МАГІСТЕРСЬКА РОБОТА

15.04 – МР. 1859 “С” 2021.11.01.010 ПЗ

Леуса Михайла Валентиновича

2022 р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.78:004.056.53

ПОГОДЖЕНО

Декан факультету
Інформаційних технологій
/ Глазунова О.Г к.пед.н /
підпис ПІБ, вчене звання і ступінь

«___»_____2022р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

в.о. Завідувач кафедри
Комп'ютерних систем і мереж
/ Касаткін Д.Ю., к.пед.н., доц /
підпис ПІБ, вчене звання і ступінь

«___»_____2022 р.

МАГІСТЕРСЬКА РОБОТА

На тему: «Методи та засоби класифікації атрибутів якості комп'ютерних систем»

Спеціальність 123 «Комп'ютерна інженерія»

Освітня програма Комп'ютерні системи та мережі

Орієнтація освітньої програми_____

Виконав:_____/Леус М.В. /
підпис ПІБ

Керівник дипломного проекту:_____/ Місюра М.Д. /
підпис ПІБ

КИЇВ-2022

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Затвердив
завідувач кафедри
комп'ютерних систем і мереж
_____/ Касаткін Д. Ю.., к.п.н., доц /
підпис ПІБ, вчене звання і ступінь
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Леусу Михайлу Валентиновичу
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): комп'ютерна інженерія

Освітня програма: комп'ютерні системи та мережі

Орієнтація освітньої програми:_____

Тема магістерської роботи: «Методи та засоби класифікації атрибутів якості комп'ютерних систем»

затверджена наказом ректора НУБіП України від " 1 " листопада 2021 р. № 1859 «С»

Термін подання завершеної роботи на кафедру: 16 листопада 2022 р.

Вихідні дані до магістерської роботи: три додатки з графічним інтерфейсом, що реалізують алгоритм k-найближчих сусідів, метод опорних векторів та наївний байєсівський класифікатор, метрики оцінювання ефективності алгоритмів класифікації

Перелік питань, що підлягають дослідженню:

1. Аналітичний огляд
2. Проектування додатків
3. Реалізація додатків, та їх тестування

Перелік графічного матеріалу (за потреби): схематичні структури додатків які реалізують алгоритм k-найближчих сусідів, метод векторів та наївний байєсівський класифікатор.

Дата видачі завдання "___1___"_____ листопада____**2021** р.

Керівник магістерської роботи _____ Місюра М.Д. к.т.н., доцент.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Леус М.В.
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка задачі магістерської роботи	05.11.2021	Виконано
2	Аналіз предметної області	25.11.2021	Виконано
3	Проектування системи	06.12.2021	Виконано
4	Реалізація системи	03.03.2022	Виконано
5	Тестування системи	10.06.2022	Виконано
6	Оформлення пояснювальної записки	10.09.2022	Виконано
7	Оформлення графічного матеріалу	01.11.2022	Виконано

Студент _____ Леус М.В.
підпис ПІБ

Керівник проекту (роботи) _____ Місюра М.Д.
підпис ПІБ

РЕФЕРАТ

Пояснювальна записка: 87 сторінки, 16 рисунків, 8 таблиць, 2 додатка, 15 Джерел.

МЕТОД, ЗАСІБ, КЛАСИФІКАЦІЯ, АТРИБУТ, КОМП'ЮТЕРНА СИСТЕМА, ЯКІСТЬ.

Об'єкт дослідження – процес застосування алгоритмів класифікації.

Мета дослідження. Метою даної магістерської роботи є виявлення кращого з декількох найвідоміших алгоритму класифікації за допомогою практичного дослідження та порівняння.

Предмет дослідження – програмні засоби реалізації алгоритмів класифікації.

Перший розділ присвячено аналізу предметної області. Проведено детальний огляд об'єкта.

У другому розділі розділі описано процес проектування програмних додатків до трьох досліджуваних алгоритмів класифікації, звернено увагу на особливості архітектури, які потрібно реалізувати максимально ефективно. Створено схематичні структури роботи програм у випадках реалізації.

У третьому розділі повністю описано обґрунтування мови розробки, протестовано розроблені алгоритми, підраховано їх метрики F-міри та визначено найточніший алгоритм.

Четвертий розділ присвячено розрахунку економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором.

В результаті виконання магістерської роботи перевірено актуальність обраної теми, проаналізовано предметну область системи, об'єкт проектування, перелічено існуючі системи-аналоги, їх функціонал та призначення.

ЗМІСТ

ВСТУП

Актуальність теми. Класифікація — це процес розпізнавання, розуміння та групування ідей і об’єктів у попередньо встановлені категорії або «підгрупи». Використовуючи попередньо категоризовані навчальні набори даних, програми машинного навчання використовують різноманітні алгоритми для класифікації майбутніх наборів даних за категоріями.

Алгоритми класифікації в машинному навчанні використовують вхідні навчальні дані, щоб передбачити ймовірність того, що наступні дані потраплять до однієї із заздалегідь визначених категорій. Одним із найпоширеніших застосувань класифікації є фільтрація електронних листів на «спам» і «не спам».

Отже, класифікація — це форма «розпізнавання шаблонів» з алгоритмами класифікації, застосованими до навчальних даних, щоб знайти той самий шаблон (схожі слова чи почуття, послідовності чисел тощо) у майбутніх наборах даних.

Мета дослідження. Метою даної магістерської роботи є виявлення кращого з декількох найвідоміших алгоритму класифікації за допомогою практичного дослідження та порівняння.

Об’єкт дослідження – процес застосування алгоритмів класифікації.

Предмет дослідження – програмні засоби реалізації алгоритмів класифікації.

Методи дослідження. Для досягнення мети дослідження застосовувалися методи алгоритмізації, ООП, машинного навчання, інженерії знань.

Наукова новизна одержаних результатів полягає в наступному:

Проведено порівняння характеристик декількох найвідоміших алгоритмів класифікації, на основі чого визначено який з алгоритмів буде доцільніше застосувати у тому чи іншому випадках.

Наукова новизна. Розроблено алгоритми класифікації із зручним для користувача інтерфейсом.

Практичне значення одержаних результатів полягає у розробленому на основі проведених досліджень програмному забезпеченні, що реалізує можливість тестування алгоритмів класифікації на практиці.

1 ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області алгоритмів класифікації

У штучному інтелекті та машинному навчанні – завдання поділу безлічі спостережень (об'єктів) на групи, які називають класами, на основі аналізу їх формального опису. При класифікації кожна одиниця спостереження відноситься до певної групи або номінальної категорії на основі деякої якісної властивості.

Нехай X — множина описів об’єктів, Y — кінцева множина номерів (імен, міток) класів. Існує невідома цільова залежність — відображення y*:X→Y, значення якої відомі лише на об’єктах кінцевої навчальної вибірки Xm=(x1,y1),…,(xm,ym). Необхідно побудувати алгоритм a:X→Y, здатний класифікувати довільний об’єкт x∈X.

У математичній статистиці завдання класифікації називаються задачами дискримінантного аналізу.

У машинному навчанні завдання класифікації вирішується з використанням навчання з учителем, оскільки класи визначаються заздалегідь і для прикладів навчальної множини мітки класів задані. Аналітичні моделі, що вирішують завдання класифікації, називають класифікаторами. [1]

Завдання класифікації є одним із базових завдань прикладної статистики та машинного навчання, а також штучного інтелекту в цілому. Це пов'язано з тим, що класифікація є однією з найзрозуміліших і найпростіших для інтерпретації технологій аналізу даних, а класифікуючі правила можуть бути сформульовані природною мовою.

До поширених методів розв'язання задачі класифікації відносяться:

- нейронні мережі;
- логістична та пробіт-регресія;
- дерева рішень;
- метод найближчого сусіда;
- машини опорних векторів;
- дискримінантний аналіз.

Завдання класифікації застосовується у багатьох областях:

- у торгівлі — класифікація клієнтів та товарів дозволяє оптимізувати маркетингові стратегії, стимулювати продажі, скорочувати витрати;
- у сфері телекомунікацій — класифікація абонентів дозволяє визначати рівень лояльності, розробляти програми лояльності;
- у медицині та охороні здоров'я — діагностика захворювань, класифікація населення за групами ризику;
- у банківській сфері — кредитний скоринг.

Опишемо найбільш відомі алгоритми класифікації та визначимо їх ключові особливості.

Кластеризація (або кластерний аналіз) — завдання розбиття безлічі об'єктів на групи, які називаються кластерами. Усередині кожного кластера мають бути схожі об'єкти, а об'єкти різних груп повинні бути якомога відміннішими.

Цілі кластеризації:

- Розуміння структури даних. Розбиття безлічі об'єктів на групи дозволяє спростити їх подальшу обробку, наприклад, знаючи структуру даних, можна застосовувати певний конкретний алгоритм для кожного кластера.
- Виявлення новизни. Можна знайти об'єкт, який неможливо віднести до жодної групи.

Відмінністю кластеризації від класичної класифікації є те, що перелік груп чітко не заданий та визначається у процесі роботи алгоритму.

Метод k-найближчих сусідів (англ. k-nearest neighbors algorithm) – метричний алгоритм, призначений для класифікації.

Метод найближчого сусіда

Для визначення класу нового об'єкта необхідно знайти найближчу до нього точку. Результатом роботи алгоритму буде клас цієї точки.

Метод k-найближчих сусідів

Для підвищення точності класифікації зазвичай збільшують число сусідів, якими визначають клас. Новий документ слід віднести до того класу, до якого належить більшість його сусідів. Якщо розглядати завдання бінарної класифікації,

то число сусідів беруть непарним, щоб не виникало неоднозначності, коли однакова кількість сусідів належить різним класам.

Метод виважених найближчих сусідів

У задачах із числом класів більше двох, непарність вже не допомагає і все одно може виникнути ситуація неоднозначності. Для вирішення цієї проблеми і − сусіду приписується вага wi. Об'єкт слід віднести до класу, який набирає більшої сумарної ваги серед k-найближчих сусідів. [1]

Метод опорних векторів (англ. Support Vector Machine, SVM) − лінійний класифікатор, заснований на поділі безлічі векторів з n-мірного простору гіперплощиною. Цей метод потребує навчання перед класифікацією.

Наївний байєсовський класифікатор − імовірнісний класифікатор, заснований на теоремі Байєса. Він називається наївним, бо вважає, що наявність однієї з ознак ніяк не залежить від наявності іншої. З першого погляду, таке припущення може здатися надто сильним, але на практиці цей алгоритм показує досить добрі результати.

1.2 Огляд систем аналогів

Необхідно оглянути кращі комп'ютерні реалізації на прикладі алгоритму k-найближчих сусідів задля розуміння потреб користувача при впровадженні автоматизованих версій таких алгоритмів. Розглянемо реалізацію на сайті https://axd.semestr. Дана реалізація вирізняється великою кількістю пояснень і підказок для необізнаних користувачів, зручний помітний інтерфейс, кнопки та поля для вводу. Розрахунки детальні та надзвичайно чіткі. Скріншоти даної програми та процесу розрахунку алгоритму k-найближчих сусідів можна побачити на рисунках 1.1-1.3.

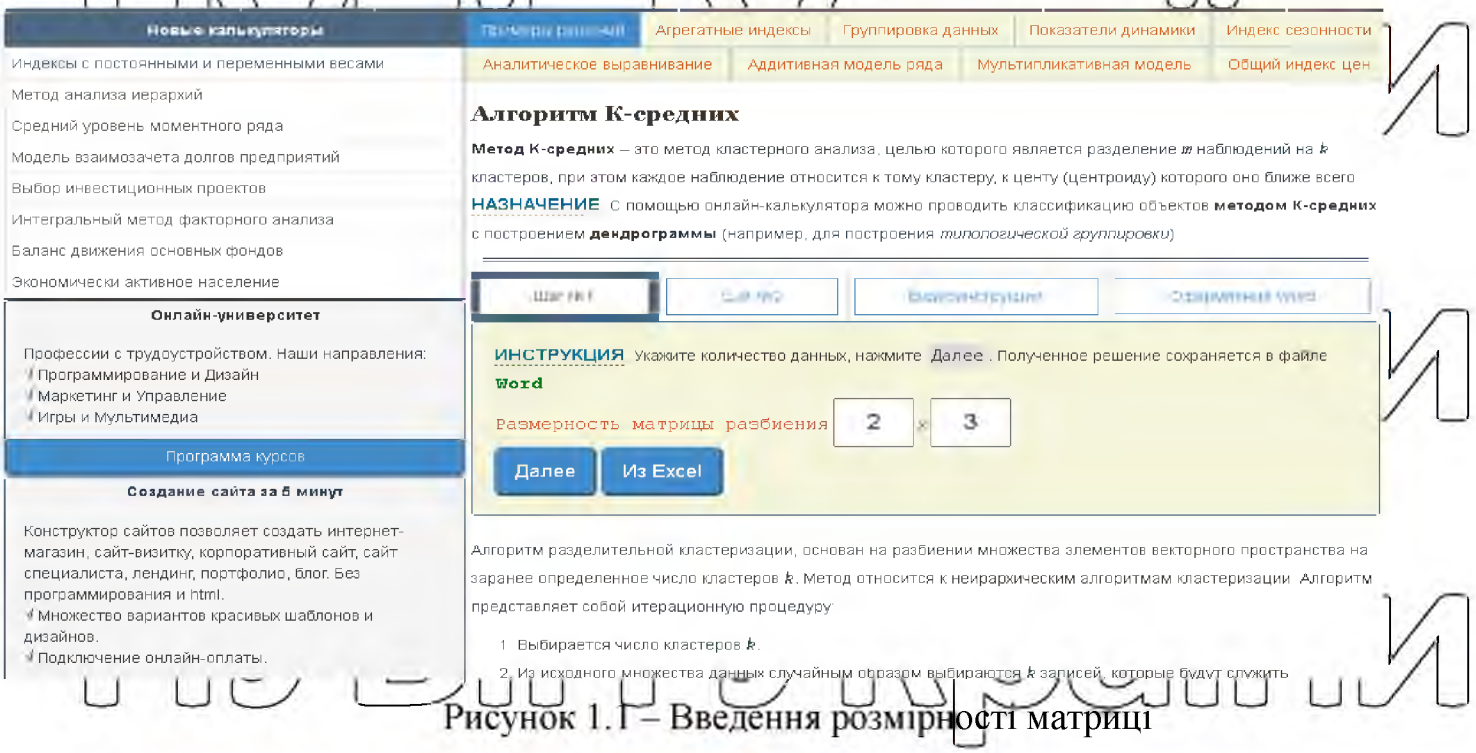

Рисунок 1.1 – Введення розмірності матриці

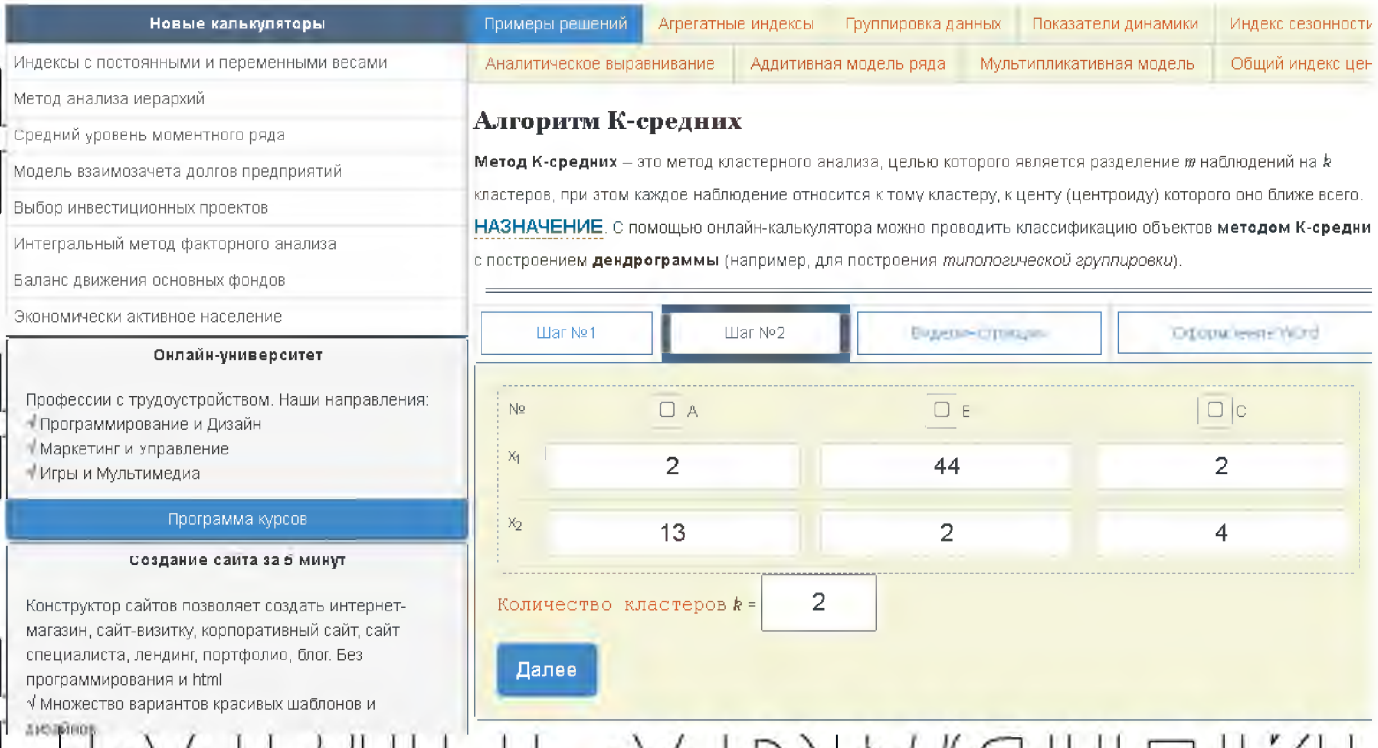

Рисунок 1.2 – Введення матричних даних та встановлення кількості кластерів

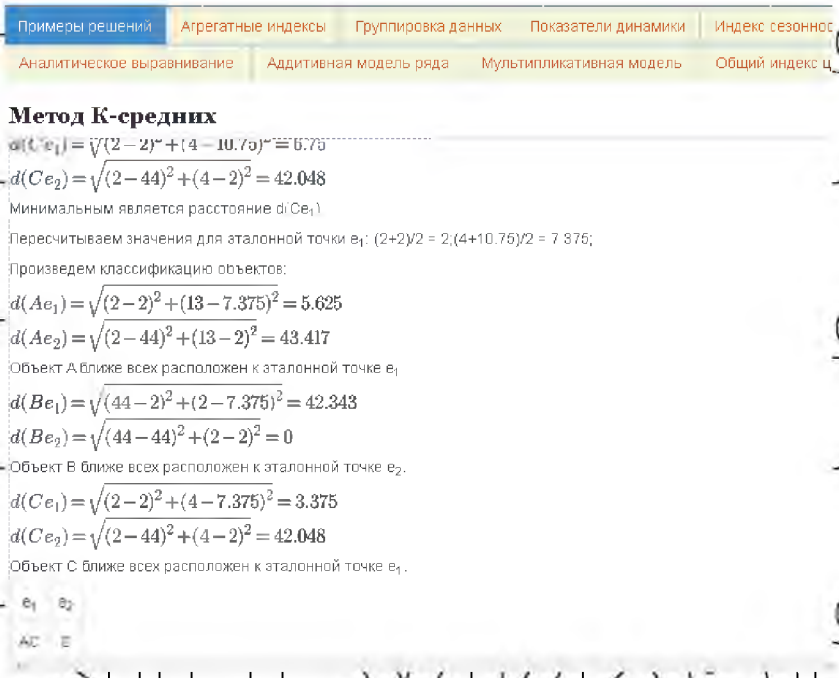

Рисунок 1.3 – Обчислення результатів кластеризації

1.3 Оцінювання результатів алгоритмів класифікації

Важливо описати базові методики та метрики при оцінюванні ефективності у вирішенні задач класифікації. Найвідомішою є методика true/false.

Таким чином, є 4 різновидності ситуації при застосуванні даної методики:

-TP - true positive: класифікатор вірно відніс об'єкт до класу, що розглядається.

-TN - true negative: класифікатор вірно стверджує, що об'єкт не належить до класу, що розглядається.

-FP - false positive: класифікатор невірно відніс об'єкт до класу, що розглядається.

-FN - false negative: класифікатор невірно стверджує, що об'єкт не належить до класу, що розглядається.

TP, TN, FP, FN та поняття, через які виражаються, пояснені в рамках одного класу бінарної класифікації. Тобто в такій системі мається на увазі, що реальна кількість об'єктів класу 0 (для бінарного випадку 0/1) може виражатися як $TP_0+FN_0=FP_1+TN_1$.

Також необхідно виділити ряд оцінок, що базуються на використанні даної метрики.

Accuracy – (точність) показує частку правильних класифікацій. Незважаючи на очевидність і простоту, є однією з малоінформативних оцінок класифікаторів.

Recall – (повнота, sensitivity, TPR (true positive rate)) показує відношення правильно класифікованих об'єктів класу до загального числа елементів цього класу.

Precision – (точність, переклад збігається з accuracy) показує частку правильно класифікованих об'єктів серед усіх об'єктів, які до цього класу відніс класифікатор.

Specificity – Вказує відношення правильних спрацьовувань класифікатора до загальної кількості об'єктів за межами класу. Інакше кажучи, наскільки часто класифікатор правильно не відносить об'єкти до класу.

Fall-out – (FPR (false positive rate)) показує частку неправильних спрацьовувань класифікатора до загальної кількості об'єктів за межами класу. Інакше кажучи, наскільки часто класифікатор помиляється при віднесенні того чи іншого об'єкта до класу. [2]

Для наочної оцінки якості алгоритму застосовується ROC-крива. Крива будується на площині, визначеній TPR (осі ординат) і FPR (осі абсцис).

Для побудови графіка використовується м'яка класифікація: замість чітко віднести об'єкт до класу, класифікатор повертає ймовірності приналежності об'єкта до різних класів. Ця впевненість порівнюється з порогом (який впевненості «достатньо», щоб зарахувати об'єкт до позитивного класу). Залежно від цього порога змінюються значення TPR і FPR.

Алгоритм побудови кривої:

- Запустити класифікатор на тестовій вибірці;
- Відсортувати результати за впевненістю класифікатора у приналежності об'єкта до класу.

Поки що не скінчилися елементи:

- Взяти об'єкт із максимальною впевненістю;
- Порівняти мітку з реальною;
- Перерахувати TPR та FPR на взятих об'єктах;
- Поставити точку, якщо обидві характеристики не NaN / $\pm\infty$;

- Побудувати криву за точками.

Таким чином: число точок вбирається у число об'єктів ідеальному алгоритму відповідає ROC-крива, що проходить через точку (0;1) гіршому алгоритму (наприклад, монеті) відповідає пряма TPR = FPR.

Для чисельної оцінки алгоритму ROC-кривої використовується значення площі під нею (AUC, area under curve). Ідеальний алгоритм має AUC, що дорівнює 1, найгірший — 0,5.

З іншого боку, для побудови ROC-кривої не обов'язково перераховувати TPR та FPR.

2 ПРОЕКТУВАННЯ АЛГОРИТМІВ КЛАСИФІКАЦІЇ

2.1 Проектування програмного додатку алгоритму k-найближчих сусідів

Необхідно окреслити вимоги для проектування даного алгоритму класифікації. До них належать, в першу чергу, наочність та графічний інтерфейс користувача, а також можливості для побудови графіку, що ілюструє кластерний розподіл. Виходячи з описаних особливостей, було вирішено обрати мову програмування C# в комбінації з фреймворком Windows Forms.

Windows Forms — інтерфейс програмування програм (API), який відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Цей інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код класи, що реалізують API для Windows Forms, не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні на C#, C++, так і на VB.Net, J# та ін.

З одного боку, Windows Forms розглядається як заміна старішій і складнішій бібліотеці MFC, спочатку написаної мовою C++. З іншого боку, WF не пропонує парадигми, порівнянної з MVC. Для виправлення цієї ситуації та реалізації цієї функціональності у WF існують сторонні бібліотеки. Однією з найбільш використовуваних подібних бібліотек є User Interface Process Application Block, випущена спеціальною групою Microsoft, яка займається прикладами та рекомендаціями для безкоштовного скачування. Ця бібліотека також містить вихідний код та навчальні приклади для прискорення навчання. [3]

Усередині .NET Framework Windows Forms реалізується у межах простору імен System.Windows.Forms.

Як і Abstract Window Toolkit (AWT) (подібний до API для мови Java), бібліотека Windows Forms була розроблена як частина .NET Framework для спрощення розробки компонентів графічного інтерфейсу користувача. Windows

Forms побудована на основі застарілого Windows API і є по суті обгорткою низькорівневих компонентів Windows.

Windows Forms надає можливість розробки кросплатформенного графічного інтерфейсу користувача. Проте, Windows Forms фактично є лише обгорткою Windows API-компонентів, і її методів здійснює прямий доступ до Win32-функцій зворотного виклику, які недоступні інших платформах.

У .NET Framework версії 2.0 бібліотека Windows Forms отримала багатший інструментарій розробки інтерфейсів, toolstrip-елементи інтерфейсу в стилі Office 2003, підтримку багатопоточності, розширені можливості проектування та прив'язки до даних, а також підтримку технології ClickOnce для розгортання веб-додатків.

З виходом .NET Framework 3.0 Microsoft випустила новий API для малювання інтерфейсів користувача: Windows Presentation Foundation, який базувався на DirectX 11 і декларативній мові опису інтерфейсів XAML. Однак, навіть незважаючи на це, Windows Forms і WPF все ще пропонують схожу функціональність, і тому Windows Forms не був скасований на користь WPF, а продовжує використовуватися як альтернативна технологія побудови інтерфейсів поряд з WPF.

Відповідаючи на запитання на конференції Build 2014, Microsoft пояснила, що Windows Forms буде підтримуватися, помилки будуть виправлятися, але нові функції додаватися не будуть. Пізніше покращена підтримка високої роздільної здатності для різних елементів інтерфейсу Windows Forms все ж таки була анонсована в релізі .NET Framework 4.5.

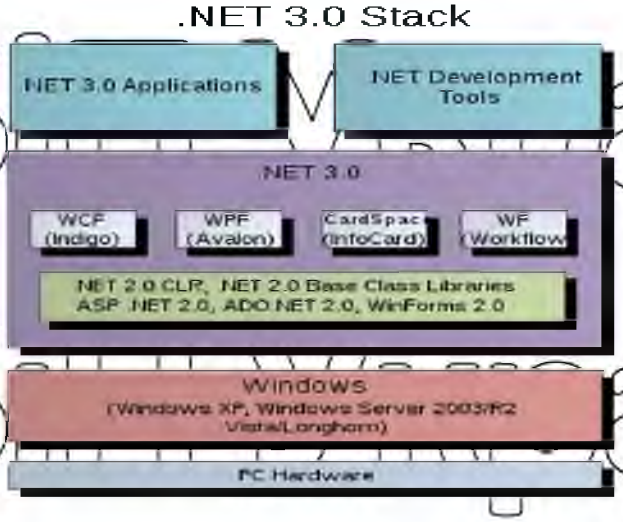

Рисунок 2.1 – Схематична структура .NET Framework

Для програмування додатку скористуємся середовищем Visual Studio.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Решта вбудованих інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми баз даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як, наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування) або інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

Далі коротко оглянемо саме середовище. Після запуску Visual Studio перш за все необхідно вибрати поєднання визначених параметрів інтегрованого середовища розробки.

Передбачається, що діють Звичайні параметри розробки, що відповідають мінімальному обсягу налаштувань інтегрованого середовища розробки. Якщо ви вже обрали C#, немає необхідності змінювати свої налаштування. Якщо ви хочете змінити налаштування, можна скористатися Майстром імпорту і експорту параметрів. Додаткові відомості є у розділі Налаштування параметрів розробки в Visual Studio.

Після відкриття Visual Studio ми бачимо вікна інструментів, меню і панель інструментів, а також головну область вікна. Вікна інструментів закріплені в лівій і правій частинах вікна програми, а панель Швидкий запуск, рядок меню і стандартна панель інструментів закріплені у верхній його частині. У центрі вікна програми знаходиться Початкова сторінка. При завантаженні рішення або проекту редактори і конструктори відображаються в області Початкової сторінки. При розробці програми найчастіше використовується саме ця область (рис 2.2).

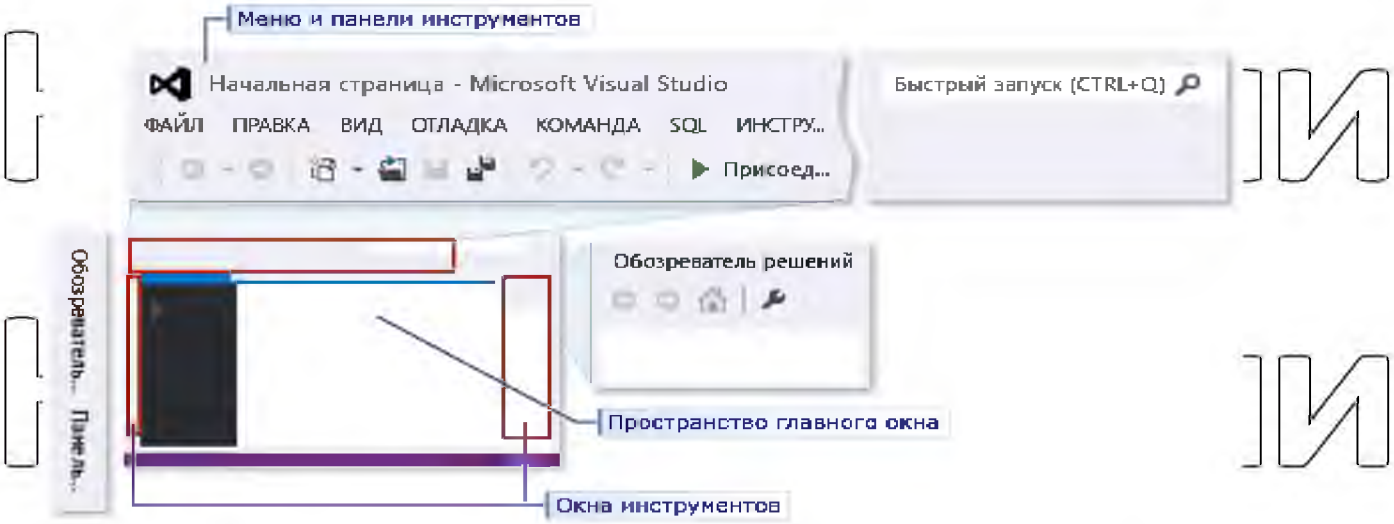

Рисунок 2.2 – Головна область Visual Studio 2015 при створенні програми

За допомогою діалогового вікна Параметри можна додатково налаштувати Visual Studio, наприклад змінити в редакторі накреслення і розмір шрифту тексту або змінити колірну тему інтегрованого середовища розробки. Залежно від застосованого поєднання параметрів деякі елементи в цьому діалоговому вікні можуть не відображатися автоматично. Щоб відображалися всі можливі параметри, потрібно встановити прапорець Показати всі параметри. [4]

Щоб створити новий проект, потрібно послідовно вибрати у меню пункти: Файл, Створити, Проект. Також можна створити проект у вікні Швидкий запуск. Можна створити шаблон Visual C#, використавши меню Встановлені, Шаблони, Visual C#, Windows, а потім обрати Додаток WPF. На рисунку 2.3 показано вікно, де можна обрати тип додатку, що створюється.

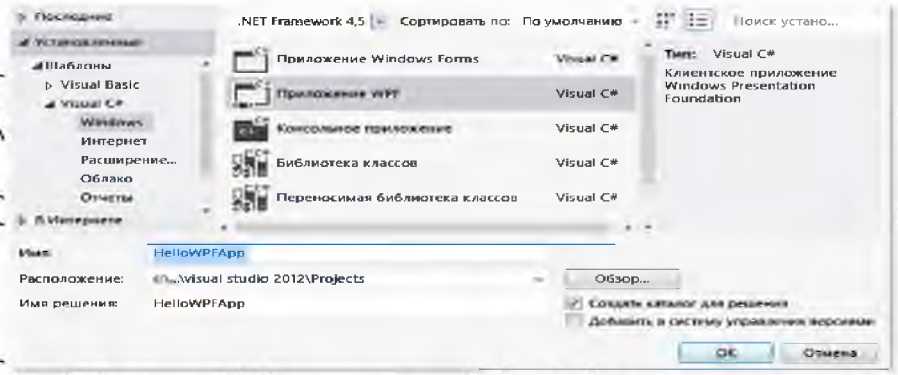

Рисунок 2.3 – Вікно вибору типу додатку

Після створення проекту його можна налаштовувати. За допомогою вікна Властивості (в меню Вид) можна відображати і змінювати параметри елементів

проекту, елементів управління та інших об'єктів у програмі. За допомогою властивостей проекту і сторінок властивостей можна відображати і змінювати параметри проектів і рішень.

Схематичну структуру додатку алгоритму k-найближчих сусідів показано на рисунку 2.4.

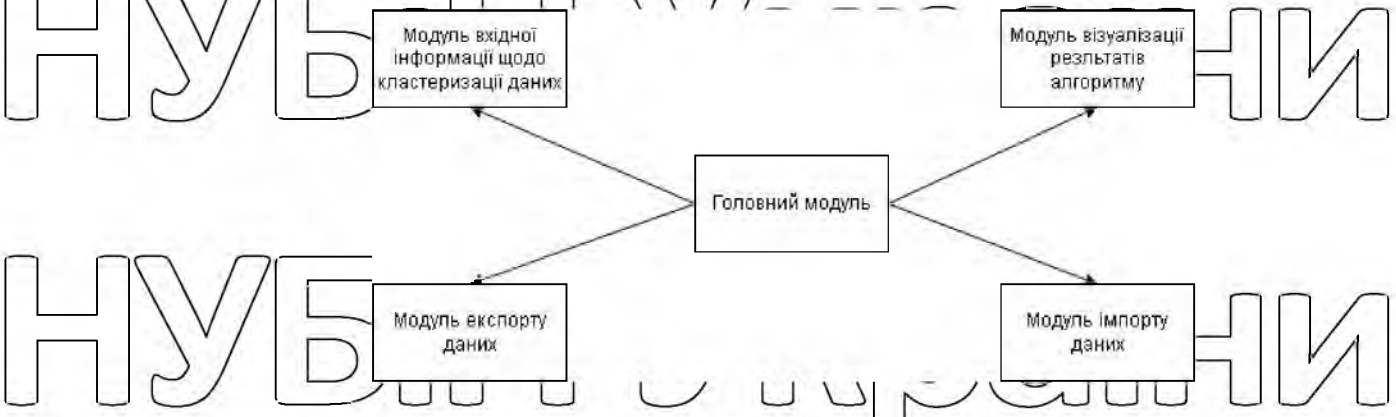

Рисунок 2.4 – Схематична структура додатку алгоритму k-найближчих сусідів

2.2 Проектування програмного додатку алгоритму SVM

Інструментальні засоби для проектування даного алгоритму будуть аналогічними, як у випадку з алгоритмом k-найближчих сусідів. Структурна схема алгоритму зображена на рисунку 2.5.

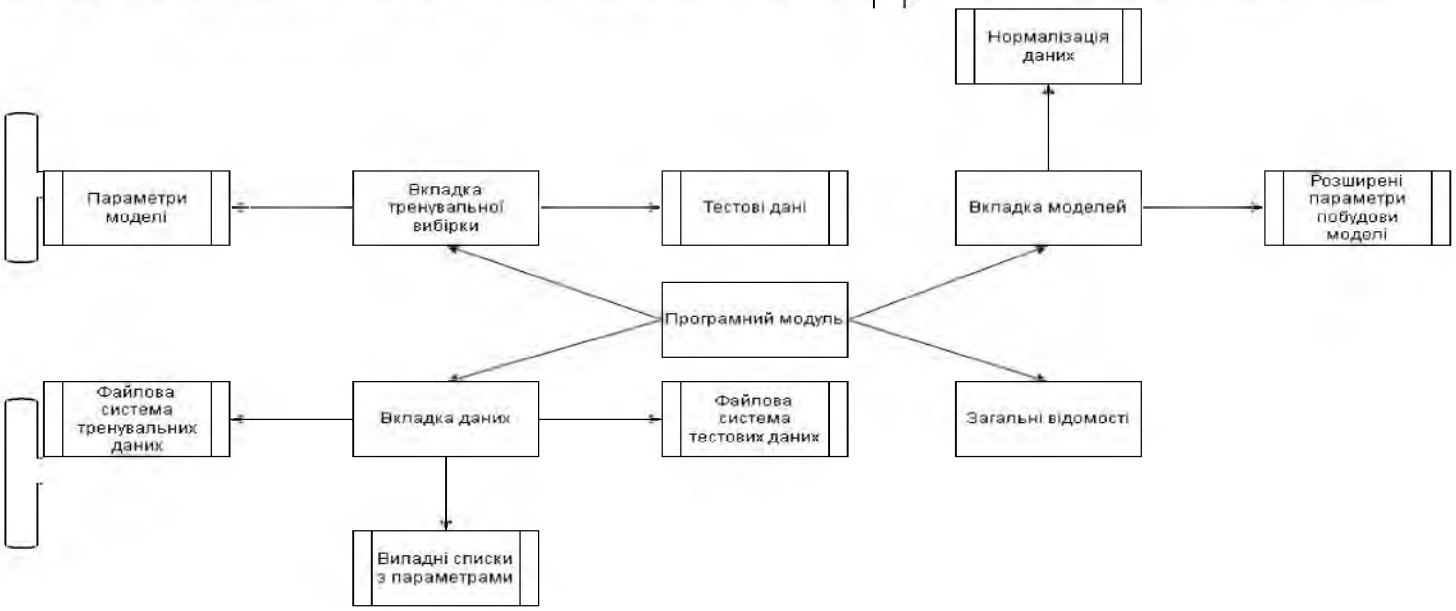

Рисунок 2.5 – Схематична структура додатку алгоритму SVM

2.3 Проектування програмного додатку алгоритму Naive Bayes

Інструментальні засоби для даного додатку будуть подібними, мова програмування так само буде C#, проте фреймворком буде обрано уже WPF – так як, він надає глибші можливості для дизайну інтерфейсу. Алгоритм Naive Bayes зазвичай може бути дуже різним в реалізації в залежності від поставленої задачі, тому його краще показати на конкретному випадку, а саме найбільш популярною прикладною задачею для даного алгоритму є фільтрація спаму в електронній пошті.

Windows Presentation Foundation (WPF, кодова назва — Avalon) — графічна (презентаційна) підсистема (аналог WinForms), яка починаючи з .NET Framework 3.0 в складі цієї платформи. Має пряме відношення до XAML. WPF разом з .NET Framework 3.0 вбудована в Windows Vista, а також доступна для установки в Windows XP Service Pack 2 і Windows Server 2003. [5]

Це перше реальне оновлення технологічного середовища призначеного для користувача інтерфейсу з часу випуску Windows 95. Воно включає нове ядро для заміни GDI і GDI+, використовувані в Windows Forms. WPF є високорівневим об'єктно-орієнтованим функціональним шаром (англ. framework), що дозволяє створювати двовимірні та тривимірні інтерфейси.

Вікно WPF може містити тільки один елемент. Щоб розмістити більше одного елемента й створити більше практичний користувальницький інтерфейс, вам потрібно помістити у вікно спеціальний елемент керування — контейнер і потім додавати елементи в цей контейнер.

В WPF компонування визначається використовуваним контейнером. Хоча є кілька контейнерів, серед яких можна вибирати будь-який «ідеальне» вікно WPF має піддаватись описаним нижче ключовим принципам

- Елементи (такі як елементи керування) не повинні мати явно встановлених розмірів. Замість цього вони ростуть, щоб заповнити їхній уміст. Наприклад, кнопка збільшується, коли ви додаєте в неї текст. Ви можете обмежити елементи керування прийнятними розмірами, установлюючи максимальне й мінімальне їхні значення.

- Елементи не вказують свою позицію в екранних координатах. Замість цього вони впорядковуються своїм контейнером на основі розміру, порядку й (необов'язково) іншої інформації, специфічної для контейнера компонування. Якщо ви хочете додати пробіл між елементами, то використовуєте для цього властивість Margin.
- Контейнери компонування «розділяють» доступний простір між своїми дочірніми елементами. Вони намагаються надати кожному елементу його найкращий розмір (на основі його вмісту), якщо дозволяє вільний простір. Вони можуть також виділяти додатковий простір дочірнім елементам.
- Контейнери компонування можуть бути вкладеними. Типовий користувальницький інтерфейс починається з Grid — найрозвиненішого контейнера, і містить інші контейнери компонування, які організовують менші групи елементів, такі як текстові поля з мітками, елементи списку, піктограми в панелі інструментів і т. д.

Компонування WPF відбувається за дві стадії: стадія виміру й стадія розміщення. На стадії виміру контейнер виконує прохід у циклі дочірніми елементами й опитує їхні найкращі розміри. На стадії розміщення контейнер поміщає дочірні елементи у відповідні позиції.

Всі контейнери компонування WPF є панелями, які успадковані від абстрактного класу System.Windows.Controls.Panel Фактично, базовий клас Panel — це не що інше, як початкова точка для побудови інших більш спеціалізованих класів. Як і всі елементи керування WPF і більшість візуальних елементів, ці класи перебувають у просторі імен System.Windows.Controls. [6]

На рисунку 2.6 зображено схематичну структуру програмного засобу з алгоритмом Naive Bayes. Наївний класифікатор використовує дві моделідля фільтрації спаму, виробляє навчальну вибірку даних після тренувальної, має власний механзім усунення помилок та повторів даних.

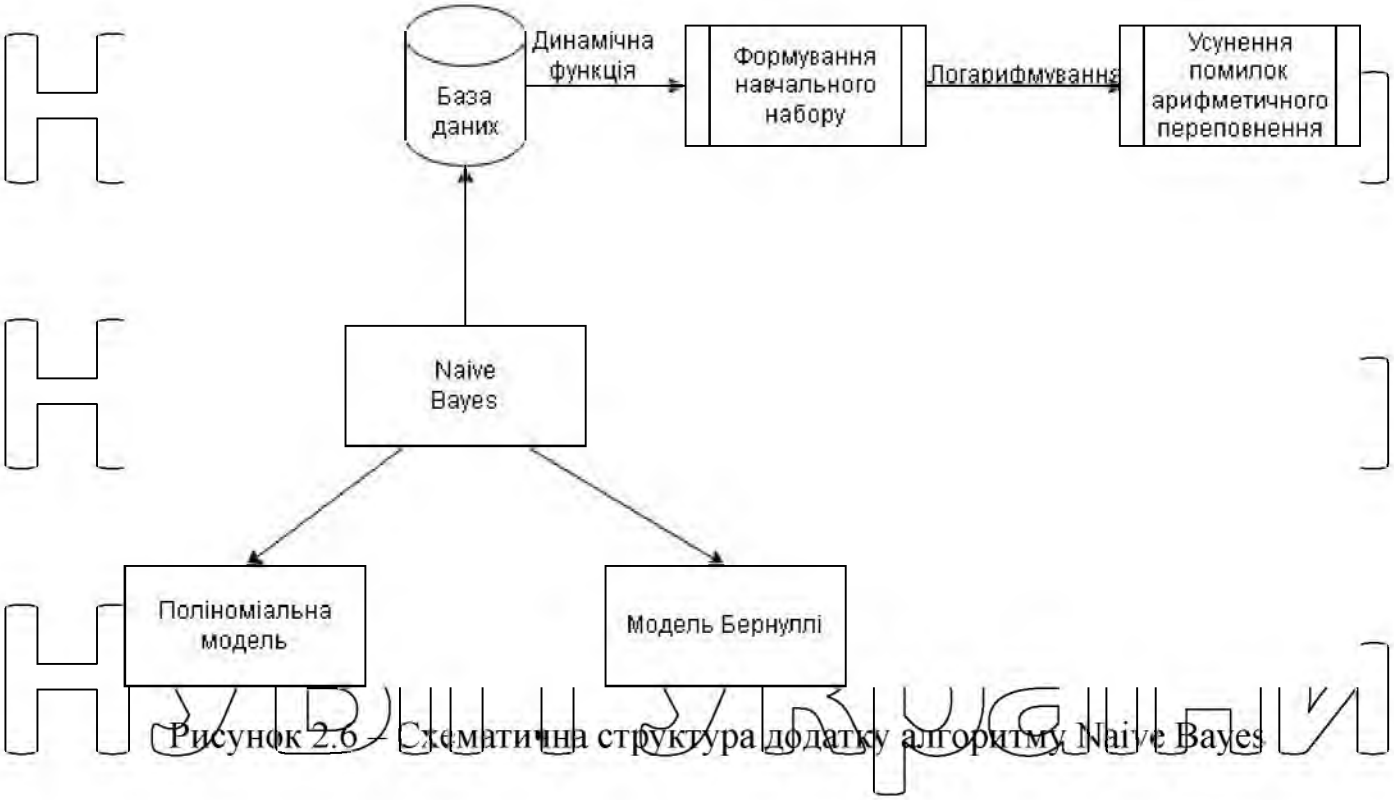

Рисунок 2.6 – Схематична структура додатку алгоритму Naive Bayes

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РЕАЛІЗОВАНИХ АЛГОРИТМІВ КЛАСИФІКАЦІЇ

3.1 Обґрунтування вибору мови програмування

Розглянемо декілька мов програмування, для того щоб довести ефективність застосування саме мови C# для програмної реалізації даних алгоритмів.

Порівняльна характеристика цих мов подана у табл. 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Назва	Опис	Переваги	Недоліки
C#	Розробка ОС та офісних рішень для платформи Microsoft, розробка веб-додатків, настільних графічних додатків (використовуються платформи Windows Forms и WPF), серверних додатків в сфері створення динамічного веб-контенту за	• Висока швидкість розробки в малобюджетних проектах • Простота і лаконічність коду • Наявність власного середовища розробки • Можливості спадкування й універсалізації • Типи, вбудовані в мову, представлені	• Проблеми при розробці додатків під не-Windows платформи • Сильна прив’язка до Microsoft • Нема самодостатності додатків, так як використовується .net framework • Відсутність повноцінних

Продовження таблиці 3.1

	допомогою платформи ASP.NET, мобільних додатків для операційної системи Windows Phone, що розроблена компанією Microsoft.	класами • Збереження і об’єднання кращих рис мов С і С++ • Підвищення ефективності коду, оскільки виконавче середовище CLR являє собою компілятор проміжної мови • Зручність побудови різних типів додатків дозволяє легко розробляти Web-служби	деструкторів, повноцінних макросів, дуже обмежена підтримка шаблонів
С++	Створення ОС, прикладних програм, драйверів приладів, додатків для вбудованих систем, високопродуктивних серверів, а також	• Висока сумісність з мовою С • Підтримка різних стилів програмування • Можливість побудови загальних контейнерів і алгоритмів для	• Погано продуманий синтаксис робить мову незручною в деяких задачах • Нема самодостатньості додатків, для

	програмування ігор.	різних типів даних • Кросплатформенність • Доступність і популярність	цього використовується runtime • Продуктивність праці програмістів на мові виявляється невиправдано низькою, а продукт праці є низькоякісним
Java	Створення десктопів і аплетів, мобільних додатків, серверних додатків, що орієнтовані на роботу з мережею, програмування ігор.	• Універсальність і широта застосування • Кросплатформенність • Відкритий вихідний код і велика кількість бібліотек • Гнучка система безпеки • Висока продуктивність • Багатозадачність	• Мова не має таких властивостей об'єктно-орієнтованої мови, як індивідуальні змінні та множинне спадкування • Відсутність спливаючих підказок до методів, які можна примінити до певного

Продовження таблиці 3.1

			об’єкта
Delphi	Об'єктно-орієнтована мова програмування, нащадок Pascal, більш відома як основна мова програмування середовища Delphi.	• Підтримка багатьох компіляторів • Зручність в створенні експертних систем • Якісний графічний інтерфейс • Простота	• Консервативність

З усіх перерахованих мов програмування, було вирішено обрати C# як оптимальну мову розробки додатку.

Опишемо дану мову більш детально.

C# (вимовляється *Сі-шарп*) — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів.

Станом на 2021 рік поточна стабільна версія мови C# 10.0, яка була випущена в 2021 році як частина платформи .NET 6.0.

Хоча визначення мови C# і CLI стандартизовані ISO та Ecma, що забезпечує розумний і недискримінаційний ліцензійний захист (RAND) від патентних позовів, Microsoft використовує C# і CLI у своїй бібліотеці Base Class Library (BCL), яка є фундаментом їхньої власницької платформи .NET framework, і яка надає низку нестандартизованих класів (розширений I/O, GUI Windows Forms, вебслужби тощо).

У деяких випадках, де патенти Microsoft відносяться до стандартів, використаних у .NET framework, документовані Microsoft, і застосовані патенти доступні через інші RAND умови або через Обітницю Відкритої Специфікації Microsoft (Microsoft's Open Specification Promise, OSP), які випускають патентні права публічно. Але є деякі застереження і обговорення про те, що існують додаткові аспекти, патентовані Microsoft, що не покриті, які можуть утримувати незалежних реалізаторів повного фреймворку.

Microsoft також погодився не позиватися проти розробників відкритого програмного забезпечення щодо порушення прав у неприбуткових проєктах для частини свого фреймворку, покритого OSP. Microsoft погодився не порушувати патентних вимог щодо продуктів Novell проти платних клієнтів Novell за винятком переліку продуктів, що явно не згадують C#, .NET чи реалізацію .NET від Novell (проєкт Mono). Проте Novell дотримується точки зору, що Mono не порушує жодного патенту Microsoft. Microsoft також уклав спеціальну угоду не позиватися проти браузерного плагіну Moonlight, який спирається на Mono, отриманного від Novell.

У зауваженні, опублікованому на сайті новин Free Software Foundation у червні 2009 Річард Столлман попереджає, що він вважає, що «Microsoft можливо планує одного дня оголосити всі вільні реалізації C# такими, що використовують програмні патенти» і рекомендував розробникам уникати того, що він називає «безвідплатним ризиком», пов'язаним із «залежністю вільних реалізацій C#». Free Software Foundation пізніше повторила свої попередження, стверджуючи, що розширення Microsoft Community Promise на специфікації ECMA C# і CLI можуть не вберігти від шкідництва Microsoft відкритим реалізаціям C#, оскільки багато специфічних для Windows бібліотек, включених у .NET та Mono, не покриті цими обіцянками. Тому більшість провідних дистрибутивів Лінукс, за винятком Novell SUSE Linux, не

включають Mono в установку за умовчанням (хоча його і можна завантажити з репозиторіїв).

C# розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність буде порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо. [7]

C# підтримує строго типізовані неявні оголошення змінних з ключовим словом var і неявно типізовані масиви з ключовим словом new [], за яким слідує ініціалізатор колекції.

C# підтримує суворий тип даних Boolean, bool. Вирази, які приймають умови, такі як while та if, вимагають висловлювання, що реалізує оператор true або false. Хоча C++ також має тип Boolean, він може бути вільно перетворений в цілі числа та з них, а вирази, такі як if(a), вимагають тільки того, щоб a був конвертований в bool, що дозволяє бути a int-типу або вказівником. C# забороняє «ціле значення означає справжній або помилковий підхід» на тій підставі, що примус програмістів використовувати вирази, які повертають точно bool, можуть створювати деякі типи помилок програмування, наприклад if (a = b) (використання присвоювання = замість рівності == , які, хоча і не є помилкою на C або C++, все одно будуть спіймані компілятором).

C# безпечніший в порівнянні з C++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад, розширення цілих чисел. Це застосовується під час компіляції, під час JIT і, в деяких випадках, під час виконання. Не відбувається неявних перетворень між булевими і цілими числами, а

також між членами перерахування і цілими числами (крім літерала 0, який може бути неявно перетворений в будь-який нумерований тип). Будь-яке призначене для користувача перетворення повинно бути явно позначене як явне або неявне, на відміну від конструкторів копіювання C++ і операторів перетворення, які за умовчанням є неявними.

C# має явну підтримку коваріації та контраваріантності в родових типах, на відміну від C++, яка має певний рівень підтримки контраваріантності просто через семантику типів,що повертаються, на віртуальні методи.

Члени перерахування розміщуються в своєму власному обсязі.

Мова C # не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні та функції.

Мета програмування через атрибути C# є частиною мови. Багато з цих атрибутів дублюють функціональні можливості директив препроцесора, орієнтованих на платформу GCC і VisualC ++.

Методи в мові програмування є членами класу в проєкті, деякі методи мають підписи, а деякі не мають підпису.

Методи можуть бути недійсними або можуть повертати щось на зразок рядка, цілого, подвійного, десяткового, float і bool. Якщо метод недійсний, це означає, що метод не повертає жодного типу даних.

Подібно C++, і на відміну від Java, програмісти на C # повинні використовувати ключове слово virtual, щоб дозволити перевизначати методи підкласами.

Методи розширення в C# дозволяють програмістам використовувати статичні методи, як якщо б вони були методами з таблиці методів класу, дозволяючи програмістам додавати методи до об’єкта, який, на їхню думку, повинен існувати на цьому об’єкті і його похідних.

Динамічний тип dynamic допускає прив’язку методу під час виконання, що дозволяє використовувати JavaScript-подібні виклики методів і склад часу виконання.

У C# є підтримка строго типізованих покажчиків функцій через delegate ключового слова. Подібно псевдо-C ++ — signal і slot фрейма Ct, C# має семантику,

спеціально пов’язану з подіями стилю публікації-підписки, хоча C# використовує делегати для цього. C # пропонує Java-подібні синхронізовані synchronized виклики методів через атрибут [MethodImpl (MethodImplOptions.Synchronized)] і підтримує взаємовиключні блокування за допомогою блокування ключових слів. [8]

C# надає властивості як синтаксичного цукру для загального шаблону, в якому пара методів, accessor (getter) і mutator (setter) інкапсулює операції по одному атрибуту класу. Не потрібно писати надлишкові сигнатури методів для реалізацій геттера / сетера і до цієї властивості можна отримати доступ, використовуючи синтаксис атрибутів, а не більш докладні виклики методів.

3.2 Тестування розроблених додатків алгоритмів класифікації

Тестування програмного коду - процес виконання програмного коду, спрямований на виявлення існуючих в ньому дефектів. Під дефектом тут розуміється ділянку програмного коду, виконання якого за певних умов призводить до несподіваного поведінки системи (тобто поведінки, що не відповідає вимогам). Несподівану поведінку системи може призводити до збоїв у її роботі і відмов, в цьому випадку говорять про істотні дефекти програмного коду. Деякі дефекти викликають незначні проблеми, що не порушують процес функціонування системи, але кілька утрудняють роботу з нею. У цьому випадку говорять про середні або малозначні дефекти.

Опишемо методи тестування, засобами якого будем тестувати дані алгоритми класифікації.

Тестування чорного та білого ящиків є двома основними методами оцінки поведінки та продуктивності продукту. Тестування чорного ящика, яке також називають функціональним або тестуванням на основі специфікацій, зосереджується на результатах. Тестерів не хвилюють внутрішні механізми. Вони лише перевіряють, чи програмне забезпечення робить те, що воно повинно робити. Знання

програмування не є обов'язковим, а тестери працюють на рівні інтерфейсу користувача.

Тестування білого ящика використовує досвід кодування як частину процедури тестування. Коли продукт виходить з ладу, тестери заглиблюються в код, щоб знайти причину. Розробники програмного забезпечення зроблять це самі, оскільки клієнти очікують, що вони змусять продукт працювати. Тестування білого ящика також називають тестуванням "на основі структури" або "скляного ящика".

Під час статичного тестування перевіряється вихідний код і будь-яка супровідна документація, але програма не виконується. Статичні випробування починаються на ранніх етапах розробки продукту під час процесу перевірки.

Динамічне тестування використовує різні вхідні дані під час роботи програмного забезпечення, а тестери порівнюють результати з очікуваною поведінкою. Тестування графічного інтерфейсу користувача оцінює форматування тексту, текстові поля, кнопки, списки, макет, кольори та інші елементи інтерфейсу. Тестування GUI займає багато часу, і сторонні компанії часто беруться за це завдання замість розробників. [10]

Різні рівні тестування використовуються для виявлення слабких місць і збігів на кожному етапі життєвого циклу розробки програмного забезпечення. Рівні тестування:

-Модульний тест

-Інтеграційне тестування

-Тестування системи

-Приймальні випробування

Під час модульного тестування розробники тестують основні частини коду, такі як класи, інтерфейси та функції/процедури. Вони знають, як має реагувати їхній код, і можуть вносити зміни залежно від результату.

Інтеграційне тестування також відоме як «модульне» або «програмне» тестування. Це схоже на модульне тестування, але містить вищий рівень інтеграції. Модулі програмного забезпечення перевіряються на наявність дефектів, щоб перевірити їхню роботу. Тестування інтеграції визначає помилки під час інтеграції

модулів. Різні методи інтеграційних тестів включають «знизу вгору», «зверху вниз» і «функціональний інкрементний».

Системне тестування перевіряє компоненти проекту в цілому в різних середовищах. Тестування системи підпадає під метод чорного ящика і є одним із останніх тестів у процесі. Він визначить, чи готова система задовольнити потреби бізнесу та користувачів.

Зазвичай існує два типи приймальних випробувань. Під час альфа-тестування програмне забезпечення виконується внутрішньо на сайті розробника в змодельованому або реальному середовищі. Програмне забезпечення працює так, ніби ним користуються живі кінцеві користувачі. Розробники роблять нотатки про будь-які проблеми та починають виправляти помилки та інші проблеми.

Бета-тестування або польове тестування дозволяє клієнтам перевірити продукт на своїх сайтах у реальних умовах. Клієнти можуть запропонувати групі кінцевих користувачів можливість протестувати програмне забезпечення за допомогою попередніх або бета-версій. Метою бета-тестування є отримання фактичних відгуків користувачів, які надсилаються розробнику.

Різні типи тестів програмного забезпечення призначені для досягнення конкретних цілей. Тестовий інженер і менеджер конфігурації використовують тестування встановлення, щоб переконатися, що кінцевий користувач може встановити та запустити програму. Він охоплює такі області, як інсталяційні файли, розташування інсталяції та адміністративні привілеї.

Розвивальне тестування реалізує низку синхронізованих стратегій для виявлення та запобігання дефектам. Він включає статичний аналіз коду, рецензування коду, відстеження та аналіз показників. Мета – зменшити ризики та заощадити витрати.

Взаємодія з користувачем потрапляє в центр уваги завдяки тестуванню зручності використання. Він вимірює, наскільки простий графічний інтерфейс користувача. Він перевіряє точність і ефективність функцій і емоційні реакції підопослідних.

Перевірка працездатності показує, чи варте програмне забезпечення часу та витрат на продовження подальших тестів. Якщо недоліків забагато, більш агресивні тести не підуть.

Тестування працездатності виконується на етапі випуску програмного забезпечення, де проводиться димове тестування, щоб побачити, чи буде програмне забезпечення працювати достатньо, щоб його можна було перевірити.

Тестування диму виявляє фундаментальні недоліки, які є досить серйозними, щоб запобігти викиду. Коли розробники тестують нову збірку, це називається тестом «перевірки збірки». Коли система зазнає модифікації, регресійне тестування відстежує неочікувану поведінку. Він вказує на негативний вплив на модулі чи компоненти.

Тестувальники вводять ненормальні записи та розпізнають здатність програмного забезпечення керувати неочікуваними введеннями в руйнівних тестах. Це показує розробникам, наскільки надійна програма в управлінні помилками.

У разі збою обладнання або інших функцій тестування відновлення показує, наскільки добре програмне забезпечення може відновитися та продовжити роботу.

Автоматизація виконує функції, які складно реалізувати вручну. Тестування передбачає використання спеціального програмного забезпечення для проведення тестів і надання даних про фактичні та очікувані результати.

Програмне забезпечення має працювати в різних обчислювальних середовищах, тому тестування на сумісність перевіряє, як програмне забезпечення реагує на різні системи. Наприклад, програмісти тестують програмне забезпечення з різними операційними системами та веб-браузерами.

Випробування мають бути широкими та вирішувати всі проблеми клієнтів, інакше проект швидко перетвориться на марну трату ресурсів.

Тестування продуктивності перевіряє продуктивність програмного забезпечення в різних сценаріях. Збирається інформація про оперативність, стабільність, розподіл ресурсів і швидкість. Суб-тести, такі як об'єм, ємність і пікове тестування, відіграють важливу роль у цьому процесі.

Тестування безпеки визначає здатність програмного забезпечення захищати безпеку користувачів. Функції авторизації, автентифікації, конфіденційності,

цілісності, доступності та неспростовності – це приклади функцій, які необхідно протестувати.

Тестування доступності відрізняється від тестування зручності використання. Це визначає ступінь, до якої користувачі з різними здібностями можуть використовувати програмне забезпечення.

Результати тестів інтерналізації та локалізації показують, як програмне забезпечення може адаптуватися до різних мов і регіональних потреб. Це включає додавання компонентів для певних місць і переклад тексту.

Запустимо додаток з алгоритму класифікації методом к найближчих сусідів, підвантажимо з файлової системи дані і введемо необхідні дані щодо кластерів. Результати роботи додатку показано на рисунках 3.1 – 3.2.

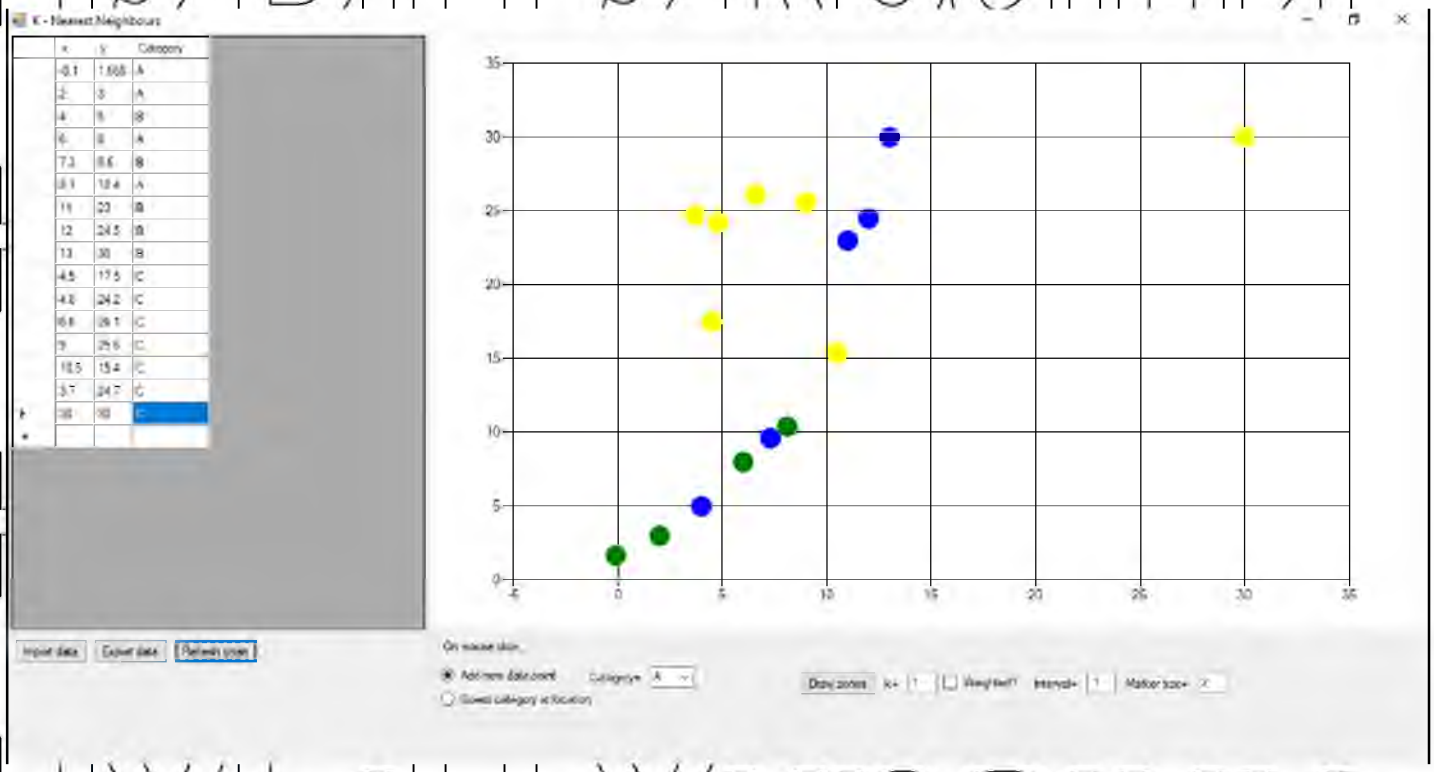

Рисунок 3.1 – Відображення точок з вибірки даних

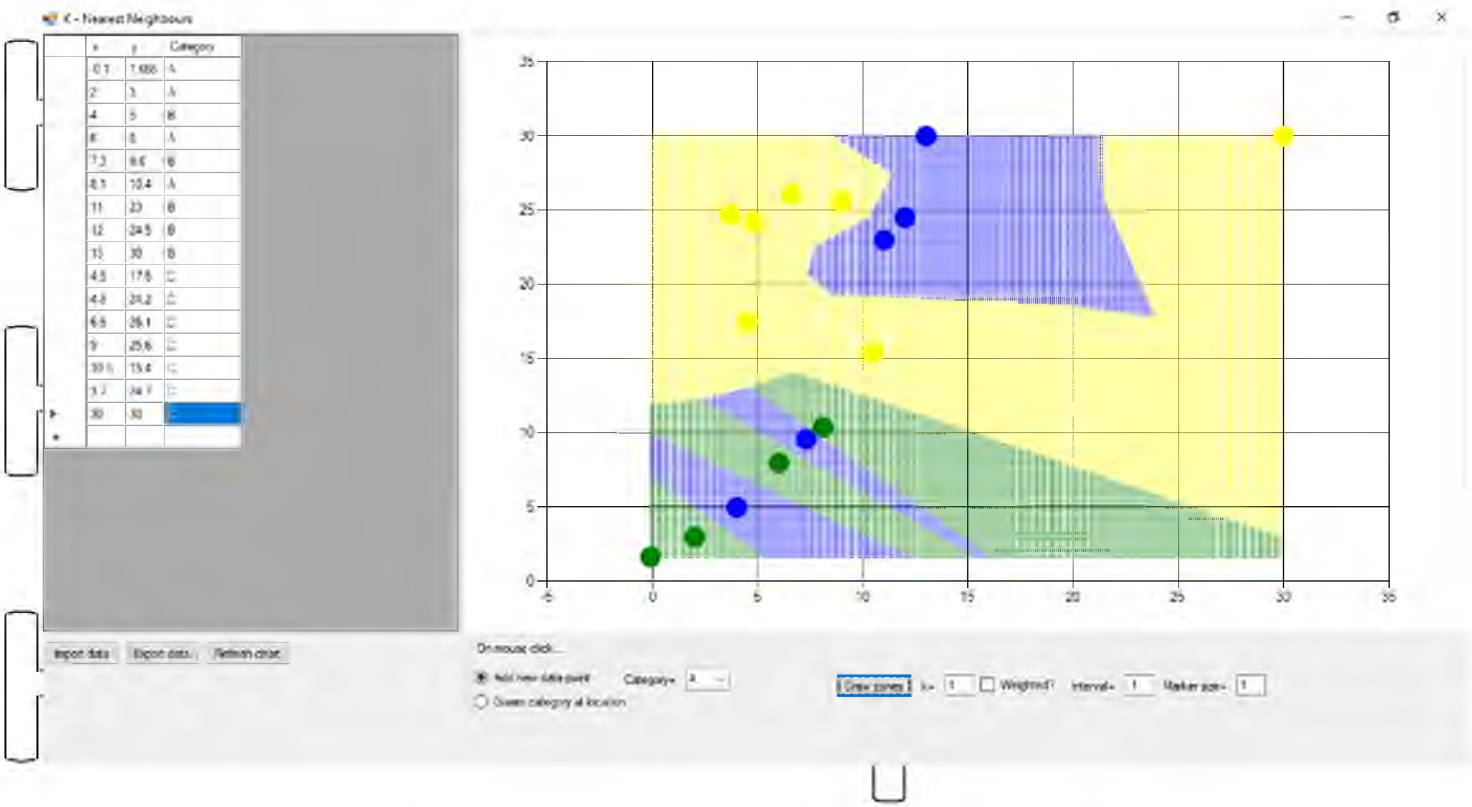

Рисунок 3.2 – Результуючі кластерні області за методом к найближчих сусідів

Додаток, що реалізує алгоритм класифікації методу опорних векторів, продемонстровано на рисунках 3.3 – 3.5.

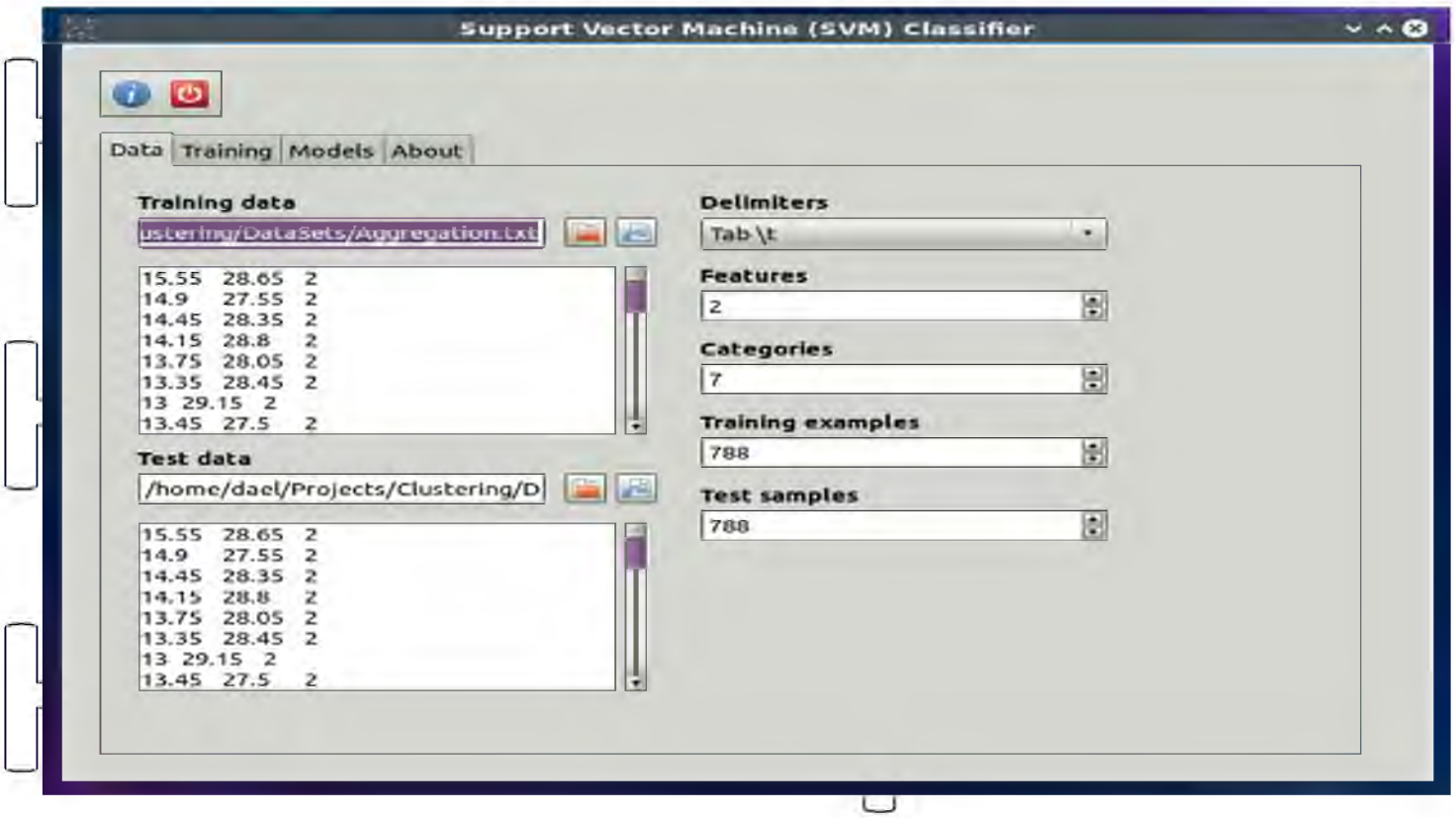

Рисунок 3.3 – Метод опорних векторів, вкладка дані

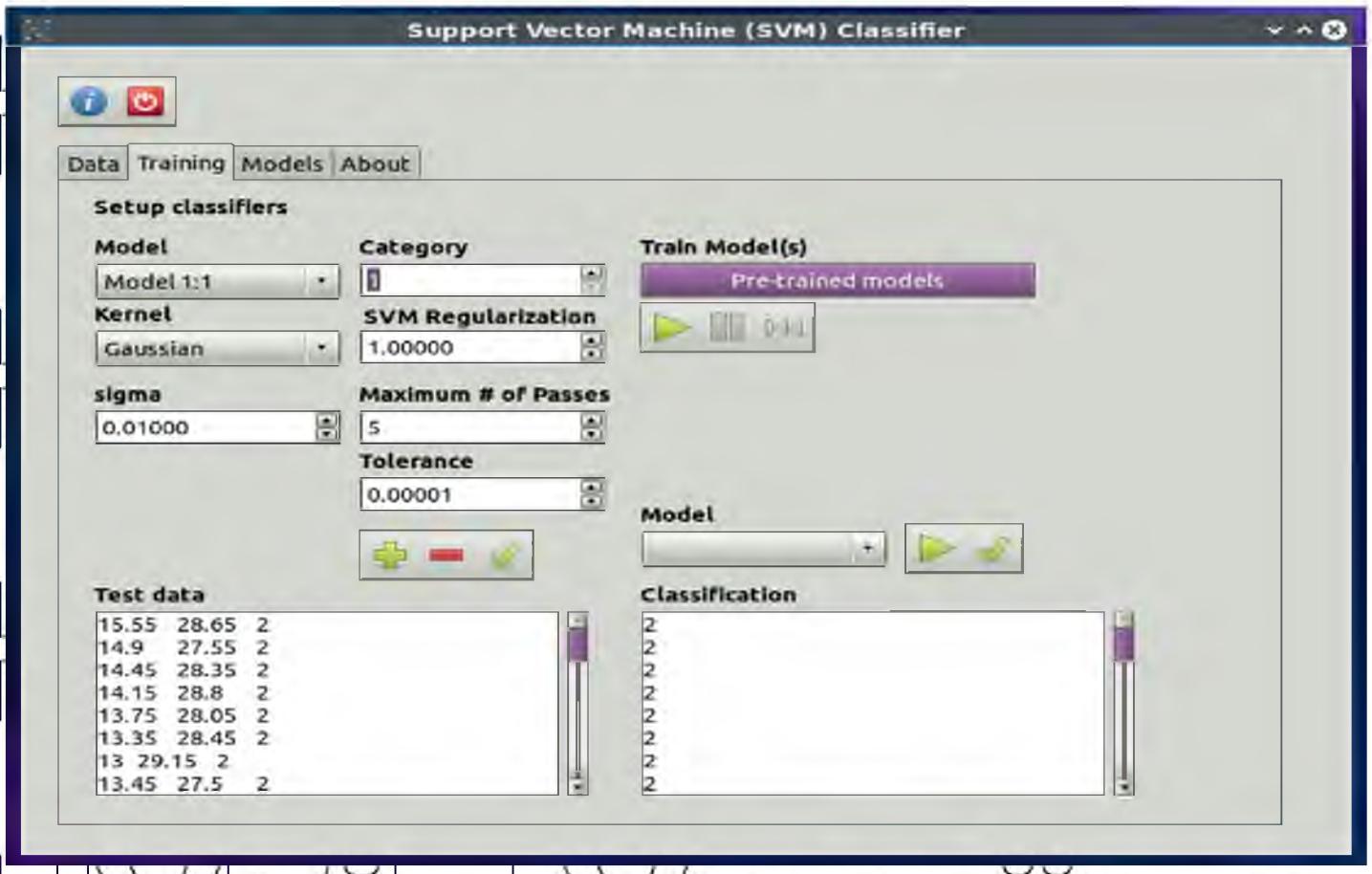

Рисунок 3.4 – Метод опорних векторів, вкладка тренування

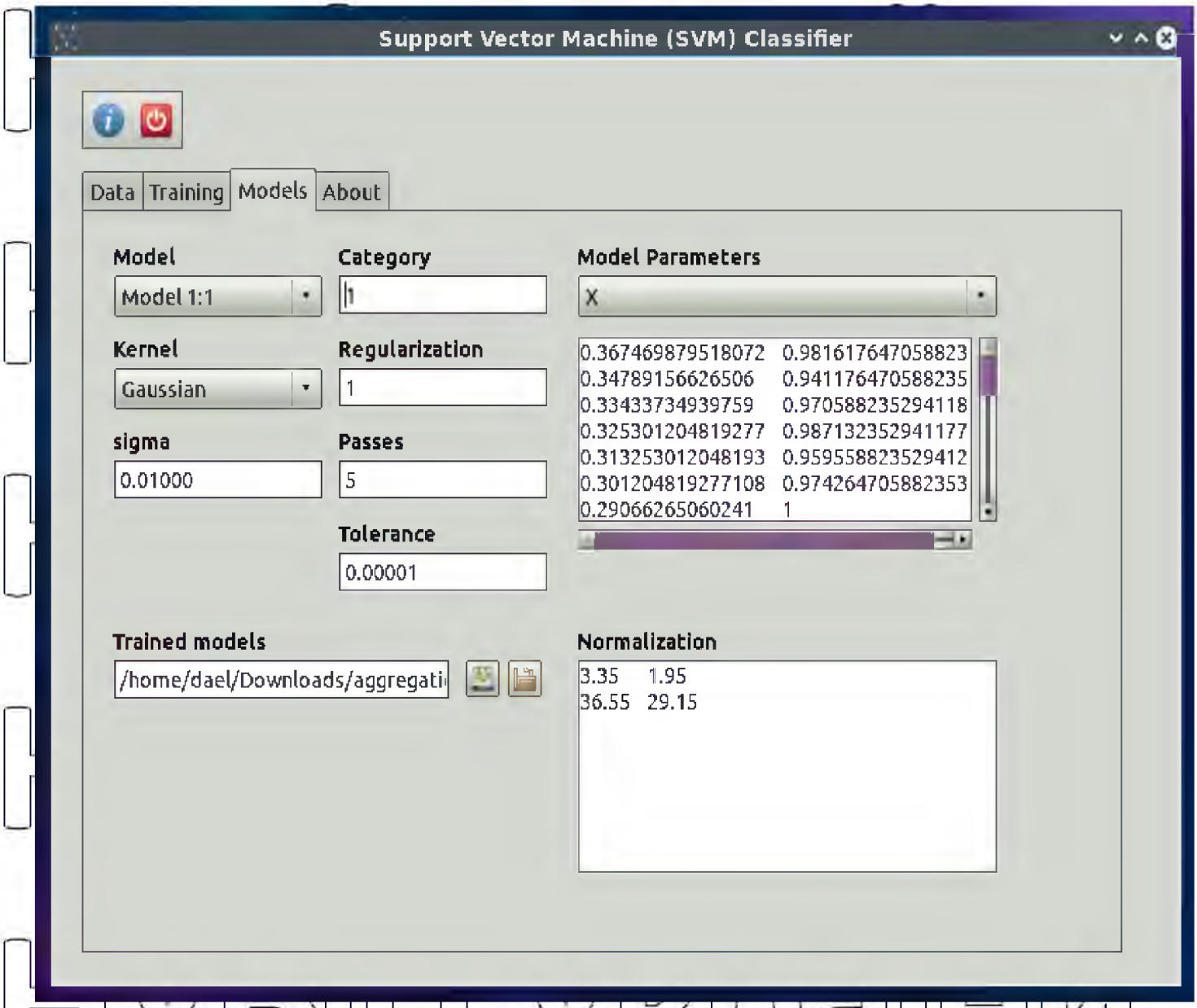

Рисунок 3.5 – Метод опорних векторів, вкладка моделі

Рисунки 3.6 – 3.7 показують роботу третього з реалізованих алгоритмів класифікації, а саме наївного байєсовського класифікатору на конкретному прикладі фільтрації спаму в електронній пошті.

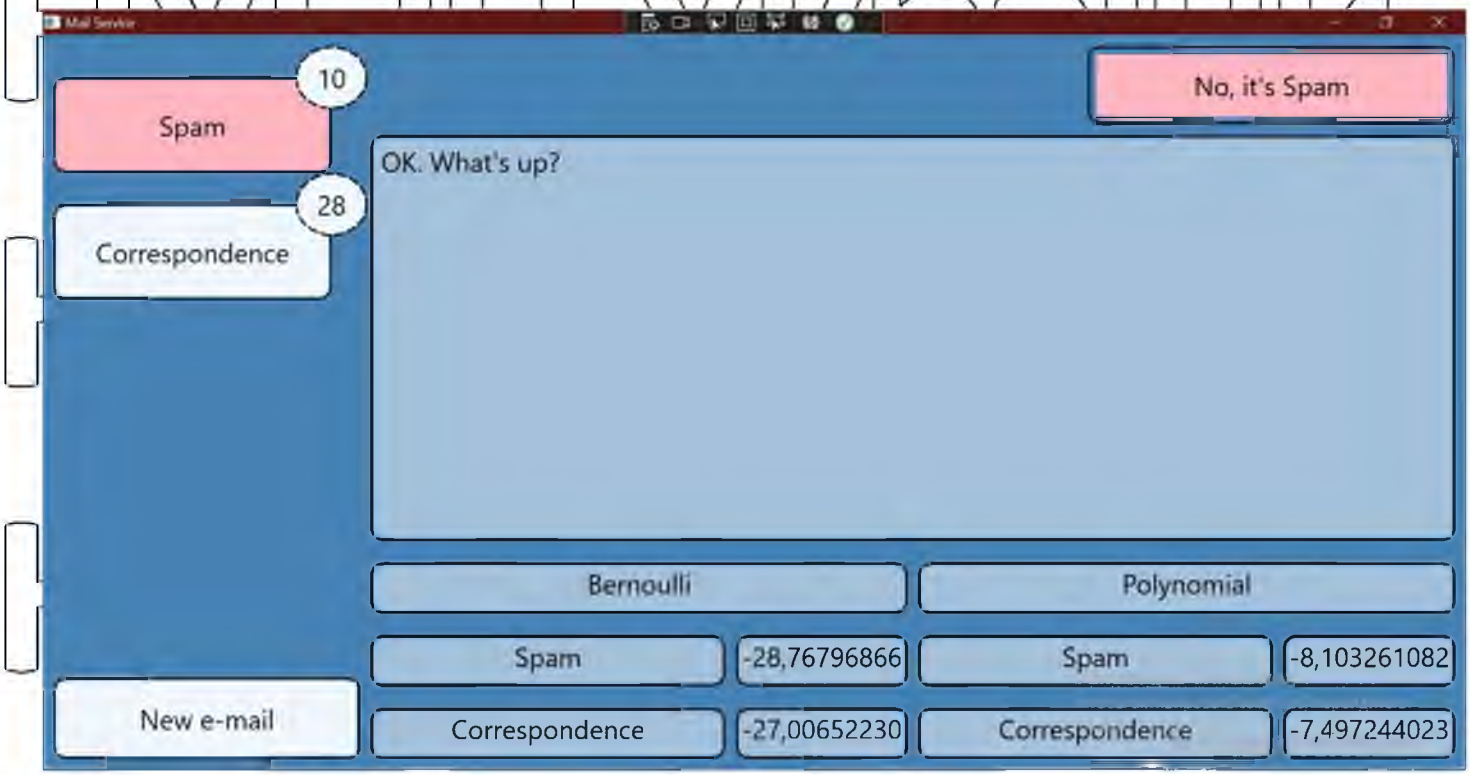

Рисунок 3.6 – Реалізація алгоритму класифікації Naive Bayes

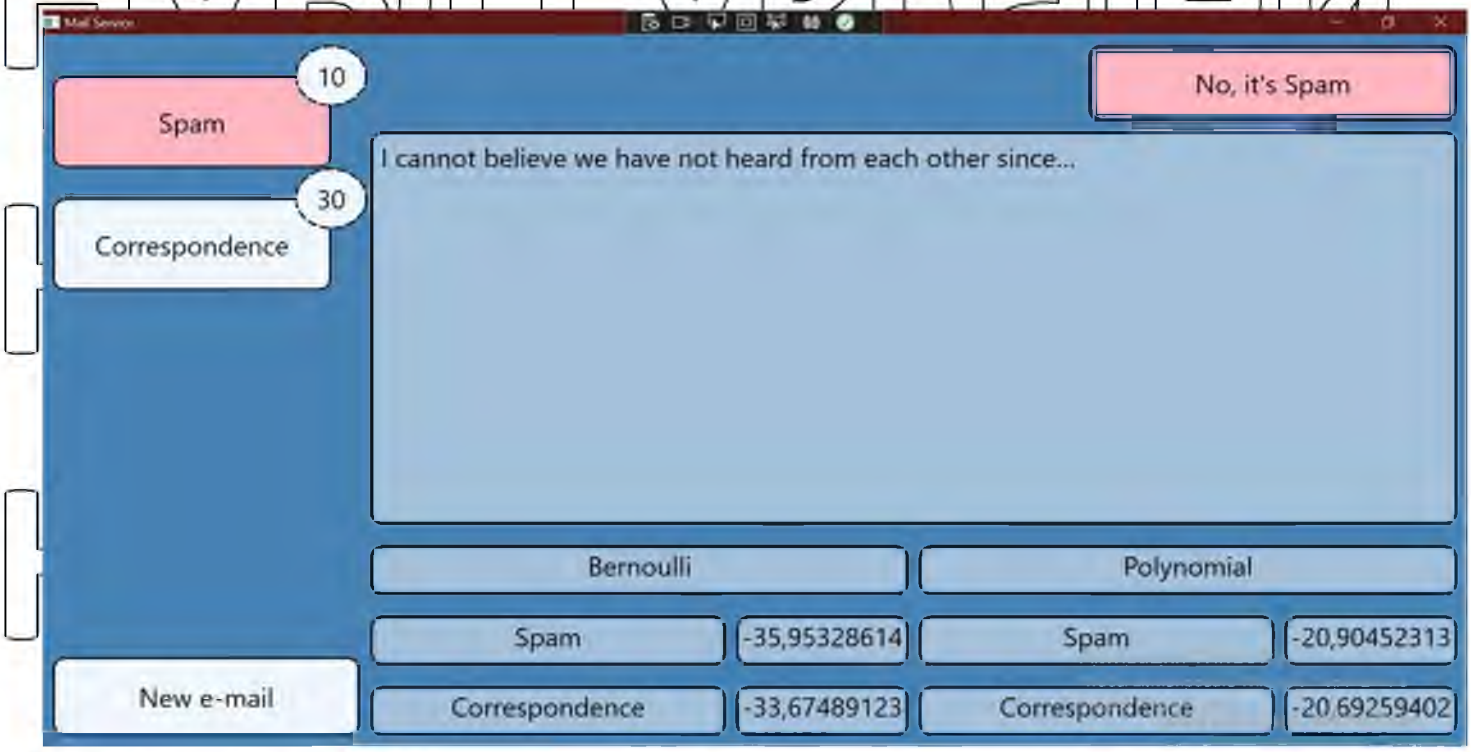

Рисунок 3.7 – Реалізація алгоритму класифікації Naive Bayes

3.3 Тестування алгоритмів класифікації за різними параметрами

Оцінка алгоритму машинного навчання є важливою частиною будь-якого проекту. Модель може дати задовільні результати при оцінці за допомогою метрики, скажімо, accuracy_score, але може дати погані результати при оцінці за іншими показниками, такими як logarithmic_loss або будь-яким іншим подібним показником. У більшості випадків ми використовуємо точність класифікації для вимірювання ефективності нашої моделі, однак цього недостатньо, щоб справді оцінити нашу модель.

Точність класифікації – це те, що ми зазвичай маємо на увазі, коли використовуємо термін точність. Це відношення кількості правильних прогнозів до загальної кількості вхідних вибірок.

Це добре працює, лише якщо є однакова кількість зразків, що належать до кожного класу.

Наприклад, нехай в навчальному наборі є 98% зразків класу A і 2% зразків класу B. Тоді наша модель може легко отримати 98% точності навчання, просто передбачивши кожну навчальну вибірку, що належить до класу A.

Коли ту саму модель перевіряють на випробувальному наборі з 60% зразків класу A та 40% зразків класу B, то точність тесту впаде до 60%. Точність класифікації велика, але дає нам помилкове відчуття досягнення високої точності.

Справжня проблема виникає, коли вартість неправильної класифікації зразків другорядного класу дуже висока. Якщо ми маємо справу з рідкісною, але смертельною хворобою, вартість недіагностування захворювання хворої людини набагато вища, ніж вартість відправки здорової людини на додаткові тести.

Логарифмічна втрата або логарифмічна втрата працює шляхом покарання за помилкові класифікації. Це добре працює для багатокласової класифікації. Під час роботи з Log Loss класифікатор повинен призначити ймовірність кожному класу для всіх зразків. Припустимо, що є N зразків, що належать до M класів, тоді Log Loss обчислюється, як показано нижче:

де,

y_ij, вказує, чи належить вибірка i до класу j чи ні

p_ij, вказує на ймовірність приналежності вибірки i до класу j

Log Loss не має верхньої межі та існує в діапазоні $[0, \infty)$. Якщо Log Loss ближче до 0, це вказує на вищу точність, тоді як якщо Log Loss віддалений від 0, це вказує на нижчу точність.

Загалом мінімізація втрат журналу забезпечує більшу точність класифікатора.

Матриця плутанини, як випливає з назви, дає нам матрицю як результат і описує повну продуктивність моделі.

Припустимо, що ми маємо проблему двійкової класифікації. У нас є кілька зразків, що належать до двох класів: ТАК або НІ. Крім того, у нас є власний класифікатор, який передбачає клас для даної вхідної вибірки. Перевіривши нашу модель на 165 зразках, ми стримали наступний результати, які подані в таблиці 3.1.

Таблиця 3.2 – Результати перевірки моделі на 165 зразках

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Площа під кривою (AUC) є одним із найбільш широко використовуваних показників для оцінювання. Він використовується для задачі двійкової класифікації. AUC класифікатора дорівнює ймовірності того, що класифікатор оцінить навмання вибраний позитивний приклад вище, ніж навмання вибраний негативний приклад. Перш ніж визначити AUC, давайте розберемося з двома основними термінами:

Істинно позитивний рівень (чутливість) : Справжній позитивний рівень визначається як TP/(FN+TP). Справжня позитивна оцінка відповідає частці позитивних точок даних, які правильно вважаються позитивними, відносно всіх позитивних точок даних.

Справді негативний рівень (специфічність): Справжній негативний показник визначається як TN / (FP+TN). Рівень хибнопозитивних результатів відповідає частці негативних точок даних, які правильно вважаються негативними, відносно всіх негативних точок даних.

Частота помилкових позитивних результатів: Частота помилкових позитивних результатів визначається як FP / (FP+TN). Рівень хибнопозитивних результатів відповідає частці негативних точок даних, які помилково вважаються позитивними, відносно всіх негативних точок даних.

Рівень хибнопозитивних і істинно позитивних результатів мають значення в діапазоні [0, 1]. FPR і TPR обчислюються при різних порогових значеннях, наприклад (0,00, 0,02, 0,04, …, 1,00), і будується графік. AUC – це площа під кривою графіка частоти помилкових позитивних результатів проти частоти справжніх позитивних результатів у різних точках [0, 1].

Оцінка F1 – це гармонійне середнє між точністю та пам’яттю. Діапазон для оцінки F1 становить [0, 1]. Він повідомляє вам, наскільки точним є ваш класифікатор (скільки екземплярів він класифікує правильно), а також наскільки він надійний (він не пропускає значну кількість екземплярів).

Висока точність, але нижче запам’ятовування дає вам надзвичайно точний результат, але потім пропускає велику кількість випадків, які важко класифікувати. Чим вищий показник F1, тим краща продуктивність нашої моделі. Математично це можна виразити так:

F1 Score намагається знайти баланс між точністю та запам’ятовуванням.

Точність: це кількість правильних позитивних результатів, поділена на кількість позитивних результатів, передбачених класифікатором.

Відкликання: це кількість правильних позитивних результатів, поділена на кількість усіх відповідних зразків (усі зразки, які мали бути визначені як позитивні).

Середня абсолютна похибка – це середнє значення різниці між початковими значеннями та прогнозованими значеннями. Це дає нам міру того, наскільки далекі прогнози були від фактичного результату. Однак вони не дають нам жодного уявлення про напрямок помилки, тобто про те, чи ми прогнозуємо дані за недостатньо, чи надто прогнозуємо їх. Математично це представлено у вигляді:

Середня квадратична помилка (MSE) дуже схожа на середню абсолютну помилку, єдина відмінність полягає в тому, що MSE бере середнє значення квадрата різниці між початковими значеннями та прогнозованими значеннями. Перевагою MSE є легше обчислити градієнт, тоді як середня абсолютна похибка потребує складних інструментів лінійного програмування для обчислення градієнта. Оскільки ми беремо квадрат помилки, ефект більших помилок стає більш вираженим, ніж менша помилка, отже, модель тепер може більше зосереджуватися на більших помилках.

Розроблені реалізації трьох алгоритмів класифікації необхідно протестувати за метриками точності і повноти, описаними в першом розділі. Тоді можна буде знайти значення F-міри для трьох алгоритмів, і визначити найбільш ефективну реалізацію. F-міра обчислюється за формулою:

$$F = 2\frac{Precision \times Recall}{Precision + Recall}$$

В таблиці 3.3 подані обраховані показники даних метрик.

Таблиця 3.3 – Метрики оцінювання ефективності алгоритмів класифікації

	к найближчих сусідів	SVM	Naive Bayes
Precision	0,75	0,83	0,78

Продовження таблиці 3.3

Recall	0,5	0,55	0,57
F-міра	0,6	0,66	0,65

В результаті даного порівняння визначено, що найвищою за значенням метрикою F-міри є метрика методу опорних векторів або ж SVM, таким чином, даний алгоритм є найточнішим.

4 ЕКОНОМІЧНА ЧАСТИНА

Економічна частина є завершальним розділом магістерської роботи, в якому розробляються остаточні висновки щодо економічної ефективності запропонованої розробки. В даному розділі розглянемо основні питання конкурентоспроможності продукту та комерційного потенціалу розробки.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності, тобто під час виконання магістерської кваліфікаційної роботи.

Економічний аналіз стосується оцінки витрат і вигод для перевірки життєздатності проекту, інвестиційної можливості, події чи будь-якого іншого питання. Іншими словами, це включає визначення, оцінку та порівняння витрат і вигод. Крім того, існує багато інших важливих понять.

Процес аналізу сприяє оптимальному розподілу та використанню ресурсів, утворюючи важливий елемент у процесі прийняття рішень. Наприклад, мікроекономічний аналіз намагається описати, як люди та організації функціонують у певній економіці, макроекономічний аналіз зосереджується на ВВП, безробітті та інфляції, а техноекономічний аналіз (ТЕА) передбачає вивчення економічної ефективності промислового процесу.

Економічний аналіз визначає оцінку сценарію витрат і вигод проекту, події або дії. Це допомагає організаціям зрозуміти альтернативну вартість.

Існують різні інструменти економічного аналізу, які використовують економісти та власники бізнесу, щоб визначити відповідність плану чи вибору.

Підприємства здійснюють будь-який проект або фінансову діяльність лише

після застосування економічного аналізу для обмеження потенційних небезпек. Економічні змінні, нахили, оптимізація та лінійне програмування – це деякі інструменти або концепції, які використовуються.

Це також сприяє поясненню економічного зростання країни та того, як бізнес працює та засновується в ній.

Економічний аналіз оцінює проекти, сценарії, завдання, теми або дії, щоб зрозуміти їх прибутковість або негативні наслідки. Він демонструє взаємозв'язок із дослідженням визначення альтернативної вартості будь-якого проекту чи завдання. У бізнесі керівництво використовує його в різних сценаріях. Наприклад, компанії застосовують його під час ідентифікації нового продукту або процесу розширення чи інтеграції. Процес аналізу проходить через плюси і мінуси та розуміє суть.

Інструменти, які використовуються для цілей аналізу, демонструють елементи та методи статистики та основну математику для економічного аналізу. Крім того, процес включає різні інструменти та базується на багатьох припущеннях.

Давайте розглянемо важливі інструменти, задіяні в аналізі:

Економічні змінні: це найпоширеніший інструмент, за допомогою якого змінні визначають результат процесу, і зазвичай використовуються чотири типи змінних – залежні, незалежні, ендогенні та екзогенні змінні. Значення цих змінних дає важливу інформацію. Приклади включають ВВП, інфляцію, економічне зростання, процентні ставки тощо.

Нахили: нахили та графіки є іншими інструментами аналізу, оскільки вони відображають зміни в залежних змінних, коли відбувається зміна в незалежних змінних. Нахил відображає зміну та складається на основі спільного впливу як залежних, так і незалежних змінних.

Оптимізація: використовується для прийняття управлінських рішень і допомагає визначити рівні виробництва, вихідні витрати та шляхи максимізації прибутку. Це робиться шляхом вивчення зміни залежної змінної та розгляду записів і тенденцій, які можна знову використати для прогнозування майбутніх поворотів ринку.

Лінійне програмування. Застосування лінійного програмування може забезпечити чисельне розв'язання різноманітних бізнес-задач. Застосування задач

лінійного програмування або оптимізації допоможе знайти найкраще рішення з усіх можливих.

Для проведення комерційного і технологічного аудиту залучимо 3-х незалежних експертів. У нашому випадку такими експертами будуть провідні викладачі випускової та споріднених кафедр.

Результати оцінювання комерційного потенціалу розробки заносимо до таблиці 4.1.

Таблиця 4.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Технічна здійсненність концепції	3	3	4
Ринкові переваги (наявність аналогів)	4	3	3
Ринкові переваги (ціна продукту)	3	4	3
Ринкові переваги (технічні властивості)	4	3	4
Ринкові переваги (експлуатаційні витрати)	3	4	4
Ринкові перспективи (розмір ринку)	3	4	3
Ринкові перспективи (конкуренція)	3	3	3
Практична здійсненність (наявність фахівців)	4	3	3

Практична здійсненність (наявність фінансів)	3	4	3
Практична здійсненність (необхідність нових матеріалів)	3	3	4
Продовження таблиці 4.1			
Практична здійсненність (термін реалізації)	3	3	3
Практична здійсненність (розробка документів)	4	4	4
Сума балів	40	40	40
Середньоарифметична сума балів $\overline{СБ}$	40		

За даними таблиці 4.1 робимо висновок щодо рівня комерційного потенціалу розробки. При цьому користуємося рекомендаціями, наведеними в таблиці 4.2.

Таблиця 4.2 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 50	Високий

Оскільки середньоарифметична сума балів складає 40, то рівень комерційного

потенціалу розробки вище середнього, тому дана розробка є реальною для подальшої її реалізації та впровадження.

Можливі декілька шляхів реалізації розробки.

4.2 Розрахунок витрат на здійснення науково-дослідної роботи

Що стосується управління витратами на IT, існуючим рішенням є повернення коштів, яке вимірює та контролює витрати, понесені в IT-діяльності. Іншими словами, системи повернення коштів мають на меті розподілити IT-витрати між бізнес-підрозділами, які використовують IT-послуги (Friedman and Grayson, 1996). Виникнення повернення коштів IT сягає того часу, коли компанія придбала мейнфрейми, які використовувалися всіма відділами організації. Повернення платежів — це метод, за допомогою якого IT-сфера стягувала б інші підрозділи організації за використання цього ресурсу. Водночас, щоб запровадити відкликання платежів, потрібно було зібрати низку показників. Однак наявні на той час інструменти були дуже дорогими та не мали необхідного рівня автоматизації; таким чином, повернення платежів втратило прихильність через високі витрати на впровадження та незначну вигоду від його використання (Drury, 1997).

З часом висока конкуренція та постійний розвиток IT призвели до покращення його ресурсів та процесів. Ці вдосконалення були спрямовані на раціоналізацію витрат і підвищення ефективності, що змусило IT-ринок зосередити свої зусилля на оптимізації використання доступних ресурсів. Ці зусилля призвели до спільного використання серверів, які до того часу використовувалися виключно відділами/бізнесами, на додаток до використання віртуальних серверів (Dukaric and Juric, 2013). Використання послуг віртуалізації дозволяє швидше розробляти рішення та призводить до скорочення витрат завдяки більш ефективному використанню ресурсів. Хоча дуже важко розподілити витрати у віртуальній інфраструктурі, віртуальні машини також повинні мати свої витрати, щоб довести, що вони дешевші, ніж фізичні машини; це нове середовище представляє другий шанс

для IT-повернення (Baars et al., 2014).

Незважаючи на розуміння важливості повернення платежів, інструменти підтримки повернення платежів досягають зрілості лише зараз. Герлах та ін. Дослідження (2002) показує, що найефективніші методи повернення платежів – це ті, які легко зрозуміти внутрішнім клієнтам. Згідно з Agarwala та ін. (2008), IT-методи повернення платежів є надто складними або спеціальними та не досягають мети, заради якої було реалізовано повернення коштів. Таким чином, труднощі впровадження залишаються перешкодою для повернення коштів (Cummings, 2009). Крім того, література не демонструє структурованої системи для її впровадження, чітко пов'язаної з літературою з управління витратами.

При аналізі літератури з управління витратами основною розробкою в системах управління витратами була калькуляція витрат на основі діяльності (ABC) (Купер і Каплан, 1988). З моменту появи ABC було адаптовано, створюючи деякі його варіації: калькуляцію витрат на основі діяльності (TDABC) (Kaplan and Anderson, 2004), зусилля виробничого підрозділу (UEP) (Filomena та ін., 2011) і Cost Центри (CC) (Vogl, 2014). ABC широко застосовується у великих, середніх і малих компаніях у багатьох різних середовищах: логістика (Baykasoglu та Kaplanoglu, 2008), газова промисловість (Langmaak та ін., 2013), витрати на розробку (Qian та Ben-Arieh, 2008; Filomena та ін., 2009), моделювання витрат (Pan та ін., 2015), охорона здоров'я (Vanberkel та Moayed, 2016), виробництво (Suthummanon та ін., 2011; Nachtmann та Al-Rifai, 2004) та будівельні проекти (Di Петоріо і Соарес, 2013). Навіть невизначеність була включена в деякі дослідження ABC (Nachtmann and Needy, 2001, 2003).

Незважаючи на популярність методу ABC, широко відомо, що його важко реалізувати, що відкриває основу для TDABC (Everaert та ін., 2008). TDABC розглядається як метод, який набагато легше реалізувати порівняно з ABC (Hoozée та Bruggeman, 2010) і вже застосовувався в багатьох різних середовищах: спільні служби (Becker et al., 2009; Stouthuysen et al., 2010; Siguenza). -Guzman та ін., 2016), охорона здоров'я (Demmere та ін., 2009; Kaplan та ін., 2014; Yun та ін., 2015), фармацевтичні послуги (Gregório та ін., 2016), ланцюг постачання (Schulze та ін. ., 2012), інфраструктурні проекти (Yang et al., 2016). Зовсім недавно цей метод також

використовувався в хмарних обчисленнях (Adeoti and Valeverde, 2014; Baars et al., 2014).

Незважаючи на переваги, пов'язані з поверненням коштів, поточна література не представляє систематизованої структури, яка б поєднувала характеристики та особливості сфери IT з літературою з управління витратами. Таким чином, це фокус нашого дослідження: пропозиція системи управління витратами для IT. Запропонована система використовує концепції з різних методологій калькуляції та представляє гібридне рішення, тобто рішення, яке використовує концепції з різних методів для систематизації адекватного рішення. Ми використовуємо не лише ABC, але й TDABC залежно від кожного об'єкта витрат. Ми також надаємо зведення результатів, отриманих у деяких бразильських програмах.

Що стосується організації та структури, IT загалом поділяються на 2 операційні сфери: розробка та виробництво. Виробництво зазвичай підрозділяється на центр підтримки та центр обробки даних. IT-організація може мати або обидві сфери, або лише одну з них, залежно від її спрямованості. Нові специфічні програми для кожного бізнесу розробляються в області розвитку. За даними Bilal et al. (2014), Центр обробки даних — це структура мережевих ресурсів, яка використовує комунікаційну інфраструктуру для зберігання даних і розміщення програм. Відстеження доступності Центру обробки даних здійснюється в зоні підтримки. Парадигма управління IT знаходиться під центром обробки даних (Bilal et al., 2014).

Загалом ЦОД працюють із надлишковою потужністю, що можна пояснити різними технічними причинами. Однак незаперечним є той факт, що така надмірна потужність призводить до вищих витрат. Згідно з McKinsey (2014), багато IT-менеджерів впроваджують плани скорочення витрат. Архітектура центру обробки даних безпосередньо впливає на його витрати (Hammadi and Mhamdi, 2014).

Технічно традиційна структура центру обробки даних складається з 3 рівнів: доступу, агрегації та ядра (Hammadi and Mhamdi, 2014). Рівень доступу складається зі структури стійок і комутаторів. Комутатори рівня доступу підключаються до інших комутаторів рівня агрегації, які об'єднують кластери серверів. Цей шар називається шаром агрегації. Третій і останній рівень, базовий рівень, відповідає за зв'язок між центром обробки даних і зовнішніми користувачами.

У літературі вже описані нові типи організацій центрів обробки даних, які відрізняються від традиційної моделі (Hammadi and Mhamdi, 2014). Однак ці нові типи мають однакові елементи традиційної організації та відрізняються лише способом з'єднання цих елементів. Це не залежить від архітектури та ваги кожної складової вартості, значна частина витрат прямо чи опосередковано пов'язана з серверами. Пропонована система управління витратами на ІТ представлена в наступному розділі.

Операційні витрати ІТ поділяються на 2 складові: поточні витрати та основні засоби. Цей поділ випливає з класифікації об'єктів основних засобів під час їх придбання. У багатьох випадках ця класифікація не дотримується, що не забороняє використання системи, хоча апостеріорна класифікація є дуже складною, оскільки вона включає різні елементи, які важко ідентифікувати. Крім того, ці витрати мають високу вартість і можуть мати значний вплив на результати; таким чином, до них потрібно ставитися по-різному, щоб досягти більш точного контролю цього конкретного елемента.

Поточні витрати – це витрати, пов'язані з роботою, структурою та функціонуванням ІТ. Ці витрати зазвичай розподіляються в структурі центрів витрат компаній і фактично оплачуються в аналізованому періоді. Однак витрати на основні засоби зазвичай консолідуються в одному центрі витрат. Ця консолідація відбувається через складність класифікації цих елементів і через велику кількість елементів. Витрати пов'язані з іммобілізацією ІТ-активів і не витрачаються ефективно в періоді; тим не менш, вони повинні бути розглянуті для цілей управління. Таким чином, ІТ-витрати складаються із суми 2 компонентів, як показано в рівнянні (4.1).

$$C_{total} = C_{depreciation} + C_{current} \quad (4.1)$$

В ідеалі розрахунок вартості повинен виконуватися для кожного елемента ЦОД. Однак через високу складність і великий обсяг інформації, яка буде необхідною, необхідно зрозуміти, як ці елементи пов'язані один з одним, щоб виконати обчислення для груп елементів. Таким чином, передбачається, що сервери є найбільшими статтями витрат у Центрі обробки даних, а інші статті є частиною

структури, яка підтримує роботу серверів. З цієї причини методологія консолідує витрати на різні елементи та розглядає їх разом із витратами на сервери. Тому необхідно ідентифікувати всі сервери ЦОД і класифікувати їх за двома критеріями: середовищем і об’єктами, які їх використовують.

Класифікація середовищ може бути необхідною, а може і не бути необхідною. Поділ середовищ вказується, щоб визначити відмінності як у його структуруванні, так і в типі обробки, яку вони виконують. Ця класифікація спрямована на полегшення управління різними середовищами. Що стосується об'єктів, які використовують кожен сервер, вони повинні бути узгоджені з цілями системи управління витратами.

4.2.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.2)$$

де M_{ni} – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_p – середня кількість робочих днів в місяці, $T_p \approx 21 \ldots 23$ дні;

t_i – кількість днів роботи конкретного дослідника.

Дану розробку буде проводити інженер, величина окладу буде становити 11000 грн. на місяць. Кількість робочих днів у місяці складає 22, а кількість робочих днів дослідника складає 60. Зведемо сумарні розрахунки до таблиця 4.3.

Таблиця 4.3 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на заробітну плату, грн
Керівник проекту	15000	680	10	6800
Інженер	12000	545	60	32700
Всього				39500

Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ розраховується як 10-12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.3)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати.

На даному підприємстві додаткова заробітна плата нараховується в розмірі 10% від основної заробітної плати.

$$З_д = 0{,}10 \cdot 395000 = 39500(грн.)$$

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою :

$$З_{дод} = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.4)$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{зп} = (39500 + 3950) \cdot \frac{22}{100} = 9559(грн.)$$

Отже, нарахування на заробітну плату складають 9559 грн.

4.2.3 Розрахунок витрат на комплектуючі

Виробничі витрати охоплюють усі прямі та непрямі витрати, з якими стикаються підприємства у зв'язку з виробництвом продукції чи наданням послуг. Виробничі витрати можуть включати різноманітні витрати, такі як оплата праці, сировина, витратні виробничі матеріали та загальні накладні витрати.

Виробничі витрати відносяться до витрат, які компанія несе на виробництво продукту або надання послуги, що генерує дохід для компанії.

Виробничі витрати можуть включати різноманітні витрати, такі як оплата праці, сировина, витратні виробничі матеріали та загальні накладні витрати.

Загальні витрати на продукт можна визначити шляхом додавання загальних прямих витрат на матеріали та робочу силу, а також загальних накладних витрат на виробництво.

Виробничі витрати, які також називаються витратами на продукцію, несе підприємство, коли воно виробляє продукт або надає послугу. Ці витрати включають різноманітні витрати. Наприклад, виробники мають виробничі витрати, пов'язані із сировиною та робочою силою, необхідною для створення продукту. Сфери послуг

несуть виробничі витрати, пов'язані з робочою силою, необхідною для надання послуги, і будь-якими витратами на матеріали, задіяні в наданні послуги.

Податки, що стягуються урядом, або роялті, які вимагають компанії з видобутку природних ресурсів, також розглядаються як виробничі витрати.

Після того, як продукт буде готовий, компанія записує вартість продукту як актив у своїй фінансовій звітності, поки продукт не буде продано. Запис готової продукції як активу служить для виконання вимог компанії щодо звітності та інформування акціонерів.

Щоб вважатися витратами на виробництво, витрати мають бути безпосередньо пов'язані з отриманням прибутку для компанії.

Загальні витрати на продукт можна визначити шляхом додавання загальних прямих витрат на матеріали та робочу силу, а також загальних накладних витрат на виробництво.

Такі дані, як собівартість виробництва на одиницю, можуть допомогти підприємству встановити відповідну ціну продажу готового продукту.

Щоб отримати собівартість продукції на одиницю, виробничі витрати діляться на кількість одиниць, виготовлених за період, охоплений цими витратами. Для беззбитковості ціна продажу повинна покривати собівартість одиниці. Ціни, які вищі за собівартість одиниці, приносять прибуток, тоді як ціни, нижчі за собівартість за одиницю, призводять до збитків.

Виробництво несе як постійні, так і змінні витрати. Наприклад, постійні витрати на виробництво автомобіля включатимуть обладнання, а також зарплату робітникам. У міру зростання темпів виробництва постійні витрати залишаються постійними.

Змінні витрати збільшуються або зменшуються в міру зміни обсягу виробництва. Витрати на комунальні послуги є яскравим прикладом змінних витрат, оскільки, як правило, потрібно більше енергії, коли виробництво збільшується.

Граничні витрати на виробництво стосуються загальних витрат на виробництво однієї додаткової одиниці. В економічній теорії фірма продовжуватиме розширювати виробництво товару, поки її граничні витрати виробництва не дорівнюватимуть її граничному продукту (граничному доходу). Це, у свою чергу,

матиме тенденцію дорівнювати його продажу.

Витрати на комплектуючі, які використовують при дослідженні нового технічного рішення, розраховуються, згідно з їхньою номенклатурою за формулою:

$$K_{\text{в}} = \sum_{j=1}^{n} H_j \cdot \text{Ц}_j \cdot K_j, \quad (4.5)$$

де H_j – кількість комплектуючих j-го виду, шт.;

Ц_j – покупна ціна комплектуючих j-го виду, грн;

Kj - коефіцієнт транспортних витрат, (1,1...1,15);

Проведені розрахунки зводимо до таблиці 4.4.

Таблиця 4.4 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, кг	Ціна за штуку, грн.	Разом
Папір	150	1	150
Ручка	10	1	10
Флешка	300	1	300
Всього			504

4.2.4 Амортизація обладнання, програмних засобів та приміщення

Перш ніж ми заглибимося в тонкощі амортизації програмного забезпечення, важливо мати загальне розуміння того, що таке амортизація та як вона працює.

Амортизація — це природний знос будівлі та її активів з часом. Власники нерухомості, що приносить дохід, і підприємства можуть вимагати амортизації як податкову знижку щороку.

Власники бізнесу можуть вимагати амортизації програмного забезпечення для комп'ютерного програмного забезпечення, яке вони використовують у своїй діяльності. Це тому, що програмне забезпечення потрібне їм для отримання доходу. Амортизація програмного забезпечення працює зовсім по-іншому в порівнянні з фізичними елементами, як-от обладнання та арматура.

Програмне забезпечення n-house можна визначити як дві речі. Це або комп'ютерне програмне забезпечення, або право на використання програмного забезпечення, придбаного та розробленого для бізнесу.

Наприклад, якщо компанія розробила програмне забезпечення для зберігання вакансій і інформації про клієнтів, це буде класифікуватися як власне програмне забезпечення.

Будь-які витрати, пов'язані з розробкою власного програмного забезпечення, можна помістити в пул розробки програмного забезпечення. Витрати на передвстановлення також включені, наприклад гонорари консультанта з програмного забезпечення або юридичні витрати.

Коли компанія використовує пул розробки програмного забезпечення, усі майбутні власні витрати на програмне забезпечення повинні амортизуватися за допомогою пулу. Для витрат кожного року створюється новий пул. Звичайні власні витрати на майбутню розробку програмного забезпечення включають оновлення або покращення безпеки.

У спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.6)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувалиєь для проведення досліджень, грн.;

$t_{вик}$ – термін корисного використання обладнання, програмних засобів, приміщень під час досліджень, місяців.

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень

тощо, років.

Проведені розрахунки зводимо до таблиці 4.5.

Таблиця 4.5 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн.
Комп'ютер	15000	5	5	1500
Всього				1500

4.2.5 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію розраховують за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} t_i \cdot Ц_e \cdot К_{впі}}{\eta_i}, \quad (4.7)$$

де W_{yi} – встановленя потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності;

η_i – коефіцієнт корисної дії обладнання.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Комп’ютер	0,5	75	135
Освітлення приміщення	0,6	52	112
Всього			247

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В = З_о + З_д + З_н + К + А_{обл} + В_е, \quad (4.8)$$

$$В = 39500 + 3950 + 9559 + 504 + 1500 + 247 = 55260 \text{ (грн)}$$

Загальні витрати на завершення науково-дослідної роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (4.9)$$

Загальні витрати складають

$$ЗВ = \frac{455260}{0,9} = 61400 \text{ (грн.)}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Розрахуємо можливе збільшення чистого прибутку у потенційного інвестора для

кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки за формулою:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right), \quad (4.10)$$

де $\pm \Delta Ц_0$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної діяльності;

$Ц_0$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги), рекомендується приймати 0,2...0,5;

ϑ – ставка податку на прибуток.

В середньому в рік продається 300 розробок. Середня вартість такої розробки становить 1000 грн.

Впровадження зразка розробки дозволяє збільшити ціну кожного зразка на 200 грн, враховуючи ціни конкурентів. Також прогнозується, що попит на даний продукт зросте, оскільки даний продукт відрізняється якістю від конкурентних.

Попит збільшиться за перший рік на 250 примірників, за наступний на 200 та протягом третього року – ще на 150 примірників.

Ставка податку на додану вартість в 2021 році залишилась на рівні 20%, а коефіцієнт λ=0,8333. Ставка податку на прибуток складає 18%.

Коефіцієнт, який враховує рентабельність продукту, дорівнює 0,3.

Отже, розрахуємо збільшення чистого прибутку підприємства на 2022 -2024 рр.:

$$\Delta П_{2022} = (1000 \cdot 300 + (1000 + 200) \cdot 250) \cdot 0{,}8333 \cdot 0{,}3 \cdot \left(1 - \frac{18}{100}\right) = 122995{,}08 \text{ (грн.)}$$

$$\Delta П_{2023} = (1000 \cdot 300 + (1000 + 200) \cdot (250 + 200)) \cdot 0{,}8333 \cdot 0{,}3 \cdot \left(1 - \frac{18}{100}\right) = 172193{,}11 \text{ (грн.)}$$

$$\Delta П_{2024} = (1000 \cdot 300 + (1000 + 200) \cdot (250 + 200 + 150)) \cdot 0{,}8333 \cdot 0{,}3 \cdot \left(1 - \frac{18}{100}\right) = 209091{,}63 \text{ (грн.)}$$

Далі розрахуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{122995{,}08}{(1+0{,}1)^2} + \frac{172193{,}11}{(1+0{,}1)^3} + \frac{209091{,}63}{(1+0{,}1)^4} = 373832{,}44 \text{ (грн.)}$$

Далі розрахуємо величину початкових інвестицій, які потенційний інвестор має

вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{\text{інв}} \cdot ЗВ, \quad (4.12)$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науко-во-технічної розробки та її комерціалізацію.

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \cdot 54555,91 = 109111,82 \text{ (грн)}$$

Тоді абсолютний економічний ефект або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{\text{абс}} = ПП - PV, \quad (4.13)$$

де *ПП* – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$Е_{\text{абс}} = (373832,44 - 109111,82) = 264720,62 \text{ (грн.)}$$

Внутрішня економічна дохідність інвестицій, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$Е_{\text{в}} = \sqrt[T_{\text{ж}}]{1 + \frac{Е_{\text{абс}}}{PV}} - 1, \quad (4.14)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[3]{1 + \frac{264720,62}{109111,82}} - 1 = 0,50 = 50\%$$

Далі визначимо бар'єрну ставку дисконтування, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть.

Мінімальна внутрішня економічна дохідність вкладених інвестицій визначається за формулою:

$$\tau_{min} = d + f, \quad (4.15)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках;

f – показник, що характеризує ризикованість вкладення інвестицій.

$$\tau = 0,12 + 0,05 = 0,17$$

Далі розрахуємо період окупності інвестицій, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (4.16)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{0,50} = 2 \text{ роки}$$

Термін окупності складає 2 роки, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

ВИСНОВКИ

При виконанні магістерської роботи поставлена задача реалізації та порівняння найвідоміших алгоритмів класифікації.

Під час виконання поставленої задачі перевірено актуальність обраної теми, проаналізовано предметну область системи, об'єкт проектування, перелічено існуючі системи-аналоги, їх функціонал та призначення.

Визначено предмет, об'єкт дослідження, сформовано мету дослідження.

Описано основні особливості алгоритмів класифікації, їх застосування на практиці та метрики оцінювання. Проаналізовано та описано вимоги до програмного забезпечення, сформовано перелік основних функцій системи. Розроблено структурні схеми комп'ютерних реалізацій алгоритмів.

В результаті виконання даної роботи проведено аналіз засобів для реалізації системи. Розглянуто мови програмування. Для реалізації поставлених вимог було обрано мову C#.

Після вибору усіх компонентів та їх реалізації, системи було протестовано та виявлено, що метод опорних векторів працює найбільш ефективно.

Економічна частина перевірила та підтвердила користь розробки даної інформаційної технології, термін окупності становитиме 0,62 роки. Збільшення прибутку протягом третього року знаходження додатку на ринку становитиме 1260000 грн, що є досить гарним показником і доводить важливість впровадження розробки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adams J. B. A probability model of medical reasoning and the MYCIN model./ Adams J. B. – Mathematical Biosciences, 32. 1976. – 177-186 p.
2. Blackboard: Teaching and Learning [Електронний ресурс]. – Режим доступу : http://www.blackboard.com/International/EMEA.aspx?lang=en-us.
3. Exploring WebCT Campus Edition 6.0. Designer and Instructor Tutorials [Електронний ресурс]. – Режим доступу : http://tutorials.webct.com/exploring/.
4. The official web-site of Moodle LMS [Електронний ресурс]. – Режим доступу : http://moodle.org/
5. Довбиш А. С. Основи проектування інтелектуальних систем [Электронныйресурс].URL:https://essuir.sumdu.edu.ua/handle/123456789/1407 .
6. Boose J. H. Expertise Transfer for Expert System Design. / Boose J. H. –New York: Elsevier. 1986. – 23 p.
7. Boose J. H. Knowledge Acquisition Tools for Expert Systems. / Boose J. H., Gaines B. – New York: Academic Press. 1988. – 7-65 p.
8. Метод К-найближчих сусідів:. https://studopedia.com.ua/1_42785_metod-K-nayblizhchih-susIdIv.html (дата звернення: 03.09.2022).
9. Kononenko W. Machine learning for medical diagnosis: history, state of the art and perspective. / Kononenko W. // Artificial Intelligence in Medicine. Ljubjana. 2001. – 89-109 p. 10. Lipmann R.P. “Pattern classification using neural networks” / Lipmann R.P. – IEEE Communications Magazine, 1989. – 47-67 p.
10. NET FRAMEWORK URL:. https://learn.microsoft.com/ru-ru/dotnet/framework/get-started/overview (дата звернення: 05.10.2022).
11. Методи класифікації персональних комп'ютерів:. https://sites.google.com/site/strukturatakomputera/metodi-klasifikaciie-personalnih-komp-uteriv (дата звернення: 07.10.2022).
12. База знаний Allbest. https://knowledge.allbest.ru/programming/3c0b65635a3bd78b4c53a88521306c27_0.html (дата звернення: 8.10.2022).

13. Bayes_naive_classifier - GitHub:. https://github.com/ponjoru/bayes_naive_classifier (дата звернення: 5.10.2022).

14. Шпаргалка по Git от GitHub:. https://training.github.com/downloads/ru/github-git-cheat-sheet/ (дата звернення: 12.10.2022).

15. Buchanan B. G. Rule-Based Expert Systems / Buchanan B. G. and Shortliffe E. H. – Reading MA: Addison-Wesley. 1984– 1-3 p.

Додаток А Лістинг програм

К сусідів

```
using System;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;

namespace knn
{
    public partial class KNN : Form
    {
        string fname = ""; // Filename to use.
        string fdir = @"c:\test\"; // Initial directory in which to look for files.
        string ffilter = "All files (*.*)|*.*|CSV files (*.csv)|*.csv"; // File filter.
        Point clickPosition;
        DataTable dataTable = new DataTable();

        public KNN()
        {
            InitializeComponent();
        }

        private void kField_Validating(object sender, CancelEventArgs e)
        {
            try { int k = Convert.ToInt16(kField.Text); }
            catch (Exception ex) { e.Cancel = true; MessageBox.Show("k must be an integer
value"); }
        }

        private void importButton_Click(object sender, EventArgs e)
        {

            // Request a file.
            OpenFileDialog fdlg = new OpenFileDialog();
            fdlg.Title = "Import File";
            fdlg.InitialDirectory = fdir;
            fdlg.Filter = ffilter;
            fdlg.FilterIndex = 2;
```

```
fdlg.RestoreDirectory = true;
if (fdlg.ShowDialog() == DialogResult.OK)
{
  this.fname = fdlg.FileName;
}
fdlg.Dispose();

// Clear series.
chart1.Series.Clear();

// Set up chart properties.
chart1.ChartAreas[0].AxisX.LabelStyle.Format = "0";
chart1.ChartAreas[0].AxisX.RoundAxisValues();

// Set up series 1.
var series1 = new System.Windows.Forms.DataVisualization.Charting.Series
{
  Name = "Original Data",
  Color = System.Drawing.Color.Green,
  IsVisibleInLegend = false,
  IsXValueIndexed = true,
  ChartType = SeriesChartType.Point
};
this.chart1.Series.Add(series1);

// Set up data grid.
dataTable.Rows.Clear();
if (dataTable.Columns.Count == 0)
{
  dataTable.Columns.Add("x");
  dataTable.Columns.Add("y");
  dataTable.Columns.Add("Category");
}

StreamReader streamReader = new StreamReader(this.fname);
string[] dataLine = new string[File.ReadAllLines(this.fname).Length];
double x, y;
string category;

// Read first line.
dataLine = streamReader.ReadLine().Split(',');
x = Convert.ToDouble(dataLine[0]);
y = Convert.ToDouble(dataLine[1]);
category = Convert.ToString(dataLine[2]);
dataTable.Rows.Add(x, y, category);

// Read rest of lines.
```

```
        while (!streamReader.EndOfStream)
        {
            dataLine = streamReader.ReadLine().Split(',');
            x = Convert.ToDouble(dataLine[0]);
            y = Convert.ToDouble(dataLine[1]);
            category = Convert.ToString(dataLine[2]);
            dataTable.Rows.Add(x, y, category);
        }

        // Close file
        streamReader.Close();

        // Display data grid.
        dataGridView1.DataSource = dataTable;

        // Display chart.
        refreshChart(sender, e);
    }

    private void refreshChart(object sender, EventArgs e)
    {
        double x, y;
        string category;

        DataTable xy = new DataTable();
        xy = (DataTable)this.dataGridView1.DataSource;

        // Reset original data.
        chart1.Series["Original Data"].Points.Clear();
        chart1.Series["Original Data"].IsVisibleInLegend = false;
        chart1.Series["Original Data"].IsXValueIndexed = false;
        chart1.Series["Original Data"].MarkerSize = 20;
        chart1.Series["Original Data"].MarkerStyle = MarkerStyle.Circle;

        // Refresh from data grid.
        for (int rows = 0; rows < dataGridView1.Rows.Count - 1; rows++)
        {
            var xcell = dataGridView1.Rows[rows].Cells[0].Value;
            var ycell = dataGridView1.Rows[rows].Cells[1].Value;
            var catcell = dataGridView1.Rows[rows].Cells[2].Value;

            if (xcell != DBNull.Value && ycell != DBNull.Value && catcell !=
DBNull.Value)
            {
                // Read next data pair.
                x = Convert.ToDouble(xcell);
```

```
            y = Convert.ToDouble(ycell);
            category = Convert.ToString(catcell);

            //Add to chart.
            chart1.Series["Original Data"].Points.AddXY(x, y);
            chart1.Series["Original Data"].Points[rows].Color = catColor(category);
        }
        chart1.Invalidate();
    }
}

private void exportButton_Click(object sender, EventArgs e)
{
    // Find file to write to.
    OpenFileDialog fdlg = new OpenFileDialog();
    fdlg.Title = "Export File";
    fdlg.InitialDirectory = fdir;
    fdlg.Filter = ffilter;
    fdlg.FilterIndex = 2;
    fdlg.RestoreDirectory = true;
    if (fdlg.ShowDialog() == DialogResult.OK)
    {
        this.fname = fdlg.FileName;
    }
    fdlg.Dispose();

    // Create the CSV file to which grid data will be exported.
    StreamWriter sw = new StreamWriter(this.fname, false);

    // Loop through all the rows.
    foreach (DataGridViewRow dr in dataGridView1.Rows)
    {
        if (!dr.IsNewRow)
        {
            List<string> columnData = new List<string>();
            foreach (DataGridViewCell cell in dr.Cells)
            {
                columnData.Add(cell.Value.ToString());
            }
            sw.WriteLine(string.Join(",", columnData.ToArray()));
        }
    }

    sw.Close();
}
```

```
    private void addNewData_CheckedChanged(object sender, EventArgs e)
    {
        if (addNewData.Checked)
        {
            categoryLabel.Visible = true;
            categorySelector.Visible = true;
        }
    }

    private void guessCategory_CheckedChanged(object sender, EventArgs e)
    {

        if (guessCategory.Checked)
        {
            categoryLabel.Visible = false;
            categorySelector.Visible = false;
        }
    }

    private void chart1_MouseClick(object sender, MouseEventArgs e)
    {

        var pos = e.Location;
        clickPosition = pos;
        var results = chart1.HitTest(pos.X, pos.Y, false, ChartElementType.PlottingArea);
        foreach (var result in results)
        {
            if (result.ChartElementType == ChartElementType.PlottingArea)
            {
                double xVal =
Math.Round(result.ChartArea.AxisX.PixelPositionToValue(pos.X), 1);
                double yVal =
Math.Round(result.ChartArea.AxisY.PixelPositionToValue(pos.Y), 1);
                if (addNewData.Checked)
                {
                    this.dataTable.Rows.Add(xVal, yVal, categorySelector.Text);
                    refreshChart(sender, e);
                }
                else
                {
                    int k = Convert.ToInt16(kField.Text);
                    bool weighted = weightedBox.Checked;
                    DataTable dt = new DataTable();
                    dt = (DataTable)this.dataGridView1.DataSource;
                    string categoryGuess = KNN_calculator.guess(dt, xVal, yVal, k, weighted);
```

```
                //Add to chart.
                chart1.Series["Original Data"].Points.AddXY(xVal, yVal);
                int point_count = chart1.Series["Original Data"].Points.Count - 1;
                chart1.Series["Original Data"].Points[point_count].MarkerStyle =
MarkerStyle.Square;
                chart1.Series["Original Data"].Points[point_count].MarkerSize = 10;
                if (categoryGuess.Length > 1) { chart1.Series["Original
Data"].Points[point_count].Color = Color.Gray; }
                else { chart1.Series["Original Data"].Points[point_count].Color =
catColor(categoryGuess); };
                chart1.Invalidate();
            }
        }
    }

    private System.Drawing.Color catColor(string category)
    {
        if (category == "A") { return Color.Green; }
        else if (category == "B") { return Color.Blue; }
        else if (category == "C") { return Color.Yellow; }
        else { return Color.Black; };
    }

    private void refreshChartButton_Click(object sender, EventArgs e)
    {
        refreshChart(sender, e);
    }

    private void drawZoneButton_Click(object sender, EventArgs e)
    {
        // Work out data boundaries.
        // (x1,y1) is bottom LH corner.
        // (x2,y2) is top RH corner.
        double x, y, x1=0, y1=0, x2=0, y2=0;

        for (int rows = 0; rows < dataGridView1.Rows.Count - 1; rows++)
        {
            var xcell = dataGridView1.Rows[rows].Cells[0].Value;
            var ycell = dataGridView1.Rows[rows].Cells[1].Value;
            var catcell = dataGridView1.Rows[rows].Cells[2].Value;

            if (xcell != DBNull.Value && ycell != DBNull.Value && catcell !=
DBNull.Value)
            {
                // Read next data pair.
```

```
            x = Convert.ToDouble(xcell);
            y = Convert.ToDouble(ycell);

            // Update boundaries.
            if (rows == 0)
            {
                x1 = x; y1 = y; x2 = x; y2 = y;
            }
            else
            {
                x1 = Math.Min(x1, x); y1 = Math.Min(y1, y); x2 = Math.Max(x2, x); y2 =
Math.Max(y2, y);
            }
        }
    }

    // Guess category at each point in zone.
    int k = Convert.ToInt16(kField.Text);
    bool weighted = weightedBox.Checked;
    double interval = Convert.ToDouble(intervalBox.Text);
    int marker_size = Convert.ToInt16(markerSizeBox.Text);
    DataTable dt = new DataTable();
    dt = (DataTable)this.dataGridView1.DataSource;
    for (double xVal = x1; xVal <= x2; xVal += interval)
    {
        for (double yVal = y1; yVal <= y2; yVal += interval)
        {
            // Guess category.
            string categoryGuess = KNN_calculator.guess(dt, xVal, yVal, k, weighted);

            // Add to chart.
            chart1.Series["Original Data"].Points.AddXY(xVal, yVal);
            int point_count = chart1.Series["Original Data"].Points.Count - 1;
            chart1.Series["Original Data"].Points[point_count].MarkerStyle =
MarkerStyle.Square;
            chart1.Series["Original Data"].Points[point_count].MarkerSize = marker_size;
            if (categoryGuess.Length > 1) { chart1.Series["Original
Data"].Points[point_count].Color = Color.Gray; }
            else { chart1.Series["Original Data"].Points[point_count].Color =
catColor(categoryGuess); };
        }
    }
    chart1.Invalidate();
  }
 }
}
```

SVM

```
using DeepLearnCS;
using System;

namespace SupportVectorMachine
{
    public enum ModelParameters
    {
        X = 0,
        Y = 1,
        ALPHA = 2,
        W = 3,
        B = 4
    };

    public class Model
    {
        public ManagedArray ModelX;
        public ManagedArray ModelY;
        public KernelType Type;
        public ManagedArray KernelParam;
        public ManagedArray Alpha;
        public ManagedArray W;
        public double B;
        public double C;
        public double Tolerance;
        public int Category;
        public int Passes;
        public int Iterations;
        public int MaxIterations;
        public bool Trained;

        Random random = new Random(Guid.NewGuid().GetHashCode());

        // Internal variables
        ManagedArray K;
        ManagedArray E;
        ManagedArray alpha;
        ManagedArray dx;
        ManagedArray dy;
        ManagedArray kparam;
        double b;
        double eta;
        double H;
```

```
        double L;
        KernelType ktype;

        public Model()
        {

        }

        public Model(ManagedArray x, ManagedArray y, KernelType type, ManagedArray kernelParam, ManagedArray alpha, double b, ManagedArray w, int passes)
        {
            ModelX = x;
            ModelY = y;
            Type = type;
            KernelParam = kernelParam;
            Alpha = alpha;
            B = b;
            W = w;
            Passes = passes;
            Trained = true;
        }

        int Rows(ManagedArray x)
        {
            return x.y;
        }

        int Cols(ManagedArray x)
        {
            return x.x;
        }

        public void Setup(ManagedArray x, ManagedArray y, double c, KernelType kernel, ManagedArray param, double tolerance = 0.001, int maxpasses = 5, int category = 1)
        {
            ManagedOps.Free(dx, dy);
            dx = new ManagedArray(x);
            dy = new ManagedArray(y);

            ManagedOps.Copy2D(dx, x, 0, 0);
            ManagedOps.Copy2D(dy, y, 0, 0);

            ktype = kernel;
            // Data parameters
            var m = Rows(dx);
```

```
Category = category;
MaxIterations = maxpasses;
Tolerance = tolerance;
C = c;

// Reset internal variables
ManagedOps.Free(K, kparam, E, alpha);

kparam = new ManagedArray(param);
ManagedOps.Copy2D(kparam, param, 0, 0);

// Variables
alpha = new ManagedArray(1, m);
E = new ManagedArray(1, m);
b = 0;
Iterations = 0;

// Pre-compute the Kernel Matrix since our dataset is small
// (In practice, optimized SVM packages that handle large datasets
// gracefully will *not* do this)
if (kernel == KernelType.LINEAR)
{
    // Computation for the Linear Kernel
    // This is equivalent to computing the kernel on every pair of examples
    var tinput = ManagedMatrix.Transpose(dx);

    K = ManagedMatrix.Multiply(dx, tinput);

    double slope = kparam.Length() > 0 ? kparam[0] : 1;
    double inter = kparam.Length() > 1 ? kparam[1] : 0;

    ManagedMatrix.Multiply(K, slope);
    ManagedMatrix.Add(K, inter);

    ManagedOps.Free(tinput);
}
else if (kernel == KernelType.GAUSSIAN || kernel == KernelType.RADIAL)
{
    // RBF Kernel
    // This is equivalent to computing the kernel on every pair of examples
    var pX2 = ManagedMatrix.Pow(dx, 2);
    var rX2 = ManagedMatrix.RowSums(pX2);
    var tX2 = ManagedMatrix.Transpose(rX2);
    var trX = ManagedMatrix.Transpose(dx);

    var tempK = new ManagedArray(m, m);
    var temp1 = new ManagedArray(m, m);
```

```
        var temp2 = ManagedMatrix.Multiply(dx, trX);

        ManagedMatrix.Expand(rX2, m, 1, tempK);
        ManagedMatrix.Expand(tX2, 1, m, temp1);
        ManagedMatrix.Multiply(temp2, -2);

        ManagedMatrix.Add(tempK, temp1);
        ManagedMatrix.Add(tempK, temp2);

        double sigma = kparam.Length() > 0 ? kparam[0] : 1;
        var g = Math.Abs(sigma) > 0 ? Math.Exp(-1 / (2 * sigma * sigma)) : 0;

        if (Type == KernelType.RADIAL)
            ManagedMatrix.Sqrt(tempK);

        K = ManagedMatrix.Pow(g, tempK);

        ManagedOps.Free(pX2, rX2, tX2, trX, tempK, temp1, temp2);
    }
    else
    {
        // Pre-compute the Kernel Matrix
        // The following can be slow due to the lack of vectorization
        K = new ManagedArray(m, m);

        var Xi = new ManagedArray(Cols(dx), 1);
        var Xj = new ManagedArray(Cols(dx), 1);

        for (var i = 0; i < m; i++)
        {
            ManagedOps.Copy2D(Xi, dx, 0, i);

            for (var j = 0; j < m; j++)
            {
                ManagedOps.Copy2D(Xj, dx, 0, j);

                K[j, i] = KernelFunction.Run(kernel, Xi, Xj, kparam);

                // the matrix is symmetric
                K[i, j] = K[j, i];
            }
        }
        ManagedOps.Free(Xi, Xj);
    }
```

```
        eta = 0;
        L = 0;
        H = 0;
        // Map 0 (or other categories) to -1
        for (var i = 0; i < Rows(dy); i++)
        {
            dy[i] = (int)dy[i] != Category ? -1 : 1;
        }
    }

    public bool Step()
    {
        if (Iterations >= MaxIterations)
            return true;

        // Data parameters
        var m = Rows(dy);

        var num_changed_alphas = 0;

        for (var i = 0; i < m; i++)
        {
            // Calculate Ei = f(x(i)) - y(i) using (2).
            E[i] = b;

            for (var yy = 0; yy < m; yy++)
            {
                E[i] += alpha[yy] * dy[yy] * K[yy, i];
            }
            E[i] -= dy[i];

            if ((dy[i] * E[i] < -Tolerance && alpha[i] < C) || (dy[i] * E[i] > Tolerance &&
alpha[i] > 0))
            {
                // In practice, there are many heuristics one can use to select
                // the i and j. In this simplified code, we select them randomly.
                var j = i;

                while (j == i)
                {
                    // Make sure i != j
                    j = (int)Math.Floor(m * random.NextDouble());
                }

                // Calculate Ej = f(x(j)) - y(j) using (2).
```

```
            E[j] = b;

            for (var yy = 0; yy < m; yy++)
            {
                E[j] += alpha[yy] * dy[yy] * K[yy, j];
            }

            E[j] -= dy[j];

            // Save old alphas
            var alpha_i_old = alpha[i];
            var alpha_j_old = alpha[j];

            // Compute L and H by (10) or (11).
            if ((int)dy[i] == (int)dy[j])
            {
                L = Math.Max(0, alpha[j] + alpha[i] - C);
                H = Math.Min(C, alpha[j] + alpha[i]);
            }
            else
            {
                L = Math.Max(0, alpha[j] - alpha[i]);
                H = Math.Min(C, C + alpha[j] - alpha[i]);
            }

            if (Math.Abs(L - H) <= Double.Epsilon)
            {
                // continue to next i
                continue;
            }

            // Compute eta by (14).
            eta = 2 * K[j, i] - K[i, i] - K[j, j];

            if (eta >= 0)
            {
                // continue to next i.
                continue;
            }

            // Compute and clip new value for alpha j using (12) and (15).
            alpha[j] = alpha[j] - (dy[j] * (E[i] - E[j])) / eta;

            // Clip
            alpha[j] = Math.Min(H, alpha[j]);
            alpha[j] = Math.Max(L, alpha[j]);
```

```
            // Check if change in alpha is significant
            if (Math.Abs(alpha[j] - alpha_j_old) < Tolerance)
            {
                // continue to next i.
                // replace anyway
                alpha[j] = alpha_j_old;

                continue;
            }

            // Determine value for alpha i using (16).
            alpha[i] = alpha[i] + dy[i] * dy[j] * (alpha_j_old - alpha[j]);

            // Compute b1 and b2 using (17) and (18) respectively.
            var b1 = b - E[i] - dy[i] * (alpha[i] - alpha_i_old) * K[j, i] - dy[j] * (alpha[j] -
alpha_j_old) * K[j, i];
            var b2 = b - E[j] - dy[i] * (alpha[i] - alpha_i_old) * K[j, i] - dy[j] * (alpha[j] -
alpha_j_old) * K[j, j];

            // Compute b by (19).
            if (0 < alpha[i] && alpha[i] < C)
            {
                b = b1;
            }
            else if (0 < alpha[j] && alpha[j] < C)
            {
                b = b2;
            }
            else
            {
                b = (b1 + b2) / 2;
            }

            num_changed_alphas++;
        }
    }

    if (num_changed_alphas == 0)
    {
        Iterations++;
    }
    else
    {
        Iterations = 0;
    }

    return Iterations >= MaxIterations;
```

```
    }
    public void Generate()
    {
        var m = Rows(dx);
        var n = Cols(dx);

        var idx = 0;

        for (var i = 0; i < m; i++)
        {
            if (Math.Abs(alpha[i]) > 0)
            {
                idx++;
            }
        }

        ManagedOps.Free(ModelX, ModelY, Alpha, W, KernelParam);

        ModelX = new ManagedArray(Cols(dx), idx);
        ModelY = new ManagedArray(1, idx);
        Alpha = new ManagedArray(1, idx);
        KernelParam = new ManagedArray(kparam);
        var ii = 0;

        for (var i = 0; i < m; i++)
        {
            if (Math.Abs(alpha[i]) > 0)
            {
                for (int j = 0; j < n; j++)
                {
                    ModelX[j, ii] = dx[j, i];
                }
                ModelY[ii] = dy[i];

                Alpha[ii] = alpha[i];

                ii++;
            }
        }

        B = b;
        Passes = Iterations;
        ManagedOps.Copy2D(KernelParam, kparam, 0, 0);
        Type = ktype;
```

```
        var axy = ManagedMatrix.BSXMUL(alpha, dy);
        var tay = ManagedMatrix.Transpose(axy);
        var txx = ManagedMatrix.Multiply(tay, dx);

        W = ManagedMatrix.Transpose(txx);

        Trained = true;

        ManagedOps.Free(dx, dy, K, kparam, E, alpha, axy, tay, txx);
    }

    // SVMTRAIN Trains an SVM classifier using a simplified version of the SMO
    // algorithm.
    //
    // [model] = svm_train(X, Y, C, kernelFunction, kernelParam, tol, max_passes) trains
an
    // SVM classifier and returns trained model. X is the matrix of training
    // examples.  Each row is a training example, and the jth column holds the
    // jth feature.  Y is a column matrix containing 1 for positive examples
    // and 0 for negative examples.  C is the standard SVM regularization
    // parameter.  tol is a tolerance value used for determining equality of
    // floating point numbers. max_passes controls the number of iterations
    // over the dataset (without changes to alpha) before the algorithm quits.
    //
    // Note: This is a simplified version of the SMO algorithm for training
    // SVMs. In practice, if you want to train an SVM classifier, we
    // recommend using an optimized package such as:
    //
    // LIBSVM   (http://www.csie.ntu.edu.tw/~cjlin/libsvm/)
    // SVMLight (http://svmlight.joachims.org/)
    //
    // Converted to R by: SD Separa (2016/03/18)
    // Converted to C# by: SD Separa (2018/09/29)
    //
    public void Train(ManagedArray x, ManagedArray y, double c, KernelType kernel,
ManagedArray param, double tolerance = 0.001, int maxpasses = 5, int category = 1)
    {
        Setup(x, y, c, kernel, param, tolerance, maxpasses, category);

        // Train
        while (!Step()) { }

        Generate();
    }

    // SVMPREDICT returns a vector of predictions using a trained SVM model
```

```
//(svm_train).
//
// pred = SVMPREDICT(model, X) returns a vector of predictions using a
// trained SVM model (svm_train). X is a mxn matrix where there each
// example is a row. model is a svm model returned from svm_train.
// predictions pred is a m x 1 column of predictions of {0, 1} values.
//
// Converted to R by: SD Separa (2016/03/18)
// Converted to C# by: SD Separa (2018/09/29)
public ManagedArray Predict(ManagedArray input)
{
    var predictions = new ManagedArray(1, Rows(input));

    if (Trained)
    {
        var x = new ManagedArray(input);
        if (Cols(x) == 1)
        {
            ManagedMatrix.Transpose(x, input);
        }
        else
        {
            ManagedOps.Copy2D(x, input, 0, 0);
        }

        var m = Rows(x);

        predictions.Resize(1, m);

        if (Type == KernelType.LINEAR)
        {
            ManagedMatrix.Multiply(predictions, x, W);
            ManagedMatrix.Add(predictions, B);
        }
        else if (Type == KernelType.GAUSSIAN || Type == KernelType.RADIAL)
        {
            // RBF Kernel
            // This is equivalent to computing the kernel on every pair of examples
            var pX1 = ManagedMatrix.Pow(x, 2);
            var pX2 = ManagedMatrix.Pow(ModelX, 2);
            var rX2 = ManagedMatrix.RowSums(pX2);

            var X1 = ManagedMatrix.RowSums(pX1);
            var X2 = ManagedMatrix.Transpose(rX2);
            var tX = ManagedMatrix.Transpose(ModelX);
            var tY = ManagedMatrix.Transpose(ModelY);
```

```
            var tA = ManagedMatrix.Transpose(Alpha);

            var rows = Rows(X1);
            var cols = Cols(X2);

            var tempK = new ManagedArray(cols, rows);
            var temp1 = new ManagedArray(cols, rows);
            var temp2 = ManagedMatrix.Multiply(x, tX);

            ManagedMatrix.Multiply(temp2, -2);

            ManagedMatrix.Expand(X1, cols, 1, tempK);
            ManagedMatrix.Expand(X2, 1, rows, temp1);

            ManagedMatrix.Add(tempK, temp1);
            ManagedMatrix.Add(tempK, temp2);

            var sigma = KernelParam.Length() > 0 ? KernelParam[0] : 1;

            if (Type == KernelType.RADIAL)
                ManagedMatrix.Sqrt(tempK);

            var g = Math.Abs(sigma) > 0 ? Math.Exp(-1 / (2 * sigma * sigma)) : 0;
            var Kernel = ManagedMatrix.Pow(g, tempK);

            var tempY = new ManagedArray(Cols(tY), rows);
            var tempA = new ManagedArray(Cols(tA), rows);

            ManagedMatrix.Expand(tY, 1, rows, tempY);
            ManagedMatrix.Expand(tA, 1, rows, tempA);

            ManagedMatrix.Product(Kernel, tempY);
            ManagedMatrix.Product(Kernel, tempA);

            var p = ManagedMatrix.RowSums(Kernel);

            ManagedOps.Copy2D(predictions, p, 0, 0);
            ManagedMatrix.Add(predictions, B);

            ManagedOps.Free(pX1, pX2, rX2, X1, X2, tempK, temp1, temp2, tX, tY, tA,
tempY, tempA, Kernel, p);
        }
        else
        {
            var Xi = new ManagedArray(Cols(x), 1);
            var Xj = new ManagedArray(Cols(ModelX), 1);
```

```
            for (var i = 0; i < m; i++)
            {
                double prediction = 0;

                ManagedOps.Copy2D(Xi, x, 0, i);

                for (var j = 0; j < Rows(ModelX); j++)
                {
                    ManagedOps.Copy2D(Xj, ModelX, 0, j);
                    prediction += Alpha[j] * ModelY[j] * KernelFunction.Run(Type, Xi, Xj,
KernelParam);
                }
                predictions[i] = prediction + B;
            }

            ManagedOps.Free(Xi, Xj);
        }

        ManagedOps.Free(x);
    }

    return predictions;
}

public ManagedIntList Classify(ManagedArray input, double threshold = 0)
{
    var classification = new ManagedIntList(Rows(input));

    var predictions = Predict(input);

    for (var i = 0; i < predictions.Length(); i++)
    {
        classification[i] = predictions[i] > threshold ? Category : 0;
    }

    ManagedOps.Free(predictions);

    return classification;
}

public int Test(ManagedArray output, ManagedIntList classification, int category =
1)
{
    var errors = 0;
```

```
            for (var i = 0; i < classification.Length(); i++)
            {
                var correct = (int)output[i] != category ? 0 : category;

                errors += correct != classification[i] ? 1 : 0;
            }

            return errors;
        }

        public void Free()
        {
            // public variables
            ManagedOps.Free(ModelX, ModelY, Alpha, W, KernelParam);

            // internal variables
            ManagedOps.Free(K, E, alpha, kparam, dx, dy);
        }
    }
}
```

Naive Bayes

```
using System.Collections.Generic;

namespace MailService
{
    public class NaiveBayes
    {
        private readonly string _text;

        public NaiveBayes(string text) => _text = text;

        private IEnumerable<string> GetDocumentVector()
        {
            var vectorization = new DataProcessing(_text);

            return vectorization.Start();
        }

        private IEnumerable<string> GetVocabulary()
        {
            using var facade = new VocabularyFacade();

            return facade.GetAllTheWords();
        }
```

```
    private void CreateIfDoesNotExist(IEnumerable<string> vector)
    {
        foreach (var word in vector)
        {
            var facade = new VocabularyFacade();

            if (!facade.DoesSuchAWordExist(word))
            {
                facade.Create(word);
            }
        }
    }

    private ModelResult GetResult(INaiveBayesModel model)
    {
        var results = new List<Result>();

        results.Add(model.Calculate<Spam>());
        results.Add(model.Calculate<Correspondence>());

        return new ModelResult(model.Attributes, results);
    }

    private InputData GetInputData()
    {
        IEnumerable<string> vector = GetDocumentVector();

        CreateIfDoesNotExist(vector);

        IEnumerable<string> vocabulary = GetVocabulary();

        return new InputData(vector, vocabulary);
    }

    public EmailClassification Classify()
    {
        InputData data = GetInputData();

        var bernoulliModel = new BernoulliModel(data);
        var polynomialModel = new PolynomialModel(data);

        ModelResult bernoulliResult = GetResult(bernoulliModel);
        ModelResult polynomialResult = GetResult(polynomialModel);

        var function = new LikelihoodFunction(bernoulliResult.Results,
polynomialResult.Results);
```

```
            string category = function.ClassifyEmail();

            return new EmailClassification(_text, category, bernoulliResult,
polynomialResult);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace MailService
{
    public class DataProcessing
    {
        private readonly string _text;
        private readonly static char[] _separators;
        private readonly static string[] _stopWords;
        private readonly static IEnumerable<KeyValuePair<string, string>> _endings;

        static DataProcessing()
        {
            _stopWords = new string[]
            {
                "a", "the", "am", "is", "are", "i", "he", "she", "it", "you",
                "they", "them", "this", "that", "those", "will", "be",
                "do", "does", "have", "has", "would", "may", "to","and",
                "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", string.Empty
            };

            _separators = new char[] { ' ', ',', '-', '.', '!',
                '?', '/', '\"', '\t', '\n', '\r' };

            _endings = new Dictionary<string, string>
            {
                { "ies", "y" },
                { "n't", string.Empty },
                { "'ll", string.Empty },
                { "'d", string.Empty },
                { "'s", string.Empty },
                { "'ve", string.Empty },
                { "ing", string.Empty },
                { "s", string.Empty },

            };
```

```
}
public DataProcessing(string text) => _text = text;
private List<string> Tokenization() => _text.Split(_separators).ToList();

private string Stemming(string word)
{
    string normalizedWord = word;

    foreach (var ending in _endings)
    {
        var template = new Regex($".+{ending.Key}$");

        if (template.IsMatch(word))
        {
            normalizedWord = word.Replace(ending.Key, ending.Value);
            break;
        }
    }

    return normalizedWord;
}

public IEnumerable<string> Start()
{
    List<string> textVector = Tokenization();

    for (int i = 0; i < textVector.Count; i++)
    {
        textVector[i] = textVector[i].ToLower();

        if (_stopWords.Contains(textVector[i]))
        {
            textVector.RemoveAt(i);
            i--;
        }
        else
        {
            textVector[i] = Stemming(textVector[i]);
        }
    }

    return textVector;
}
```

Додаток Б Автореферат

Актуальність теми. Класифікація — це процес розпізнавання, розуміння та групування ідей і об'єктів у попередньо встановлені категорії або «підгрупи». Використовуючи попередньо категоризовані навчальні набори даних, програми машинного навчання використовують різноманітні алгоритми для класифікації майбутніх наборів даних за категоріями.

Алгоритми класифікації в машинному навчанні використовують вхідні навчальні дані, щоб передбачити ймовірність того, що наступні дані потраплять до однієї із заздалегідь визначених категорій. Одним із найпоширеніших застосувань класифікації є фільтрація електронних листів на «спам» і «не спам».

Отже, класифікація — це форма «розпізнавання шаблонів» з алгоритмами класифікації, застосованими до навчальних даних, щоб знайти той самий шаблон (схожі слова чи почуття, послідовності чисел тощо) у майбутніх наборах даних.

Мета дослідження. Метою даної магістерської роботи є виявлення кращого з декількох найвідоміших алгоритму класифікації за допомогою практичного дослідження та порівняння.

Об'єкт дослідження – процес застосування алгоритмів класифікації.

Предмет дослідження – програмні засоби реалізації алгоритмів класифікації.

Методи дослідження. Для досягнення мети дослідження застосовувалися методи алгоритмізації, ООП, машинного навчання, інженерії знань.

Наукова новизна одержаних результатів полягає в наступному:

Проведено порівняння характеристик декількох найвідоміших алгоритмів класифікації, на основі чого визначено який з алгоритмів буде доцільніше застосувати у тому чи іншому випадках.

Наукова новизна. Розроблено алгоритми класифікації із зручним для користувача інтерфейсом.

Практичне значення одержаних результатів полягає у розробленому на основі проведених досліджень програмному забезпеченні, що реалізує можливість тестування алгоритмів класифікації на практиці.

В першому розділі проаналізовано предметну область алгоритмів класифікації та їх практичного застосування. Проаналізовано систему-аналог реалізації алгоритму класифікації, вивчено ряд метрик для їх подальшого оцінювання.

В другому розділі описано процес проектування програмних додатків до трьох досліджуваних алгоритмів класифікації, звернено увагу на особливості архітектури, які потрібно реалізувати максимально ефективно. Створено схематичні структури роботи програм у випадках реалізації.

У третьому розділі повністю описано обґрунтування мови розробки, протестовано розроблені алгоритми, підраховано їх метрики F-міри та визначено найточніший алгоритм.

В четвертому розділі було проведено оцінювання комерційного потенціалу розробки алгоритмів класифікації.

Здійснено технологічний аудит із залученням трьох незалежних експертів, під час якого визначено, що рівень комерційного потенціалу розробки вище середнього.