Міністерство освіти і науки України

Запорізька державна інженерна академія
(повне найменування вищого навчального закладу)

Факультет енергетики, електроніки та інформаційних технологій
(назва факультету)

Кафедра програмного забезпечення автоматизованих систем
(повна назва кафедри)

Пояснювальна записка

до магістерської роботи

рівень вищої освіти другий (магістерський)
(другий (магістерський) рівень)

на тему **Дослідження проблеми нечіткого пошуку в словниках з використанням методу опорних векторів**

Виконала: студентка 2 курсу, групи ІПЗ-16-1м
Стаднікова Анастасія Євгенівна _____
(ПІБ) (підпис)

спеціальності
121 Інженерія програмного забезпечення
(шифр і назва)

освітньо-професійна програма
Інженерія програмного забезпечення
(назва)

Керівник: Вербицький В. Г. _____
(прізвище та ініціали) (підпис)

Запоріжжя – 2018 року

Запорізька державна інженерна академія
(повне найменування вищого навчального закладу)

Факультет ___енергетики, електроніки і інформаційних технологій_
Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
(другий (магістерський) рівень)
Спеціальність _____121 Інженерія програмного забезпечення_____
(шифр і назва)
Освітньо-професійна програма Інженерія програмного забезпечення
(назва)

ЗАТВЕРДЖУ
Завідувач кафедри
В.Г. Вербицьки
"_04_" _вересня_____2017 ро

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТЦІ

Стадніковій Анастасії Євгенівні
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи ___Дослідження проблеми нечіткого пошуку словниках з використанням методу опорних векторів
керівник магістерської роботи __Вербицький Володимир Григорович, доктор фізико-математичних наук, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від
"04" _вересня_2017_року № 352-01
2. Строк подання студентом магістерської роботи___09.01.2018____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно ро робити)
- огляд та збір літератури стосовно теми дипломної роботи;
- дослідження предметної області;
- огляд та аналіз існуючих аналогів;
- створення програмного продукту та його опис;
- перелік вимог в роботі з програмою та системою;
- реалізація та тестування створеного продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
___слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв
Нормоконтроль	Полякова Н.П., доц.каф.ПЗАС	29.12.17

7. Дата видачі завдання 04.09.2017

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	04.09-10.09.17	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.17	виконано
3	Аналіз існуючих методів рішення	13.09-17.09.17	виконано
4	Дослідження області нечіткого пошуку	18.09-24.09.17	виконано
5	Узгодження подальших дій з науковим керівником	25.09-26.09.17	виконано
6	Аналіз теоретичних відомостей	27.09-15.10.17	виконано
7	Проектування програмної системи та робота зі словником WordNet	15.10-23.10.17	виконано
8	Узгодження програмної системи з науковим керівником	23.10-24.10.17	виконано
9	Розробка базової версії програми для нечіткого пошуку у словнику	25.10-14.11.17	виконано
10	Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження	15.11-16.11.17	виконано
11	Вдосконалення базової версії програми та додавання можливості розширення словника	16.11-23.11.17	виконано
12	Проведення дослідження щодо залежності часу виконання пошуку до кількості помилкових даних у вхідному тексті	24.11-09.12.17	виконано
13	Оформлення звіту	10.12-23.12.17	виконано

Студентка _____ **А. Є. Стаднікова**
(підпис) (прізвище та ініціали)

Керівник магістерської роботи _____ **В.Г. Вербицький**
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Сторінок: 109

Рисунків:30

Таблиць: 5

Використаних джерел: 26

Мета роботи: дослідити проблему нечіткого пошуку в словниках та створити програмний застосунок, який, використовуючи метод опорних векторів, визначає наявність помилок у текстових даних і пропонує користувачу найбільш відповідне слово.

Результати: на основі результатів дослідження розроблено програмний застосунок, який за допомогою словника виконує перевірку тексту на наявність помилок. У разі знаходження помилки програма повідомляє користувача та пропонує найбільш відповідне слово, а також надає загальну статистику оброблених даних. Даний програмний застосунок є розширюваним, так як дозволяє користувачу додавати нові слова у словник.

Публікація: Стаднікова А.Є., магістрант, Вербицький В.Г., проф., д.ф.-м.н. — науковий керівник. «Дослідження проблеми нечіткого пошуку в словниках з використанням методу опорних векторів.» Матеріали XXII науково-технічної конференції студентів, магістрантів, аспірантів і викладачів ЗДІА «ЕНЕРГЕТИКА, ЕЛЕКТРОНІКА ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ» том III, Запоріжжя, ЗДІА, 2017, с.141.

ПОШУКОВА СИСТЕМА, ПОВНОТА ЗБІГУ, НЕЧІТКИЙ ПОШУК, МЕТОД ОПОРНИХ ВЕКТОРІВ, МЕТРИКА ЛЕВЕНШТЕЙНА, WORDNET, ВЕБ-ТЕХНОЛОГІЇ, СТВОРЕННЯ.

РЕФЕРАТ

Страниц: 109

Рисунков: 30

Таблиц: 5

Использованных источников: 26

Цель работы: исследовать проблему нечеткого поиска в словарях и создать программное приложение, которое, используя метод опорных векторов, определяет наличие ошибок в текстовых данных и предлагает пользователю наиболее подходящее слово.

Результаты: на основе результатов исследования разработано программное приложение, которое с помощью словаря выполняет проверку текста на наличие ошибок. В случае нахождения ошибки программа оповещает пользователя и предлагает наиболее подходящее слово, а также предоставляет общую статистику обработанных данных. Данное программное приложение является расширяемым, так как позволяет пользователю добавлять новые слова в словарь.

Публикация: Стаднікова А.Є., магістрант, Вербицький В.Г., проф., д.ф.-м.н. — науковий керівник. «Дослідження проблеми нечіткого пошуку в словниках з використанням методу опорних векторів.» Матеріали XXII науково-технічної конференції студентів, магістрантів, аспірантів і викладачів ЗДІА «ЕНЕРГЕТИКА, ЕЛЕКТРОНІКА ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ» том III, Запоріжжя, ЗДІА, 2017, с.141.

ПОИСКОВАЯ СИСТЕМА, ПОЛНОТА СОВПАДЕНИЯ, НЕЧЕТКИЙ ПОИСК, МЕТОД ОПОРНЫХ ВЕКТОРОВ, МЕТРИКА ЛЕВЕНШТЕЙНА, WORDNET, ВЕБ-ТЕХНОЛОГИИ, АНАЛИЗ, СОЗДАНИЕ.

THE ABSTRACT

Pages: 109

Figures: 30

The tables: 5

The references: 26

The purpose of work: to analyze the problem of fuzzy search in dictionaries and to create a software application that, using a reference vector method, determines the existence of errors in text data and offers the user the most appropriate word.

Results: based on the results of the study, a software application was developed that, using the dictionary, checks the text for errors. In case of an error, the program notifies the user and offers the most appropriate word, and also provides general statistics of processed data. This software application is extensible as it allows the user to add new words to the dictionary.

Publication: Стаднікова А.Є., магістрант, Вербицький В.Г., проф., д.ф.-м.н. — науковий керівник. «Дослідження проблеми нечіткого пошуку в словниках з використанням методу опорних векторів.» Матеріали XXII науково-технічної конференції студентів, магістрантів, аспірантів і викладачів ЗДІА «ЕНЕРГЕТИКА, ЕЛЕКТРОНІКА ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ» том III, Запоріжжя, ЗДІА, 2017, с.141.

SEARCH SYSTEM, COMPLETENESS OF COINCIDENCE, FUZZY SEARCH, SUPPORT VECTOR METHOD, LEVENSTEIN METRICS, WORDNET, WEB TECHNOLOGY, ANALYSIS, CREATION.

ЗМІСТ

ВСТУП

З кожним роком обсяги Інтернету збільшуються в рази, тому ймовірність знайти необхідну інформацію різко зростає. Інтернет об'єднує мільйони комп'ютерів, безліч різних мереж, число користувачів збільшується на 15-80% щорічно. І, тим не менш, все частіше при зверненні до мережі Інтернет основною проблемою виявляється не відсутність шуканої інформації, а неможливість її знайти.

Пошук інформації являє собою процес виявлення в деякій множині документів (текстів) всіх тих, які присвячені зазначеній темі, задовольняють заздалегідь визначеним умовам пошукового запиту або містять необхідні факти, відомості або дані, що відповідають інформаційній потребі. Методи пошуку можуть бути класифіковані різними способами. У більшості випадків під пошуком розуміють задачу вибірки, в якій потрібно знайти елемент, що задовольняє деяким умовам. Однак виникають ситуації, коли пошук являє собою знаходження об'єктів, схожих, в певному сенсі, на заданий об'єкт. У таких випадках прийнято говорити про використання алгоритмів нечіткого пошуку.

Багато в чому виникнення задач нечіткого пошуку обумовлено проблемами управління якістю даних, в тому числі пов'язаними з наявністю як в запитах, так і безпосередньо в базах даних, помилок введення інформації. Саме тому питання правильного і грамотного написання пошукових запитів для отримання бажаних даних є дуже важливим, адже у разі помилки користувач може не отримати необхідні або отримати помилкові дані.

Актуальність роботи

Задача аналізу алгоритмів нечіткого пошуку на сьогоднішній момент актуальна, так як область застосування даних алгоритмів неймовірно велика і різноманітна. Почнемо з розпізнавання рукописних символів, яке з масовим поширенням пристроїв з сенсорним екраном активно використовується для забезпечення зручності введення. Введений символ перетворюється в комбінацію цифр в залежності від послідовності жестів, і отримана комбінація порівнюється зі значеннями, заздалегідь відомими для всіх символів використовуваного алфавіту, записаними в таблицю. Символ, для якого збіг буде найбільш повним, і вважається розпізнаним. Саме для визначення повноти збігу і використовуються в алгоритмах нечіткого пошуку, оскільки розпізнані значення можуть відрізнятися від закладених в таблиці для деякого конкретного символу.

Інша область, в якій є необхідність використання алгоритмів нечіткого пошуку – це у системах моніторингу лісопожежної ситуації. Такі системи являють собою розподілені системи з обліку інформації про лісові пожежі, данні до яких передаються з різних підрозділів авіаційної та наземної охорони лісів.

Наступна область – це форми заповнення інформації на сайтах та повноцінні пошукові системи Google, Yahoo та ін. Наприклад, такі алгоритми використовуються для функцій на зразок «Можливо ви мали на увазі ...» в тих же пошукових системах і для виявлення помилок в полях введення програм.

В біоінформатиці дані алгоритми успішно застосовуються для порівняння генів, білків та хромосом, також для обробки масивів даних в інтересах кредитних організацій і багатьох інших областях. Саме тому важливо знати особливості основних застосовуваних на сьогоднішній момент алгоритмів, щоб для конкретної ситуації була можливість обрати максимально ефективний з них, оскільки в більшості завдань це може зіграти важливу роль, а у випадках, схожих з моніторингом лісопожежної ситуації, навіть коштувати комусь то життя.

Мета і задачі дослідження

Метою кваліфікаційної роботи магістра є дослідження методів реалізації нечіткого пошуку у тексті та словнику, а також створення програмного застосунку для визначення помилок у тексті.

Досягнення поставленої мети передбачає:

- аналіз наближеного пошуку на основі метрики відстані Левенштейна;
- дослідження методу опорних векторів;
- використання навчальної вибірки, представленої у вигляді семантичного словника WordNet;
- аналіз та обробка вхідних текстових даних;
- реалізації алгоритмів вибору нечіткого пошуку;
- створення програмної системи, яка визначає помилкові слова у текстових даних та пропонує користувачу найбільш відповідне за значенням слово;
- тестування та представлення результатів досліджень та роботи на реальних прикладах у зручному і зрозумілому для користувача вигляді.

Об’єкт дослідження

Дані у текстовому вигляді, які можуть містити орфографічні помилки.

Предмет дослідження

Методи реалізації нечіткого пошуку у тексті та словнику та можливість створення застосунку для виявлення та виправлення орфографічних помилок.

Методи дослідження

На етапі аналізу вимог до застосунку використовувалися методи абстрагування та формалізації. Під час дослідження предметної області використовувалися методи спостереження та порівняння існуючих реалізацій нечіткого пошуку у тексті. На основі отриманих результатів висунута гіпотеза про можли-

вість створення застосунку для виявлення та виправлення орфографічних помилок у тексті із застосуванням сучасних веб-технологій.

Наукова новизна одержаних результатів

Методи що використовуються для дослідження проблеми вже відомі. Проте, використовуючи їх разом з новими підходами та технологіями вони дадуть оптимізацію часу та продуктивності. Серед існуючих альтернатив вже є системи, які мають можливість пошуку у словниках, але вони не підтримують нечіткий пошук, а є ті, які підтримують нечіткий пошук та виправлення помилок у текстових запитах, але вони не надають вільне користування таким функціоналом.

Практичне значення одержаних результатів

На основі результатів досліджень, проведених у рамках даної роботи, розроблено програмний продукт, який дає змогу виконати швидкий аналіз текстових даних на наявність орфографічних помилок, та представити користувачу результати у зручному вигляді. У разі необхідності додаток може бути використаний у роботі пошукових систем або на звичайних веб-сайтах.

Апробація роботи

Основні положення й результати досліджень повідомлені й обговорені на XXII науково-технічній конференції студентів, магістрантів, аспірантів і викладачів ЗДІА на секції програмного забезпечення автоматизованих систем, 2017 р. (м. Запоріжжя).

Публікації

Стаднікова А.Є., магістрант, В.Г. Вербицький, професор, А. І. Безверхий, доцент. «Дослідження проблеми нечіткого пошуку в словниках з використанням методу опорних векторів.» Матеріали XXII науково-технічної конференції студентів, магістрантів, аспірантів і викладачів ЗДІА «ЕНЕРГЕТИКА, ЕЛЕКТРОНІКА ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ» том III, Запоріжжя, ЗДІА, 2017, с. 141.

Глосарій

Архітектура клієнт-сервер – це один із архітектурних шаблонів програмного забезпечення та домінуюча концепція у створенні розподілених мережних застосунків, яка передбачає такі основні компоненти: набір серверів, набір клієнтів, мережа.

Бінарна класифікація – клас задач класифікації елементів набору даних на дві групи на підставі правила класифікації.

Бінарний пошук – алгоритм знаходження заданого значення у впорядкованому масиві, який полягає у порівнянні серединного елемента масиву з шуканим значенням, і повторенням алгоритму для тієї або іншої половини, залежно від результату порівняння.

Відстань Левенштейна (або редакційна відстань) – метрика, що дозволяє визначити «схожість» двох рядків – мінімальна кількість операцій вставки одного символу, видалення одного символу і заміни одного символу на інший, необхідних для перетворення одного рядка в іншу.

Ключові слова (ключові запити) – це слова і словосполучення, за якими користувач знаходить необхідні для нього дані за допомогою пошукових систем.

Метод опорних векторів – це метод аналізу даних для класифікації та регресійного аналізу за допомогою моделей з керованим навчанням з пов'язаними

алгоритмами навчання, які називаються опорно-векторними машинами (ОВМ, англ. support vector machines, SVM).

Метрика – функція, яка визначає відстані в метричному просторі.

Морфологія – розділ граматики, в якому вивчають явища, що характеризують граматичну природу слова як граматичної одиниці мови. Це вчення про будову та граматичні класи слів (частини мови), граматичні категорії і систему їх словозміни.

Морфологічна форма слова – це така зміна слова, при якому зберігається його лексичне значення.

Мультикласифікація – клас задач класифікації елементів набору даних на декілька груп на підставі правила класифікації.

Нечіткий пошук – це пошук інформації, при якому виконується співставлення інформації заданому зразку пошуку або близькому до цього зразка значенню.

Пошуковий запит – запит, який користувач вводить в пошукову систему, щоб задовольнити свої потреби в інформації. Особливість пошукових запитів полягає в тому, що вони являють собою простий текст або гіпертекст з додатковими пошуковими директивами (наприклад, «і» чи «або»).

Пошуковий індекс – структура даних, яка містить інформацію про документи та використовується в пошукових системах. Індексування, що здійснюється пошуковою машиною, – процес збору, сортування та зберігання даних з метою забезпечення швидкого та точного пошуку інформації.

Пошукова машина – комплекс програм, що забезпечує функціональність пошукової системи.

Пошуковий робот – програма, що є складовою частиною пошукової системи та призначена для обходу сторінок інтернету з метою занесення інформації про них (ключових слів) до бази даних.

Пошукова система (або скорочено пошуковик) – онлайн-служба (апаратно-програмний комплекс з веб-інтерфейсом), що надає можливість пошуку інформації в Інтернеті. У просторіччі під пошуковою системою розуміють вебсайт, на котрому розміщено інтерфейс системи. Програмною частиною пошукової системи є пошукова машина.

Ранжування – це процес, результат якого користувач бачить, отримуючи відповідь пошукової системи на своє питання.

Релевантність – це міра відповідності результатів пошуку завданню, поставленому в пошуковому запиті.

Семантична мережа – інформаційна модель предметної області, що має вигляд орієнтованого графа, вершини якого відповідають об'єктам предметної області, а дуги (ребра) задають відносини між ними.

Семантичний словник – це словник, у якому основним є значення слова.

Стемінг – це процес скорочення слова до основи шляхом відкидання допоміжних частин, таких як закінчення чи суфікс. Результати стемінгу іноді дуже схожі на визначення кореня слова.

Хеш-таблиця – це структура даних, яка реалізовує інтерфейс – асоціативний масив, а саме, вона дозволяє зберігати пари (ключ, значення) і здійснювати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення за ключем.

Частотний словник — словник, у якому кожне слово характеризується певним числом, що вказує на кількість вживань цього слова в обстеженому масиві текстів, тобто на його абсолютну частоту в цих текстах.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ НЕЧІТКОГО ПОШУКУ В СЛОВНИКАХ З ВИКОРИСТАННЯМ МЕТОДУ ОПОРНИХ ВЕКТОРІВ

1.1 Аналіз стану питання пошуку в словниках

У глобальній мережі Інтернет можна знайти інформацію з будь-якої теми, що цікавить. Але найважче в роботі з мережею Інтернет – знайти потрібну інформацію. Так як інтернет не має чіткої централізованої структури, розвивається хаотично, і в світі з'являються все нові і нові сервери, питання пошуку інформації стає дуже актуальним.

Пошукові системи значно полегшують роботу в Інтернет і допомагають нам швидко знайти потрібну інформацію в величезному масиві серверів Інтернет (WWW, FTP та ін.). У всесвітній павутині знаходиться кілька тисяч пошукових систем, серед яких є як ті, що вже добре зарекомендували себе, так і менш відомі. З найбільш відомих пошукових машин можна назвати Bing, Google, Yahoo та ін.

Кожна з них має свої переваги і недоліки, які визначаються принципом роботи пошукової машини, зручністю використання, її оформленням, складністю мови запитів, наявністю різних розширених функцій, швидкістю роботи. Вибір пошукової системи для конкретного застосування визначається метою пошуку, характером шуканої інформації та бажаним форматом даних.

Незважаючи на те, якими характеристиками володіє та чи інша система, однією з найголовніших проблем є правильність даних, що вводяться користувачем. Від цього залежить чи знайде користувач необхідну йому інформацію, та який час він витратить на пошук. Тому дослідження та аналіз вхідних текстових даних займають важливе місце у роботі пошукових систем.

1.2 Сучасні пошукові системи

Пошукова система – це комп'ютерна система, призначена для пошуку інформації. Одне з найбільш відомих застосувань пошукових систем – веб-сервіси для пошуку текстової або графічної інформації у Всесвітній павутині. Існують також системи, здатні шукати файли на FTP-серверах, товари в інтернет-магазинах та ін..

Для пошуку інформації за допомогою пошукової системи користувач формулює запит. Робота пошукової системи полягає в тому, щоб за запитом користувача знайти документи, що містять або зазначені ключові слова, або слова, будь-яким чином пов'язані з ключовими словами. При цьому пошукова система генерує сторінку результатів пошуку. Така пошукова видача може містити різні типи результатів, наприклад: веб-сторінки, зображення, аудіо файли. Деякі пошукові системи також витягають інформацію з відповідних баз даних і каталогів ресурсів в Інтернеті.

За методами пошуку і обслуговування розділяють чотири типи пошукових систем:

Системи, що використовують пошукові роботи – складаються з трьох частин: краулер («бот», «робот» або «павук»), індекс і програмне забезпечення пошукової системи. Краулер потрібен для обходу мережі і створення списків веб-сторінок. Індекс – великий архів копій веб-сторінок. Мета програмного забезпечення – оцінювати результати пошуку. Завдяки тому, що пошуковий робот в цьому механізмі постійно досліджує мережу, інформація більшою мірою актуальна. Більшість сучасних пошукових систем є системами даного типу.

Системи, керовані людиною (каталоги ресурсів) – ці пошукові системи одержують списки веб-сторінок. Каталог містить адресу, заголовок і короткий опис сайту. Каталог ресурсів шукає результати тільки з описів сторінки, представлених йому веб-майстрами. Перевага каталогів в тому, що всі ресурси пере-

віряються вручну, отже, і якість контенту буде краще в порівнянні з результатами, отриманими системою першого типу автоматично. Але є і недолік – оновлення даних каталогів виконується вручну і може істотно відставати від реального стану справ. Ранжування сторінок не може миттєво змінюватися. Як приклади таких систем можна привести каталог Yahoo , dmoz і Galaxy.

Гібридні системи – такі пошукові системи, як Yahoo, Google, MSN, поєднують в собі функції систем, що використовують пошукові роботи, і систем, керованих людиною.

Мета-системи – мета пошукові системи об'єднують і ранжують результати відразу декількох пошукових систем. Ці пошукові системи були корисні, коли у кожної пошукової системи був унікальний індекс, і пошукові системи були менш «розумними». Оскільки зараз пошук набагато покращився, потреба в них зменшилася. Приклади: MetaCrawler і MSN Search.

Пошукова система тим краще, чим більше документів, релевантних запиту користувача, вона буде повертати.

1.3 Аналіз існуючих систем пошуку даних у веб-проектах

Будь-який розробник, що реалізує сьогодні певний проект, зустрічається з потребою реалізувати пошук в своєму веб-додатку. Звичайно, для деяких проектів цілком підійде і просте рішення, наприклад, від Google. Але чим складніше додаток, і чим складніша структура контенту, якщо потрібні особливі види пошуку та обробки результату – тим більша потреба у власній реалізації. Далі ми розглянемо різні варіанти пошукових рішень, придатних для вбудовування або розгортання на власному сервері.

1.3.1 Загальна архітектура та терміни

Під пошуковим сервером ми розуміємо бібліотеку або компонент, тобто програмне рішення, яке самостійно веде свою базу даних (насправді це може бути і СУБД, і просто файли) документів, в яких і відбувається пошук. Пошукова система також надає стороннім додаткам можливість додавати, видаляти і оновлювати документи в цій базі. Цей процес називається індексацією, і може бути реалізовано окремим компонентом або сервером (індексатором). Інший компонент, пошуковий механізм, приймає запит на пошук і обробляючи створену базу, виконує вибірку даних, які відповідають запиту. Крім цього, він може обчислювати додаткові параметри для результатів пошуку (ранжувати документи, обчислювати ступінь відповідності пошуковому запиту і т.д.). Це найважливіші системи пошукового сервера, і вони можуть бути як монолітно реалізовані в одній бібліотеці, так і бути самостійними серверами, доступ до яких реалізується через різні прикладні протоколи і API.

Окремо виділяється наявність модуля реалізації веб-пошуку. Тобто, в пошуковому сервері може бути реалізована вбудована можливість отримувати документи з веб-сайтів за протоколом HTTP і заносити їх в індекс. Цей модуль називається зазвичай «павук» або «crawler», і таким чином, пошуковий сервер вже може бути схожий на «справжній» і звичний всім пошук на зразок Google або Yandex. Так можна реалізувати власну пошукову систему за потрібними вам сайтами, наприклад, присвяченим одній темі – досить просто створити список адрес і налаштувати їх періодичний обхід.

Обираючи пошуковий механізм слід враховувати наступні параметри:

- швидкість індексування – тобто, як швидко пошуковий сервер обходить документи і заносить їх в свій індекс, роблячи доступним пошук по ним. Зазвичай вимірюється в Мб чистого тексту в секунду (зазвичай в тестах це 1U сервер, на зразок 2 ГГц CPU і 1 Гб RAM + SATA диск або RAID);

- швидкість переіндексації – в процесі роботи документи змінюються або додаються нові, тому доводиться заново індексувати інформацію. Якщо сервер підтримує інкрементне індексування, то ми обробляємо тільки нові документи, а оновлення всього індексу залишаємо на потім або навіть можемо взагалі не робити;
- підтримуваня API – якщо ви використовуєте пошукову систему в зв'язці з веб-додатком, зверніть увагу на наявність вбудованого API для вашої мови або платформи;
- підтримувані протоколи – зазвичай підтримуються XML-RPC або JSON-RPC, SOAP або доступ через http/socket;
- розмір бази та швидкість пошуку – ці параметри дуже взаємопов'язані і якщо ви реалізуєте щось унікальне і передбачаєте, що у вас можуть бути мільйон і більше документів в базі, то подивіться на відомі реалізації обраної платформи. Хоча ніхто не заявляє явно про обмеження на кількість документів в базах, і на невеликих колекціях (наприклад, кілька десятків тисяч документів) всі пошукачі будуть приблизно однакові, але якщо мова йде про мільйони документів – це може стати проблемою;
- підтримувані типи документів – будь-який сервер підтримує звичайний текст (хоча слід дивитися на можливість роботи з багатомовними документами і кодуванням UTF-8), але якщо вам необхідно індексувати різні типи файлів, наприклад, HTML, XML, DOC або PDF, то варто подивитися на ті рішення, де є вбудований компонент для цього. Звичайно, все це можна зробити прямо в вашому додатку, але краще пошукати готові рішення. Сюди ж відноситься і підтримка індексування і пошуку інформації, яка зберігається в СУБД – не секрет, що таке зберігання найбільш поширене для веб-додатків, і краще, щоб пошуковий сервер працював безпосередньо з базою даних;
- робота з різними мовами і стемінг – для коректного пошуку з використанням різних мов необхідна рідна підтримка не тільки кодувань, а й робота з

особливостями мови. Всі підтримують англійську мову, яка для пошуку і обробки досить проста. Модуль стемінг дозволяє відмінювати і розбирати слова в пошуковому запиті для більш коректного пошуку;

- підтримка додаткових типів полів в документах – крім самого тексту, який індексується і в якому проводиться пошук, необхідна можливість зберігати в документі необмежену кількість інших полів, які зберігають мета-інформацію про документ, що необхідно для подальшої роботи з результатами пошуку. Дуже бажано, щоб кількість і типи полів не обмежувалися, а їх індексованість можна було налаштовувати. Наприклад: в одному полі зберігається назва, у другому анотація, в третьому ключові слова, в четвертому – ідентифікатор документа в системі. Необхідне гнучке налаштовування області пошуку (в кожному полі або тільки в зазначених), а також ті поля, які будуть вилучатись з бази пошукача і видаватися з результатами пошуку;
- платформа і мова – якщо ви збираєтеся виділяти пошук в окремий від програми модуль або сервер, або навіть винесете його на окремий сервер, то роль платформи не така і значна. Зазвичай це або C ++ або Java;
- наявність вбудованих механізмів ранжування і сортування – особливо добре, якщо пошукову систему можна розширювати (і вона написана відомою вам мовою) і писати потрібні вам реалізації цих функцій, адже існує безліч різних алгоритмів, і не факт, що використовуваний підійде за замовчуванням.

1.3.2 Характеристики пошукових рішень

Розглянемо ті пошукові рішення, на які слід звернути увагу, якщо ви вирішили досліджувати пошук даних (див. Табл. 1.1).

Таблиця *1.1*

Характеристики пошукових рішень

	Sphinx	Apache Lucene	Xapian
Тип	окремий сервер або MySQL storage engine	окремий сервер або сервлет, вбудована бібліотека	вбудована бібліотека
Платформа	C++	Java (порти на PHP, C # /. NET, Perl, Ruby, Python)	C++
Індекс	монолітний + дельта-індекс, можливість розподіленого пошуку	інкрементний індекс, але вимагає операції злиття сегментів (оптимізації)	інкрементний «живий» індекс, inmemory індекси для невеликих баз
Варіанти пошуку	булевий пошук, пошук за фразами, врахування близькості слів	булевий пошук, пошук за фразами, нечіткий пошук, врахування близькості слів, пошук за маскою	булевий пошук, пошук за фразами, пошук з ранжируванням, пошук за маскою, пошук за синонімами
API та протоколи	SQL DB, власний XML-протокол, вбудовані API для PHP, Ruby, Python, Java, Perl	Java API	C++, Perl API, Java JINI, Python, PHP, TCL, C# и Ruby, CGI інтерфейс з XML/CSV фо-

	Sphinx	Apache Lucene	Xapian
			рматом
Підтримка мов	вбудований англійський та російський стемінг, soundex для реалізації морфології	відсутня морфологія, є стемінг (Snowball) і аналізатори для ряду мов (включаючи російську)	відсутня морфологія, є стемінг для ряду мов (включаючи російську), перевірка правопису в пошукових запитах
Додаткові поля документів	необмежена кількість	необмежена кількість	відсутні
Формати	лише текст або SQL DB	текст, можливо індексація бази даних через JDBC	лише текст
Розмір індексу / швидкість	дуже швидкий, індексація близько 10 Мб / сек (залежить від CPU), пошук близько 0.1 сек / ~ 2 - 4 Гб індексі, підтримує розміри індексу в сотні Гб і сотні мільйонів документів, однак є приклади робіт на те-	близько 20 Мб / хвилина, розмір індексних файлів обмежений 2 Гб (на 32-bit ОС). Є можливості паралельного пошуку по кількох індексам і кластеризація (вимагає сторонніх платформ)	тести швидкості на офф. сайті відсутні. Відомі працюють інсталяції на 1.5 Тб індексу

	Sphinx	Apache Lucene	Xapian
	рабайтних базах даних		
Ліцензія	GPL 2 або комер-ційна	Apache License 2.0	GPL
URL	sphinxsearch.com	lucene.apache.org	xapian.org

Sphinx search engine, ймовірно, найпотужніший і найшвидший з усіх відкритих движків, які ми розглядаємо. Особливо зручний тим, що має пряму інтеграцію з популярними базами даних і підтримує розвинені можливості пошуку, включаючи ранжування і стемінг для російської та англійської мови. Підтримуються і нетривіальні можливості на зразок розподіленого пошуку та кластеризації, проте фірмовою рисою є дуже і дуже висока швидкість індексації та пошуку, а також здатність відмінно утилізувати ресурси сучасних серверів. Відомі дуже серйозні інсталяції, що містять терабайти даних, тому Sphinx цілком можна рекомендувати як виділений пошуковий сервер для проектів будь-якого рівня складності і обсягу даних. Прозора робота з найпопулярнішими базами даних MySQL і PostgreSQL дозволяє його використовувати в звичайному для веб-розробки оточенні, до того ж відразу «з коробки» є API для різних мов, в першу чергу, для PHP. Але сам пошуковик необхідно компілювати і встановлювати окремо, тому на звичайному хостингу він непридатний – тільки VDS або власний сервер, причому бажано побільше пам'яті. Індекс у пошукача монолітний, тому доведеться трохи налаштувати дельта-індекс для коректної роботи у випадку, коли дуже багато нових або змінених документів, хоча величезна швидкість індексації дозволяє організувати перебудову індексу за розкладом і це не позначиться на роботі пошуку.

SphinxSE – це версія, яка функціонує як движок зберігання даних для MySQL (вимагає патча і перекомпіляції бази), Ultrasphinx – конфігуратор і кліт-

єнт для Ruby (крім присутнього в дистрибутиві API), крім цього є плагіни для багатьох відомих CMS і блог-платформ, які замінюють стандартний пошук (повний список).

Apache Lucene – найвідоміший з пошукових движків, споконвічно орієнтований саме на вбудовування в інші програми. Зокрема, його широко використовують в Eclipse (пошук по документації) і навіть в IBM (продукти з серії OmniFind). В плюсах проекту – розвинені можливості пошуку, хороша система побудови і зберігання індексу, який може одночасно поповнюватися і оптимізуватися разом з пошуком. Доступний і паралельний пошук за безліччю індексів з об'єднанням результатів. Сам індекс побудований із сегментів, проте для поліпшення швидкості рекомендується його періодично оптимізувати. Спочатку присутні варіанти аналізаторів для різних мов, включаючи російську з підтримкою стемінгу (приведення слів до нормальної формі). Однак мінусом є все ж дуже низька швидкість індексації (особливо в порівнянні з Sphinx), складність роботи з базами даних і відсутність API (крім Java). І хоча для досягнення серйозних показників Lucene може кластеризуватися і зберігати індекси в розподіленій файловій системі або базі даних, для цього потрібні сторонні рішення, так само як і для всіх інших функцій – наприклад, спочатку він вміє індексувати тільки звичайний текст. Але саме в плані використання в складі сторонніх продуктів Lucene «попереду планети всієї» – жоден інший движок не має стільки портів на інші мови. Одним з чинників такої популярності є і дуже вдалий формат файлів індексів, який використовують сторонні рішення.

Solr – найкраще рішення на базі Lucene, значно розширює її можливості. Це самостійний сервер корпоративного рівня, що надає широкі пошукові можливості в якості веб-сервісу. Стандартно Solr приймає документи по протоколу HTTP в форматі XML і повертає результат також через HTTP (XML, JSON або інший формат). Повністю підтримується кластеризація і реплікація на кілька серверів, розширена підтримка додаткових полів в документах (на відміну від

Lucene, для них підтримуються різні стандартні типи даних, що наближає індекс до баз даних), підтримка фасетного пошуку і фільтрації, розвинені засоби адміністрування, а також можливості кешування і бекапу індексу в процесі роботи. З одного боку, це самостійне рішення на базі Lucene, з іншого – її можливості істотно розширені щодо базових.

Nutch – інший найвідоміший проект на базі Lucene. Це веб-пошуковий движок (пошуковий механізм + веб-павук для обходу сайтів) суміщений з розподіленою системою зберігання даних Hadoop. Nutch «з коробки» може працювати з віддаленими вузлами в мережі, індексує не тільки HTML, але і MS Word, PDF, RSS, PowerPoint і навіть MP3 файли (мета-теги, звичайно). Але водночас присутнє значне урізання функціоналу Lucene, наприклад не підтримуються булеві оператори в пошуку, не використовується стемінг. Якщо стоїть завдання зробити невеликий локальний пошуковик за місцевими ресурсами або заздалегідь обмеженому наборі сайтів, при цьому потрібен повний контроль над усіма аспектами пошуку, або ви створюєте дослідний проект для перевірки нових алгоритмів, в такому випадку Nutch стане вашим кращим вибором. Однак треба враховувати його вимоги до апаратної частини.

Lucene відомий і популярний не тільки за рахунок проектів-надбудов. Будучи лідером серед відкритих рішень і втіливши в себе безліч відмінних алгоритмів, Lucene перший кандидат в портування на інші платформи і мови. Зараз є такі порти:

- *Lucene. Net* – повний порт Lucene, повністю алгоритмічно, за класами та API ідентичне перенесення на платформу MS. NET / Mono і мову C#;
- *Ferret* – порт на мову Ruby;
- *CLucene* – версія на мові C++, що обіцяє дати істотний приріст продуктивності. За деякими тестами, він швидше оригіналу в 3-5 разів, а іноді і більше (на індексації, пошук порівняний або швидше на всього 5-10%). Виявилося, що цю версію використовує велика кількість проектів і компаній – Dig, Flock, Kat

(пошуковик для KDE), BitWeaver CMS і навіть такі компанії, як Adobe (пошук по документації) і Nero;

- *Plucene* – реалізація на Perl;
- *PyLucene* – реалізація для Python-додатків, проте не повна і вимагає java;
- *Zend_Search_Lucene* – єдиний порт на мову PHP, доступний в складі Zend Framework.

Xapian поки є єдиним претендентом на конкуренцію з Lucene і Sphinx, вигідно відрізняється від них наявністю «живого» індексу, що не потребує перебудови при додаванні документів, дуже потужною мовою запитів, включаючи вбудований стемінг, перевірку орфографії, і навіть підтримку синонімів. Однак немає ніякої інформації про можливості додавати до документів довільні додаткові поля і отримувати їх з результатами пошуку, тому зв'язок цієї системи пошуку з власною може представляти певні труднощі. У пакет входить Omega – надбудова над бібліотекою, яка готова для використання в якості самостійного пошукача і саме вона відповідає за можливості індексації різних типів документів і CGI інтерфейс.

Хоча існує ще безліч пошукових механізмів, частина з них є портами або надбудовами над вже розглянутими. Наприклад, промислового рівня пошуковий сервер для власної CMS компанії eZ, ezFind насправді не окремий пошуковик, а інтерфейс до стандартного Lucene Java і включає його в свою поставку. Це ж стосується і компонента Search з їх пакету eZ Components – він надає уніфікований інтерфейс для доступу до зовнішніх пошукових серверів. І навіть таке цікаве і потужне рішення, як Carrot і SearchBox – це серйозно модифіковані версії тієї ж Lucene, значно розширені і доповнені новими можливостями. Самостійних же пошукових рішень, з відкритим кодом, які повністю реалізують індексацію і пошук по власних алгоритмах на ринку не так і багато.

1.4 Аналіз існуючих алгоритмів нечіткого пошуку

Розглянемо існуючі алгоритми нечіткого пошуку і відберемо найбільш актуальні на даний момент для подальшого аналізу.

1.4.1 Алгоритм розширення вибірки

Алгоритм розширення вибірки – часто застосовується в системах перевірки орфографії [1]. Він ґрунтується на зведенні задачі нечіткого пошуку до задачі точного пошуку. Даний метод передбачає побудову найбільш ймовірних «неправильних» варіантів пошукового шаблону. Тобто будується безліч різноманітних «помилкових» слів (див. Рис. 1.1), наприклад, що виходять з початкового слова в результаті однієї операції редагування, після чого побудовані терміни порівнюються на точну відповідність.

Основний плюс даного алгоритму полягає в легкості його модифікації для генерації «помилкових» варіантів по довільним правилам. Однак є і мінуси, головний з яких – велика кількість перевірок для слів суттєвою довжини, оскільки з них можна отримати багато «помилкових» слів.

Час його роботи сильно залежить від числа k помилок і від розміру алфавіту A, і в разі використання бінарного пошуку по словнику становить:

$$O((m|A|)^k * m * \log(n)).$$

Наприклад, при k = 1 і слова довжини 7 (наприклад, «Крокодил») в українському алфавіті безліч помилкових слів буде розміром близько 450, тобто буде необхідно зробити 450 запитів до словника, що цілком прийнятно.

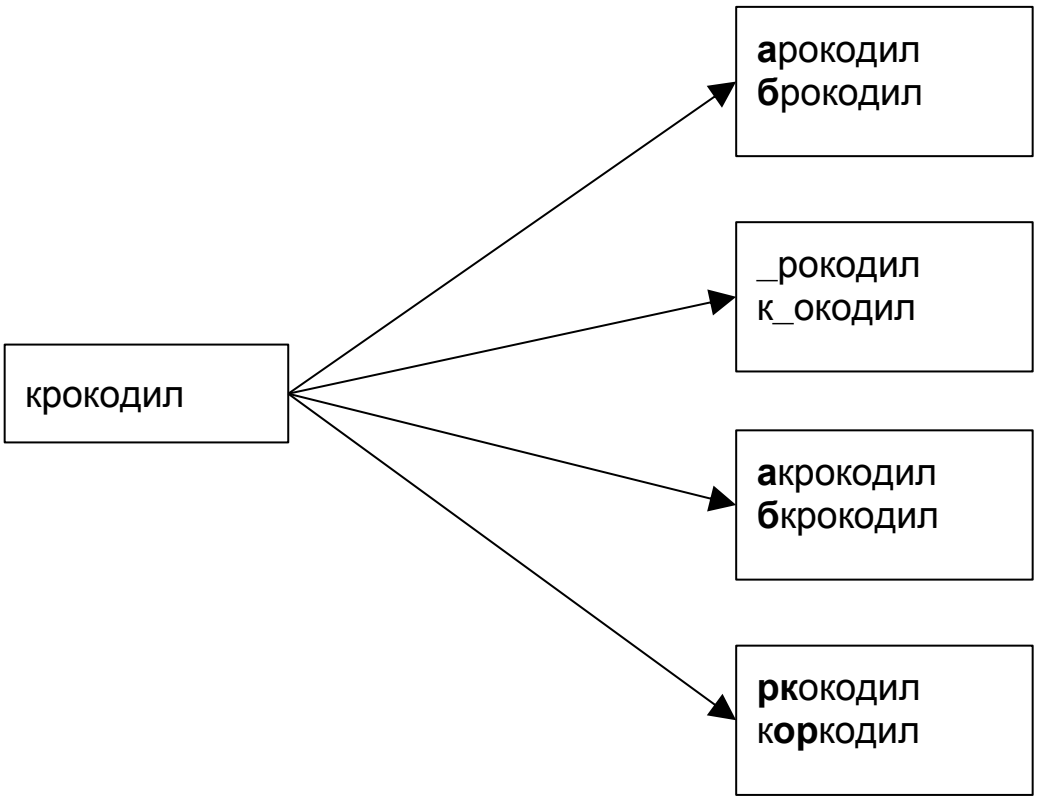


Рис. *1.1 Приклад алгоритму розширення вибірки*

Але вже при k = 2 розмір такої безлічі становитиме понад 115 тисяч варіантів, що відповідає повному перебору невеликого словника, або ж 1/27 в нашому випадку, і, отже, час роботи буде досить довгим. При цьому не потрібно забувати ще й про те, що для кожного з таких слів необхідно провести пошук на точний збіг в словнику.

В якості покращення алгоритму можна генерувати не всю безліч «помилкових» слів, а лише ті з них, які найбільш ймовірно можуть зустрітися в реальній ситуації, наприклад, слова з урахуванням поширення орфографічних помилок або помилок набору.

1.4.2 Алгоритм хешування за сигнатурою

Іншим варіантом реалізації може бути алгоритм хешування за сигнатурою. Даний метод ґрунтується на складанні спеціальних хеш-таблиць слів [2]. При цьому, кожному слову в тексті у відповідність ставиться бітова маска (сигнатура).

Сигнатурою *sign(w)* слова *w* будемо називати вектор розмірності m, k-й елемент якого дорівнює одиниці, якщо у слові *w* є символ *a* такий, що *f(a) = k,* і нулю в іншому випадку. Номером сигнатури слова будемо позначати число

$$H(w) = \sum_{i=0}^{m-1} 2^i \, sign(w)_{i+1}.$$

При індексації такий хеш обчислюється для кожного зі слів, і в таблицю заноситься відповідність списку словникових слів цього хешу. Потім, під час пошуку, для запиту обчислюється хеш і перебираються всі сусідні хеші, що відрізняються від вихідного не більше ніж в k бітах. Для кожного з таких хешів проводиться перебір списку відповідних йому слів.

Процес обчислення хешу – кожному біту хешу зіставляється група символів з алфавіту. Біт 1 на позиції і в хеші означає, що в початковому слові присутній символ з *i*-ої групи алфавіту. Порядок букв в слові ніякого значення не має (див. Рис. 1.2).

Видалення одного символу або не змінить значення хешу (якщо в слові ще залишилися символи з тієї ж групи алфавіту), або ж відповідний цій групі біт зміниться в 0. При вставці, аналогічним чином або один біт встане в 1, або ніяких змін не буде. При заміні символів все трохи складніше – хеш може або зовсім залишитися незмінним, або ж зміниться в 1 або 2 позиціях. При перестановках ніяких змін і зовсім не відбувається, тому що порядок символів при побудові хешу, як і було зазначено раніше, не враховується.

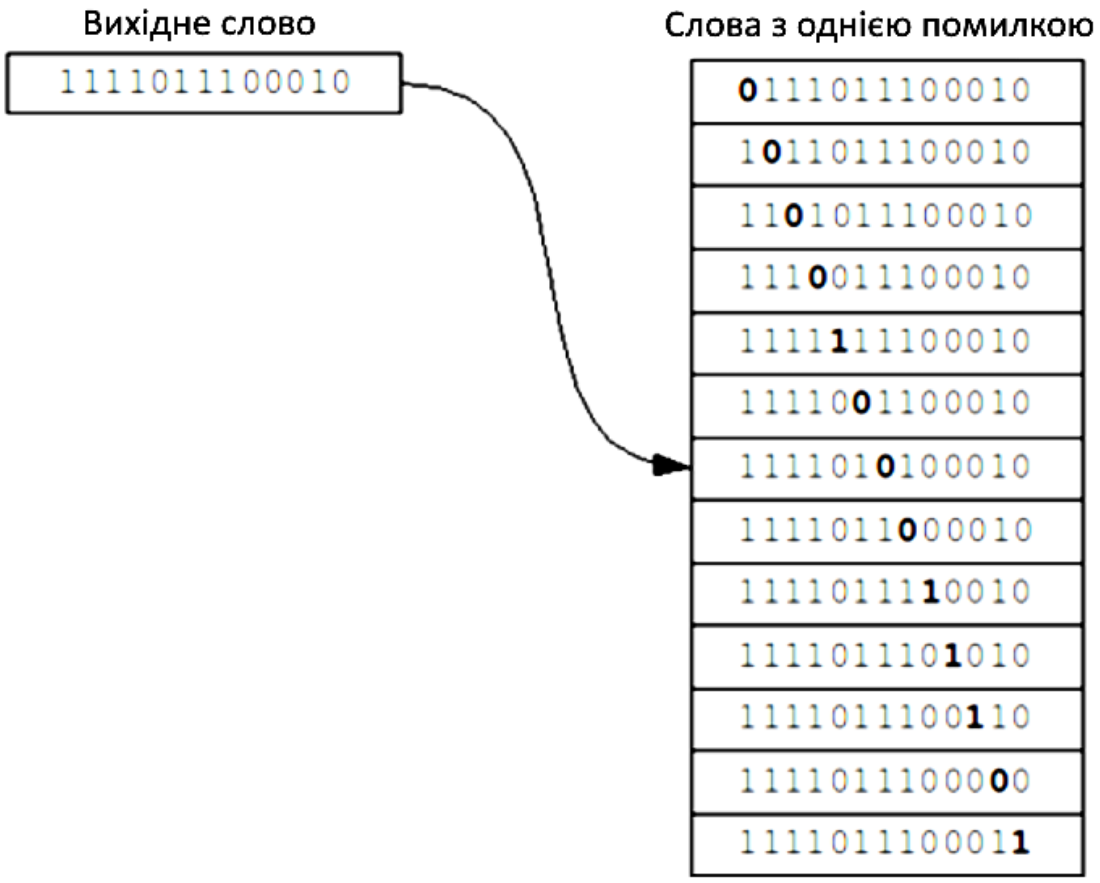
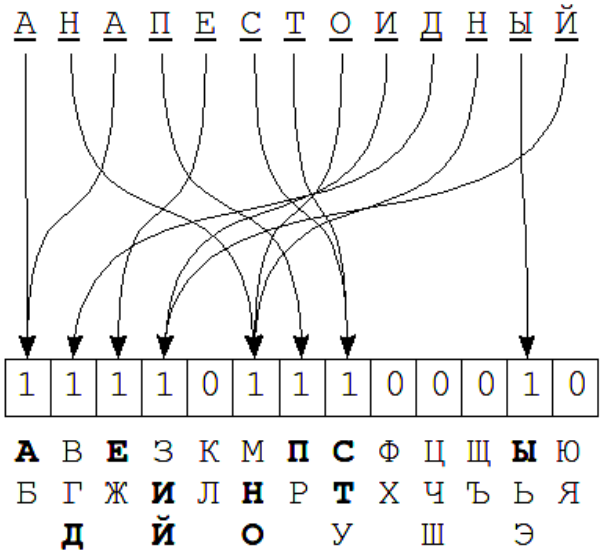


Рис. *1.2 Приклад заміни біт у хеші*

Таким чином, для повного покриття k помилок потрібно змінювати не менше 2k біт у хеші (див. Рис. 1.3).

Через те, що при заміні одного символу можуть змінюватися відразу два біта, алгоритм, який реалізує, наприклад, спотворення не більше 2 бітів одночасно, в дійсності не видаватиме повного обсягу результатів через відсутність значної (залежить від відношення розміру хешу до алфавіту) частини слів з двома замінами (і чим більше розмір хешу, тим частіше заміна символу буде приводити до спотворення відразу двох біт, і тим менш повним буде результат). До того ж, цей алгоритм не дозволяє проводити префіксний пошук.

Час роботи, в середньому, при k «неповних» (вставки, видалення і транспозиції, а також мала частина замін) помилках: $O(|H|^k \cdot \frac{n}{2^{|H|}})$.



Хеш	Список слов
0000000000000	-
...	
1111011100001	АВТОПРЕДПРИЯТИЕ, БЕЗОТРАДНАЯ, ВЕТЕРИНАРИЯ, ...
1111011100010	ПРЕВРАТНЫЙ, БЕЗРАССУДНЫЙ, АНАПЕСТОИДНЫЙ, ...
1111011100011	СОРИЕНТИРОВАТЬСЯ, БЕСПРЕПЯТСТВЕННЫЙ, ...
...	
1111111111111	ЛЕГКОИСЧЕРПЫВАЮЩИХСЯ, ВЫСОКОРАЗРЕШАЮЩИХ, ...

Рис. *1.3 Приклад зіставлення групи символів кожному біту хешу*

Даний метод вимагає значних витрат часу на створення хеш-таблиць, що також ускладнює його використання на мобільних застосунках, але він досить простий при реалізації та дозволяє виконувати швидкий пошук як при повній відповідності слова, так і при наявності однієї чи двох помилок у слові. Проблема затрат часу може бути вирішена шляхом створення готових індексів по текстам чи словникам.

1.4.3 Коди Хемінга

Для нечіткого пошуку можуть бути використані широко відомі коди Хемінга. Коди Хемінга давно і успішно застосовуються при кодуванні і декодуванні інформації, дозволяючи успішно відновити втрачену при передачі інформацію. Особливо ефективно коди Хемінга працюють разом з апаратом нечіткої логіки.

Нейронні мережі Хемінга так само активно використовуються для оптичного розпізнавання символів (OCR) та можуть бути застосовані не тільки в теорії кодування інформації, а й в питаннях інформаційного пошуку. Принцип роботи мереж Хемінга заснований на визначенні відстані між об'єктами і знаходженні найбільш близького.

Слід зазначити, що, незважаючи на велику ефективність кодів Хемінга, вони не позбавлені певних недоліків. Лінійні коди, як правило, добре справляються з рідкісними і великими помилками. Однак, їх ефективність при порівнянні слів з частими, але невеликими помилками, менш висока. Також варто звернути увагу на те, що в даному алгоритмі присутні додаткові витрати на кодування інформації.

1.4.4 Тріангуляційні дерева

Наступний з розглянутих алгоритмів не зовсім підходить для поставленої задачі, але згадати про нього все ж можна. Це алгоритм, який використовує тріангуляційні дерева, які дозволяють індексувати безліч довільних структур за умови, що на них задана метрика. Існує досить багато різних модифікацій даного методу, але всі вони не надто ефективні в разі текстового пошуку та частіше використовуються в базі даних зображень або інших складних об'єктів.

1.4.5 Метод N-грам

Цей метод був придуманий досить давно, і є найбільш широко використовуваним, так як його реалізація дуже проста, і він забезпечує досить добру продуктивність. Алгоритм ґрунтується на принципі: «Якщо слово А збігається зі словом Б з урахуванням декількох помилок, то з великою часткою ймовірності у них буде хоча б один загальний підрядок довжини N». Ці підрядки довжини N і називаються N-грамами [3].

Під час індексації слово розбивається на такі N-грами, а потім це слово потрапляє в списки для кожної з цих N-грам. Під час пошуку запит також розбивається на N-грами, і для кожної з них виконується послідовний перебір списку слів, що містять такий підрядок.

Найбільш часто використовуваними на практиці є триграми – підрядки довжини 3 (див. Рис. 1.4).

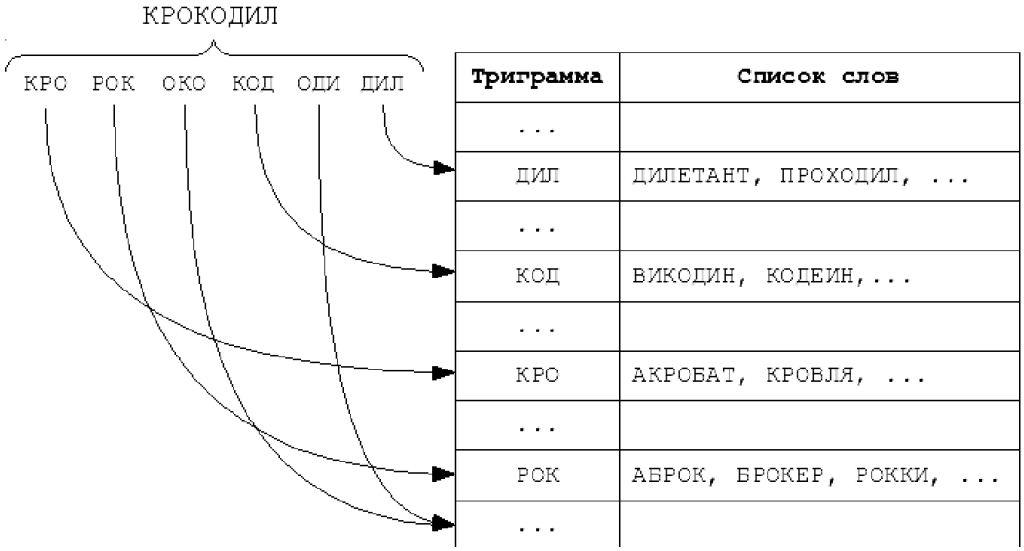


Рис. *1.4 Метод N-грам на прикладі триграм*

Вибір більшого значення N веде до обмеження на мінімальну довжину слова, при якій вже можливе виявлення помилок.

Метод N-грам не завжди знаходить помилки. Дана проблема виникає, наприклад, коли користувач помилився в одній букві в корні слова при вводі. Також даний метод при зверненні до бази даних кожен раз буде виконувати перебір 10-20% інформації. При наявності словників великих розмірів даний метод буде довго виконуватись и вимагати значних ресурсів та потужності.

До можливих покращенньь можна віднести розбиття хеш-таблиці N-грам по довжині слів і по позиції N-грами в слові (див. Рис. 1.5). Як довжина шуканого слова і запиту не можуть відрізнятися більш ніж на k, так і позиції N-грами в слові можуть відрізнятися не більше ніж на k. Таким чином, необхідно буде перевірити лише таблицю, що відповідає позиції цієї N-грами в слові, а також k таблиць ліворуч і k таблиць праворуч, тобто всього 2k + 1 сусідніх таблиць.

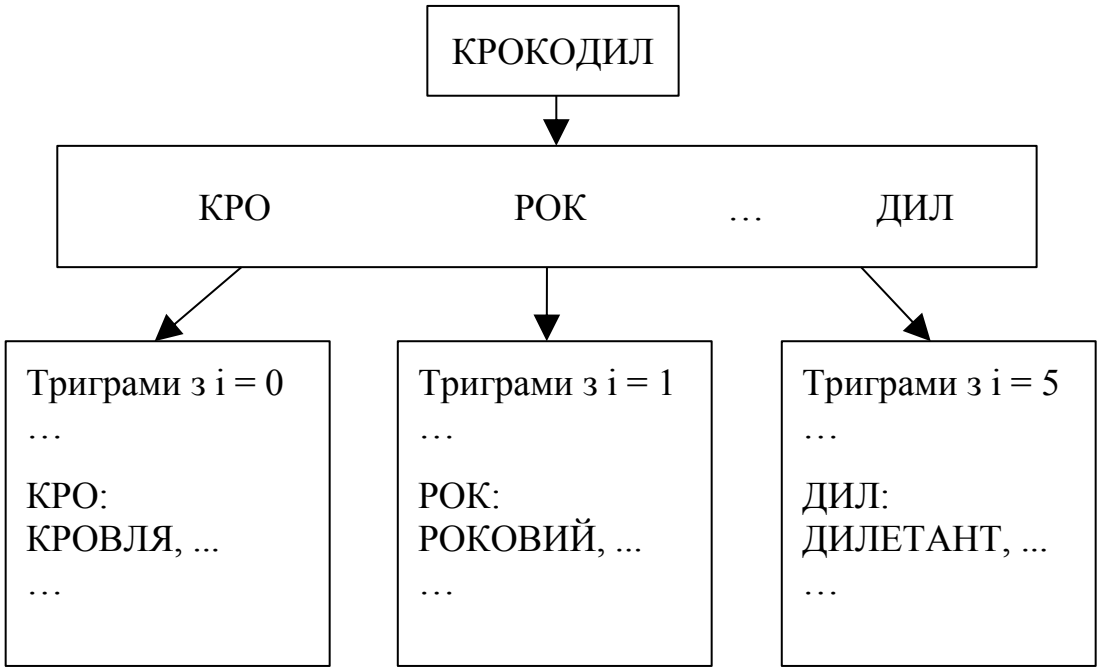


Рис. *1.5 Покращення методу N-грам*

Можна ще трохи зменшити розмір необхідної для перегляду безлічі, розбивши таблиці по довжині слова, і аналогічним чином переглядаючи тільки сусідні 2k + 1 таблиці.

1.4.6 BK-дерева

При оптимізації нечіткого пошуку часто виникає питання з організації даних таким чином, щоб пошук виконувався швидше і витрачав менше ресурсів. Одним з варіантів подібної організації даних є дерева Burkhard-Keller (див. Рис. 1.6). Це метричні дерева, алгоритми побудови яких засновані на властивості метрики відповідати нерівності трикутника: $p(x, y) \leq p(x, z) + p(z, y), \quad x, y, z \in X.$

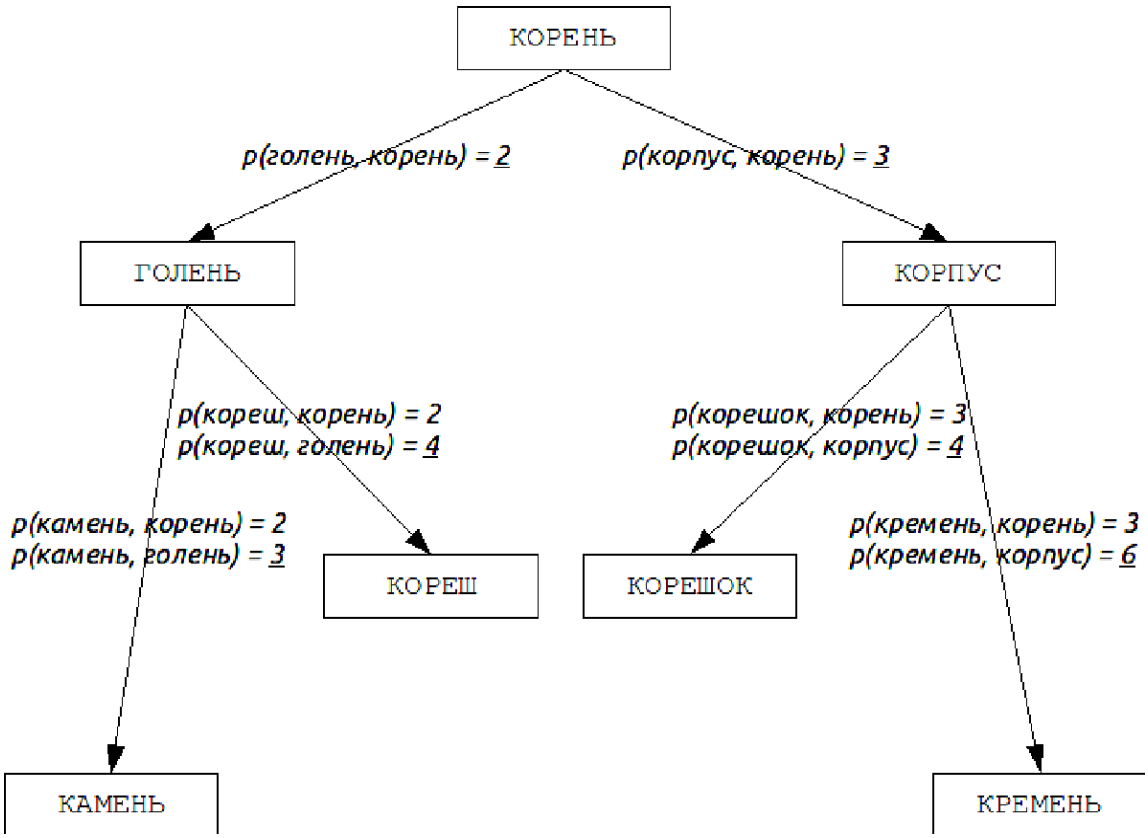


Рис. *1.6 Організація BK-дерева*

Ця властивість дозволяє метрикам утворювати метричний простір довільної розмірності.

Такі метричні простори не обов'язково є евклідовими, так, наприклад, метрики Левенштейна і Дамерау-Левенштейна утворюють неевклідові простори. На основі цих властивостей можна побудувати структуру даних, яка здійснює пошук в такому метричному просторі, яким і є дерева Баркхарда-Келлера.

Для покращення можна використовувати можливість деяких метрик обчислювати відстань з обмеженням, встановлюючи верхню межу, що дорівнює сумі максимальної відстані до нащадків вершини і результуючої відстані, що дозволить трохи прискорити процес:

$$\mathrm{P}_{limit} \leq max_{children} d(this, child) + k.$$

Даний метод дозволяє скоротити час виконання пошуку за рахунок більш зручної структури даних, але ускладнює алгоритм пошуку.

1.4.7 Алгоритм Вагнера-Фішера

Далі розглянемо алгоритм Вагнера-Фішера, який дозволяє для двох рядків знайти відстань Левенштейна – мінімальна кількість операцій вставки одного символу, видалення одного символу і заміни одного символу на інший, необхідних для перетворення одного рядка в іншу [4].

Для відстані Левенштейна справедливі наступні твердження:

- $d(S_1, S_2) \geq \big||S_1| - |S_2|\big|$
- $d(S_1, S_2) \leq \max(|S_1|, |S_2|)$
- $d(S_1, S_2) = 0 \leftrightarrow S_1 = S_2$,

Де $d(S_1, S_2)$ – відстань Левенштейна між рядками S_1 та S_2, а $|S|$ – довжина рядка S.

Відстань Левенштейна є метрикою[5]. Для того щоб довести це досить довести що виконується нерівність трикутника:

$$d(S_1, S_3) \leq d(S_1, S_2) + d(S_2, S_3).$$

Нехай $d(S_1, S_3) = x$, $d(S_1, S_2) = y, d(S_2, S_3) = z$,

де x – найкоротша редакційна відстань від до S_3,

y – найкоротша редакційна відстань від S_1 до S_2,

z – найкоротша редакційна відстань від S_2 до S_3,

$y + z$ – якась відстань від S_1 до S_3.

В інших випадках $d(S_1, S_3) < d(S_1, S_2) + d(S_2, S_3)$. Отже, виконується нерівність трикутника.

Ціни операцій можуть залежати від виду операції (вставка, видалення, заміна) і/або від символів, що беруть в ній участь, відображаючи різну ймовірність різних помилок при введенні тексту, і т.д. У загальному випадку:

- $\omega(a, b)$ – ціна заміни символа a на символ b;
- $\omega(\varepsilon, b)$ – ціна вставки символа b;
- $\omega(a, \varepsilon)$ – ціна видалення символа a.

Для вирішення завдання про редакційну відстань, необхідно знайти послідовність замін, що мінімізувала сумарну ціну. Відстань Левенштейна є окремим випадком цієї задачі при $\omega(a, a) = 0$:

- $\omega(a, b) = 1$, при $a \neq b$;
- $\omega(\varepsilon, b) = 1$;
- $\omega(a, \varepsilon) = 1$,

де ε – порожня послідовність.

Як окремий випадок, так і завдання для довільних *w*, вирішує алгоритм Вагнера-Фішера, наведений нижче. Тут і нижче ми вважаємо, що усі *w* невід'ємні, і діє правило трикутника: якщо дві послідовні операції можна замінити однією, це не погіршує загальну ціну (наприклад, замінити символ *x* на *y*, а потім з *y* на *z* не краще, ніж відразу *x* на *z*).

Будемо вважати, що елементи рядків нумеруються з першого, як прийнято в математиці, а не з нульового.

Нехай S_1 и S_2 – два рядка (довжиною *M i N* відповідно) над деяким алфавітом, тоді редакційну відстань $d(S_1, S_2)$ можна підрахувати за такою рекурент-

ною формулою: $d(S_1, S_2) = D(M, N)$, де $\min(a, b, c)$ повертає найменший з аргументів.

$$\begin{cases} 0; \quad i = 0, j = 0 \\ i; \; i > 0, j = 0 \\ j; \; i = 0, j > 0 \\ D(i-1, j-1); \; S_1[i] = S_2[j] \\ \min(\\ D(i, j-1) + insertCost \\ D(i-1, j) + deleteCost \\ D(i-1, j-1) + replaceCost \\ ); \\ i > 0, j > 0, S_1[i] \neq S_2[j] \end{cases}$$

Доведення:

Розглянемо формулу більш докладно. Тут *D (i, j)* – відстань між префіксами рядків: першими i символами рядка S_1 і першими j символами рядка S_2. Для нашої динаміки повинен виконуватися принцип оптимальності на префіксі. Очевидно, що редакційна відстань між двома порожніми рядками дорівнює нулю. Так само очевидним є те, що щоб отримати порожній рядок з рядка довжиною i, потрібно зробити i операцій видалення, а щоб отримати рядок довжиною j з порожньою, потрібно зробити j операцій вставки. Залишилося розглянути нетривіальний випадок, коли обидва рядки непорожні.

Для початку зауважимо, що в оптимальній послідовності операцій, їх можна довільно міняти місцями. Справді, розглянемо дві послідовні операції:

- дві заміни одного і того ж символу – не оптимально (якщо ми замінили x на y, потім y на z, вигідніше було відразу замінити x на z);
- дві заміни різних символів можна міняти місцями;
- два стирання або дві вставки можна міняти місцями;

- вставка символу з його подальшим стиранням – не оптимально (можна їх обидві скасувати);
- стирання і вставку різних символів можна міняти місцями;
- вставка символу з його подальшою заміною – не оптимально (зайва заміна);
- вставка символу і заміна іншого символу міняються місцями;
- заміна символу з його подальшим стиранням – не оптимально (зайва заміна);
- стирання символу і заміна іншого символу міняються місцями.

Нехай S_1 закінчується на символ a, S_2 закінчується на символ b. Є три варіанти:

1. Символ «a», на який закінчується S_1, в якийсь момент був стертий. Зробимо це стирання першою операцією. Тоді ми стерли символ a, після чого перетворили перші *i-1* символів S_1 в S_2 (на що було потрібно *D (i-1, j)* операцій), значить, всього було потрібно *D (i-1, j) + 1* операцій.

2. Символ b, на який закінчується S_2, в якийсь момент був доданий. Зробимо це додавання останньою операцією. Ми перетворили S_1 в перші *j-1* символів S_2, після чого додали b. Аналогічно попередньому випадку, треба було *D (i, j-1) + 1* операцій.

3. Обидва попередніх твердження невірні. Якщо ми додавали символи праворуч від фінального *a*, то щоб зробити останнім символом *b*, ми повинні були або в якийсь момент додати його (але тоді твердження 2 було б вірним), або замінити на нього один з цих доданих символів (що теж неможливо , тому що додавання символу з його подальшою заміною неоптимальне). Отже, символів праворуч від фінального *a* ми не додавали. Фінального *a* ми не стерли, оскільки твердження 1 невірно. Значить, єдиний спосіб зміни останнього символу – його заміна. Замінювати його 2 або більше разів не оптимально. Отже,

- Якщо $a = b$, ми останній символ не змінювали. Оскільки ми його теж не стирали і не приписували нічого праворуч від нього, він не впливав на наші дії, і, значить, ми виконали $D(i-1, j-1)$ операцій.
- Якщо $a \neq b$, ми останній символ змінювали один раз. Зробимо цю заміну першої. Надалі, аналогічно попередньому випадку, ми повинні виконати $D(i-1, j-1)$ операцій, значить, все буде потрібно $D(i-1, j-1) + 1$ операцій.

Даний алгоритм має ряд значних переваг перед усіма описаними до цього, а саме: відносно невисоку складність реалізації, можливість якісного порівняння схожості більш ніж двох рядків, кілька варіантів реалізації, які можна використовувати в залежності від конфігурації системи, універсальність для всіляких алфавітів. До недоліків можна віднести те, що при перестановці місцями слів або їх частин виходять порівняно великі відстані. Значення між абсолютно різними короткими словами виявляються маленькими, в той час як між схожими і довгими рядками – значними.

1.4.8 Алгоритм Bitap та його модифікації

Алгоритм Bitap і різні його модифікації найбільш часто використовуються для нечіткого пошуку без індексації. Його варіація використовується, наприклад, в unix-утиліті agrep, яка виконує функції аналогічно стандартному grep, але з підтримкою помилок в пошуковому запиті і навіть надаючи обмежені можливості для застосування регулярних виразів.

Вперше ідею цього алгоритму запропонували громадяни Ricardo Baeza-Yates і Gaston Gonnet, опублікувавши відповідну статтю в 1992 році.

Оригінальна версія алгоритму має справу тільки з замінами символів, і, фактично, обчислює відстань Хеммінга. Але трохи пізніше Sun Wu і Udi Manber запропонували модифікацію цього алгоритму для обчислення відстані Левенш-

тейна, тобто привнесли підтримку вставок і вилучень, і розробили на його основі першу версію утиліти agrep.

Операція вставки:

$$R^k_{\text{вставки } j+1} = (Bitshift(R^k_j) \cup s_x) \cap R^{k-1}_j$$

Операція видалення:

$$R^k_{\text{видалення } j+1} = (Bitshift(R^k_j) \cup s_x) \cap Bitshift(R^{k-1}_{j+1})$$

Операція заміни:

$$R^k_{\text{заміни } j+1} = (Bitshift(R^k_j) \cup s_x) \cap Bitshift(R^{k-1}_j)$$

Результуюче значення:

$$R^k_{j+1} = R^k_{\text{вставки } j+1} \cup R^k_{\text{видалення } j+1} \cup R^k_{\text{заміни } j+1})$$

Де k – кількість помилок, j - індекс символу, sx - маска символу (в масці поодинокі біти розташовуються на позиціях, що відповідають позиціям даного символу в запиті). Збіг чи розбіжність запиту визначається останнім бітом результуючого вектора R (див. Рис. 1.7).

Висока швидкість роботи цього алгоритму забезпечується за рахунок бітового паралелізму обчислень – за одну операцію можливо провести обчислення над 32 і більше бітами одночасно.

$$R = \begin{matrix}0\\0\\0\\0\\0\end{matrix} \xrightarrow{Bitshift} \begin{matrix}1\\0\\0\\0\\0\end{matrix} \xrightarrow{Bitshift} \begin{matrix}1\\1\\0\\0\\0\end{matrix} \xrightarrow{Bitshift} \begin{matrix}1\\1\\1\\0\\0\end{matrix}$$

Рис. *1.7 Операція Bitshift*

При цьому тривіальна реалізація підтримує пошук слів довжиною не більше 32. Це обмеження обумовлюється шириною стандартного типу int (на 32-бітних архітектурах). Можна використовувати і типи великих розмірностей, але це може в деякій мірі уповільнити роботу алгоритму.

Не дивлячись на те, що асимптотичний час роботи цього алгоритму O(kn) збігається з часом роботи лінійного методу, він значно швидший при довгих запитах і кількості помилок k більше 2.

1.4.9 Метод опорних векторів

Даний метод відноситься до бінарних класифікаторів, хоча існують способи змусити його працювати і для задач мультикласифікації [6].

Ідея методу: дані точки на площині, розбиті на два класи. Проведемо лінію, що розділяє ці два класи (Рис. 1.8).

Далі, усі нові точки (не з навчальної вибірки) автоматично класифікуються наступним чином:

- точка вище прямої потрапляє в клас A;
- точка нижче прямої – в клас B.

Таку пряму назвемо поділяюча пряма. Однак, в просторах вищих розмірностей пряма вже не буде розділяти наші класи, так як поняття «нижче прямої» або «вище прямої» втрачає будь-який сенс. Тому замість прямих необхідно розглядати гіперплощини - простору, розмірність яких на одиницю менше, ніж розмірність початкового простору.

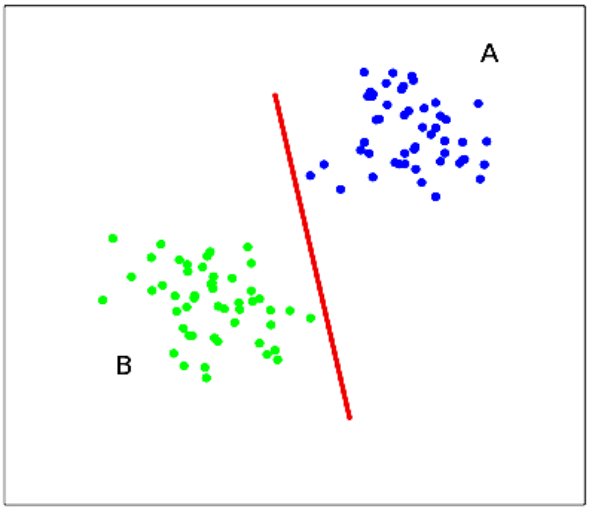


Рис. 1.8 *Метод опорних векторів з бінарною класифікацією*

У наступному прикладі існує кілька прямих, які поділяють два класи (див. Рис. 1.9).

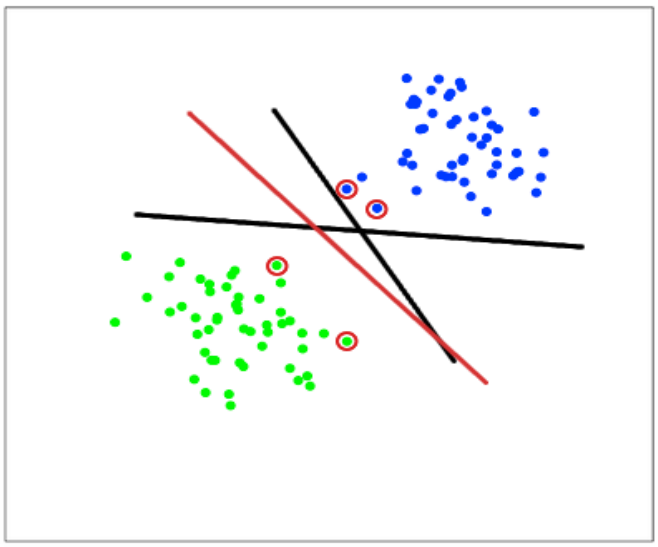


Рис. 1.9 *Визначення оптимальної прямої*

З точки зору точності класифікації найкраще обрати пряму, відстань від якої до кожного класу максимальна. Іншими словами, оберемо ту пряму, яка розділяє класи найкращим чином.

Така пряма, а в загальному випадку – гіперплощина, називається оптимальною.

Вектори, що лежать ближче за всіх до поділяючої гіперплощини, називаються опорними векторами (support vectors).

1.5 Семантичний словник WordNet

WordNet – це велика лексична база даних англійської мови. Іменники, дієслова, прикметники та прислівники згруповані в набори когнітивних синонімів (synsets), кожен з яких виражає чітке поняття. Синсети взаємопов'язані за допомогою концептуально-семантичних та лексичних відносин (див. Рис. 1.10). Результуюча мережа значущою мірою пов'язаних слів і понять може бути навігація за допомогою браузера. WordNet також є вільно доступним для завантаження. Структура WordNet робить його корисним інструментом для обчислювальної лінгвістики та обробки природної мови.

WordNet поверхнево нагадує тезаурус, оскільки він об'єднує слова на основі їх значень. Однак існують деякі важливі відмінності. По-перше, WordNet пов'язує не лише слова-форми-рядки букв, але й специфічні почуття слів. В результаті, слова, знайдені в безпосередній близькості один від одного в мережі, семантично збігаються. По-друге, WordNet називає семантичні відносини між словами, тоді як групування слів у тезаурусі не випливає ні з явним шаблоном, окрім значення схожості.

Головне відношення між словами в WordNet – це синонімія. Синоніми – слова, що позначають одне поняття і є взаємозамінними у багатьох контекстах – згруповані в неупорядковані множини (синхронізації). Кожен з 117 000 синсетів WordNet пов'язаний з іншими синхронізаціями за допомогою невеликої кількос-

ті "концептуальних зв'язків". Крім того, синсет містить коротке визначення ("блиск") і, в більшості випадків, один або декілька коротких речень, що ілюструють використання членів синусоїди. Форми слів з кількома різними значеннями представлені в якомога більшій кількості різних посилань. Таким чином, кожна форма-значення пара в WordNet є унікальною.

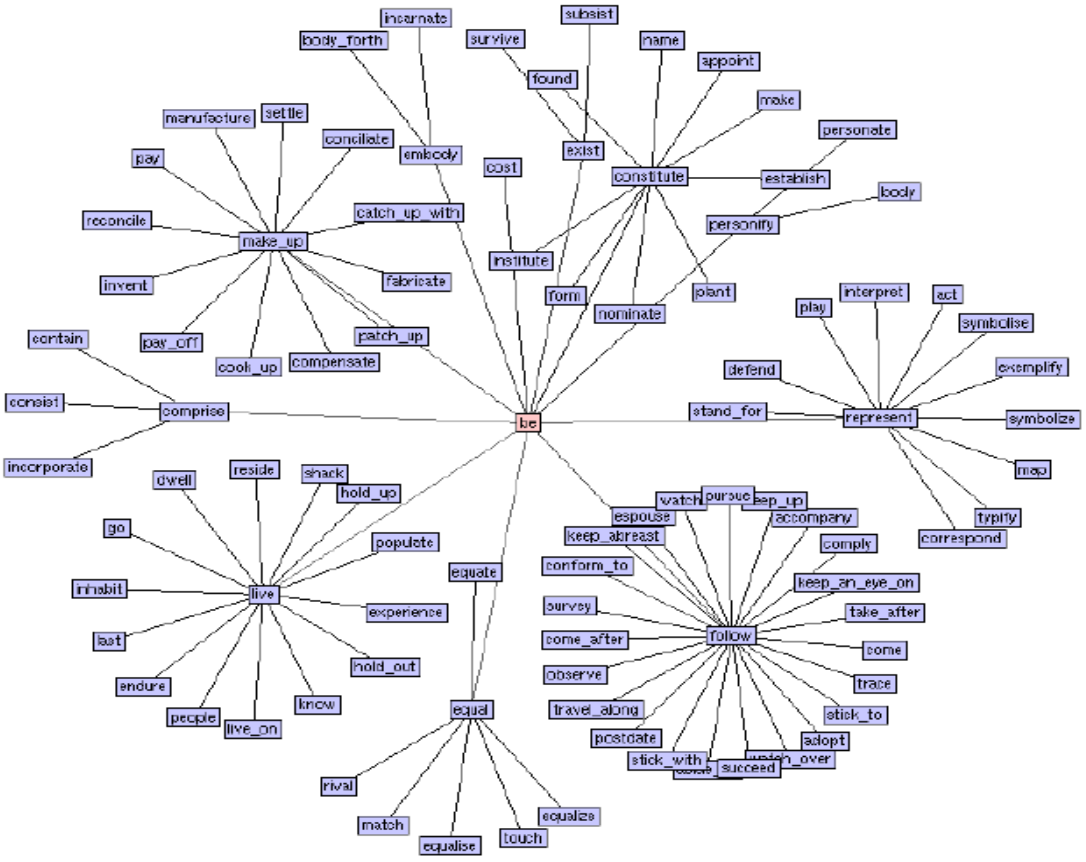


Рис. *1.10 Приклад структури бази даних WordNet*

Найчастіше закодоване відношення між синсетамами – це суперадорідне відношення (також називається гіперонімією, гіпонімією або співвідношенням ISA). Він поєднує більш загальні синтаксичні схеми, такі як {furniture, piece_of_furniture}, до більш конкретних, таких як {bed} та {bunkbed}. Таким чином, WordNet заявляє, що меблі категорії включають в себе ліжко, що в свою чергу включає в себе двоспальне ліжко. Навпаки, такі поняття, як "ліжко" та

"двоспальне ліжко", складають категорію меблів. Всі існуючі ієрархії в кінцевому рахунку піднімаються до кореневого вузла {entity}. Відношення гіпонімій є транзитивним: якщо крісло є свого роду стільцем, а якщо стілець – це свого роду меблі, то крісло – це свого роду меблі. WordNet виділяє серед типів (загальних імен) та екземплярів (конкретних осіб, країн та географічних об'єктів). Таким чином, крісло є типом меблів, Барак Обама є прикладом президента. Випадки завжди є листовими (термінальними) вузлами в їх ієрархіях.

Меронімія, часткове ціле відношення тримається між синсетами, такими як {chair} та {back, backrest}, {seat} та {leg}. Частини успадковуються від їхніх непевностей: якщо стілець має ноги, то крісло також має ноги. Частини не успадковуються "вгору", оскільки вони можуть бути характерними лише для конкретних видів речей, а не для класу в цілому: стільці та види стільців мають ноги, але не всі види меблів мають ноги.

Синсети дієслів також розташовані в ієрархіях. Дієслова в бік листів дерев (тропоніми) виражає все більш конкретні манери, що характеризують подію, як в {спілкуватися} – {говорити} – {шепіт}. Специфічний виражений спосіб залежить від семантичного поля. Об'єм (як у прикладі вище) – це лише одне вимірювання, за яким дієслова може бути розроблена. Інші – це швидкість (рух-біг) або інтенсивність емоцій (люблять-любові-ідолізувати). Дієслова, що описує події, що обов'язково та однонаправлено тягнуть один до одного, пов'язані: {buy} –{pay}, {succeed} – {try}, {show} – {see} та ін.

Прикметники організовані з точки зору антонімії. Пара "прямих" антонімів, таких як сухий і сухий, відображає сильний семантичний договір їх членів. Кожна з цих полярних прикметників, в свою чергу, пов'язана з рядом "семантично подібних": сухий пов'язаний з пересохнутими, засушеними, вичерпаними та кістковими, і мокрими до сипких, заболочених та ін. Семантично подібними прикметниками є "непрямі антоніми". Реляційні прикметники ("pertainyms") вказують на імена, які вони походять від (кримінально-злочинність).

У WordNet є лише декілька прислівників (навряд чи, в основному, дійсно і т.д. Оскільки більшість англійських прислівників прямо походять від прикметників через морфологічну афіксацію (на диво, дивно та інше).

Більшість відносин WordNet пов'язують слова з тієї ж частини мови (POS). Таким чином, WordNet дійсно складається з чотирьох підмереж, по одному для іменників, дієслів, прикметників та прислівників, з кількома крос-POS-покажчиками. Крос-POS відносини включають в себе "морфосемантичні" ланки, що тримаються серед семантично подібних слів, що поділяють стовбур з тим же значенням: спостерігати (дієслово), спостережуване (прикметник) спостереження, обсерваторію (іменники). У багатьох пар з іменник-дієслів визначено семантичну роль іменника відносно дієслова: {sleeper, sleeping_car} – LOCATION для {sleep} і {paintworker} є агентом {paint}, while {painting , picture} – його РЕЗУЛЬТАТ.

Деякі статистичні дані щодо структури WordNet наведені у Табл. 1.2:

Таблиця *1.2*

Статистичні дані WordNet

Statistics	*Nouns*	*Verbs*	*Adjectives*
Synsets (Word-senses)	*8100*	*424*	*291*
Wordforms	*9813*	*633*	*485*
Average of Word-forms per synset	*1.21*	*1.49*	*1.67*
Glossed synsets	*7235*	*332*	*219*
Hyponymy relations	*8357*	*393*	*0*
Meronymy rela-	*2781*	*0*	*0*

Statistics	*Nouns*	*Verbs*	*Adjectives*
tions			
Sub_event relations	*0*	*53*	*0*
ILI relations	*8089*	*729*	*260*
ILI synonymy relations	*5281*	*250*	*206*

1.6 Висновок з розділу

В цілому, нечіткий пошук – досить розповсюджена та важлива у наш час область, яка може бути застосована в розпізнаванні рукописного вводу, в біоінформатиці, у моніторингу лісопожежної ситуації, у сучасних пошукових системах та ін. Використання нечіткого пошуку може значно скоротити час та полегшити знаходження даних.

Проаналізовані існуючі системи мають різні вимоги до оточення та дещо різні функціональні можливості, але усі вони не мають підтримки нечіткого пошуку, саме тому є доцільним створення власного програмного засобу для нечіткого пошуку орфографічних помилок у тексті.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ОБРАНИХ МЕТОДІВ ВИРІШЕННЯ ПРОБЛЕМИ НЕЧІТКОГО ПОШУКУ

Важливим етапом є визначення методів, алгоритмів та інструментів розробки, які можуть вирішити поставлену задачу, їх порівняння та вибір найефективніших, найгнучкіших та найзручніших для використання.

Після дослідження предметної області було визначено, що для реалізації програмного застосунку буде використано:

- метод опорних векторів (Support Vector Machine, SVM);
- метрика відстані Левенштейна;
- лексична база WordNet.

2.1 Метод опорних векторів (Support Vector Machine, SVM)

Однією з найбільш популярних методологій машинного навчання по прецедентах є метод опорних векторів, відомий в англомовній літературі як support vector machine (SVM) [7]. Це машинний алгоритм, котрий навчається на прикладах та використовується для класифікації об’єктів. Наприклад, SVM може розрізнити аварійний режим роботи електромеханічної системи та класифікувати його за наявності попередніх досліджень, можливих за технологічними вимогами режимів роботи. Цей тип методів статистичного оцінювання функцій (або новий вид учнів машин) був запропонований в середині 1990-х рр. [8]. Такий підхід розкриває значні можливості для побудування адаптивних систем автоматичного керування.

Переваги використання методу:

- немає необхідності попередньо розуміти поведінку даних;
- для аналізу даних і вилучення патернів методу досить спостережень за даними і зв'язками всередині них;

- метод опорних векторів працює за принципом "чорного ящика", отримуючи входи і генеруючи виходи, які можуть виявитися дуже корисними в знаходженні патернів в дуже складних і неочевидних даних;
- дуже добре справляється з помилками і шумами в даних;
- будує модель, відповідну основному патерну, на відміну від моделі, яка б враховувала дані всієї навчальної вибірки.

В основі SVM лежить деяка математична сутність [9] – алгоритм максимізації деякої математичної функції відносно наявного набору даних. Для розуміння того, як працює SVM, потрібно мати уявлення про чотири ключові поняття:

- відділяюча гіперплощина (the separating hyperplane);
- гіперплощина максимальної межі (the maximum-margin hyperplane);
- м’яка межа (the soft margin);
- функція ядра (the kernel function).

Відділяюча гіперплощина є математичною сутністю, що відділяє між собою класи об’єктів з однаковими ознаками [10]. Наприклад, на Рис. 2.1 у тривимірному просторі площина відділяє кульки світлого кольору від темних кульок.

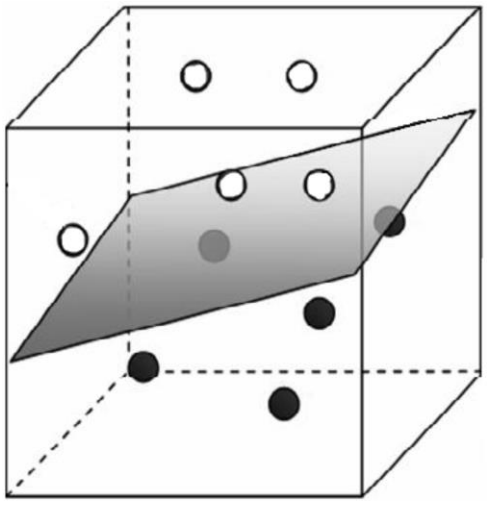


Рис. *2.1 Приклад відділяючої площини*

Можна екстраполювати цю процедуру математично до вимірів, значно вищих за третій. Загальний термін для лінії, котра відділяє елементи різних класів, – багатовимірна гіперплощина.

Спосіб, яким можна провести відділяючу гіперплощину за методом SVM, не є унікальним. Завжди існує багато різних можливостей розташування гіперплощини (Рис. 2.2).

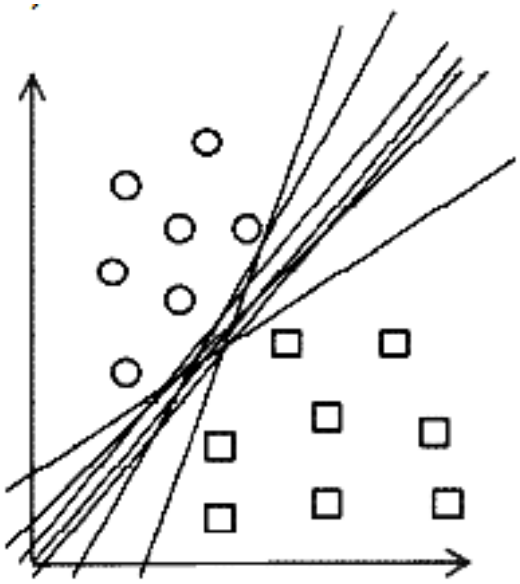


Рис. *2.2 Можливі варіанти розташування гіперплощини у двовимірному просторі*

SVM відрізняється від інших гіперплощинних методів класифікації тим, що він дозволяє обирати оптимальне розташування гіперплощини. Гіперплощина обирається таким чином, щоб бути розташованою на максимальній відстані від елементів кожного з класів, тобто посередині деякої зони, що відділяє між собою ці елементи (на Рис. 2.3 граничні елементи заретушовані). В цьому полягає сутність другого ключового поняття – гіперплощина максимальної межі.

Об’єкти, що класифікуються, не завжди можуть бути розділені гіперплощиною.

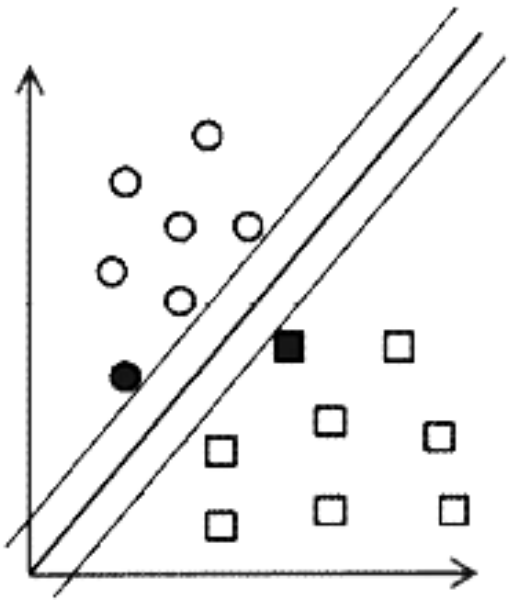


Рис. *2.3 Розташування гіперплощини максимальної межі*

У реальних системах будуть наявними похибки в даних [11], внаслідок яких гіперплощина не виконає розподіл абсолютно точно (див. Рис. 2.4).

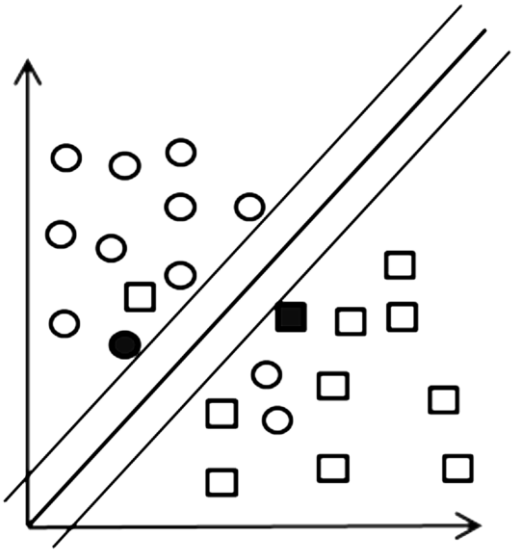


Рис. *2.4 Можливі похибки під час класифікації об'єктів*

Тому для роботи методу SVM вводять допустиму похибку класифікації, що називається м'якою межею.

Звісно, що метод SVM не повинен враховувати забагато похибок класифікації об’єктів, тому потрібно вводити додатковий параметр, котрий встановлює скільки невірно класифікованих об’єктів можуть перетинати гіперплощину максимальної межі і як далеко вони можуть розташовуватись відносно неї. Таким чином, вводиться так звана м’яка межа похибки навколо гіперплощини.

Об’єкти, що класифікуються, можуть бути поділені лінійно лише в окремих випадках [12]. Здебільшого вони не є такими, що допускають лінійне розподілення. Для вирішення проблеми лінійного розподілення використовують функції ядра, що проектують дані з низьковимірного простору у багатовимірний. При вірному виборі функції ядра об’єкти можуть бути розділені лінійно гіперплощиною у багатовимірному просторі. Таким чином, функції ядра виконують роль спрямовуючого простору.

Графічний приклад переходу від задачі, що не має лінійного розподілення об’єктів, до такої, яка дозволяє побудування гіперплощини максимальної межі, наведено на Рис. 2.5.

Розглянемо головні математичні залежності [13], на базі яких працює SVM, та поставимо типову задачу класифікації. Кожен з об’єктів класифікації розглядається як вектор у n-вимірному просторі.

Кожна координата вектора – це деяка ознака, і вона тим більша, чим більше ця ознака відображена у даного об’єкта. Чим менше ця координата, тим менше ознака відповідає об’єкту.

Під час навчання SVM користуються учбовими колекціями, тобто множиною векторів *(x1, x2, ... xn)*, які належать до гіперповерхні R^d та чисел *(y1, y2, ... yn)*, значення яких належать до множини {-1,1}. Причому число y_i дорівнює 1, якщо відповідний йому вектор x_i належить до категорії, що розглядається, та –1 – у протилежному випадку. Нехай є деяка гіперплощина, що розділяє позитивні та негативні приклади. Тоді точки x, що лежатимуть на цій площині, бу-

дуть задовольняти умові: $w \times x + b = 0$, де w – нормальний до гіперплощини вектор.

Перпендикуляр, що визначає відстань від гіперплощини до початку координат, визначиться як $|b|/\|w\|$, де вираз $\|w\|$ називають Евклідовою нормою або довжиною вектора w.

Позначимо найкоротшу відстань від відділяючої гіперплощини до найближчого "позитивного" прикладу d_+ , а до найближчого "негативного" – d_- . Тоді межа навколо гіперплощини матиме ширину $d_+ + d_-$. У тому випадку, коли задача є лінійно розділимою, SVM шукає відділяючу гіперплощину з максимальною межею (таким чином, щоб відстань між елементами, які належать до класу та тими елементами, що не належать до нього, була найбільшою).

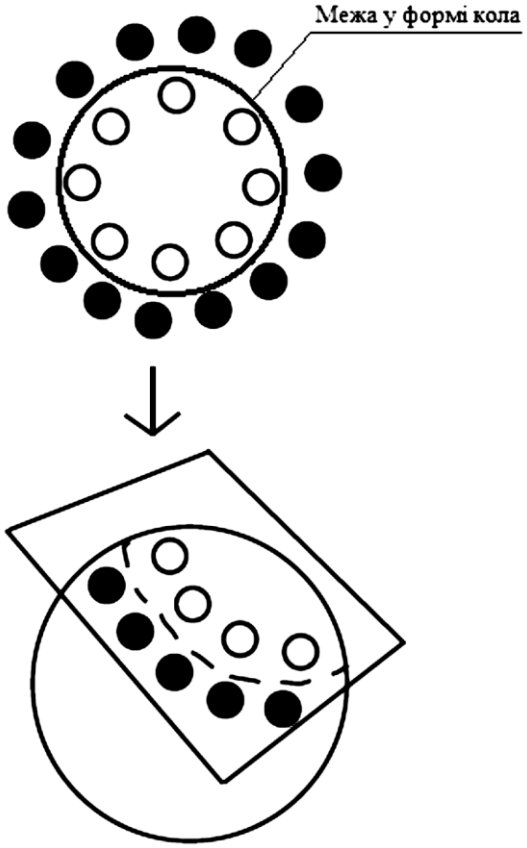


Рис. 2.5 *Приклад спрямлення простору*

Тоді розташування векторів у лінійно розділимій задачі можна описати наступною системою нерівностей [14]:

$$\begin{cases} x_i w + b \geq +1 \text{ для } y_i = +1 \\ x_i w + b \leq -1 \text{ для } y_i = -1 \end{cases}$$

Дану систему нерівностей можна спростити до вигляду:

$$y_i(x_i w + b) - 1 \geq 0 \quad \forall_i$$

Межі навколо гіперплощини максимальної межі також являють собою гіперплощини, а саме – гіперплощину H_1 , що описується рівнянням $x_i w + b = 1$, та гіперплощину H_2 з рівнянням $x_i w + b = -1$. Ці гіперплощини будуть паралельними одна до одної та матимуть один нормальний вектор w . Відстань від площини H_1 до початку координат буде |1-b|/||w|| , а від гіперплощини H_2 до початку координат – становитиме |-1-b|/||w||. Відповідно, значення $d_+ = d_- = 1/||w||$, а ширина максимальної межі визначиться як $d_+ + d_- = 2/||w||$. Таким чином, при умові того, що задача є лінійно розділимою, виконується пошук пари гіперплощин, котрі розташовані одна від одної на максимальній відстані, для чого мінімізують ||w|| . Гіперплощина максимальної межі буде розташовуватись посередині на рівній відстані від H_1 і H_2 паралельно до них.

Лінійне розділення точок за зазначеним вище принципом для двовимірної задачі наведено на рисунку 2.6. Точки, що лежать у гіперплощинах H_1 та H_2 , та виключення яких з прикладів призвело б до зміни положення гіперплощини максимальної межі, називають опорними векторами (на Рис. 2.6 вони обведені колами).

Відомо, що для знаходження мінімуму функції потрібно дослідити її похідну [15]. У випадку мінімізації ||w|| задача ускладнюється тим, що задані лінійні обмеження, які слід враховувати при мінімізації. Множина точок, які за-

довольняють обмеженням, в n-вимірному просторі являє собою багатогранник: простір ділиться гіперплощинами або напівгіперплощинами у залежності від того, стоїть в обмеженнях знак рівності чи нерівності. Пошук мінімуму відбувається в цьому обмеженому просторі.

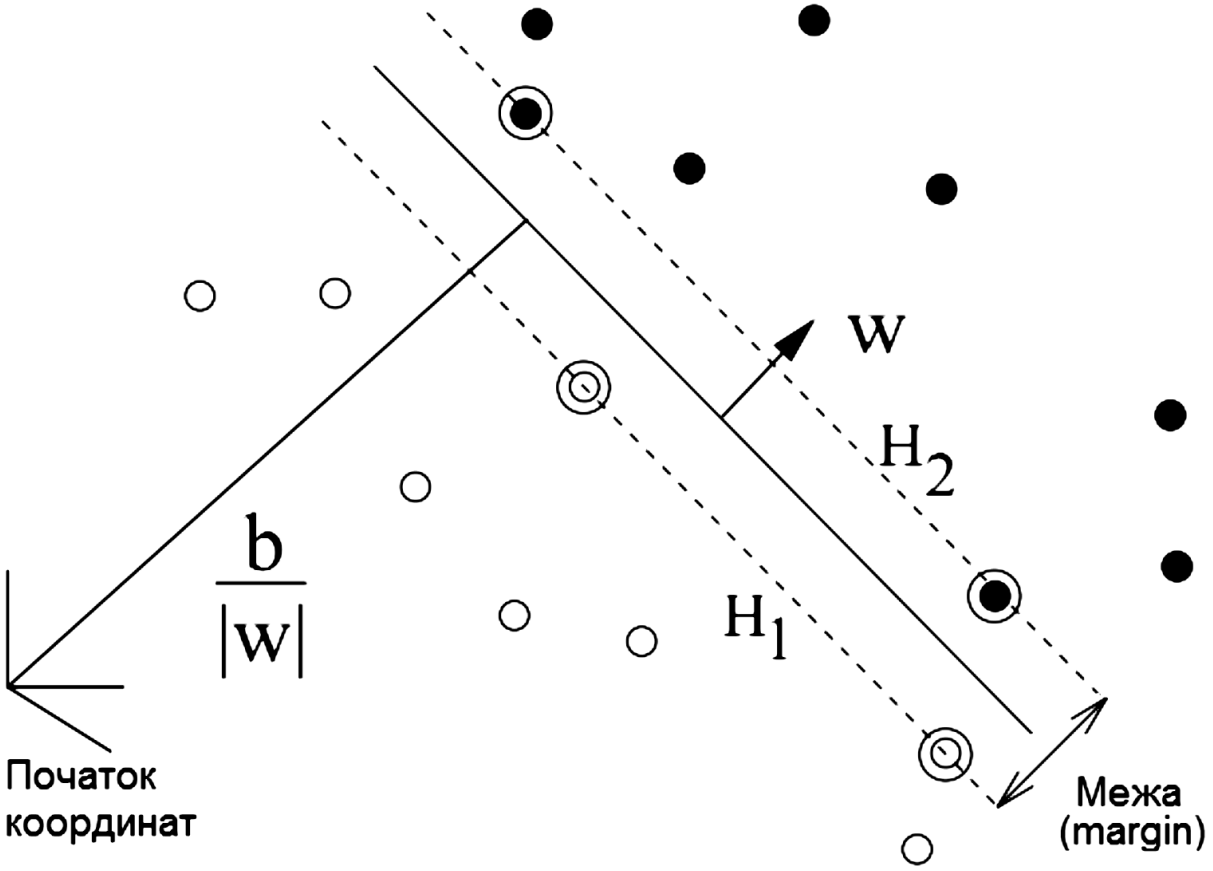


Рис. 2.6 *Лінійне розділення точок*

Розв’язати задачу мінімізації в обмеженому просторі можна за допомогою метода Лагранжа [16]. Лагранж звів задачу пошуку умовного мінімуму до задачі мінімізації без обмежень, щоб потім скористатись стандартним методом пошуку мінімуму функції. Для використання метода Лагранжа потрібно змінити функцію, що підлягає мінімізації.

Для формування нової функції потрібно ввести множники Лагранжа α_i, і = 1, 2, ... , n – один для кожного елементу нерівності(2). Лагранжіан має наступний вигляд:

$$L_p = \frac{1}{2} ||w||^2 - \sum_{i=1}^{n} \alpha_i y_i (x_i w + b) + \sum_{i=1}^{n} \alpha_i \qquad (3)$$

Тепер потрібно мінімізувати лагранжіан (3) відносно w, b, одночасно вимагаючи, щоб похідні відносно всіх α_i дорівнювали нулю для $\alpha_i \geq 0$. Ці вимоги призводять до необхідності виконання наступних умов:

$$w = \sum_i \alpha_i y_i x_i \qquad (4)$$

$$\sum_i \alpha_i y_i = 0 \qquad (5)$$

Тоді, з урахуванням формул (4) та (5) вираз (3) можна представити у вигляді:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \qquad (6)$$

У формулі (3) та (6) для позначення лагранжіана використовуються різні індекси: у формулі (3) – індекс “P” (primal – головний), а у формулі (6) – індекс “D” (dual – подвійний). Ці два вирази дещо не співпадають: вони формуються на основі тієї ж оптимізаційної функції, проте з різними обмеженнями, та розв’язок знаходиться шляхом мінімізації L_p або максимізації L_D . Якщо задача оптимізації формулюється таким чином, що b = 0 , тобто гіперплощини проходять через початок координат, тоді обмеження (5) не з’являється [17].

У наведених рівняннях точки, для яких $\alpha_i > 0$ називаються опорними векторами, вони лежать на одній з гіперплощин H_1 або H_1 . Для всіх інших точок

$\alpha_i = 0$. Для віртуальних машинних методів класифікації, що навчаються за таким принципом, опорні вектори є критичними точками навчальної множини. Якщо інші точки будуть змінюватись або пересуватись у просторі, не зачіпаючи опорні вектори, то результат навчання (відділяюча гіперплощина) не буде змінюватись.

Під час оптимізації у просторі з обмеженнями застосовують умови Каруша-Куна-Таккера (Karush-Kuhn-Tucker) або ККТ [18], які є узагальненням метода множників Лагранжа. В теорії оптимізації умови ККТ – це необхідні умови розв'язку задач нелінійного програмування. Щоб розв'язок був оптимальним, повинні виконуватись умови (7) – (11) для лагранжіана LP .

$$\frac{\partial}{\partial w_v} L_p = w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \dots d \qquad (7)$$

$$\frac{\partial}{\partial b} L_p = -\sum_i \alpha_i y_i = 0 \qquad (8)$$

$$y_i(x_i w + b) - 1 \geq 0 \quad i = 1 \dots n \qquad (9)$$

$$\alpha_i \geq 0 \quad \forall_i \qquad (10)$$

$$\alpha_i(y_i(x_i w + b) - 1) = 0 \quad \forall_i \qquad (11)$$

Таким чином, оптимізація за ККТ у випадку лінійно розділимої задачі буде являти собою один з варіантів категоризації за методом SVM.

Під час виконання категоризації реальних даних можуть виникати похибки, котрі призводитимуть до неможливості лінійного розподілення. Для усунення впливу похибок вводяться так звані коефіцієнти вартості у нерівності обмежень.

Лінійне розділення точок за наявності похибки наведено на Рис. 2.7.

Все, що розглядалося вище, стосується лінійно розділимих задач. На практиці такий вид категоризації зустрічається, проте він є лише винятком, здебільшого категоризаційні задачі не допускають лінійного розподілення об’єктів.

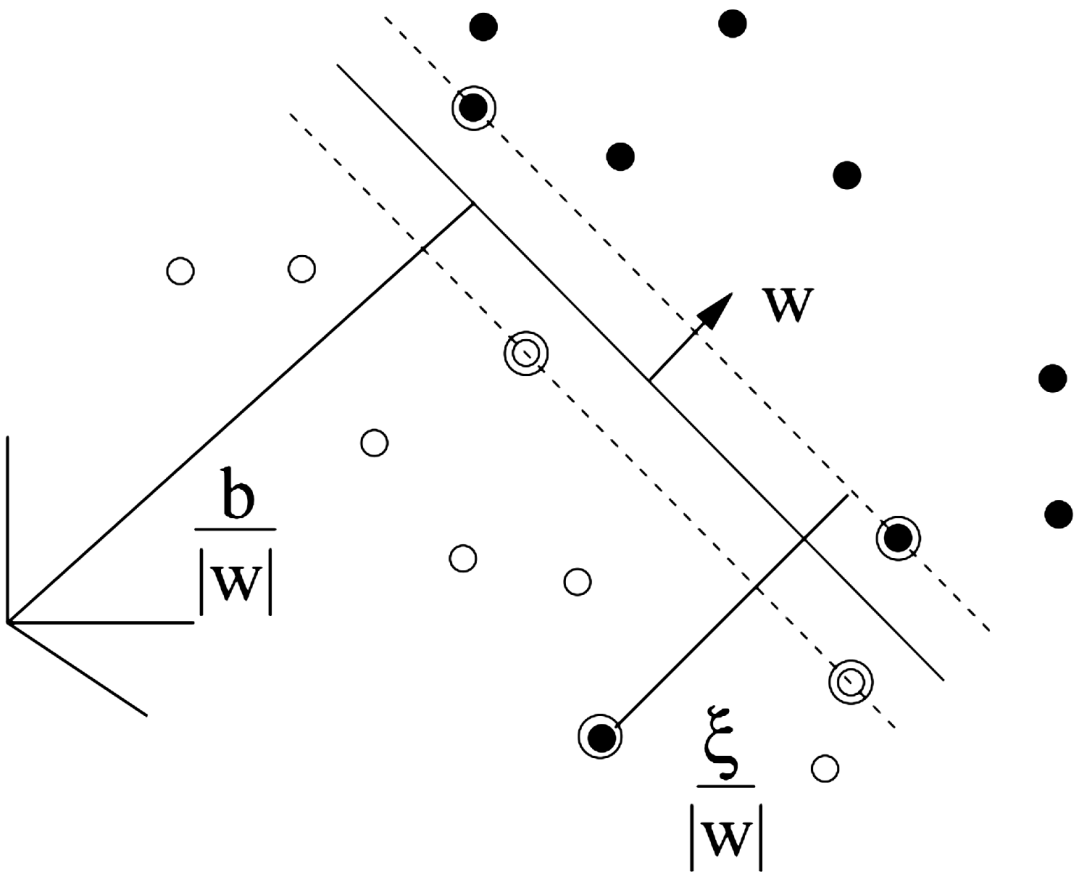


Рис. 2.7 *Лінійне розділення точок за наявності похибки*

Для того, щоб задача знову стала лінійно розділимою і можна було застосовувати розглянуті вище лагранжіани, треба розмістити дані у просторі більш високого порядку, де можливе лінійне розділення об’єктів гіперплощиною [19]. Цю операцію називають спрямленням простору або мапінгом даних (Рис. 2.8).

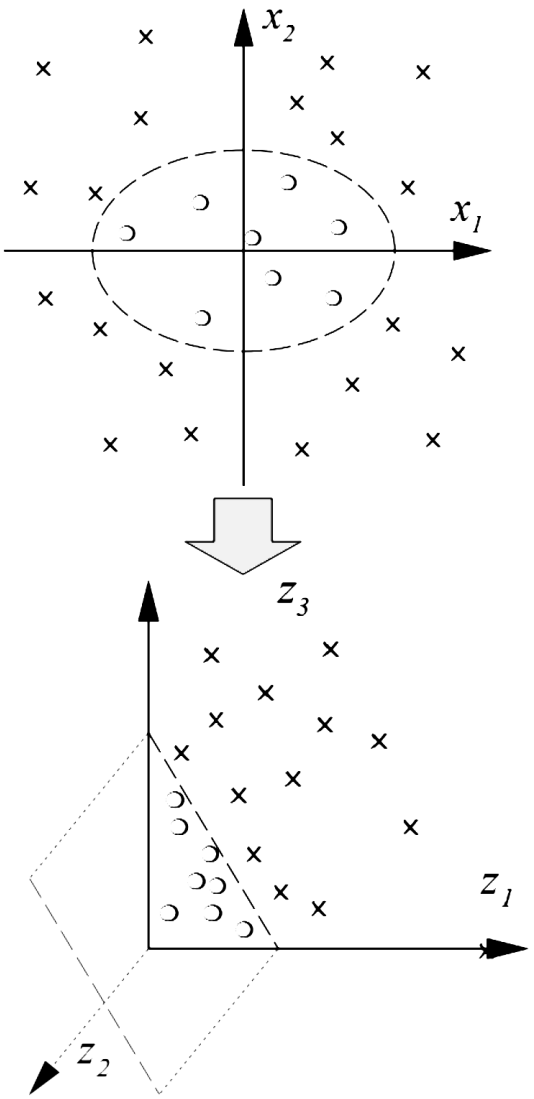


Рис. 2.8 *Спрямлення простору або мапінг даних*

Мапінг даних позначається Ф і виконує відображення даних із вихідного гіперпростору R^d до евклідового простору ℵ. [20]

$$\Phi = R^d \rightarrow \aleph \qquad (12)$$

Отже, для того, щоб задача стала лінійно розділимою, до кожної з точок треба застосувати перетворення (12) і надалі оперувати не з самими точками, а з відповідними їм Ф(x).

Логіка такого розподілення добре ілюструється на Рис. 2.9.

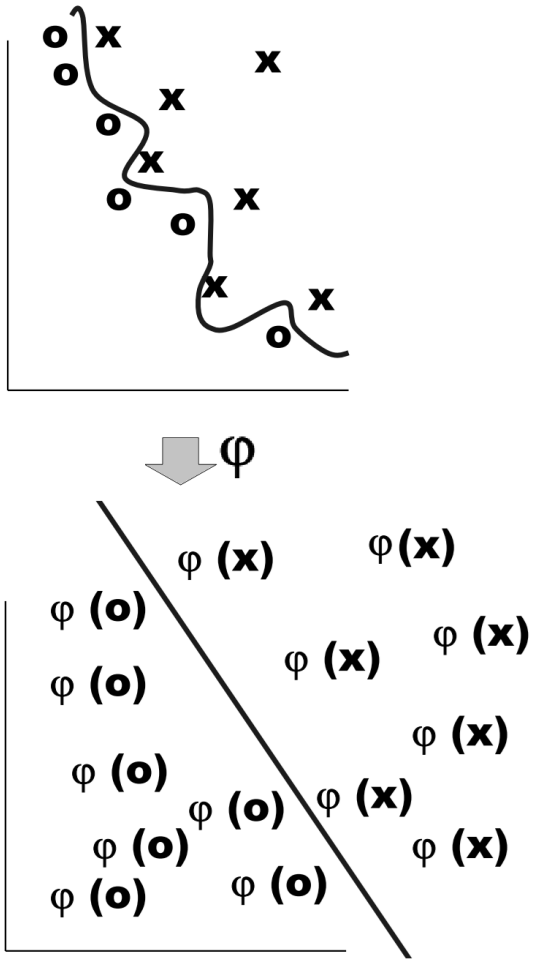


Рис. 2.9 *Ілюстрація до застосування Ф(х) до вихідних даних*

У розглянутих лагранжіанах [21] часто застосовуються скалярні добутки точок. Для спрощення оптимізаційної задачі було розроблено спосіб, що називається kernel trick, в основі якого лежить перехід від скалярних добутків до так званих функцій ядра. При цьому кожен скалярний добуток замінюється нелінійною функцією ядра (скалярним добутком у просторі з більшою розмірністю). Функція ядра може бути представлена наступним чином [7 – 11]:

$$K(x_i, x_j) = \Phi(x_i)\Phi(x_j)$$

Використання функції ядра взагалі дає можливість відійти від необхідності користуватись *Ф(x)*. Користувач методу може навіть не знати, яку *Ф(x)* використовує та чи інша функція ядра.

На етапі навчання SVM використовується для обчислення f (x) за вихідними даними навчання з використанням формули (13) [22]:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)\Phi(x) + b = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \quad (13)$$

Найпростіше реалізувати нелінійний метод SVM на базі LD , або так званого lagrangian trick із застосуванням kernel trick. При цьому максимізується рівняння (14) відносно рівностей (4) та (5). Функція ядра при цьому може бути представлена рівнянням (15) [23]:

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j \quad (14)$$

$$K(x_i, x_j) = (x_i x_j + 1)^p \quad (15),$$

де p – деякий параметр, що підлягає налаштуванню користувачем.

В результаті проведеного дослідження було виявлено, що метод опорних векторів зводить навчання класифікатора до оптимізаційної задачі, яка розв’язується евристичними алгоритмами. Для побудування нелінійних класифікаторів використовується розширення простору та функції ядер. Метод SVM у тестах перемагає інші методи за швидкістю та точністю категоризації. При різному підході до вибору ядер метод може емулювати роботу інших математичних методів. SVM може працювати як нейронна мережа, проте таке використання обмежує його призначення, оскільки метод значно перевершує їх за можливостями.

SVM може бути успішно застосований для керування складними електромеханічними системами, він може забезпечити адаптивність алгоритмів керування, виконувати функції спостерігача, ідентифікатора невідомих параметрів, деякої еталонної моделі, за його допомогою можна керувати складними нелінійними об’єктами, а також об’єктами зі стохастичними параметрами.

У реалізації програмного застосунку метод опорних векторів представлений у вигляді груп, у які об’єднані слова. Кожна група являє собою число, що позначає довжину слова. Використання таких груп дозволяє зменшити кількість лем, серед яких виконується пошук слова у базі даних, і відповідно пришвидшити час виконання програми.

2.2 Метрика відстані Левенштейна

Відстань Левенштейна (так само відома як редакційна відстань або дистанція редагування) – обчислюється як мінімальна кількість операцій вставки, видалення і заміни, необхідних для перетворення однієї послідовності в іншу, наприклад: ('ABC', 'ABC') = 0, ('ABC', 'ABCDEF') = 3, ('ABC', 'BCDE') = 3, ('BCDE', 'ABCDEF') = 2. Алгоритм дозволяє суб'єктивно оцінити, наскільки рядки несхожі один на одного [24].

Для розрахунку відстані Левенштейна найчастіше використовують простий алгоритм, в якому використовується матриця розміром (n + 1) * (m + 1), де n і m – довжини порівнюваних рядків. Окрім цього вартість операцій вилучення, заміни та вставки вважається однаковою. Для конструювання матриці використовують таке рекурсивне рівняння:

$$D(i,j) = \begin{cases} 0; \quad i = 0, j = 0 \\ i;\ i > 0, j = 0 \\ j;\ i = 0, j > 0 \\ \\ \min(\\ D(i, j-1) + 1 \\ D(i-1, j) + 1 \\ D(i-1, j-1) + m(S_1[i], S_2[j]) \\ ); \\ i > 0, j > 0, \end{cases}$$

Для того щоб отримати дистанцію Левенштейна між рядками s і t (довжиною m і n відповідно, індексація починається з нуля) та редакційний припис (які саме правки потрібно вносити), розраховується матриця відстаней D (розмірність m + 1 * n + 1), кожен елемент D [i, j] містить дистанцію між першими i-символами рядка s та першими j-символами рядка t. Наприклад, матриця дистанцій Левенштейна для рядків s = 'ABC' і t = 'ABF' :

$$\begin{bmatrix} & & A & B & F \\ & 0 & 1 & 2 & 3 \\ A & 1 & 0 & 1 & 2 \\ B & 2 & 1 & 0 & 1 \\ C & 3 & 2 & 1 & 1 \end{bmatrix}$$

Стовпці відповідають підрядкам рядка t, а рядки матриці – підрядкам s. Рядок і стовпець з нульовим індексом відповідають порожнім підрядкам s і t. Кожен елемент цієї матриці містить відстань між підрядками, що відповідають його індексам. Наприклад, D [3,2] = 1 – це відстань між ABC і AB (лише одна правка – видалити C). Таким чином, D [3,3] = 1 це і є шукана дистанція між ABC і ABF (заміна C на F). Крім дистанції ця матриця містить у собі інформа-

цію про ті виправлення, які необхідно внести в рядок s щоб отримати рядок t – редакційний припис.

Побудова матриці дистанцій схожа на прокладання маршруту через лабіринт: починаємо з лівого верхнього кута матриці-карти і повинні потрапити в правий нижній. Частину матриці можна заповнити без обчислень: стовпець і рядок з нульовими індексами заповнюються числами по порядку, починаючи з нуля. Це просто пояснити тим, що для того, щоб з порожнього рядка отримати якийсь рядок T (довжиною k), потрібно рівно k вставок - по одній на кожен символ. Аналогічно і в зворотний бік: для того, щоб з рядка S, довжиною l, отримати порожній рядок, потрібно рівно l вилучень. Таким чином, числа в нульовому рядку і стовпці не залежать від вмісту порівнюваних рядків.

Правила заповнення матриці дистанцій Левенштейна [25]:

● щодо клітинки D [i, j], клітинка, розташована зліва зверху від неї, D [i-1, j-1], являє собою «пройдену дистанцію» – редагування необхідні для того, щоб перші i-1 символів рядка s перетворити в перші j-1 символів t, тобто підготувати s до операції заміни s [i-1] на t [j-1] (індексація в рядку починається з 0, тобто s [i-1] це i-ий за рахунком символ). D [i, j-1] – клітинка зліва – це правки, витрачені на підготовку рядка s до вставки символу t [j-1]. Аналогічно, клітинка зверху – D [i-1, j] – це дистанція, пройдена на перетворення підрядка s [0..i-2] в t [0..j-2] (що дозволяє видалити i-ий за рахунком символ , тим самим привівши s [0..i-1] до t [0..j-1]);

● якщо s [i-1] == t [j-1], то значення можна скопіювати з клітинки зліва-зверху. Відстань Левенштейна для підрядків s [0..i-1] і t [0..j-1] в даному випадку буде дорівнювати відстані s [0..i-2] і t [0..j-2], так як на i-ий за рахунком символ рядка s не потрібні правки - він і так вже має потрібне значення. D [i, j] = D [i-1, j-1];

● якщо ж символи s [i-1] != T [j-1], то можливі три варіанти, з яких вибирається один з мінімальною дистанцією:

1. Операція заміни: символ s [i-1] потрібно замінити на t [j-1]. В такому випадку дистанція дорівнює дистанції зліва-зверху + 1. Це схоже на рух також по діагоналі, але зі штрафом. D [i, j] = D [i-1, j-1] + 1;

2. Операція вставки: символ t [j-1] потрібно вставити після на s [i-1]. Дистанція дорівнює пройденої на утворення t [0..j-2] з s [0..i-1] + одна операція вставки. Це рух праворуч по карті. D [i, j] = D [i, j-1] + 1;

3. Операція видалення: символ s [i-1] потрібно видалити. Дистанція дорівнює правок, витраченим на формування t [0..j-1] з s [0..i-2] + 1 правка, яка відображає видалення. Це можна порівняти з рухом вниз по карті. D [i, j] = D [i -1, j] + 1;

● якщо представляти правки у вигляді рухів по карті, тоді побудова матриці – це пошук оптимального шляху;

● заповнювати матрицю можна як завгодно, але слід врахувати, що для того щоб зробити вибір мінімальної дистанції, перебуваючи в певній клітинці, потрібні три сусідні клітинки: зліва, зверху і зліва-зверху. Тому починати потрібно з лівого верхнього кута матриці.

Редакційний припис – це послідовність дій, необхідних для отримання з першого рядка другий найкоротшим чином. Зазвичай дії позначаються так: D (англ. Delete) – видалити, I (англ. Insert) – вставити, R (replace) – замінити, M (match) – збіг.

Для рядків ABC і ABF редакційний припис буде виглядати так:

M M R

A B C

A B F

Щоб побудувати редакційний припис, найпростіше запам'ятовувати вибрані операції під час побудови матриці. Також можна проаналізувати матрицю, рухаючись від правого нижнього кута до лівого верхнього з мінімальними вага-

ми, запам’ятовуючи ходи: хід вліво – I (вставка), вгору – D (видалення), вліво-вгору - R (заміна), якщо символи розрізняються, інакше – M (збіг).

2.3 Семантична мережа WordNet

У якості словника за основу було обрано семантичну мережу WordNet з наступних причин:

- база даних містить 155,287 слів, що організовані у 117,659 синсетів;
- мережа включає в себе такі лексичні категорії як іменники, прикметники, прислівники та дієслова;
- вона є у вільному доступі.

Для реалізації нечіткого пошуку немає необхідності використовувати весь словник, адже WordNet – це семантична мережа, яка включає в себе не лише слова, але і синсети. Саме тому для роботи програмного застосунку було взято за основу значення з таблиць «*words*» і «*samples*» та об’єднано у одну таблицю для зручності використання.

РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ НЕЧІТКОГО ПОШУКУ В СЛОВНИКАХ

3.1 Постановка задачі

Створюваний програмний застосунок повинен аналізувати та обробляти текстові дані на наявність помилкових слів, а також пропонувати користувачу можливі варіанти виправлення, якщо вони існують. Крім того, необхідно реалізувати можливість додавання нових слів до словника для забезпечення розширюваності.

Також програма повинна надавати загальну статистику оброблених слів, що включає в себе відсоткове відношення кількості знайдених, виправлених та нерозпізнаних слів.

Діаграма варіантів використання наведена на Рис. 3.1.

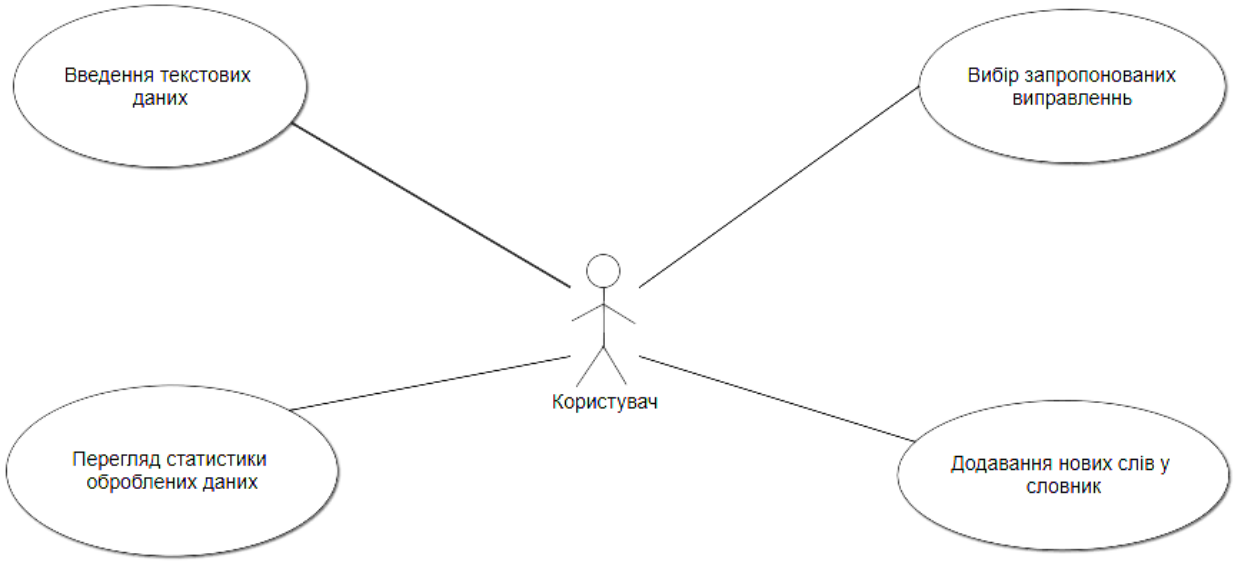


Рис. *3.1 Діаграма варіантів використання*

3.2 Вимоги до оточення

3.2.1 Апаратне забезпечення

Дана програмна система реалізована на апаратному забезпеченні (комп’ютері) з наступними параметрами:

- 2-х ядерний процесор типу Intel Core з тактовою частотою 2.6 ГГц;
- оперативна пам'ять об'ємом 4ГБ;
- мінімальна роздільна здатність екрана 1366x768px.

3.2.2 Вимоги до програмного забезпечення

Для роботи програми необхідно, щоб на клієнтських машинах були встановлені веб-браузери. Бажано, останні версії браузерів Mozilla Firefox, Opera або Google Chrome.

На серверній стороні повинні бути встановлені та налаштовані:

- веб-сервер Apache;
- мова програмування PHP з підтримкою версії 5.6 або вище;
- система керування базами даних MySQL.

3.2.3 Вимоги до користувача

Для ефективної роботи з системою користувач, окрім базових навичок роботи з комп’ютером, повинен також орієнтуватися у предметній області та володіти англійською мовою на достатньому рівні.

3.3 Засоби реалізації

3.3.1 Клієнтська частина

- мови HTML5, CSS3, JavaScript;
- бібліотеки jQuery та Bootstrap;

- середовище розробки PhpStorm;
- браузери Mozilla Firefox та Google Chrome;
- VCS Git та GitHub.

3.3.2 Серверна частина

- мови PHP 5.6, JavaScript, SQL;
- веб-сервер Apache;
- база даних MySQL;
- середовище розробки PhpStorm;
- VCS Git та GitHub.

3.4 Архітектура програмної системи

Програмний застосунок для нечіткого пошуку у словнику реалізується у вигляді веб-застосунку та має клієнт-серверну архітектуру, яка включає в себе:

- веб-сервер;
- веб-клієнт;
- сервер баз даних.

Схема розгортання системи наведена на Рис. 3.2.

Веб-сервер відповідальний за виконання всіх обчислень. Це єдина централізована точка доступу до будь-якої інформації в системі. Тут реалізується алгоритм обчислення метрики відстані Левенштейна, метод опорних векторів, парсинг вхідних даних, аналіз і обробка тексту та інше.

Клієнтська частина представляє собою «зовнішню» частину веб-застосунку та слугує у якості інтерфейса користувача. Вона містить всі графічні елементи управління програмним застосунком, до яких має доступ користувач. Клієнтська частина спілкується з веб-сервером за допомогою HTTP запитів.

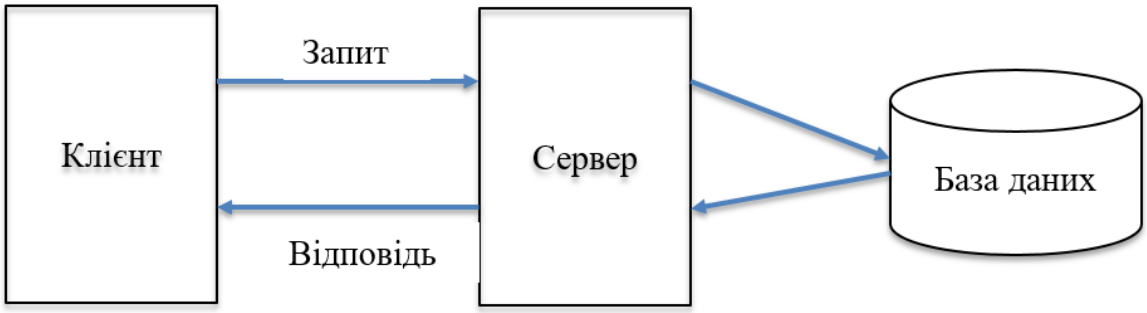


Рис. *3.2 Схема розгортання модулів системи*

Сервер баз даних використовується для збереження даних. Тут зберігається словник лем, які об’єднані у групи, приклади їх використання, а також частотний словник слів.

3.4.1 База даних

Структура бази даних наведена на Рис. 3.3.

Рис. *3.3 Структура бази даних*

Таблиця *words* зберігає інформацію про слово, а саме лему, групу, до якої воно відноситься та приклад використання. На даний момент таблиця нараховує 303312 записів.

Таблиця *word-frequency* є частотним словником та зберігає інформацію про те, як часто зустрічається слово в англомовних творах. На даний момент таблиця нараховує 6715 записів.

3.4.2 Діаграма класів

На Рис.3.4 наведена діаграма класів. Програмна система представлена у вигляді двох класів: StringManager та WordnetManager.

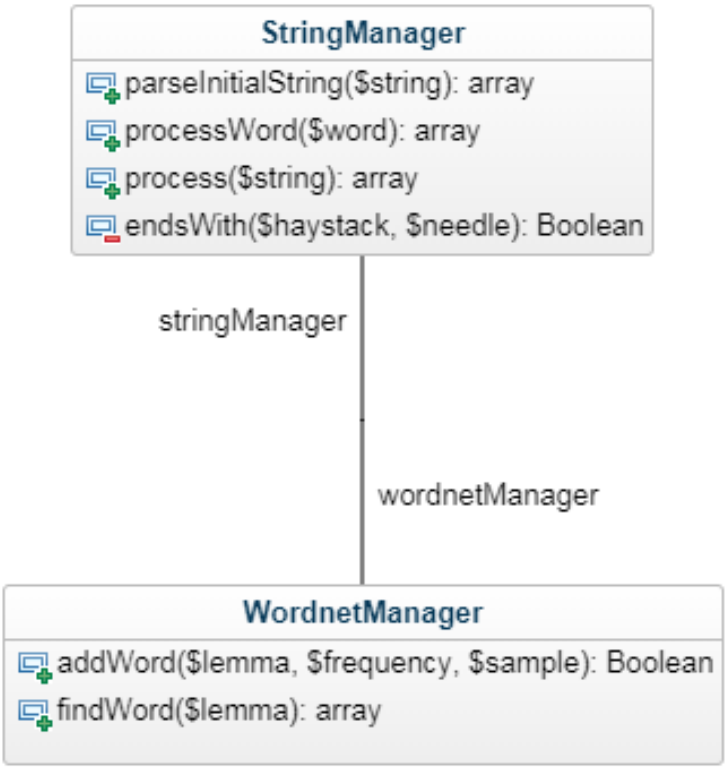


Рис. *3.4 Діаграма класів*

Клас StringManager відповідає за розбиття, обробку та аналіз вхідних текстових даних. Основним методом є *process($string*), якому на вхід подається текстовий рядок, а на виході повертаються вже оброблені дані у вигляді масиву. У ньому викликаються метод *parseInitialString($string),* який розбиває рядок на слова та повертає масив, метод *processWord($word),* який виконує обробку кожного слова, зберігає його початковий стан та розділові знаки, та відбувається пошук слів у базі даних.

Клас WordnetManager призначений для роботи з базою даних та має методи *addWord($lemma, $frequency, $sample)* та *findWord($lemma),* які відповідають за додавання та пошук слова відповідно.

3.4.3 Реалізація та модулі системи

Програмна реалізація включає в себе наступні модулі:

- модуль обробки вхідних даних;
- модуль аналізу та пошуку даних;
- модуль розширення словника;
- модуль обробки вихідних даних.

Модуль обробки вхідних даних реалізує розбиття вхідного тексту на окремі слова, зберігаючи початковий стан слова та розділові знаки, що зустрічаються у тексті. Даний модуль представлений у вигляді методів *parseInitialString()* та *processWord()* класу *StringManager,* код якого наведений у лістингу 1.

Метод *parseInitialString()* розділяє вхідний рядок на слова з розділовими знаками та повертає ці слова у вигляді масиву. У методі *processWord()* зберігається початковий стан слова, далі відбувається перебір усіх символів та видалення розділових знаків.

Лістинг *1*

Реалізація методів *parseInitialString() та processWord() у класі StringManager*

```
class StringManager {

...

public static function parseInitialString($string)
{
    $string = trim($string);
```

```
    $array = preg_split("/([^ \n\r]+[ \n\r]+)/", $string,
-1, PREG_SPLIT_DELIM_CAPTURE | PREG_SPLIT_NO_EMPTY);
    $array = array_map('trim', $array);

    return $array;
}

public static function processWord($word)
{
    $lettersArray = str_split($word);
    $newWord = '';
    for ($k = 0; $k < count($lettersArray); $k++)
    {
        if ($lettersArray[$k] == '.'
|| $lettersArray[$k] == ',' || $lettersArray[$k] == '—' ||
$lettersArray[$k] == ';' || $lettersArray[$k] == '!' || $let-
tersArray[$k] == '?' || $lettersArray[$k] == ':' || $letter-
sArray[$k] == '(' || $lettersArray[$k] == ')' || $lettersAr-
ray[$k] == '{' || $lettersArray[$k] == '}' || $lettersAr-
ray[$k] == '"' || $lettersArray[$k] == '\'')
{
            if ($lettersArray[$k] == '\'')
            {
                if (($k + 1 < count($lettersArray))
&& ($k + 2 < count($lettersArray))
&& ($lettersArray[$k + 1] == 'l'
&& $lettersArray[$k + 2] == 'l'
|| $lettersArray[$k + 1] == 'v'
&& $lettersArray[$k + 2] == 'e'))
            {
                    $newWord .= $lettersArray[$k];
```

```
                    continue;
                }
                else if
(($k + 1 < count($lettersArray))
&& $lettersArray[$k + 1] == 't')
                {
                    $newWord .= $lettersArray[$k];
                    continue;
                }
                else continue;
            }
else continue;
}
$newWord .= $lettersArray[$k];
}
return $newWord;
    }
    ...
    }
```

Модуль аналізу даних визначає до якої із груп відноситься кожне слово. Класифікація здійснюється за кількістю символів у слові. Такі групи слів і є опорними векторами, які допомагають зменшити кількість елементів при переборі та значно пришвидшити пошук.

Далі обчислюється значення метрики Левенштейна для кожного вхідного слова та для усіх слів зі словника, у яких значення групи співпадає з значенням групи заданого слова, або є на одиницю більшим чи на одиницю меншим. Врахування значень не однієї групи, а одразу трьох зумовлене тим, що метрика відстані має три операції: вставка, видалення та заміна, саме тому під час видален-

ня або вставки довжина слова може зменшуватись або збільшуватись відповідно, а слово при цьому буде перенесене з однієї групи в іншу.

Після отримання значень метрики обираються лише ті слова, які мають не більше однієї відмінності (тобто допущена лише одна помилка у слові). Далі програма перевіряє чи є якість з цих слів у частотному словнику. Якщо співпадіння знайдене, то користувачу буде запропоноване слово, яке зустрічається найчастіше, а якщо жодного зі слів у частотному словнику не знайдено – користувачу буде запропоноване випадкове слово або декілька слів.

Даний модуль використовує вбудовану у PHP функцію levenshtein(), яка повертає відстань між двома рядками у вигляді числа та метод *process()* класу *StringManager*, код якого наведений у лістингу 2 .

Лістинг *2*
Реалізація методу process() у класі StringManager

```
class StringManager
{
    ...
    public static function process($string)
    {
    $recognizedWords = [];
    $unknownWords = [];
    $fixedWords = [];
    $similarWordsResult = [];

    ...

    $array = self::parseInitialString($string);
    if ($array) {
    for ($i = 0; $i < count($array); $i++) {
```

```
    $initialWord = $array[$i];                  //save initial
word
    $newWord = self::processWord($array[$i]);

    $array[$i] = strtolower($newWord);

    if (in_array($array[$i], self::$partsOfSpeech)) {
    $array[$i] = '&' . $array[$i] . '&';
    $similarWordsResult[$i]['word'] = $array[$i];
    } else {
    if (strpos($array[$i], '\'ll') !== false
    || strpos($array[$i], '\'ve') !== false
    || strpos($array[$i], 'n\'t') !== false
    ) {
    $array[$i] = '&' . $array[$i] . '&';
    $similarWordsResult[$i]['word'] = $array[$i];
    }
    $stmt = $db->query("SELECT lemma FROM words WHERE lemma
LIKE '" . $array[$i] .
    "' OR sample LIKE '% " . $array[$i] . " %' LIMIT 1");
    if (!$stmt)
    continue;
    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

    if ($result && ($result[0]['lemma'])) {
    $array[$i] = '&' . $array[$i] . '&';
    $similarWordsResult[$i]['word'] = $array[$i];
    } else {

    $similarWords = [];
    foreach (self::$partsOfSpeech as $word) {
```

```
    $distance = levenshtein($array[$i], $word, 1, 1, 1);
    if ($distance == 1) {
    array_push($similarWords, $word);}
    }

    $length = strlen($array[$i]);
    $in = " $length, $length + 1, $length - 1 ";
    $sql = "SELECT lemma FROM words WHERE wordGroup IN ($in)
";
    $globalStmt = $db->query($sql);
    if (!$globalStmt)
    continue;

    $dbWords = $globalStmt->fetchAll(PDO::FETCH_ASSOC);

    foreach ($dbWords as $dbWord) {
    if ($dbWord['lemma'] != null) {
    $distance = levenshtein($array[$i], $dbWord['lemma'], 1,
1, 1);
    if ($distance == 1) {
    array_push($similarWords, $dbWord['lemma']);
    }
    }
    }
    //find which word has max frequency and suggest it
    //if there is no such word => suggest any

    $maxFrequency = -1;
    $suggestedWord = '';
    foreach ($similarWords as $word) {
```

```
    $sql = "SELECT frequency FROM `word-frequency` WHERE
lemma LIKE '$word' LIMIT 1";
    $globalStmt = $db->query($sql);
    if (!$globalStmt)
    continue;
    $dbWord = $globalStmt->fetchAll(PDO::FETCH_ASSOC);
    if ($dbWord) {
    if ($dbWord[0]['frequency'] > $maxFrequency) {
    $maxFrequency = $dbWord[0]['frequency'];
    $suggestedWord = $word;
    }}}
    if (count($similarWords) > 1) {
    $j = 0;
    $a = [];
    foreach ($similarWords as $similarWord) {
    if ($j == 2) break;
    if ($similarWord == $suggestedWord) continue;

    $a[] = $similarWord;
    $j++;
    }
    $similarWordsResult[$i]['similar-words'] = $a;
    }}}

    $newWordReplaced = '';
    $wordStatus = '';

    if ($array[$i][0] == '&' && $array[$i][strlen($array[$i])
- 1] == '&') {
    $wordStatus = 'correct';
    $resultString = trim($array[$i], "&");
```

```
    $recognizedWords[] = $resultString;
    } else if ($array[$i][0] == '/' &&
$array[$i][strlen($array[$i]) - 1] == '/') {
    $wordStatus = 'error';
    $resultString = trim($array[$i], "/");
    $fixedWords[] = $resultString;
    } else if ($array[$i][0] == '#' &&
$array[$i][strlen($array[$i]) - 1] == '#') {
    $wordStatus = 'unknown';
    $resultString = trim($array[$i], "#");
    $unknownWords[] = $resultString;
    }

    $foundWordArray = str_split($resultString);
    $initialWordArray = str_split($initialWord);
    $index = 0;
    for ($g = 0; $g < count($initialWordArray); $g++) {
    if ($initialWordArray[$g] == '.' || $initialWordArray[$g]
== ',' || $initialWordArray[$g] == '—' ||
    $initialWordArray[$g] == ';' || $initialWordArray[$g] ==
'!' || $initialWordArray[$g] == '?' ||
    $initialWordArray[$g] == ':' || $initialWordArray[$g] ==
'(' || $initialWordArray[$g] == ')' ||
    $initialWordArray[$g] == '{' || $initialWordArray[$g] ==
'}' || $initialWordArray[$g] == '"' ||
    $initialWordArray[$g] == '\''
    ) {
    $newWordReplaced .= $initialWordArray[$g];
    } else {
    if ($index < count($foundWordArray)) {
```

```
    if (ctype_upper($initialWord[$g]) &&
strtoupper($foundWordArray[$index]) == $initialWord[$g]) {
    $newWordReplaced .= strtoupper($foundWordArray[$index]);
    $index++;
    } else {
    $newWordReplaced .= $foundWordArray[$index];
    $index++;
    }
    } else {
    $newWordReplaced .= $initialWord[$g];
    }
    }
    }
    if ($wordStatus == 'correct') {
    $array[$i] = '&' . $newWordReplaced . '&';
    } else if ($wordStatus == 'error') {
    $array[$i] = '/' . $newWordReplaced . '/';
    } else if ($wordStatus == 'unknown') {
    $array[$i] = '#' . $newWordReplaced . '#';
    }

    $similarWordsResult[$i]['word'] = $array[$i];
    }

    } else {
    $similarWordsResult = $string;
    }
    $similarWordsResult['statistics'] = array('recognized' =>
$recognizedWords, 'fixed' => $fixedWords, 'unknown' =>
$unknownWords);
    return $similarWordsResult;
```

```
    }
}
```

Модуль розширення словника дозволяє користувачу додавати нові слова у словник, вказувати їх частоту, значення та приклади використання. Ця можливість є корисною у разі необхідності створення тематичного словника. Користувач повинен лише задати словам окремої теми більшу частоту використання, і таким чином це слово буде запропоноване програмою в першу чергу.

Основною задачею *модуля обробки вихідних даних* є відтворення початкового стану текстових даних з урахуванням розділових знаків та морфологічних форм слів, повідомлення користувача про можливі помилки та пропозиції їх виправлення. Даний модуль реалізований за допомогою фонових запитів на мові JavaScript (код наведений у лістингу 3).

Лістинг *3*

Реалізація модуля обробки вихідних даних

```
    $.post("algorithms/levenstain.php", form.serialize(),
function (data) {
    data = JSON.parse(data);
    ...
    if (result['statistics']) {
    showStatistics(result);
    }
    for (var i = 0; i < Object.keys(result).length; i++) {
    if (!result[i]) continue;
    if (result[i]['word'].indexOf('/') !== -1) {
    var word = result[i]['word'].replace(/\//g, '');
    if (result[i]['similar-words']) {
    list = list + '<div class="tooltip-container"><span
class="error-message">' + word + '</span>' +
```

```
    '<span class="tooltip-text">';
    for (var j = 0; j < result[i]['similar-words'].length;
j++) {
    list = list + '<span>' + result[i]['similar-words'][j] +
'</span>'; }
    list = list + '</span></div> ';
    }
    else {
    list = list + '<span class="error-message">' + word +
'</span>' + ' ';  }
    }
    else if (result[i]['word'].indexOf('&') !== -1) {
    word = result[i]['word'].replace(/&/g, '');
    list = list + '<span>' + word + '</span>' + ' '; }
    else {
    word = result[i]['word'].replace(/#/g, '');
    list = list + '<span class="unknown-message">' + word +
'</span>' + ' ';  }
    }
    form.find('#processed-text').html(list); });
```

3.5 Проект інтерфейсу

Інтерфейс програми представлений у вигляді веб-сторінки, яка має 2 основні поля для введення початкових та виведення оброблених текстових даних (див. Рис. 3.5).

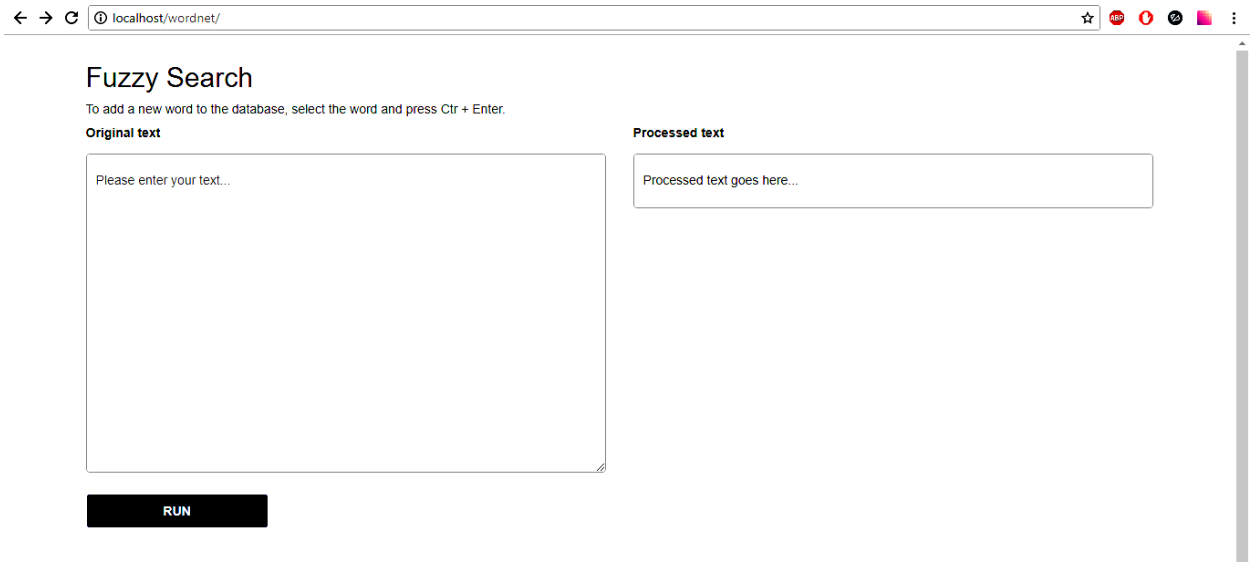


Рис. *3.5 Інтерфейс програми*

Для запуску програми користувач повинен ввести текст, який необхідно перевірити, у поле «Original text» та натиснути кнопку «RUN». Результат з’явиться у полі «Processed text» як показано на рисунку 3.6.

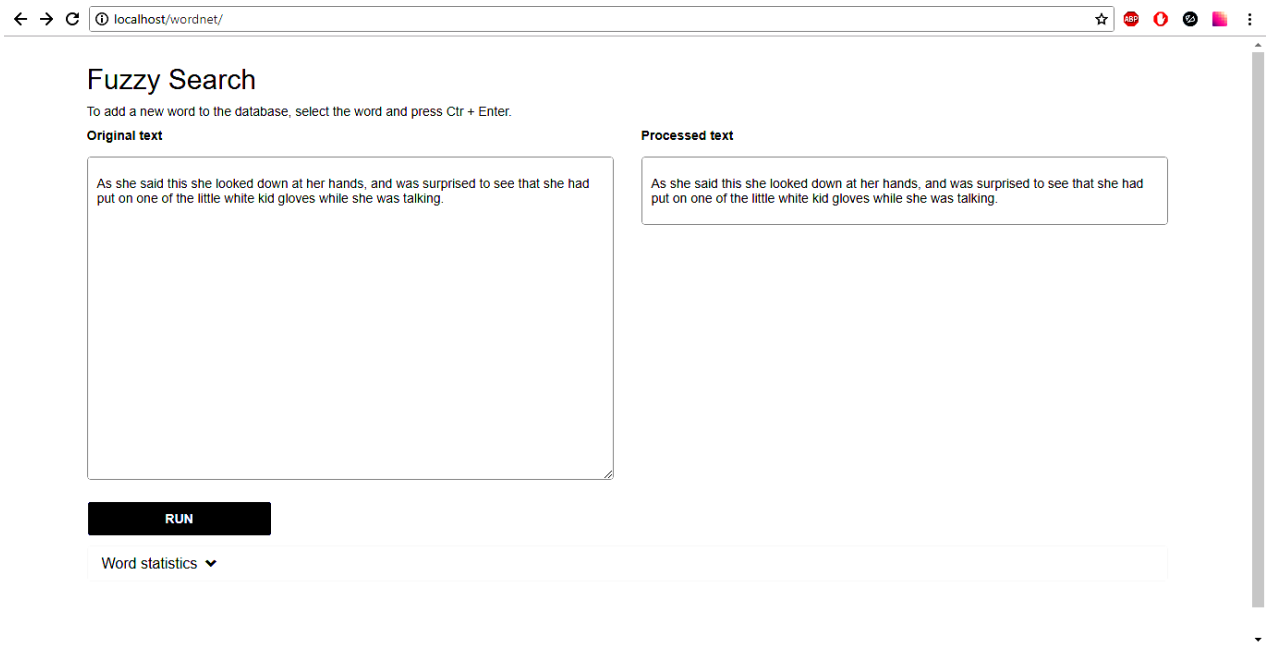


Рис. *3.6 Приклад виведення оброблених текстових даних*

У разі виявлення помилки у будь-якому із слів, блок виведення буде містити усі вхідні текстові дані, а замість помилкового слова користувачу буде запропоновано один або декілька варіантів виправлення. Кожен варіант виправлення виділяється червоним кольором. Усі слова, які будуть знайдені у словнику залишаться такого ж формату та кольору як і оригінальний текст. Якщо слово не було знайдене у словнику, то воно помічається як невідоме та позначається сірим кольором (див. Рис. 3.7).

Під час роботи програма спочатку перевіряє наявність слова у словнику. Якщо його у словнику немає, то далі, за довжиною слова визначається його приналежність до певної групи, а зі словника обираються такі слова, у яких номер групи або дорівнює або більше або менше на одиницю за значення групи шуканого слова. Після отримання набору таких слів, програма обчислює відстань Левенштейна між вхідним словом та кожним словом з отриманого набору, відкидаючи усі ті, у яких відстань не дорівнює одиниці. Після цього кожне слово з отриманого набору перевіряється на наявність у частотному словнику. Ті слова, які є у частотному словнику мають вищий пріоритет та будуть запропо-

новані користувачу у першу чергу. Якщо жодного зі слів результуючого набору немає у частотному словнику, то користувачу будуть запропоновані випадкові схожі слова, кількістю до 3-х варіантів.

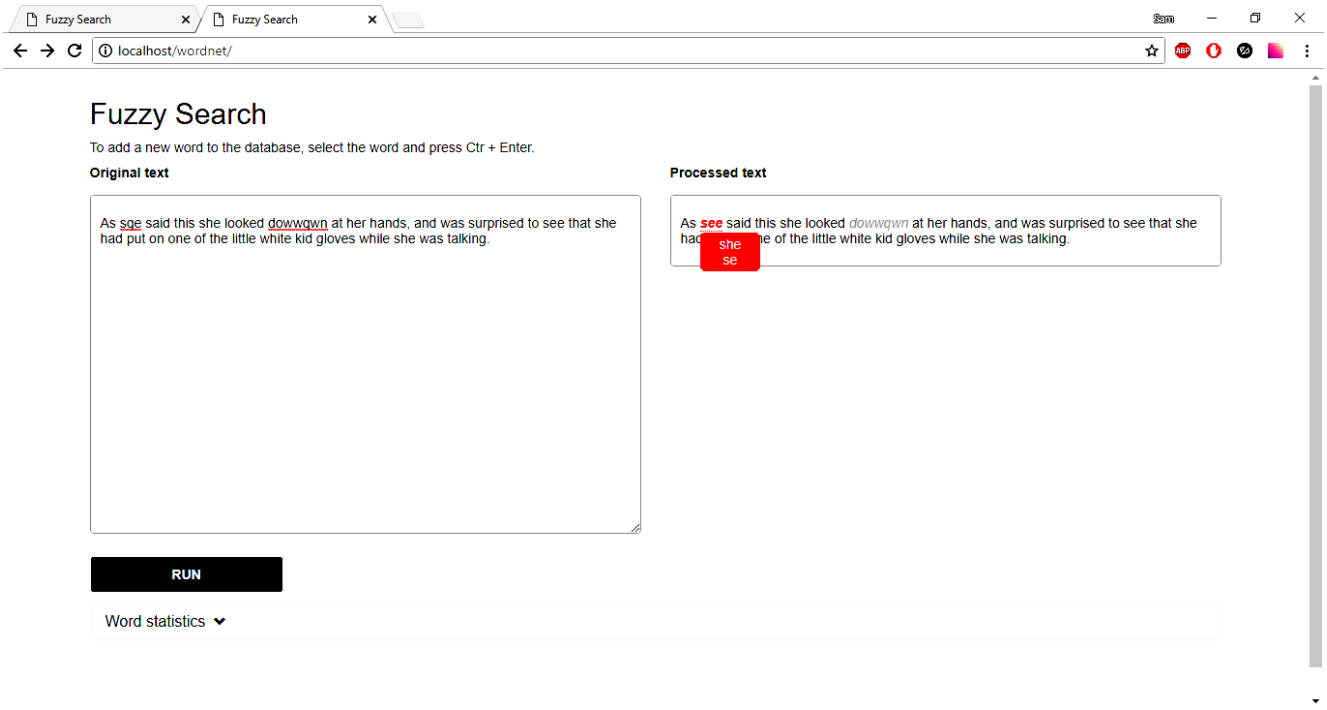


Рис. *3.7 Формат вихідних даних*

Після обробки тексту та отримання вихідних даних користувач може переглянути загальну статистику оброблених слів, натиснувши на поле «Word statistics».

Після натискання з’явиться блок зі статистичними даними. Він містить кількість розпізнаних, невідомих та помилкових слів, а також відсоткове співвідношення усіх цих показників і перелік самих слів для кожної з категорій. У нижній частині цього блока виводиться час виконання програми у мілісекундах (див. Рис. 3.8).

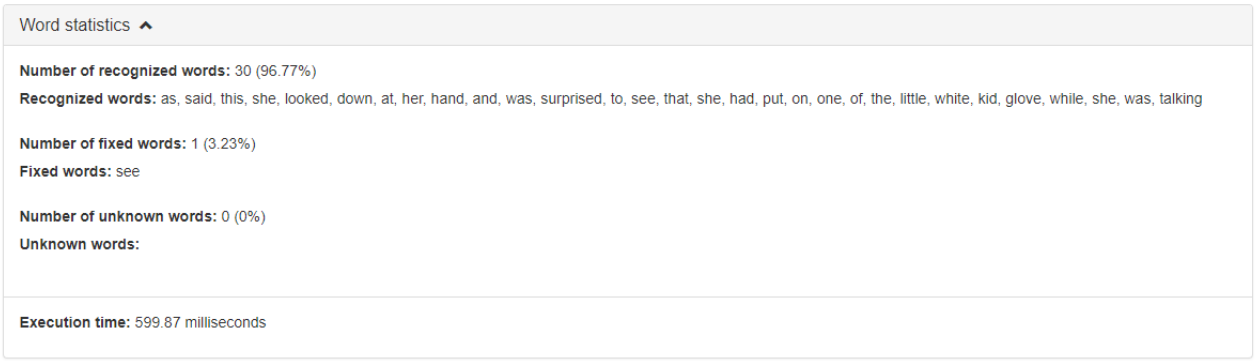


Рис. *3.8 Приклад статистичних даних*

Розширюваність словника означає можливість додавати нові слова, їх частоту використання та приклади. Користувач може додати нове слово виділивши його у тексті та натиснувши комбінацію клавіш «Ctrl» + «Enter». Після цього з’явиться вікно додавання нового слова з відповідними полями (див. Рис. 3.9).

Рис. *3.9 Додавання нового слова у словник*

Після того, як слово буде збережене у словнику, користувачу виведеться повідомлення про те, що слово додано успішно (див. Рис. 3.10).

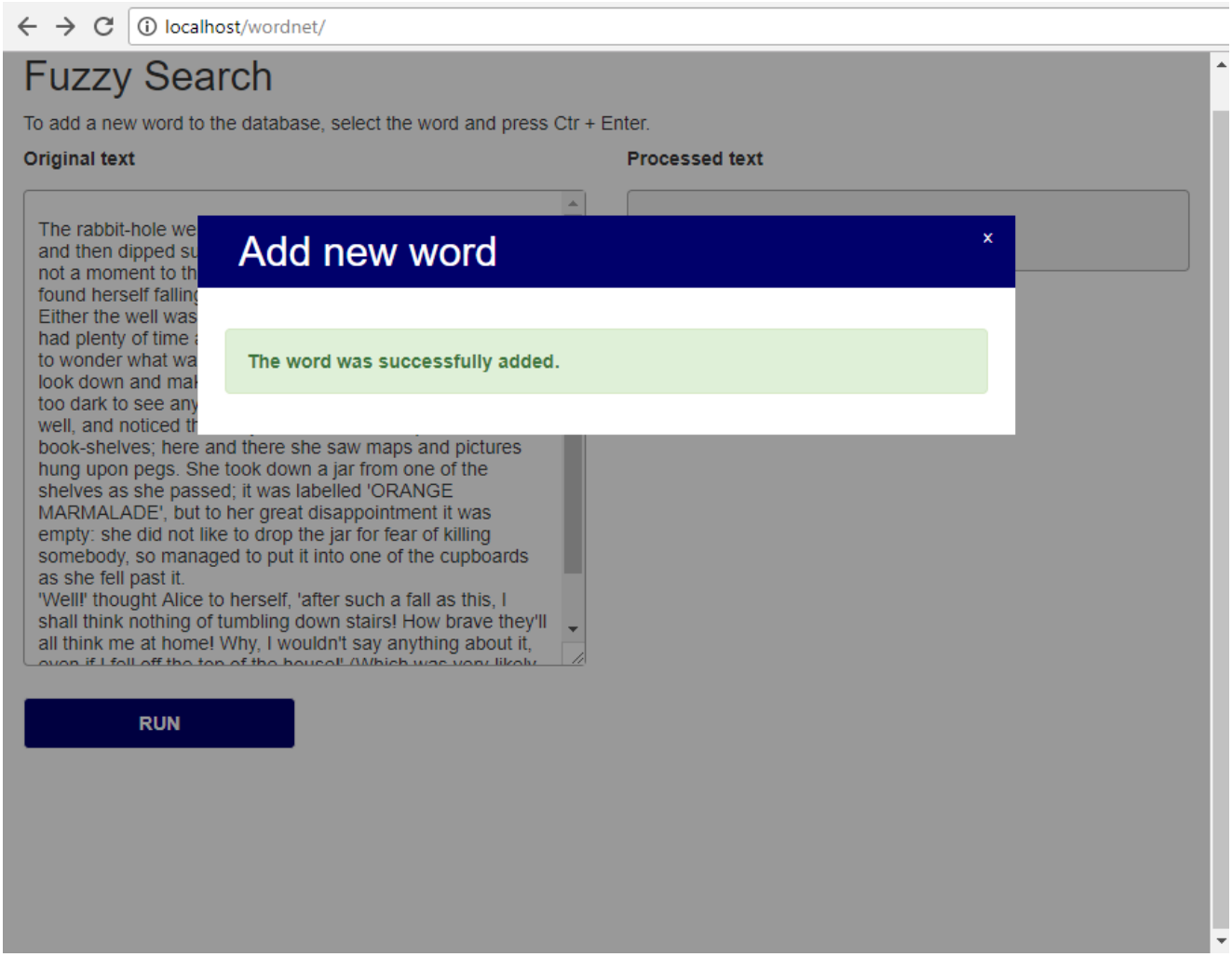


Рис. *3.10 Результат додавання нового слова у словник*

РОЗДІЛ 4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ ПРОБЛЕМИ НЕЧІТКОГО ПОШУКУ В СЛОВНИКАХ

4.1 Дослідження продуктивності та можливостей створюваної програмної системи

Основною метою дипломної роботи було дослідження можливості нечіткого пошуку у словнику, використовуючи метод опорних векторів. У випадку успіху, можна буде стверджувати, що метод опорних векторів є найбільш оптимальним для вирішення задачі нечіткого пошуку у тексті та словнику.

Дослідження проводилось з використанням вхідних текстових даних з різними характеристиками (див. Рис. 4.1).

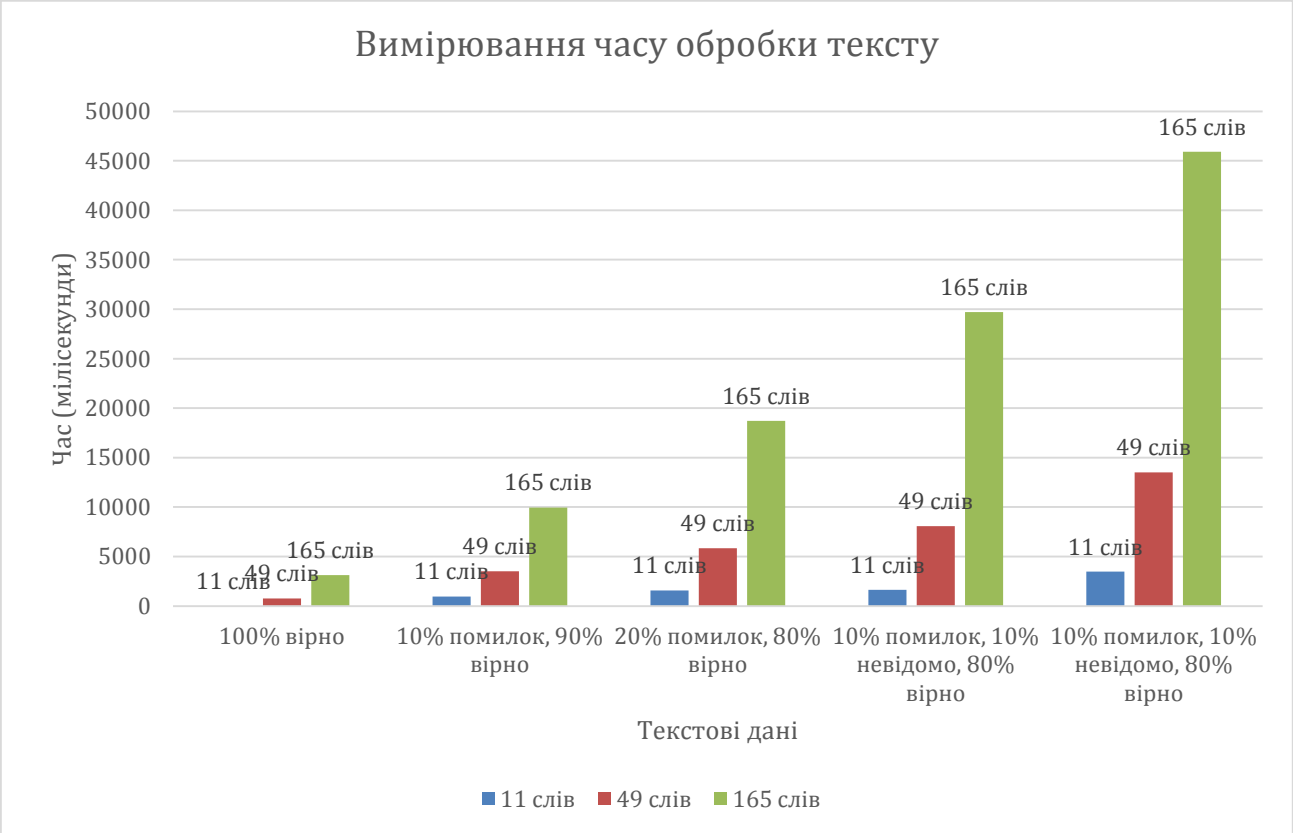


Рис. *4.1 Діаграма виміру часу обробки тексту в залежності від різних характеристик вхідних даних*

Для отримання достовірного результату було проведено декілька вимірювань часу обробки тексту, змінюючи наступні показники:

- загальну кількість слів;
- кількість правильних слів;
- кількість завідомо помилкових слів;
- кількість невідомих слів (або слів, яких немає у словнику).

Діаграма на Рис. 4.1 показує, що швидкість обробки текстових даних залежить від кількості помилкових та невідомих слів. При цьому, якщо присутні лише помилкові слова, наприклад 20% від загальної кількості слів, а невідомих немає, то час виконання значно менший, ніж якщо 10% слів були б помилковими, а інші 10% – невідомими. Це пов’язано з тим, що під час пошуку невідомого слова програмі необхідно перевірити абсолютно усі слова певної групи на відповідність, а під час пошуку помилкового – ймовірніше за все підходяще слово буде знайдене раніше, ніж перебрані усі варіанти.

4.2 Дослідження проблеми словоформ у створеній програмній системі

Під час розробки програмної системи було виявлено проблему розпізнання словоформ в англійській мові. Суть проблеми полягає у тому, що семантична мережа WordNet, яка була взята за основу лексичного словника, зберігає для кожного слова лише лему, яка є умовною основною одиницею слова, та не враховує різноманітні словоформи.

Словоформа – це будь-яка форма слова, яка утворюється за рахунок зміни часу, роду, відмінку, набуття закінчень, подвоєння літер та ін.. Наприклад, слово having є словоформою для слова have, або слово looked є словоформою для слова look.

Кількість словоформ у англійській мові дуже велика, а враховуючи те, що деякі слова є ще й виключеннями, реалізувати обробку усіх словоформ та передбачити усі можливі випадки виключень досить важко.

На даний момент програмна система виконує обробку лише основних словоформ (-ing, -ed, -s, -es та ін.), але у майбутньому вона може бути вдосконалена та розширена за рахунок додавання можливості розпізнання більшої кількості словоформ та винятків шляхом розширення словників.

РОЗДІЛ 5 ОХОРОНА ПРАЦІ ТА ТЕХНОГЕННА БЕЗПЕКА

5.1 Аналіз потенційно небезпечних та шкідливих чинників, що впливають на працівника

В робочому приміщенні на працівника можуть здійснювати шкідливий вплив наступні чинники:

- недостатня освітленість;
- періодично підвищений рівень шуму;
- зависока або занижена температура повітря;
- можливість враження електричним струмом;
- погана ергономіка робочого місця;
- перенавантаження кістково-м’язового апарату та м’язів кистей рук;
- зорове перенапруження;
- розумове перенапруження.

Найбільш впливовими шкідливими чинниками в робочому приміщенні є можливість враження електричним струмом та погана ергономіка робочого місця, що викликає фізичні перенавантаження. Основними джерелами враження електричним струмом є комп’ютерна та інша побутова техніка.

Зважаючи на легку (1а) категорію важкості праці та шкідливу категорію напруженості трудового процесу (1-го ступеня), приміщення повинно відповідати наступним вимогам [26]:

- рівень шуму – до 50 дБ;
- температура повітря – 22-24 ^{o}C в теплу пору року і 23-25 ^{o}C в холодну;
- відносна вологість 60-40%;
- швидкість руху повітря не вище, ніж 0.1 м/с.

Аналіз шкідливих факторів і карта умов праці наведені в таблиці 5.1.

Таблиця *5.1*

Оцінка чинників виробничого і трудового процесу

№	Чинники виробничого середовища і трудового процесу	Нормативне значення	Фактичне значення	III клас: шкідливі і небезпечні умови			Тривалість дії чинників за зміну %
				I ступінь	II ступінь	III ступінь	
1	Шкідливі хімічні речовини, мг/м^3 1 клас небезпеки____ 2 клас небезпеки____ 3-4 класи небезпеки____						
2	Пил, переважно фіброгенної дії, мг/м^3						
3	Вібрація (загальна і локальна), дБ						
4	Шум, дБА	50	40-70				65
5	Інфразвук, дБ						
6	Ультразвук, дБ						
7	Неіонізуючі випромінювання: -радіочастотний діапазон, в/м -діапазон промислової частоти, кв/м -оптичний діапазон (лазерне випромінювання), Вт/м^2						
8	Мікроклімат в приміщенні: - температура повітря, ºС в теплу пору року в холодну пору року - швидкість руху повітря, м/с - відносна вологість повітря %	 22-24 23-25 ≤0.1 40-60	 20-30 15-22 0.05-0.5 50				 100 100 100 100

№	Чинники виробничого середовища і трудового процесу	Нормативне значення	Фактичне значення	III клас: шкідливі і небезпечні умови			Тривалість дії чинників за зміну %
				I ступінь	II ступінь	III ступінь	
9*	Категорія важкості і напруженість праці	Категорія важкості праці – легка (1а) категорія напруженості праці – шкідлива (1 к.)					

Отже, можна зробити висновок про невідповідність умов праці в робочому приміщенні нормативним значенням. Фактичні умови праці знижують концентрацію і призводять до погіршення продуктивності працівників, а також, сприяють розвитку професійних патологій, пов’язаних з надмірними навантаженнями на органи зору та кістково-м’язовий апарат.

5.2 Заходи з поліпшення умов праці

Проблеми з ергономікою викликані невідповідністю оснащення робочого місця нормативам. Перш за все, необхідно додати ще один робочий стіл або замінити існуючий на більший за розмірами. Це дозволить збільшити робочу площу, відстань між комп’ютерами та відстань від очей працівника до монітора. Також необхідно оснастити робочі місця підйомно-поворотними кріслами з підлокітниками для зниження статичного напруження м’язів рук. Для зниження блисткості поверхонь моніторів слід використовувати сонцезахисні плівки для вікон або спеціальні антиблікові фільтри для моніторів.

Робоче місце працівника повинне забезпечувати підтримання оптимальної робочої пози з наступними ергономічними характеристиками:

- ступні ніг повинні бути розташовані на підлозі або підставці;
- стегна повинні бути розташовані в горизонтальній площині;
- передпліччя повинно бути розташоване вертикально;
- лікті повинні бути зігнуті під кутом 70-90 градусів відносно горизонтальної площини;
- нахил голови повинен бути в межах 15-20 градусів відносно вертикальної площини;
- простір для ніг висотою не менше 600 мм, шириною не менше 300 мм та глибиною не менше 400 мм.

Ширина та глибина сидіння повинні бути не меншими за 400 мм. Висота поверхні сидіння має регулюватися в межах 400 - 500 мм, а кут нахилу поверхні – від 15 град. вперед до 5 град. назад. Висота спинки сидіння має становити 300+ -20 мм, ширина – не менше 380 мм. Кут нахилу спинки повинен регулюватися в межах 0 - 30 град. відносно вертикального положення. Відстань від спинки до переднього краю сидіння повинна регулюватись у межах 260 - 400 мм.

Розташування монітору має забезпечувати зручність спостереження під кутами від -30 до +30 градусів.

Відстань від екрана до очей працівника повинна відповідати нормативним значенням в залежності від діагоналі монітора:

- 14-15" – 60-70 см;
- 17" – 70-80 см;
- 19" – 80-90 см;
- 21" – 90-100 см.

Клавіатура повинна бути розміщена на поверхні столу або спеціальній робочій поверхні, що регулюється за висотою, на відстані 100-300 мм від пра-

цівника. Кут нахилу клавіатури повинен бути в межах 5-15 градусів відносно поверхні столу.

Виходячи з результатів аналізу важкості та напруженості праці пропоную скоротити час роботи за комп'ютером, робити перерви, сумарний час яких повинен складати 50 хвилин при 8-ми годинній зміні.

5.3 Виробнича санітарія

У даному розділі розглянемо повітряне середовище робочої зони програміста – температуру та вологість повітря, швидкість руху повітря, інтенсивність теплового випромінювання. Робота програміста за енерговитратами відноситься до категорії легких робіт Іа, Іб, тому повинні дотримуватися наступні вимоги згідно ДСН 3.3.6.042-99:

- оптимальна температура повітря – 22°С (допустима – 23-28°С);
- оптимальна відносна вологість – 40-60% (допустима – не більш 75%);
- швидкість руху повітря не більш 0,1 м/с.

Виміряні за допомогою приладів (психрометр Августа) температура та вологість у лабораторії майже відповідають вказаним у таблиці для теплого періоду року.

Розташовані у приміщенні ПК являються джерелами тепловиділень, крім того для підтримання у приміщенні в холодний період року оптимальних параметрів мікроклімату використовуються нагріті поверхні опалювальної системи. Нормованим показником ІЧВ являється гранично допустима густина потоку енергії Іг.д. Вт/м^2, яка встановлюється в залежності від площі опромінюваної поверхні тіла людини (Sопр).

Нормовані рівні складають:

- Іг.д = 35 Вт/м^2 при Sопр > 50%;
- Іг.д = 70 Вт/м^2 при Sопр ~ 25-50%;

- Іг.д = 100 Вт/м^2 при Sonp < 25%.

Для створення й автоматичної підтримки в приміщенні незалежно від зовнішніх умов оптимальних значень температури, вологості, чистоти і швидкості руху повітря, у холодний час року використовується водяне опалення, у теплий час року застосовується кондиціонування повітря.

Нормованим параметром природного освітлення згідно ДБН В.2.5-28 – 2006 являється коефіцієнт природного освітлення (КПО). КПО встановлюється в залежності від розряду виконуваних зорових робіт. Робота програміста відноситься до робіт середньої точності (IV розряд зорових робіт, мінімальний розмір об'єкту розрізнення складає 0,5 - 1,0мм), для яких при використанні бокового освітлення КПО=1,5%. Для штучного освітлення нормованим параметром виступає мінімальний рівень освітленості та коефіцієнт пульсації світлового потоку, який не повинний бути більшим ніж 20%. Мінімальна освітленість встановлюється в залежності від розряду виконуваних зорових робіт. Для IV розряда зорових робіт вона складає 300-500 лк.

За даними вимірювань (люксметр Ю-116) рівень природної освітленості поверхні, де розташований ПК програміста, складає 350 лк при освітленості тієї же поверхні відкритим небосхилом в 35000 лк, тобто КПО = 1%, що не відповідає нормативному КПО.

Для штучного освітлення у приміщенні використовуються люмінесцентні лампи, які в порівнянні з лампами розжарювання мають ряд істотних переваг:

- за спектральним складом світла вони близькі до природного світла;
- мають підвищену світлову віддачу (у 2-5 разів вищу, ніж у ламп розжарювання);
- мають триваліший термін служби (до 10 тис. годин).

Допустимі значення параметрів неіонізуючих електромагнітних випромінювань від монітору комп'ютера представлені в таблиці. Нормованим параметром невикористаного рентгенівського виромінювання виступає потужність екс-

позиції дози. На відстані 5 см від поверхні екрану монітору її рівень не повинен перевищувати 100 мкР/год. Максимальний рівень рентгенівського випромінювання на робочому місці зазвичай не перевищує 20 мкР/год.

Допустимі значення параметрів неіонізуючих електромагнітних випромінювань наведені у таблиці 5.2.

Таблиця *5.2*

Допустимі значення параметрів неіонізуючих електромагнітних випромінювань

Найменування параметра		Допустимі значення
Напруженість електричної складової електромагнітного поля на відстані 50 см від поверхні відеомонітора		10 В/м
Напруженість магнітної складової електромагнітного поля на відстані 50 см від поверхні відеомонітора		0,3 Ам
Напруженість електростатичного поля не повинна перевищувати	Для дорослих користувачів	20 кВ/м
	Для дітей	15 кВ/м

На відстані 5-10 см від екрана і корпуса монітора рівні напруженості можуть досягати 140 В/м по електричній слкладовій, що значно перевищує допустимі значення.

Для попередження впровадження небезпечної техніки всі дисплеї повинні бути сертифіковані.

5.4 Електробезпека

Приміщення за небезпекою ураження електричним струмом можна віднести до 1 класу, тобто це приміщення без підвищеної небезпеки (сухе, без пилу, з нормальною температурою повітря, ізольованими підлогами і малим числом заземлених приладів).

У приміщенні використовується мережа змінного струму з напругою 220 В. Персональні комп‘ютери, периферійні пристрої та інша побутова техніка повинні підключатися до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення.

Основними споживачами енергії в приміщенні є:

- 3 комп’ютери;
- кондиціонер;
- 2 Wi-Fi роутери.

Основними причинами ураження електричним струмом можуть бути:

- дотик до металевих неструмоведучих частин (корпусу, периферії комп'ютера), що можуть виявитися під напругою в результаті ушкодження ізоляції;
- використання електричних приладів під час перепадів напруги;
- використання саморобних електроприладів;
- використання приладів, що не відповідають характеристикам мережі;
- недотримання правил техніки безпеки.

Категорично забороняється:

- експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості за час експлуатації, ізоляцією;
- залишення під напругою кабелів та проводів з неізольованими провідниками;
- використання саморобних подовжувачів, які не відповідають вимогам ПВЕ до переносних електропроводок;

- застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання;
- користування пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має сліди затемнення або випинання;
- підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною та іншими горючими матеріалами, експлуатація їх зі знятими ковпаками (розсіювачами);
- використання електроапаратури та приладів в умовах, що не відповідають вказівкам (рекомендаціям) підприємств-виготовлювачів;
- відкрита прокладка кабелів;
- використання кабелів в ізоляції з вулканізованої гуми та інших сірковмісних матеріалів.

Основними заходами щодо уникнення ураження електричним струмом є автоматичне вимкнення подачі струму, недоступне розташування відкритих струмоведучих частин, проведення інструктажів з техніки безпеки та інших організаційних заходів. Під час проведення технічних або ремонтних робіт заходи безпеки забезпечуються попередженням та відключенням всіх електроспоживаючих пристроїв від мережі.

5.5 Пожежна безпека

Будівля, в якій знаходиться робоче приміщення, відноситься до категорії Д по вибуховій і пожежній небезпеці, оскільки в ньому наявні і горючі (побутова техніка, меблі, книги та різні інші паперові носії), і негорючі речі (металеві частини побутової техніки та меблів). За конструктивними характеристиками будівля відноситься до I класу вогнестійкості (будівлі з несучими і огороджую-

чими конструкціями з природних або штучних кам'яних матеріалів, бетону або залізобетону з застосуванням листових і плиткових негорючих матеріалів).

Основними причинами виникнення пожеж у будівлі можуть бути:

- недбале використання приладів з нагрівальними елементами (кондиціонер);
- несправності електропроводки, розеток і вимикачів, які можуть призвести до короткого замикання;
- використання пошкоджених електроприладів, кабелів чи подовжувачів;
- необережне поводження з вогнем;
- недотримання заходів пожежної безпеки;
- загорання будівлі через зовнішні фактори;
- влучення блискавки в будинок.

Для сповіщення про виникнення пожежонебезпечної ситуації використовується автоматична пожежна сигналізація з автоматичними димовими сповіщувачами в коридорах і окремих кімнатах. Для гасіння пожеж використовується вогнегасник ОУ-5. Будівля у якій знаходиться робоче приміщення відноситься до будівель блокового типу і має вихід на пожежну драбину для евакуації з кожного блоку. План евакуації передбачає вихід працівників окремих блоків з будівлі через пожежні драбини.

Основними засобами запобігання пожеж є дотримання правил пожежної безпеки, використання тільки справних побутових приладів, своєчасний ремонт побутових приладів, проведення інструктажів з техніки безпека та правил евакуації.

5.6 Розрахунок середнього шуму на робочому місці програміста

Розрахунок середнього рівня шуму на робочому місці програміста виконуємо з урахуванням рівня звукового тиску від різних джерел (див. Табл. 5.3).

Таблиця *5.3*

Середній рівень шуму на робочому місці програміста

Джерело шуму	Рівень шуму, дБА
Жорсткий диск	45
Вентилятор	45
Принтер	55
Сканер	50

Рівень шуму, що виникає від декількох некогерентних джерел, що працюють одночасно, підраховується на підставі принципу енергетичного підсумовування рівня інтенсивності окремих джерел:

$$L_{\Sigma} = 10 * lg \sum 10^{0,1L_i}$$

де, L_i – рівень звукового тиску і-го джерела шуму; n – кількість джерел шуму. Підставивши значення рівняння звукового тиску для кожного виду устаткування у формулу, отримаємо:

$$L = 10 * \lg\ (10^{4,5} +\ 10^{4,5} + 10^{5,5} +\ 10^{5,0}) = 44{,}2\ \text{дБА}.$$

За наявності декількох джерел шуму з однаковим рівнем інтенсивності L_i загальний рівень шуму визначають за формулою:

$$L = Li + 10 * \lg(n)$$

У нашому випадку таких джерел три, отже загальний рівень шуму буде визначатися так:

$$L = 44{,}2 + 10 * \lg(3) = 48{,}9 \text{ дБА}.$$

Розраховане значення середнього рівня шуму не перевищує гранично допустимий рівень шуму для робочого місця програміста, тобто не треба передбачати заходи по зниженню рівня шуму.

5.7 Висновки з розділу

В даному розділі дипломної роботи проаналізовано і визначено основні шкідливі фактори і чинники, що впливають на працівника, перевірено умови праці на відповідність нормативам і запропоновано заходи щодо поліпшення умов праці.

Зроблено опис повітряного середовища робочої зони – температури, вологості повітря та швидкості руху повітря.

В підрозділі «Електробезпека» визначено основні споживачі струму, виявлено фактори, що можуть призвести до ураження електричним струмом та описано заходи щодо його уникнення.

В підрозділі «Пожежна безпека» визначено категорію вибухової та пожежної небезпеки, клас вогнестійкості будівлі, визначено потенційні причини виникнення їх пожеж, описано заходи щодо їх уникнення.

ВИСНОВКИ

В результаті виконання магістерської роботи:

1. Встановлена актуальність проблеми нечіткого пошуку слів у словниках та текстах.

2. Шляхом порівняння та аналізу існуючих рішень і алгоритмів було сформовано набір методів для дослідження даної проблеми, а також визначено технології та інструменти розробки.

3. Для реалізації програмного застосунку обрано мови PHP, JavaScript, HTML, CSS та семантичний словник WordNet, що мотивовано високою ефективністю та зручністю використання. У якості СУБД використано MySQL.

4. Обрані методи вирішення дозволяють досягнути якісного результату, та дають уявлення про основні проблеми, що виникають при рішенні задач цього класу, та методи їх вирішення.

5. В ході роботи була реалізована програмна система для нечіткого пошуку у тексті та словнику з використання методу опорних векторів та метрики Левенштейна.

6. Проведене дослідження продуктивності і надійності програмної системи та аналіз отриманих результатів.

7. Дослідження проблеми показали, що у наш час на ринку відсутня подібна загальнодоступна програмна система, тому створений програмний застосунку для нечіткого пошуку у словнику є доцільним.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гасфилд. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, 2003.
2. *Nello Cristianini, John Shawe-Taylor.* An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. – Cambridge University Press, 2000. – ISBN 978-1-139-64363-4.
3. *Bishop C. M.* Pattern Recognition and Machine Learning. — Springer, 2006. — 738 p.
4. R. A. Wagner, M. J. Fischer. The string-to-string correction problem. J. ACM 21 1 (1974). P. 168—173.
5. Расстояние Левенштейна – определяем «похожесть» строк [Електронний ресурс] // muzhig. – 2012. – Режим доступу до ресурсу: http://muzhig.ru/levenstein-distance-python/.
6. *Alexander Statnikov, Constantin F. Aliferis, Douglas P. Hardin.* A Gentle Introduction to Support Vector Machines in Biomedicine: Theory and methods. – World Scientific, 2011. — ISBN 978-981-4324-38-0.
7. *Alexey Nefedov.* Support Vector Machines: A Simple Tutorial. – 2016.
8. Нечёткий поиск в тексте и словаре [Електронний ресурс] // ntz. – 2011. – Режим доступу до ресурсу: https://habrahabr.ru/post/114997/.
9. *Hastie T., Tibshirani R., Friedman J.* The Elements of Statistical Learning. Springer. – 2014.
10. R.-E. Fan, P.-H. Chen, C.-J. Lin. Working set selection using second order information for training SVM Journal of Machine Learning Research, V. 6, 2005, pp. 1889–1918.
11. J. Friedman, T. Hastie, R. Tibshirani. Regularized Paths for Generalized Linear Models via Coordinate Descent, Journal of Statistical Software, V. 33, No. 1, 2010.

12. О.С. Середин. Методы и алгоритмы беспризнакового распознавания образов, Дисс. к.ф.- м.н., Тульский государственный университет, 2001.
13. Burges C.J.C. A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery 2:121–167, 1998
14. Burges C.J.C., Knirsch P., Haratsch R. Support vector web page: http://svm.research.bell-labs.com. Technical report, Lucent Technologies, 1996.
15. Cortes C., Vapnik V. Support vector networks. Machine Learning, 20. — P. 273—297, 1995.
16. Bartlett P., Shawe-Taylor J. Generalization performance of support vector machines and ot he r pattern classifiers // Advances in Kernel Methods: MIT Press, Cambridge , USA , 1998.
17. Platt J. C. Fast training support vector machines using sequential minimal optimization // Advances in Kernel Methods / Ed. by B . Schölkopf, C. C. Burges, A. J. Smola. MIT Press, 1999. P. 185—208.
18. Vapnik V., Chapelle O. Bounds on error expectation for support vector m achine s // Neural Computation. — 2000. — Vol. 12, no. 9. — P. 2013—2036.
19. Smola A. J., Schölkopf B. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. Algorithmica, 22:211 — 231, 1998.
20. Nello Cristianini, John Shawe-Taylor. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press,2000. — 204 p.
21. John Shawe-Taylor, Nello Cristianini. Kernel Methods for Pattern Analysis. Cambridge University Press, 2004. — 462 p.
22. Ingo Steinwart, Andreas Christmann,; Support Vector Machines, Springer-Verlag, New York, 2008. — 602 p.
23. Muller K., Mika S.An Introduction to Kernel-Based Learning Algorithms, IEEE Neural Networks. — 2001. — №12(2), — P. 181—201.

24. Расстояние Левенштейна [Електронний ресурс] // 2017. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Расстояние_Левенштейна.
25. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848.
26. Кожемякін Г. Б. ОХОРОНА ПРАЦІ ТА ТЕХНОГЕННА БЕЗПЕКА Методичні вказівки до виконання розділу магістерських робіт для студентів ЗДІА всіх спеціальностей денної та заочної форм навчання / Г. Б. Кожемякін, В. Г. Рижков, К. В. Бєлоконь. – Запоріжжя: ЗДІА, 2012. – 49 с.