Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)

Кафедра обчислювальної техніки
(повна назва кафедри)

Пояснювальна записка
до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: «Автоматизована система контролю контенту студентських робіт»

Виконав: студент 2 курсу, групи 1КІ -19м
спеціальності:
123 «Комп'ютерна інженерія»
(шифр і назва напряму підготовки)

Чорний Д. С.
(прізвище та ініціали)

Керівник: к.т.н., доц. Захарченко С. М.
(прізвище та ініціали)

м. Вінниця - 2020 рік

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена проектуванню та розробці автоматизованої системи контролю контенту студентських робіт.

Під час виконання роботи розглянуто основні методи побудови таких систем.Проаналізовано аналогічні системи виявлення плагіату. Розроблено теоретико-математичну модель та програмне забезпечення.

Проведене тестування та експерименти показали коректність та правельність роботи розробленого програмного забезпечення.

ANNOTATION

The master's qualification work isdedicated to the design and development for automated system of control of content of student work.

During the work the main methods of building such systems are considered. Similar plagiarism detection systems are analyzed. Theoretical and mathematical model and software are developed.

Testing and experiments have confirmed the efficiency and correct functioning of the developed software.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій і глобальної мережі Інтернет надав користувачам доступ до величезних масивів інформації. З'явилася велика кількість online-бібліотек, що містять художню і науково-технічних літературу [1]. Стало можливим читати книги, новини та газети безпосередньо з екрану комп'ютера. Крім того, з'явилися величезні колекції рефератів, готових лабораторних робіт, курсових і дипломних проектів і навіть дисертацій. Використання комп'ютерної техніки сильно полегшило завдання пошуку і копіювання подібної інформації.

Почав поширюватися метод написання робіт, що отримав назву «Copy & Paste». Метод полягає в простому копіюванні інформації з одного або декількох джерел з мінімальним редагуванням [2].

Перевірка контенту текстових робіт є у дуже широко розповсюдженою задачею, вона може застосовуватися у різних сферах діяльності людини: для боротьби з плагіатом, для встановлення авторства, для експертизи та встановлення особистості в криміналістиці та у багатьох інших задачах та напрямах.

Перевірка контенту робіт на плагіат та правильність завдання є однією з проблем, що зростають в наукових колах, і завжди викликають занепокоєння в університетах та інших навчальних закладах, оскільки це досить поширене явище серед студентів під час подання будь-яких завдань чи звітів. Викладачі стикаються з труднощами при перевірці багатьох завдань, які включають у себе описову (текстову) частину, тому створення системи перевірки контенту студентських робіт є актуальною задачею. Це дозволить виявити списування, перевірити правильність завдання, варіанту та полегшити роботу викладача.

Метою дослідження є вдосконалення технології перевірки контенту студентських робіт.

Задачі магістерської роботи:

— здійснити аналіз аналогічних систем перевірки контенту;

— запропонувати математичну модель для представлення файлів;

— запропонувати математичну модель для виявлення плагіату;

— розробити алгоритми та програмні засоби для системи перевірки;

— провести експериментальну перевірку розробленої системи контролю контенту студентських робіт.

Об’єкт дослідження магістерської роботи — процес перевірки контенту у текстових файлах.

Предмет дослідження магістерської роботи — методи, алгоритми та програмні зособи для створення автоматизованої системи перевірки контенту студентських робіт.

Наукова новизна отриманих результатів магістерської роботи полягає у тому, що вперше запропоновано комплексну автоматизовану систему перевірки студентських робіт, що дозолить не тільки перевіряти роботи студентів на наявність плагіату, а і контролювати наявність обов’язкових компонентів відповідно до технічного завдання, номеру варіанту тощо.

Практичне значення одержаних результатів магістерської роботи:

— реалізовано алгоритм виявлення плагіату у текстових файлах;

— реалізовано алгоритм пошуку наявності обов’язкових компонентів у текстових файлах;

— розроблено автоматизовану систему перевірки контенту студентських робіт;

— виконано програмну реалізацію основних модулів системи перевірки контенту студентських робіт.

Публікації — за результатами магістерської кваліфікаційної роботи опубліковано тези доповіді на науко-практичній Інтернет-коференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2021)[3].

1 АНАЛІЗ СИСТЕМ КОНТРОЛЮ КОНТЕНТУ ТА МЕТОДІВ ПОРІВНЯННЯ ТЕКСТОВИХ ФАЙЛІВ

Інформатизація суспільства, стрімкий розвиток технологій та вільний доступ до продуктів інтелектуальної власності полегшують процес використання та розповсюдження інформації. Тому розробка систем, які перевірили контент на правильність, плагіат є як ніколи актуальною задачею.

В даний час існує велика кількість сервісів і програм для перевірки контенту текстових файлів. Вони дозволяють різними способами перевірити схожість досліджуваних об’єктів.

Даний розділ магістерської роботи присвячений розгляду та аналізу побудови автоматизованих систем контролю контенту у текстових файлах.

1.1 Основні характеристики плагіату

Основним завданням перевірки контенту текстових файлів є знаходження схожих фрагментів у інших файлах, тобто знаходження плагіату.

Плагіат — це умисне привласнення авторства, думок, ідей або виразів іншого учасника як власної оригінальної роботи. В освітніх контекстах існують різні визначення плагіату в залежності від установи. Серед видатних дослідників плагіату — Ребекка Мур Ховард, Сьюзен Блюм, Трейсі Бретаг і Сара Елейн Ітон [4].

Плагіат вважається порушенням академічної чесності та журналістської етики. Він підлягає санкціям, таким як штрафи, відсторонення від занять, виключення зі школи або роботи і навіть тюремне ув'язнення. Останнім часом в академічних колах були виявлені випадки «крайнього плагіату». Сучасна концепція плагіату як аморальності і оригінальності як ідеалу виникла в Європі в 18 столітті, особливо з романтичним рухом.

Як правило, плагіат сам по собі не є злочином, але, як і підробка підробок, шахрайство може бути покарано в суді за упередження [5], викликані порушенням авторських прав, порушенням особистих немайнових прав або

правопорушеннями. У наукових колах і на виробництві це серйозне етичне порушення.

Основні принципи плагіату:

— копіювання чужої роботи (як без відома автора, так і з відома автора) та публікація його під своїм ім’ям;

— подання суміші власних та запозичених аргументів без належного цитування джерел інформації;

— виклад чужих ідей як власних;

— перефразування запозиченої роботи без цитування першоджерела;

— фальсифікація результатів роботи ;

— використання власних, раніше опублікованих ідей у нових роботах без належного цитування та ін.

Величезні інформаційні ресурси Інтернету істотно змінили можливості навчання. В ідеалі мета навчання у вузі — отримання не тільки диплома, але і знань [6]. Тоді студенти повинні «по максимуму» використовувати можливості Інтернету для підготовки якісних робіт. Однак такі роботи вимагають багато часу. У разі «недбайливих» студентів ресурси можуть використовуватися, і дійсно, використовуються для прискореної підготовки — для «скачування» рефератів, контрольних та курсових робіт. У багатьох таких випадках змінюється тільки прізвище автора. Тобто, має місце плагіат «в чистому вигляді». Однак слід критично підходити до матеріалів, представлених в мережі Інтернет, особливо в частині, що стосується «готових рефератів», «готових курсових робіт». Автори їх не несуть ніякої відповідальності за рівень і якість представлених матеріалів, так що всі претензії викладач буде пред'являти «недбайливому студенту». Тобто, навіть якщо факт плагіату і не виявиться, робота може бути незадовільною сама по собі. В даний час системи аналізу схожості текстів отримують все більш широке поширення.

1.2 Основні етапи вирішення задачі створення системи контролю контенту

Першим етапом вирішення проблеми автоматичної перевірки текстів є перетворення документів. На цьому етапі документи у вигляді послідовності символів перетворюються у форму, придатну для алгоритмів машинного навчання відповідно до класифікаційної задачі. Зазвичай алгоритми машинного навчання мають справу з векторами у просторі, який називається простором ознак.

Другий етап — побудова класифікаційної функції. Якість класифікації залежить як від того, як документи будуть перетворені у векторне подання, так і від алгоритму, який буде застосовано на другому етапі. Важливо зазначити, що методи перетворення тексту у вектор є специфічними для завдання класифікації текстів і можуть залежати від колекції документів, типу тексту (простого, структурованого) та мови документа. Методи машинного навчання, що використовуються на другому етапі, не є специфічними для проблеми класифікації тексту, а також використовуються в інших областях, наприклад, для розпізнавання шаблонів.

Отже, задачу інтелектуальної перевірки можна розбити на два основні етапи:

— попередня обробка / індексування — відображення тексту документа до його логічного подання, наприклад, який потім подається на вхід алгоритму класифікації;

— класифікація / навчання — етап класифікації документа або навчання за кількома документами на основі логічного подання документа, важливо зазначити, що для класифікації та навчання можна використовувати загальний метод попередньої обробки / індексації тексту.

Після цього можна здійснювати порівняння документа запиту з іншими та перевіряти окремі елементи. Отже, схема основних етапів контролю буде мати вигляд (рисунок 1.1):

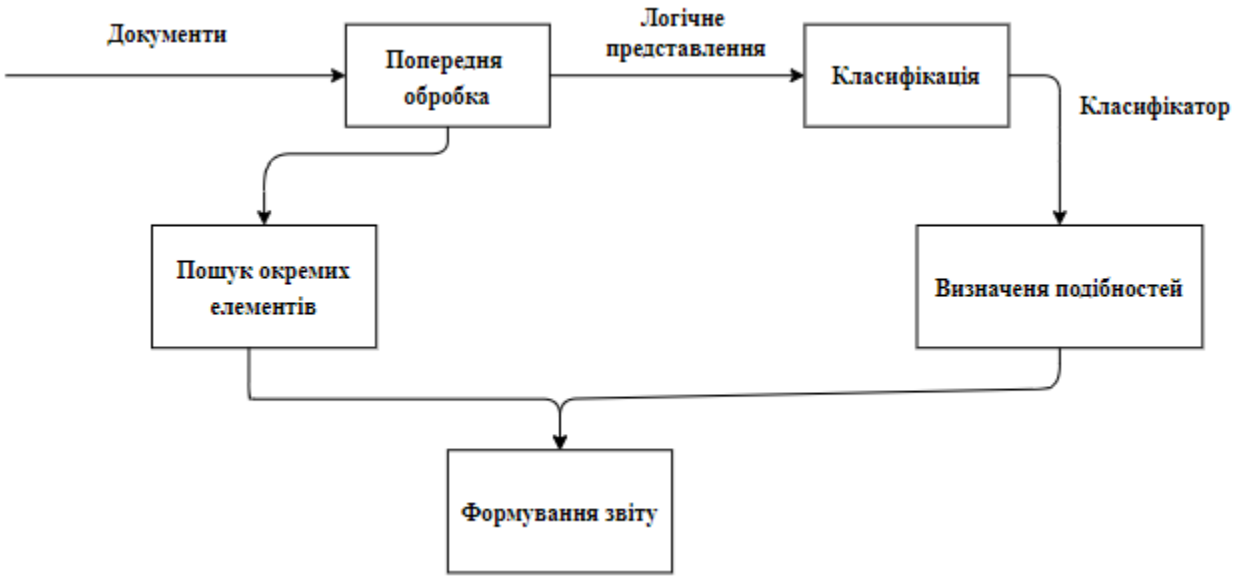

Рисунок 1.1 — Основні етапи класифікації

1.3 Огляд існуючих методів порівняння текстових документів на схожість

Ми зосередимося на виявленні моделей схожості, які часто зустрічаються у звітах студентів:

— шаблон A: скопійовано весь текст (повна копія);

— шаблон B: скопійовано весь текст один раз і частково модифіковано (як правило, кілька місць), наприклад обмін, вставку та видалення слів та символів у реченні, щоб спробувати втекти від викриття плагіату;

— шаблон C: скопійовано частину текстів із певного або декількох документів та поєднано їх оригінальними реченнями. У деяких випадках змінюється розміщення речень у тексті та слів у реченні, у порівнянні з оригінальним текстом;

— шаблон D: гібрид B і C.

Фактичний приклад шаблону плагіату C занадто великий, щоб його можна було показати, тому він показаний на рис. 1.2 як концептуальна модель. Тут текст A, B та C означає плагіат, а текст P1, P2 та P3 породжує тексти, отримані плагіатом. Крім того, рядки, виражені x, y, p, q, є розділами плагіату, а дефіси - неплагіатними розділами.

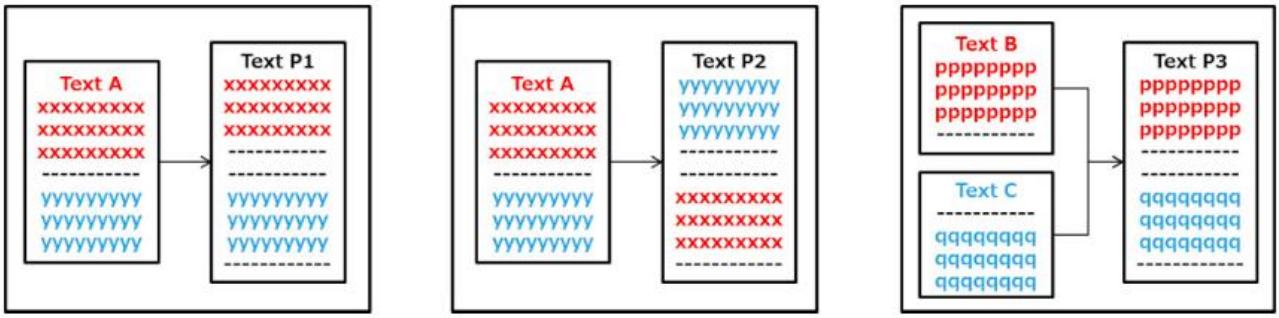

Рисунок 1.2 — Приклади шаблону плагіату

Існує ряд методів для знаходження подібності в корпусах текстів. Деякі заслуговують увагу деякі ні, оскільки їх легко можна обійти. Далі озглянемо найбільш елементарний з них.

Виявлення плагіату полягає у пошуку розділу відповідних слів довільного розміру між двома текстами, які потрібно порівняти.

Цей процес вимагає більших обчислень, ніж пошук наявності заздалегідь заданого рядка. Найбільш стійким і простим методом виявлення для наших цільових моделей плагіату є перетворення двох текстів в одновимірні рядки відповідно та повторення зсуву рядків та порівняння символів між ними.

Приклад методу показаний на рисунку 1.3. Алфавіт на малюнку може бути літерою, або може бути словом після морфологічного розкладання. Можна виявити плагіат незалежно від його розміщення, але є купа обмежень та умов. У цьому прикладі "ABC" та "abc" виявляються як розділи плагіату.

У цьому методі необхідно заздалегідь визначити поріг відповідності розміру рядка для виявлення, але простіше інтуїтивно встановити його. А також є перевага, що сам процес дуже простий. З іншого боку, є недолік, що порівняння між великими обсягами текстів стає важким, оскільки обсяг розрахунку для порівняння великий. А також, цей метод можна буде обманути таким простим способом, як обмін словами, і його не вдається виявити, оскільки він базується на ідеальному узгодженні. Найпростіша заміна ‘ і ’ на ‘ та ’, ‘ у ’ на ‘ в’ зробить неможливим виявлення плагіату .

З вище сказаного, даний метод нам не підходить. Тому звернемо свою увагу на більш складні та продуктивні алгоритми.

Рисунок 1.3 — Метод базового виявлення плагіату

З вище сказаного, даний метод нам не підходить. Тому звернемо свою увагу на більш складні та продуктивні алгоритми.

Найбільш часто процес пошуку здійснюється за алгоритмом описаним нижче.

Евристичний пошук. З корпусу відбираються документи, в яких містяться фрагменти тексту, схожі на деякі частини тексту T0, що перевіряється на плагіат. На цьому кроці використовуються різні алгоритми для виявлення схожих текстів [7].

Докладний аналіз. Для кожного з відібраних документів аналізується ступінь подібності з початковим текстом, якщо вона досить висока — передбачається випадок плагіату. У багатьох системах цей крок застосовується без евристичного пошуку: порівняння йде з кожним з текстів корпусу.

Пост-обробка. Отримані результати перевіряються, найчастіше вручну, для виключення помилкових виявлень (наприклад, випадків, коли за плагіат приймається оформлена за всіма правилами цитата).

Пошук нечітких дублікатів дозволяє припустити, чи є два об'єкти частково однаковими чи ні. Під об'єктом можуть розумітися текстові файли та інші типи

даних [8]. Ми будемо працювати з текстом, але зрозумівши, як працює алгоритм, нам не складе труднощів перенести реалізацію на необхідні нам об'єкти.

1.3.1 Алгоритм шинглів

Реалізація алгоритму складається з кількох етапів:

— канонізація текстів;

— розбиття тексту на шингли;

— знаходження контрольних сум;

— пошук однакових підпослідовностей.

Шингли — послідовності слів. Спочатку з тексту видаляються усі сполучники, спеціальні символи, короткі слова, крім пробілів. Далі з отриманих слів складаються самі шингли — послідовності. Плагіатом можна вважати збіг у 6 слів. Аналогічна операція виконується для другого документа. Таким чином, порівнюються шингли, і якщо у документах співпадають один або більше шинглів, тексти можуть вважатися подібними.

Існує ряд модифікацій алгоритма на основі шинглів . Наприклад, одна із модифікацій базується на сортуванні, тобто перед обчисленняями слова у шинглі будуть відсортовані. Це покращує пошук запозичень у випадках, коли слова у тексті проаналізованої роботи були переставлені місцями для обману системи.

До переваг простого алгоритму шинглів можна віднести гнучкість, точність, швидкість та популярність. Проте недоліком є те, що не можна вказати точне місце виявлення подібних фрагменів. На рисунку 1.4 представлені етапи роботи системи, яка використовує алгоритм шинглів.

1.3.2 Пошук схожих документів з MinHash

Алгоритм MinHash розроблений як продовження алгоритму Shingles. Він потребує менше часу на порівняння образів різних текстів і меншу кількості займаної пам'яті. Після обробки тексту алгоритмом шинглів алгоритм MinHash здійснює обчислення хеш-функцій і відбір їх мінімальних значень [9]. Алгоритм MinHash використовує h хеш-функцій.

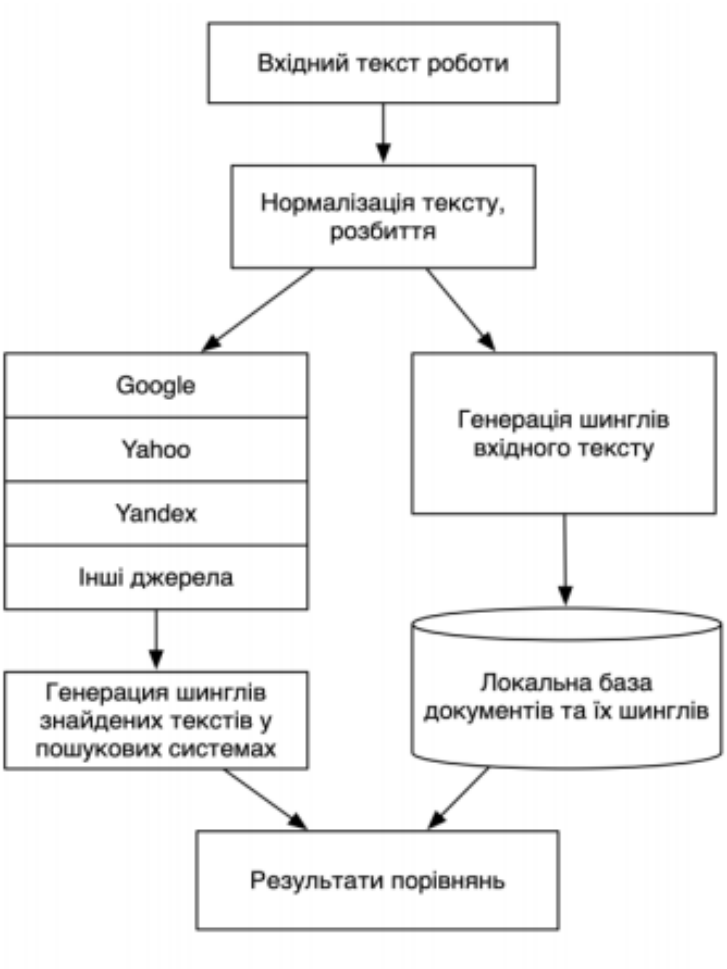

Рисунок 1.4 — Загальна схема алгоритму шинглів у системі виявлення плагіату

Кожній хеш-функції ставиться у відповідність число — найменше її значення для всіх шинглів досліджуваного рядка. Отриманий в результаті вектор довжини h утворює сигнатуру або короткий числовий образ документа. На вхід алгоритму MinHash подається масив шинглів довжини N . Для порівняння будь-якого документа за допомогою MinHash його необхідно спочатку перетворити в множину елементів. Для текстів непогано підійде перетворення в множину ID слів. Також можна розбити тексти на N-грами. На виході алгоритм видає одновимірний масив (вектор) хеш-кодів довжини h , тобто сигнатуру або короткий числовий образ документа [10].

Алгоритм MinHash використовує однаковий набір хеш-функцій в заданому порядку для обробки всієї множини текстів, тому список хеш-функцій зазвичай є частиною реалізації алгоритму. Порівняння документів здійснюється за

значеннями векторів hashes1 і hashes2, побудованих алгоритмом MinHash з використанням одного і того ж складу хеш-функцій. Побудовані вектори hashes1 і hashes2 завжди мають однакову довжину, рівну h. Для виявлення дублікатів в алгоритм MinHash вбудовано 60 хеш-функцій.

Цей алгоритм відрізняється від інших своєю неймовірною швидкістю і здатен набагато пришвидшити пошук ймовірних дублікатів.

1.3.3 I-Match

Розглянемо сигнатурний підхід заснований на лексичних принципах. Головна ідея алгоритму полягає в обчисленні дактілограмми I-Match для подання змісту документів. Для цього для вихідної колекції документів будується словник L, який включає слова з середніми значеннями IDF, оскільки такі слова забезпечують, як правило, більш точні результати при виявленні нечітких дублікатів. Слова з великими і маленькими значеннями IDF відкидаються. Потім для кожного документа формується безліч U різних слів, що входять в нього, і визначається перетин U і словника L. Якщо розмір цього перетину більше деякого мінімального порогу 9 (визначається експериментально), то список слів, які входять в перетин впорядковується, і для нього обчислюється I-Match сигнатура (hash-функція SHA1).

Документа вважаються подібними, якщо у них збігаються I-Match сигнатури (має місце колізія hash-кодів). Алгоритм має високу обчислювальну ефективність, що перевершує показники алгоритму шинглів. Іншою перевагою алгоритму (також, в порівнянні з алгоритмом шинглів) є його висока ефективність при порівнянні невеликих за розміром документів. Основний недолік — нестійкість до невеликих змін змісту документа.

Додатково до основного словника L створюються K різних словників L1—LK, одержуваних шляхом випадкового видалення з вихідного словника деякого невеликої фіксованої частини p слів, складової порядку 30% -35% від початкового обсягу L[11].

Для кожного документа замість однієї обчислюється (K + 1) I-Match

сигнатура, тобто документ розглядаються у вигляді вектора розмірності (K + 1), і два документа вважаються дублікатами, якщо у них збігається хоча б одна з координат. Якщо документ піддається невеликим змінам (порядку n слів), то ймовірність того, що принаймні одна з K додаткових сигнатур залишиться незмінною.

1.3.4 Пошук схожих документів з SimHash

SimHash є хеш-функцією, і її особливість полягає в тому, що чим більше схожих текстових рядків на вході, тим менша відстань Хеммінга в їх хешах (відстань Хеммінга — кількість позицій, на яких відповідні символи різні). Алгоритм працює, розбиваючи текст на шматки та хешуючи кожен фрагмент вибраною функцією хешування. Кожен хешований фрагмент представлений у вигляді бінарного вектора, і бітові значення перетворюються на +1 або -1 залежно від того, значення біта дорівнює 1 або 0. Для отримання SimHash ми складаємо бітові вектори. Нарешті, отримані біти встановлюються на 0, якщо сума негативна, інакше -1.

Для створення сигнатури необхідно виконати такі дії:

— вибираємо розмір сигнатури;

створюємо масив V такого ж розміру, як кількість бітів у сигнатурі, і заповнюємо його нулями;

— вибираємо хеш-функцію, що дає результати розміру, аналогічно вибраному;

— для кожного набору, для кожного елементу в наборі розраховуємо значення хеш-функції;

— для кожного отриманого хешу в наборі, для кожного біта i, якщо біт=1, додаємо 1 до V [i] ;

— для кожного отриманого хешу в наборі, для кожного біта i, якщо біт=0, віднімаємо 1 від V [i] ;

— для кожного елементу i масиву V :

— якщо V [i] > 0, V [i] = 1 ;

— якщо V [i] ≤ 0, V [i] = 0 .

Отримані сигнатури матимуть вигляд двійкових кодів. Виконавши операцію побітового додавання по модулю 2 (XOR), отримаємо двійкове значення, яке є умовним значенням відмінності наборів. Завдяки цьому значенню можна вирахувати відсоток подібності наборів[12] .

Алгоритм накладає обмеження на те, наскільки різними повинні бути документи, щоб бути виявленими цим алгоритмом як дублікат. Тому дані мають бути дуже схожі і невеликого розміру. Алгоритм підійде для пошукових систем у Інтернеті

1.3.5 Опис методу порівняння текстів на ідентичність на основі ущільнення

Одним з методів порівняння текстів на ідентичність є метод на основі ущільнення, що використовує normalized compression distance (NCD). NCD — це спосіб виміру подібності між двома об’єктами. Даний метод є доволі новим, він був запропонований С. Ващіліном і базується на засадах механізму ущільнення даних, а саме — усунення надлишку інформації, що міститься у вихідних даних. Його суть полягає у наступному [13].

Нехай ми маємо текстові документи A, B. Для порівняння їх на вмісту на ідентичність необхідно виконати наступні кроки (кроки 2—3 наведені на рисунку 1.5):

— зчитати текстові документи A B, у двійковому вигляді у відповідні змінні;

— виконати конкатенацію A B, у зміну AB;

— почергово застосувати обраний алгоритм ущільнення на змінні A, B, AB та записати результат у змінні cA, cB, cAB відповідно;

— вирахувати нормалізовану відстань ущільнення.

Далі, відповідно до значення NCD та встановлених допустимих меж, що обираються емпірично, визначається ступінь схожості документів. Даний метод не визначає конкретний алгоритм ущільнення, що необхідно застосовувати.

Головним недоліком даного методу є те, що він не стійкий до завад.

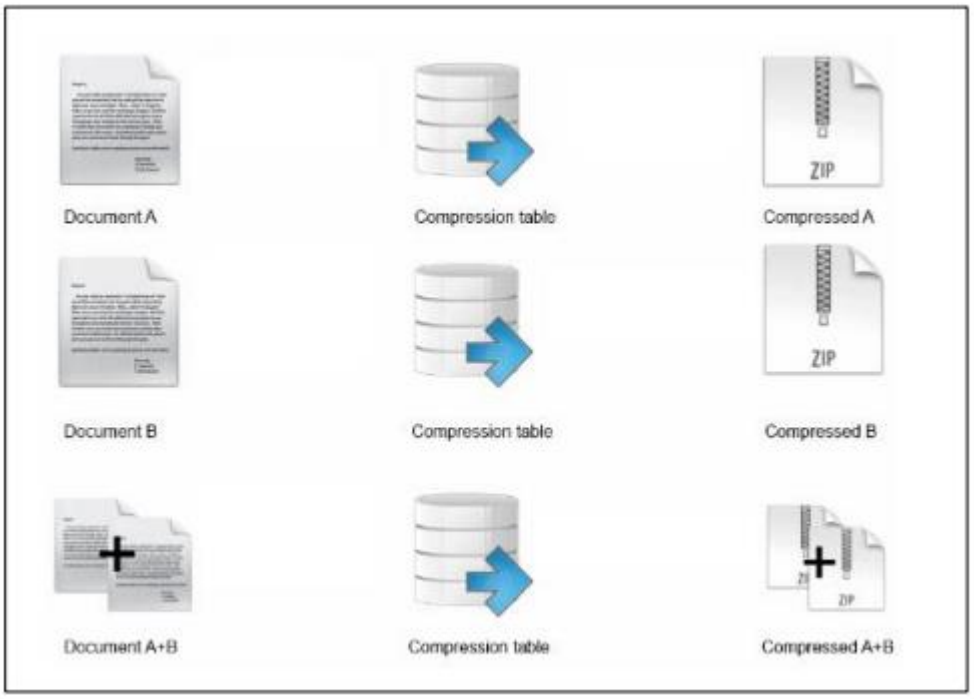

Рисунок 1.5 — Метод NCD

Невідомо як поведе себе з великою кількістю однотипних слів, стоп-слів і подібне. Все це випливає з того, що підхід новий і потребує більш тривалих і глибоких досліджень.

1.4 Огляд існуючих аналогів систем контролю контенту

Системи контролю контенту можна класифікувати за наступними ознаками [14]:

Класифікація за корпусом:

а) за типом БД:

— внутрішня — наповнюється користувачем;

— зовнішня — електронний репозиторій, дані знаходяться у мережі— Інтернет;

— змішана.

б) Типи даних, які обробляє:

— текстові;

— програмні коди;

— інші (зображення, таблиці, схеми, відео тощо);

Класифікація за характеристиками засобів, які використовуються для контролю контенту:

а) за типом:

— локальне програмне забезпечення, яке встановлюється на комп'ютер користувача;

— програмне забезпечення, яке встановлюється на комп'ютер користувача, але під'єднується до інших ресурсів (віддалені сервери, Інтернет);

— он—лайн ресурси;

б) власність:

—вільно поширене;

— платне;

— змішане.

в) класифікація за метрикою, що використовується:

— поверхневі показники (кількість однакових шматків у документах);

— складні показники (міра (відсоток) подібності документів із урахуванням додаткових параметрів) [15].

Існують і інші ознаки, за якими можна класифікувати, було розглянуто основні.

Розглянемо деякі приклади систем перевірки контенту текстових файлів.

TextDiff — безкоштовна програма для порівняння файлів. Крім порівняння вмісту файлів, програма TextDiff може зробити порівняння двох каталогів, виділяючи кольорами назви однакових файлів (рисунок 1.6).

Головне вікно програми TextDiff розділене по вертикалі (можна і по горизонталі) на два суміжних вікна. У них відображається вміст порівнюваних файлів. Скролінг рядків в області перегляду одного файлу одночасно веде до автоматичного скролінгу рядків іншого файлу, і користувач бачить в якому рядку між файлами є відмінності. TextDiff не вимагає встановлення, повністю підтримує юнікод, вміє порівнювати каталоги. Результати порівняння файлів можна зберегти в текстовий файл.

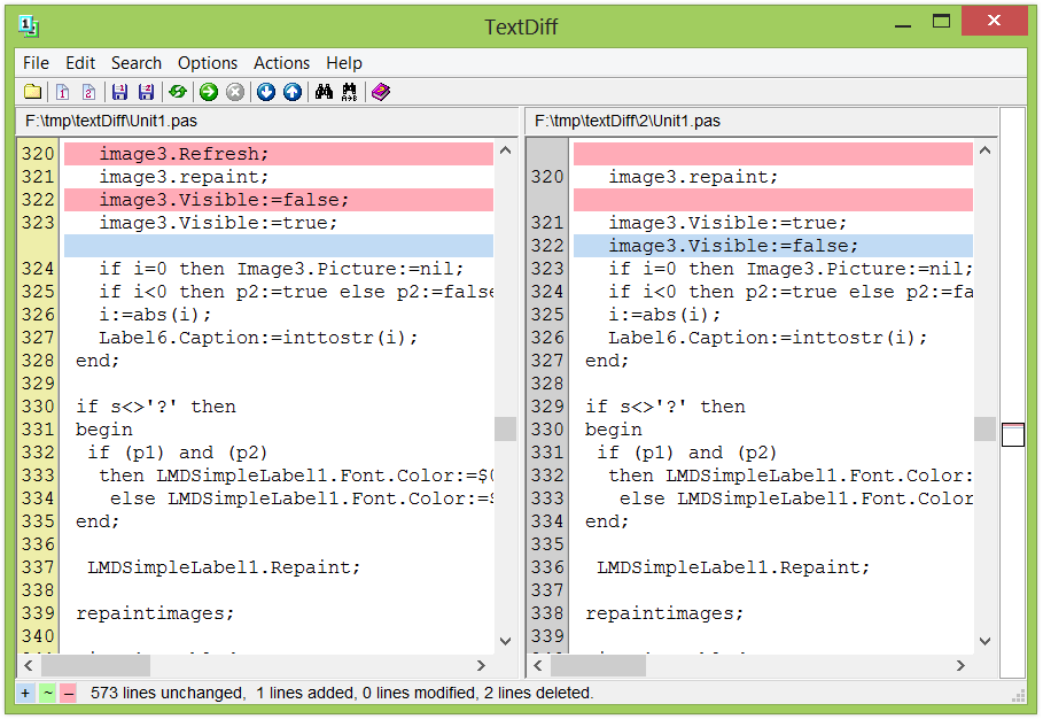

Рисунок 1.6 — інтерфейс програми TextDiff

Недоліки:

— може порівнювати тільки звичайні текстові файли;

— порівнює файли не великого розміру.

Content-Watch.ru — це мінімалістичний сервіс, з точки зору дизайну, а також функціональності. Цей сервіс здійснює перевірку вхідних даних тільки на унікальність (рисунок 1.7). Сервіс має безкоштовну та платну версію.

Безкоштовна версія сервісу має купу обмежень:

— 7 перевірок в день;

— текст до 10 тисяч знаків.

Платна версія дозволяє перевіряти 150—3000 статей в день з обмеженим обсягом символів до 20 тисяч. Користувач зможе проводити регулярні перевірку документів, а також переглядати власну історію перевірок на плагіат. Сервіс передбачений на малі та середні обсяги інформації, але не здатен перевіряти на плагіат велику кількість інформації. У сервісу є проблема з невеликими текстами — він вказує на велику кількість запозичень, навіть при відсутності таких.

eTXT Антиплагиат (http://www.etxt.ru/antiplagiat/) — сервіс перевірки текстів на плагіат, що доступний як програмне забезпечення для встановлення на

комп’ютер, так і у форматі он-лайн ресурсу.

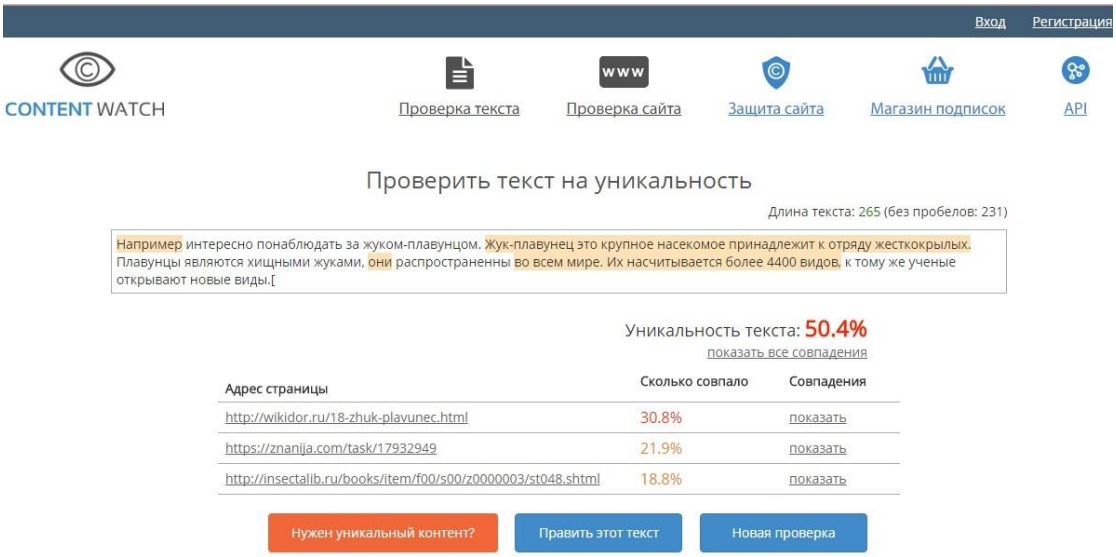

Рисунок 1.7 — Система виявлення плагіату Content-Watch.ru

Можливо здійснювати перевірку текстових фрагментів, окремих файлів та пакетів файлів, а також сторінок сайтів. Передбачено 2 функції перевірки тексту: на плагіат (наявність дослівних збігів) за допомогою опції «Метод виявлення копій», та на наявність рерайта (смислових збігів) — опція «Метод виявлення рерайта» (рисунок 1.8). Серед переваг можна зазначити наступні:

— присутня градаційна перевірка текстів на чотирьох рівнях:
— користувач може налаштувати програму під себе;
— є можливість пакету документів;
— є можливість перевірки двох локальних текстів;
— привабливий та продуманий інтерфейс.

Наявні і свої мінуси. Головне — швидкість досить нижче, ніж в онлайн-сервісах, це пов’язано з тим що онлайн-сервіси мають найсучасніше обладнання та великі потужності, на яких відбуваються процеси обробки та пошуку необхідної інформації. Також програма все частіше просить користувача вводити коди безпеки, якщо було підвищено рівень перевірки. Програма є платною, якщо застосовувати он-лайн перевірку та певні інші функції.

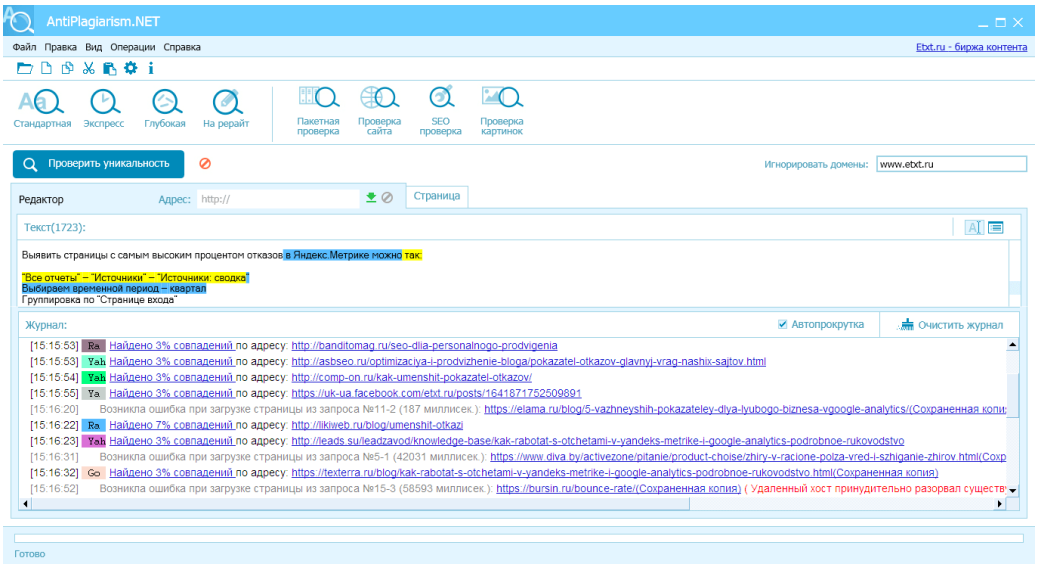

Рисунок 1.8 — Меню eTXT Антиплагіат

Advego Plagiatus — додаток пошуку в Інтернеті часткових або повних копій текстового документа, що показує ступінь унікальності тексту. Наявна функція перевірка унікальності за URL-адресою (рисунок 1.9). Пошук плагіату доступний в режимі «Швидкої перевірки» та «Глибинної перевірки».

Мінуси Advego Plagiatus такі ж самі, що і у eTXT Антиплагіат, а також відсутність пакетної перевірки та порівняння двох текстів.

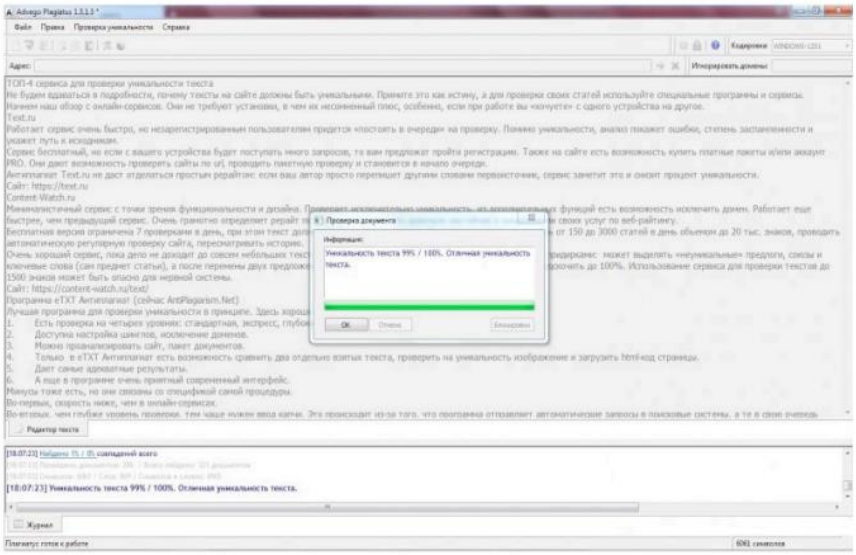

Рисунок 1.9 — Advego Plagiatus

Отже,у розділі було розглянуто основні завдання при створенні автоматизованої системи перевірки контенту, що дозволило визначати етапи, які

потрібно розробити та реалізувати у теоретико-математичній моделі.

Розглянуто основні методи побудови даних систем, з огляду на це для подальшої роботи було обрано алгоритм шинків на основі n- грам. Це пов`язано з тим, що вони найбільш повно задовольняють наші потреби. До переваг даних ознак можна віднести високу точність оцінки і простоту реалізації.

Було про класифіковано системи контролю контенту за різними ознаками. Також було розглянуто існуючі системи виявлення плагіату та визначено їхні основні плюси та мінуси, що показало доцільність нашої розробки, оскільки жодна з систем не може задовольнити наші потреби.

.

2 РОЗРОБКА ТЕОРЕТИКО-МАТЕМАТИЧНОЇ МОДЕЛІ СИСТЕМИ ПЕРЕВІРКИ КОНТЕНТУ

Для вирішення поставлених завдань просто вхідного тексту із файлів недостатньо. Нам необхідно здійснити перед обробку тексту. Далі, для більш глибокого вивчення вхідного тексту необхідно використати методи, які дозволять виявити і виокремити певні характеристики, параметри для вхідних даних. У даному розділі ми розробимо теоректико-метематичну модель нашої системи.

2.1 Токенізація тексту

Два важливі етапи обробки текстів включають токенізацію та нормалізацію. Очевидно, що одиницею тексту є слово. Токенізація означає поділ тексту на окремі одиниці слів, які називаються лексемами. В результаті токенізації оригінальний рядок тексту перетворюється на список слів, з яких він складався. Токенізація тексту виконується в кілька етапів [19]:

— приведення вхідного тексту до нижнього регістру;

— видалення усіх розділових знаків та заміна їх на пробіли;

— оголошення слів окремими лексемами

Цей спосіб попередньої обробки досить простий і зрозумілий, але незважаючи на це саме на кожному з етапів можуть бути різні нюанси, а саме:

Доведення абревіатур до малої літери можна сплутати з вираженням емоцій.

Коли всі розділові знаки замінено пробілами, оригінальні слова можуть бути втрачені. Наприклад, є складні слова, які пишуться через дефіс (червоно-синій). Після видалення дефіса ми отримуємо два слова замість початкового, яке несе різне навантаження. Текст, написаний у вільному стилі, може містити смайли, які відіграють особливу роль у семантичному аналізі тексту. Видалення розділових знаків виключає можливість використання цієї функції.

Власні імена можуть йти парами. Наприклад, назви окремих країн, міст, організацій (Саудівська Аравія). Під час лексемування вони будуть розділені на два окремі слова. Але з точки зору логіки, було б корисно розглядати їх як єдине ціле.

Подібне питання виникає зі скороченнями. Існують прийняті абревіатури, які будуть втрачені під час токенізації і, крім того, будуть містити неправдиву інформацію в отриманому наборі.

Крім того, ви можете виділити проблему, яка може виникнути при токенізації текстів певного типу. А саме, пробіли рідко використовуються в китайській мові, в результаті чого текст є безперервним набором ієрогліфів, до яких алгоритм токенізації не можна застосувати так легко.

Крім того, у німецькій мові часто використовується практика утворення складних слів шляхом об'єднання кількох. Для обробки таких текстів потрібна сегментація тексту до слова.

2.2 Моделі подання інформації

2.2.1. Логічна модель

Логічна модель (булева модель: "заснована на логічній логіці та класичній теорії множин, оскільки документи, що підлягають пошуку, і запит користувача розглядаються як набори термінів". Цю модель часто називають моделлю "точного збігу" [20].

Булева модель — це перша модель пошуку інформації і, мабуть, також найбільш критикувана модель. Модель можна пояснити, розглядаючи термін запиту як однозначне визначення набору документів. Наприклад, термін запиту економічний просто визначає набір усіх документів, які індексуються терміном економічний.

Використовуючи оператори математичної логіки Джорджа Буля, терміни запитів та відповідні їм набори документів можна об’єднати, щоб сформувати нові набори документів. Буле визначив три основні оператори, логічний продукт, який називається І, логічну суму, що називається АБО, і логічну різницю, що називається НЕ. Поєднання термінів з оператором AND визначає набір документів, який менший або дорівнює наборам документів будь-якого з окремих термінів. Наприклад, запит соціальне І економічне дасть набір документів, які індексуються як терміном соціальний, так і терміном економічний, тобто перетин

обох наборів. Поєднання термінів з оператором АБО визначає набір документів, який більший або дорівнює наборам документів будь-якого з окремих термінів. Отже, запит соціальне АБО політичне дасть набір документів, які індексуються або терміном соціальний, або терміном політичний, або обома, тобто об'єднанням обох наборів. Це візуалізується на діаграмах Венна (рисунок 2.1), на якому кожен комплект документів візуалізується на диску.

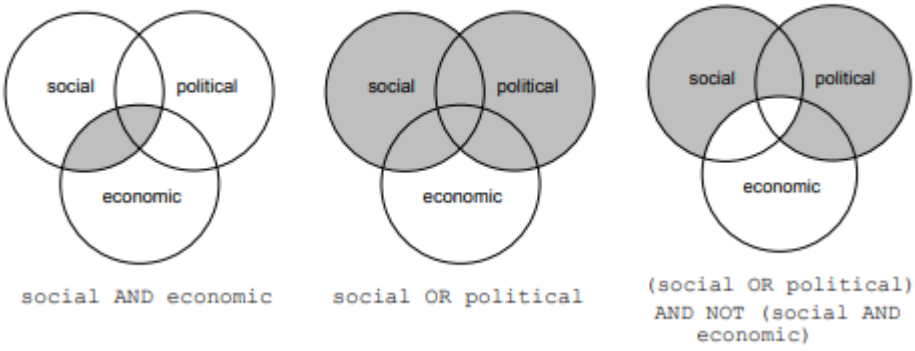

Рисунок 2.1 — Булеві комбінації наборів, що візуалізуються як діаграми Венна

Перекриття цих дисків та їх доповнення поділяють колекцію документів на 8 областей, профспілки яких дають 256 різних булевих комбінацій "соціальних, політичних та економічних документів" [21]. На рисунку 2.1 отримані набори візуалізуються затіненими областями.

Перевагою булевої моделі є те, що вона дає (експертам) користувачам відчуття контролю над системою. Одразу зрозуміло, чому документ було отримано за запитом. Якщо результуючий набір документів або занадто малий, або занадто великий, прямо зрозуміло, які оператори виготовлять відповідно більший або менший набір.

Для непідготовлених користувачів модель має ряд явних недоліків. Основним його недоліком є те, що вона не дає рейтингу отриманих документів. Це може призвести до того, що система прийме досить неприємні рішення.

2.2.2 Ймовірнісна модель

Імовірнісна модель, за якої релевантність документа для даного запиту може бути оцінена за допомогою вірогідності знаходження відповідної інформації та ймовірності знаходження нерелевантної інформації.

Кілька підходів, які намагаються визначити зважування термінів більш формально, базуються на теорії ймовірностей. Поняття ймовірності чогось, наприклад, ймовірність релевантності, позначене як P (R), зазвичай формалізується через концепцію експерименту, де експеримент — це процес, за допомогою якого здійснюється спостереження [22]. Сукупність усіх можливих результатів експерименту називається простором вибірки. У випадку P (R) вибірка може бути {актуальним, нерелевантним}, і ми можемо визначити випадкову величину R, щоб взяти значення {0, 1}, де 0 = нерелевантно і 1 = релевантно.

Давайте визначимо експеримент, для якого ми навмання беремо один документ із колекції: якщо ми знаємо кількість відповідних документів у колекції, скажімо, 100 документів є релевантними, і ми знаємо загальну кількість документів у колекції, скажімо 1 мільйон тоді коефіцієнт цих двох визначає ймовірність релевантності:

$$P (R = 1) = 100/1000000 = 0,0001$$

Припустимо, крім того, що P (Dk) — це ймовірність того, що документ містить термін k із простором вибірки {0, 1}, (0 = документ не містить термін k, 1 = документ містить термін k), тоді ми будемо використовуйте P (R, Dk) для позначення спільного розподілу ймовірностей із результатами {(0, 0), (0, 1), (1, 0) та (1, 1)}, і ми будемо використовувати P (R | Dk) для позначення умовного розподілу ймовірності з результатами {0, 1}. Отже, P (R = 1 | Dk = 1) - це ймовірність релевантності, якщо розглядати документи, що містять термін k.

Кожного разу, коли ми говоримо про іншу випадкову величину, ми також говоримо про інший захід P. Отже, одне рівняння може посилатися на декілька мір ймовірностей, всі неоднозначно позначаються як P. У різних моделях можуть

бути різні пробіли. Наприклад, D в імовірнісній моделі індексації — це випадкова величина, що позначає „це відповідний документ”, що має якнайбільші результати ідентифікатори документів у колекції. Однак D у моделі імовірнісного пошуку — це випадкова величина, яка має як можливий результат усі можливі описи документів, які в даному випадку є векторами з двійковими компонентами dk, які позначають, індексується документ терміном k чи ні.

Дана модель нам теж не підходить. Вона має ряд недолік, які роблять неможливим її використання. Її результати не можна трактувати однозначно також вони не дають конкретної відповіді на запит при великій кількості документів.

2.2.3 Модель векторного простору

Модель векторного простору представляє документи та запити як вектори у багатовимірному просторі, розміри яких є термінами, що використовуються для побудови індексу для представлення документів. Модель векторного простору може присвоїти документу високий рейтинг, який містить лише кілька термінів запиту, якщо ці терміни трапляються нечасто в колекції, але часто в документі.

Модель векторного простору робить такі припущення:

— чим більше вектор документа схожий на вектор запиту, тим більша ймовірність того, що документ має відношення до цього запиту;

— слова, що використовуються для визначення розмірів простору, є ортогональними або незалежними.

Які переваги концепції векторного простору? Виявляється, ми можемо використовувати метафору простору для обчислення подібності між об’єктами, тобто для порівняння об’єктів, заснованих лише на дуже простих геометричних уявленнях, не більш складних, ніж теорема Піфагора. Тепер кожен об'єкт є вектором у просторі N розмірностей. Як ми можемо порівняти ці вектори? Геометричний принцип стверджує, що вектори в більш-менш однаковому напрямку нагадують один одного [23]. Формально кажучи, чим гостріший кут між векторами, тим більша подібність. Це дуже чітко та інтуїтивно зрозуміло у

двовимірному просторі. Наприклад, на рисунку 2.2, чим гостріший кут між кожною парою стрілок, тим більше "подібних" стрілок цієї пари, тобто їх напрямки найбільш близькі.

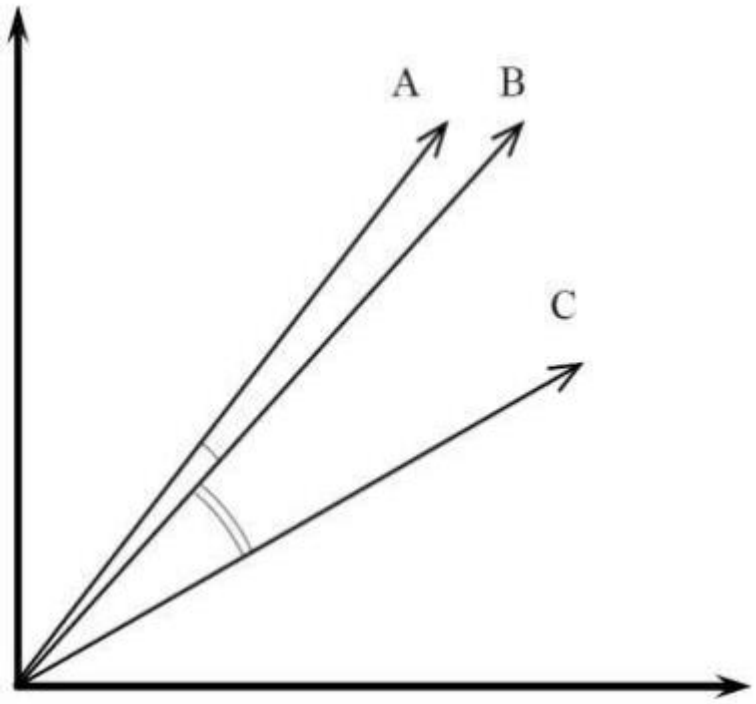

Рисунок 2.2 — Приклад векторного простору: подібність між векторами.

Наприклад, вектори A і B нагадують один одного більше, ніж вектори B і C. І, звичайно, вектори A і C найменш схожі один на одного. Маючи три вектори, ми можемо порівняти три пари векторів загалом. У просторі з більшою кількістю вимірів важче уявити цю подібність, тому краще розглядати приклади у двовимірному просторі, беручи до уваги, що в просторі з більшою кількістю вимірів принципи були б такі самі.

Отже, надалі у розробці ми будемо використовувати модель векторного простору, яка найбільше задовольняє наші потреби.

2.3 TF-IDF показник

У попередньому підрозділи ми визначилися з тим, що будемо використовувати просторову векторну модель. Але комп’ютер нас не зрозуміє, якщо ми накладемо слова просто на вектор, потрібна якась міра, яка б задовольнила б наші потреби. Було обрано TF-IDF.

TF-IDF або TFIDF, скорочення від терміну „частота — зворотна частота документа”, — це числова статистична величина, яка призначена для відображення того, наскільки важливим є слово для документа чи колекції документів. Часто використовується як ваговий фактор при пошуку інформації, та моделюванні інформації користувачами [24]. Ця вага є статистичним показником, який використовується для оцінки того, наскільки важливим є слово для документа в колекції чи корпусі. Важливість зростає пропорційно кількості випадків, коли слово з'являється в документі, але компенсується частотою слова у корпусі.

TF-IDF — одна з найпопулярніших схем зважування термінів сьогодні. Опитування, проведене в 2015 році, показало, що 83% текстових систем рекомендацій у цифрових бібліотеках використовують TF-IDF. Цей показник може використовуватися для аналізу контенту різних текстових файлів та пошуку плагіату у них.

Варіації схеми зважування TF-IDF часто використовуються пошуковими системами як центральний інструмент підрахунку та ранжування відповідності документа з урахуванням запиту користувача. TF-IDF можна успішно використовувати для фільтрації стоп-слів у різних тематичних полях, включаючи узагальнення та класифікацію тексту.

Частота документа (TF) — вага терміна, який зустрічається в документі, просто пропорційний частоті термінів. Припустимо, у нас є набір текстових документів, і ми хочемо класифікувати їх за тим документом, який більше відповідає запиту, "на вулиці сонячно". Простий спосіб — це виключити документи, які не містять усіх трьох слів "на", "вулиці" і "сонячно", але це все одно залишає багато документів. Для подальшого їх розрізнення ми можемо підрахувати кількість випадків, коли кожен термін зустрічається в кожному документі. Кількість випадків, коли термін зустрічається в документі, називається частотою його терміну. Однак у випадку, коли обсяг документів сильно варіюється, часто вносяться корективи. Вперше це пояснив Ганс Пітер Лун (1957).

Зворотна частота документа — інверсія частоти, з якою деяке слово

зустрічається в інших документах. Оскільки термін "на" є настільки поширеним, частота термінів, як правило, неправильно наголошує на документах, які частіше використовують слово "на", не надаючи достатньої ваги більш значущим термінам "вулиці" і "сонячно". Термін "на" не є гарним ключовим словом для розрізнення відповідних не релевантних документів та термінів, на відміну від менш поширених слів "вулиця " та "сонячно". Отже, включений зворотний коефіцієнт частоти документа, який зменшує вагу термінів, які дуже часто трапляються в наборі документів, і збільшує вагу термінів, які трапляються рідко.

Карен Спарк Джонс (Karen Spärck Jones, 1972) задумала статистичну інтерпретацію специфіки термінів, звану зворотною частотою документів IDF). Специфічність терміна може бути визначена кількісно як обернена функція кількості документів, в яких він зустрічається.

TF-IDF є добутком двох статистичних даних. Існують різні способи визначення точних значень обох статистичних даних. Формула, яка має на меті визначити важливість ключового слова чи фрази в документі.

У випадку терміна частота TF (t, d), найпростішим вибором є використання необробленої кількості терміна в документі, тобто кількість разів, коли термін t зустрічається в документі d. Якщо ми позначимо необроблений підрахунок через $f_{t.d}$ то найпростіша схема обрахунку TF записується формулою 2.1.

$$\text{TF}\ (t, d) = f_{f,d} \tag{2.1}$$

Також є булеві "частоти", які обраховується за формулою 2.2.

$$\text{TF}\ (t, d) = 1 \tag{2.2}$$

Якщо t виникає через d, а 0 — інакше.

Періодичність терміна з урахуванням тривалості документа обчислюється за формулою 2.3.

$$\text{TF}\ (t, d) = \frac{f_{t,d}}{\text{кількість слів у } d} \tag{2.3}$$

Логарифмічно масштабована частота обраховується за формулою 2.4.

$$\text{TF}\ (t,d) = \log(1 + f_{f,d}) \tag{2.4}$$

Збільшена частота, щоб запобігти упередженюю щодо довших документів, наприклад необроблена частота, поділена на вихідну частоту найпоширенішого терміна в документі (формула 2.5).

$$\text{TF}\ (t,d) = 0.5 + 0.5\,\frac{f_{t,d}}{\max\{f_{t,d}{:}t` \in d\}} \tag{2.5}$$

Зворотна частота документа (IDF) — це показник того, скільки інформації надає слово, тобто якщо воно загальне або рідкісне для всіх документів (формула 2.6). Це логарифмічно масштабована зворотна частка документів, яка містить слово (отримана шляхом ділення загальної кількості документів на кількість документів, що містять термін, а потім прийняття логарифму цього коефіцієнта)

$$idf(t,D) = log\,\frac{N}{|\{\text{d} \in \text{D:t} \in \text{d}\}|}\,, \tag{2.6}$$

де N: загальна кількість документів у корпусі N=|D|;

D|{d ∈ : t ∈ d}| : кількість документів, де з’являється термін (tf(t,d) ≠ 0).

Якщо термін відсутній у корпусі, це призведе до поділу на нуль. Тому загальноприйнятим є налаштування знаменника на $1 + |\{\text{d} \in \text{D} : \text{t} \in \text{d}\}|$.

Тоді TF-IDF обчислюється за формулою 2.7 (приклад обрахованої таблиці забражено на рисунку 2.3) :

$$tfidf(t,d,D) = tf(t,d)idf(t,D) \tag{2.7}$$

Висока вага у TF-IDF досягається високою періодичністю терміну (у даному документі) та низькою періодичністю терміну в усій колекції документів; ваги, отже, мають тенденцію відфільтровувати загальні терміни. Оскільки відношення всередині функції журналу IDF завжди більше або дорівнює 1, значення IDF (і TF-IDF) більше або дорівнює 0. Оскільки термін з’являється в більшості документів, співвідношення всередині логарифму наближається до 1 , наближаючи IDF та TF-IDF до 0.

Ідея, що лежить в основі TF-IDF, також стосується сутностей, відмінних від термінів. У 1998 р. Поняття IDF було застосовано до цитат.

	Document Keyword	tf						idf	wdt=tf*idf				
		d	d_1	d_2	d_3	d_4	df	log(n/df)	d	d_1	d_2	d_3	d_4
k_1	google	3	0	0	2	0	2	2.7	8.1	0	0	5.4	0
k_2	*pasar* (market)	1	2	1	1	0	4	2.4	2.4	4.8	2.4	2.4	0
k_3	browsing	2	0	0	3	0	2	2.7	5.4	0	0	8.1	0
k_4	online	1	0	0	4	0	2	2.7	2.7	0	0	10.8	0
k_5	*iklan* (ad)	3	1	0	0	0	2	2.7	8.1	2.7	0	0	0
k_6	network	1	0	0	1	2	3	2.52	2.52	0	0	2.52	5.04
k_7	user	1	0	0	0	0	1	2.7	2.7	0	0	0	0
k_8	facebook	2	0	0	1	0	2	2.7	5.4	0	0	2.7	0
k_9	mobile	4	0	0	3	0	2	2.7	10.8	0	0	8.1	0
k_{10}	sosial	1	1	0	0	1	3	3	3	3	0	0	3

Рисунок 2.3 — Приклад розрахунку таблиці TF-IDF

Автори стверджували, що "якщо дуже рідкісне цитування поділяється двома документами, це повинно зважуватися вище, ніж цитування, зроблене великою кількістю документів". Крім того, TF-IDF застосовувалося до «візуальних слів» з метою проведення збігу об’єктів у відео, та цілих реченнях. Однак концепція TF-IDF не виявилася ефективнішою у всіх випадках, ніж звичайна схема TF (без idf). Коли TF-IDF застосовували до цитат, дослідники не могли знайти покращення порівняно з простою вагою підрахунку цитат, яка не мала компонента IDF.

Ряд схем зважування термінів отримано з TF-IDF. Одним із них є TF-PDF (Термінова частота * Пропорційна частота документів).T-FPDF було введено в 2001 році в контексті виявлення нових тем у ЗМІ. Компонент PDF вимірює різницю між тим, як часто термін зустрічається в різних доменах. Іншим похідним є TF-IDuF. У TF-IDuF idf не обчислюється на основі корпусу документів, який потрібно здійснити пошук або рекомендувати. Натомість idf обчислюється на основі колекцій особистих документів користувачів. Автори повідомляють, що TF-IDuF був настільки ж ефективним, як TF-IDF, але також може застосовуватися в ситуаціях, коли, наприклад, система моделювання користувачів не має доступу до глобального корпусу документів.

Основними перевагами TF-IDF є:

— врахування не тільки конкретного документа, що містить необхідний терм, а й інших документів колекції;

— присвоюванням широко поширеним словам низької ваги;

— простота обчислення.

Основними недоліками TF-IDF можна назвати:

— неврахування взаємного розташування слів;

— перевантажений ключовими словами документ, але який не містить корисної для користувача інформації, буде завжди вище інших.

2.4 N-грам

Отже, об’єктами, які ми розглядаємо, є документи (тексти). Слова (терміни) є особливостями цих документів; кожне слово має значення TF-IDF для кожного документа. Це означає, що кожен документ відповідає вектору значень TF-IDF слів. Ми також можемо представити всю цю інформацію як матрицю. Ця матриця називається “матрицею термінів документів”. Приклад представлений на рисунку 2.4

	Doc_1	Doc_2	Doc_3	Doc_4
$Word_1$	0.3	0	0	0.02
$Word_2$	0.7	0.01	0	0.5
$Word_3$	0	0	0	0
$Word_4$	0	0	2.2	0

Рисунок 2.4 — Матриця термінів документів

Документи Doc1 ... DocN представляють документи в нашій колекції, слова Word1 ... WordM представляють усі слова, що містяться в цих документах; слова упорядковані певним чином. Значення в таблиці відповідають передбачуваним значенням TF-IDF у колекції.

Потрібно звернути увагу, що при використанні слів як ознак інформація про синтаксичні відношення між словами втрачається. Слова стають тим, що називається «мішком слів». Однак для багатьох завдань така втрата інформації є прийнятною. Це є одним з недоліків використання TF-IDF. Для подолання цього пропоную використовувати метод N-грами.

Традиційні n-грами — це послідовності елементів, як вони з’являються в документі [25]. У цьому випадку буква N вказує, скільки елементів слід врахувати, тобто довжину послідовності або N-грама. Наприклад, існують біграми (2 грами), триграми (3 грами), 4 грами, 5 грамів тощо.

Таким чином, якщо ми говоримо про уніграми, тобто N-грами, побудовані з одного елемента, це те саме, що говорити про слова або букви. Навіть створення уніграм, має певний сенс. Дослідження появи одного елементарного елемента в наборі таких самих елементів, може нести важливе статистичне навантаження. Для прикладу на рисунку 2.5 зображено частотний розподіл літер у англійській мові. Одна літра відповідає конкретній уніграмі, відсоток від загальної кількості унаграм, вказує на ймовірність появи конкретної уніграми в документах.

Існують різні типи елементів, що утворюють n-грами. Ці елементи можуть бути лемами або словами; вони також можуть бути частиною мовних тегів (POS-тегів), таких як іменники, дієслова тощо. Теги можуть бути більш детальними, тобто включати граматичні особливості, наприклад, мітку VIP1S може означати "дієслово, вказівне, теперішнє, від першої особи, однина". Ми можемо побудувати N-грами за допомогою такого роду тегів. В останні роки N-грами символів (послідовності символів, взяті з тексту) використовуються в різних різних завданнях.

Для деяких завдань, таких як приписування авторства, N-грами символів дають досить хороші результат. Їх лінгвістичне тлумачення недостатньо чітке і залишається відкритим питанням. Подивимось приклад традиційних N-грамів слів. Для речення:This is a sentence, ми можемо отримати різні N-грами (рисунок 2.6) Ми можемо замінити кожне слово на лему або частину мови та побудувати відповідні N-грами.

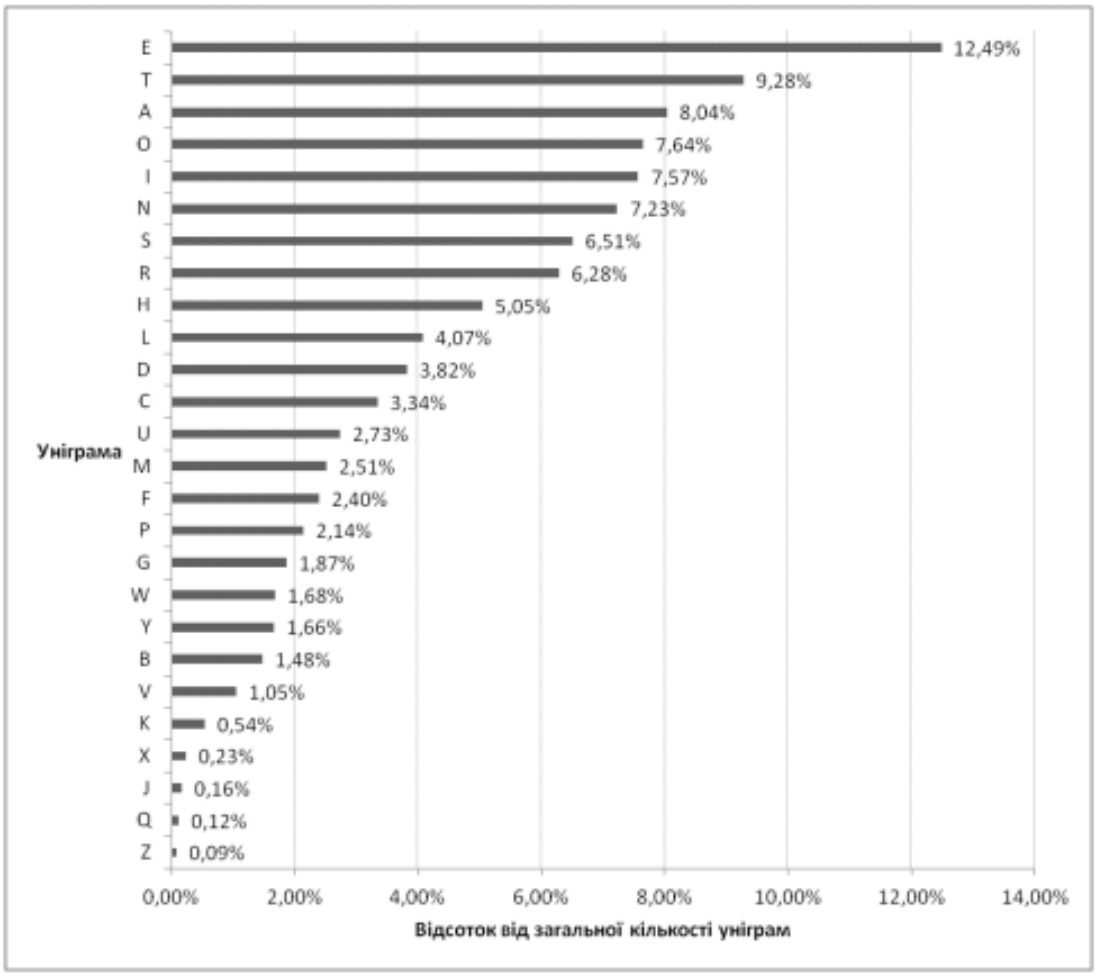

Рисунок 2.5 — Частотний розподіл уніграм літер

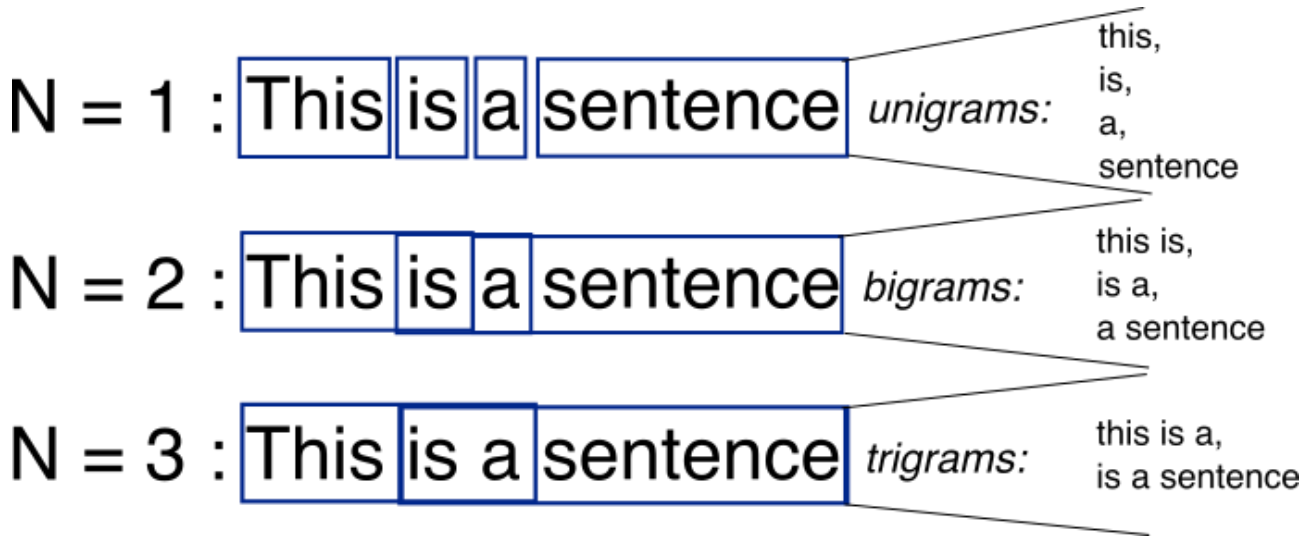

Рисунок 2.6 — N-грами речення

Як бачимо, процес є дуже простий, але він успішно використовується в обчислювальних лінгвістичних системах. Якщо N-грами використовуються як

ознаки, які значення вони можуть мати? Як і у випадку зі словами (уніграмами), це значення, пов'язані з їх TF-IDF. Зауважте, що частоти N-грамів, як правило, набагато нижчі, ніж частоти слів, тобто N-грамів відображаються набагато менше в тексті. Це логічно, оскільки ми фактично спостерігаємо появу послідовностей двох або більше слів разом, що є набагато менш імовірною подією, ніж одне слово. Таким чином, для того, щоб застосувати модель векторного простору до текстів, ми можемо використовувати N-грами як ознаки. Ці N-грами можуть бути різного розміру, складатися з елементів різного типу, і їх значеннями можуть бути частоти TF-IDF .

Типовий графік частоти N-грам технічного документа зображено на рисунку 2.7

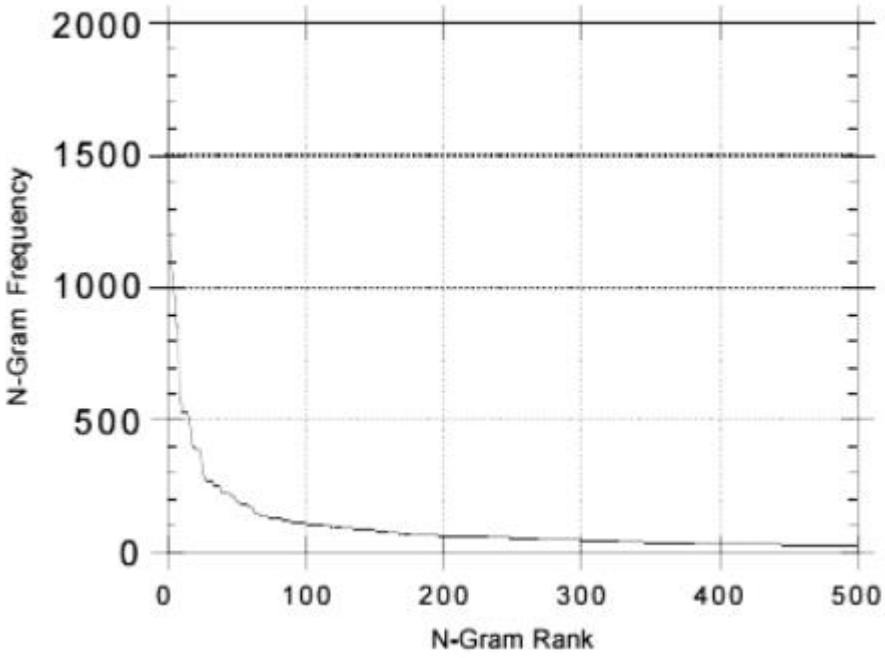

Рисунок 2.7 — Частоти N-грам у технічному документі

Як можемо бачити перші N-грами можна не брати до уваги, оскільки вони дуже часто повторюються і не несуть ніякої інформативності. Саме для цього ми будемо використовувати TF-IDF, в якості статистичної міри. TF-IDF ця проблема вирішена.

2.5 Визначення подібності

Після того, як ми виділили з тексту необхідні нам n-грами, обрахували статистичні показники та нанесли все це вектор необхідно визначити подібність двох документів. Далі ми розглянемо три основних прийоми характерних для моделей векторного простору.

Передгільбертів простір — це перша техніка векторного просторового простору. Ця техніка розглядається як основа для інших технік [26]. Всі інші методи залежать від результатів цієї техніки для обчислення результатів їх функцій. Ідея тут полягає в тому, щоб реалізувати кожен документ у системі пошуку і запит як вектори та знайти схожість між цими документами та запитом (рисунок 2.8). Вектор документів, подібних за напрямком до вектора запиту, буде вважатися родичами запиту і, сподіваємось, задовольняти потреби користувача.

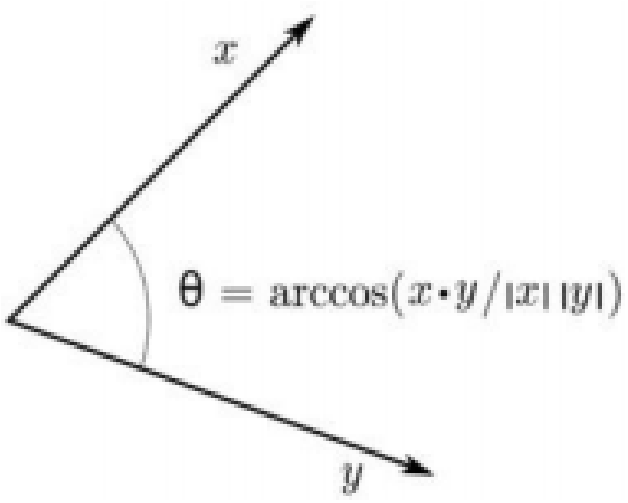

Рисунок 2.8 — Інтерпритація передгільбертового простору

Індекс Джакарта — також відомий коефіцієнт схожості Джакара (спочатку французька назва коефіцієнт комунікації Полом Джакартом), є статистичною мірою, яка використовується для оцінки подібності та різноманітності наборів вибірки. Коефіцієнт Джакарта вимірює схожість між кінцевими множинами вибірки і визначається як розмір перетину, поділений на величину об'єднання множин вибірки (формула 2.8).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |AB \cap|} \quad (2.8)$$

Якщо A і B порожні, то

$$J(\mathrm{A},\mathrm{B}) = 1$$

Індекс Джакарта може приймати значення від 0 до 1 включно. Альтернативна інтерпретація відстані Джакарта — це відношення величини симетричної різниці (формула 2.9).

$$A \Delta B = (A \cup B) - (A \cap B) \quad (2.9)$$

Відстань Джакарта зазвичай використовується для обчислення матриці n × n для кластеризації та багатовимірного масштабування n наборів вибірки.

Ця відстань є метрикою для збору всіх скінченних множин. Існує також версія відстані Джакарта для вимірювань, включаючи ймовірні виміри. Якщо є мірою на вимірюваному просторі, тоді ми визначаємо коефіцієнт і відстань Джакарта за формулами 2.10 та 2.11.

$$J_\mu(A,B) = \frac{\mu(A \cap B)}{\mu(A \cup B)} \quad (2.10)$$

$$d_\mu(A,B) = 1 - J_\mu(A,B) = \frac{\mu(A \Delta B)}{\mu(A \cup B)} \quad (2.11)$$

Подібність косинусів між векторами — ще один з приймів векторного простору.

Для того, щоб виразити подібність, ми використовуємо косинусну міру кута між векторами: чим гостріший кут, тим більший косинус, тобто чим більша подібність між векторами, і, отже, порівняні об'єкти більш схожі [27].

Для обчислення подібності косинусів між двома векторами V і U використовується внутрішній добуток (крапковий добуток) нормованих векторів. Нормалізація полягає в діленні результату на довжину кожного вектора або, що еквівалентно, у множенні їх довжини. Довжину називають евклідовою нормою і позначають, наприклад, ‖V‖ для вектора V.

Точковий добуток двох векторів — це величина, яку легко отримати:

множення векторних значень у кожному вимірі підсумовуються. У двох вимірах (розмірність 1 та розмірність 2) для векторів U та V обчислюється за формулою 2.12.

$$\text{Добуток} = V_1 \cdot U_1 + V_2 \cdot U_2 \quad (2.12)$$

Тобто перші елементи векторів множаться між собою, потім те саме з іншими елементами векторів, а в кінці — підсумовуються добутки. Більш загально це можна побачити у формулі 2.13.

$$\text{Добуток}(V, U) = \sum_{n=1}^{m}(V_n \cdot U_n), \quad (2.13)$$

де m — кількість розмірностей у векторній просторовій моделі (яка дорівнює довжині векторів).

Як уже зазначалося, нормалізація векторів — це другий крок при розрахунку косинуса (перший — обчислення крапкового добутку). З цією метою слід обчислити довжину векторів та отримати евклідову норму. Норма Евкліда перетворює кожен вектор в одиничний вектор (вектор довжиною 1). Нормалізація за допомогою евклідової норми ‖V‖ є діленням вектора на його довжину. Довжина вектора обчислюється наступним чином:

$$||V|| = \sqrt{\sum_{k=1}^{m} V_k^2} \quad (2.14)$$

У випадку двох вимірів це відповідає застосуванню теореми Піфагора і обраховується за формулою 2.15.

$$\left|||V|\right| = \sqrt{V_1^2 + V_2^2}, \quad (2.15)$$

де V1 і V2 — значення вектора V на осі 1 і 2.

Власне, V1 — значення вектора в розмірності 1, тобто довжина першого катета, а V2 — значення вектора в розмірність 2, тобто довжина другого катета. У цьому випадку сам вектор відповідає гіпотенузі, і застосовується теорема Піфагора. У разі більшої кількості розмірів, відповідні елементи додаються до формули таким же чином.

Таким чином, остаточна формула для обчислення подібності косинусів полягає у отриманні крапкового добутку двох векторів та застосуванні норми Евкліда (формула 2.16).

$$sim(V,U) = \frac{\sum_{n=1}^{m}(V_n \cdot U_n)}{||V|| \cdot ||U||} \quad (2.16)$$

У цьому випадку подібність косинусів вказує, наскільки вектори V і U схожі. Для позитивних значень косинус коливається в межах від 0 до 1. Зверніть увагу, що подібність косинусів визначається рівно для двох об’єктів (двох векторів) [28].

Подібність об’єкта із самим собою буде дорівнювати 1. Для об’єктів, які відповідають ортогональним векторам (тобто кут між ними дорівнює 90°), подібність косинусів дорівнює 0.

Отже, у розділі було розроблено теоретико-математичну модель системи. Спочатку текст необхідно токенізувати, оскільки є фрагменти, які не несуть інформаційного навантаження. Далі було обрано векторну просторову модель, яка може застосовуватися для будь-якого типу об’єктів. Ця модель полягає у виборі ознак та присвоєнні значень цим ознакам, що дозволяє представити наші тексти як вектори. У якості статистичного показника обрано TF-IDF, оскільки надає оцінку важливості слова у документі. Щоб зберегти порядок слів було вирішено використовути N-грами. Для вимірювання схожості векторів обрано косиносову подібність.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КОНТРОЛЮ КОНТЕНТУ

3.1 Вибір мови програмування

На сьогоднішній день у світі існує різноманітна кількість мов програмування. Вони надають програмісту різноманітні можливості.

У всіх мов програмування наявні свої особливості. Певні мови програмування краще використовувати для вирішенні одних цілей, а деякі — для інших. Ці особливості потрібно враховувати на етапі проектування.

Для вирішення нашого завдання, доцільно є застосування мови, яка є простою у використанні, дозволяє швидко створювати необхідні нам компоненти і не потребує написання великих шматків громісткого коду. Також програми розроблені на обраній мові мають чудово працювати на цільовій операційний системі — у нашому випадку Windows [29].

C# — це об'єктно-орієнтована мова програмування. C# був розроблений завдяки зусиллям Андерса Хейлсберга, творця компілятора, який склав основу для Turbo Pascal та мови програмування Delphi. Перша версія мови була випущена в червні 2000 р. (Можливо, Microsoft хотіла відзначити нове тисячоліття), а остаточна версія випущена в 2002 р. Разом із Visual Studio. Зараз C # став однією з найпопулярніших мов програмування, навіть трохи випередивши свого попередника.Спочатку Microsoft збиралася випустити власну версію мови Java (Microsoft Java або J ++), але на них подати до суду власники авторських прав (Sun Microsystems) через деякі суперечливі моменти. Тому керівництво вирішило створити власну мову, яка відповідала б їхнім вимогам і розробкою якої вони могли контролювати. Так з’явився C#.

Говорити про будь-яку філософію мови, коли мова заходить про C #, важко. Справа в тому, що з самого початку мова не була відкритою, оскільки вона була створена спеціально для однієї відомої корпорації. Багато понять і конструкцій були запозичені з інших мов, таких як C, C++, Java тощо (перші версії мови були дуже схожі на Java, хоча зараз C # більше не можна вважати просто клоном цієї мови).

C# — це мова із C-подібним синтаксисом. Тут він близький у цьому відношенні до C ++ та Java.

Як об’єктно-орієнтована мова, вона багато чому навчилася з Java та C ++. Як і Java, C# спочатку був розроблений для веб-розробки, і приблизно 75% можливостей його синтаксису збігаються з Java. C# також називають "очищеною версією Java". Ще 10% запозичили у ++, а 5% — у Visual Basic. Решта 10% C# — це реалізація власних ідей розробників. Об'єктно—орієнтований підхід дозволяє використовувати C# для створення великих, але в той же час гнучких, масштабованих та розширюваних додатків.

C# давно підтримує безліч корисних функцій:

— інкапсуляція;

— спадковість;

— поліморфізм;

— перевантаження оператора;

— статична типізація.

Водночас вона все ще активно розвивається, і з кожною новою версією стає все цікавіше.

C # має безліч переваг:

— підтримка переважної більшості продуктів Microsoft;

— безкоштовність ряду інструментів для невеликих компаній та окремих розробників - Visual Studio, хмара Azure, Windows Server, Parallels Desktop для Mac Pro тощо;

— типи даних мають фіксований розмір (32-бітний int і 64-бітний довгий), що збільшує "переносимість" мови та спрощує програмування, оскільки ви завжди точно знаєте, з чим маєте справу;

— автоматичний збір сміття — це означає, що в більшості випадків нам не доведеться турбуватися про звільнення пам'яті, CLR сам викличе збирач сміття та очистить пам’ять;

— велика кількість "синтаксичного" цукру " — спеціальних конструкцій, призначених для розуміння та написання коду, вони не мають значення при

компіляції;

— низький поріг входу — синтаксис C# має багато схожості з іншими мовами програмування, що полегшує перехід для програмістів, мова C # часто вважається найбільш зрозумілою та придатною для початківців;

— за допомогою Xamarin ви можете писати програми та програми на C # для таких операційних систем, як iOS, Android, MacOS та Linux;

Але C # також має деякі недоліки:

— пріоритетний фокус на платформі Windows;

— мова безкоштовна лише для невеликих фірм, окремих програмістів, стартапів та студентів, для великої компанії придбання ліцензійної версії цієї мови буде коштувати круглу суму.

З огляду на все вище сказане, було вирішено використовувати C# для подальшої роботи.

3.2 Архітектура .NET

.NET Framework — це програмне забезпечення, розроблене корпорацією Майкрософт, яке працює в основному на Microsoft Windows. Він включає велику бібліотеку класів під назвою Framework Class Library (FCL) і забезпечує взаємодію мови (кожна мова може використовувати код, написаний іншими мовами) для декількох мов програмування. Програми, написані для .NET Framework, виконуються в програмному середовищі (на відміну від апаратного середовища) з назвою Common Language Runtime (CLR). CLR — це віртуальна машина програми, яка надає такі послуги, як безпека, управління пам’яттю та обробка винятків. Таким чином, комп'ютерний код, написаний за допомогою .NET Framework, називається "керованим кодом". FCL та CLR разом складають .NET Framework.

.Net це один з надзвичайно важливих програмних засобів доступних сьогодні розробникам. Цей потужний фреймворк використовує такі популярні мови програмування, як C #, Visual Basic та інші, для створення високоякісних, надскладних веб та настільних додатків для корпоративного використання.

Структура .Net є одним із найдавніших інструментів програмування, що використовується сьогодні. Це інфраструктура програмування, яка використовує популярні мови, такі як C # та Visual Basic, для написання коду для настільних та веб-додатків корпоративного типу. Розробка фреймворку розпочалася наприкінці 1990-х років, коли Microsoft вирішила розробити його як частину своєї більшої ".Net Strategy". Це передбачало продаж серії комп'ютерних товарів із фірмовою мережею .Net. Перший прототип .Net під назвою .Net 1.0 був випущений наприкінці 2000 року.

.Net працює відповідно до парадигми об'єктно-орієнтованого програмування (ООП). Підтримувана Microsoft фреймворк, поряд з Python, Ruby та C ++, використовує ООП для спрощення процесу розробки. За цим методом об'єкти використовуються для компартменталізації даних. З іншого боку, декларації класів використовуються для опису вмісту та дій об'єкта.

Цей підхід до програмування також використовує абстракцію, інкапсуляцію, поліморфізм та приховування даних для вирішення багатьох проблем, пов'язаних із процедурним програмуванням, головним конкурентом ООП.

Архітектура .NET Framework описана і опублікована в специфікації Common Language Infrastructure (CLI), яка розроблена Microsoft. У CLI описані типи даних .NET, формат метаданих, структура програм та інше.

Важливо розуміти, що не всі мови програмування .NET повинні підтримувати всі типи даних, визначені в CLI. Специфікація загальної мови (CLS) встановлює основні правила, що визначають закони, яким повинні керуватися всі мови: ключові слова, типи, примітивні типи, перевантаження методів тощо. Специфікація CLS визначає мінімальні вимоги до мови платформи. НЕМАЄ. Компілятори, які відповідають цій специфікації, створюють об'єкти, які можуть взаємодіяти між собою. Будь—яка мова, сумісна з CLS, може повною мірою скористатися перевагами FCL (Бібліотека класів платформ).

.NET є багаторівневим, модульним і ієрархічним. Кожен рівень .NET Framework є шаром абстракції. Мови .NET — це найвищий і найбільш

абстрагований рівень серед інших. Common Language Runtime (CLR) - це нижній рівень, найменш абстрагований і найбільш близький до вихідного середовища. .NET Framework розділений на модулі, кожен з яких має свою особливу сферу відповідальності. Через те що вищі рівні надсилають запити до служби тільки з більш низьких рівнів, .NET є ієрархічним. Загальна архітектура .NET Framework наведена на рисунку 3.1.

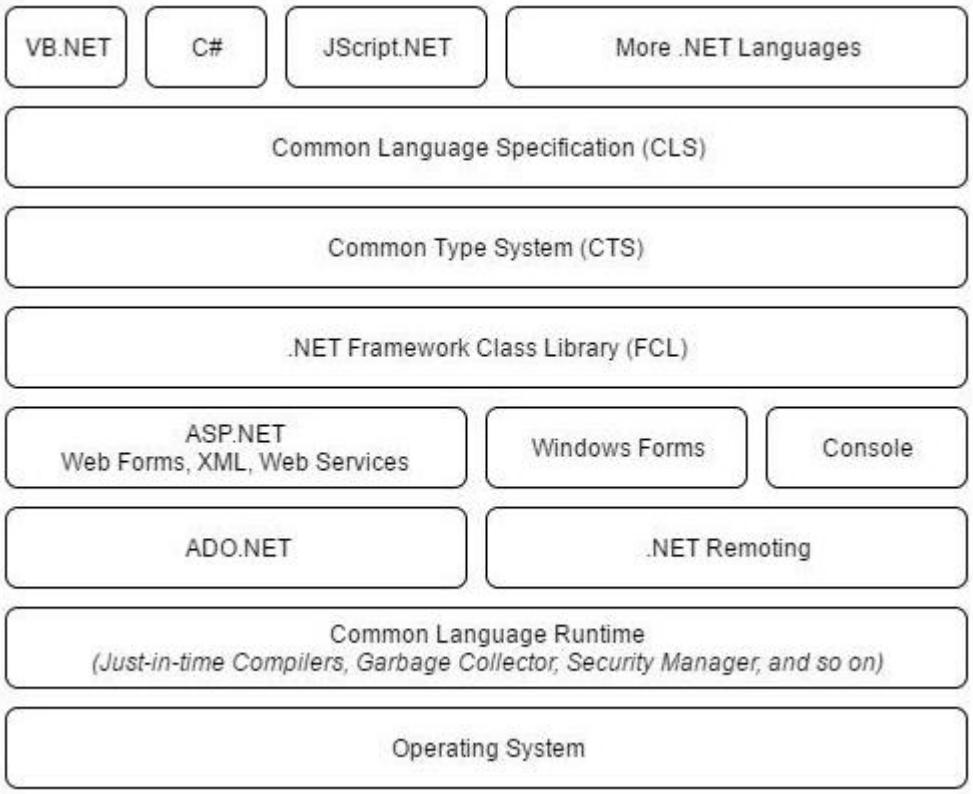

Рисунок 3.1 — Архітектура .NET

Об'єктні класи .NET, доступні для всіх підтримуваних мов програмування, містяться в бібліотеці Framework Class Library (FCL). У FCL входять класи Windows Forms , ASP.NET, ADO.NET, Windows Presentation Foundation, Language Integrated Query, Windows Communication Foundation та інші (рисунок 3.2).

3.3 Середовище розробки

Для розробки було прийнято рішення використовувати Microsoft Visual Studio.

Microsoft Visual Studio — це лінійка програмних продуктів Microsoft, яка

включає інтегроване середовище розробки (IDE) програмного забезпечення, а також інші засоби розробки.

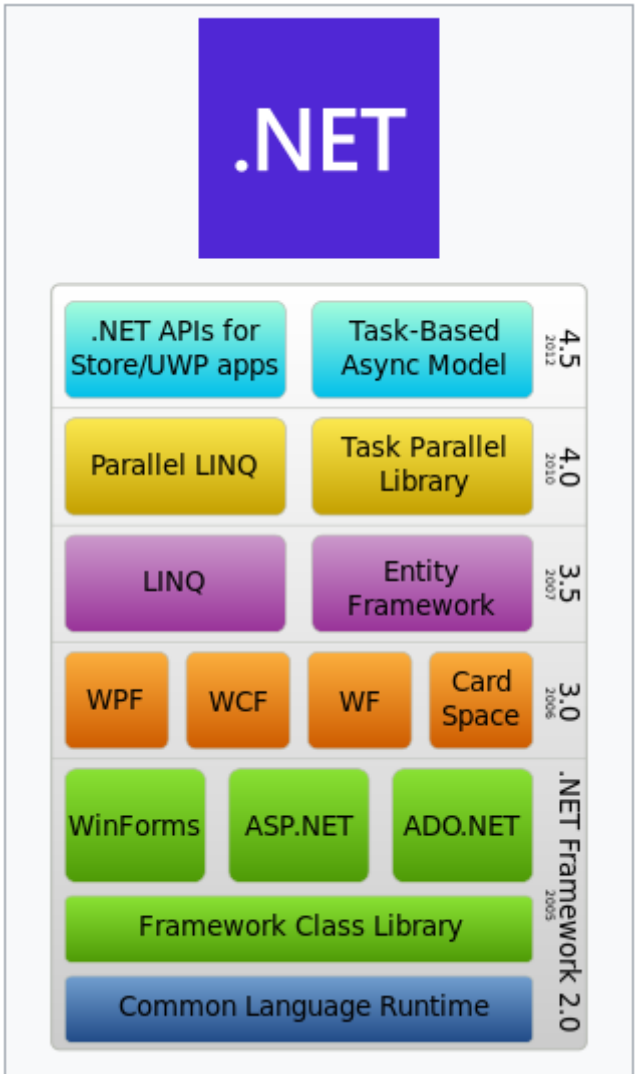

Рисунок 3.2 — Стек .NET

IDE — це багатофункціональна програма, яка може бути використана для багатьох аспектів розробки програмного забезпечення. Більш стандартний редактор та налагоджувач, який надає більшість середовищ розробки, Visual Studio включає компілятори, інструменти для завершення коду, графічні розробники та багато інших функцій, які значно полегшують роботу розробника.

Visual Studio включає редактор вихідного коду, який підримує IntelliSense (підказки в реальному часі), а також можливість серйозного рефакторингу (англ. Refactoring) коду. Вбудовані компоненти налагодження надають можливість працювати як налагодження на рівні джерела, так і як налагодження на рівні

операційної системи. Усі інші вбудовані інструменти складаються з редактора форм для спрощення розробки графічного інтерфейсу програми, веб—редактора, конструктора структури класів та схем баз даних. Visual Studio дозволяє розробляти та підключати сторонні програми (плагіни) для додавання функціональності на всіх рівнях, включаючи підтримку систем контролю версій вихідного коду, таких як Subversion і Visual SourceSafe, Team Foundation Server, Git, а також додавати нові набори інструментів (приклад - редагування та візуальне оформлення коду мовами програмування для конкретних доменів).

На рисунку 3.3 показаний інтерфейс Visual Studio з відкритим проектом та декількома основними вікнами, які найчастіше використовуються, далі наведено опис основних вікон:

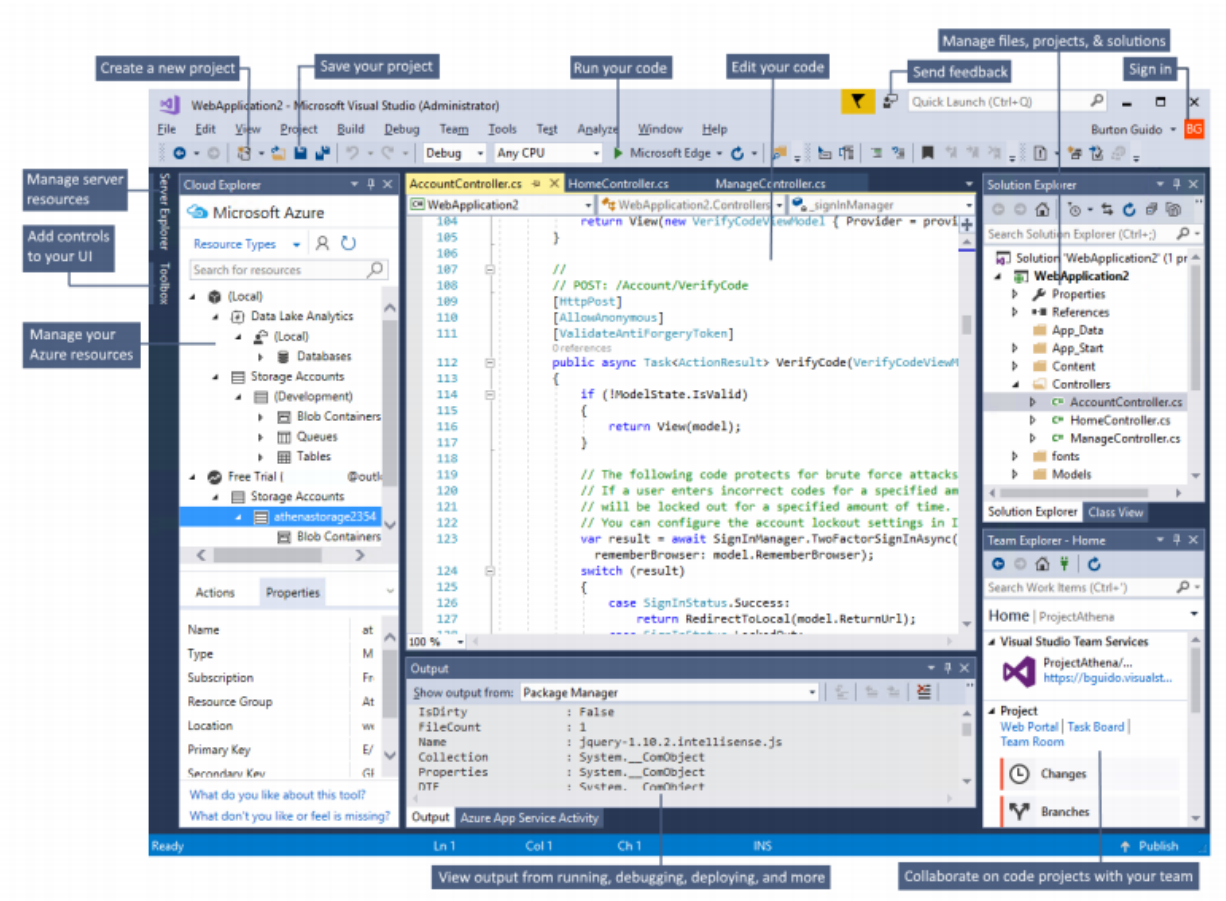

Рисунок 3.3 — Інтерфейс Visual Studio

— вікно Провідника рішень (угорі праворуч) дозволяє переглядати, переміщатися та керувати кодами файлів, провідник рішень може допомогти впорядкувати ваш код, комбінуючи файли між рішеннями та проектами;

— вікно редактора (в центрі) відображає вміст файлу, тут ви можете редагувати код або розробляти користувальницький інтерфейс, наприклад вікно з кнопками та текстовими полями;

— вікно виводу (внизу по центру) — це те місце, де Visual Studio повідомляє, наприклад, про налаштування та помилки, попередження компілятора, опублікування повідомлень про стан тощо, кожне джерело повідомлень має власну вкладку;

— вікно Team Explorer (внизу праворуч) дозволяє відстежувати робочі елементи та обмінюватися кодом з іншими користувачами за допомогою технологій контролю версій, таких як Git та Team Control Version Control (TFVC).

Існує три видання Visual Studio — Community, Professional та Enterprise.

3.4 Структура програми

Для досягнення поставлених цілей нам необхідно реалізувати наступні алгоритми (рисунок 3.4)

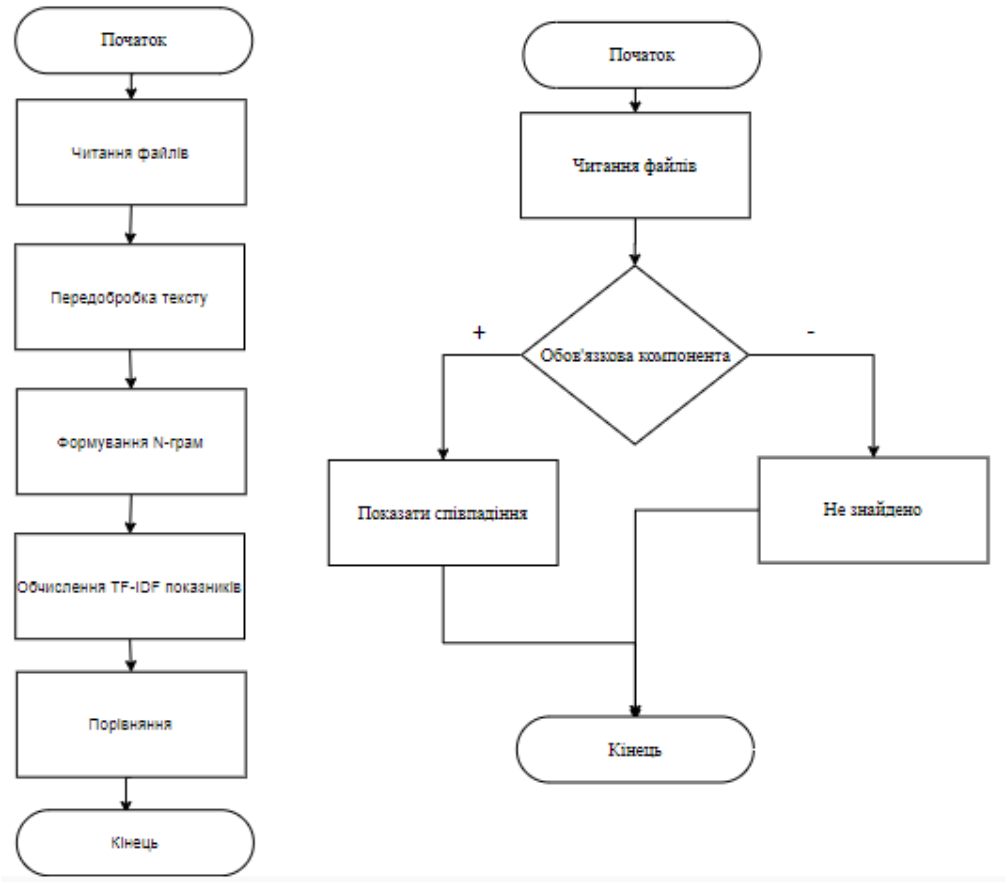

Рисунок 3.4 — Алгоритм роботи програми

Для цього потрібно реалізувати наступні модулі:

— читання документів — програма буде приймати на вхід файли формату PDF, у даному модулі має здійснюватися перед обробка файлу, яка буде включати видалення стоп-символі та стоп-слів, переведення всіх слів у нижній регістр;

— модуль поділу тексту на N-грами, розмір N-грами має задавати користувач;

— модуль обрахунку TF-IDF статистичних показників з подальшим занесенням їх на вектор;

— порівняння векторів і оцінка схожості;

— перевірка окремих компонент;

— графічний інтерфейс користувача.

3.5 Читання і аналіз документів

Для роботи з документами формату PDF було обрано використовувати бібліотеку iText.

iText — це бібліотека для створення та обробки PDF-файлів у Java та .NET.

iText написав Бруно Ловагі. Спочатку вихідний код розповсюджувався у відкритому коді за ліцензіями Mozilla Public Public або GNU. Але починаючи з версії 5.0.0, вона розповсюджується під загальною публічною ліцензією Affero версії 3. Форк ліцензованої версії iText LGPL / MPL в даний час активно підтримується як бібліотека OpenPDF. iText також доступний через власну ліцензію, яку розповсюджує iText Software NV. Проекти, які не бажали надавати свій вихідний код (як вимагає AGPL), могли або придбати комерційну ліцензію на iText 5 і вищу версію, або продовжувати використовувати попередні версії iText під MPL / LGPL.

iText забезпечує підтримку найбільш передових функцій PDF, таких як підписи на основі PKI, 40-бітове та 128-бітове шифрування, корекція кольорів, PDF з тегами, PDF-форми (AcroForms), PDF / X, управління кольором за допомогою профілів ICC та штрих-кодів.

Було розроблено бібліотеку класів Pdf.Parser. Вона призначена для зчитування і обробки документів у форматі PDF. Розроблено клас PdfParser. Алгоритм роботи зображено на рисунку 3.5.

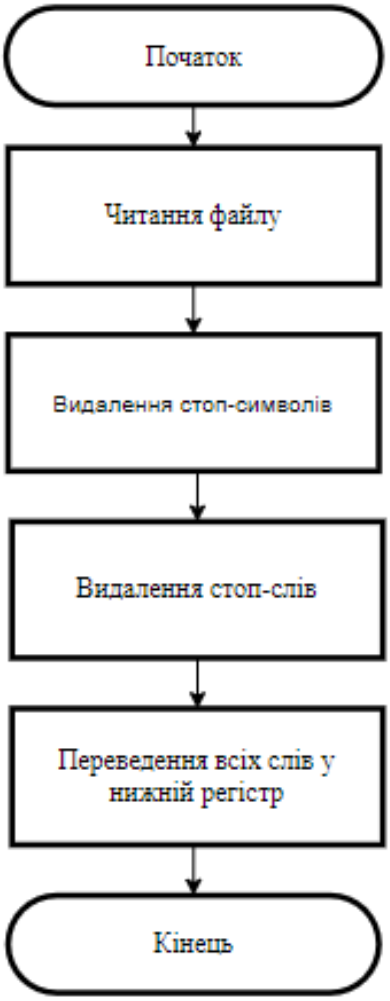

Рисунок 3.5 — Алгоритм роботи PdfParser

У класі реалізовано метод GetText, як параметр метод приймає адресу файлу, який потрібно прочитати і обробити.

Спочатку здійснюємо читання файлу.

```
using (var reader = new PdfReader(filePath))
        {
           var text = new StringBuilder();
           for (var i = 1; i <= reader.NumberOfPages; i++)
           {
```

```
            text.Append(PdfTextExtractor.GetTextFromPage(reader, i));
        }
```

Далі необхідно поєднати прочитані сторінки у одну текстову змінну.

```
var fileTextWithoutNewLine = new Regex("[\r\n]+").Replace(text.ToString(), " «)
```

Наступним важливим етапом є видалення стоп—символів, адже вони не несуть змістовного навантаження.

```
Var fileText = new Regex(«[0-9/|_!@#$%^&*()_+=?:;.,{}№><«»’\»`~» + @»\\[\]— -]* «).Replace(fileTextWithoutNewLine, « «);
```

Також необхідно видалити стоп—слова. Це, слова які, як і стоп-символи у більшості випадків не несуть змістовного навантаження. Для цього було створено масив стоп—слав. Його шматок та процес видалення стоп—слів наведено нижче.

```
Private static readonly string[] StopWords = {
        «I», «II», «III», «IIII»,» а «, « б «, « в «, « г «, « д «, « е «, « ё «, « ж «, « з «, « и «,  « й «, « к «, « л «, « м «, « н «, « о «,  «v», «w», «x», «y», «z»,»без», «біля», ««результаті», «ролі», «силу», «сторону», «супроводі», «ході», «верх», «вздовж», «уздовж», «вище»}
    foreach (string sub in StopWords)
            fileTextWithoutNewLine = fileTextWithoutNewLine.Replace(sub, « «);
```

Завершальний етап — переведення усього тексту у нижній регістр.

```
Var words = regex.Matches(fileText).Cast<Match>().Select(m => m.Value.ToLower());
```

3.6 Бібліотека класів для формування N-грам, підрахунку їх частот та формування документальної матриці

Дана бібліотека призначена для формування N-грам, підрахунку їх частот та формування документальної матриці на основі деякого текстового корпусу.

Теоретико-математична модель даного функціоналу була описана у другому розділі даної роботи.

Розглянемо більш детально створену бібліотеку. У бібліотеці реалізовано 3 класи Ngram, DocumentTerrn, Similarity. Розберемо детально кожен з них.

Клас Ngram — призначений для формування N-грам з вхідного тексту. Він містить один метод, метод на вхід приймає стрічку (вхідний текст документа) та розмір N-грами.

Спочатку відбувається перевірка чи не менша наша стрічка за розмір заданої N-грами. Якщо все добре продовжуємо обрахунки. Далі ми проходимо по всій стрічці формуючи N-грами згідно алгоритму. Повний код даного класу наведено нижче.

```
Public class Ngram
{
  public List<string> Create(string input, int n)
  {
    var nGrams = new List<string>();
    var words = input.Split(‘ ‘);
    if (words.Length <= n)
    {
      nGrams.Add(string.Join(« «, words));
      return nGrams;
    }
    for (var w = 0; w < words.Length; w++)
    {
      if (w + n > words.Length) break;
      var nGramElements = new List<string>();
      for (var i = w; i < w + n; i++)
      {
        nGramElements.Add(words[i]);
```

```
            }
            nGrams.Add(string.Join(« «, nGramElements));
        }
        return nGrams;
    }
}
```

Клас DocumentTerrn призначений для обрахунку статистичного показника TF-IDF.У даному класі реалізовано один метод Create. Як параметр даний метод отримує на вхід словник документів. Public Dictionary<string, List<string>> documents — документ та побудовані для нього відповідні N-грами.

```
Public class DocumentTermFrequency
    {
        public double[,] Create(Dictionary<string, List<string>> documents)
        {
            var allWords = new List<string>();
            var totalDocuments = documents.Keys.Count;
            int i = 0;
            foreach (var documentsValue in documents.Values)
            {
                var sw = new System.Diagnostics.Stopwatch();
                sw.Start();
                System.Diagnostics.Debug.Write($»Extracting document words for #{i++} ...»);
                allWords.AddRange(documentsValue);
                System.Diagnostics.Debug.WriteLine($»completed in {sw.ElapsedMilliseconds}ms.»);
            }
            var distinctTerms = allWords.Distinct().ToArray();
            var totalDistinctTerms = distinctTerms.Count();
```

```
            var tfIdfMatrix = new double[totalDocuments, totalDistinctTerms];
            for (var r = 0; r < totalDocuments; r++)
            {
               var sw = new System.Diagnostics.Stopwatch();
               sw.Start();
               System.Diagnostics.Debug.Write($»Processing document {r}...»);
               for (var c = 0; c < totalDistinctTerms; c++)
               {
                  var key = documents.Keys.ElementAt®;
                  var tf = (double) documents[key].Count(w => w == distinctTerms[c]);
                  tfIdfMatrix[r, c] = tf;
               }
               System.Diagnostics.Debug.WriteLine($»completed in {sw.ElapsedMilliseconds}ms.»);
            }
            return tfIdfMatrix;
         } }
```

На виході ми отримуємо матрицю обрахованих TF-IDF статистичних показників для документів.

Клас Similarity призначений для обчислення подібносте. У даному класі реалізовано два методи. CreateRowWise обчислює подібність на основі косинусної відстані, обрахування якої реалізовано у методі CosineSimilarity. Параметрами даного методу є двомірний масив обрахованих TF-IDF статистичних показників.

```
   Public double[,] CreateRowWise(double[,] matrix)
  {
     var similarityMatrix = new double[matrix.GetLength(0), matrix.GetLength(0)];
     for (var r1 = 0; r1 < matrix.GetLength(0); r1++)
     {
        for (var r2 = 0; r2 < matrix.GetLength(0); r2++)
```

```
        {
            if (r1 == r2)
            {
                similarityMatrix[r1, r2] = 0;
                continue;
            }
            var currentRow = new double[matrix.GetLength(1)];
            var nextRow = new double[matrix.GetLength(1)];
            for (var c = 0; c < matrix.GetLength(1); c++)
            {
                currentRow[c] = matrix[r1, c];
                nextRow[c] = matrix[r2, c];
            }
            var sim = CosineSimilarity(currentRow, nextRow);
            similarityMatrix[r1, r2] = sim;
        }
    }
    return similarityMatrix;
}
```

Як ми бачимо метод CosineSimilarity обраховує подібність косинусів за алгоритмом описаним у розділі 2. Метод приймає два параметри — два масиви, це наші два вектори для яких потрібно обрахувати подібність косинусів та повертає її.

```
Public double CosineSimilarity(double[] v1, double[] v2)
{
    var dotSum = 0.0;
    var v1SquaredSum = 0.0;
    var v2SquaredSum = 0.0;
    for (var i = 0; i < v1.Length; i++)
```

```
        {
            dotSum += v1[i] * v2[i];
            v1SquaredSum += v1[i] * v1[i];
            v2SquaredSum += v2[i] * v2[i];
        }
        return dotSum / (Math.Sqrt(v1SquaredSum) * Math.Sqrt(v2SquaredSum));
    }
```

3.7 Створення інтерфейсу користувача

Windows Forms — це API інтерфейс, який повністю відповідає віку .NET Framework. Порівняно з WPF, Windows Forms досить проста технологія, яка надає більшість можливостей, які необхідні для написання простого типового Windows-додатку. Вона також важлива для для підтимки успадкованих додатків. Однак Windows Forms свої недоліки, якщо порівнювати з WPF:

— розміри та позиція елементів задаються у пік селях, це призводить до великих ризиків некоректного відображення додатків на різних комп'ютерах — оскільки налаштування DPI відрізняється у звичайних користувачів і розробників;

— API—інтерфейс призначений для створення особливих елементів керування є GDI+, він досить хорошим, але швидкість візуалізації великих областей є низькою;

— проблемно домогтися надійності динамічного компонування.

Виходячи з вище сказаного краще використовувати WPF.

Технологія WPF з'явилася в .NET Framework 3.0 і призначена для розробки програмного забезпечення для товстих клієнтів. Розглянемо переваги WPF перед Windows Forms:

— підтримує вдосконалену графіку, включаючи довільні перетворення, тривимірну візуалізацію та справжню прозорість;

— основна одиниця виміру не базується на 57ормо кон, тому програми відображаються правильно при будь-якому налаштуванні DPI;

— він амає велику підтримку динамічного компонування, що означає можливість локалізації програми без ризику перекриття елементів;

— візуалізація використовує DirectX і є швидкою, виграючи від апаратного прискорення графіки;

— інтерфейси користувача можна декларативно описати у файлах XAML, які підтримуються незалежно від окремих файлів коду — це допомагає відокремити зовнішній вигляд від функціональних можливостей;

Типи для написання програм WPF є у просторі імен System.Windows та у всіх його підпросторах, крім System.Windows.Forms.,

Отже, для розмітки додатку було використано XAML.

Графічний інтерфейс зображено на рисунку 3.6.

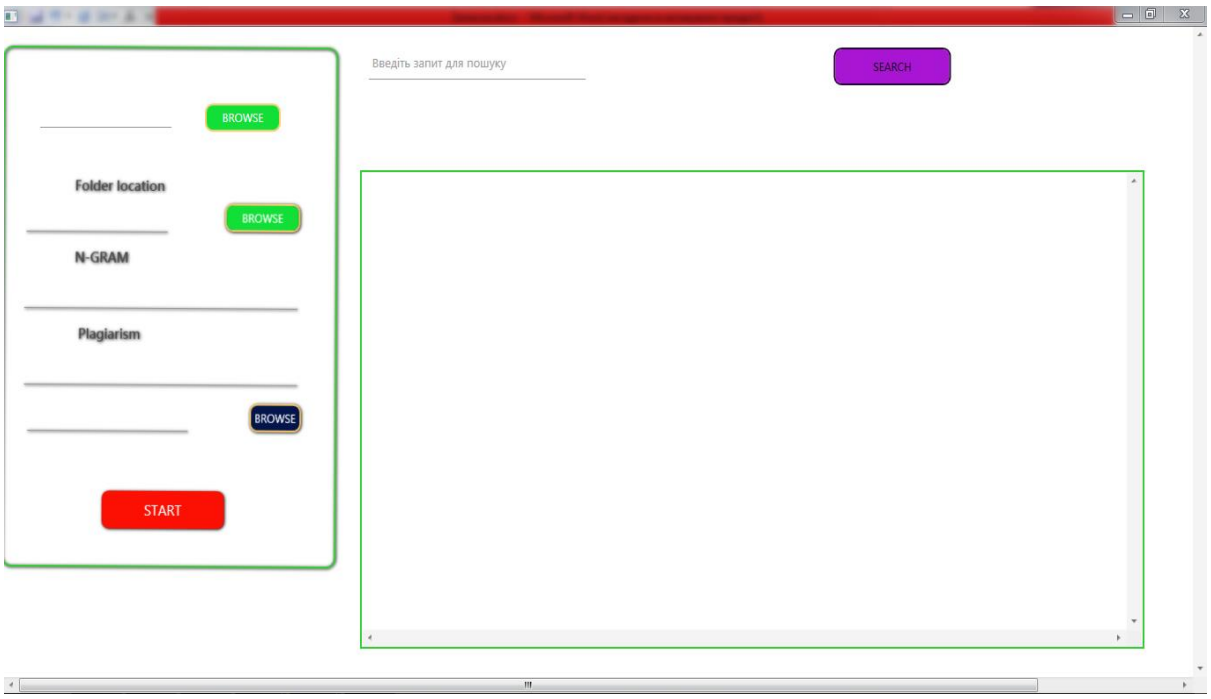

Рисунок 3.6 — Графічний інтерфейс.

Нижче наведено шматок коду для прикладу.

```
<Window x:Class=»FirstWpfApplication.MainWindow»
    xmlns=»http://schemas.microsoft.com/winfx/2006/xaml/presentation»
    mc:Ignorable=»d»
    Title=»MainWindow»
    Height=»350»
```

```
    Width=»525»>
  <Grid>
    <TextBlock
          Text=»Folder location»
          Margin=»8,0,0,0»
          FontSize=»15»
          FontWeight=»Medium»
          Foreground=»#4A4A4A»/>
    <Button
            Grid.Column=»1»
            Grid.Row=»0»
            Style=»{DynamicResource RoundedButtonStyle}»
            Cursor=»Hand»
            Foreground=»White»
            BorderThickness=»2»
            BorderBrush=»#fdcc6b»
            Background=»#FF12E038»
            Height=»30»
            Content=»BROWSE»
            Name=»BtnFolderLocation»
            Click=»BtnFolderLocation_OnClick»          Margin=»154,9,5,9»
Grid.ColumnSpan=»2»/>  </Grid>
</Window>
```

<Window> корінний контейнер, у якому розміщується контент, який візуалізує данні і дозволяє користувачам взаємодіяти з ними.

<Grid> панель макету, яка розставляє дочірні елементи в табличній структурі рідків і стобчиків.

<TextBlock> забезпечує легкий елемент керування для відображення текстового поля.

<Button> надає елемент керування кнопкою, який реагує на натискання.

Повний код наведено у дадотку Б.

Отже,для виконання програмної реалізації системи було обрано мову програмування C#, через її швидкодію, багатофункціональність, читабельність та простоту підтримки. Приділено увагу архітектурі .NET та середовищу розробки. Все це дозволяє з легкістю реалізувати необхідне програмне забезпечення, оскільки забезпечується висока продуктивність розроблених систем та широкі можливості для побудови і 60ормо контроль коду.

Було розроблену структуру програми. У розділі розроблено та описано необхідні складові частини, а сама - бібліотеки класі, необхідні модулі та інтерфейс користувача.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Експериментальні дослідження розроблених засобів

Для перших експериментів було сформовано стек документів. Стек містить 6 файлів курсових робіт з «Комп'ютерних мереж» (файли від 1 до 6) та 2 курсові з інших предметів (файли 7 та 8), у якості вхідного документа був також взятий файл курсової з «Комп'ютерних мереж» (файл 0) . Файли будуть позначатися номерами, без вказування прізвищ.

Файли для тестів у середньому мають розмір від 45 до 55 сторінок так містять від 6000 до 7500 тисяч слів. Це робить файли досить об'ємними та вимагає великих обчислювальних потужностей. Тестування буде проводитися на ноутбуці з об'ємом оперативної пам'яті 4 гібайта та процесором Intel Core i3-5005U. Виходячи з вище сказаного,при роботі з програмою рекомендовано використовувати обладнання параметрами не нижче тестового, оскільки в іншому випадку процес тестування займе багато часу.

У перших досліджень перевіримо усі файли теки між собою. У якості параметрів виставимо кількість N-грам 5, 8 і 10 та подивимося усі результати, для цього відсоток плагіату визначимо як 0. Результати експериментів наведені у таблиці 4.1, таблиці 4.2 та таблиці 4.3.

Таблиця 4.1 — Результати для 5 N-грам

	1	2	3	4	5	6	7	8
1		55,58	55,22	50,44	86,6	68,83	5,26	22,17
2	55,58		63,58	67,67	58	57,77	5,48	23,19
3	55,22	63,58		57,67	51,71	54,49	5,34	22,61
4	50,44	67,67	57,67		51,71	54,49	5,34	22,61
5	86,6	58	55,91	51,71		69,22	5,35	22,67
6	68,83	57,77	61,21	54,49	69,22		5,76	24,22
7	5,26	5,48	5,24	5,34	5,35	5,76		15,88
8	22,17	23,19	22,13	22,61	22,67	24,22	15,88	

Таблиця 4.2 — Результати для 8 N-грам

	1	2	3	4	5	6	7	8
1		28,06	28,62	22,17	74,61	45,23	0	0
2	28,06		40,58	45,2	30,56	29,49	0	0,01
3	28,62	40,58		33,9	27,99	36,3	0	0
4	22,17	45,2	33,9		23,33	26,03	0	0
5	74,61	30,56	27,99	23,33		45,75	0	0
6	45,23	29,49	36,3	26,03	45,75		0	0
7	0	0	0	0	0	0		3,8
8	0,01	0	0	0	0	0	3,8	

Таблиця 4.3 — Результати для 10 N-грам

	1	2	3	4	5	6	7	8
1		11,09	11,63	6,18	66,1	28,93	0	0
2	11,09		25,78	31,62	12,55	10,91	0	0
3	11,63	25,78		20,07	9,95	19,32	0	0
4	6,18	31,62	20,07		6,8	8,47	0	0
5	66,1	12,55	9,95	6,8		29,19	0	0
6	28,93	10,91	19,32	8,47	29,19		0	0
7	0	0	0	0	0	0		0
8	0	0	0	0	0	0	0	

Можемо бачити що при збільшенні кількості N-грам, можна більш точно сказати чи було запозичення інформації з тих чи інших файлів. Але у багатьох файлах спостерігаємо схожість згідно подібності косинусів при 5 N-грам 50-60 і при наступних 8 і 10 N-грам деяке зменшення — це пов’язано з тим, що файли однієї тематики — курсова з «Комп’ютерних мереж», тому спостерігаємо рівномірний спад при збільшенні N-грам. Додатково там є однакові додатки та обов’язкові однакові типові елементи. Тому з стовідсотковою вірогідністю

довіряти результатам у всіх випадках не можна і потрібна обов’язкова додаткова ручка перевірка людино.

Як можемо бачити, якщо файл мав достатньо високу кількість запозичень з іншого, то завдяки програмі ми можемо точно сказати про подібність. У нашому випадку 1 та 5 файл плагіатні. Файли 2 і 3, 1 і 6, 4 і 2, 5 і 6 — мають також схожість тому потребують детальної уваги оскільки є підозра на схожість. Графік визначення подібності для 1 файлу всіма N-грамами зображено на рисунку 4.1.

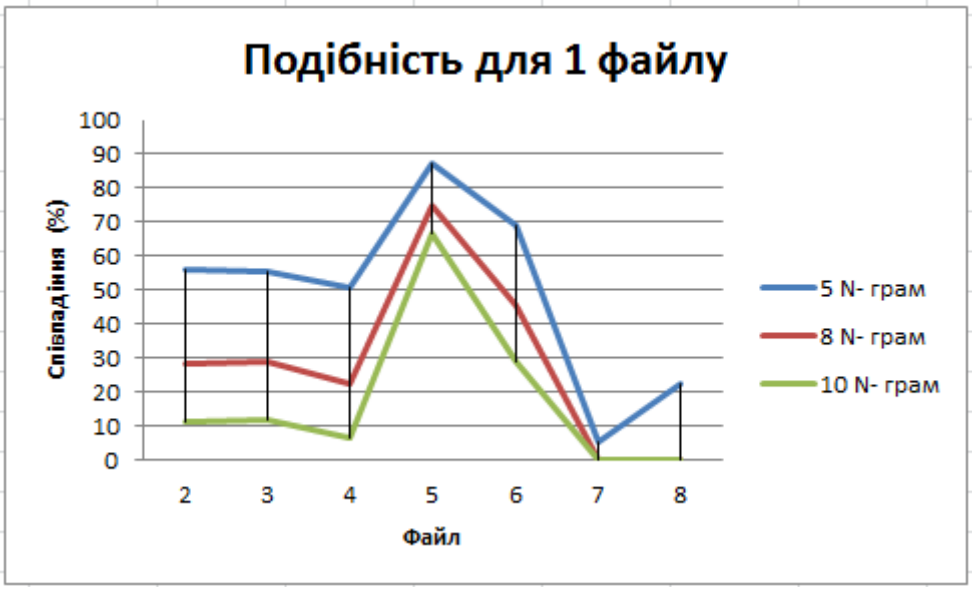

Рисунок 4.1 — Визначення подібності 1 файлу

Розглянувши рисунки 4.1 ми можемо сказати, що подібність 1 і 5 файлів всі N-грами визначили, також вони дали подібні результати і з іншими файлами. Схожі ситуація спостерігається при аналізі інших даних для різних файлів.

4.2 Керівництво оператора

При запуску програми перед користувачем з’являється вікно зображене на рисунку 4.2.

Програма містить три області:

— введення даних для виявлення плагіату (меню) (рисунок 4.3);

— область для виведення вмісту файлу, якщо вибраний один файл для перевірки, а якщо буде перевірятися тека — буде виведено звіт;

панель пошуку елементів (рисунок 4.4).

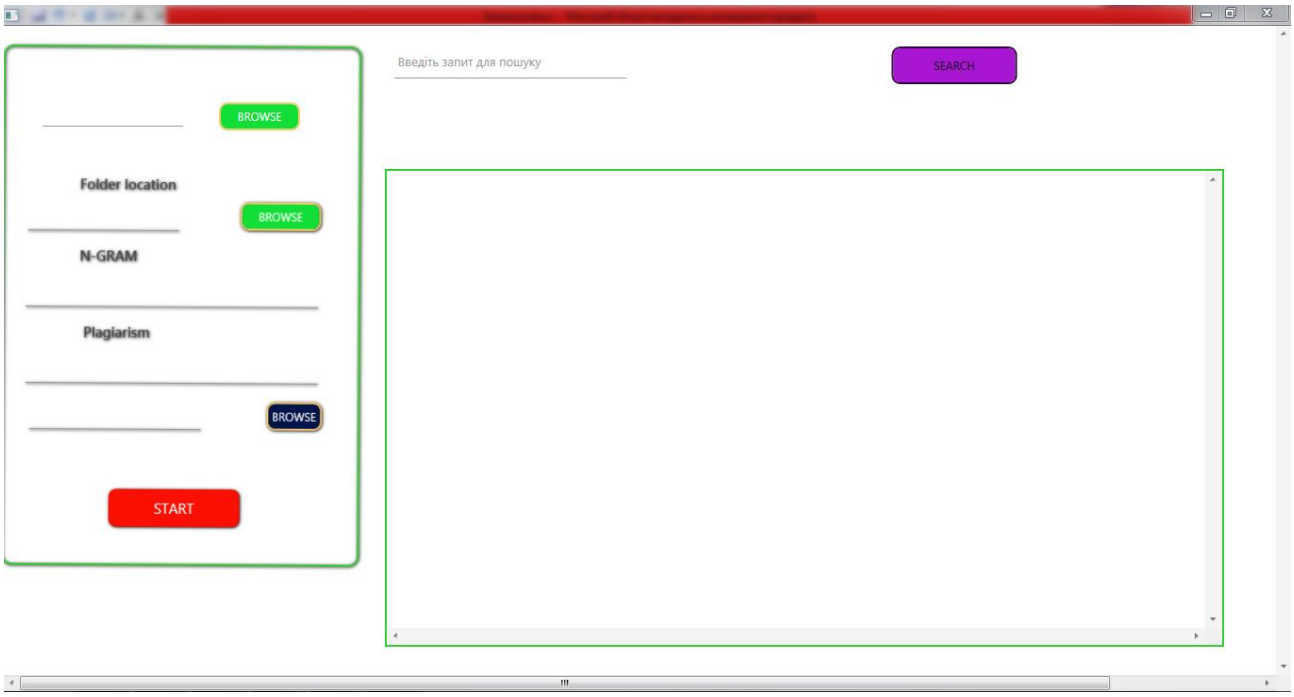

Рисунок 4.2 — Вікно програми.

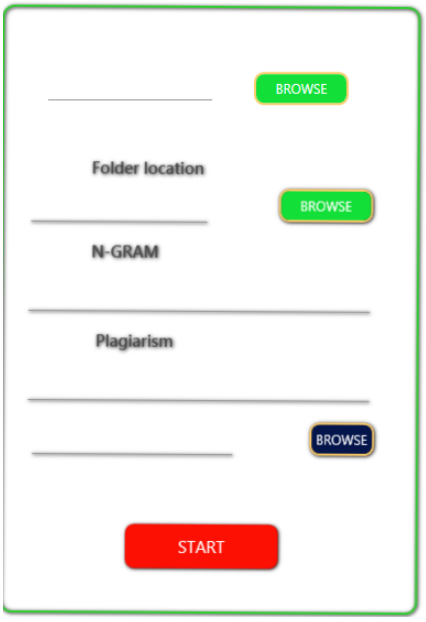

Рисунок 4.3 — Меню для введення даних для виявлення плагіату

Меню містить наступні поля:

— введення локації для файлу, який ми хочемо перевірити;

— введення локації теки, у якій знаходяться файли для порівняння;

— кількість N-грам;

— рівень плагіату;

— вибір місця збереження звіту;

— кнопка для початку перевірки.

Рисунок 4.4 — Панель пошуку елементів.

Якщо користувач бажає перевірити тільки один файл йому необхідно обов’язково заповнити поле локації файлу для перевірки. Якщо дане поле не буде заповнене, то відбудеться перевірка всієї теки.

При перевірці всієї теки всі файли, які містяться у ній, будуть порівняні один з одним і буде видано відповідний звіт.

Якщо було вибрано один файл, його вміст одразу з’явиться на екрані. Після цього ми можемо здійснювати пошук окремих елементів у файлі. Для цього потрібно ввести текст, який ми бажаємо знайти у відповідне поле та натиснути кнопку. Буде здійснено пошук (рисунок 4.5).

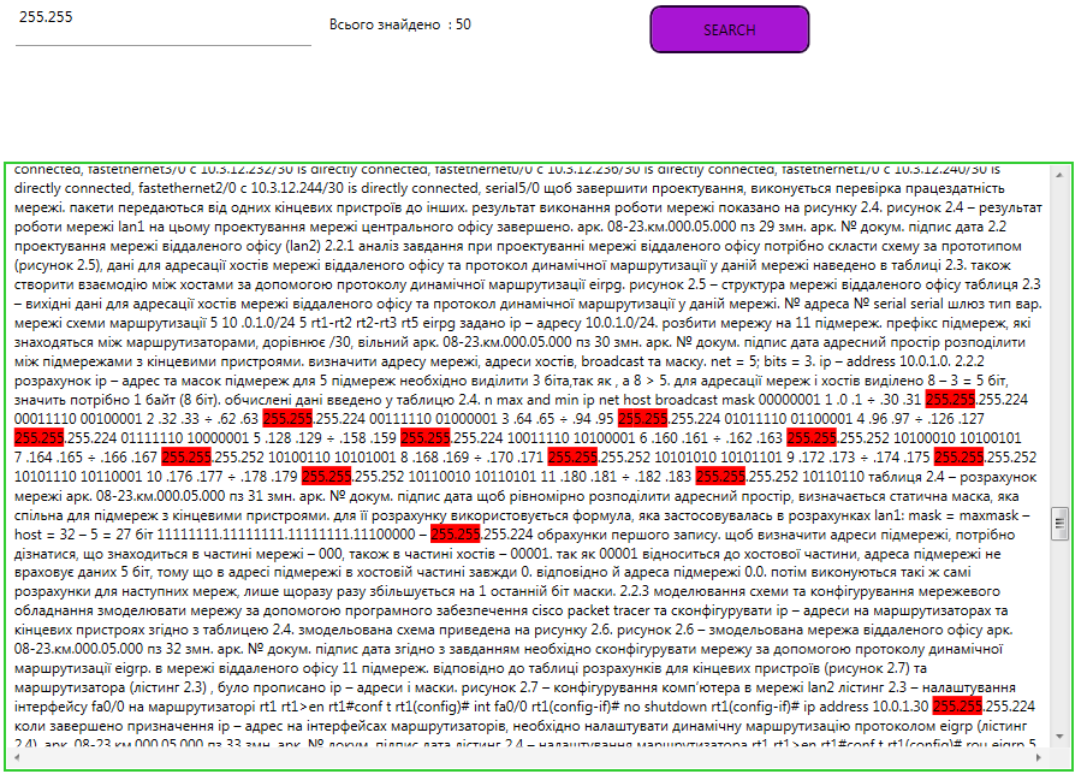

Рисунок 4.5 — Результати пошуку

Після здійснення пошуку буде виведена кількість знайдених елементів або повідомлення про їх відсутність. Збіги будуть виділені у тексті червоним кольором.

Якщо при запуску програми на виявлення плагіату не було введено обов'язкові дані з'явиться відповідне повідомлення. Коли відбувся запуск програми на виявлення плагіату, панель з вводом даних для користувача буде заблоковано. Але якщо в області для вмісту файлу, наявний відкрити текст, то ми можемо здійснювати у цей час пошук елементів. Після закінчення перевірки звіт з'явиться на екрані та у txt файлі у місці, яке обрали.

Отже, для виконання програмної реалізації системи було обрано мову програмування C#, через її швидкодію, багатофункціональність, читабельність та простоту підтримки. Приділено увагу архітектурі .NET та середовищу розробки. Все це дозволяє з легкістю реалізувати необхідне програмне забезпечення, оскільки забезпечується висока продуктивність розроблених систем та широкі можливості для побудови і 66ормо контроль коду. Було розроблену структуру програми. У розділі розроблено та описано необхідні складові частини, а сама – бібліотеки класі, необхідні модулі та інтерфейс користувача.

5. ЕКОНОМІЧНІ ЧАСТИНА

1.1 Оцінювання комерційного потенціалу розробки

Для початку, для оцінювання комерційного потенціалу розробки необхідно провести експертне опитування фахівців [30]. Для цього було залучено 3-х незалежних експертів к.т.н., доц. Захарченко С. М., к.т.н., доц. Крупельницький, 67ор.. викладач Л. В., Очкуров М. А. Їм було запропоновано оцінити розробку за 12-ма критеріями та виставити свої оцінки за 5-ти більною шкалою. Критерії оцінювання та їх варіанти оцінки наведені у таблиці 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовір-ність концепції не підтвердь-жена	Концепція підтвердь-жена експертними висновками	Концепція підтверджена розрахун-ками	Концеп-ція перевіре-на на практиці	Перевірено роботоздат-ність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1

3	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуата-ційні витрати значно вищі, ніж в аналогів	Експлуата-ційні витрати дещо вищі, ніж в аналогів	Експлуата-ційні витрати на рівні експлуата-ційних витрат аналогів	Експлуата-ційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитив-ною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція компаній	Активна конкуренція	Помірна конкурент-ція	Незначна конкурен-ція	Конкурентів немає

Продовження таблиці 5.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військовопромисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Кінець таблиці 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламент-тних документів та отримання великої кількості дозвільних документів на виробництв-во та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробниц-тва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідом-лення відповідним органам про виробницт-во та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки зведені в таблицю 5.2.

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище та ініціали експерта		
	Захарченко С. М.	Крупельницький Л. В.	Очкуров М. А.
	Бали, виставлені експертами:		
1	4	3	3
2	3	4	3
3	4	4	3
4	4	3	4
5	4	3	3
6	3	3	2
7	2	2	3
8	4	4	4
9	3	3	2
10	3	4	3
11	4	4	3
12	4	4	3
Сума балів	42	41	36
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = 39{,}67$		

Щоб визначити рівень комерційного потенціалу розробки за даними отриманими у таблиці 5.2 потрібно скористатися таблицею 5.3, у якій показні рівні комерційного потенціалу.

Проаналізувавши дані з таблиці 5.2 та порівнявши їх з даними таблиці 5.3, можемо зробити висновок що рівень комерційного потенціалу розробки відповідає «Вище середнього», оскільки за оцінками експертів середньоарифметична сума балів складає 39,67.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівні комерційного потенціалу розробки
0 — 10	Низький
11 — 20	Нижче середнього
21 — 30	Середній
31 — 40	Вище середнього
41 — 48	Високий

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько-технічної роботи

Для початку проведемо прогнозування витрат на виконання науково-дослідної, для розробки програмного забезпечення, яке складається з таких етапів [31]:

— розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

— розрахунок загальних витрат на виконання даної роботи;

— прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

Перший етап.Основна заробітна плата кожного з дослідників (розробників) $З_0$ розраховується за формулою 5.1:

$$З_0 = \frac{М}{Т_р} t \, [72орм], \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), 72орм.;

Тр — число робочих днів в місяці, у нас буде Тр = 21 день;

t — число робочих днів роботи розробника (дослідника).

В розробці програмного продукту бере участь один інженер—програміст.

Програма розроблялась 42 робочі дні. Оклад програміста складає 5100 грн. Керівник консультував 10 днів, оклад керівника керівника — 7000 грн.

Отже, основна заробітна плата програміста та керівника роботи за весь період роботи дорівнює:

$$З_0 = \frac{5100}{21} 42 = 10200(73орм)$$

$$З_0 = \frac{7000}{21} 10 = 3333{,}33(73орм)$$

Зроблені розрахунки зведено до таблиці 5.4

Таблиця 5.4 — Розрахунок основної заробітної плати

Працівник	Оклад М, 73орм.	Оплата за робочий день,73орм.	Число днів роботи, t	Витрати на оплату праці, 73орм.
1. Науковий керівник	7000	333,33	10	3333,33
2. Інженер-програміст	5100	242,86	42	10200
Всього:				13533,33

Обрахуємо витрати на додаткову заробітну плату $З_д$, вона обраховується як 10% від суми основної заробітної плати (формула 5.2):

$$З_д = 0{,}1 \cdot З_0 \quad (5.2)$$

$$З_д = 0{,}1х13533{,}33 = 1353{,}33\ (73орм)$$

Нарахування на заробітну плату НЗП розраховується як 22 % від суми їхньої основної та додаткової заробітної плати, для бюджетної сфери за формулою 5.3.

$$Н_{зп} = (З_0 + З_д)\frac{\beta}{100}, \quad (5.3)$$

де β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, 22 %.

У нашому випадку НЗП буде мати наступне значення:

$$Н_{зп} = (13533{,}33 + 1353{,}33)\frac{22}{100} = 3275{,}06\ (\ 74орм)$$

Розрахунок амортизаційних витрат виконується за формулою 5.4.

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (5.4)$$

де Ц — балансова вартість обладнання, 74орм.;

$Н_а$ — річна норма амортизаційних відрахувань % (25 %);

Т — фактична тривалість використання, у нашому випадку 3 місяці.

Зроблені розрахунки зведено до таблиці 5.5

Таблиця 5.5 — розрахунок амортизаційних відрахувань

Найменування	Балансова вартість, 74орм.	Норма амортизації, %	Термін використання, місяців	Величина амортизаційних відрахувань, 74орм..
Ноутбук	8500	25	3	531,25
Всього:				531,25

При розробці програмного забезпечення бали також витрати на комплектуючі К. До таких витрат у нас належить портативний носій інформації (флеш-пам'ять), зошит, ручка.

Витрати на комплектуючі розраховуються за формулою 5.5.

$$К = \sum_1^n Н_i \cdot Ц_i \cdot К_i, \quad (5.5)$$

де n — кількість комплектуючих;

$Н_i$ — кількість комплектуючих і-го виду;

$Ц_i$ — ціна комплектуючих і-го виду;

$К_i$— коефіцієнт транспортних витрат (1.1).

Обрахунки наведені в таблиці 5.6.

Таблиця 5.6 — Перелік використаних комплектуючих

Найменування матеріалу	Кількість	Ціна за 75ор., грн	Вартість витраченого матеріалу, 75орм..
Флеш-пам’ять	1	100	100
Зошит	1	15	15
Ручка	1	15	15
Всього з урахуванням транспортних витрат			143

Під час розробки програмного продукту використовувались лише безкоштовні програмні засоби.

Витрати на електроенергію визначаються на основі витрат на одиницю продукції та тарифів на енергію за формулою 5.6.

$$В_е = В \cdot П \cdot Ф \cdot К_і \left[75орм\right], \quad (5.6)$$

де В — вартість 1 кВт електроенергії (2,36 грн за 1кВт);

П — потужність обладнання кВт (0,6 кВт);

Ф — фактична кількість годин роботи комп’ютера, годин (210);

$К_і$ — коефіцієнт використаної потужності (0,65).

Отже, витрати на електроенергію становлять:

$$В_е = 2{,}36 \cdot 0{,}6 \cdot 210 \cdot 0{,}65 = 193{,}28\ (75орм)$$

Інші витрати $В_{ін}$ охоплюють: витрати на управління організацією, оплату службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші $І_і$ можна прийняти як (100-300)% від суми основної заробітної плати розробників та робітників, які виконували дану роботу та обрахувати за формулою 5.7.

$$В_{ін} = (1..3)(З_о + З_р) \quad (5.7)$$

Отже, у нашому випадку:

$$В_{ін} = 1(13533,33 + 1353,33) = 14886,66 \text{ (76орм)}$$

В результаті сума всіх витрат на виконання даного етапу роботи обрахується за формулою 5.8.

$$В = З_0 + З_д + Н_{ЗП} + А + К + В_е + В_{ін} \quad (5.8)$$

В=13533,33+1353,33+3275,06+531,25+143+193,28+14886,66=33915,91 (76орм.)

5 етап. Розрахунок загальних витрат на виконання даної роботи [32].

Загальна вартість всієї наукової роботи визначається за формулою 5.9.

$$В_{заг} = \frac{В}{а} \text{ [грн]}, \quad (5.9)$$

де а — частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відносних одиницях.

У нашому випадку 1.

$$В_{заг} = \frac{33915,91}{1} = 33915,91 \text{ (грн)}$$

6 етап. Прогнозування загальних витрат на виконання та впровадження результатів виконаної роботи.

Прогнозування загальних витрат ЗВ здійснюється за формулою 5.10.

$$ЗВ = \frac{В_{заг}}{\beta} \text{ (76орм)}, \quad (5.10)$$

де β — коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Так, якщо розробка знаходиться:

— на стадії науково-дослідних робіт, то $\beta \approx 0,1$;

— на стадії технічного проектування, то $\beta \approx 0,2$;

— на стадії розробки конструкторської документації, то $\beta \approx 0,3$;

— на стадії розробки технологій, то β ≈ 0,4;

— на стадії розробки дослідного зразка, то β ≈ 0,5;

— на стадії розробки промислового зразка, β ≈ 0,7;

— на стадії впровадження, то β ≈ 0,9.

Підставивши наші дані у формулу ми отримаємо:

$$ЗВ = \frac{33915,91}{0,9} = 37684,34\ (77орм)$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спробуємо кількісно спрогнозувати, яку вигоду, можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Зрозуміло, що всі зроблені розрахунки будуть приблизними і не передбачають деталізації.

В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємець від впровадження результатів нової розробки, є збільшення чистого прибутку. Зростання чистого прибутку спробуємо оцінити у теперішній вартості грошей.

Ми не можемо прямо оцінити зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою 5.11.

$$\Delta П_i = \sum_1^n (\Delta Ц_я \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right), \qquad (5.11)$$

де $\Delta Ц_я$ — покращення основного якісного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки;

λ — коефіцієнт, який враховує сплату податку на додану вартість (ставка 20%, коефіцієнт λ =0,8333);

ρ — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток (18%).

В результаті впровадження результатів наукової розробки покращується якість програмного продукту, що дозволяє підвищити ціну його реалізації на 500 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 30 шт., протягом другого року — на 25, протягом третього — на 20.

Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 10шт., а її ціна —3000.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом першого року складе:

$$\Delta П_1 = [3000 \cdot 10 + (3000 + 500) \cdot 30] \cdot 0.8333 \cdot 0.28 \left(1 - \frac{18}{100}\right) =$$

$$= 20663{,}17 \text{ (78орм)}$$

Збільшення чистого прибутку підприємства $\Delta П_i$ протягом другого року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_2 = [3000 \cdot 10 + (3000 + 500) \cdot (30 + 25)] \cdot 0.8333 \cdot 0.28 \left(1 - \frac{18}{100}\right) =$$

$$= 42569{,}96 \text{(78орм)}$$

Збільшення чистого прибутку підприємства $\Delta П_і$ протягом третього року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_3 = [3000 \cdot 10 + (3000 + 500) \cdot (30 + 25 + 20)] \cdot 0.8333 \cdot 0.28\left(1 - \frac{18}{100}\right) =$$

$$= 55962{,}76\ (79орм)$$

Загальне збільшення прибутку від впровадження розробки за три роки становить:

$$\Delta П = 20663{,}17 + 42569{,}96 + 55962{,}76 = 119195{,}89(79орм)$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Основними показниками, які визначають доцільність фінансування наукової розробки інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності. Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт:

1-й крок — розраховується теперішня вартість інвестицій PV, що вкладаються в наукову розробку. Такою вартістю можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, розраховану за формулою, тобто будемо вважати, що ЗВ = PV=37684,34 грн.

2-й крок — розраховується очікуване збільшення прибутку ΔПі , що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження. Таке збільшення прибутку також було розраховане нами раніше.

3-й крок — будуємо вісь часу, на яку наносимо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково—дослідної роботи та впровадження її результатів.

Платежі показуємо у ті терміни, коли вони здійснюються.

Загальні витрати ЗВ на виконання та впровадження результатів наукової роботи (або теперішня вартість інвестицій PV) дорівнює 37684,34 грн. Результати

вкладених у наукову розробку інвестицій почнуть виявлятися протягом трьох років. Вони виявляться у тому, що у першому році підприємство отримає збільшення чистого прибутку на 20663,17 грн., відносно базового року, у другому році — збільшення чистого прибутку на 42569,96 грн., (відносно базового року), у третьому році — збільшення чистого прибутку на 55962,76 грн., (відносно базового року).

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рисунку 5.1.

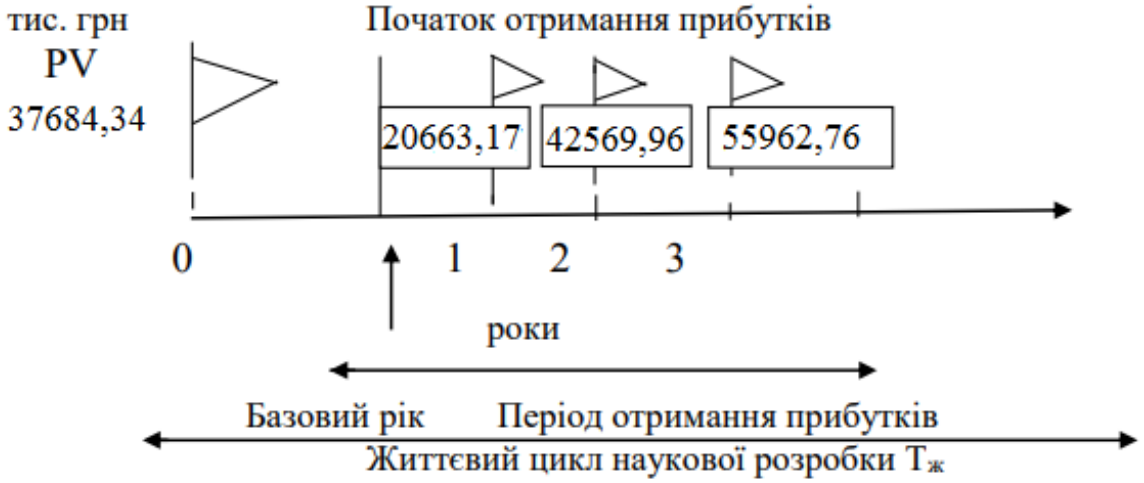

Рисунок 5.1 — Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

4-й крок — розраховується абсолютна ефективність вкладених інвестицій $E_{абс}$.

Абсолютна ефективність $E_{абс}$вкладених інвестицій розраховується за формулою 5.12.

$$E_{абс} = (ПП - PV), \qquad (5.12)$$

де ПП — приведена вартість всіх чистих прибутків, що їх отримає підприємство (орга

80ормо к) від реалізації результатів наукової розробки, 80орм.;

PV — теперішня вартість інвестицій PV=ЗВ, 80орм.

У свою чергу, приведена вартість всіх чистих прибутків ПП обчислюється за формулою 5.13.

$$ПП = \sum_{1}^{m} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.13)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, 81орм;

т — період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t — період часу (в роках) від моменту отримання чистого прибутку до точки «0».

Розрахуємо вартість чистого прибутку:

$$ПП = \frac{20663,17}{(1+0,1)^1} + \frac{42569,96}{(1+0,1)^2} + \frac{55962,76}{(1+0,1)^3} = 96012,11 \text{ (81орм.)}$$

Тоді $Е_{абс}$:

$$Е_{абс} = 96012,11 - 37684,34 = 58327,77 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

5-й крок — розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою 5.14.

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (5.14)$$

де $Е_{абс}$ — абсолютна ефективність вкладених інвестицій, 81орм;

PV — теперішня вартість інвестицій PV = ЗВ, 81орм;

$Т_ж$ — життєвий цикл наукової розробки, роки.

Далі, розраховану величина $Е_в$порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть.

У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою 5.15.

$$\tau = d + f, \quad (5.15)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2020 році в Україні d = 0,2;

f — показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$T = 0{,}2 + 0{,}08 = 0{,}28$$

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{58327{,}77}{37684{,}34}} - 1 = 0{,}366 \text{ або } 36{,}6\ \%$$

Оскільки $E_в$ = 36,6% > $\tau_{мін}$ = 0,28 = 28%, то інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

6-й крок. Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою 5.16

$$T_{ок} = \frac{1}{E_в} \quad (5.16)$$

Якщо $T_{ок}$ буде менше 5 років, то фінансування даної наукової розробки є доцільним.

Для даної розробки термін окупності вкладених у реалізацію проекту інвестицій $T_{ок}$ складе:

$$T_{ок} = \frac{1}{0{,}366} = 2{,}73 \text{ (року)}$$

Це свідчить, що доцільно фінансувати дану наукову розробку.

Отже, даному розділі було здійснено оцінювання комерційного потенціалу розробки автоматизованої системи контролю контенту студентських робіт.

Проведено технологічний аудит з залученням трьох незалежних експертів. Було визначено,що рівень комерційного потенціалу вище середнього.

Аналіз комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною розробкою. Він має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку.

Після проведення розрахунків можна зробити висновок про доцільність розробки та впровадження НДДКР. Це підтверджується наступними показниками:

— абсолютна ефективність вкладених інвестицій, яка дорівнює 58327.77, що є більшим 0 — це показує, що інвестори будуть зацікавлені у нашій розробці;

— відносна ефективність розробки становить 36,6 %, що є досить гарним показником та вищим за мінімальну ставку дисконтування, тому вкласти гроші у нашу наукову розробку вигідніше, ніж покласти на депозит у банку;

— термін окупності інвестицій складає 2,73 роки, що не перевищує максимального порогу у 5 років та вказує на відносно швидку окупність вкладених грошей.

Результати проведених розрахунків свідчать про доцільність розробки та подальшу перспективу інвестицій.

ВИСНОВКИ

У результаті аналізу предметної області доведено актуальність створення автоматизованої системи перевірки контенту документів на плагіат з додатковими функціональними можливостями.

Розглянуто основні методи побудови систем аналізу контенту та проведено їх класифікацію за різними ознаками, проаналізовано існуючі системи виявлення плагіату та визначено їхні основні переваги і недоліки, що показало доцільність розробки комплексної системи, оскільки існуючі системи не задовольняють повному переліку вимог.

Розроблено теоретико-математичну модель системи. Запропоновано для початку текст токенізувати, оскільки не всі слова та знаки несуть об’єктивну інформацію. Для представлення документів обрано векторну просторову модель, яка може застосовуватися для будь-якого типу об’єктів. Ця модель полягає у виборі ознак та присвоєнні значень цим ознакам, що дозволяє представити тексти як вектори. У якості даних для вектора обрано TF-IDF, через те що він може продемонструвати важливість слова для документа. Щоб зберегти порядок слів було використано n-грами. Для вимірювання схожості векторів обрано косинусну подібність.

Для програмної реалізації системи обрано мову програмування C#, через її швидкодію, багатофункціональність, читабельність та простоту підтримки. Приділено увагу архітектурі .NET та середовищу розробки. Все це дозволяє спростити реалізацію необхідного програмне забезпечення, створюючи широкі можливості для побудови і 84ормо контроль коду та забезпечуючи високу продуктивність розробленого програного продукту.

Розроблену структуру програми, необхідне програмне забезпечення, а саме — бібліотеки класів, необхідні модулі та інтерфейс користувача.

Проведено експериментальну перевірку розробленої автоматизованої системи контролю контенту студентський робіт, що показало коректність та правильність роботи програмних засобів.

Економічні розрахунки показали, що розробка за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною. Вона має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку. Результати проведених розрахунків свідчать про доцільність розробки та подальшу перспективу інвестицій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bolilyi V. O. Check the Uniqueness of the Text in the Assessment of Student Work Creative or Exploratory / V.O. Bolilyi, V. V. Kopotii // Naukovi zapysky NDU im. M. Hoholia. — 2011. — № 7 (34).— P. 134—145. (in Ukrainian).
2. Болілий В.О. Перевірка унікальності тексту при оцінюванні студентських робіт творчого або дослідницького характеру/ В.О. Болілий, В. В. Копотій // Наукові записки НДУ ім. М. Гоголя. — 2011. — № 7 (34).— С. 134—145.
3. Чорний Д.С., Захарченко С. М. «АВТОМАТИЗОВАНА СИСТЕМА КОНТРОЛЮ КОНТЕНТУ СТУДЕНТСКЬИХ РОБІТ» в Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2021)», Вінниця, 2020. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2021/paper/view/10993 Дата звернення: Листопад 2020.
4. Михайловський Ю.Б. Система Anti-Plagiarism як інструмент запобіганняплагіату в навчальній та науковій діяльності / Ю.Б. Михайловський, Н.А.Длугунович // Вісник Хмельницького національного університету. Технічнінауки. — 2013. — № 3.— С. 162—168.
5. Шинкаренко В. І. Система контролю плагіату в студентських роботах[Електронний ресурс] / В. І. Шинкаренко, О. С. Куроп’ ятник // СхідноЄвропейський журнал передових технологій. — 2012. — Том. 4. — № 2(58). — С.32-36.
6. Захист авторських прав в інтернеті: нові техніки плагіату. Checkpoint Business Media, 2016. Режим доступу: https://ckp.in.ua/articles/11504.
7. Захарченко, С. Мережа академій Cisco — перспективний шлях підготовки фахівців з мережевих технологій / Сергій Захарченко // Proceedings of the ninth international scientific-practical conference “Internet-Education-Science” (IES-2014), Vinnytsia, 14 – 17 October, 2014. – Vinnytsia : VNTU, 2014. – С. 109.
8. Morris J. Dworkin. SHA-3 Standard: Permutation-Based Hash and ExtendableOutput Functions. Federal Inf. Process. Stds. (NIST FIPS) – 202, 2015.

9. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002, p. 380
10. Комп'ютерні мережі / О. Д. Азаров, С. М. Захарченко, О. В. Кадук, М. М. Орлова, В. П. Тарасенко // Навчальний посібник. — Вінниця: ВНТУ, 2013./МОНУ (Лист №1/11 — 8260 від 15.05 2013 р.) – 500 с.
11. А. Щербаков, А. Домашев. Прикладная 87ормо контро. — М.: Русская Редакция, 2003. — 404 с.
12. Захарченко С. М. Дослідження можливостей генетичного алгоритму в задачі кластеризації користувачів мережі internet / С. М. Захарченко, Н. Р. Романівна, О. О. Манаєва // Інформаційні технології та комп ютерна інженерія. – 2010. № 2.
13. John Kelsey, Stefan Lucks. Collisions and Near-Collisions for Reduced-Round Tiger // Graz, 2006.
14. Шарапова Е. В. " Универсальная система проверки текстов на 87ормо к «Автор.net»" / Е. В. Шарапова, Р. В. Шарапов // Информатика и её применения — 2012. — № 3 (6).— С. 52—58
15. Чижова А. А. Алгоритми пошуку плагіату [Електронний ресурс] / А. А. Чижова // Східно-Європейський журнал передових технологій. — 2010. – 4, №2
16. Метод шинглов при 87ормо плагиата [Електронний ресурс] — дата візиту 28.11.2017. — Режим доступу до ресурсу: https://goo.gl/Cm3Sgg
17. Sadowski, C., & Levin, G. (2007). Simhash: Hash-based similarity detection. Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi= 10.1.1.473.7179&rep=rep1&type= pdf.
18. Morris J. Dworkin. SHA-3 Standard: Permutation-Based Hash and ExtendableOutput Functions. Federal Inf. Process. Stds. (NIST FIPS) – 202, 2015.
19. Sanadhya S.K., Sarkar P. (2008) Deterministic Constructions of 21-Step Collisions for the SHA-2 Hash Family. In: Wu TC., Lei CL., Rijmen V., Lee DT. (eds)

Information Security. ISC 2008. Lecture Notes in Computer Science, vol 5222. Springer, Berlin, Heidelberg

20. Tf—idf [Електронний ресурс]// Режим доступу: https://en.wikipedia.org/wiki/Tf—idf — Назва з екрану
21. Language Models: N-Gram [Електронний ресурс]// Режим доступу: https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9 — Назва з екрану
22. N-gram [Електронний ресурс]// Режим доступу: https://en.wikipedia.org/wiki/N-gram — Назва з екрану
23. Quantitative authorship attribution: An evaluation of techniques[Електронний ресурс]// Режим доступу: https://academic.oup.com/dsh/article-abstract/22/3/251/951481?redirectedFrom=fulltext — Назва з екрану
24. The advanced theory of language as choise and chance / Herdan Gustav // Kommunikation und Kybernetik in Einzeldarstellungen. — 1966. — С. 14—437.
25. Support vector machines and machine learning on documents [Електронний ресурс]// Режим доступу:https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-and-machine-learning-on-documents-1.html — Назва з екрану
26. Sidorov, Grigori (2013). “Syntactic Dependency-Based n-grams in Rule Based Automatic English as Second Language Grammar Correction”. International Journal of Computational Linguistics and Applications. 4 (2): 169—188.
27. Wołk, K.; Marasek, K.; Glinkowski, W. (2015). “Telemedicine as a special case of Machine Translation”. Computerized Medical Imaging and Graphics. 46 Pt 2: 249-56
28. Jones K. S. A statistical interpretation of term specificity and its application in retrieval // Journal of Documentation : журнал. — MCB University : MCB University Press, 2004. — Т. 60, № 5. — С. 493-502.
29. Robertson, S. (2004). “Understanding inverse document frequency: On theoretical arguments for IDF”. Journal of Documentation. 60 (5): 503—520.
30. C# documentation [Електронний ресурс]// Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/ — Назва з екрану

31. Козловський В.О. Бізнес-планування. / В.О. Козловський, О.Й. Лесько. — Вінниця: ВНТУ, 2005 — 188 с.

32. Козловський В.О. Техніко-економічні обґрунтування та економічні розрахунки в дипломних проектах та роботах. Навчальний посібник. — Вінниця: ВДТУ, 2003. — 75 с.

33. Адлер О. О. Методичні вказівки для підготовки та написання курсової роботи з дисципліни «Економічне обґрунтування інноваційних рішень» для студентів, що навчаються за напрямами підготовки: 050101 Комп'ютерні науки, 050102 Комп'ютерна інженерія, 050103 Програмна інженерія, 170101 Безпека інформаційних і комунікаційних систем, 170103 Управління інформаційною безпекою // О.О.Адлер, І.В. Причепа. — ВНТУ, 2013. — 35 с.