідомлень, що генеруються ними. Зрозуміло, що коли система видає велику кількість попереджень (особливо таких, що не передбачають подальших дій), це набридає лікарям і вони можуть перестати приділяти попередженням достатньо уваги, а це може потенційно призвести до ігнорування критично важливих повідомлень [5].

Підсумовуючи, можна виділити основні проблеми сучасних систем підтримки прийняття лікарських рішень:

- складнощі для лікарів в освоєнні;
- необхідність витрачати велику кількість часу для взаємодії з системою;
- складнощі в інтегруванні СППЛР в робочий процес;
- необхідність в постійному супроводженні та актуалізації системи.

Всі ці проблеми будуть враховані під час постановки завдання, розробки технічних вимог до майбутньої системи, а також реалізації самої СППЛР.

1.2.2 СППР на основі дерев рішень

У своїй практичній діяльності лікар стикається з великою кількістю завдань, які потребують швидкого та точного реагування. При постановці діагнозу та лікуванні необхідно врахувати:

- дані огляду;
- індивідуальні особливості пацієнта;
- результати лабораторних та інструментальних методів дослідження.

Рутинні операції – оформлення медичної документації, моніторинг стану пацієнтів, контроль за дотриманням призначень створюють додаткове навантаження на фахівців.

Ситуація ускладнюється появою нових регламентів надання медичної допомоги: клінічних рекомендацій, стандартів та протоколів. У клініцистів не завжди є можливість актуалізувати свої знання. Це з зростаючим обсягом спеціалізованої інформації та її джерел.

При цьому важливо забезпечити своєчасність та безпеку клінічних заходів: вчасно виявити захворювання та розпочати правильне лікування. Часткова автоматизація лікувально-діагностичного процесу та інформаційна підтримка фахівця – напрямки, які покликані знизити навантаження на лікаря.

Помічником лікаря під час вирішення клінічних завдань може стати система підтримки прийняття лікарських рішень.

Результатом діагностик має бути правильно поставлений діагноз та виписка рекомендованого курсу лікування. Формалізацію процесу діагностики представимо на рис.7.

Система підтримки прийняття рішень є чудовим інструментом для клінічної та лабораторної діагностики інфекційних захворювань методами інформаційних технологій обробки даних.

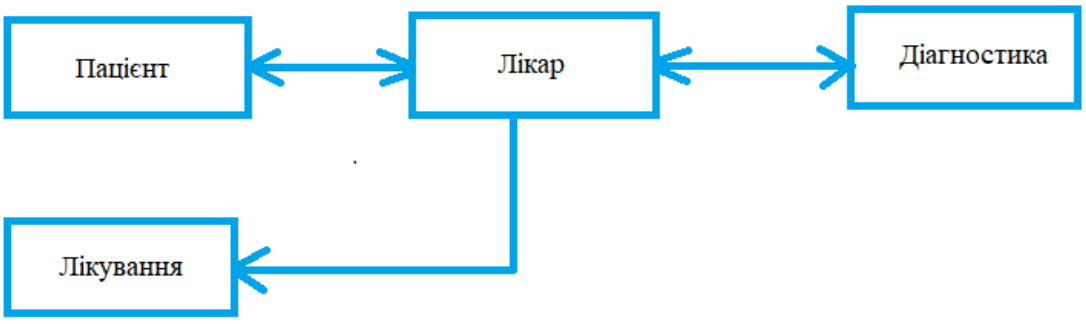

Рисунок 7 – Алгоритм діагностики

СППР можуть суттєво оптимізувати роботу амбулаторних клінік та прискорити постановку правильного діагнозу пацієнтів.

На етапі диференціальної діагностики вирішуються два завдання:

- виділення найбільш актуального захворювання від інших захворювань;
- розробка діагностичної гіпотези про нозологічну форму та особливості перебігу хвороби пацієнта.

Абсолютно точного і повністю достовірного діагнозу не існує, і в даний момент необхідна достатня точність діагнозу, що відображає найвірогідніше захворювання, що дозволяє проводити ефективні медичні втручання. Завдання диференціальної діагностики захворювань з погляду інформатики є дуже складним, часто – нечітким, некоректно поставленим багатопараметричним завданням. Для її вирішення необхідна тісна взаємодія лікаря із цифровою системою підтримки прийняття рішень.

Таким чином, медична інформація потребує засобів її опису, обробки даних та їх аналізу. Завдання у розробці подібних засобів є важко формалізованими. Існує необхідність розробки таких моделей, які б точно і зрозуміло описували процеси збору, обробки, зберігання та передачі інформації. Системи, які допомагають приймати рішення, є одними з найефективніших для вирішення цього завдання.

Розглянемо, наприклад, спосіб дерева рішень виявлення прихованих закономірностей в хворих на бронхіальну астму.

Представимо модель дерева рішення. Застосовуючи її, можна визначати як стать, вік, вага та зріст впливають на захворювання на бронхіальну астму пацієнта. Побудоване дерево рішень представлене рис.8.

Також розглянемо модель дерева, в якій як вхідні дані використовувалися характеристики психологічного стану пацієнта. Дане дерево рішень представлене на рис.9.

Побудовані моделі активно використовувалися при створенні прототипу віртуального центру оцінки та моніторингу стану дітей з найбільш

поширеними неінфекційними захворюваннями. А також для розширення наявної бази знань системи підтримки наукових досліджень про бронхіальну астму.

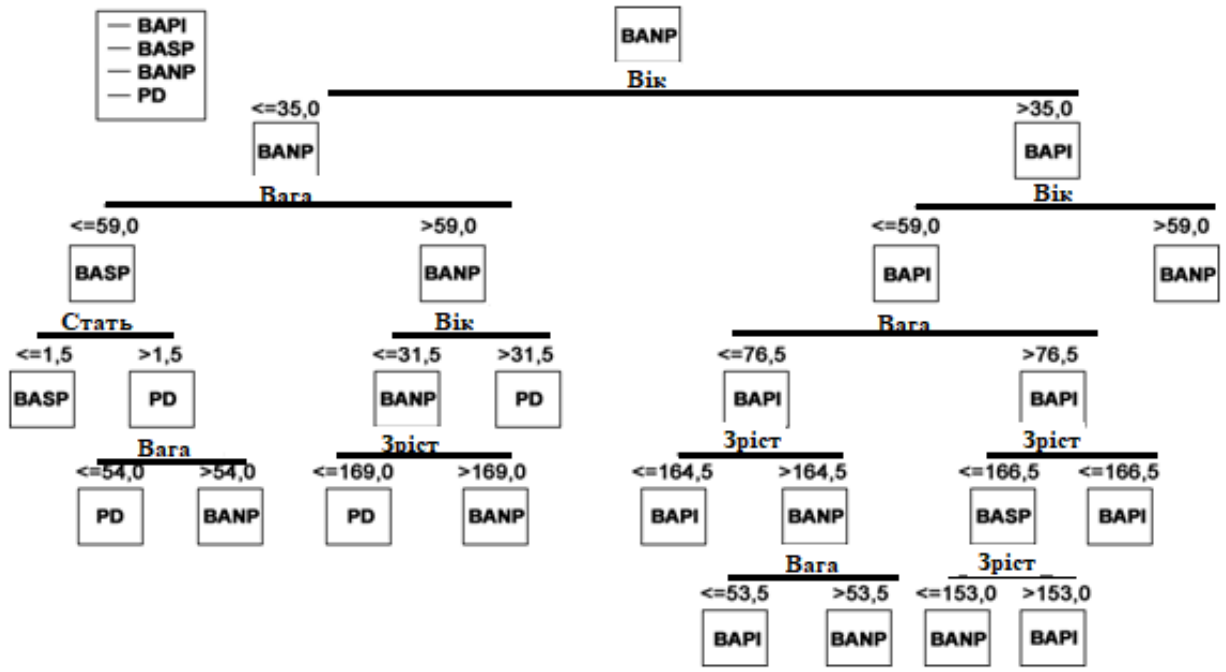

Рисунок 8 – Дерево рішень, побудоване за даними анамнезу

Гострі респіраторні інфекції найбільш поширені серед дітей, яким надається первинна медична допомога. Але при цьому лише невелика частина з них може мати такі інфекції як сепсис, менінгіт, пневмонія. Раннє діагностування захворювання допомагає уникнути ускладнень чи смертельних випадків. Правила клінічного прогнозування мають величезний потенціал для поліпшення прийняття діагностичних рішень за рідкісних, але критичних станів.

Для того, щоб оцінити ймовірність серйозної інфекції, клініцисти використовують ознаки та симптоми захворювань. Такий підхід дозволяє прийняти найкраще рішення щодо подальшого лікування пацієнта.

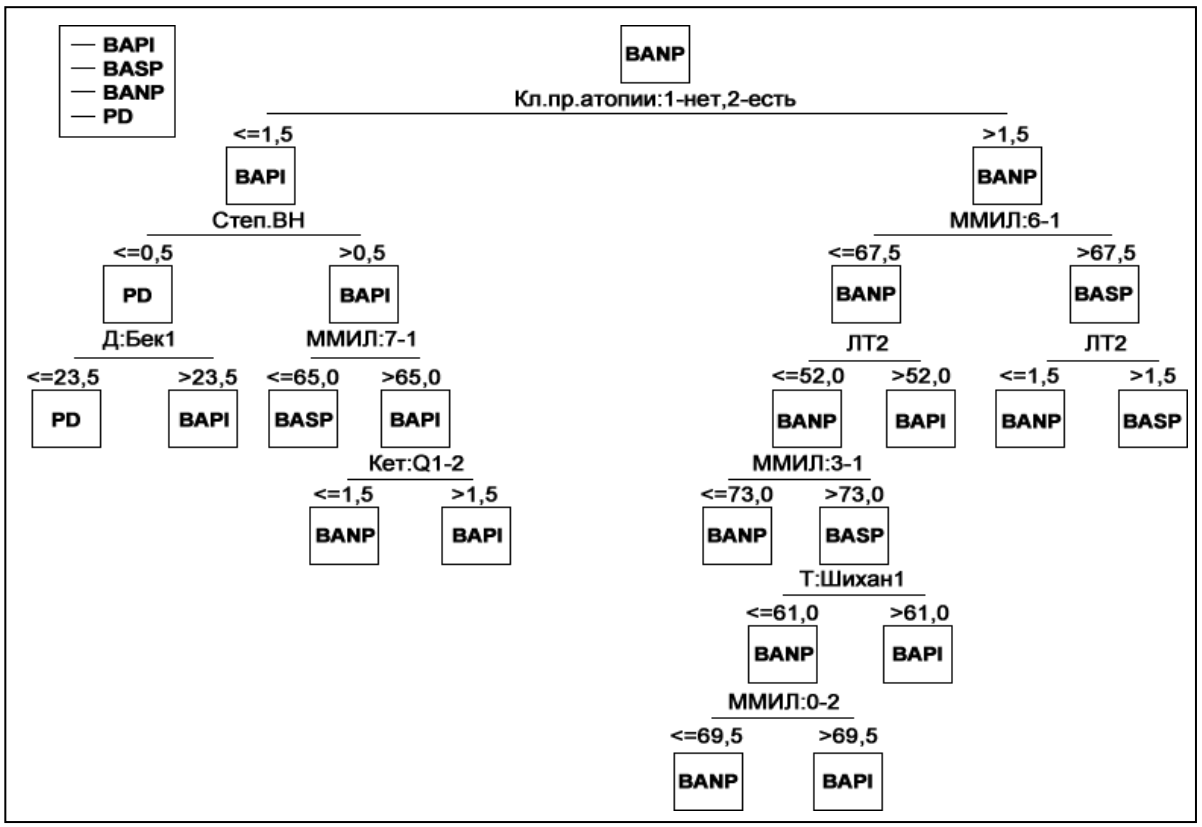

Рисунок 9 – Дерево рішень, побудоване з урахуванням психологічного стану пацієнта

Van den Bruel побудував модель дерева рішень, що складається з чотирьох щаблів. Ця модель була виведена на основі вихідного набору бази даних, що включає симптоми 4000 дітей. Дерево рішень складається з наступних атрибутів: відчуття клініциста «щось не так», «задишка», «температура > 39,95 ° C» і «діарея у дітей віком 1-2, 5 років». Дерево рішень представлене на рис.10.

Якщо на якесь із цих чотирьох питань виникає позитивна відповідь, тоді дерево видає позитивний результат, тобто рекомендує госпіталізацію пацієнта. Чутливість та негативна прогностична цінність у цьому дослідженні становлять майже 100%. Дерево рішень також продемонструвало високу чутливість при ретроспективній валідації в іншому наборі даних первинної медичної допомоги з використанням апроксимацій

для кишкового відчуття та задишки, проте проспективна валідація ще не була проведена.

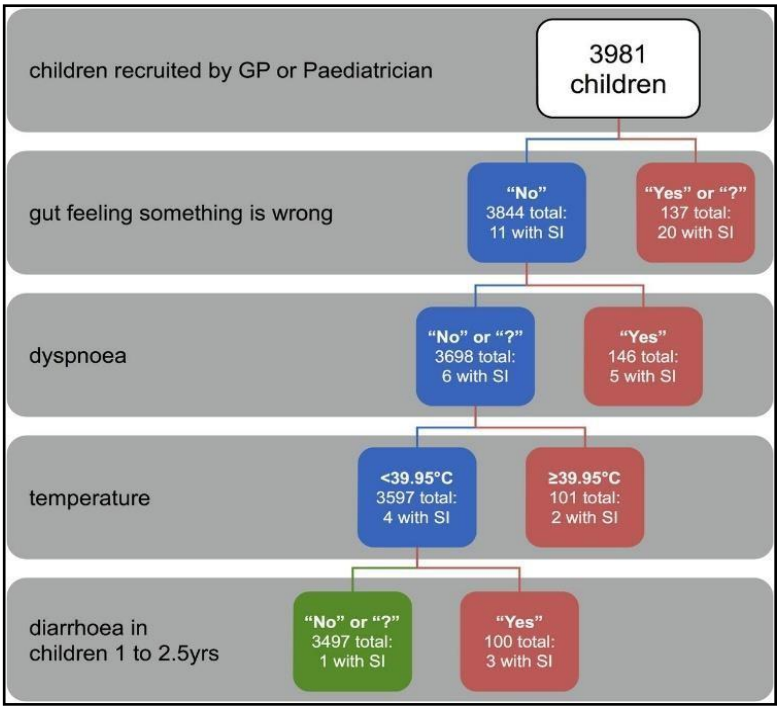

Рисунок 10 – Чотириступінчасте дерево рішень, розроблене Van den Bruel

У незалежній валідаційній когорті ця модель дерева рішення для клінічного прогнозування виявилася досить чутливою для ухвалення рішення про госпіталізацію дітей, які мають ризик серйозної інфекції. Таким чином, дерево рішень може застосовуватись на практиці [5].

1.3 Респіраторна вірусна інфекція

Вірусні інфекції відносяться до групи найпоширеніших інфекцій, що викликають захворювання у людей будь-якого віку та статі. Кількість вірусів величезна і відрізняється своїми особливостями поширення, поразки різних органів прокуратури та систем організму, перебігу хвороби, ускладненнями тощо. Найчастіше зустрічаються віруси грипу, гострих респіраторних вірусних інфекцій, аденовіруси, ротовіруси, віруси герпесу, папіломи

людини, риновіруси та ін. Віруси відрізняються високим рівнем ураження населення. Кожен із них сприяє зниженню функцій імунної системи організму та провокує розвиток патогенної бактеріальної мікрофлори.

Вірусні інфекційні захворювання небезпечні не лише своїми ускладненнями, а й тим фактом, що вони суттєво знижують здатність організму протистояти новим хворобам, тобто знижують ефективність імунної системи людини. До чого може спричинити зниження імунітету? До того що хвороба часто переходить у хронічну форму. Крім того, людина стає вразливою перед новими вірусами та мікробами.

Термін «гостра респіраторна вірусна інфекція» включає велику кількість захворювань. Всі вони викликаються вірусами і проявляються схожими симптомами, які при одужанні безвісти зникають.

Коли захисна функція організму знижена, сприйнятливість до зовнішніх впливів, зокрема збудників захворювань, стає вищою.

Віруси гострих респіраторних інфекцій передаються двома шляхами:

1. Контактно-побутовий (рукостискання, загальні предмети користування, іграшки, посуд та ін.).
2. Повітряно-крапельний, або аерозольний (розмова, крик, плач, чхання та кашель із крапельками слизу). Зараження людини відбувається при вдиханні повітря з крапельками слизу, що містять збудник хвороби.

Збудники проникають в організм через ніс, ротову порожнину, кон'юнктиву (всі слизові оболонки). Процес починається у верхніх дихальних шляхах. Респіраторні віруси інфікують клітини, прикріплюючись своїми активними центрами до специфічних рецепторів. Інкубаційний період від одного до п'яти днів.

Розглянемо як пов’язані ГРВІ та стан порожнини рота.

Порожнина рота найбільш сприйнятлива до різних інфекцій, особливо ГРВІ, через спосіб поширення. Клінічні прояви ГРВІ в ротовій порожнині

характеризуються змінами слизової оболонки – почервоніння, невеликі крововиливи, посилений судинний малюнок, набряклість, наліт [6].

1.4 Постановка задачі

Метою магістерської роботи є розробка системи підтримки прийняття рішень для діагностики інфекційних захворювань на основі алгоритму дерева рішень.

Завданням роботи є розробка та дослідження компонентів системи підтримки прийняття рішення для діагностики інфекційних захворювань, а саме респіраторних захворювань.

У користувача системи (лікаря) мають бути вхідні данні – характеристики слини пацієнтів. Процедура відбору зразків матеріалів пацієнтів проводилася за допомогою біосенсора діелектричної проникності. Далі користувач вводить відомості про пацієнта та отримує від системи відповідь до якої групи з яким захворюванням належить пацієнт. Система має містити блок прийняття рішень з використанням методу дерева рішень. Попередньо система має пройти навчання на основі великої кількості клінічних даних. Точність класифікації СППР має бути вище 80%. При цьому система має бути максимально простою для користувача, та давати відповідь на основі клінічних даних пацієнта.

Таким чином, для розробки компонентів СППР необхідно виконати наступні завдання:

1) ознайомитись із принципами алгоритму дерева рішень, математичними моделями алгоритмів побудови дерева рішень;
2) проаналізувати особливості систем прийняття рішень, що використовуються в медицині;
3) розробити та реалізувати алгоритм побудови дерева рішення для діагностики інфекційних захворювань;
4) спроектувати СППР для діагностики інфекційних захворювань з

урахуванням алгоритму дерева рішень;

5) вибрати технології реалізації СППР з використанням алгоритмів побудови дерева рішень;
6) реалізувати систему;
7) провести тестування та описати результати.

2 ПРОЄКТУВАННЯ СППР ДЛЯ ДІАГНОСТИКИ ІНФЕКЦІЙ

2.1 Визначення вимог до системи

Розглянемо спочатку схему роботи системи (рис.11). Загальний алгоритм буде виглядати наступним чином:

1. Збирання інформації про пацієнта.
2. Генерація анамнестичного опитувальника.
3. Пропозиція щодо постановки диференціального діагнозу.
4. Постановка лікарем діагнозу.
5. Пропозиція схеми лікування пацієнта.

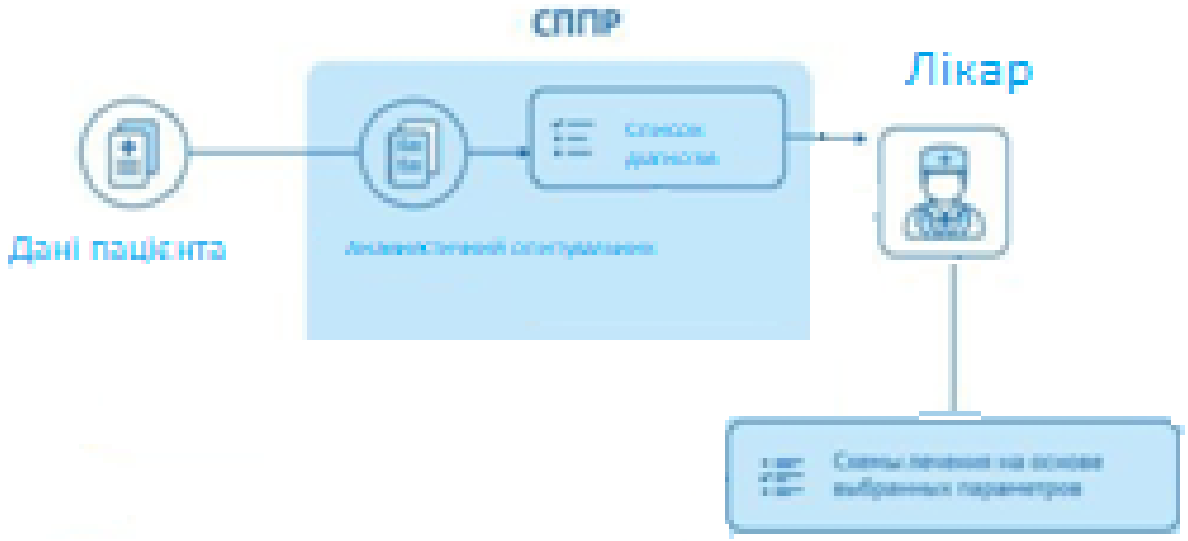

Рисунок 11 – Схема роботи СППР

Визначимо функціональні вимоги до системи.

Функціональні вимоги – це вимоги, які визначають дії, які має виконувати система, не враховуючи обмежень, що з її реалізацією, тобто визначають поведінку системи у процесі обробки інформації.

Додаток, що розробляється, повинен задовольняти наступним функціональним вимогам:

- користувач повинен мати можливість ввести дані про пацієнта для прогнозування діагнозу;
- користувач повинен мати можливість одержати результат прогнозування діагнозу;
- перегляд історії;
- система повинна повідомити про некоректні дії користувача.

Визначимо нефункціональні вимоги.

Нефункціональні вимоги – це вимоги, які визначають поведінка системи, але описують атрибути системи чи атрибути системного оточення. Додаток, що розробляється, повинен задовольняти наступним нефункціональним вимогам:

- програма має бути реалізована мовою Python;
- додаток має використовувати алгоритм дерева рішень.

2.2 Проєктування діаграм системи

Для об'єктивної формалізації системи і визначення її кордонів необхідно скласти концептуальну модель даних. Дана модель визначає основні поняття і потоки інформації. Для побудови була використана методологія IDEF0, в рамках якої модель представлена у вигляді набору модулів, підпорядкованих один одному [7]. У лівому верхньому кутку знаходиться більш важлива функція, далі розташовуються модулю за ступенем важливості, в яких відображаються вхідні, вихідні дані, механізми і способи управління модулями, їх логічна послідовність. Контекстна діаграма показує головний бізнес-процес і пов'язані з нею потоки даних, механізми та інструкції. Діаграма складається з головного функціонального блоку «Виявлення захворювання» (рисунок 12).

Рисунок 12 – Контекстна діаграма

Стрілки поділяються на чотири види: вхідні дані, вихідні дані, дані для управління, механізми. Зліва від блоку відображені вхідні дані. Праворуч від блоку відображені вихідні дані. Вгорі блоку відображені дані для управління. Дані управління включають в себе інструкція користувача, медична документація. Внизу блоку відображені механізми, за допомогою яких відбувається перетворення вхідних даних у вихідні дані. До механізмів відносяться: лікар користувач та СППЛР.

Систему бажано проектувати так, щоб її фрагменти можна було використовувати повторно в інших системах. Над програмою, як правило, працює багато людей – одні йдуть, приходять нові, яким доводиться супроводжувати програму. Система має бути добре структурованою, не містити дублювання, мати добре оформлений код та бажано документацію.

Таким чином, можна сформулювати критерії, яким буде відповідати система підтримки прийняття рішень для діагностики захворювання:

- ефективність системи;
- гнучкість системи;
- розширюваність системи;

- масштабованість процесу розробки;
- можливість ретельно протестувати;
- можливість повторного використання;
- добре структурований та зрозумілий код/супроводжуваність.

Спроєктуємо діаграму варіантів використання. Згадаємо, що вона описує, який функціонал програмної системи, що розробляється, доступний кожній групі користувачів [8].

В системі, що розробляється є тільки один користувач – це лікар, що буде вносити дані пацієнта і отримувати діагноз згідно даних.

Для проектування системи було використано мову графічного опису об'єктного моделювання UML.

UML – уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. UML є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. Це мова широкого профілю, а також відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи. Мета UML – визначення, проектування, візуалізація, а також документування програмних систем. UML – це не мова програмування, але генерація коду також можлива. Варіанти використання є описами типових взаємодій між користувачами системи і самою системою. Вони відображають зовнішній інтерфейс системи і вказують форму того, що система повинна зробити (саме що, а не як).

Діаграми використовується для опису взаємини і залежності між групами варіантів використання і дійових осіб, які беруть участь в процесі. Діаграми варіантів використання не призначені для відображення проєкту і не можуть описувати внутрішній устрій системи, вони призначені саме для спрощення взаємодії з майбутніми користувачами системи, з клієнтами, і для визначення необхідних характеристик системи. Тобто, діаграми варіантів використання говорять про те, що система повинна робити, без яких –небудь вказівок методів.

Як було виявлено під час аналізу, для максимально ефективного впровадження системи в робочий процес діагностування інфекції, вона має бути максимально простою та виконувати тільки основну функцію – виявлення інфекції у пацієнта виходячи з показників аналізу слини та додаткової інформації про пацієнта, що вводиться користувачем (лікарем).

Було побудовано модель взаємодії актора «Користувач» із додатком. Діаграма варіантів використання (use-case diagram) представлена на рисунку 13.

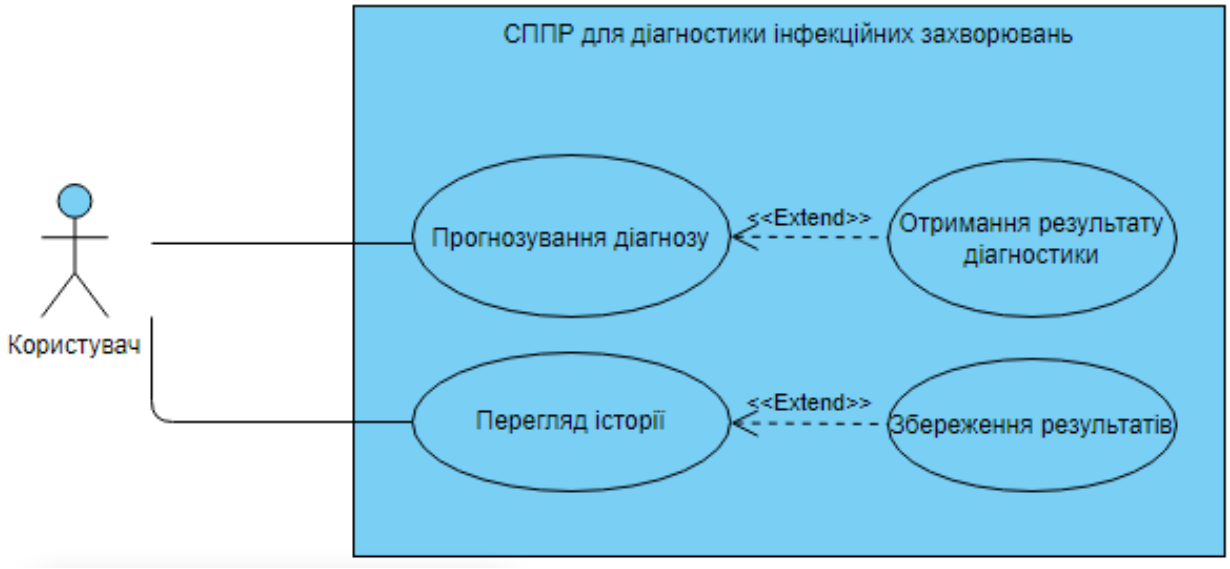

Рисунок 13 – Діаграма варіантів використання

Наведемо специфікацію основних варіантів використання.

Першим розглянемо варіант використання «Прогнозування діагнозу».

Прецедент: спрогнозувати діагноз.

ID: 1.

Короткий опис: прогнозування діагнозу пацієнта.

Головні актори: користувач.

Другі актори: ні.

Передумови: відсутні.

Основний потік: прецедент починається, коли користувач переходить на сторінку Діагностика.

Користувач вводить дані.

Система класифікує отримані дані.

Постумова: система виводить результат на екран.

Альтернативні потоки: подія починається тоді, коли введені користувачем дані некоректні.

Система видає помилку "Некоректні дані".

Розглянемо тепер специфікацію варіанта використання «Перегляд історії» прогнозування пацієнтів.

Прецедент: переглянути історію прогнозування пацієнтів.

ID: 2.

Короткий опис: перегляд списку пацієнтів.

Головні актори: користувач.

Другі актори: ні.

Передумови: відсутні.

Основний потік: прецедент починається, коли користувач переходить до сторінки «Журнал».

Постумова: система відображає список пацієнтів на екрані.

Альтернативні потоки: ні.

Наступним кроком буде проектування діаграми компонентів.

Згадаємо,що це структурна діаграма мови уніфікованого моделювання, вона визначає особливості фізичного уявлення системи. Діаграма компонентів дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами.

Діаграма компонентів надає загальну картину архітектури системи, допомагає розробникам та архітекторам краще зрозуміти її структуру та взаємозв'язки, а також є корисним інструментом для комунікації та документування архітектурних рішень.

Діаграма компонентів розробляється для таких цілей:

- візуалізація загальної структури вихідного коду програмної системи;
- специфікація можливого варіанта програмної системи;
- забезпечення багаторазового використання окремих фрагментів програмного коду.

Компонент є окремою частиною системи, яка виконує певну функцію або має певну роль. Він може бути програмним модулем, класом, бібліотекою, сервісом, фізичним пристроєм тощо. Він зазвичай відображається у вигляді прямокутника з його ім'ям або позначенням.

В роботі було розроблено діаграму компонентів, яка показує розбиття системи на структурні компоненти та зв'язки між ними. Розроблена діаграма представлена рисунку 14.

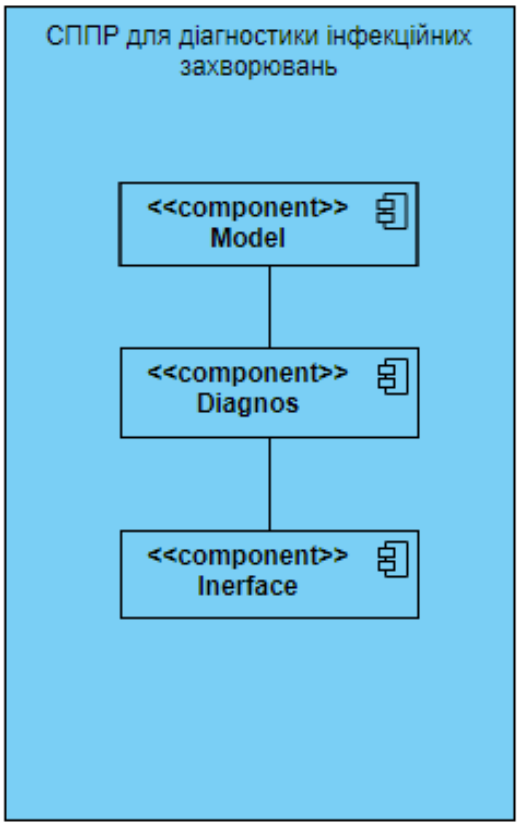

Рисунок 14 – Діаграма компонентів

Ця діаграма складається з наступних артефактів:

1. Model – програмний модуль, що відповідає за створення моделі дерева рішень та її навчання, а також для оцінки точності побудованої моделі;
2. Diagnos – програмний модуль, у якому реалізовано логіку роботи програми при взаємодії з інтерфейсом;
3. Interface – програмний модуль, в якому реалізовано підмальовування елементів інтерфейсу користувача.

При моделюванні поведінки проєктованої системи виникає необхідність не тільки представити процес зміни її станів, а й деталізувати особливості алгоритмічної і логічної реалізації виконуваних системою операцій. Важливо підкреслити ту обставину, що зі збільшенням складності системи суворе дотримання послідовності виконуваних операцій набуває все більшого значення. Для моделювання процесу виконання операцій у мові UML використовуються звані діаграми діяльності. Вони дозволяють реалізувати в мові UML особливості процедурного та синхронного управління, обумовленого завершенням внутрішніх діяльностей та дій. На діаграмі діяльності відображається логіка або послідовність переходу від однієї діяльності до іншої, при цьому увага фіксується на результаті діяльності. Сам результат може призвести до зміни стану системи або повернення деякого значення [9].

Таким чином, діаграма діяльності – ще одна важлива діаграма UML, що описує динамічні аспекти системи. Це, по суті, блок-схема, що представляє потік від однієї дії до іншої. Діяльність може бути описана як робота системи.

Потік керування передається від однієї операції до іншої. Цей потік може бути послідовним, розгалуженим або паралельним. Діаграми дій стосуються всіх типів керування потоком із використанням різних елементів.

Відповідно до вимог реалізовано діаграму діяльності, яка показує, як відбувається процес взаємодії користувача із системою. Вона представлена рисунку 15.

У представленій діаграмі користувач знаходиться на головній сторінці програми. Користувач вводить дані про пацієнта. Після цього система класифікує отримані дані і виводить результат на екран. Користувач переглядає результати.

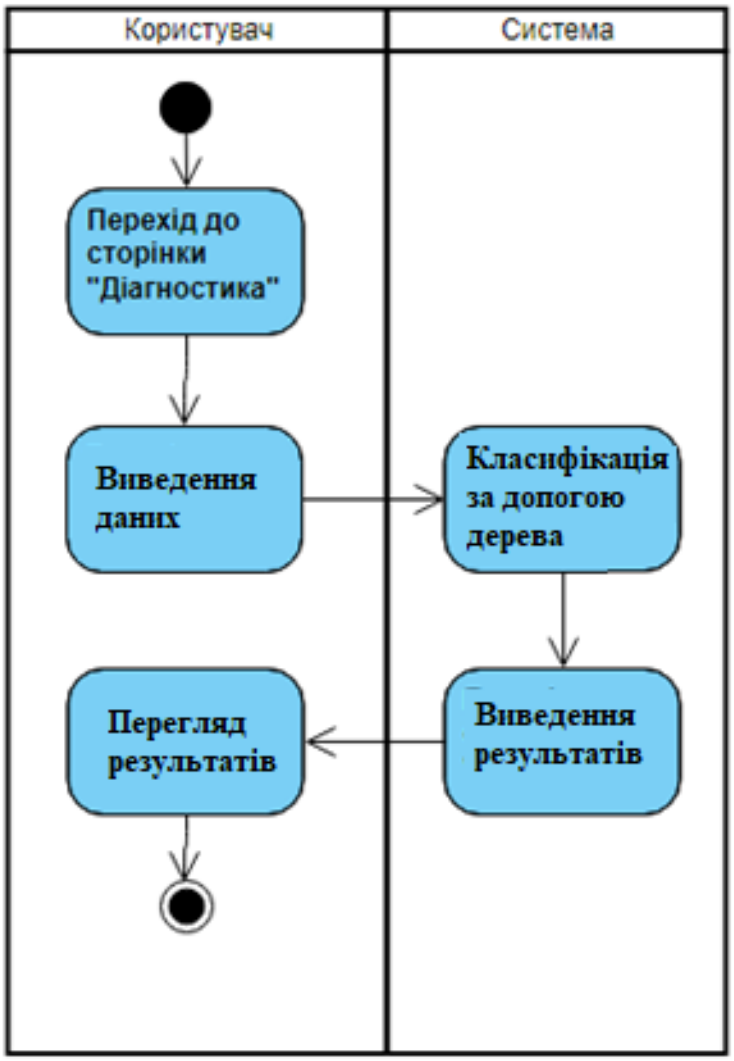

Рисунок 15 – Діаграма діяльності

2.3 Побудова прогностичної моделі

2.3.1 Підготовка набору даних

При підготовці даних зазвичай проводиться первинна обробка даних, яка спрямована на впорядкування інформації про об'єкт, що досліджується: упорядкування вихідних даних; виявлення та усунення помилок, некоректно введених даних, викидів, прогалин у відомостях; виявлення прихованих закономірностей та зв'язків. Оскільки брали готовий набір даних, то і підготовку даних не потрібно було проводити.

Темою диплому є розпізнавання інфекційних захворювань, проте даних про інфекції не було знайдено, тому як вихідні дані було взято вже готовий набір Exasens із сайту Machine Learning Repository, що стосується респіраторних захворювань, тобто такі, що відносяться до ураження дихальних шляхів. Набір включає характеристики зразків слини 399 пацієнтів. Процедура відбору зразків матеріалів пацієнтів проводилася за допомогою біосенсора діелектричної проникності [10]. Пацієнти класифіковані за 4 групами зразків слини:

1) амбулаторні та госпіталізовані пацієнти з хронічним захворюванням легень без гострої респіраторної інфекції (ХОЗЛ);
2) амбулаторні та госпіталізовані пацієнти з астмою без гострих респіраторних інфекцій (астма);
3) пацієнти з респіраторними інфекціями, але без хронічних захворювань легень та астми (інфіковані);
4) здорових пацієнтів.

Для вирішення задачі формування бази знань було обрано метод побудови дерева рішень, як один з найбільш популярних і хороших результатів. Вихідними для побудови дерева рішень є набір даних, про який вже було сказано. Дані були поділені на навчальну та тестову вибірки у співвідношенні 70% на 30% відповідно. Точність навченої моделі становила 0,83, що є добрим показником.

2.3.2 Побудова дерева рішень

Створюємо поділ у наборі даних за алгоритмом роботи дерева рішень.

1. Розрахунок балів Джіні.
2. Розділення набору даних. Його можна визначити як поділ набору даних на два списки рядків, що мають індекс атрибуту та значення поділу цього атрибуту. Отримавши дві групи – праву та ліву, з набору даних, ми можемо обчислити значення поділу, використовуючи показник Джині, розрахований у першій частині. Значення поділу визначатиме, у якій групі буде атрибут.
3. Оцінка всіх розщеплень. Наступна частина після знаходження оцінки Джіні та набору даних розщеплення – це оцінка всіх розщеплень. Для цього ми повинні перевірити кожне значення, пов'язане з кожним атрибутом, як поділ кандидатів. Потім нам потрібно знайти найкращий можливий поділ, оцінивши вартість поділу. Найкращий поділ буде використовуватися як вузл в дереві рішень.

Побудована модель дерева рішення представлена рисунку 16.

Як критерій розщеплення для цієї моделі використовувався індекс Gini, а також дерево рішень обмежене 5 ступенями.

Оцінка точності збудованої моделі становить 0.83. Для перевірки якості збудованої моделі було проведено самостійне візуальне тестування моделі з вихідними даними пацієнтів. Результат роботи моделі та апріорне віднесення пацієнта до однієї з 4 груп збіглися у 95% випадків.

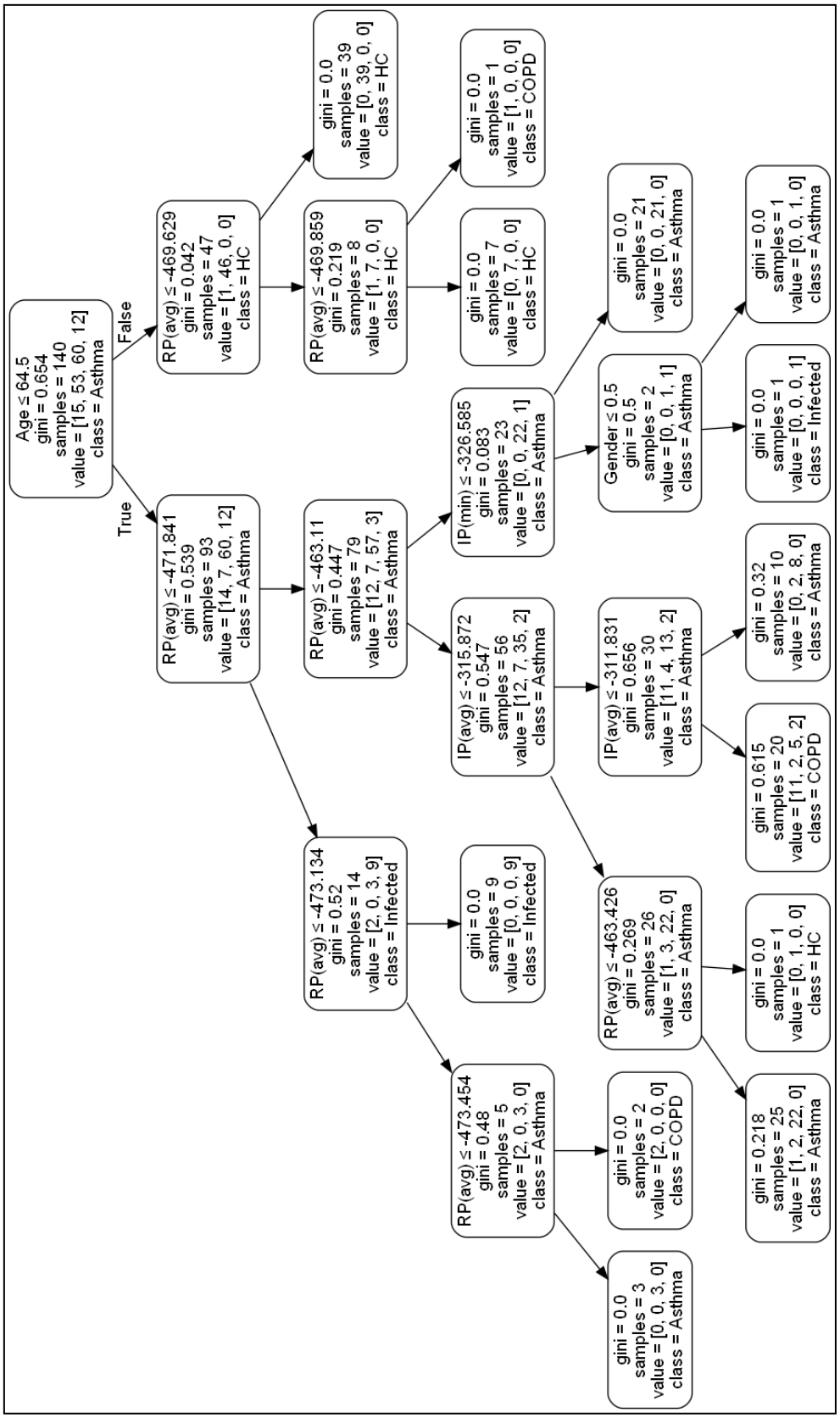

Рисунок 16 – Побудована модель дерева рішень

3 РЕАЛІЗАЦІЯ СППР

3.1 Вибір засобів розробки

Для розробки програмної частини системи було обрано високорівневу мову Python 3.8.5. Розробка велася в середі розробки PyCharm Community Edition 2022.3. Для реалізації компонентів було використано такі бібліотеки для мови Python: pandas, scikit-learn.

Мова програмування Python – це об'єктно-орієнтована мова програмування високого рівня загального призначення з відкритим кодом. Python активно використовують для аналізу даних. Дані стали цінним активом у будь-якій сучасній галузі, і більшість компаній зацікавлені у збиранні, обробці та аналізі релевантних даних, щоб витягти з них цінну інформацію для бізнесу. І тут Python виходить за межі будь-якої конкуренції. Ця мова особливо цінна тим, що, крім великої стандартної бібліотеки, вона надає величезний набір додаткових модулів, розроблених спеціально для аналітичних цілей. Найвідоміші бібліотеки Python для проведення аналізу даних – це pandas та NumPy. Ці інструменти дозволяють робити з вашими даними майже все, наприклад, очищати і аналізувати їх, вивчати статистику або візуалізувати приховані тенденції у ваших даних. Для візуалізації даних Візуалізація даних – це окрема частина аналізу даних, яка допомагає нам подавати інформацію, необроблену чи очищену та перетворену, у більш привабливій та змістовній формі. Тут Python знову входить у гру, пропонуючи широкий спектр інструментів візуалізації даних. Найпопулярніші з них – matplotlib і заснований на ній seaborn. Використовуючи їх, ми можемо створювати буквально всі види візуалізації: від найпростіших до складніших [11]. Машинне навчання лежить в основі більшості завдань науки про дані. Він є область штучного інтелекту, пов'язану з використанням алгоритмів, що дозволяють машинам вивчати закономірності та тенденції на основі історичних даних, щоб робити прогнози на основі невідомих даних. Використовуючи методи ML, ми

можемо створювати моделі, які можуть точно передбачити швидкість відтоку клієнтів компанії, оцінити ризик виникнення у людини певного захворювання, визначити оптимальне розташування автомобілів таксі і т.д. За допомогою Python ми можемо побудувати модель ML, використовуючи лише три рядки коду. Для розробки програмного забезпечення крім свого багатостороннього застосування в галузях науки про дані, ця мова використовується на кожному етапі розробки програмного забезпечення, включаючи контроль складання, автоматичну безперервну компіляцію, прототипування, відстеження помилок, тестування та обслуговування програмного забезпечення. З його допомогою ми можемо створювати аудіо- або відеопрограми на основі методів штучного інтелекту або машинного навчання, API, GUI або будь-якого іншого типу програмного забезпечення. Для веб-розробки: в той час як для створення візуальної частини веб-сайту ми будемо в основному використовувати такі мови, як HTML, CSS та JavaScript, для його невидимої частини ми часто вибираємо Python. Серед масштабних веб-сайтів та програм, створених за допомогою цієї мови, варто згадати Google, Facebook, Instagram, YouTube, Dropbox та Reddit. Це відмінний інструмент для написання програм для автоматизації різних завдань, що повторюються. Цей процес називається скриптингом. Зокрема, можна робити скрипти для роботи з файлами та папками. Наприклад, можна створювати, перейменовувати, перетворювати, розділяти, об'єднувати або видаляти файли, перевіряти їх на наявність помилок. Ви також можете використовувати автоматизацію Python для пошуку та завантаження інформації з Інтернету, заповнення та надсилання онлайн-форм, а також надсилання регулярних повідомлень або електронних листів.

PyCharm був розроблений компанією JetBrains – відомою міжнародною компанією, що спеціалізується на розробці інструментів для програмування різними мовами, таких як Java, Kotlin, C#, F#, C++, Ruby, Python, PHP, JavaScript та багато інших [12].

PyCharm надає розробникам багато ключових функцій, які значно спрощують процес програмування. Ось кілька із них:

1. Налагодження коду. Потужний налагоджувач дозволяє знаходити та виправляти помилки, встановлюючи точки зупинки та аналізуючи значення змінних.
2. Рефакторинг. Інструменти рефакторингу допомагають змінювати структуру коду без втрати функціональності.
3. Підтримка систем керування версій. Інтеграція з Git, Mercurial та іншими системами дозволяє зручно працювати з історією змін.
4. Автодоповнення коду. Інтелектуальне автодоповнення прискорює процес написання коду та знижує ймовірність помилок.
5. Інспектування коду. Статичний аналіз коду допомагає виявляти потенційні помилки та підтримувати високу якість коду.
6. Інтеграція із віртуальними оточеннями. PyCharm пропонує зручний інтерфейс для роботи з віртуальними оточеннями Python. Ви можете створювати та керувати різними оточеннями, що дозволяє ізолювати залежності та бібліотеки для кожного проекту.

Для аналізу даних знадобляться модулі pandas та scikit-learn. За допомогою pandas ми проведемо початковий аналіз даних, а sklearn допоможе обчислити прогнозну модель [13].

Python цінують за лаконічність, універсальність і стабільність. На ньому можна за короткий термін створювати веб-додатки, які вирішують нагальні завдання. Головні переваги Python – велике співтовариство програмістів і швидкість розробки.

Завдяки лаконічній мові програмування і відмінним бібліотекам проекти на Python робити дешевше і швидше. А широкий набір функцій дозволяє вирішувати не тільки типові для інтернет-додатків завдання, а й втілювати в життя унікальні ідеї.

Синтаксис Пітона простий, він схожий на природну мову. Лаконічний, читабельний код легше підтримувати, інспектувати і виправляти. Стандартна

бібліотека Пітона спрощує виконання багатьох рутинних завдань програміста і дає доступ до системного функціоналу за допомогою модулів, написаних на C.

Під завдання конкретного проекту можна підключати зовнішні бібліотеки. Вони покривають всі сфери розробки – від e-commerce і корпоративних систем до машинного навчання і візуалізації статистики.

Python добре інтегрується з іншими мовами. Наприклад, в вихідний код можна вбудовувати фрагменти на C ++.

С допомогою Пітона також можна «склеювати» фрагменти коду на інших мовах [14].

3.2 Реалізація компонентів системи

Система, що розробляється, складається з трьох компонентів.

Перший компонент – Model.

Цей компонент складається з 3 функцій, у яких реалізовано алгоритм побудови моделі дерева рішень та її подальше збереження.

Функція read_dataset розроблена для читання файлу з вихідним набором даних для майбутньої моделі. Представимо реалізація цієї функції:

```
def read_dataset():
df = pd.read_csv('exasens.txt', sep=",", header=None,
names=["Diagnos", "IP(min)", "IP(avg)", "RP(min)",
"RP(avg)", "Gender", "Age", "Smoking"]) return df
```

Функція TreeClassifier розроблена для побудови моделі дерева класифікації. Представимо реалізацію функції:

```
def TreeClassifier():
x = df.loc[:, 'IP(min)':'Smoking'] y = df['Diagnos']
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=1)
clf = DecisionTreeClassifier() clf = clf.fit(x_train,
y_train) y_pred = clf.predict(x_test)
print(classification_report(y_test, y_pred)) return clf
```

Функція save_model розроблена для збереження моделі файлу, щоб використовувати її надалі. Представимо реалізацію функції:

```
def save_model(model, filename): file = filename
pickle.dump(model, open(file, 'wb')) return True
```

Наступний компонент – Diagnos.

Цей компонент складається з трьох функцій, за допомогою яких реалізовано завантаження раніше побудованої моделі та класифікація нового об'єкта. Розглянемо кожну функцію.

Функція load_model розроблена для завантаження раніше збудованої моделі з файлу. Представимо реалізацію функції:

```
def load_model(filename): file = filename
model = pickle.load(open(file, 'rb')) return model
```

Функція prediction розроблена для прогнозування респіраторної інфекції у пацієнта за введеними параметрами. Представимо реалізацію функції:

```
def
prediction(model,ip_min,ip_avg,rp_min,rp_avg,sev,age,smoking):
Xnew = [[ip_min,ip_avg,rp_min,rp_avg,sev,age,smoking]]
ynew = model.predict(Xnew) return ynew[0]
```

Функція write_info розроблена для запису даних пацієнта у файл.

Представимо реалізацію функції:

```
def
write_info(file,fam,name,otch,ip_min,ip_avg,rp_min,rp_avg,sex,ag
e,smok-ing,diag):
f = open(file, 'a')
str = fam + ',' + name + ',' + otch + ',' + ip_min + ',' +
ip_avg + ','
+ rp_min + ',' + rp_avg + ',' + sex + ',' + age + ',' +
smoking + ',' + diag
+ '\n'
f.write(str) f.close()return True
```

Розглянемо більш детально компонент Interface.

Для навігації сторінками на кожній з них представлено меню зверху. Воно включає в себе пункти: «Головна» сторінка, «Діагностика», «Журнал».

Компонент Interface складається з 4 веб-сторінок. Розглянемо кожну з них.

Сторінка index.html (Головна) є стартовою сторінкою програми з коротким описом предметної області. Інтерфейс головної сторінки представимо на рисунку 17.

Рисунок 17 – Інтерфейс головної сторінки

На сторінці diagnos.html (Діагностика) є форма для введення даних пацієнта для подальшої класифікації.

До цієї форми необхідно внести необхідні для моделі дерева класифікації параметри (характеристики діелектричної проникності слини, стать, вік, статус курця). Інтерфейс представлений рисунку 18.

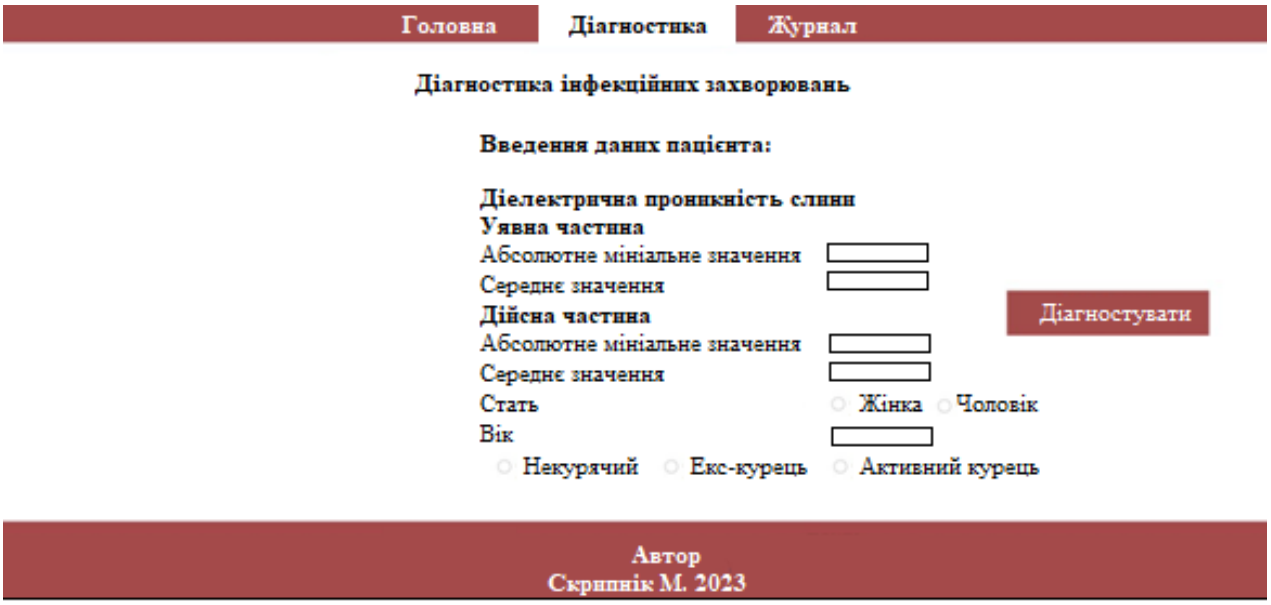

Рисунок 18 – Інтерфейс сторінки «Діагностика»

Сторінка prediction.html призначена для перегляду результатів прогнозування діагнозу респіраторної інфекції пацієнта. Виводяться введенні лікарем параметри та виводиться діагноз. Інтерфейс представлений рисунку 19.

Сторінка journal.html призначена для перегляду списку пацієнтів, дані яких були проаналізовані за допомогою програми.

Пригадаємо класифікацію пацієнтів за аналізами:

- амбулаторні та госпіталізовані пацієнти з хронічним захворюванням легень без гострої респіраторної інфекції (ХОЗЛ);
- амбулаторні та госпіталізовані пацієнти з астмою без гострих респіраторних інфекцій (астма);
- пацієнти з респіраторними інфекціями, але без хронічних захворювань легень та астми (інфіковані);
- здорових пацієнтів.

Інтерфейс представлений на рисунку 20.

Рисунок 19 – Інтерфейс сторінки «Результат»

Рисунок 20 – Інтерфейс сторінки «Результати»

4 ТЕСТУВАННЯ СИСТЕМИ ТА ОЦІНКА ЯКОСТІ МОДЕЛІ

Для оцінки якості моделей використовуються різні метрики.

Для оцінки якості роботи алгоритму кожному з класів окремо введемо метрики precision (точність), recall (повнота) та F-міра.

Precision можна інтерпретувати як частку об'єктів, названих класифікатором позитивними і при цьому дійсно позитивними, а recall показує, яку частку об'єктів позитивного класу з усіх об'єктів позитивного класу знайшов алгоритм.

Recall демонструє здатність алгоритму виявляти цей клас взагалі, а precision – здатність відрізняти цей клас від інших класів.

F-міра – середнє гармонійне precision і recall. F-мера досягає максимуму при повноті і точності, що дорівнює одиниці, і близька до нуля, якщо один з аргументів близький до нуля.

У таблиці 2 продемонстровано оцінку моделі за допомогою наступних метрик: precision, recall, f1-score.

Таблиця 2 – Оцінка точності моделі

	Precision	**Recall**	**F1-score**
Астма	0,57	0,4	0,47
ХОЗЛ	1	0,93	0,96
Здоровий	0,71	0,85	0,77
Інфікований	1	0,75	0,86

Точність навченої моделі становила 0,83, що є добрим показником.

Вибірка даних була поділена на навчальну та тестову у співвідношенні 70% на 30% відповідно.

Після розробки програми було проведено функціонально тестування з метою перевірки реалізованості функціональних вимог, тобто можливості

програмного забезпечення у певних умовах вирішувати завдання, необхідних користувачам.

Згадаємо, основні види тестування, що використовуються в роботі.

Функціональне тестування – один із видів тестування, спрямованого на перевірку відповідності функціональних вимог ПЗ до його реальних характеристик. Основним завданням функціонального тестування є підтвердження того, що програмний продукт, що розробляється, володіє всім функціоналом, необхідним замовником.

Модульне тестування – тестування кожної атомарної функціональності програми окремо, у штучно створеному середовищі. Це середовище для деякого юніту створюється за допомогою драйверів і заглушок (рис.21).

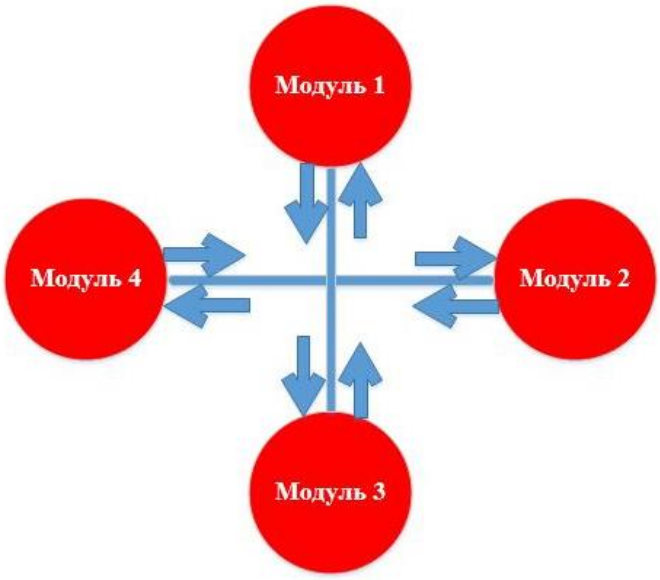

Рисунок 21 – Схема модульного тестування

Драйвер – певний модуль тесту, який виконують тестований нами елемент.

Заглушка – частина програми, яка симулює обмін даними з компонентом, що тестується, виконує імітацію робочої системи [15].

Заглушки потрібні для:

1. Імітування компонентів для роботи даного елемента.
2. Подання або повернення модулю певного значення, можливість надати тестеру самому ввести потрібне значення.
3. Відтворення певних ситуацій (виключення чи інші нестандартні умови роботи елемента).
4. Модульне тестування мотивує програмістів писати код максимально оптимізованим, проводити рефакторинг (спрощення коду програми, не торкаючись її функціональності), оскільки за допомогою юніт-тестування можна легко перевірити працездатність компонента, що розглядається.

Тест-кейс – це професійна документація тестувальника, послідовність дій, спрямована на перевірку будь-якого функціоналу, що описує як прийти до очікуваного результату.

Під тест-кейсом також може матися на увазі відповідний документ, який являє собою формальний запис тест-кейса.

Виділяють такі види тест-кейсів:

– високорівневий тест-кейс (high level test case) – тест-кейс без конкретних вхідних даних та очікуваних результатів. Як правило, такий тест-кейс обмежується загальними ідеями та операціями, схожий за своєю суттю з докладно описаними пунктами чекліста. Досить часто зустрічається в інтеграційному і системному тестуванні, а також на рівні димового тестування. Може служити відправною точкою для проведення дослідного тестування або для створення низькорівневих тест-кейсів.

– низькорівневий тест-кейс (low level test case) – тест-кейс з конкретними вхідними даними та очікуваними результатами.

Набір функціональних тестів наведено в таблиці 3.

Таблиця 3 – Тестування розробленої системи

№	Назва тесту	Кроки	Очікуваний результат	Тест пройдений?
1	Прогнозування діагнозу з незаповненими даними пацієнта	Відкрити програму. Перейти на сторінку "Діагностика". Натиснути кнопку «Діагностувати».	Програма має видати повідомлення про помилку «Для того, щоб визначити діагноз, заповніть всі поля».	Так.
2	Введення неправильних даних	Відкрити програму. Перейти на сторінку "Діагностика". Ввести літерні символи в поля для введення чисел.	Програма має видати повідомлення про помилку «Має бути введене число».	Так.
3	Прогнозування діагнозу з коректними даними	Відкрити програму. Перейти на сторінку "Діагностика". Ввести всі необхідні дані у поля введення даних. Натиснути кнопку «Надіслати».	Програма має спрогнозувати діагноз пацієнта із введеними параметрами.	Так.
4	Перехід на іншу сторінку	Відкрити програму. Перейти на сторінку, що цікавить, з меню.	Програма повинна переходити на сторінку, яку вибрав користувач у меню.	Так.

ВИСНОВКИ

В результаті виконання магістерської роботи було розроблено систему підтримки прийняття рішення для діагностики інфекційних захворювань на основі алгоритму дерева рішень

Була розглянута теоретична частина методу дерева рішень і для класифікації (принципи побудови та скорочення).

Були визначені функціональні та нефункціональні вимоги та представлені контекстна діаграма та діаграма діяльності та варіантів використання, а також спроектована система, що демонструє роботу програми для діагностики типу респіраторних інфекцій, що є веб-додатком. Було протестовано роботу програми. Система виконала своє завдання та здійснила класифікацію набору вхідних даних.

Було вирішено такі завдання:

- розглянуто принципи алгоритму дерева рішень;
- розроблено алгоритм дерева рішення для діагностики респіраторної інфекції;
- проведено тестування;
- описано результати.

У разі, коли у всьому світі борються з пандемією, стає ясною необхідність автоматизації роботи лікарів. І частково звільнення від рутинного опитування пацієнтів. Саме такі програми можуть допомогти у цьому. Не виключає діагностичного огляду, але може звільнити лікаря інших невідкладних справ. Тому я намагався зробити програму, яка б полегшила роботу лікарів під час лікування пацієнтів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Maimon, Oded Z., і Rokach, Lior. Data Mining with Decision Trees: Theory and Applications. Сінгапур, World Scientific, 2008, 16 p.
2. AstraZeneca and IBM collaborate in order to strengthen digital health expertise [Електронний ресурс] URL: https://nordiclifescience.org/astrazeneca-and-ibm-collaborate-in-order-tostrengthen-digital-health-expertise/ (дата звернення 10.10.2023 р.).
3. Класифікація [Електронний ресурс] URL: http://www.machinelearning.ru/wiki/index.php (дата звернення 10.10.2023 р.).
4. Breiman L., Friedman JH, Olshen RA, Stone CT Classification and Regression Trees. - Wadsworth, Belmont, California, 1984. - 20 p.
5. Жаркова О. С. та ін. Побудова систем підтримки ухвалення рішень у медицині на основі дерев рішень // Сучасні наукомісткі технології, 2016. – № 6. – С. 33-37.
6. Obesity and overweight World Health Organization [Електронний ресурс] URL: https://www.who.int/topics/obesity/en/(accessed 11.11.2023).
7. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник для студентов, обучающихся по специальности «Прикладная информатика» и «Прикладная математика и информатика/А.М. Вендров. М.: Финансы и статистика. 2001. 221 с.
8. Маклаков С.В. BPwin и ERwin. CASE – средства разработки информационных систем. Учебное пособие /С.В. Маклаков. Москва: Диалог – МИФИ. 1999. 256с.
9. Прошкина Е. Н. Методы и средства проектирования информационных систем и технологий. Методические указания к

выполнению лабораторных работ/Е.Н. Прошкина. Пенза: ПГУ. 2014. 92с.

10. UCI Machine Learning Repository Exasens Data Set [Електронний ресурс] URL: https://archive.ics.uci.edu/ml/datasets/Exasens (дата звернення 15.11.2023 р.).
11. Python 3 Documentation [Електронний ресурс] URL: https://docs.python.org/3/ (дата звернення 19.11.2023).
12. PyCharm – IDE для професійної розробки на Python [Електронний ресурс] URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата звернення: 19.11.2023 р.).
13. Pandas Python Data Analysis Library [Електронний ресурс] URL: https://pandas.pydata.org/(дата звернення: 22.11.2023 р.).
14. Scikit-learn: machine learning in Python [Електронний ресурс] URL: https://scikit-learn.org/stable/index.html (дата звернення: 25.11.2023 р.).
15. Куликов С.С. Тестування програмного забезпечення. Базовий курс. 2 видання. – Видавництво Чотири чверті, 2017 – С. 312.

–