МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АДАПТАЦІЇ НАВЧАЛЬНОГО КОНТЕНТУ ВИПУСКОВОЇ КАФЕДРИ ДО ВИМОГ РИНКУ ПРАЦІ

Здобувач освіти гр. ІН.м – 01н А. Є. МІЩЕНКО

Науковий керівник, А. С ДОВБИШ
професор, доктор технічних наук

Завідувач кафедри А. С. ДОВБИШ
професор, доктор технічних наук

Суми 2022

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:

зав.кафедри _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Міщенко Антону Єгоровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія адаптації навчального контенту випускової кафедри до вимог ринку праці*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи:

Результати опитування респондентів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити):

1) Огляд методів інтелектуального аналізу та класифікації даних; 2) Постановка завдання й формування завдань дослідження; 3) Розробка інформаційного та програмного забезпечення СППР; 4) Реалізація СППР; 5) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
1) Титульний слайд; 2) Актуальність; 3)Мета, об'єкт дослідження, предмет дослідження; 4)Категорійні моделі; 5) Схема алгоритму машинного навчання СППР; 6) Інформаційний критерій оптимізації параметрів машинного навчання 7) Формування вхідного інформаційного опису СППР; 8) Результати комп'ютерного моделювання; 9) Висновки.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд літератури за темою кваліфікаційної роботи		
2.	Аналіз проблеми дослідження		
3.	Опис методу розв’язання поставленої задачі		
4.	Розробка інформаційного та програмного забезпечення СППР		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

Реферат

Записка: 67 стор., 22 рис., 4 табл., 1 додаток, 33 джерела.

Мета роботи. Підвищення функціональної ефективності системи підтримки прийняття рішень для адаптації навчального контенту випускової кафедри до вимог ринку праці шляхом машинного навчання.

Об'єкт дослідження. Процес комп'ютерного оцінювання навчального контенту випускової кафедри вимогам ринку праці.

Предмет дослідження. Категорійні моделі та метод машинного навчання інформаційної системи оцінки відповідності навчального контенту вимогам ринку праці.

Методи дослідження. Інформаційно-екстремальна інтелектуальна технологія аналізу даних, методи теорії інформації та машинного навчання.

Суперечність, що вирішується у роботі. Відсутність точних та надійних автоматизованих систем для своєчасного відстеження та актуалізації застарілих навчальних програм.

Гіпотеза. Для оцінювання якості навчального контенту та навчальних програм може бути використана інформаційно-екстремальна інтелектуальна технологія аналізу даних, отриманих за результатами опитування ІТ-фахівців з актуальності навчальних програм ОПП «Інформатика» спеціальності 122 Комп'ютерні науки.

СИСТЕМА ПІДТРИМКИ ПРИЙНЯТТЯ РІШЕНЬ, ІНФОРМАЦІЙНО-ЕКСТРЕМАЛЬНА ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ, ДЕКУРСИВНЕ ДЕРЕВО, МАШИННЕ НАВЧАННЯ.

ЗМІСТ

ВСТУП

У період інформатизації суспільства особливо гостро постає проблема підвищення якості вищої освіти. Стрімкий розвиток інформаційних технологій вимагає оперативної реакції на будь-які зміни зі сторони навчальних закладів, успішним може бути лише той навчальний заклад, який навчає студентів актуальним технологіям, які вони можуть застосовувати у своїй роботі. Немає жодного сенсу навчати застарілим речам. При цьому, навчальні заклади є достатньо вільними у формуванні освітніх програм, їх вдосконаленні та актуалізації. Тому важливим завданням є забезпечення постійного контролю основних показників якості освіти та розробка рекомендацій щодо поліпшення усіх складових підготовки фахівців. Однією із таких складових є удосконалення системи моніторингу потреб ринку праці з метою адаптації навчального контенту до сучасних вимог.

Актуальність отриманих знань найкраще можуть оцінити студенти-випускники та роботодавці. Тому постає задача розроблення системи зворотного зв’язку між ними та навчальними закладами. При цьому необхідно мати механізми обробки та аналізу отриманих відповідей респондентів. Для цього є доцільним використання інформаційно-інтелектуальних технологій аналізу даних.

Метою магістерської кваліфікаційної роботи є створення системи підтримки прийняття рішень (СППР) для оцінки якості навчального контенту спеціальності «Комп’ютерні науки» бакалаврського рівня на основі машинного навчання та розпізнавання образів. Дана СППР може дозволити:

- підвищити оперативність корекції навчального контенту випускової кафедри до вимог ринку праці;
- суттєво зменшити матеріальні та часові витрати на оброблення результатів опитування роботодавців;
- суттєво зменшити вплив суб’єктивного фактору на оцінку ступеню адаптації навчального контенту до вимог ринку праці.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Проблематика підготовки фахівців в Україні

На сьогодні до проблем підготовки фахівців в Україні можна віднести наступні [1–3]:

- невідповідність структури знань, умінь у молодих фахівців виробничим вимогам унаслідок розриву раніше наявних системних зв’язків між виробництвом, наукою та освітою;
- проблема оцінки рівня конкурентоспроможності підготовки фахівців і виявлення чинників його підвищення;
- проблема неадекватності вмісту навчального плану і робочих програм необхідним компетенціям відповідно до потреб розвитку ринку праці та регіону;
- проблема формування контингенту студентів із мотиваційними установками і рівнем професійної придатності, адекватними вимогам вмісту ОПП, скоректованою відповідно до потреб розвитку регіону, ринку праці;
- проблема розвитку системи моніторингу і контролю якості підготовки фахівців, що включає: а) систему внутрішньої оцінки якості підготовки, а саме методи оцінки при вхідному, поточному і підсумковому контролі якості підготовки фахівців; б) систему зовнішньої оцінки якості підготовки фахівців;
- проблема вдосконалення навчального процесу у напрямі розвитку конкурентних переваг у майбутніх фахівців
- проблема залучення роботодавців до внесення коректив у підготовку майбутніх фахівців;

- проблема забезпечення наступності дворівневої підготовки фахівця в системі (дотримання наступності у побудові навчальних планів та програм);
- проблема створення системи управління якістю підготовки у ВНЗ, оскільки невелика кількість українських університетів упроваджують системи управління якістю та отримують міжнародні сертифікати.

Говорячи про оцінювання навчальних програм перше що спадає на думку це їх акредитація з боку держави. А саме, критерії оцінювання якості освітньої програми чітко визначено у [1]. Проте говорячи про оцінювання актуальності навчальних програм необхідно враховувати відгуки безпосередньо випускників та роботодавців. Подібне опитування можна віднести до соціологічних. Зазвичай опрацювання результатів такого оцінювання проводиться за допомогою експертної оцінки або з використанням методів статистичного аналізу. Загальні методики створення, проведення та опрацювання результатів подібних опитувань описано у роботах [4–6]. Подібний спосіб оцінювання якості освіти описано у [7]. У цій праці запропоновано розраховувати інтегральний показник якості освітньої програми на основі відгуків від двох експертних груп – студентів та роботодавців – про сприйняття та очікування якості освіти, які дозволяють вирахувати задоволеність якістю. Але подібні методи потребують залучення експертів для початкового визначення критеріїв оцінювання (у наведеному прикладі – оцінку експертами важливості кожного з критеріїв) та будь-яких їх змін надалі. До того ж їх складно автоматизувати.

1.2 Методи інтелектуального аналізу та класифікації даних

Допомогти у розв'язанні цієї проблеми можуть методи інтелектуального аналізу та класифікації даних. Задача класифікації полягає у визначенні того,

до якого з набору класів належать об’єкти. Для розв’язання задач класифікації даних використовують наступні методи: метод найближчого сусіда, метод k-найближчих сусідів, баєсів класифікатор, метод індукції дерев рішень, нейронні мережі та інші.

Задача пошуку найближчого сусіда є однією з фундаментальних задач обчислювальної геометрії. Базова постановка задачі звучить так [8]: для заданого набору P який складається з n точок у метричному просторі (X, D) необхідно побудувати таку структуру даних, щоб для будь-якої точки q цього простору можна було отримати точку з P, яка є найближчою до q. Цей метод використовується для класифікації розпізнавання образів, статистичної класифікації, побудови рекомендаційних систем, моделей комп’ютерного зору, у кластер аналізі, тощо. Базова задача пошуку найближчого сусіда може бути трансформована у такі задачі як: пошук k-найближчих сусідів, пошук приблизного найближчого сусіда, пошук сусіда у фіксованому радіусі, пошук усіх найближчих сусідів, пошук відношення відстані до найближчого сусіда.

Метод k-найближчих сусідів є похідним від методу пошуку найближчого сусіда. Він працює наступним чином [9]: необхідно визначити k найближчих класів до заданого об’єкта і зберегти інформацію про те, до якого класу вони належать. Тоді відносимо заданий об’єкт до того класу, об’єктів якого з них найбільше серед k найближчих сусідів (нічия розв’язується випадково). Таким чином, цей метод можна назвати узагальненням методу найближчого сусіда при $k = 1$.

Баєсів класифікатор використовую теорему Баєса для того, щоб знаходити ймовірність належності об’єкта до якогось класу. При цьому припускається що змінні є незалежними, тобто $P(X|C) = \prod_{u=1}^{n} P(X_i|C)$, де $X = (X_1, \dots X_n)$ це вектор змінних, а C – клас. Баєсів класифікатор найкраще працює у предметних областях, де припущення є вірним. В іншому випадку дещо втрачається точність роботи. Баєсів класифікатор може як однозначно визначати приналежність об’єкта до певного класу, так і давати ймовірність

цього. Дуже часто він застосовується для класифікації текстів (наприклад, у системах фільтрації спаму).

Дерево рішень – це структура, яка має кореневий вузол, гілки, проміжні та кінцеві вузли. Кореневий та проміжні вузли містять перевірки характеристик, гілки – їх значення, а кінцеві вузли – мітки класів [10]. Рухаючись з кореневого вузла до кінцевого можна класифікувати об'єкт «відповідаючи на питання» у проміжних вузлах. Індукція дерева рішень – це процес його навчання (побудови дерева). Дерево рішень легко навчити рекурсивно, проте у такому випадку його структура може бути неоптимальною [10]. До найбільш поширених алгоритмів індукції дерева рішень відносяться ID3, C4.5, CART.

Нейронна мережа – система, яка побудована за принципами функціонування біологічних нейронних мереж. Нейронна мережа є сукупністю нейтронів. Нейрон – це елемент, головна функція якого полягає у формуванні вихідного сигналу залежно від сигналів, що надходять на його входи. [11]. Нейрони пов'язані між собою зв'язками, які мають свої значення ваги. І тоді вхідний сигнал нейрона обчислюється наступною формулою [12]: $net_j = \sum_{i=1}^{N} x_i w_{ij}$, де net_j – комбінований вхід j-го нейрона, w_{ij} – вага зв'язку між i-им та j-им нейронами. Після проходження сигналів через мережу на виході отримується результат – відповідь мережі. Існує багато різновидів архітектур нейронних мереж, які задають структуру та типи нейронів: прямого розповсюдження, перцептрон, мережа радіально-базисних функцій, нейромережа Хопфілда, машина Больцмана, автокодувальник, згорткові та глибинні згорткові мережі, розгорткові мережі. Найчастіше нейромережі застосовуються для розпізнавання образів та класифікації даних.

Інформаційно-інтелектуальна екстремальна технологія (IEIT) це сучасна технологія машинного навчання, яку розроблено у Сумському державному університеті під керівництвом професора Довбиша А.С. Становлення IEIT відбувалося під впливом ідей А.А. Харкевича, Клода

Шеннона, О.Г. Івахненка, І.В. Кузьміна, В.І. Костюка, О.А. Павлова та інших іноземних і вітчизняних вчених. За порівняно недовгий час з моменту появи, ІЕІТ була використана у багатьох сферах людської діяльності: медицині [13,14], промисловості [15,16], військовій справі [17,18], кібербезпеці [19,20], економіці [21,22], тощо. Успішне використання технології для різних цілей підтверджує її працездатність.

Базовою працею, яку присвячено ІЕІТ є [23]. Для функціональної ефективності систем на основі ІЕІТ зазвичай використовують міри Кульбака або Шеннона (або їх модифікації) [24,25]. Важливим питанням при розробці системи з використанням ІЕІТ є вибір базового класу для машинного навчання. Цю тему досліджено у [26]. Побудові ієрархічної структури даних класів розпізнавання присвячено статтю [27]. Оптимізацію словника ознак досліджено у [28–30].

ІЕІТ також можна застосовувати й у навчальному процесі (в тому числі й для адаптації навчального контенту). Це питання розглядалося ще у [31]. Перші спроби використання ІЕІТ для оцінки актуальності навчального контенту були зроблені у [32,33]. Проте дані розробки не використовували ієрархічну структуру даних для алфавіту класів.

Таким чином, розробка системи адаптації навчального контенту до вимог ринку праці на основі інформаційно-екстремальної інтелектуальної технології, є актуальною, у зв’язку з відсутністю подібних автоматизованих систем, які б використовували ієрархічну структуру даних.

1.3 Постановка задачі

Метою дослідження є створення системи підтримки прийняття рішень (СППР) для оцінки якості навчального контенту спеціальності «Комп’ютерні науки» бакалаврського рівня на основі машинного навчання та розпізнавання образів в рамках інформаційно-екстремальної інтелектуальної технології

аналізу даних. Ця технологія базується на максимізації інформаційної спроможності системи в процесі машинного навчання[31].

Нехай дано алфавіт класів $\{X_m^0 \mid m = \overline{1,M}\}$, де M – кількість класів розпізнавання. Оскільки оцінювання відбувається за європейською системою, то M = 6. Тому було вирішено перейти від лінійних вирішальних правил до ієрархічних. Таким чином було отримано ієрархічну структуру класів $\{X_{h,s,m}^0 \mid h = \overline{1,H}, s = \overline{1,S}, m = \overline{1,M}\}$, де H – кількість ярусів ієрархічної структури; S – кількість страт на h -му ярусі; M – кількість класів розпізнавання в s-й страті.

Для кожного класу маємо вхідну навчальну матрицю, стовпчики якої містять значення навчальної вибірки, а рядки – реалізації: $||y_{h,s,m,i}^{(j)} \mid i = \overline{1,N}, j = \overline{1,n}||$, де N – кількість ознак розпізнавання класу $X_{h,s,m}^0$; n – кількість векторів-реалізацій.

Параметри навчання системи розпізнавання для класу $X_{h,s,m}^0$ задають наступний вектор: $g_{h,s} =< x_{h,s,m}, d_{h,s,m}, \delta_{Kh,s,m,i} >$, де $x_{h,s,m}$ – двійковий усереднений вектор-реалізація, який визначає геометричний центр контейнеру класу $X_{h,s,m}^0$, $d_{h,s,m}$ – радіус гіперсфери контейнера класу $X_{h,s,m}^0$, $\delta_{Kh,s,m,i}$ – половина поля контрольних допусків i-ї ознаки усередненого вектору-реалізації класу розпізнавання.

Також маємо наступні обмеження:

1. $d_{h,s,m} \in [0; d(x_{h,s,m} \oplus x_{h,s,m}) - 1]$
2. $\delta_{K,h,s,i} \in [0; \delta_{E,h,s,i}/2]$, де $\delta_{E,h,s,i}$ – поле контрольних допусків i-ї ознаки усередненого вектору-реалізації класу розпізнавання
3. Машинне навчання системи полягає у наступному:
4. оптимізувати параметри вектора $g_{h,s}$ за усередненим інформаційним критерієм: $\overline{E_{h,s}} = \frac{1}{M}\sum_{m=1}^{M} \max_{G_E \cap G_d} E_{h,s,m}(d_{h,s,m})$, де $E_{h,s,m}(d_{h,s,m})$ – інформаційний критерій оптимізації параметрів машинного

навчання, G_E – робоча область визначення критерію, G_d – допустима зміна радіуса контейнеру;

5. з використанням оптимальних параметрів контейнерів класів побудувати вирішальні правила на кожній страті.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Сформувати вхідну навчальну матрицю;
2. Розробити категорійну модель машинного навчання;
3. Розробити та реалізувати алгоритм машинного навчання;
4. Оцінити функціональну ефективність розроблених моделі та алгоритму.

2 ОПИС МЕТОДУ ДОСЛІДЖЕННЯ

2.1 Основні положення інформаційно-екстремальної інтелектуальної технології

Основна ідея машинного навчання у рамках ІЕІТ полягає в трансформації апріорного у загальному випадку нечіткого розбиття простору ознак у чітке розбиття класів еквівалентності шляхом ітераційної оптимізації параметрів функціонування ІС [23]. Особливістю ІЕІТ є процес оптимізації контрольних допусків, який у тому числі трансформує вхідний нечіткий розподіл реалізацій образів у чіткий. Це дозволяє будувати безпомилкові вирішальні правила.

Нехай дано $\{X_m^o \mid m=\overline{1,M}\}$ – алфавіт класів розпізнавання. Розбиття простору ознак на класи розпізнавання є нечітким розбиттям $\tilde{\tilde{\Re}}^{|M|}$, а елементи розбиття є нечіткими класами розпізнавання. Воно відповідає наступним умовам [23]:

$$
\begin{aligned}
&1)\ \left(\forall X_m^o \in \tilde{\tilde{\Re}}^{|M|}\right)\left[X_m^o \neq \varnothing\right];\\
&2)\ \left(\exists X_k^o \in \tilde{\tilde{\Re}}^{|M|}\right)\left(\exists X_l^o \in \tilde{\tilde{\Re}}^{|M|}\right)\left[X_k^o \neq X_l^o \rightarrow X_k^o \bigcap X_l^o \neq \varnothing\right];\\
&3)\ \left(\forall X_k^o \in \tilde{\tilde{\Re}}^{|M|}\right)\left(\forall X_l^o \in \tilde{\tilde{\Re}}^{|M|}\right)\left[X_k^o \neq X_l^o \rightarrow KerX_k^o \bigcap KerX_l^o = \varnothing\right];\\
&4)\ \bigcup_{X_m^o \in \tilde{\tilde{\Re}}} X_m^o \subseteq \Omega_{Б};\ \ k \neq l;\ k,l,m=\overline{1,M}\ .
\end{aligned}
\tag{2.1}
$$

У бінарному просторі ознак контейнер класу розпізнавання є гіперсферою. У радіальному базисі його можна представити у вигляді гіперсфери. Тоді радіус цієї гіперсфери буде визначатися за формулою [23]:

$$
d_m = \sum_{i=1}^{N} (x_{m,i} \oplus \lambda_i),
\tag{2.2}
$$

де $x_{m,i}$ – i-та координата еталонного вектора x_m; λ_i – i-та координата деякого вектора λ, вершина якого належить контейнеру $K_m^o \in X_m^o$.

У радіальному базисі простору ознак відновлення оптимального контейнера відбувається за наступною процедурою [23]:

$$d_m(k) = [d_m(k-1) + h \mid d_m(k) \in G_m^d], \tag{2.3}$$

де k – змінна числа збільшень радіуса контейнера K_m^o; h – крок збільшення радіуса; G_m^d – область допустимих значень радіуса d_m.

Для того, щоб уникнути ситуації коли один із класів поглинає ядро сусіднього класу (тобто такого, який має мінімальну міжцентрову відстань до даного) вирази (2.1) доповнюються наступною умовою [23]:

$$\left(\forall X_k^o \in \tilde{\Re}^{|M|}\right)\left(\forall X_l^o \in \tilde{\Re}^{|M|}\right)[X_k^o \neq X_l^o \rightarrow (d_k^* < d(x_k \oplus x_l))\,\&$$
$$\&\,(d_l^* < d(x_k \oplus x_l))], \tag{2.4}$$

де d_k^*, d_l^* – оптимальні радіуси контейнерів K_k^o і K_l^o відповідно.

Таким чином, за IEIT технологією вирішальні правила будуються у багатоциклічній процедурі пошуку максимального граничного усередненого інформаційного критерію оптимізації машинного навчання [23]:

$$g_\xi^* = \arg\max_{G_\xi}\{\max_{G_{\xi-1}}\{\ldots\{\max_{G_1 \cap G_E} \frac{1}{M}\sum_{m=1}^{M} E_m\}\ldots\}\}, \tag{2.5}$$

де E_m – інформаційний критерій оптимізації параметрів машинного навчання; G_ξ – допустима область значень ξ-го параметра машинного

навчання; G_E – робоча (допустима) область визначення функції інформаційного критерію.

2.2 Базовий алгоритм машинного навчання

Базовий алгоритм оптимізації параметрів функціонування ІС реалізується у внутрішньому циклі процедури (2.1). Призначенням базового алгоритму навчання є [23]:

- оптимізація геометричних параметрів контейнерів класів розпізнавання;
- обчислення інформаційного КФЕ навчання системи;
- пошук глобального максимуму КФЕ у робочій (допустимій) області визначення його функції.

Етапи реалізації базового алгоритму [23]:

1) Формування бінарної навчальної матриці $\| x_{m,i}^{(j)} \|$.

2) Формування масиву еталонних двійкових векторів-реалізацій $\{x_{m,i} \mid m = \overline{1, M},\ i = \overline{1, N}\}$, елементи яких визначаються за правилом

$$x_{m,i} = \begin{cases} 1, & if \quad \frac{1}{n}\sum_{j=1}^{n} x_{m,i}^{(j)} > \rho_m; \\ 0, & if \quad else, \end{cases} \tag{2.6}$$

де ρ_m – рівень селекції координат вектора $x_m \in X_m^o$.

3) Розбиття множини еталонних векторів на пари найближчих «сусідів»: $\Re_m^{|2|} = <x_m, x_l>$, де x_l – еталонний вектор сусіднього класу X_l^o, за такою схемою алгоритму:

а) структурується множина еталонних векторів, починаючи з вектора x_1 базового класу X_1^o, який характеризує найбільшу функціональну ефективність ІС;

б) будується матриця кодових відстаней між еталонними векторами розмірності $M \times M$;

в) для кожного рядка матриці кодових відстаней знаходиться мінімальний елемент, який належить стовпчику вектора, найближчого до вектора, що визначає рядок. За наявності декількох однакових мінімальних елементів вибирається з них будь-який, оскільки вони є рівноправними;

г) формується структурована множина елементів попарного розбиття $\{\Re_m^{|2|} \mid m = \overline{1, M}\}$, яка задає план навчання.

4) Оптимізація кодової відстані d_m відбувається за рекурентною процедурою $d_m(k) = [d_m(k-1) + h \mid d_m(k) \in G_m^d]$. При цьому береться $E_m(0) = 0$.

5) Процедура закінчується при знаходженні максимуму КФЕ в робочій області його визначення: $E_m^* = \max\limits_{\{d\}} E_m$, де $\{d\} = \{d_1, ..., d_k, ..., d_{\max}\} \in [0; d(x_m \oplus x_l) - 1]$ – множина радіусів концентрованих гіперсфер, центр яких визначається вершиною еталонного вектора $x_m \in X_m^o$. При цьому множина $\{d\}$ є так само множиною кроків навчання ІС.

Категорійну модель машинного навчання за базовим алгоритмом зображено на рисунку 2.1 [24]:

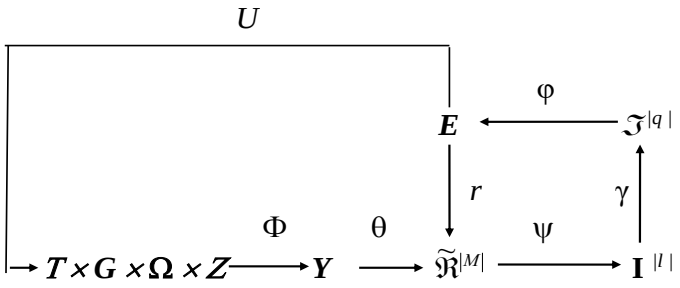

Рисунок 2.1 – Категорійна модель машинного навчання за базовим алгоритмом

На рисунку 2.1 прийнято такі позначення:

$T \times G \times \Omega \times Z$ – джерело інформації (T – множина моментів часу формування векторів-реалізацій класів розпізнавання; G – фактори, що впливають на функціонування системи; Ω – простір діагностичних ознак; Z – простір технічних станів системи, які визначають алфавіт класів розпізнавання);

Y – вхідна навчальна матриця;

$I^{(l)}$ – множина, що перевіряє основну статистичну гіпотезу;

$\mathfrak{I}^{|q|}$ – множина точнісних характеристик;

E – множина значень інформаційного критерію;

D – система контрольних допусків;

$\tilde{\Re}^{|M|}$ – розбиття простору ознак на класи розпізнавання;

θ – оператор нечіткої факторизації простору ознак: $\theta : Y \to \tilde{\Re}^{|M|}$

ψ – оператор класифікації який перевіряє основну статистичну гіпотезу про належність реалізацій $\{ x_m^{(j)} | j=\overline{1,n} \}$ нечіткому класу $\psi : \tilde{\Re}^{|M|} \to \mathrm{I}^{|l|}$, X_m^o;

γ – оператор, який шляхом оцінки статистичних гіпотез формує множину точнісних характеристик $\mathfrak{I}^{|q|}$: γ: $I^{|l|} \to \mathfrak{I}^{|q|}$;

φ – оператор, який обчислює множину значень інформаційного КФЕ, який є функціоналом точнісних характеристик: $\varphi : \mathfrak{I}^{|q|} \to E$;

r – оператор, який замикає контур оптимізації геометричних параметрів нечіткого розбиття $\tilde{\Re}^{|M|}$ r: $E \to \tilde{\Re}^{|M|}$;

U – оператор, який регламентує процес навчання: U: $T \times G \times \Omega \times Z$.

2.3 Ієрархічная структура даних

ІЕІТ підтримує використання ієрархічної структури даних у вигляді декурсивного дерева. Це дозволяє зменшити вплив багатовимірності ознак на

ефективність машинного навчання. Приклад такої структури зображено на рисунку 2.2:

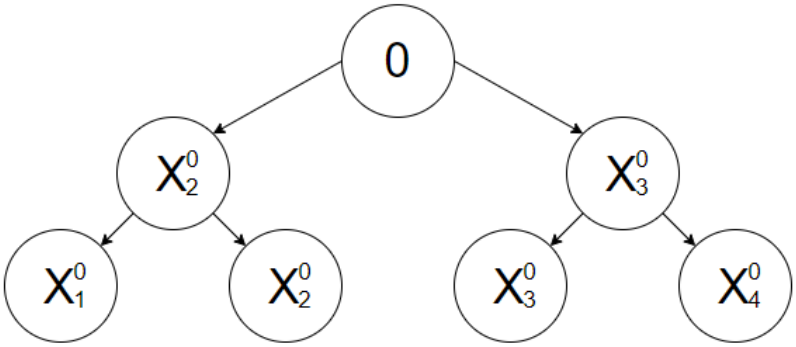

Рисунок 2.2 – Ієрархічна структура даних у вигляді декурсивного дерева

За такої структури атрибут з батьківської страти передається на одну з вершин дочірньої страти. У такому випадку, машинне навчання відбувається лише для двох класів на кожній зі страт. Таким чином, подання ієрархічної структури даних у вигляді декурсивного бінарного дерева дозволяє автоматично розбивати алфавіт класів розпізнавання великої потужності на пари найближчих сусідів.

2.4 Інформаційний критерій оптимізації параметрів машинного навчання

Як інформаційний критерій оптимізації параметрів машинного навчання було обрано модифіковану міру Кульбака у вигляді [24]:

$$E_m^{(k)} = \log_2\left(\frac{2-(\alpha_m^{(k)}(d)+\beta_m^{(k)}(d))}{\alpha_m^{(k)}(d)+\beta_m^{(k)}(d)}\right) * \left[1-(\alpha_m^{(k)}(d)+\beta_m^{(k)}(d))\right], \qquad (2.7)$$

де $\alpha_m^{(k)}(d)$ – помилка першого роду при прийнятті рішень на k-му кроці машинного навчання; $\beta_m^{(k)}(d)$ – помилка другого роду; d – дистанційна міра, яка визначає радіуси гіперсферичних контейнерів класів розпізнавання, побудованих в радіальному базисі простору Хеммінга.

Нормована модифікація критерію (2.7) [24]:

$$E_{K,m}^{(k)} = \frac{E_{K\,m}^{(k)}}{E_{K\max}^{(k)}}, \tag{2.8}$$

де $E_{K\max}^{(k)}$ – значення інформаційного критерію при $D_{1,m}^{(k)}(d) = D_{2,m}^{(k)}(d) = 1$ і $\alpha_m^{(k)}(d) = \beta_m^{(k)}(d) = 0$ для формули (2.7).

При цьому оцінки помилки першого та другого роду мають наступний вигляд:

$$\alpha_m^{(k)}(d) = \frac{K_{1m}^{(k)}}{n_{\min}}; \qquad \beta_m^{(k)}(d) = \frac{K_{2m}^{(k)}}{n_{\min}}, \ldots \tag{2.9}$$

де $K_{1,m}^{(k)}$ – кількість подій, при яких реалізації, що належать класу X_m^o, помилково до нього не відносяться; $K_{3,m}^{(k)}$ – кількість подій, при яких помилково належать класу розпізнавання X_m^o реалізації сусіднього класу розпізнавання X_c^o; $n_{\min}$ – мінімальний обсяг навчальної вибірки.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО, АЛГОРИТМІЧНОГО ТА ПРОГРАМНОГО ЗАБЕСПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідного математичного опису

Навчальний контент оцінювався за європейською системою, тобто кількість класів розпізнавання, які характеризували відповідні рівні якості, дорівнювала шести. Для кожного з класів (A, B, C, D, E, F) було визначено центр розсіювання векторів ознак розпізнавання та радіус його контейнера:

$$X_F^0: C_F = 56, d_F=6$$
$$X_E^0: C_E = 64, d_E=5$$
$$X_D^0: C_D = 72, d_D=5$$
$$X_C^0: C_C = 79, d_C=5$$
$$X_B^0: C_B = 86, d_B=5$$
$$X_A^0: C_A = 94, d_A=6$$

Положення класів у просторі зображено на рисунку 3.1. Клас F позначено помаранчевим кольором, E – зеленим, D – синім, C – червоним, B – фіолетовим та A – чорним:

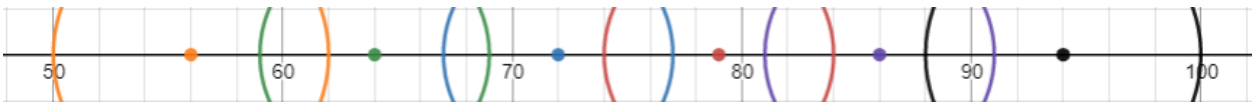

Рис. 3.1 – Положення класів розпізнавання у просторі

Вхідний інформаційний опис СППР створювався імітаційним шляхом. Для кожного класу було згенеровано 40 реалізацій. При цьому 20% реалізацій класу належить кожному перетину класу з іншими класами.

3.2 Ієрархічна структура даних

Оскільки маємо 6 класів, які розташовані на одній прямій у просторі (рис. 3.1), то інформаційно-екстремальне машинне навчання СППР здійснювалося за ієрархічною структурою даних у вигляді декурсивного бінарного дерева.

Побудова декурсивного бінарного дерева здійснювалася за схемою:

1) алфавіт $\{X_m^o \mid m = \overline{1,6}\}$ впорядкованих класів розпізнавання розбивається на дві групи, які визначають відповідно дві гілки декурсивного дерева;

2) як атрибути вершин верхнього (першого за дендрографічною класифікацією) ярусу декурсивного дерева вибираються навчальні матриці сусідніх межевих для кожної із груп класів розпізнавання, тобто класи X_3^o і X_4^o;

3) атрибути страти верхнього ярусу переносяться у вершини відповідних страт нижнього ярусу;

4) страти нижніх ярусів кожної гілки дерева містять крім транспортованої з верхнього ярусу навчальної матриці також навчальну матрицю найближчого сусіднього в своїй групі класу розпізнавання;

5) побудова дерева продовжується до тих пір, поки не будуть сформовані фінальні страти, які містять навчальні матриці всіх класів розпізнавання.

Таким чином, побудоване за вище наведеною схемою бінарне декурсивне дерево розбиває заданий алфавіт на страти, кожна з яких містить по два найближчих сусідніх класи, що дозволяє для класів розпізнавання кожної фінальної страти застосовувати лінійний алгоритм інформаційно-екстремального машинного навчання. При цьому побудова безпомилкових за навчальною матрицею вирішальних правил досягається вибором необхідної глибини машинного навчання.

На рисунку 3.2 показано побудоване декурсивне дерево, в якому вершини позначено латинськими літерами класів розпізнавання:

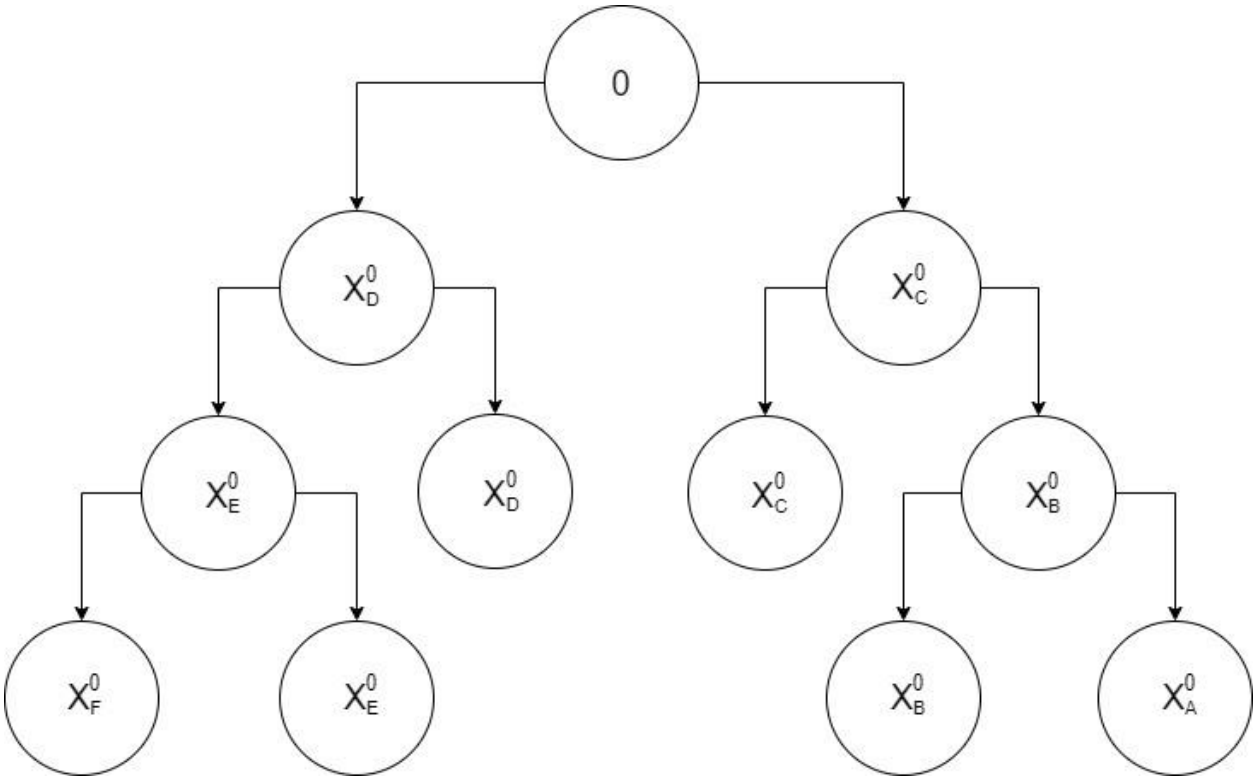

Рисунок 3.2 – Ієрархічна структура даних у вигляді декурсивного бінарного дерева

Рисунок 3.2 показує, що машинне навчання необхідно провести для п’яти фінальних страт: (X_F^0, X_E^0), (X_E^0, X_D^0), (X_D^0, X_C^0), (X_C^0, X_B^0), (X_B^0, X_A^0).

3.3 Алгоритм інформаційно-екстремального машинного навчання з оптимізацією контрольних допусків на ознаки розпізнавання з використанням ієрархічної структури даних

Оптимізацію контрольних допусків на ознаки розпізнавання можна проводити за паралельним (коли допуски оптимізуються одночасно для всіх ознак) або послідовним (коли допуски оптимізуються послідовно для кожної

з ознак розпізнавання). Паралельний алгоритм є швидким та оптимальним при використанні ієрархічної структури даних.

Алгоритм паралельної оптимізації [15]:

1) обнуляється лічильник кроків зміни параметра δ: $l := 0$;

2) запускається лічильник $l := 1+1$ і обчислюються нижні та верхні контрольні допуски для всіх ознак: $\{A_{HK,i}[l] \coloneqq y_{1,i} - \delta[l]\}$ і $\{A_{HK,i}[l] \coloneqq y_{1,i} + \delta[l]\}, i = \overline{1,N}$, де $y_{1,i}$ – вибіркове середнє значення i-ї ознаки для векторів-реалізацій класу X_1^0, який є найбільш бажаним для особи, що приймає рішення;

3) реалізується базовий алгоритм навчання, задачами якого є обчислення на кожному кроці навчання значення інформаційного КФЕ і пошук у робочій області визначення його функції її глобального значення.

4) якщо $E_l^*[l] \geq E_l^*[l-1]$, то виконується пункт 5, інакше - пункт 6;

5) якщо $\delta \leq \delta_H/2$, то виконується пункт 2, інакше - пункт 6;

$\{A_{HK,i}^* \coloneqq A_{HK,i}[l-1]\}$; $\{A_{BK,i}^* \coloneqq A_{BK,i}[l-1]\}, i = \overline{1,N}$

6) ЗУПИН.

Категорійна функціональна модель машинного навчання СППР з оптимізацією контрольних допусків на ознаки розпізнавання для кожної страти декурсивного дерева показана на рисунку 3.3, де до категорійної моделі базового алгоритму навчання додано контур оптимізації контрольних допусків:

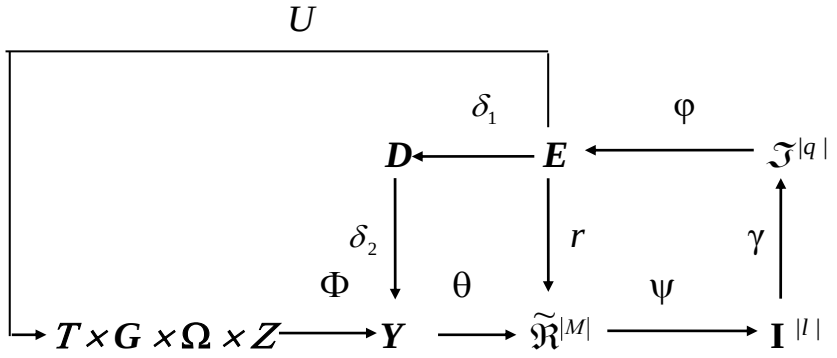

Рисунок 3.3 – Категорійна модель навчання ІС за базовим алгоритмом з використання паралельно-послідовного алгоритму оптимізації СКД

На рисунку 3.3 контур оптимізації контрольних допусків замикається через терм-множину *D*, яка містить значення контрольних допусків на ознаки розпізнавання.

Оскільки навчання буде відбуватися з використанням ієрархічної структури даних, то необхідно застосовувати наступний алгоритм машинного навчання [27]:

1) Обнуління лічильника варіантів ієрархічних структур (кроків навчання): $r:=0$;

2) ініціалізація лічильника варіантів ієрархічних структур: $r:=r+1$;

3) обнуління лічильника ярусів структури даних: $h:=0$;

4) ініціалізація лічильника ярусів структури даних: $h:=h+1$;

5) обнуління лічильника страт ярусу: $s:=0$;

6) ініціалізація лічильника ярусу: $s:=s+1$;

7) для кожної s-ї страти h-го ярусу r-ї ієрархічної структури реалізується інформаційно-екстремальний алгоритм навчання з паралельною оптимізацією контрольних допусків на діагностичні ознаки, який обчислює усереднене по всім стратам ярусу максимальне значення інформаційного критерію $\bar{E}^*_{r,h,s}$;

8) якщо $s \leq S_h$, де S_h – кількість страт на h -му ярусі, то виконується пункт 6, інакше – пункт 9;

9) якщо $h \leq h_{max}$, де h_{max} – кількість ярусів r-ї структури даних, то виконується пункт 4, інакше – пункт 10;

10) обчислюється усереднене по всім ярусам структури даних максимальне значення інформаційного критерію оптимізації $\bar{E}^*_{r,h}$;

11) якщо $r \leq r_{max}$, де r_{max} – кількість ієрархічних структур даних, то виконується пункт 2, інакше – пункт 12;

12) визначається оптимальна ієрархічна структура даних: $h^*_r = arg \max\limits_{(r)} \bar{E}^*_{r,h}$

13) ЗУПИН

Таким чином, до категорійної моделі (рис. 3.3) необхідно додати новий контур оптимізації (рис. 3.4).

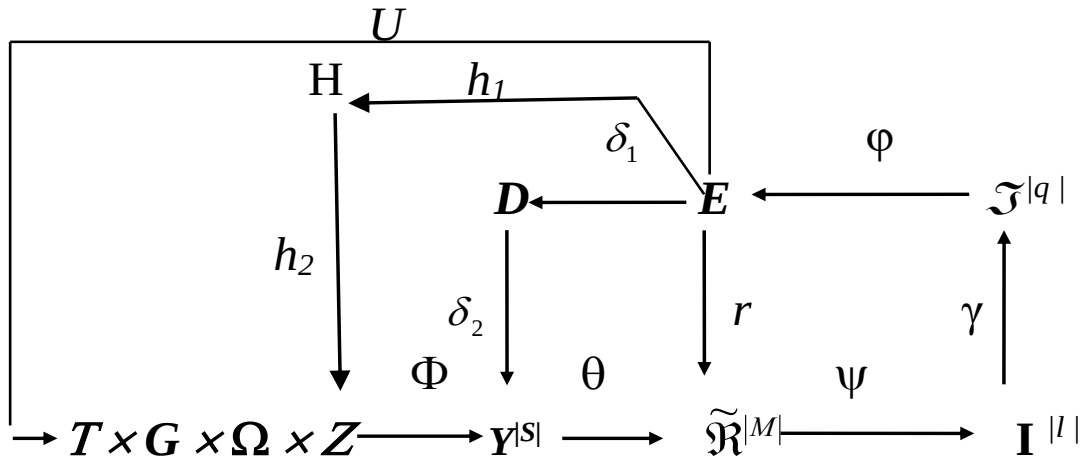

Рисунок 3.4 – Категорійна модель ієрархічного машинного навчання СППР

На рисунку 3.4 прийнято такі додаткові позначення:

H – множина варіантів структур даних в заданій ієрархічній структурі;

h_1 – оператор вибору страти в заданій ієрархічній структурі: h_1: $E \rightarrow H$;

h_2 – оператор формування алфавіту класів розпізнавання для кожної страти декурсивного дерева;

$Y^{|S|}$ – множина навчальних матриць S страт декурсивного дерева.

3.4 Опис програмної реалізації

Алгоритм машинного навчання СППР для оцінки якості навчального контенту було реалізовано на мові Java. Нижче наведено класи та опис коду.

Лістінг програми приведено у Додатку А.

Клас IEITCartesianCoordinates реалізує алгоритм інформаційно-екстремального машинного навчання за ієрархічною структурою даних у вигляді декурсивноготдерева.

Таблиця 3.1 – Опис складових класу IEITCartesianCoordinates

Методи		
Метод	**Змінні**	**Призначення**
void launchHierarchical	List<String> classesNames – список імен класів (шляхів до реалізацій)	Метод для машинного навчання з використанням декурсивного дерева
List<TreePair> getTreePairs	List<String> classesNames –список імен класів (шляхів до реалізацій)	Метод для побудови декурсивного дерева із заданих класів розпізнавання. Повертає декурсивне дерево із заданими класами розпізнавання
void launchParallel	TreePair treePair – страта декурсивного дерева	Метод для машинного навчання з паралельною оптимізацією контрольних допусків
	int baseClass – базовий клас у страті	

int getOptimalDeltaParallel	List<int[][]> classesValues – реалізації класів	Метод для отримання оптимального значення дельти для СКД з паралельною оптимізацією контрольних допусків. Повертає оптимальне значення дельти для СКД
	int baseClass – базовий клас	
List<Integer> getRadii	int[][] classVectors – еталонні вектори кожного класу	Метод для отримання оптимальних радіусів контейнерів кожного класу. Повертає список радіусів контейнерів усіх класів
	List<int[][]> classBinaryMatrices – бінарні матриці класів	
List<List<CriterionValue>> getCriterionValuesForClassesAndRadii	int[][] classVectors – еталонні вектори кожного класу	Метод для обчислення значення критерію для класів та радіусів їх контейнера.
	List<int[][]> classBinaryMatrices –	

	бінарні матриці класів	Повертає значення критерію для класів та радіусів їх контейнера
	int[][] pairs – сусідні класи	
int[][] makePairs	int[][] classVectors – еталонні вектори кожного класу	Метод для пошуку сусідів кожного класу. Повертає сусідів кожного класу
List<Integer> getDistancesBetweenVectorAndBinaryMatrix	int[] vector – вектор	Метод для пошуку відстаней між вектором та рядками бінарної матриці. Повертає відстані між вектором та рядками бінарної матриці
	int[][] binaryMatrix – бінарна матриця	
int getDistanceBetweenVectors	int[] firstVector – перший вектор	Метод для пошуку відстаней між двома векторами. Повертає відстань між двома векторами
	int[] secondVector – другий вектор	

int[][] txtToArray	String classPath – шлях до файлу з реалізаціями класу	Метод для отримання оцінок з файлу. Повертає значення оцінок
int[][] getBinaryMatrix	int[][] values – оцінки, які необхідно перетворити у бінарну матрицю List<Double> limitVector – вектор, який задає СКД int delta – значення дельти для СКД	Метод для перетворення оцінок у бінарний вигляд відносно СКД. Повертає бінарну матрицю оцінок
int[][] getBinaryMatrix	int[][] values – оцінки, які необхідно перетворити у бінарну матрицю List<Double> limitVector – вектор, який задає СКД int[] deltaVector – значення дельти для кожної ознаки	Метод для перетворення оцінок у бінарний вигляд відносно СКД. Повертає бінарну матрицю оцінок
List<Double> getLimitVector	int[][] values – оцінки базового класу	Метод для отримання вектора, який задає СКД.

		Повертає вектор, який задає СКД
int[] getVectorFromBinaryMatrix	int[][] binaryMatrix – бінарна матриця класу, для якого необхідно знайти еталонний вектор	Метод для отримання еталонного вектора із бінарної матриці класу. Повертає еталонний вектор класу
double calculateCriterion	double alpha – помилка першого роду	Метод для розрахунку критерію функціональної ефективності. Повертає значення критерію
	double beta – помилка другого роду	
double calculateKullback	double alpha – помилка першого роду	Метод для розрахунку критерію Кульбака. Повертає значення критерію Кульбака
	double beta – помилка другого роду	

Клас CriterionValue зберігає значення критерію (табл. 3.2)

Таблиця 3.2 – Опис складових класу CriterionValue

Змінні	
Змінна	**Призначення**
double d1	Перша достовірність
double alpha	Помилка першого роду
double beta	Помилка другого роду
double criterionValue	Значення критерію
boolean isWorkingArea	Чи належить значення критерію робочій області

Клас Delta зберігає значення дельти СКД (табл. 3.3).

Таблиця 3.3 – Опис складових класу CriterionValue

Змінні	
Змінна	**Призначення**
double delta	Дельта
double criterionValue	Значення критерію
boolean isWorkingArea	Чи належить значення критерію робочій області

Клас TreePair зберігає дані про страту декурсивного дерева (табл. 3.4).

Таблиця 3.4 – Опис складових класу CriterionValue

Змінні	
Змінна	**Призначення**
List<String> classesNames	Список імен класів страти
List<int[][]> classesValues	Список матриць значень класів страти
List<Integer> classesRadii	Список радіусів гіперсферичних контейнерів класів страти
int[] deltaVector	Дельта-вектор СКД страти

3.5 Результати машинного навчання

У процесі машинного навчання було отримано наступні результати:

Страта (X_F^0, X_E^0): $\delta = 3$, $d_F = 78$, $d_E = 55$

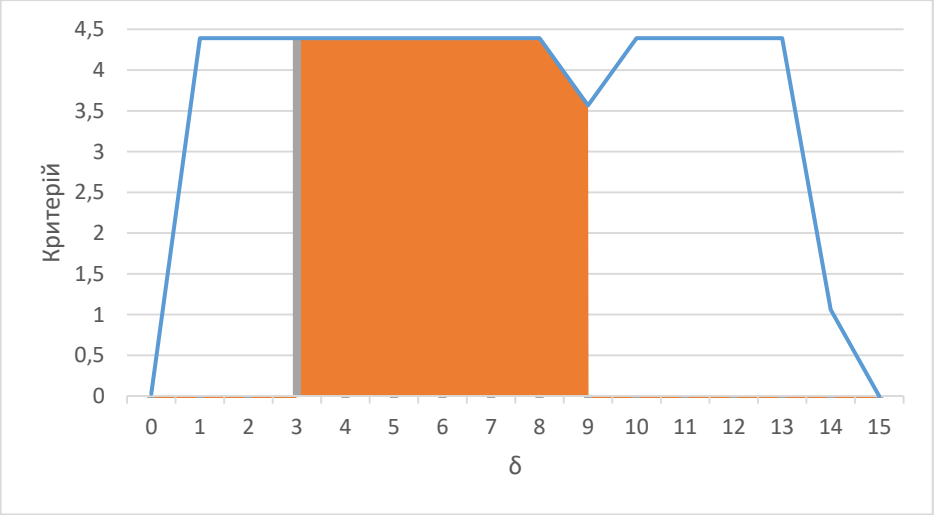

Рисунок 3.5 – Залежність КФЕ від значення δ для страти (X_F^0, X_E^0)

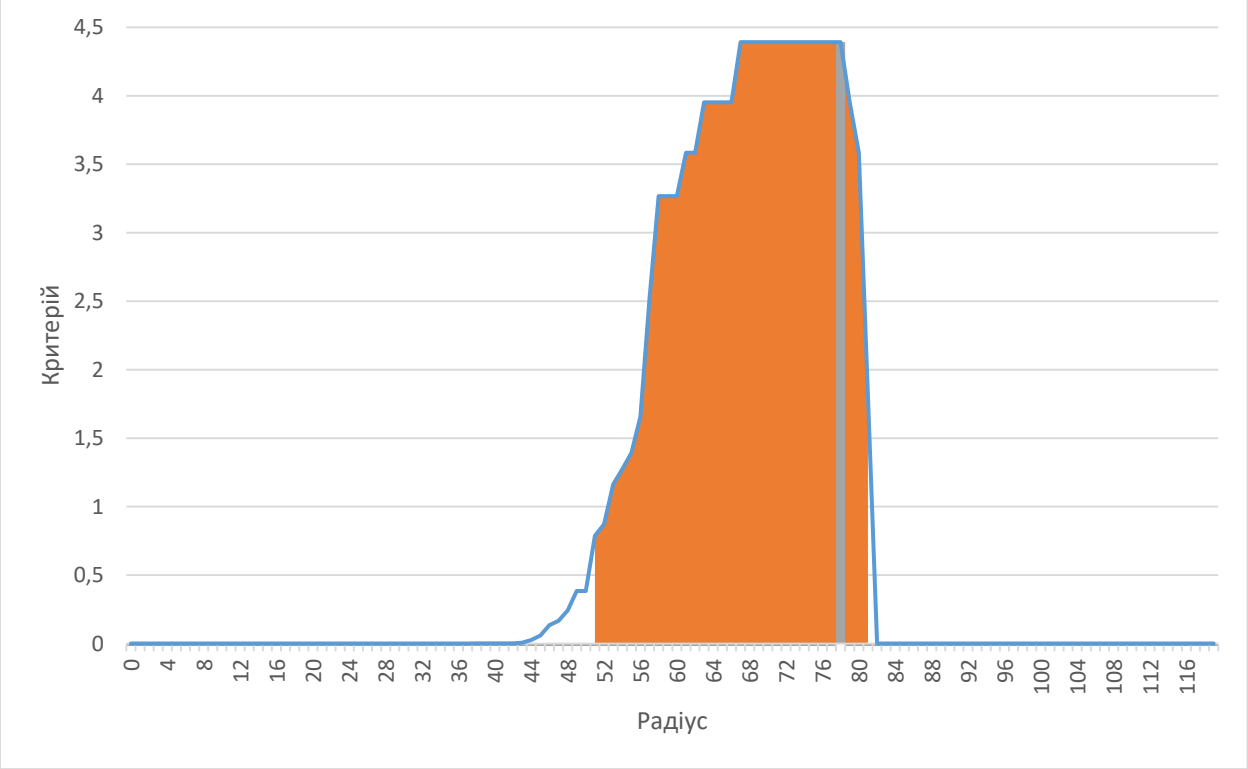

Рисунок 3.6 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_F^0 для страти (X_F^0, X_E^0)

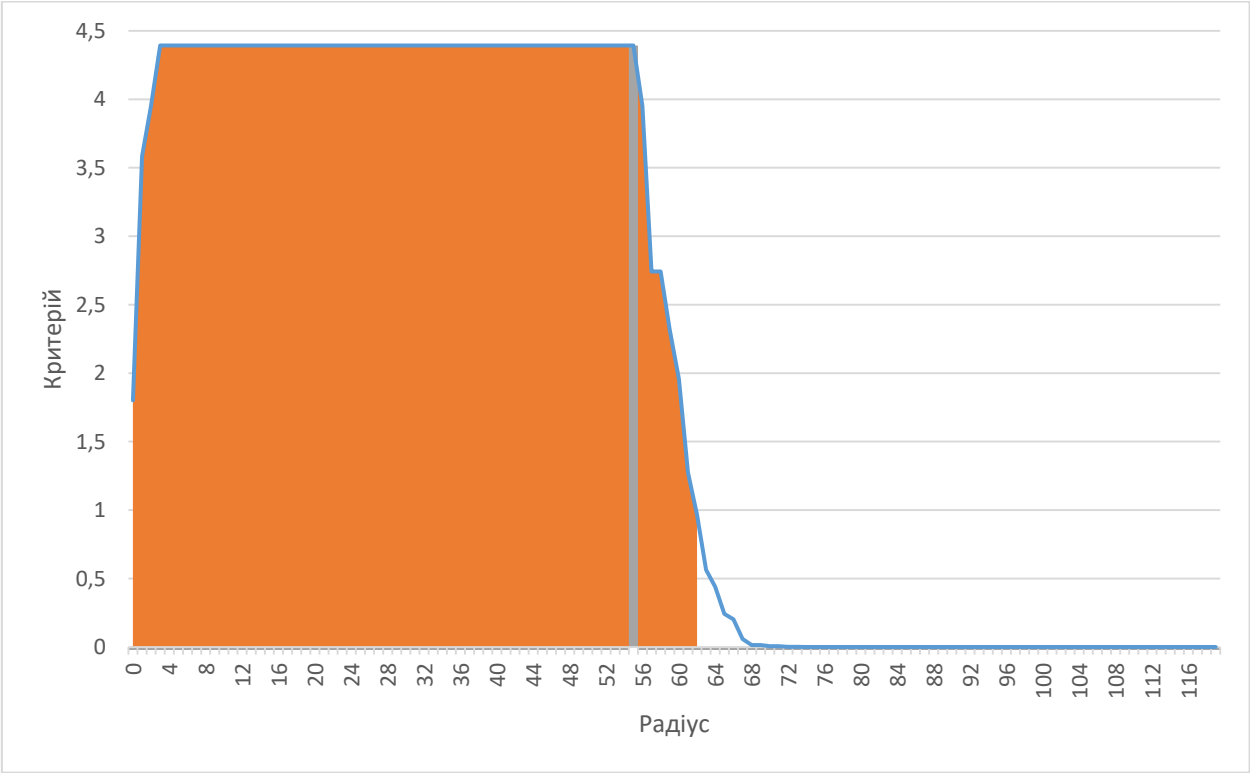

Рисунок 3.7 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_E^0 для страти (X_F^0, X_E^0)

Страта (X_E^0, X_D^0): $\delta = 3$, $d_E = 109$, $d_D = 88$

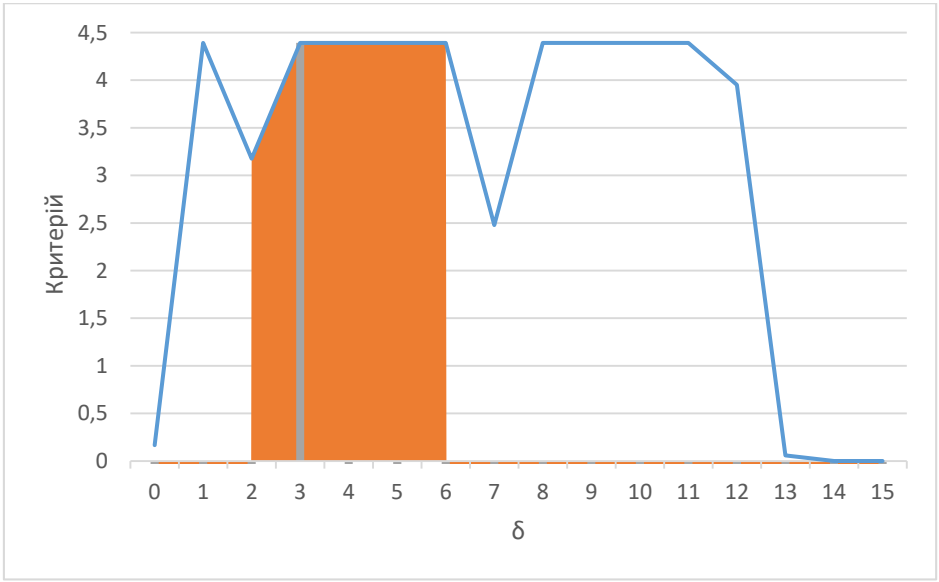

Рисунок 3.8 – Залежність КФЕ від значення δ для страти (X_E^0, X_D^0)

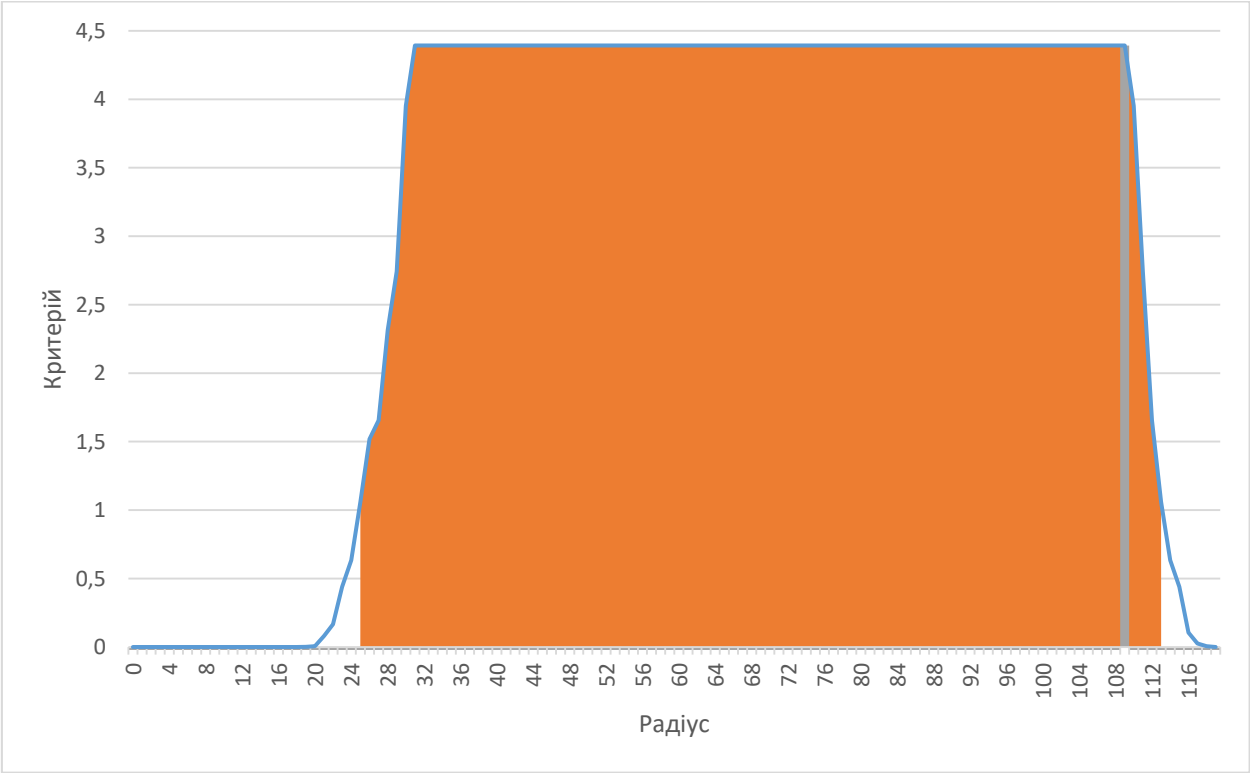

Рисунок 3.9 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_E^0 для страти (X_E^0, X_D^0)

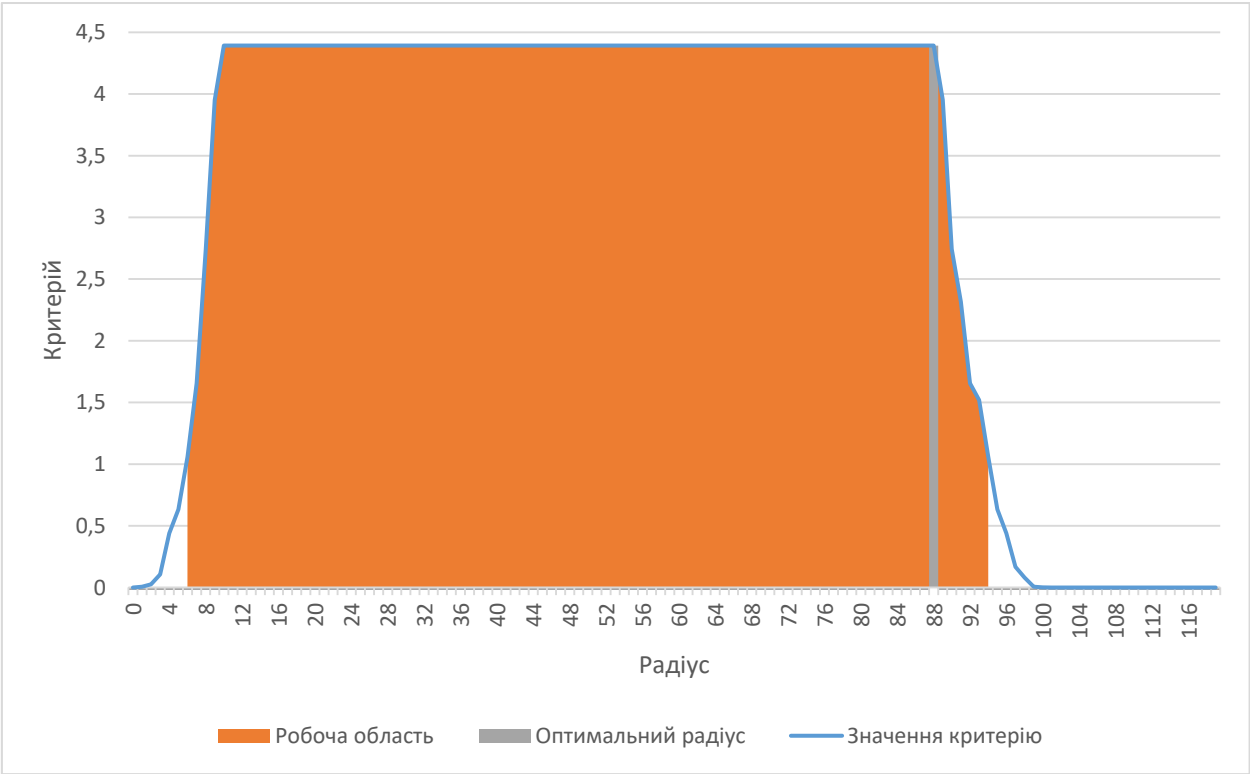

Рисунок 3.10 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_D^0 для страти (X_E^0, X_D^0)

Страта (X_D^0, X_C^0): $\delta = 3$, $d_D = 106$, $d_C = 82$

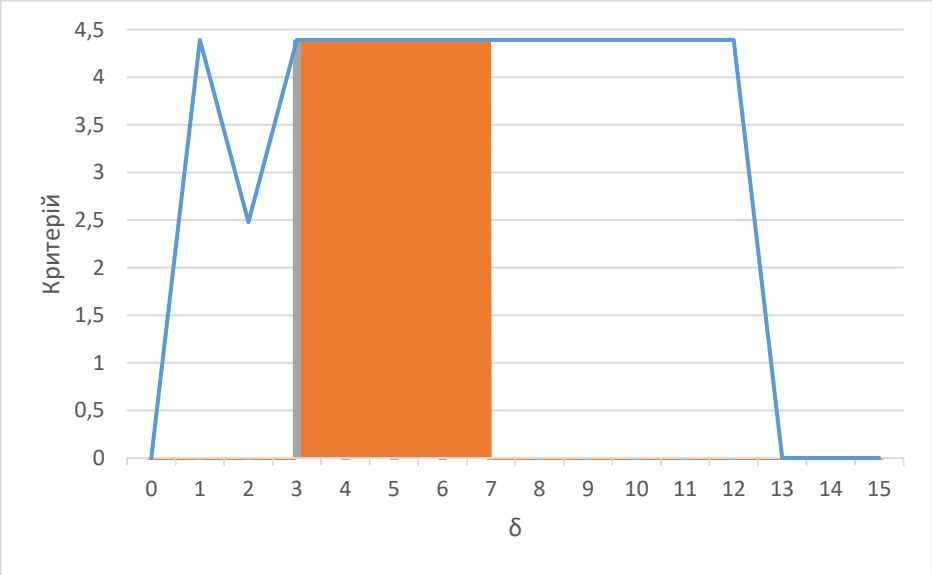

Рисунок 3.11 – Залежність КФЕ від значення δ для страти (X_D^0, X_C^0)

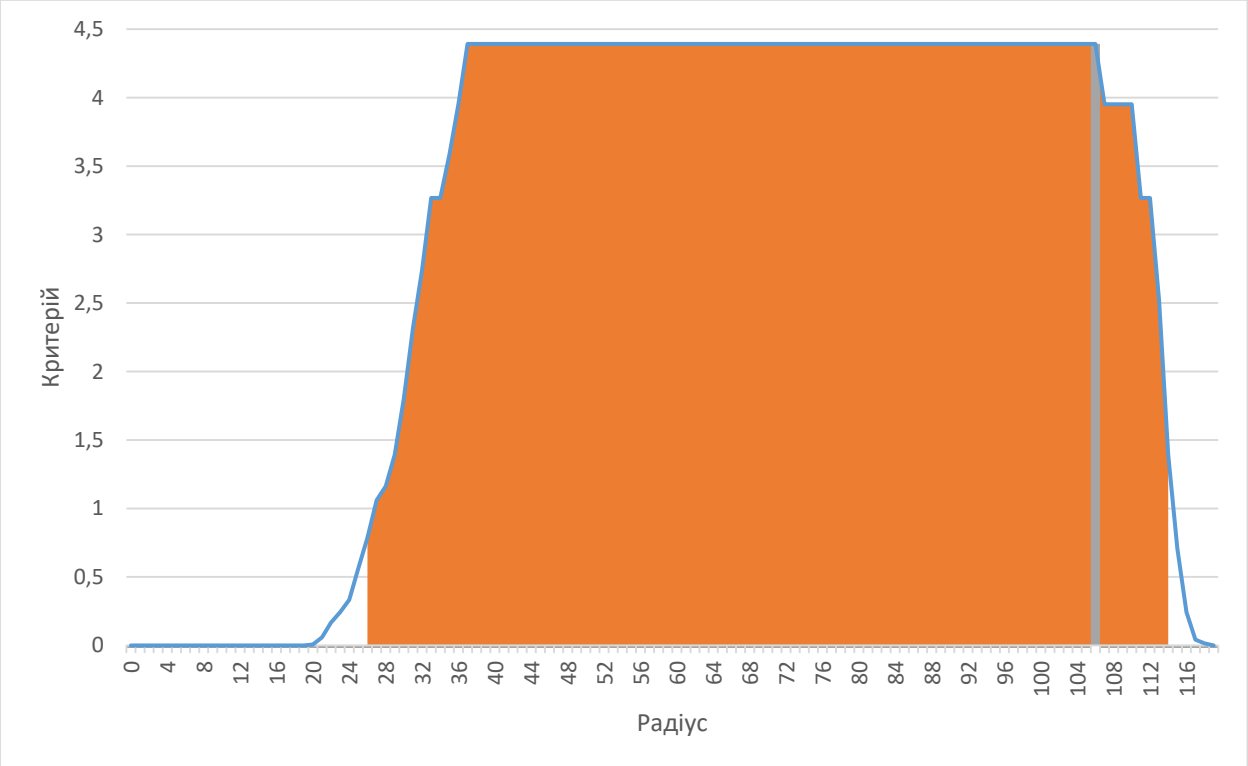

Рисунок 3.12 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_D^0 для страти (X_D^0, X_C^0)

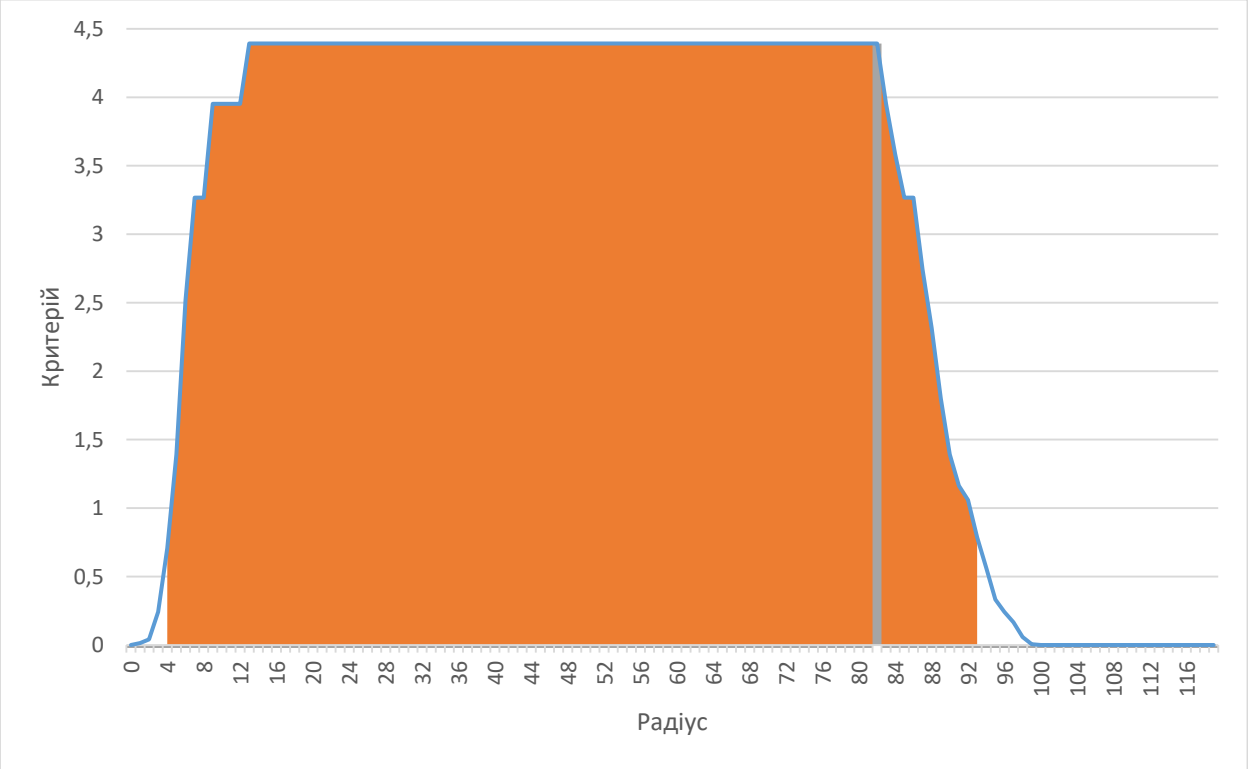

Рисунок 3.13 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_C^0 для страти (X_D^0, X_C^0)

Страта (X_C^0, X_B^0): $\delta = 2$, $d_C = 113$, $d_B = 64$

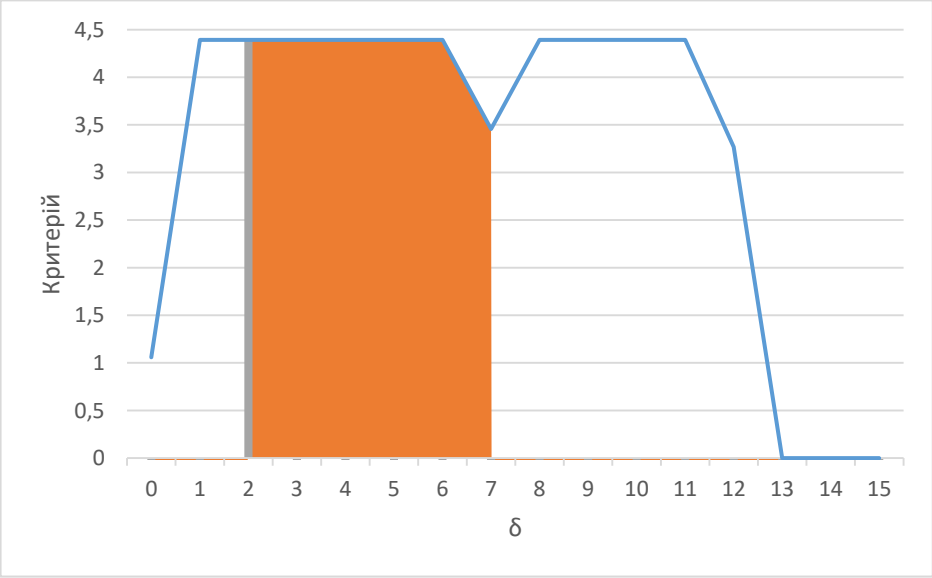

Рисунок 3.14 – Залежність КФЕ від значення δ для страти (X_C^0, X_B^0)

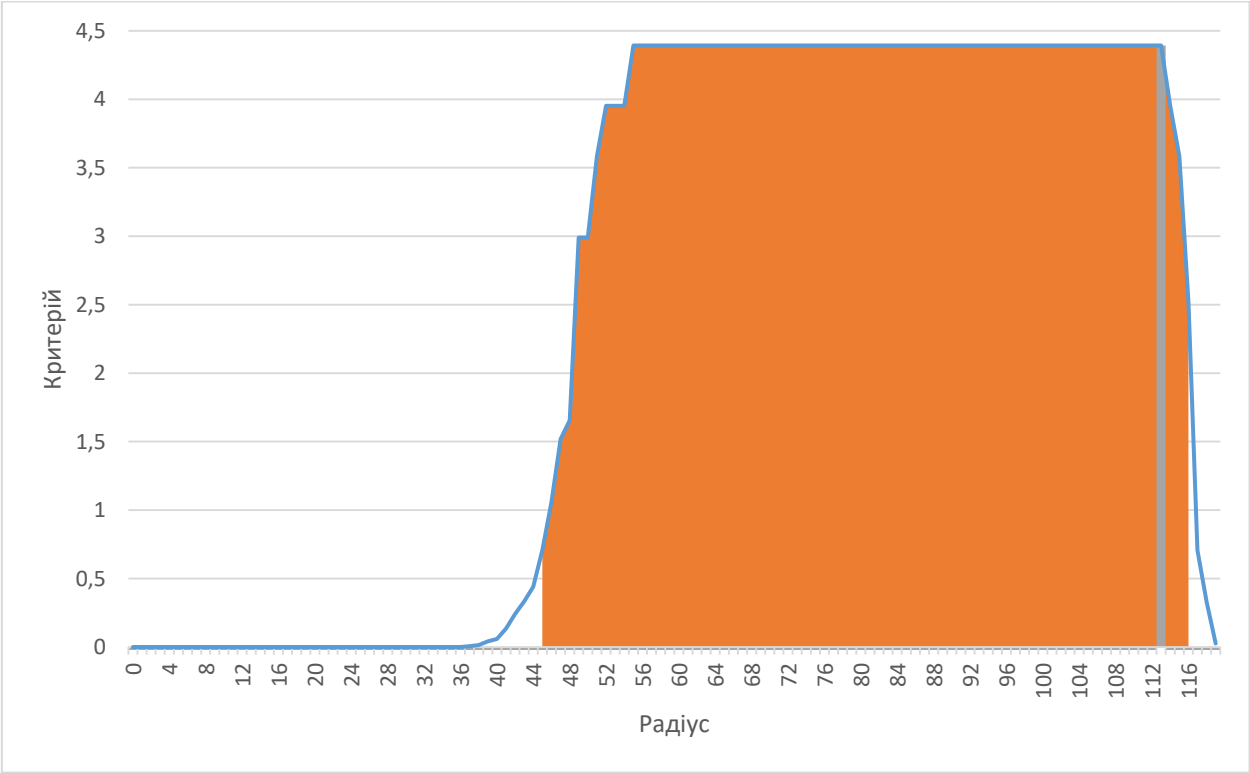

Рисунок 3.15 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_C^0 для страти (X_C^0, X_B^0)

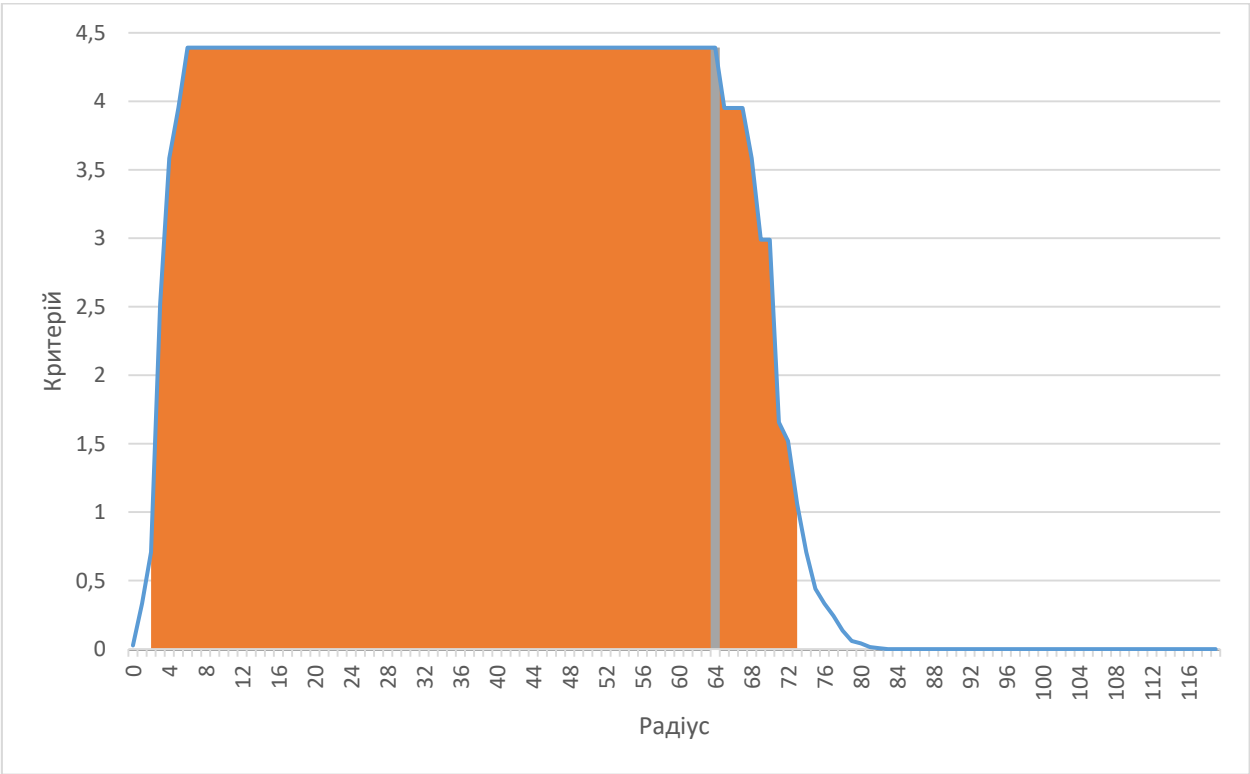

Рисунок 3.16 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_B^0 для страти (X_C^0, X_B^0)

Страта (X_B^0, X_A^0): $\delta = 2$, $d_C = 109$, $d_B = 63$

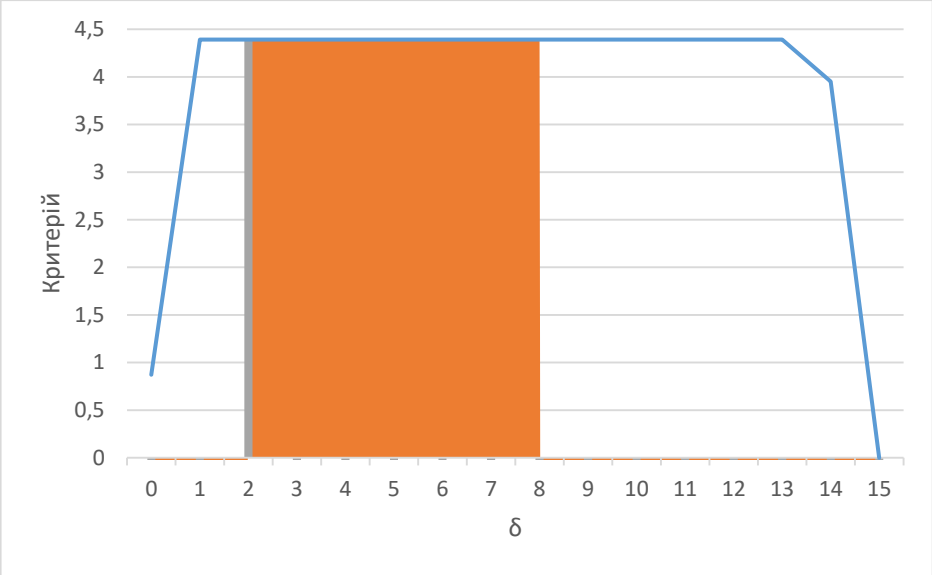

Рисунок 3.17 – Залежність КФЕ від значення δ для страти (X_B^0, X_A^0)

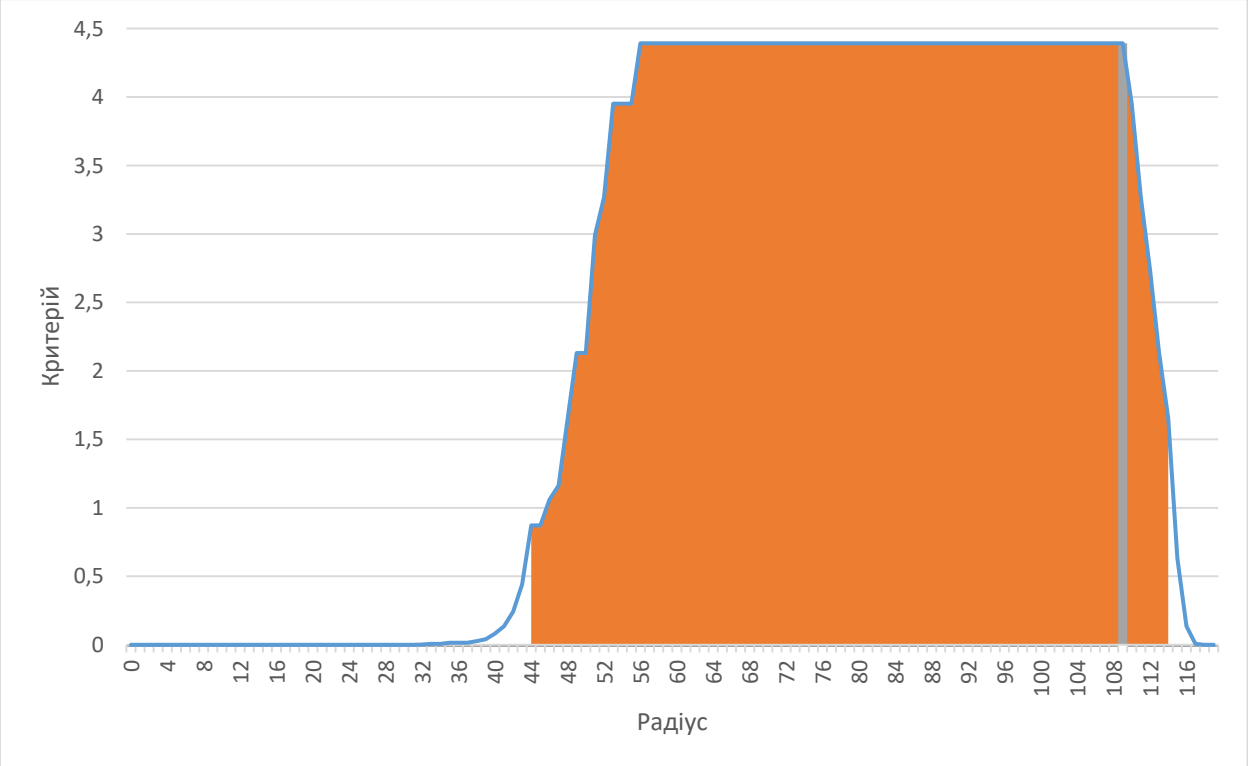

Рисунок 3.18 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_B^0 для страти (X_B^0, X_A^0)

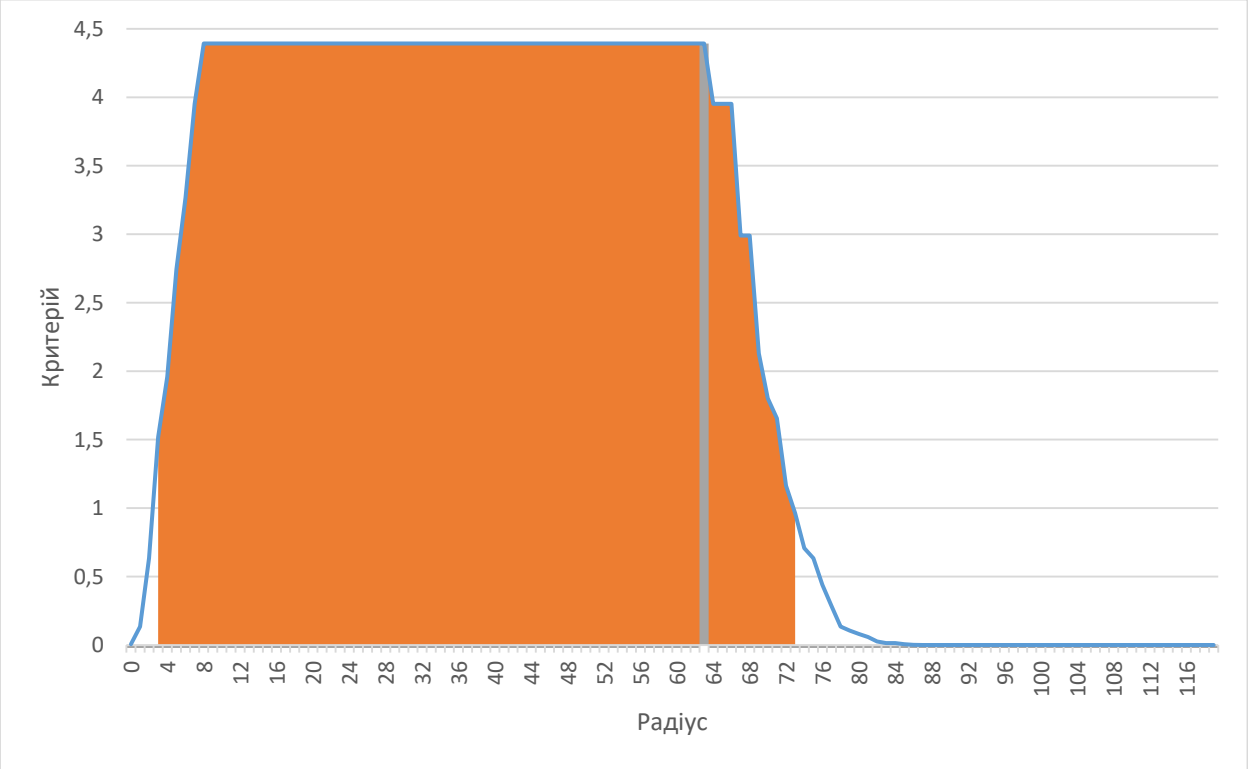

Рисунок 3.19 – Залежність КФЕ від радіусу гіперсферичного контейнера класу X_A^0 для страти (X_B^0, X_A^0)

На рисунку 3.20 показано декурсивне дерево, де для кожної страти вказано отримані за результатами машинного навчання оптимальні параметри функціонування СППР: параметр поля контрольних допусків на ознаки розпізнавання, який дорівнює половині двобічного симетричного поля контрольних допусків, і радіуси гіперсферичних контейнерів класів розпізнавання, які відновлюються в процесі машинного навчання у радіальному базисі простору ознак розпізнавання.

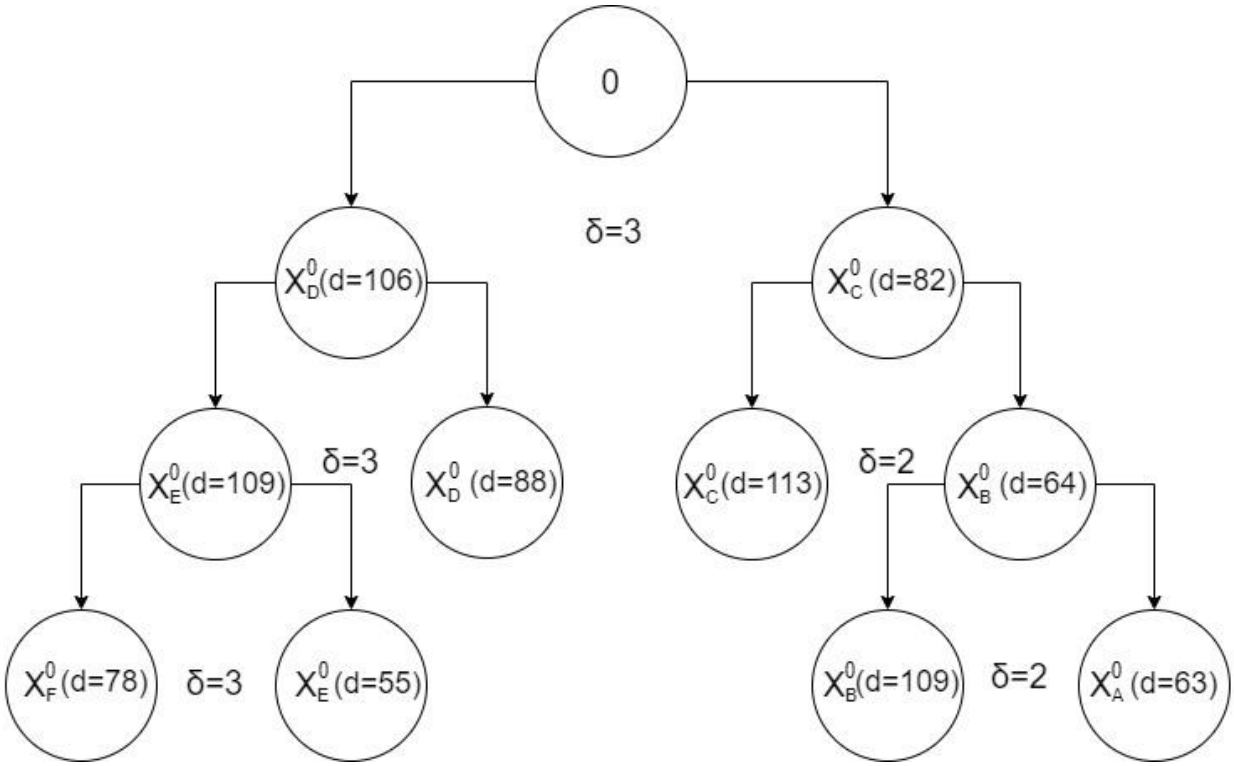

Рисунок 3.20 – Результати машинного навчання

Кожна страта мала свій оптимальний в інформаційному розумінні параметр поля контрольних допусків на ознаки розпізнавання, що забезпечило високу достовірність розпізнавання у порівнянні з лінійною структурою даних. У результаті було побудовано безпомилкові вирішальні правила

ВИСНОВОК

1. У ході написання магістерської кваліфікаційної роботи було проведено аналіз проблеми, описано деталі методу дослідження та сформульовано формалізовану постановку задачі. Було розроблено алгоритм та категорійну модель СППР для адаптації навчального контенту до вимог ринку праці. Розроблений алгоритм було імплементовано мовою Java.

2. З використанням вхідних даних, отриманих імітаційним шляхом, було проведено машинне навчання СППР. За отриманими в процесі машинного навчання оптимальними геометричними параметрами контейнерів класів розпізнавання побудовано безпомилкові вирішальні правила.

3. Отримані правила можуть бути використані як стартові у системі підтримки прийняття рішень для корегування навчального контенту з можливістю перенавчання у процесі роботи.

СПИСОК ЛІТЕРАТУРИ

1. Гнезділова К. М. Шляхи оцінювання якості вищої освіти / К. М. Гнезділова // Вісн. Черкас. ун-ту. Сер. Пед. науки. - 2009. - Вип. 165. - С. 33-37.
2. Єсіна О. Г. Критерії оцінки якості підготовки сучасних фахівців / О. Г. Єсіна // Теорія та методика навчання фундаментальних дисциплін у вищій школі : збірник наукових праць. – Кривий Ріг: НМетАУ, 2012. – Вип. VII. – С. 84-90.
3. Мурашко М. І. Формування моделі оцінювання якості вищої освіти і можливість її реалізації в системі підготовки фахівців [Електронний ресурс] / І. М. Мурашко, С.О. Назарко // Вісник Чернігівського державного технологічного університету. Серія «Економічні науки» : зб. наукових праць. – Чернігів : ЧДТУ, 2010. – №43.
4. Ядов В. А. Социологическое исследование: методология, программа, методы / В. А. Ядов. – Москва: Наука, 1972. – 238 с.
5. Dillman D. Mail and telephone surveys: The total design method / Don A Dillman. – New York: Wiley, 1978. – 325 с.
6. Survey Methodology / [R. Groves, F. Fowler, M. Couper та ін.]. – Hoboken: Wiley, 2009. – 496 с.
7. Волокитіна Л.О. Маркетингова система освітніх послуг вищого навчального закладу: автореф. дис. канд. екон. наук/ Донец. нац. ун-т економіки і торгівлі ім. М.Туган-Барановського. — Донецьк, 2008. — 21 с.
8. Andoni A., Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions // Commun ACM. 2008. Vol. 51, № 1. P. 117–122.
9. Sarma T.H. et al. An improvement to k-nearest neighbor classifier. 2013.
10. Zhu B., Shoaran M. Tree in Tree: from Decision Trees to Decision Graphs. 2021.

11. Лысцов Н.А., Мартышкин А.И. НЕЙРОННЫЕ СЕТИ: ПРИМЕНЕНИЕ И ПЕРСПЕКТИВЫ // Научное обозрение. Педагогические науки. – 2019. – № 3-2. – С. 35-38.
12. Гафаров Ф.М. Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 121 с.
13. Руденко, М.С. Розпізнавання онкологічних захворювань [Текст] / М.С. Руденко, С.С. Мартиненко // Інформатика, математика, механіка (ІММ-2009) : матеріали та програма IV міжвузівської науково-технічної конференції викладачів, співробітників, аспірантів і студентів, 21-24 квітня 2009 р. / Відпов. за вип. С.І. Проценко. - Суми : СумДУ, 2009. - С. 45.
14. Довбиш, А. С. Система підтримки прийняття рішень для визначення схеми лікування гострої кишкової інфекції [Текст] / А.С. Довбиш, Г.А. Стадник, К.С. Полов'ян // Вісник Сумського державного університету. Серія Технічні науки. - 2012. - №1. - С. 25-31.
15. Довбиш А.С. Оптимізація параметрів навчання системи підтримки прийняття рішень для керування виробництвом композитних матеріалів / А.С. Довбиш, В.О. Боровик, Н.І. Андрієнко // Вісник СумДУ. Серія "Технічні науки". - 2012. -№3. - С. 10-15.
16. Проценко, С.І. Інформаційна технологія розпізнавання електронограм на просвічуючому електронному мікроскопі [Текст]: робота на здобуття кваліфікаційного ступеня магістра; спец.: 122 - комп`ютерні науки (інформатика) / С.І. Проценко; наук. керівник А.С. Довбиш. - Суми: СумДУ, 2021. - 76 с.
17. Бортова система безпілотного літального апарату для автономного розпізнавання наземних малогабаритних об'єктів [Текст] : звіт про НДР (проміжний) / кер. А. С. Довбиш. — Суми : СумДУ, 2020. — 96 с.
18. Савченко, Т.Р. Інформаційна технологія розпізнавання об'єктів. Машинне навчання бортової системи розпізнавання наземних об'єктів

[Текст]: робота на здобуття кваліфікаційного ступеня бакалавра; спец.: 122 - комп`ютерні науки (інформатика) / Т.Р. Савченко; наук. кер. А.С. Довбиш. - Суми: СумДУ, 2021. - 67 с.

19. Сучасні інформаційні технології в кібербезпеці [Текст]: монографія / А.С. Довбиш, В.К. Ободяк, І.В. Шелехов та ін.; за ред. В.К. Ободяка, І.В. Шелехова. — Суми: СумДУ, 2021. — 348 с.
20. Теницька, А.О. Система виявлення кібератак. Інформаційна технологія функціонування системи виявлення атак в режимі моніторингу [Текст]: робота на здобуття кваліфікаційного ступеня бакалавра; спец.: 125 - кібербезпека / А.О. Теницька; наук. кер. А.С. Довбиш. - Суми: СумДУ, 2021. - 65 с.
21. Довбиш, А.С. Підхід до класифікаційного прогнозування валютного курсу [Текст] / А.С. Довбиш, А.В. Чала // Інтелектуальні системи в промисловості і освіті : тези доповідей Третьої міжнародної науково-практичної конференції, м. Суми, 2-4 листопада 2011 р. / Ред.кол.: А.С. Довбиш, О.А. Борисенко, С.П. Шаповалов. — Суми : СумДУ, 2011. — Т.2. — С. 26-30.
22. Довбиш, А.С. Оптимізація інвестиційного портфеля за методом функціонально-статистичних випробувань [Текст] / А.С. Довбиш, В.А. Тронь // Вісник Сумського державного університету. Серія Технічні науки. — 2006. — №4(88). — С. 43-53.
23. Довбиш А. С. Основи проектування інтелектуальних систем: навч. посіб. / А. С. Довбиш – Суми: Видавництво СумДУ, 2009. — 171 с.
24. Довбиш, А.С. Оцінка функціональної ефективності навчання автоматизованої системи керування технологічним процесом [Текст] / А.С. Довбиш, О.Б. Берест // Вісник Сумського державного університету. Серія Технічні науки. - 2012. - № 3. - С. 38-45.
25. Оптимізація параметрів навчання комп’ютеризованої системи діагностування емоційно-психічного стану людини / І. В. Шелехов, Д. В.

Прилепа // Радіоелектронні і комп’ютерні системи. - 2014. - № 1. - С. 116–121.

26. Шелехов, І.В. Вибір базового класу при розпізнаванні зображень [Текст] / І.В. Шелехов, К.В. Барило // Вісник Сумського державного університету. Серія Технічні науки. - 2010. - №3, Т.2. - С. 95-102.
27. Оптимізація ієрархічної структури даних інтелектуальної системи функціонального діагностування технічного стану складної машини / А. С. Довбиш, В. І. Зимовець, М. В. Бібик // Вісник Національного технічного університету “ХПІ”. Серія : Системний аналіз, управління та інформаційні технології. - 2018. - № 44. - С. 42-49.
28. Довбиш А. С. Оптимізація словника ознак розпізнавання системи керування, що навчається / А. С. Довбиш, І. В. Шелехов, О. В. Коробченко // Адаптивні системи автоматичного управління : міжвідомчий науково-технічний збірник. – 2015. – № 2(27). – С. 44–50. .
29. Шелехов, І.В. Інформаційно-екстремальна оптимізація словника ознак розпізнавання [Текст] / І.В. Шелехов, М.М. Бірюкова // Вісник Сумського державного університету. Серія Технічні науки. - 2012. - № 3. - С. 46-54.
30. Довбиш А. С. Оптимізація словника ознак розпізнавання для інформаційно-екстремального гіпереліпсоїдного класифікатора / А. С. Довбиш, В. В. Москаленко // Вісник Нац. техн. ун-ту “ХПІ” : зб. наук. пр. Темат. вип. : Системний аналіз, управління та інформаційні технології. – Харків : НТУ “ХПІ”. – 2012. – № 30. – С. 65-78.
31. Довбиш А. С. Інтелектуальні інформаційні технології в електронному навчанні: монографія / А. С. Довбиш, А. В. Васильєв, В. О. Любчак. — Суми: Сумський державний університет, 2013. — 177 с.
32. Гладченко, М.С. Інформаційна система інтелектуального аналізу якості освітнього контенту кафедри [Текст]:робота на здобуття кваліфікаційного ступеня магістра; спец.: 122 - комп`ютерні науки (інформатика) / М.С. Гладченко; наук. керівник І.В. Шелехов - Суми: СумДУ, 2019. - 58 с.

33. Сальник, К.О. Інформаційно-аналітична система адаптації навчального контенту до вимог ринку праці. Функціонування системи в режимі моніторингу [Текст]: робота на здобуття кваліфікаційного ступеня магістра; спец.: 122 - комп’ютерні науки (інформатика) / К.О. Сальник; наук. керівник А.С. Довбиш. - Суми: СумДУ, 2020. - 74 с.

ДОДАТОК

Повний код програмної реалізації:

```
public class IEITCartesianCoordinates {
   /**
    * Метод для машинного навчання з використанням декурсивного дерева
    *
    * @param classesNames список імен класів (шляхів до реалізацій)
    * @throws IOException
    */
   public static void launchHierarchical(List<String> classesNames) throws IOException {
      /* Створюємо декурсивне дерево із заданих класів розпізнавання */
      List<TreePair> treePairs = getTreePairs(classesNames);
      Map<String, List<TreePair>> classesPairs = new HashMap<>();
      Set<TreePair> finalTreePairs = new HashSet<>();
      /* Для кожної страти проводимо машинне навчання з паралельною оптимізацією контрольних допусків */
      for (TreePair treePair : treePairs) {
         System.out.println(treePair.getClassesNames());
         launchParallel(treePair, 0);
         System.out.println(treePair.getClassesRadii());
         for (int i = 0; i < treePair.getClassesNames().size(); i++) {
            classesPairs.computeIfAbsent(treePair.getClassesNames().get(i), k -> new ArrayList<>()).add(treePair);
         }
      }
      /* Визначаємо оптимальні правила */
      classesPairs.forEach((key, value) -> {
```

```
            if (value.size() == 1) {
                finalTreePairs.addAll(value);
            } else {
                TreePair res = value.stream()
                        .min(Comparator.comparingInt(v -> v.getRadiusByClassName(key)))
                        .get();
                finalTreePairs.add(res);
            }
        });
        System.out.println(finalTreePairs);
    }


    /**
     * Метод для побудови декурсивного дерева із заданих класів розпізнавання
     *
     * @param classesNames список імен класів (шляхів до реалізацій)
     * @return декурсивне дерево із заданими класами розпізнавання
     * @throws IOException
     */
    private static List<TreePair> getTreePairs(List<String> classesNames) throws IOException {
        List<TreePair> treePairs = new LinkedList<>();
        for (int i = 0; i < classesNames.size() - 1; i++) {
            treePairs.add(new TreePair(classesNames.get(i), classesNames.get(i + 1), txtToArray(classesNames.get(i)), txtToArray(classesNames.get(i + 1))));
        }

        return treePairs;
```

```
    }

    /**
     * Метод для машинного навчання з паралельною оптимізацією контрольних допусків
     *
     * @param treePair  страта декурсивного дерева
     * @param baseClass базовий клас у страті
     * @throws IOException
     */
    public static void launchParallel(TreePair treePair, int baseClass) throws IOException {
        /* Знаходимо оптимальне значення дельти для СКД */
        int delta = getOptimalDeltaParallel(treePair.getClassesValues(), baseClass);
        int[] deltaVector = new int[treePair.getClassesValues().get(0)[0].length];
        Arrays.fill(deltaVector, delta);
        /* Переводимо значення у бінарний вигляд та знаходимо еталонні вектори кожного класу */
        List<int[][]> classBinaryMatrices = new LinkedList<>();
        int[][] classVectors = new int[treePair.getClassesValues().size()][];
        List<Double> limitVector = getLimitVector(treePair.getClassesValues().get(baseClass));
        for (int i = 0; i < treePair.getClassesValues().size(); i++) {
            int[][] classBinaryMatrix = getBinaryMatrix(treePair.getClassesValues().get(i), limitVector, deltaVector);
            classBinaryMatrices.add(classBinaryMatrix);
            classVectors[i] = getVectorFromBinaryMatrix(classBinaryMatrix);
        }

        /* Знаходимо радіуси контейнера кожного класу */
```

```
        List<Integer> radii = getRadii(classVectors, classBinaryMatrices);

        treePair.setClassesRadii(radii);
        treePair.setDeltaVector(deltaVector);
    }

    /**
     * Метод для отримання оптимального значення дельти для СКД з паралельною оптимізацією контрольних допусків
     *
     * @param classesValues реалізації класів
     * @param baseClass     базовий клас
     * @return оптимальне значення дельти для СКД
     */
    private static int getOptimalDeltaParallel(List<int[][]> classesValues, int baseClass) {
        List<Delta> possibleDelta = new LinkedList<>();
        /* Шукаємо оптимальне значення у інтервалі [1, 15] */
        for (int delta = 0; delta <= 15; delta++) {
            /* Розраховуємо вектор, який задає СКД, бінарні матриці та еталонні вектори кожного класу */
            List<int[][]> classBinaryMatrices = new LinkedList<>();
            int[][] classVectors = new int[classesValues.size()][];
            List<Double> limitVector = getLimitVector(classesValues.get(baseClass));
            for (int i = 0; i < classesValues.size(); i++) {
                int[][] classBinaryMatrix = getBinaryMatrix(classesValues.get(i), limitVector, delta);
                classBinaryMatrices.add(classBinaryMatrix);
                classVectors[i] = getVectorFromBinaryMatrix(classBinaryMatrix);
            }
```

```
        /* Шукаємо сусідів класів */
        int[][] pairs = makePairs(classVectors);
        List<List<CriterionValue>> criterionValues = getCriterionValuesForClassesAndRadii(classVectors, classBinaryMatrices, pairs);
        /* Обчислюємо середнє значення критерію та середнє значення критерію у робочій області */
        List<Double> sum = new LinkedList<>();
        int classesWorkingAreas = 0;
        for (List<CriterionValue> criterionValue : criterionValues) {

sum.add(criterionValue.stream().mapToDouble(CriterionValue::getCriterionValue).max().getAsDouble());
            if (criterionValue.stream().anyMatch(CriterionValue::isWorkingArea)) {
                classesWorkingAreas++;
            }
        }
        boolean isWorkingArea = classesWorkingAreas == classesValues.size();
        double currentValue = sum.stream()
                                .mapToDouble(Double::doubleValue)
                                .average()
                                .getAsDouble();
        possibleDelta.add(new Delta(delta, currentValue, isWorkingArea));
    }
    for (Delta aPossibleDelta : possibleDelta) {
        String a = "-1";
        if (aPossibleDelta.isWorkingArea()) {
            a = String.valueOf(aPossibleDelta.getCriterionValue()).replace('.', ',');
        }
        System.out.println(aPossibleDelta.getDelta() + " " + String.valueOf(aPossibleDelta.getCriterionValue()).replace('.', ',')
```

```
                + " " + a);
        }
        /* Знаходимо максимальне значення дельти у робочій області */
        Optional<Delta> optimalDelta = possibleDelta.stream()
                                        .filter(Delta::isWorkingArea)

.max(Comparator.comparing(Delta::getCriterionValue));
        /* Повертаємо максимальне значення дельти у робочій області. Якщо робоча область відсутня - повертаємо просто максимальне значення */
        if (optimalDelta.isPresent()) {
            return optimalDelta.get().getDelta();
        } else {
            optimalDelta = possibleDelta.stream()
                                .max(Comparator.comparing(Delta::getCriterionValue));
            return optimalDelta.get().getDelta();
        }
    }

    /**
     * Метод для отримання оптимальних радіусів контейнерів кожного класу
     *
     * @param classVectors        еталонні вектори кожного класу
     * @param classBinaryMatrices бінарні матриці класів
     * @return список радіусів контейнерів усіх класів
     */
    private static List<Integer> getRadii(int[][] classVectors, List<int[][]> classBinaryMatrices) {
        /* Знаходимо сусідні класи */
        int[][] pairs = makePairs(classVectors);
        /* Знаходимо значення критерію для можливих значень радіусів */
```

```
List<List<CriterionValue>> criterionValues = getCriterionValuesForClassesAndRadii(classVectors, classBinaryMatrices, pairs);
/* Знаходимо оптимальні радіуси */
List<Integer> radii = new LinkedList<>();
System.out.println("Calculation of radii for classes");
for (int i = 0; i < criterionValues.size(); i++) {
   System.out.println("Class number: " + i);
   List<CriterionValue> res = criterionValues.get(i);
   int index = -1;
   double value = -1;
   /* Проходимо по всім можливим значенням радіуса */
   for (int j = 0; j < res.size(); j++) {
      String a = "-1";
      if (res.get(j).isWorkingArea()) {
         a = String.valueOf(res.get(j).getCriterionValue()).replace('.', ',');
      }
      System.out.println(j + " " + String.valueOf(res.get(j).getCriterionValue()).replace('.', ',')
            + " " + a);
      /* Якщо значення критерію у робочій області для даного радіуса більше за поточне оптимальне, то запам'ятовуємо його та значення радіуса */
      if (res.get(j).isWorkingArea() && res.get(j).getCriterionValue() >= value) {
         value = res.get(j).getCriterionValue();
         index = j;
      }
   }
   radii.add(index);
}
```

```
        return radii;
    }


    /**
     * Метод для обчислення значення критерію для класів та радіусів їх контейнера
     *
     * @param classVectors        еталонні вектори кожного класу
     * @param classBinaryMatrices бінарні матриці класів
     * @param pairs               сусідні класи
     * @return значення критерію для класів та радіусів їх контейнера
     */
    private static List<List<CriterionValue>> getCriterionValuesForClassesAndRadii(int[][] classVectors,
                                                                                    List<int[][]> classBinaryMatrices,
                                                                                    int[][] pairs) {
        List<List<CriterionValue>> criterionValues = new LinkedList<>();
        for (int[] classVector : classVectors) {
            criterionValues.add(new LinkedList<>());
        }
        /* Обчислюємо значення критерію для радіусів у інтервалі [0, довжина реалізації] */
        IntStream.range(0, classVectors.length).forEach(
                classNumber -> IntStream.range(0, classVectors[0].length).forEach(
                        radius -> {
                            /* Перша достовірність */
                            double d1 = getDistancesBetweenVectorAndBinaryMatrix(classVectors[classNumber],
```

```
                        classBinaryMatrices.get(classNumber)).stream()
                                .mapToInt(i -> i <= radius ? 1 : 0)
                                .average()
                                .getAsDouble();
                /* Помилка другого роду */
                double beta = getDistancesBetweenVectorAndBinaryMatrix(classVectors[classNumber],
                        classBinaryMatrices.get(pairs[classNumber][0])).stream()
                                .mapToInt(i -> i <= radius ? 1 : 0)
                                .average().getAsDouble();
                double alpha = 1 - d1;
                /* Обчислюємо значення критерію*/
                double criterionValue = calculateCriterion(alpha, beta);
                /* Якщо перша достовірність >= 0.5, а помилка другого роду < 0.5, то це значення знаходиться у робочій області */
                boolean isWorkingArea = (d1 >= 0.5 && beta < 0.5);
                criterionValues.get(classNumber).add(new CriterionValue(d1, beta, alpha, criterionValue, isWorkingArea));
            }
        )
    );

    return criterionValues;
}

/**
 * Метод для пошуку сусідів кожного класу
 *
 * @param classVectors еталонні вектори кожного класу
```

```
     * @return сусідів кожного класу
     */
    private static int[][] makePairs(int[][] classVectors) {
        int[][] pairs = new int[classVectors.length][2];
        int valueToSet = classVectors[0].length + 1;
        for (int[] pair : pairs) {
            Arrays.fill(pair, valueToSet);
        }
        for (int i = 0; i < classVectors.length; i++) {
            for (int j = 0; j < classVectors.length; j++) {
                if (i != j) {
                    int distance = getDistanceBetweenVectors(classVectors[i],
classVectors[j]);
                    if (distance < pairs[i][1]) {
                        pairs[i][0] = j;
                        pairs[i][1] = distance;
                    }
                }
            }
        }

        return pairs;
    }

    /**
     * Метод для пошуку відстаней між вектором та рядками бінарної матриці
     *
     * @param vector       вектор
     * @param binaryMatrix бінарна матриця
     * @return відстані між вектором та рядками бінарної матриці
```

```
     */
    private static List<Integer> getDistancesBetweenVectorAndBinaryMatrix(int[] vector, int[][] binaryMatrix) {
        List<Integer> distances = new LinkedList<>();
        for (int[] binaryMatrixVector : binaryMatrix) {
            distances.add(getDistanceBetweenVectors(vector, binaryMatrixVector));
        }

        return distances;
    }

    /**
     * Метод для пошуку відстаней між двома векторами
     *
     * @param firstVector  перший вектор
     * @param secondVector другий вектор
     * @return відстань між двома векторами
     */
    private static int getDistanceBetweenVectors(int[] firstVector, int[] secondVector) {
        int distance = 0;
        for (int i = 0; i < firstVector.length; i++) {
            if (firstVector[i] != secondVector[i]) {
                distance++;
            }
        }

        return distance;
    }
```

```
    /**
     * Метод для отримання оцінок з файлу
     *
     * @param classPath шлях до файлу з реалізаціями класу
     * @return значення оцінок
     * @throws IOException
     */
    private static int[][] txtToArray(String classPath) throws IOException {
        try (Stream<String> linesStream = Files.lines(Paths.get(classPath),
Charset.defaultCharset())) {
            List<String> lines = linesStream.collect(Collectors.toList());
            int arrayWidth = lines.get(0).split(" ").length;
            int arrayHeight = (int) lines.size();
            int[][] values = new int[arrayHeight][arrayWidth];
            for (int i = 0; i < lines.size(); i++) {
                String[] row = lines.get(i).split(" ");
                for (int j = 0; j < row.length; j++) {
                    values[i][j] = Integer.parseInt(row[j]);
                }
            }

            return values;
        }
    }


    /**
     * Метод для перетворення оцінок у бінарний вигляд відносно СКД
     *
     * @param values      оцінки, які необхідно перетворити у бінарну матрицю
```

```
     * @param limitVector вектор, який задає СКД
     * @param delta       значення дельти для СКД
     * @return бінарну матрицю оцінок
     */
    private static int[][] getBinaryMatrix(int[][] values, List<Double> limitVector, int delta) {
        int[] deltaVector = new int[limitVector.size()];
        Arrays.fill(deltaVector, delta);
        return getBinaryMatrix(values, limitVector, deltaVector);
    }

    /**
     * Метод для перетворення оцінок у бінарний вигляд відносно СКД
     *
     * @param values      оцінки, які необхідно перетворити у бінарну матрицю
     * @param limitVector вектор, який задає СКД
     * @param deltaVector значення дельти для кожної ознаки
     * @return бінарну матрицю оцінок
     */
    private static int[][] getBinaryMatrix(int[][] values, List<Double> limitVector, int[] deltaVector) {
        int[][] binaryMatrix = new int[values.length][values[0].length];
        for (int i = 0; i < values.length; i++) {
            for (int j = 0; j < values[0].length; j++) {
                if (values[i][j] >= limitVector.get(j) - deltaVector[j] && values[i][j] <= limitVector.get(j) + deltaVector[j]) {
                    binaryMatrix[i][j] = 1;
                } else {
                    binaryMatrix[i][j] = 0;
                }
```

```
        }
    }

    return binaryMatrix;
}

/**
 * Метод для отримання вектора, який задає СКД
 *
 * @param values оцінки базового класу
 * @return вектор, який задає СКД
 */
private static List<Double> getLimitVector(int[][] values) {
    List<Double> limitVector = new LinkedList<>();
    for (int i = 0; i < values[0].length; i++) {
        double sum = 0;
        for (int[] row : values) {
            sum += row[i];
        }
        limitVector.add(sum / values.length);
    }

    return limitVector;
}

/**
 * Метод для отримання еталонного вектора із бінарної матриці класу
 *
```

```
     * @param binaryMatrix бінарна матриця класу, для якого необхідно знайти еталонний вектор
     * @return еталонний вектор класу
     */
    private static int[] getVectorFromBinaryMatrix(int[][] binaryMatrix) {
        int[] vector = new int[binaryMatrix[0].length];
        for (int i = 0; i < binaryMatrix[0].length; i++) {
            int sum = 0;
            for (int[] row : binaryMatrix) {
                sum += row[i];
            }
            vector[i] = (int) (Math.round((double) sum / binaryMatrix.length));
        }

        return vector;
    }

    /**
     * Метод для розрахунку критерію функціональної ефективності
     *
     * @param alpha помилка першого роду
     * @param beta  помилка другого роду
     * @return значення критерію
     */
    private static double calculateCriterion(double alpha, double beta) {
        return calculateKullback(alpha, beta);
    }

    /**
     * Метод для розрахунку критерію Кульбака
```

```
     *
     * @param alpha помилка першого роду
     * @param beta  помилка другого роду
     * @return значення критерію Кульбака
     */
    private static double calculateKullback(double alpha, double beta) {
        return (Math.log((2 - (alpha + beta) + 0.1) / (alpha + beta + 0.1)) / Math.log(2))
* (1 - (alpha + beta));
    }
}


public class CriterionValue {
    private double d1;
    private double alpha;
    private double beta;
    private double criterionValue;
    private boolean isWorkingArea;

    public CriterionValue(double d1, double beta, double alpha, double
criterionValue, boolean isWorkingArea) {
        this.d1 = d1;
        this.beta = beta;
        this.alpha = alpha;
        this.criterionValue = criterionValue;
        this.isWorkingArea = isWorkingArea;
    }

    public double getD1() {
        return d1;
```

```
    }

    public double getAlpha() {
        return alpha;
    }

    public double getBeta() {
        return beta;
    }

    public double getCriterionValue() {
        return criterionValue;
    }


    public boolean isWorkingArea() {
        return isWorkingArea;
    }
}


public class Delta {
    private int delta;
    private double criterionValue;
    private boolean isWorkingArea;

    public Delta(int delta, double criterionValue, boolean isWorkingArea) {
        this.delta = delta;
        this.criterionValue = criterionValue;
        this.isWorkingArea = isWorkingArea;
```

```
    }

    public int getDelta() {
        return delta;
    }

    public void setDelta(int delta) {
        this.delta = delta;
    }

    public double getCriterionValue() {
        return criterionValue;
    }

    public void setCriterionValue(double criterionValue) {
        this.criterionValue = criterionValue;
    }

    public boolean isWorkingArea() {
        return isWorkingArea;
    }

    public void setWorkingArea(boolean workingArea) {
        isWorkingArea = workingArea;
    }
}


public class TreePair {
    private List<String> classesNames;
```

```
    private List<int[][]> classesValues;
    private List<Integer> classesRadii;
    private int[] deltaVector;

    public TreePair(String firstClassName, String secondClassName, int[][]
firstClassValues, int[][] secondClassValues) {
        this.classesNames = new LinkedList<>();
        this.classesValues = new LinkedList<>();
        this.classesNames.add(firstClassName);
        this.classesNames.add(secondClassName);
        this.classesValues.add(firstClassValues);
        this.classesValues.add(secondClassValues);
    }

    public List<String> getClassesNames() {
        return classesNames;
    }

    public List<int[][]> getClassesValues() {
        return classesValues;
    }

    public List<Integer> getClassesRadii() {
        return classesRadii;
    }

    public void setClassesRadii(List<Integer> classesRadii) {
        this.classesRadii = classesRadii;
    }
```

```
    public int[] getDeltaVector() {
        return deltaVector;
    }

    public void setDeltaVector(int[] deltaVector) {
        this.deltaVector = deltaVector;
    }

    public int getRadiusByClassName(String className) {
        return classesRadii.get(classesNames.indexOf(className));
    }

    @Override
    public String toString() {
        return "TreePair{" +
                "classesNames=" + classesNames +
                ",\n classesRadii=" + classesRadii +
                ",\n delta=" + deltaVector[0] +
                "}\n";
    }
}
```