

Національний технічний університет України  
«Київський Політехнічний Інститут ім. Ігоря Сікорського»  
Міністерство освіти і науки України

Національний технічний університет України  
«Київський Політехнічний Інститут ім. Ігоря Сікорського»  
Міністерство освіти і науки України

Кваліфікаційна наукова  
праця на правах рукопису

ЯРЕМЕНКО ВАДИМ СЕРГІЙОВИЧ

УДК 004.62:004.67:004.75:004.8

**ДИСЕРТАЦІЯ**  
**МОДЕЛЬ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОЇ**  
**ПОБУДОВИ СЛОВНИКА ПРЕДМЕТНОЇ ОБЛАСТІ ПРИ ОБРОБЦІ**  
**ПОТОКОВИХ ДАНИХ**

122 – Комп'ютерні науки  
Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

\_\_\_\_\_ В. С. Яременко

Науковий керівник: Рогоза Валерій Станіславович, д.т.н, професор

Київ – 2025

## АНОТАЦІЯ

### **Яременко В. С. МОДЕЛЬ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ СЛОВНИКА ПРЕДМЕТНОЇ ОБЛАСТІ ПРИ ОБРОБЦІ ПОТОКОВИХ ДАНИХ.**

Кваліфікаційна наукова праця на правах рукопису.

Дисертаційна робота на здобуття наукового ступеня доктора філософії за спеціальністю 122 «Комп'ютерні науки». – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», 2025.

**Метою дисертаційного дослідження** є розширення функціоналу існуючих мультіагентних систем завдяки розробці моделі ефективної мультіагентної системи для обробки поточкових текстових даних, яка забезпечує швидку фільтрацію, точну класифікацію та адаптивне оновлення доменних словників, використовуючи модифікований фільтр Блума, спеціалізовану нейронну мережу та колективне голосування агентів за нові словники використовуючи комбінацію методів Шульце та TF-IDF.

**Об'єктом дослідження** є процеси обробки поточкових текстових даних, що включають фільтрацію, багатокласову класифікацію та оновлення доменних словників. **Предметом дослідження** є методи та засоби для розробки моделі мультіагентної системи для фільтрації, багатокласової класифікації та оновлення доменних словників у контексті потокової обробки текстових даних.

**В першому розділі** обґрунтовано актуальність дослідження в напрямках мультиагентних систем, обробки потоків текстових даних та використання методів машинного навчання, зважаючи на зростання обсягів інформації. Проведено аналіз наукових праць, визначено невирішені задачі та проблеми, а також описано необхідні експерименти. Запропоновано абстрактну модель мультиагентної системи для аналізу слабоструктурованих текстових даних. У результаті сформульовано задачу дисертації.

**Другий розділ** присвячений розробці моделі мультиагентної системи для автоматичної класифікації вхідних текстів і побудови словника предметної області в умовах постійного надходження великого обсягу даних. У розділі розглянуто практичні аспекти проєктування мультиагентних систем для розподілених обчислень, включаючи стандарти FIPA, мову комунікації ACL, можливі стани агентів і особливості їхнього розгортання на обчислювальних вузлах. Проаналізовано існуючі програмні бібліотеки та фреймворки для створення мультиагентних систем, їхні обмеження та можливості розширення. Запропоновано підходи до організації комунікації та прийняття рішень агентами, зокрема механізм голосування для узгодження кінцевого вигляду словника, а також детально описано мультиагентні підсистеми та формати запитів для їх роботи, запропоновано адаптацію методу Шульце для роботи в розподіленому середовищі.

**Третій розділ** присвячено вирішенню задачі багатокласової класифікації поточкових текстових даних, яка є ключовою для автоматичної побудови словників предметних областей. У розділі розглянуто теоретичні основи і визначено напрямки вдосконалення існуючих моделей. Запропоновано модифікацію фільтра Блума для багатокласової класифікації, а також описано

використання моделей нейронних мереж для цієї задачі. Проаналізовано можливість інтеграції цих методів у мультиагентну систему, де кожен агент виконує специфічні задачі. Також розглянуто підходи до автоматичної побудови словників, включаючи обробку текстів і їхніх класів, локальне оновлення словників агентами, створення нових агентів у разі перевантаження та узгодження змін у загальному словнику через комунікацію між агентами.

**Четвертий розділ** присвячений практичній реалізації запропонованої моделі та методів, які є базою для практичного втілення даної моделі в середовище МАС і створено комплекс інструментальних програм, який доводить ефективність запропонованої моделі в автоматизованій побудові словників предметної області. У розділі описано процес створення та налаштування моделі мультиагентної системи, яка здійснює багатокласову фільтрацію та класифікацію текстів і формує словник у кількох ітераціях роботи системи. Реалізація включає інтеграцію модифікованого фільтра Блума, нейронних мереж, а також організацію взаємодії між агентами для ефективного оновлення словника та узгодження змін у ньому. Результати роботи системи проілюстровано прикладами ітерацій, що демонструють функціональність та продуктивність запропонованої архітектури.

### **Наукова новизна отриманих результатів.**

Вперше запропоновано модель мультиагентної системи, яка поєднує модифікований фільтр Блума, нейронну мережу для класифікації текстів, мультиагентний підхід для побудови та оновлення словників і механізм голосування методом Шульце з використанням методу TF-IDF, що дозволяє автоматизувати процес створення словників предметної області в умовах потокової обробки текстових даних.

Вперше запропоновано модифікацію класичного фільтра Блума, який відрізняється тим, що він забезпечує швидке виявлення релевантних текстів і виконання їх попередньої класифікації, що забезпечує значне зменшення обсягу необроблених даних на наступних етапах системи та підвищує ефективність роботи в умовах обробки потокових даних.

Вперше запропоновано модифікацію методу TF-IDF в розподіленому середовищі для вирішення задачі побудови словника предметної області, яка відрізняється застосуванням адаптованого методу Шульце для використання у мультиагентних системах при голосуванні між агентами щодо оновлення доменних словників, що забезпечує ухвалення рішень на основі колективного аналізу текстових даних.

Вперше запропоновано метод оптимізації параметрів налаштування фільтру Блума з використанням генетичного алгоритму для застосування у задачі багатокласової фільтрації потокових текстових даних для підвищення точності їх попередньої класифікації.

Розроблено модель нейронної мережі, адаптовану для класифікації текстових даних за кількома предметними областями для заданого набору даних.

Розроблено механізм динамічної адаптації системи зворотного зв'язку, який передає оновлений словник назад у фільтр Блума. Це дозволяє динамічно адаптувати систему до нових даних і підвищити точність фільтрації текстів. Такий підхід забезпечує постійну актуальність системи в умовах змінного середовища даних.

**Практичне значення отриманих результатів** полягає у створенні ефективної моделі мультиагентної системи для обробки потокових текстових

даних, яка дозволяє автоматизувати процеси фільтрації, класифікації та адаптивного оновлення доменних словників.

Запропоновані підходи забезпечують високу швидкість обробки даних завдяки використанню покращеного фільтра Блума та спеціалізованої нейронної мережі, а також підвищують точність і релевантність оновлених словників завдяки колективному голосуванню агентів за методом Шульце.

Запропонована модель мультиагентної системи відзначається порівняно низькими вимогами до обчислювальних ресурсів, що дозволяє її ефективно реалізовувати на доступних апаратних платформах. Використання покращеного фільтра Блума зменшує обсяг даних, що передаються на подальші етапи обробки, а розподілений характер системи забезпечує гнучкість та можливість ефективного масштабування відповідно до потреб користувача.

Це рішення може бути застосоване для побудови систем моніторингу, аналізу текстових потоків, інформаційного пошуку та виявлення даних у різних доменах, зокрема в кібербезпеці, фінансових системах, соціальних мережах тощо.

**Ключові слова:** мультиагентна система, модель системи, паралельні і розподілені обчислення, аналіз текстових даних, обробка потоків текстів, багатопотоковість, обчислювальний інтелект, машинне навчання, нейронні мережі, глибоке навчання, LSTM, TF-IDF, методи оптимізації, фільтр Блума, метод Шульце.

## ABSTRACT

### **YAREMENKO V. S. A MULTIAGENT SYSTEM MODEL FOR AUTOMATED DOMAIN DICTIONARY CONSTRUCTION IN STREAM DATA PROCESSING.**

A qualification scientific work as a manuscript.

Dissertation for obtaining the Doctor of Philosophy scientific degree in specialty 122 "Computer Science". – National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 2025.

**The aim** of the dissertation research is to expand the functionality of existing multi-agent systems by developing a model of an efficient multi-agent system for processing streaming textual data. This system ensures fast filtering, accurate classification, and adaptive updating of domain dictionaries using a modified Bloom filter, a specialized neural network, and collective agent voting for new dictionaries through a combination of the Schulze method and TF-IDF.

**Object of the research:** the processes of streaming text data processing, which include filtering, multi-class classification, and updating domain-specific dictionaries. **Subject of the research:** the methods and tools for developing a multi-agent system model for filtering, multi-class classification, and updating domain-specific dictionaries in the context of streaming text data processing.

**The first chapter** substantiates the relevance of research in the areas of multiagent systems, streaming text data processing, and machine learning methods, emphasizing the growing volumes of information. A review of scientific literature is provided, identifying unresolved issues and challenges, along with necessary

experiments. An abstract model of a multiagent system for analyzing weakly structured text data is proposed. As a result, the dissertation's objective is formulated.

**The second chapter** focuses on the development of a multiagent system model for the automatic classification of incoming texts and the construction of a domain dictionary in conditions of a continuous influx of large amounts of data. The chapter discusses practical aspects of designing multiagent systems for distributed computing, including FIPA standards, ACL communication language, agent states, and deployment specifics on computational nodes. Existing software libraries and frameworks for multiagent system development are analyzed, highlighting their limitations and extensibility. Approaches to organizing communication and decision-making among agents, such as the Schulze and TF-IDF methods adaptation for distributed environments, are proposed. Multiagent subsystems and query formats for their operations are described in detail.

**The third chapter** addresses the problem of multiclass classification of streaming text data, which is key to the automated construction of domain-specific dictionaries. Theoretical foundations are reviewed, and directions for improving existing models are outlined. A modification of the Bloom filter for multi-class classification is proposed, along with the use of neural networks for this task. The integration of these methods into a multiagent system, where each agent performs specific tasks, is analyzed. Approaches to automated dictionary construction are examined, including processing text and its classes, local dictionary updates by agents, creating new agents in cases of overload, and synchronizing changes in the general dictionary through inter-agent communication.

**The fourth chapter** is dedicated to the practical implementation of the proposed model and methods, which serve as the foundation for deploying this



model in a multi-agent system environment. A set of instrumental software tools has been developed to demonstrate the effectiveness of the proposed model in the automated construction of domain-specific dictionaries. The chapter describes the process of creating and configuring the multi-agent system model, which performs multi-class text filtering and classification while iteratively forming the dictionary during system operation. The implementation includes the integration of a modified Bloom filter, neural networks, and the organization of agent interactions to ensure efficient dictionary updates and consensus on changes. The system's performance is illustrated with iteration examples that demonstrate the functionality and efficiency of the proposed architecture.

### **Scientific novelty of the results obtained.**

For the first time, a model of a multi-agent system has been proposed, combining a modified Bloom filter, a neural network for text classification, a multi-agent approach for dictionary construction and updating, and a voting mechanism based on the Schulze method using the TF-IDF method. This enables the automation of domain-specific dictionary creation in the context of streaming text data processing.

A novel modification of the classical Bloom filter has been proposed, distinguished by its ability to rapidly detect relevant texts and perform their preliminary classification. This significantly reduces the volume of raw data at subsequent stages of the system and enhances efficiency in the context of streaming data processing.

For the first time, a modification of the TF-IDF method in a distributed environment has been proposed to address the task of constructing a domain-specific dictionary. This modification is distinguished by the application of an adapted Schulze method for use in multi-agent systems, enabling agents to vote on domain

dictionary updates. This ensures decision-making based on collective analysis of textual data.

For the first time, a method for optimizing Bloom filter parameters using a genetic algorithm has been proposed for use in multi-class filtering of streaming textual data to improve the accuracy of their preliminary classification.

A neural network model has been developed and adapted for multi-class text data classification across multiple domains for the given data set.

A feedback system mechanism for dynamic adaptation has been developed, transmitting the updated dictionary back to the Bloom filter. This allows the system to adapt dynamically to new data and improve text classification accuracy at the filtering stage. This approach ensures the system's relevance in changing data environments.

**Practical significance of the results.** The practical significance of the research lies in the creation of an efficient multi-agent system for processing streaming text data, automating the processes of filtering, classification, and adaptive updating of domain dictionaries.

The proposed approaches ensure high data processing speed through the enhanced Bloom filter and specialized neural network and improve the accuracy and relevance of updated dictionaries via agents' collective voting using the Schulze method.

The proposed multi-agent system model is characterized by relatively low computational resource requirements, allowing for its efficient implementation on accessible hardware platforms. The use of an improved Bloom filter reduces the volume of data transmitted to subsequent processing stages, while the distributed nature of the system ensures flexibility and enables effective scaling according to user needs.

This solution can be applied in monitoring systems, text stream analysis, information retrieval, and data detection in various domains, such as cybersecurity, financial systems, and social networks.

**Keywords:** multiagent system, system model, parallel and distributed computing, text data analysis, text stream processing, multithreading, computational intelligence, machine learning, neural networks, deep learning, LSTM, TF-IDF, optimization methods, Bloom filter, Schulze method.

## СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

*Зарубіжні видання, що індексуються у наукометричній базі Scopus (Q4):*

1. Yaremenko V., Rogoza W., Spitkovskyi V. Application of neural network algorithms and naïve bayes for text classification. Journal of Theoretical and Applied Information Technology. 2021. Vol.99. No 1. P. 125-134. ISSN 1992-8645. E-ISSN 1817-3195. (Здобувачем Яременком В. С. було чітко визначено підхід до тестування системи класифікації текстів, запропоновані архітектури нейронних мереж та параметри класифікатора наївного Байєса для порівняння, були обрані тексти, які варто порівняти та проаналізовані результати.)

*Статті у наукових фахових виданнях України (категорія «Б»):*

2. Яременко В. С. Огляд наявних мультиагентних систем для задач інтелектуального аналізу даних. Вчені записки Таврійського національного університету імені В. І. Вернадського. 2018. Том 29 (68), №3. С. 47–55. (Здобувачем Яременком В. С. було проведено аналіз існуючих мультиагентних систем, які створені для вирішення задач інтелектуального аналізу даних. Були визначені спільні компоненти таких систем та підготовлено узагальнений опис моделі мультиагентної системи.)

3. Яременко В. С., Будьонний Д. Ю. Підхід до використання фільтра Блума для багатокласової класифікації текстових даних в режимі реального часу. Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво". 2019. №6. С. 153–159. (Здобувачем Яременком В. С. було проаналізована велика кількість джерел, книг та текстів, які стосуються аналізу текстових даних, а саме – швидку класифікацію потоків текстів. Було визначено, що фільтр Блума, який показує хороші результати по швидкості та точності класифікації при обробці поточкових даних, можливий лише для

однокласової класифікації, тому здобувачем запропонована його модифікація, щоб зробити можливість виконувати фільтрацію текстів, які не належать необхідним класам.)

4. Яременко В. С., Худяков А. С. Модель мультиагентної системи для семантичного аналізу текстів. Міжвузівський збірник наукових праць "Наукові нотатки". 2019. №68. С. 152–156. (Здобувачем Яременком В. С. була запропонована модель мультиагентної системи, що дозволяє обробляти текстові дані в розподіленому середовищі, проведені початкові етапи дослідження.)

5. Yaremenko V., Syrotiuk O. Development of a multi-agent system for solving domain dictionary construction problem. Technology audit and production reserves. 2020. №4/2 (54). P. 27–30. (Здобувачем Яременком В. С. було запропонована модель системи та основа даного підходу до вирішення задачі автоматичної побудови словника предметної області, проведені початкові етапи дослідження та визначене теоретичне підґрунтя – методи побудови словників, сформульоване чітке технічне завдання для програмування даної системи.)

6. Hryshchenko O. A comparative analysis of text data classification accuracy and speed using neural networks, Bloom filter and Naive Bayes / O. Hryshchenko, V. Yaremenko. // Technology audit and production reserves. – 2021. – P6-8. - №5/2(61). (Здобувачем Яременком В. С. були запропоновані параметри класифікаторів текстових даних для подальшого їх тестування та порівняльного аналізу з метою отримання оцінки точності та швидкості класифікації.)

7. Яременко В. С., Тарасенко М. В. Порівняльний аналіз програмних бібліотек для класифікації текстових даних із використанням штучних нейронних мереж. Вчені записки ТНУ імені В.І. Вернадського. Серія: технічні науки. 2019. Том 30 (89), №3. С. 214–218. (Здобувачем Яременком В. С. були проаналізовані можливості сучасних бібліотек для розробки класифікаторів текстових даних та запропонований підхід до їх порівняння, щоб обрати одну з них для використання у дослідженні.)

8. Polozniuk K., Yaremenko V. Neural networks and Monte-Carlo method usage in multi-agent systems for sudoku problem solving. Technology audit and production reserves. 2020. №6/2 (56). P. 38-41. (Здобувачем Яременком В. С. була запропонована модель мультиагентної системи, що працює на базі алгоритмів глибокого навчання з підкріпленням та здатна вирішувати задачі високої обчислювальної складності.)

# ЗМІСТ

|  |           |
|--|-----------|
| <b>АНОТАЦІЯ .....</b>  | <b>2</b>  |
| <b>ABSTRACT .....</b>  | <b>7</b>  |
| <b>СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ .....</b>   | <b>12</b> |
| <b>ЗМІСТ .....</b>   | <b>15</b> |
| <b>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ .....</b>   | <b>17</b> |
| <b>ВСТУП .....</b>   | <b>18</b> |
| <b>РОЗДІЛ 1 ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ОБРАНОГО НАПРЯМКУ ДОСЛІДЖЕНЬ ТА ФОРМУЛЮВАННЯ ЗАДАЧІ. ....</b>   | <b>21</b> |
| 1.1. АКТУАЛЬНІСТЬ ЗАДАЧІ. ....   | 21        |
| 1.1.1. Розвиток мультиагентних систем. ....  | 21        |
| 1.1.2. Оброблення текстової інформації та існуючі проблеми. ....   | 25        |
| 1.1.3. Машинне навчання та оброблення потокових текстових даних. ....  | 29        |
| 1.2. ФОРМУЛЮВАННЯ ЗАДАЧІ ШВИДКОЇ ОБРОБКИ СЛАБОСТРУКТУРОВАНИХ ТЕКСТОВИХ ПОТОКІВ ДАНИХ. ....   | 37        |
| 1.2.1. ЗАДАЧА ПОБУДОВИ АРХІТЕКТУРИ СИСТЕМИ. ....   | 39        |
| 1.2.2. ЗАДАЧА ДОСЛІДЖЕННЯ ТА РОЗРОБКИ МЕТОДІВ ШВИДКОЇ КЛАСИФІКАЦІЇ ТЕКСТІВ. ....   | 40        |
| 1.2.3. ЗАДАЧА ДОСЛІДЖЕННЯ ТА РОЗРОБКИ МЕТОДІВ АВТОМАТИЧНОЇ ПОБУДОВИ СЛОВНИКА ПРЕДМЕТНОЇ ОБЛАСТІ. ....  | 40        |
| 1.3. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ. ....  | 41        |
| 1.3.1. МУЛЬТИАГЕНТНІ СИСТЕМИ ДЛЯ АНАЛІЗУ ТЕКСТОВИХ ДАНИХ. ....   | 42        |
| 1.3.2. ОБРОБЛЕННЯ ВЕЛИКИХ ДАНИХ З ВИКОРИСТАННЯМ МУЛЬТИАГЕНТНИХ СИСТЕМ. ....  | 44        |
| 1.3.3. РІШЕННЯ ДЛЯ ПОБУДОВИ СПЕЦІАЛІЗОВАНИХ СЛОВНИКІВ ПРЕДМЕТНИХ ОБЛАСТЕЙ. ....  | 46        |
| 1.3.4. УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ ТА ПОДАЛЬШІ КРОКИ ДОСЛІДЖЕННЯ. ....  | 47        |
| 1.4. ВИСНОВКИ. ....  | 49        |
| <b>РОЗДІЛ 2 МОДЕЛЬ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ ПОТОКІВ ТЕКСТОВИХ ДАНИХ. ....</b>  | <b>51</b> |
| 2.1. ПРАКТИЧНІ АСПЕКТИ ПОБУДОВИ МУЛЬТИАГЕНТНИХ СИСТЕМ У РОЗПОДІЛЕНОМУ СЕРЕДОВИЩІ. ....   | 52        |
| 2.2. МОДЕЛЬ МАС ДЛЯ ВИРІШЕННЯ ЗАДАЧІ АВТОМАТИЧНОЇ ПОБУДОВИ СЛОВНИКА ПРЕДМЕТНОЇ ОБЛАСТІ, ВЗАЄМОДІЯ АГЕНТІВ, МОВА СПІЛКУВАННЯ МІЖ АГЕНТАМИ. ....       | 60        |
| 2.3. АНАЛІЗ ТА ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ ТА ПРОГРАМНИХ БІБЛІОТЕК ДЛЯ РОЗРОБКИ МОДЕЛІ МАС ПРИ ОБРОБЛЕННІ СЛАБОСТРУКТУРОВАНИХ ТЕКСТОВИХ ДАНИХ. .... | 70        |
| 2.4. ОПИС МОДЕЛІ МАС ТА ЗАПИТІВ НА МОВІ FIPA ACL ДЛЯ ВИРІШЕННЯ ЗАДАЧІ. ....  | 76        |
| 2.5. МЕТОДИ ГОЛОСУВАННЯ ДЛЯ ВИЗНАЧЕННЯ КІНЦЕВИХ СЛОВНИКІВ В РОЗПОДІЛЕНОМУ СЕРЕДОВИЩІ. ....   | 82        |
| 2.6. ВИСНОВКИ. ....  | 89        |
| <b>РОЗДІЛ 3 МЕТОДИ ФІЛЬТРАЦІЇ, КЛАСИФІКАЦІЇ ТА АНАЛІЗУ ПОТОКІВ ТЕКСТОВИХ ДАНИХ В МОДЕЛІ МАС. ....</b>  | <b>91</b> |
| 3.1. ЗАДАЧА ФІЛЬТРАЦІЇ, КЛАСИФІКАЦІЇ ТА АВТОМАТИЗОВАНОЇ ПОБУДОВИ СЛОВНИКІВ ПРИ ОБРОБЦІ ПОТОКІВ ТЕКСТІВ. ....   | 91        |
| 3.2. МОДИФІКАЦІЯ ФІЛЬТРУ БЛУМА ДЛЯ ШВИДКОЇ БАГАТОКЛАСОВОЇ ФІЛЬТРАЦІЇ ТЕКСТІВ. ....   | 99        |
| 3.2.1. ОПИС МОДИФІКАЦІЇ МОДЕЛІ ФІЛЬТРУ БЛУМА ДЛЯ БАГАТОКЛАСОВОЇ ФІЛЬТРАЦІЇ. ....   | 99        |
| 3.2.2. ОПТИМІЗАЦІЯ ПАРАМЕТРІВ ФІЛЬТРА БЛУМА ЗА ДОПОМОГОЮ ГЕНЕТИЧНОГО АЛГОРИТМУ. ....   | 109       |
| 3.3. ІНТЕГРАЦІЯ НЕЙРОННОЇ МЕРЕЖІ У МОДЕЛЬ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ КЛАСИФІКАЦІЇ ПОТОКОВИХ ТЕКСТІВ. ....  | 112       |

|  |   |            |
|--|---|------------|
| 3.4.   | ОПИС РОБОТИ АГЕНТІВ ПОБУДОВИ СЛОВНИКА ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ TF-IDF.....                               | 115        |
| 3.5.   | Висновки. ....  | 129        |
| <b>РОЗДІЛ 4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АПРОБАЦІЯ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОГО СТВОРЕННЯ СЛОВНИКІВ.....</b> |   | <b>131</b> |
| 4.1.   | ФОРМУЛЮВАННЯ ЕКСПЕРИМЕНТУ ПОБУДОВИ ТЕСТОВОЇ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ СЛОВНИКА ПРЕДМЕТНОЇ ОБЛАСТІ. .... | 131        |
| 4.2.   | Вирішення задачі фільтрації та класифікації для відповідних агентів.....  | 135        |
| 4.2.1.   | ХАРАКТЕРИСТИКИ ТЕКСТОВИХ ДАНИХ .....  | 135        |
| 4.2.2.   | Визначення параметрів модифікованого фільтру Блума з використанням генетичних алгоритмів.....                                     | 137        |
| 4.2.3.   | Тестування фільтру Блума для агентів фільтрації даних. ....   | 141        |
| 4.2.4.   | Навчання та тестування нейронної мережі для агентів класифікації даних. ....  | 144        |
| 4.3.   | Комплексне тестування побудованої мультіагентної системи: ітераційне вдосконалення словників. ....                                | 149        |
| 4.4.   | Висновки. ....  | 164        |
| <b>Висновки.....</b>   |   | <b>166</b> |
| <b>Список використаних джерел .....</b>  |   | <b>169</b> |
| <b>Додаток А. Список публікацій здобувача за темою дисертації.....</b>   |   | <b>174</b> |
| <b>Додаток Б. Приклад протоколу спілкування агентів мовою ACL.....</b>   |   | <b>175</b> |



## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

**МАС** – мультиагентні системи.

**RAKE** (Rapid Automatic Keyword Extraction) – метод автоматичного виділення ключових слів на основі їхньої частоти та співвідношення щільності в тексті.

**TextRank** – графовий алгоритм ранжування тексту, який базується на концепції PageRank, використовується для визначення ключових слів або речень у тексті.

**TF-IDF** (Term Frequency-Inverse Document Frequency) – статистична міра, що оцінює важливість слова в документі відносно всього корпусу текстів.

**Word2Vec** – нейронна модель для перетворення слів у вектори із збереженням їх семантичного контексту.

**N-Gram** – послідовність з N елементів (слів або символів), що використовується для аналізу тексту.

**PADE** (Python Agent DEvelopment framework) – програмний фреймворк для створення мультиагентних систем, що реалізує розподілені обчислення з використанням мови Python.

**FIPA** (Foundation for Intelligent Physical Agents) – міжнародна організація, яка розробляє стандарти для мультиагентних систем, забезпечуючи їхню сумісність та ефективну взаємодію.

**ACL** (Agent Communication Language) – стандартизована мова комунікації між агентами, визначена FIPA, що дозволяє передавати повідомлення та команди між компонентами мультиагентної системи.

## ВСТУП

В умовах стрімкого зростання обсягів потокових даних стає критично важливим автоматизувати процеси їх обробки та структурування. Побудова словників предметної області є ключовою задачею, що забезпечує підвищення точності аналізу даних та ефективність прийняття рішень у різних сферах, зокрема в науці, бізнесі та технологіях. Традиційні підходи до створення таких словників потребують значних витрат ресурсів та часу, що робить їх недостатньо ефективними для обробки великих обсягів потокових даних у реальному часі. Використання мультиагентних систем дозволяє впровадити інноваційні методи автоматизації та забезпечити адаптивність побудови словника до змін у даних. Також, мультиагентні системи (МАС) забезпечують перспективний підхід до розробки таких систем, пропонуючи такі переваги, як масштабованість, модульність і відмовостійкість.

Спеціалізовані словники предметних областей є важливими інструментами для обробки природної мови, машинного навчання та експертних систем. Вони можуть використовуватися у медицині для автоматизованого розпізнавання симптомів та діагнозів у клінічних текстах, у правових системах для аналізу нормативно-правових актів, у фінансовому секторі для виявлення шахрайських транзакцій та ризик-менеджменту, а також у кібербезпеці для ідентифікації загроз та аналізу логів безпеки. Використання таких словників дозволяє підвищити точність автоматизованих систем обробки текстових даних у вузькопрофільних галузях. Таким чином, розробка моделі мультиагентної системи для автоматизованої побудови словників предметної області є актуальним та перспективним напрямком досліджень.

Ця робота представляє мультиагентну систему для автоматичного створення словника, що складається з кількох кроків, таких як фільтрація текстів, класифікація та вилучення слів. На першому кроці тексти фільтруються за допомогою фільтру Блума, щоб ефективно видалити нерелевантні тексти з корпусу. На другому кроці тексти класифікуються за відповідними категоріями, такими як новини, наука та технології, за допомогою класифікатора машинного навчання, навченого на попередньо позначеному наборі даних. Третій крок передбачає розробку механізму голосування між агентами за оновлений доменний словник, оскільки агенти знаходяться в розподіленому середовищі.

Останнім кроком системи є автоматичне паралельне створення словника, коли агенти голосують за включення слів-кандидатів у словник на основі їх релевантності та частоти в корпусі. Процес голосування призначений для того, щоб переконатися, що отриманий словник є вичерпним і репрезентативним для мови, яка використовується в корпусі.

Ця робота має на меті зробити внесок у розробку моделі МАС для автоматизованої побудови словників, представивши комплексний і масштабований підхід, який можна розширити та адаптувати для різних програм обробки природної мови. Запропонована система використовує переваги МАС, такі як паралелізм, розподіл і відмовостійкість, щоб подолати проблеми обробки великих обсягів потокових текстових даних і створення точних і вичерпних словників. Новизна цієї роботи полягає в поєднанні різних технік і методів у цілісну та ефективну модель мультиагентної системи, а також в оцінці продуктивності системи за допомогою реальних наборів даних.

Підсумовуючи, ця робота представляє мультиагентну систему для автоматизованого створення словника, яка поєднує такі методи, як фільтрація текстів, класифікація, вилучення слів і методи голосування для створення точних і вичерпних словників. Система оцінюється з використанням реальних наборів даних, і результати демонструють її ефективність і масштабованість. Запропонована система може бути розширена та адаптована для різних додатків обробки природної мови та сприяє розвитку МАС для обробки та аналізу тексту.

## **РОЗДІЛ 1 ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ ОБРАНОГО НАПРЯМКУ ДОСЛІДЖЕНЬ ТА ФОРМУЛЮВАННЯ ЗАДАЧІ.**

### **1.1. Актуальність задачі.**

#### **1.1.1. Розвиток мультиагентних систем.**

За останні декілька десятиків років мультиагентні системи сягнули значного розвитку. Початкові дослідження цього напрямку в науці стосувались способів та протоколів для комунікації агентів, детального опису платформ для роботи агентів, а також можливостей надання агентам елементарного інтелекту. Проте в останні роки дослідження мультиагентних систем зосереджені в більшій мірі саме на інтелектуальній складовій – через швидкий розвиток нейронних мереж та інших методів машинного навчання.

На початковому етапі дослідження був проведений аналіз наукових праць, які містять поєднання мультиагентних систем, обробки великих потоків даних та методів машинного навчання. Особливу увагу було приділено нейронним мережам, але проаналізовані праці стосувались не лише їх.

Помітна тенденція, що праці, які стосуються поєднання мультиагентних систем та нейронних мереж, в першу чергу розглядають глибоке навчання з підкріпленням (Deep Reinforcement Learning). Даний напрямок чудово поєднується з концепцією мультиагентних систем через особливість визначення інтелектуального агента – як сутності, яка розміщена в певному середовищі та здатна на автономну дію в цьому ж середовищі для досягнення власних цілей (Рис. 1.1) [1]. Відповідно, взаємодіючи з середовищем, агент може отримувати позитивний або негативний зворотній зв'язок, який він і буде враховувати при навчанні. Серед недоліків даного напрямку є те, що не

кожну прикладну задачу інтелектуального аналізу даних можна звести саме до навчання з підкріпленням.

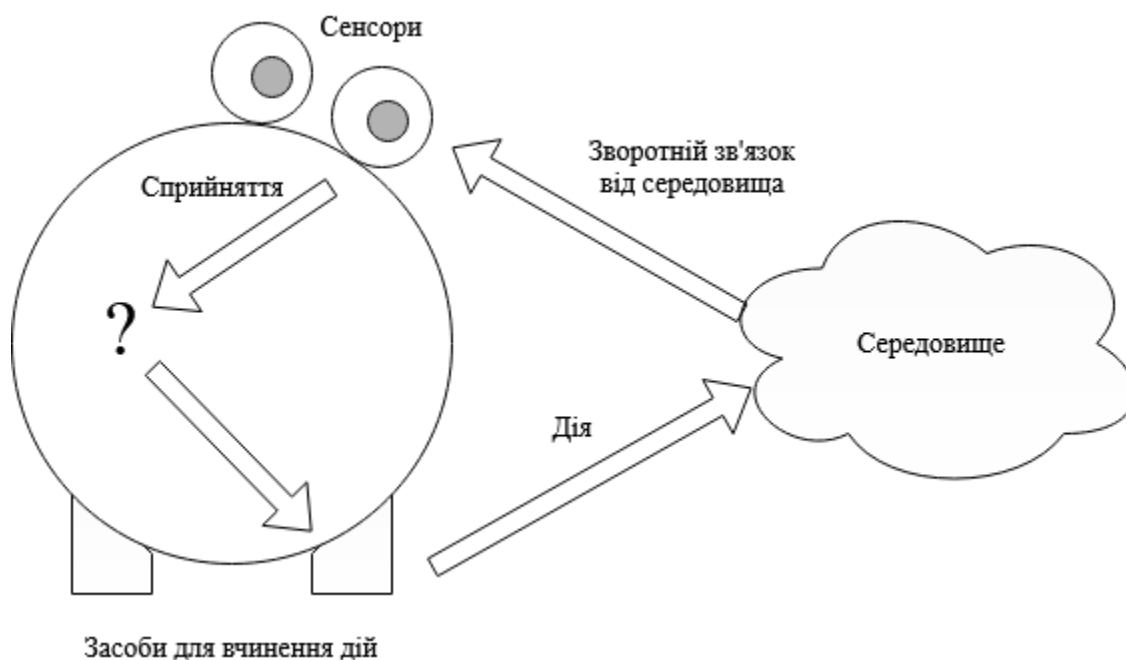


Рис 1.1. Агент у власному середовищі [1]

Відповідно робіт, що стосуються використання мультиагентних систем для навчання з вчителем та без вчителя (задач класифікації та кластеризації) значно менше, тому цей напрям є перспективним для досліджень і був обраний в даній роботі.

Розглядаючи мультиагентні системи варто зазначити й організацію FIPA, яка розробляла стандарти в даному напрямку. Як вказано в книзі [2], FIPA було створено в 1996 році як міжнародну некомерційну асоціацію з метою розробки набору стандартів, що стосуються технологій програмних агентів. На той час програмні агенти були вже дуже відомі в академічній спільноті, але кількість прикладних застосувань з боку комерційних підприємств була малою. Консорціум погодився випустити стандарти, які

становитимуть основу нової галузі та будуть використовуватись у великій кількості прикладних задач.

В основі FIPA лежить наступний набір принципів:

1. Агентні технології надають нову парадигму вирішення старих і нових проблем;
2. Деякі технології агентів досягли значного ступеня розвитку;
3. Для використання деяких технологій агентів потрібна стандартизація;
4. Стандартизація загальних технологій виявилася можливою та забезпечує ефективні результати за допомогою інших форумів стандартизації;
5. Стандартизація внутрішньої механіки самих агентів є не головним завданням, а швидше інфраструктурою та мовою, необхідними для відкритої взаємодії.

Однією з найбільш важливих розробок цієї організації була розробка стандартів для комунікації агентів, які базувались на теорії актів мовлення, яка описує перформативну суть мови та те, як мова може використовуватися як дія. Прикладом може бути «Я – агент класифікації текстів», який надає відправнику подробиці про одержувача. Фонологічний та синтаксичний набір FIPA-ACL з 22 комунікативних актів базується на пропозиції такої організації, як ARCOL (Франція), де будь-який акт представлений як за допомогою форми розповіді, так і формальної семантики на основі модальної логіки, яка визначає наслідки надсилання повідомлення на психічні установки агента відправника та одержувача.

Цей метод логіки узгоджується з парадигмою міркувань BDI - або «Віра, Бажання, Намір». Деякі з найбільш широко застосовуваних дій - це

інформувати, просити, приймати, не розуміти та відхиляти. Вони охоплюють суть більшості режимів простого спілкування. У стандартах FIPA зазначено, що агент повинен бути повністю сумісним для отримання будь-якого повідомлення про комунікативні акти FIPA-ACL і, щонайменше, відповідати на незрозумілі повідомлення, якщо його неможливо обробити. На основі цих комунікативних актів FIPA встановили серію протоколів, кожен з яких складається з послідовності комунікативних актів для організації спілкування, таких як контрактна мережа для формування угод та кілька форм аукціонів. Структура повідомлення зображена на Рис. 1.2. Відповідно, актуальною є задача розробки стандартних протоколів для комунікації агентів при вирішенні нових задач.

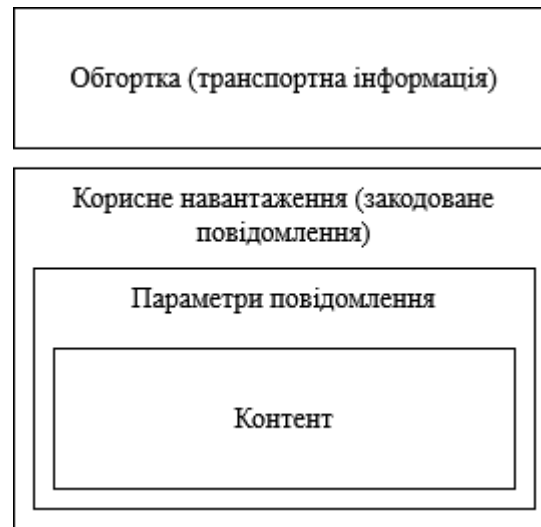


Рис 1.2. Структура повідомлення FIPA [Власна розробка]

Що стосується технологій: розглядаючи мультиагентні системи варто зазначити існуючу систему JADE [2], яка багато років є основною програмною бібліотекою для мови Java. JADE – це проміжне програмне забезпечення, яке пропонує базові функціональні можливості для програмних агентів, незалежних від окремих програм, для спрощення розгортання розподілених



програм. Основною перевагою JADE є те, що вона інтегрує ідею абстракції у добре відому об'єктно-орієнтовану мову Java, пропонуючи простий і приємний API [2].

Але не дивлячись на розвиток JADE, на сьогоднішній день вже були розроблені декілька бібліотек, в тому числі – з відкритим кодом (open-source libraries), що написані на мові програмування Python. Прикладами таких бібліотек є PADE, SPADE, osBrain. Серед досліджених наукових робіт 2015-2020 років більшість праць, що стосувались поєднання мультиагентних систем та методів штучного навчання, були написані саме з використанням мови Python. Однією з причин є те, що існуючі бібліотеки машинного навчання також мають програмний інтерфейс для використання в мові Python, тому їх інтеграція з мультиагентною системою є ще більш простим процесом (з точки зору програмування). Тому, розробляючи прототипи в даній роботі варто звернути увагу на сучасні бібліотеки та, у випадку достатнього функціоналу в них, використати саме їх.

### **1.1.2. Оброблення текстової інформації та існуючі проблеми**

В останні роки ми стали свідками різкого збільшення кількості оцифрованих текстових даних, що призвело до можливості проводити дослідження та генерувати нові уявлення про ці тексти. Оскільки штучний інтелект та машинне навчання все частіше з'являються в технологіях, застосування паралельної обробки, глибокого навчання та розпізнавання шаблонів до тексту стає все більш значущим напрямом наукових досліджень. Поточкові текстові дані існують в різних формах, наприклад – в напівструктурованому вигляді, як файли журналів (англ. log files), що містять

інформацію із серверів та мереж. Таким чином, аналіз тексту корисний як для неструктурованих, так і для напівструктурованих текстових даних [3].



Рис 1.3. Зображення суміжних до аналізу текстових даних напрямів на діаграмі Вена [5]

Аналіз текстів можна використовувати різноманітними способами, включаючи автоматичне сортування повідомлень та електронних листів. Наприклад, можна автоматично відфільтрувати електронні листи, якщо вони містять певні ключові слова, ці електронні листи будуть автоматично відкинуті або позначені як спам. Такі автоматизовані системи класифікації електронних повідомлень також можуть бути корисними. У деяких сферах ринку багато інформації збирається у письмовому форматі, тому варто її оцифрувати [4].

Особливо варто звернути увагу на 2020-й рік, коли через пандемію велика кількість підприємств та організацій почали переходити на

дистанційний режим роботи та почали переводити бізнес-процеси в цифровий вигляд. Навіть багато шкіл та університетів, де дистанційне навчання не практикувалось взагалі до цього, були змушені почати проводити заняття та оцінку знань саме в такому режимі. Це означає, що кількість текстової інформації в цифровому вигляді лише зростає, а кількість необробленої – ще більше.

Замість великої кількості знань в Інтернеті, тепер ми можемо зручно знайти лише невеликий їх фрагмент. Якщо ми спробуємо в автоматизованому вигляді поєднати інформацію з кількох джерел, питання стає складнішим. Для вирішення цієї проблеми було запропоновано багато різних методів, і всі вони орієнтовані на складний аналіз даних. Щоб мінімізувати упередженість у змісті, суб'єктивне фільтрування використовується для видалення текстів, які можуть мати неупереджену точку зору. Тексти можна диференціювати, спостерігаючи за ставленням авторів до предмета, або за розрізненням мовчазних та явних опозиційних домовленостей з іншими текстами [6]. В роботі [7] розглянуті два приклади, які стосуються проблеми в суб'єктивній оцінці чогось, яка висловлена в текстовій формі. Відповідно, існує проблема автоматичної класифікації цих висловів. В іншій праці [8] вказуються деякі з існуючих проблем при обробленні текстових даних. Однією з них є розмір наборів даних. Висока розмірність, тобто велика кількість атрибутів, що представляє проблему поганої роботи класифікатора, оскільки багато атрибутів не мають значення (являються пустими).

Напрямок інтелектуального аналізу текстів тісно пов'язаний з багатьма іншими напрямками досліджень в комп'ютерних науках. Гарну ілюстрацію цього було запропоновано в дослідженні [4] та зображено на Рис. 1.3. Наприклад, на перетині інтелектуального аналізу даних, статистики, обробки

текстів та машинного навчання існують такі напрямки, як класифікація та кластеризація документів, тощо. Загальний процес видобутку тексту містить наступні етапи:

- Збір неструктурованих даних з різних джерел, доступних у різних форматах файлів, таких як звичайний текст, веб-сторінки, PDF-файли тощо;
- Для виявлення та усунення аномалій виконуються операції попередньої обробки та очищення тексту, які охоплюють видалення стоп-слів, обробку діакритичних знаків, нормалізацію лексем, стемінг, лематизацію, ідентифікацію коренів слів, а також індексацію даних із збереженням суті тексту;
- Операції обробки та контролю застосовуються для аудиту та подальшого очищення набору даних за допомогою автоматичної обробки;
- Аналіз шаблонів здійснюється за допомогою системи управління інформацією;
- Інформація, оброблена на вищевказаних етапах, використовується для вилучення цінних знань для ефективного та своєчасного прийняття рішень та аналізу тенденцій.

Підсумовуючи описані факти та думки можна прийти до висновку, що кількість викликів, яка стоїть перед науковцями, лише зростає. Основними причинами цього є не лише постійне збільшення текстової інформації, але й необхідність в її швидкому аналізі та обробці, її представлені в різних обсягах і формах, у структурованих, напівструктурованих і неструктурованих формах, тощо. Як результат аналізу праць на предмет можливих покращень, над якими варто працювати, помітна достатньо мала кількість робіт, які пов'язані з автоматичною побудовою словників предметних областей. Словники є однією з основ оброблення текстової інформації, а належність того чи іншого слова

до певної предметної області може бути основою для подальшої класифікації чи кластеризації текстів, наприклад, за частотним аналізом появи цих слів.

### **1.1.3. Машинне навчання та оброблення поточкових текстових даних.**

За останні два десятиліття було зібрано велику кількість даних у кількох областях. За даними міжнародної корпорації даних IDC, у 2011 р. загальний обсяг даних, створених та скопійованих у світі, становив  $1,8 \cdot 10^{21}$  Байт, який за період з 2010 по 2015 роки зріс майже в дев'ять разів. Це число подвоюватиметься принаймні раз на два роки. У порівнянні з традиційними наборами даних, великі дані, як правило, стосуються більш неструктурованих та чутливих до часу даних, які дуже часто потребують швидкого аналізу. За останні роки компанії стали зацікавлені у високому потенціалі великих даних, і кілька державних установ оголосили про великі плани щодо прискорення аналізу та застосування великих даних. Відомі наукові журнали, у тому числі – Nature and Science відкрили на своїх сторінках спеціальні розділи для вирішення проблем та впливу великих даних [9]. На Рис. 1.4. можна побачити приблизну оцінку загальних обсягів існуючих даних в світі за період з 2005 по 2015 роки. В своїй роботі «Видобуток цінностей з хаосу» (англ. «Extracting Value from Chaos») [10] автор звертає увагу читачів на величезне зростання кількості даних та прогнозує, що це зростання буде продовжуватись і надалі.

Відповідно, виникає проблема швидкої обробки цієї інформації. Навіть поглянувши на графік можна прийти до висновку, що величезні обсяги даних й досі є необробленими, однією з причин цього є недостатня розвиненість апаратних можливостей та власне методів обробки цих даних, що робить обробку неможливою або занадто дорогою. Інший висновок, до якого можна прийти виходячи з попередніх тверджень – у випадку, якщо дані не будуть

одразу оброблені та з них не буде видобута «цінність», то ці дані так і залишаться надалі в такому ж стані через постійне надходження нової і нової інформації.

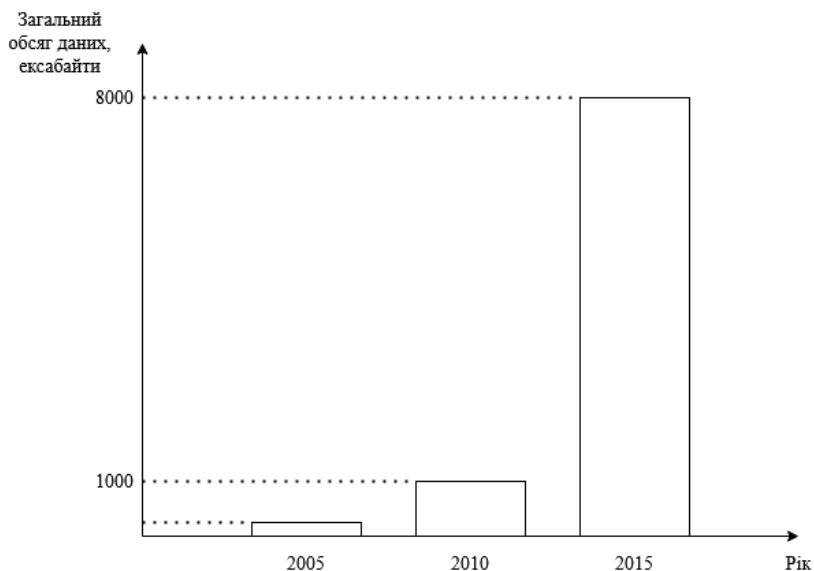


Рис 1.4. Загальний обсяг даних, оцінка отримана з дослідження [10]

Тим не менше, з великими даними виникає ряд проблем. З ростом Інтернет-послуг кількість віртуальних індексів та вмісту, що можна шукати, зростала. У відповідь на це пошуковим компаніям довелося подолати кризу великих даних. Google розробив моделі програмування GFS [10] та MapReduce [10], щоб впоратися з проблемами, пов'язаними з управлінням та аналізом даних у масштабі Інтернету. Крім того, вміст, створений користувачами, датчиками та іншими всюдисущими джерелами даних, також нагромадив величезні потоки даних, що вимагало принципових змін в обчислювальній архітектурі та широкомасштабному механізмі обробки даних. У січні 2007 року Джим Грей, піонер в напрямку обробки баз даних, назвав таку трансформацію "Четвертою парадигмою" [10]. Варто також перерахувати основні виклики, які були описані в ряді робіт [10]:

- Відображення даних: Багато наборів даних містять змінні різних типів, конфігурацій, визначень, кодів та деталізації. Візуалізація даних намагається зробити дані більш доступними та актуальними для користувача. Незважаючи на це, невідповідне представлення даних зменшить цінність вихідних даних і може навіть перешкодити ефективному аналізу даних.
- Зниження надмірності та стиснення даних: як правило, існує високий ступінь надмірності у наборах даних. Зменшення надмірності та стиснення даних ефективно для мінімізації непрямих витрат всієї системи на основі того, що це не впливає на майбутні значення даних. Наприклад, сенсорні мережі виробляють багато даних, які можна фільтрувати та стискати з високою швидкістю.
- Управління життєвим циклом даних: порівняно із достатньо повільним розвитком систем зберігання, постійне зчитування та обчислення генерують дані з неперевершеними швидкостями та у великих масштабах. Ми стикаємося з проблемою, що наявна система резервного копіювання не вміщує такі великі дані. Говорячи про потокові дані, системи не встигають їх обробляти та аналізувати. Отже, наскільки релевантні дані відносно наших цілей, слід ще визначити, щоб розуміти, які дані слід зберігати, а які – відкидати.
- Аналітичний підхід: аналітичний підхід до аналізу великих даних вимагає інтенсивної обробки протягом короткого часу. Однак звичайні СУБД побудовані з відсутністю можливостей до масштабування та розширюваності, що не може задовольнити вимоги до продуктивності. Нереляційні бази даних показали свої унікальні переваги в обробці неструктурованих даних і стали популярними в аналізі великих даних.

Але не дивлячись на це, все ще існують деякі проблеми нереляційних баз даних щодо їх продуктивності та конкретних застосувань.

- Конфіденційність даних: більшість постачальників або власників послуг з передачі великих даних в даний час можуть помітити ефективне управління та оцінку таких великих наборів даних через їх обмежені можливості. Отже, аналіз великих даних може бути наданий третім особам лише тоді, коли вживаються заходи безпеки для захисту даних та забезпечення їх захисту.
- Обчислювальне споживання енергії: кількість енергії, що використовується основними каркасними комп'ютерами, зацікавила як з точки зору сталого розвитку, так і з точки зору економіки. Зі збільшенням обсягу даних та аналітичних вимог збір, зберігання та передача великих даних з часом буде споживати все більше і більше електричної енергії. Тому для великих даних повинен бути розроблений механізм контролю та управління енергоспоживанням на рівні системи, а можливість розширення та доступність забезпечена.
- Аналітичний метод: система повинна враховувати як малі, так і великі набори даних. Аналітичний алгоритм повинен мати можливість обробляти дедалі більші та складніші набори даних [10].

Проте, поява такого напрямку як оброблення великих даних не може існувати без власне інтелектуального аналізу даних та машинного навчання. Відповідно, для обґрунтування актуальності даної дисертації необхідно розглянути й розвиток цих дисциплін. Приклади цієї тенденції до збору та видобутку великої кількості даних для поліпшення послуг та продуктивності можна знайти в багатьох сферах торгівлі, науки та державного управління [11].



Потокові текстові дані вимагають алгоритмів, що підлягають обчислюванню, високоякісні персональні дані викликають потребу в алгоритмах, що мінімізують наслідки конфіденційності, а доступність величезної кількості немаркованих даних викликає проблему розробки алгоритмів навчання, щоб скористатися ними [11].

Найбільш широко використовуваними методами машинного навчання є методи навчання «з вчителем». Такі системи можуть бути різноманітними, включаючи класифікатори спаму електронної пошти, розпізнавання облич над зображеннями та системи медичної діагностики для пацієнтів - все це ілюструє проблему наближення функцій, обговорювану раніше, де навчальні дані мають форму збору пар  $(x, y)$  та ціль полягає у створенні передбачення  $y^*$  на кожний з можливих вхідних даних  $x^*$ . Вхідні дані  $x$  можуть бути класичними векторами, або можуть бути більш комплексними об'єктами, такими як документи, зображення, послідовності ДНК або графіки.

Подібним чином було вивчено багато різних видів подібних систем. Значний прогрес був досягнутий, зосередившись на простій проблемі двійкової класифікації, в якій береться одне з двох значень (наприклад, "спам" або "не спам"), але також було багато подальші дослідження таких проблем, як багатокласова класифікація (де обирається один з багатьох можливих класів), класифікація багатознакових значень (де одночасно позначаються вхідні дані декількома класами), рейтингові проблеми (де забезпечується частковий порядок в деякому наборі) та загальні структуровані проблеми прогнозування (де комбінаторний об'єкт є графом, компоненти якого можуть знадобитися для задоволення певного набору обмежень). Прикладом останньої проблеми є позначення частини мови, де метою є одночасне

позначення кожного слова у вхідному реченні, як іменника, дієслова чи іншої частини мови [11].

Іншим напрямом машинного навчання є навчання «без вчителя», яке, як правило, включає аналіз немаркованих даних під припущеннями про структурні властивості даних (наприклад, алгебраїчні, комбінаторні чи ймовірнісні). Прикладом такого навчання може бути кластеризація – це проблема пошуку розділу спостережуваних даних (і правила для прогнозування майбутніх даних) за відсутності явних міток, що вказують на бажаний розділ.

Проте, не всі алгоритми кластеризації є адаптовані для обробки великих даних. Для прикладу, можна взяти алгоритм CURE, перевагою якого є можливість знайти кластери різних форм, не лише сферичні. Як вказано в книзі [12], цей алгоритм навіть може знаходити кластери, в середині яких знаходяться інші. На початковому етапі дослідження було детальніше розглянуто задачу кластеризації та, враховуючи відсутність адаптації даного алгоритму під обробку великих масивів даних, було запропоновано його модифікацію. Дана модифікація полягає в розбитті основних кроків на дві частини – Map та Reduce, де кожне Map-завдання може виконуватись в окремих процесах на різних обчислювальних вузлах, а Reduce – буде виконано в головному процесі. Модифікована версія алгоритму була протестована, а результати опубліковані в тезах до конференції [13].

Третім основним напрямом машинного навчання є навчання «з підкріпленням». Базова інформація про цей напрям вже була озвучена в попередніх розділах роботи. Замість навчальних прикладів, які вказують на правильний результат для даного вхідного набору, дані підготовки в навчанні,

як передбачається, надають лише вказівку на те, чи правильна їх дія чи ні; якщо дія неправильна, залишається проблема пошуку правильної дії. Більш загально, при встановленні послідовностей входів передбачається, що символи перенаправлення стосуються всієї послідовності; приписування кредиту або вини окремим діям у цьому підсумку безпосередньо не передбачено [11].

Одним із найефективніших напрямків прогресу в навчанні «з вчителем» за останні роки є використання нейронних мереж та так зване глибоке навчання. В основі цього напрямку лежать багатошарові мережі, що складаються з нейронів, кожен з яких, в свою чергу, обчислює деяку просту параметризовану функцію своїх входів [11].

Такі системи використовують алгоритми оптимізації на основі градієнта для регулювання параметрів за допомогою виходу багатошарової мережі, що базується на виводі помилок. Використовуючи сучасні архітектури паралельних обчислень, такі як модулі обробки графіки, спочатку розроблені для відеоігор, вдалося створити навчальні системи, що містять мільярди параметрів, і які можна навчити на дуже великих колекціях зображень, відео та мовних зразків, доступних в мережі інтернет.

Такі широкомасштабні системи глибинного навчання мали значний ефект протягом останніх років у комп'ютерному зорі та розпізнаванні мови [11], де вони дали значні покращення продуктивності порівняно з попередніми підходами. Методи глибинних мереж активно застосовуються в ряді додатків від перекладу природньої мови до фільтрації даних. Хоча більша частина практичного успіху в глибокому навчанні походить від контрольованих методів навчання для виявлення таких репрезентацій, також докладаються

зусилля для розробки алгоритмів глибокого навчання, які виявляють корисні презентації вхідних даних без необхідності маркованих навчальних даних [11].

Як було вказано раніше, за останні роки було розроблено багато мультиагентних систем, що базуються саме на методах навчання «з підкріпленням» та глибинного навчання, так звані Deep Q-Networks, або глибокі мережі, що базуються на Q-навчанні. Тому для більш детального дослідження даного напрямку було проведено ряд теоретичних та практичних експериментів, один з яких опублікований в роботі [14]. На підставі результатів експерименту можна зробити висновок, що побудова мультиагентної системи на основі глибокого навчання з підкріпленням дозволяє вирішити проблеми в складних умовах з великою кількістю станів. Однак це завдання є досить складним для класичних методів базового Q-навчання: DQN, DDQN, TD, PPO, через ряд причин. Дуже складне середовище, наприклад, для вирішення задачі sudoku, базується на непростих неявних правилах. Знайти ці залежності занадто складно для методів Q-навчання з використанням глибоких нейронних мереж. Обмеження Q-навчання залежать від обмежень сучасних комп'ютерів, а не від самого методу. Використання методу, заснованого на MCTS, показує хороші результати, але вивчення такої мережі є досить трудомістким завданням, яке збільшується із збільшенням розміру середовища та складності завдання.

Враховуючи проаналізовану інформацію, важливість та актуальність певних напрямків дослідження, а також – проведені експерименти, було вирішено визначити чіткі межі для основної задачі, що розкривається в даній науковій праці. Завдання зводиться до автоматичної побудови словника предметних областей в середовищі, до якого постійно надходять нові і нові документи (тексти), відповідно, ці тексти необхідно швидко класифікувати, а

далі – побудувати словники предметних областей, і після цього тексти будуть втрачені. Тому швидкість класифікації та побудови словників є важливим критерієм для оптимізації в даній роботі, бо чим швидше дані будуть проаналізовані, тим швидше можна буде отримати на обробку наступний блок даних, і, відповідно, ймовірність втрати важливої інформації буде зменшена. Відповідно, під «великими даними» в даній роботі матиметься на увазі саме їх частина, яка має швидко бути оброблена, інакше – буде втрачена. Цей напрям досліджень є окремим від оброблення великих масивів даних, що зберігаються постійно, але на великій кількості обчислювальних вузлів.

В подальших розділах сформульована дана задача більш детально, проаналізовані деякі з існуючих рішень, а також – описані результати модифікацій методів, які дозволяють вирішити дану задачу.

## **1.2. Формулювання задачі швидкої обробки слабоструктурованих текстових потоків даних.**

Як було зазначено раніше, задачу швидкої обробки текстів в даній роботі зводиться до наступної: в систему надходить великий набір текстових даних, які не зберігаються в базі даних, а мають бути оброблені одразу. Іншими словами – якщо не обробити щойно отриманий текст, то він буде загублений назавжди. Як результат обробки даних має бути отриманий словник певної предметної області. В загальному така система зображена на Рис. 1.5.

Кожен з екземплярів вхідних даних представляє собою текст, що відноситься до однієї з предметних областей. Наприклад, це може бути текст з фізики, хімії, тощо. Проте, важливо зазначити, що завчасно невідомо, до якої саме області належить конкретний текст, тому система на одному з етапів

обробки має це визначити та передати текст на наступний етап обробки – для вдосконалення словника конкретної області.

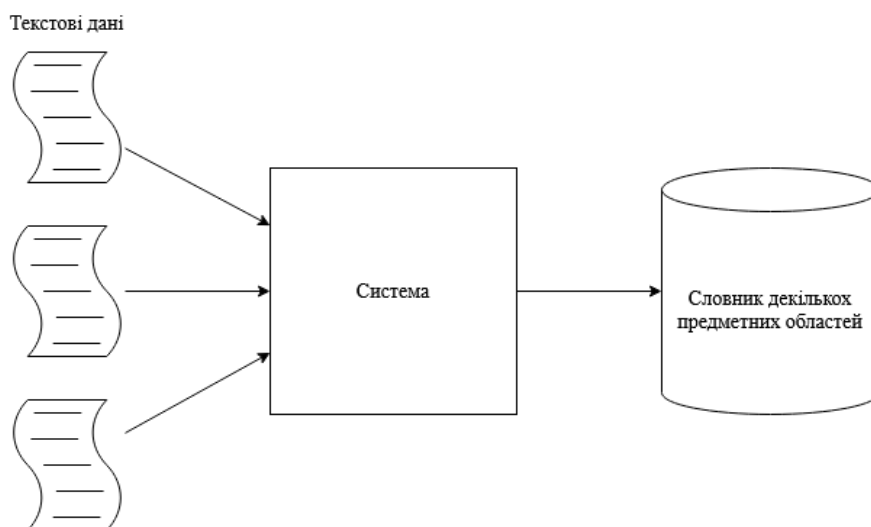


Рис 1.5. Формулювання задачі [власна розробка]

Тексти можуть надходити у довільні моменти часу та навіть одночасно, тому варто передбачити в системі можливість обробки одночасно декількох вхідних даних у різних процесах.

У випадку, якщо неможливо буде розпізнати, до якої саме предметної області належить текст – він має бути відповідним чином позначений та переданий в окреме сховище для подальшого вдосконалення методів класифікації.

Відповідно, дана задача має бути розділена на три незалежні етапи досліджень: побудова архітектури системи, дослідження та розробка методів швидкої класифікації текстів, дослідження та розробка методів автоматичної побудови словника предметної області. Це зображено на Рис. 1.6.

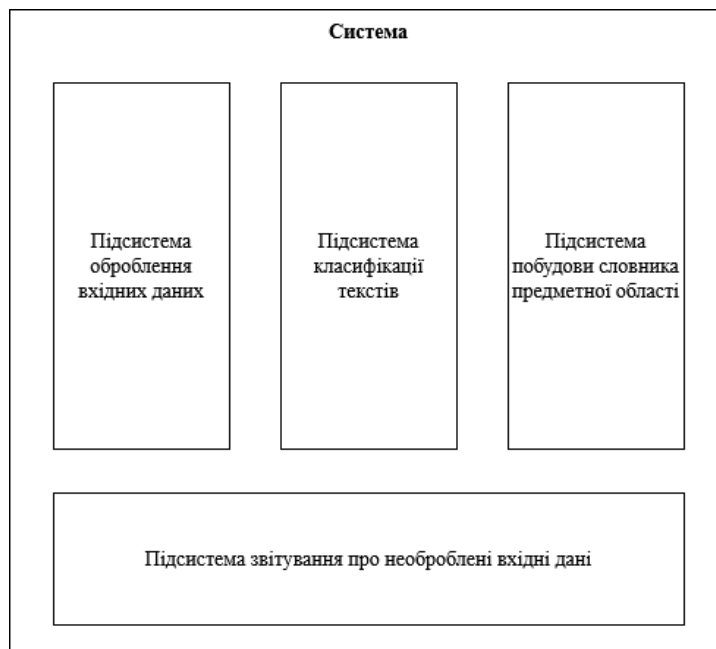


Рис 1.6. Підсистеми, які необхідно розробити [власна розробка]

### 1.2.1. Задача побудови архітектури системи.

Оскільки мова йде про великі масиви даних, тому необхідно розробити загальну архітектуру системи, що передбачатиме в собі вирішення ряду наступних проблем: швидкість обробки даних має бути допустимо низькою, система має бути відмовостійкою, система може бути масштабованою у випадку збільшення обсягів вхідної інформації, система має бути розподіленою, а не монолітною.

Для вирішення даної задачі підходить мультиагентна система, в якій кожен інтелектуальний агент має своє власне завдання та вміє «спілкуватись» з іншими агентами з використанням визначеного протоколу. Тому треба розробити не лише архітектуру самої МАС, а й додати до неї протоколи спілкування між агентами, які побудовані на стандартизованому підході, наприклад, з використанням протоколу FIPA-KIF – розробленого організацією з дослідження інтелектуальних фізичних агентів.

### **1.2.2. Задача дослідження та розробки методів швидкої класифікації текстів.**

На вхід системи надходить велика кількість текстів, кожен з яких представляє собою окремий текст певної (заздалегідь невідомої) предметної області. Відповідно, необхідно визначити як саме зробити попередню обробку даних, дослідити та вдосконалити методи класифікації текстів, зробити оцінку швидкості класифікації текстів, а також – її якість.

Важливо відмітити, що можуть траплятись тексти, які не будуть класифіковані на достатньому рівні для подальшої обробки. Тому в дослідженні варто визначити певне абсолютне значення оцінки належності до класу, яке буде використане для відсіювання текстів, в яких система не буде впевнена.

Для класифікації текстів необхідно дослідити в якому саме форматі текст може бути представлений, які існуючі моделі вже існують та яким чином вони можуть бути використані, необхідно також визначити та врахувати обмеження до вхідних даних, можливих класів текстів, тощо. В даній підсистемі варто додати й другий етап класифікації іншим методом, що дозволить прийняти правильне рішення стосовно належності того, чи іншого тексту до певного класу. У випадку, якщо жоден з методів не дасть достатніх результатів, тоді варто позначити даний текст як «некласифікований» та надіслати його експертам для подальшого вдосконалення системи класифікації.

### **1.2.3. Задача дослідження та розробки методів автоматичної побудови словника предметної області.**

Словник предметної області представляє собою набір базових термінів, які визначають дану предметну область. В даній задачі необхідно дослідити



існуючі методи, їх можливості до оброблення потоків даних, а також – їх обмеження. Словник не може бути нескінченно великим, тому при вирішенні даної задачі варто оцінити оптимальний розмір словника, або закласти можливість задання (у вигляді константи) даного розміру. У випадку, якщо немає впевненості в результаті, має бути передбачений механізм з альтернативного оброблення даного тексту для побудови словника, можливо, іншим методом, або аналогічно до попередньої задачі – визначення даного тексту як «необробленого».

Оскільки передбачається, що дані надходять у вигляді потоків, і, відповідно, інтелектуальні агенти мають працювати в різних процесах одночасно, тому будуть випадки, коли по кожному тексту базовий словник предметної області відрізнятиметься від паралельно отриманих. Тому як вирішення даної проблеми має існувати механізм голосування за більш важливі слова, які й будуть додані до кінцевої версії словника на даний момент часу. Після оброблення текстів отриманий словник має зберігатись в базі даних, а самі тексти – мають бути видалені, як передбачається постановкою задачі.

### **1.3. Аналіз існуючих рішень.**

Перш ніж почати опис результатів дослідження, необхідно розглянути існуючі рішення для розв'язку поставленої задачі. Ці рішення варто проаналізувати на предмет обмежень, переваг та недоліків. Враховуючи отримані результати варто більш конкретно сформулювати задачу, визначивши слабкі місця для покращень, а також – вказати на нові (досі неіснуючі) підходи, що мають бути запропоновані в подальшому.

Оскільки, предмет дослідження лежить на перетині декількох наукових напрямків, а саме – мультиагентні системи, оброблення великих масивів даних та машинне навчання, то варто дослідити існуючі роботи саме по даним ключовим словам.

### **1.3.1. Мультиагентні системи для аналізу текстових даних.**

Достатньо схожою роботою до даного напрямку досліджень є робота [15], суть якої полягає в розробці мультиагентної системи для обробки медичних текстових даних з використанням навчання з підкріпленням. Автори також наголошують про істотне зростання текстової інформації в світі, але зосереджуються саме на медичних текстових даних. Метою роботи є аналіз наукових праць на рівень сентиментності цих робіт. Програмні агенти колективно вивчають настрої, що стосуються конкретних ключових слів, оскільки кожен агент обробляє призначену їм підмножину даних. Авторами було проведене експериментальне дослідження, щоб довести роботу системи. Серед переваг дослідження варто відзначити поєднання такого методу як Q-навчання з використанням його в мультиагентних системах, проте, авторами не вказується який саме програмний фреймворк для реалізації мультиагентних систем був використаний, а також – великим недоліком є відсутність застосування стандартного протоколу (мови) спілкування між інтелектуальними агентами, що дозволило б в подальшому поєднувати дану розроблену систему з іншими.

В іншій роботі, що була опублікована в 2016-му році [16] описується мультиагентна система, яка аналогічно була розроблена для аналізу текстових даних на предмет сентиментності. Серед методів, що були розглянуті, варто відзначити TF та IDF для розрахунку частот виявлення певних слів в заданих

текстах. Незважаючи на переваги від розпаралелювання, цей мультиагентний підхід є дуже масштабним. Це необхідно, оскільки кількість інформації, яку необхідно обробляти, збільшується. Алгоритми отримання інформації пропонується будувати в розподіленому середовищі, що включає кілька сховищ текстових документів. Універсальний дизайн представленої системи дозволяє проводити модифікації під різні вимоги. Високий ступінь модульності пропонованих рішень робить їх дуже простими. Агентів також можна запрограмувати на автоматичне тестування своїх результатів, щоб їх можна було перекваліфікувати після того, як вони стали менш успішними. Відповідно, перераховані вище пункти є перевагами даного дослідження.

Серед розглянутих робіт достатньо важливим є дослідження про побудову динамічних онтологій 2007-го року [17]. Розроблену систему автори називають Dynamo. Dynamo націлене на зменшення потреби в ручних діях при обробці результатів текстового аналізу та на пропонування концепції мережевої роботи з метою найефективнішої побудови онтологій. Будучи представленим як перспективне рішення, що забезпечує якість моделі та їх термінологічне багатство, побудова онтології на основі аналізу текстового корпусу є складним та дорогим завданням. Це вимагає аналітичного контролю та врахування мети онтології. Використання засобів обробки природних мов полегшує контексти локалізації знань завдяки використанню мови. Тим не менш, ці інструменти виробляють величезну кількість лексичних або граматичних даних, які не є тривіальними для перевірки, щоб визначити понятійні елементи. Їх внесок на цьому етапі процесу полягає в моделюванні з текстів перед будь-якими спробами нормалізувати або формалізувати результат.

З точки зору побудови онтології, ця робота є першим кроком демонструючим актуальність обраного підходу. Також з точки зору мультиагентної системи, їх використання в динамічному контексті онтології показало свою актуальність. Цю динамічну онтологію можна розглядати як складне вирішення проблем, у такому випадку самоорганізація шляхом співпраці була ефективним рішенням. І в цілому це, мабуть, буде цікаво для інших завдань, пов'язаних з проєктуванням, навіть якщо дана робота була зосереджена на конкретній області знань. Це означає, що подібний підхід можна використати в інших дослідженнях.

### **1.3.2. Оброблення великих даних з використанням мультиагентних систем.**

Однією з найбільш значущих робіт є робота [18], в якій описується архітектура мультиагентної системи для оброблення великих даних в режимі реального часу. У статті описана архітектура обробки великих даних у режимі реального часу на основі парадигми мультиагентних систем. Представлений загальний підхід до обробки даних в режимі реального часу та даних зі сховищ. Серед переваг даної роботи варто зазначити не лише саму запропоновану архітектуру системи, а й вказання, які з сучасних програмних фреймворків можуть бути складовою частиною цієї системи. Як приклад автори наводять на рівні пакетів даних використання Hadoop, а саме – HDFS (розподілена файлова система) для зберігання даних, а систему YARN – для обробки великих даних. Проте, аналогічно, як і було вказано раніше, в роботі не описуються протоколи для комунікації агентів, що є проблемою для інтеграції цього підходу з іншими існуючими.

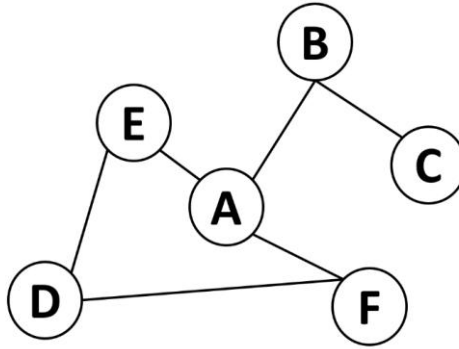


Рис 1.7. Приклад можливої адаптивної МАС для детектування відношень [18]

Наприклад, коли агент В (Рис. 1.7) взаємодіє з обома своїми сусідами А і С, він буде шукати нових агентів, щоб розмістити їх у своєму районі. Однак, коли новий агент-сусід, наприклад, Е, вже знаходиться в районі одного з своїх власних сусідів (А), тоді В збільшить впевненість у своєму відношенні з А, якщо Е і В пов'язані, інакше В зменшить цю впевненість. Також, відношення між В і Е може еволюціонувати з часом внаслідок змін даних, а потім ці зміни будуть поширюватися на відношення АВ та відношення АЕ. Це може призвести до того, що деякі відносини коливатимуться між посиленням та послабленням, що може бути виявлено, як це впливає зі складного багатофункціонального відношення. В результаті цього поширення та оновлення відносин уся система адаптується до динамічного, прогресивного та органічного шляху.

Серед переваг даного дослідження варто зазначити орієнтацію на обробку даних в режимі потокової обробки, пояснення зв'язку МАС з системами Nadoor, а також – можливість новим агентам приєднуватись до системи, її масштабованість. Проте, в роботі не вказані жодні числові характеристики роботи такої системи, а також не вказано, який саме протокол

комунікації використовувався. У будь-якому випадку, ідеї про автоматичну можливість розширення системи є важливими для сучасних реалій через постійне зростання кількості даних, які необхідно обробити.

### **1.3.3. Рішення для побудови спеціалізованих словників предметних областей.**

Проблема побудови словників є актуальною та досліджується в наступних роботах [50][51]. В роботі [50] для автоматизованого створення словників предметної області було розроблено підхід, що враховує особливості коротких документів та їх структуру. Запропоновано модель подання документа у вигляді трьох частин: заголовка, основного змісту та фінальної частини, де останні дві можуть містити інформацію, не пов'язану з предметною областю. Для ефективного вилучення термінів було розроблено метод виокремлення змістовної частини документа на основі множини ключових слів. Оскільки короткі документи не дозволяють визначити частотні характеристики слів для виявлення багатослівних термінів, запропоновано метод кластеризації, що базується на виділенні іменників та аналізі їхньої частотності. Це дало змогу обробляти кластери як звичайні документи, що дозволяє ефективно виявляти складені терміни. Крім того, для автоматичного визначення значень термінів використано попередньо розроблений метод пошуку інтерпретацій у словниках. На основі запропонованої моделі було створено програмне забезпечення для побудови словників, яке успішно пройшло експериментальну перевірку та рекомендоване для практичного використання в інформаційних системах [50].

В іншій роботі [51] вказано, що у сфері демонтажу електромобілів досі не існує спеціалізованого словника, а наявні алгоритми побудови словників не забезпечують точного вилучення термінології через її складність і

варіативність. У цьому дослідженні запропоновано алгоритм побудови словника предметної області на основі навчання з учителем, що використовує багатовимірні ознаки для екстракції термінів із текстів наукових досліджень. Розпізнавання ключових слів представлено як задачу бінарної класифікації, де модель LightGBM використовується для фільтрації кандидатів, після чого словник розширюється на основі точкової взаємної інформації між словами та їхньою категорією. Для створення корпусу було зібрано китайські керівництва з демонтажу, патенти та наукові статті, на основі яких система автоматично визначає ключові терміни, що стосуються деталей, інструментів, методів та процесів демонтажу. Експериментальні результати показали, що запропонований підхід суттєво перевищує ефективність традиційних алгоритмів у сфері демонтажу електромобілів. Це дослідження встановлює новий стандарт у побудові предметних словників для галузі, спираючись на новостворений набір даних і багатокласову термінологічну класифікацію [51].

Дані дослідження вказують на актуальність такого напрямку наукової діяльності, як створення спеціалізованих словників, а також – на те, що існують сфери, де цих словників наразі ще не існує.

#### **1.3.4. Узагальнення результатів та подальші кроки дослідження**

Провівши аналіз наукових праць було отримано ряд напрямків для подальших досліджень. Що стосується розвитку мультиагентних систем, то на сьогоднішній день дійсно існують спроби дані системи використовувати для оброблення великих масивів даних, особливо це робиться через можливості до масштабування, які лежать в основі МАС. Щодо основних проблем – це відсутність в роботах інформації про використання стандартизованих мов для комунікації агентів, що, в свою чергу, дозволило б не лише використовувати

МАС для рішення конкретної задачі, але й робити інтеграцію таких систем в майбутньому. Також, в багатьох працях про мультиагентні системи та їх поєднанні з нейронними мережами використовується інформація про алгоритми навчання з підкріпленням, і майже немає інформації про дослідження інших видів нейронних мереж.

Якщо зосередитись на обробці саме текстових даних, то в роботі про різноманіття технік машинного навчання при класифікації текстів [19] можна побачити велику кількість задач, які стоять перед дослідниками. Ще автори наводять огляд наукових праць з 1998 по 2014 роки та вказують основні цілі та результати, що були отримані. Найкращий результат класифікації був отриманий з використанням методу SVM, але серед недоліків варто зазначити відсутність порівняння по часу *класифікації, що є актуальним для задачі даної дисертації*.

Були спроби розробки універсальних архітектур мультиагентних систем для використання при обробці великих даних, деякі з них навіть отримали практичне застосування. Проте, не всі з описаних архітектур мають можливість обробляти дані швидко, і тому цей напрям також потрібно розвивати. Стосовно обробки великих текстових даних, то в досліджених роботах по класифікації текстів мало увага привертається часу класифікації, що є надважливим критерієм при класифікації в режимі потокової обробки. Також, не було знайдено досліджень про задачу автоматичної побудови словника предметної області з використанням на великих текстах.

Варто відзначити, що не всі з розглянутих робіт були описані в даному підрозділі, це пов'язано з тим, що деякі інші наукові праці, а точніше – їх результати, описані та використовуються в подальших розділах дисертації.



#### **1.4. Висновки.**

В даному розділі основна увага приверталась аргументації актуальності роботи та визначеного напрямку досліджень. Основними висновками є:

- Проаналізовано розвиток декількох напрямків, а саме – мультиагентних систем, оброблення великих масивів даних та методів машинного навчання та вказані причини актуальності та важливості досліджень в цих напрямках, серед яких основними є збільшення обсягів інформації та зростання обчислювальних потужностей.
- Проаналізовані наукові праці з вищезазначених напрямків та сформульовані недоліки, проблеми та невирішені задачі, які є актуальними на сьогоднішній день.
- На початковому етапі досліджень при вивченні даних напрямків проведені експерименти (результати викладені у вигляді публікації у фаховому виданні), перший з яких стосувався аналізу існуючих методів обробки поточкових даних. Також, другою складовою предметом досліджень є мультиагентні системи, які побудовані з використанням алгоритмів глибинного навчання з підкріпленням. Визначено, що дана задача актуальна у випадках, коли середовище в змозі надавати зворотній зв'язок агентам у вигляді штрафних балів, але не кожна задача може бути сформульована саме таким чином. У будь-якому випадку, на прикладі вирішення однієї з таких задач доведено працездатність МАС. Результати досліджень були викладені в тезах конференції та науковому фаховому журналі відповідно.

- Проведено дослідження по побудові методологічних основ визначення функціональності моделі мультиагентної системи. Вказані основні проблеми організації програмних систем з використанням мультиагентної системи при аналізі даних. Запропоновано модель мультиагентної системи для аналізу великих слабоструктурованих текстових масивів даних.
- Як результат всього вищезазначеного сформульовано чітку задачу, яку необхідно вирішити в даній дисертації. Задача стосується оброблення великих масивів поточкових текстових даних, які неможливо зберегти та потрібно швидко обробити задля автоматичного отримання словників декількох предметних областей.

## **РОЗДІЛ 2 МОДЕЛЬ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ ПОТОКІВ ТЕКСТОВИХ ДАНИХ.**

Метою даного розділу є розробити модель мультиагентної системи, яка буде здатна вирішити поставлену задачу автоматичної класифікації вхідних текстів та подальшої побудови словника предметної області при умові, що нові вхідні дані надходять постійно і у великій кількості.

В розділі 2.1 буде розглянуті практичні аспекти побудови і розгортання мультиагентних моделей як засобу для розподілених обчислень, але з врахуванням задачі обробки потоків текстових даних в системі, до якої ці тексти неперервно надходять. Зазначені можливі слабкі місця, які необхідно врахувати при проектуванні. В цьому розділі описані загальні підходи до проектування мультиагентних систем.

Прикладне застосування моделі МАС на конкретній задачі оброблення текстових даних описане в розділі 2.2. Увага зосереджена на стандартах FIPA, і зокрема на мові ACL, також розглядається розміщення агентів на окремих обчислювальних вузлах (розгортання системи). Також, в даному розділі описано мову комунікації, які саме команди можуть надсилати агенти одне одному. Для кожного з агентів вказані та описані його можливі стани та мова його спілкування.

Варто зазначити, що для програмної реалізації даної системи необхідно проаналізувати існуючі програмні бібліотеки та фреймворки для створення моделі МАС. Врахувати їх обмеження, відкритість до розширення, бо може трапитись так, що на якомусь етапі написання програмного коду, існуючого функціоналу буде недостатньо, тощо. Це питання розкрито в розділі 2.3.

В розділі 2.4 буде описана мультиагентна підсистема (враховуючи, що загальна система описана в розділі 2), також запропоновані формати запитів на мові ACL, буде описана робота кожного з агентів.

В розділі 2.5 розглядаються загальні підходи для прийняття рішень в мультиагентних системах та суміжних областях, а в результаті – запропонований підхід для даної задачі, який дозволяє голосувати агентам та приймати рішення щодо кінцевого вигляду словника.

### **2.1. Практичні аспекти побудови мультиагентних систем у розподіленому середовищі.**

Мультиагентні системи за своєю природою є розподіленими системами через автономність кожного з агентів та можливість їх знаходження на різних обчислювальних вузлах. Для початку необхідно визначити, що мати на увазі під «розподіленою системою», для цього варто звернутись до книги [20].

Проте, важливо розуміти, які саме існують можливості та обмеження в мультиагентних системах для виконання розподілених обчислень. При виконанні розподілених обчислень важливим критерієм є час обробки даних, тому на першому етапі необхідно розглянути будову МАС та можливі обмеження щодо швидкості обчислень.

Розглянемо загальну схему роботи мультиагентної системи. В нас може бути три різних випадки – МАС працює на комп'ютері (обчислювальному вузлі), МАС працює на декількох комп'ютерах, які поєднані в одну мережу (наприклад, комп'ютерний кластер), або ж МАС може працювати на суперкомп'ютері, в якому вузли з'єднані високошвидкісною шиною даних. При чому, в першому випадку кожен з агентів може бути окремим програмним

процесом, а також – окремим програмним потоком, кожен з двох варіантів має свої переваги та недоліки.

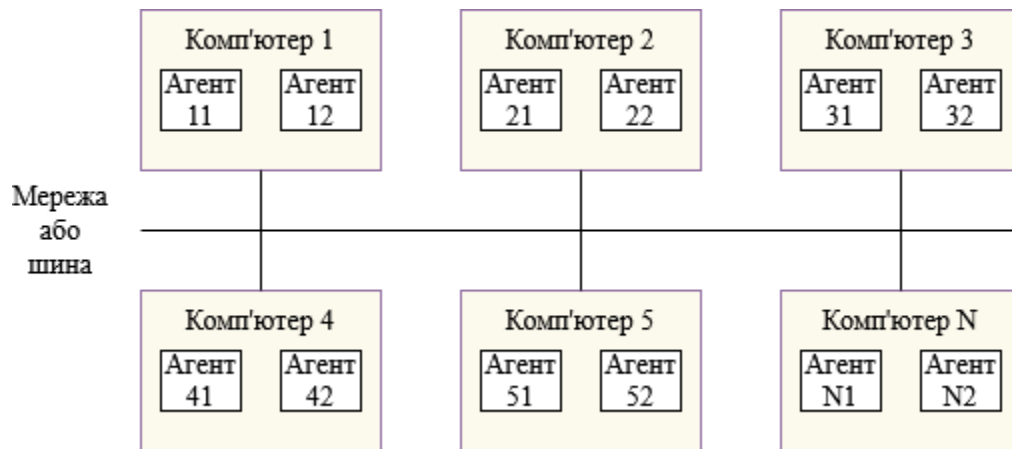


Рис 2.1. Розподілена мультиагентна система [власна розробка]

У випадку створення розподіленої мультиагентної системи основною перевагою є можливість оброблення даних великих об'ємів, заповнивши пам'ять кожного з комп'ютерів, і робити це – на різних процесорах. Очевидно, що в такому випадку створення загальний об'єм даних, можливих для обробки, буде більшим, ніж при обробці на одному комп'ютері. Аналогічна ситуація складається й з розгортанням МАС на суперкомп'ютері. Проте, у випадку розподілу даних в кластері, з'являється фактор затримки передачі інформації між різними вузлами. Таким фактором затримки є мережеве з'єднання. Відповідно, якщо агент 11 (згідно з Рис. 2.1) захоче передати інформацію агенту 21, то в такому випадку затримка буде довшою більшою, ніж при передачі інформації агенту 12.

Перевага суперкомп'ютерів полягає в тому, що, оскільки дані можуть швидко переміщуватися між процесорами, всі процесори можуть працювати разом над одними і тими ж завданнями. Суперкомп'ютери підходять для дуже складних додатків і моделювання в режимі реального часу, для обробки

потоків даних. Однак суперкомп'ютери дуже дорогі у побудові та обслуговуванні, оскільки вони складаються з великого набору найсучасніших процесорів, швидкої пам'яті, спеціального обладнання та дорогих систем охолодження. Вони також погано масштабуються, оскільки їх складність ускладнює легке додавання більшої кількості процесорів до такої точно розробленої та тонко налаштованої системи.

На відміну від цього, перевага розподілених систем полягає в тому, що в порівнянні з суперкомп'ютерами вони набагато дешевші. У багатьох розподілених системах використовуються дешеві, готові комп'ютери для процесорів та пам'яті, які вимагають лише мінімальних витрат на охолодження. Крім того, їх простіше масштабувати, оскільки додавання додаткового процесора до системи часто складається лише з підключення її до мережі. Однак, на відміну від суперкомп'ютерів, які надсилають дані на невеликі відстані через складні та оптимізовані з'єднання, розподілені системи повинні переміщувати дані від процесора до процесора через повільніші мережі, що робить їх непридатними для багатьох додатків у режимі реального часу [21].

Що стосується розгортання МАС на одному комп'ютері, як було вказано раніше, це може бути зроблено двома способами. Перший, розгортання в різних процесах – вказаний на рис. 2.2., другий – розгортання в різних потоках – вказаний на рис. 2.3. Важливо зазначити, що потік може робити все, що може процес. Але оскільки процес може складатися з декількох потоків, потік можна вважати «легким» процесом. Таким чином, суттєвою відмінністю потоку від процесу є робота, для виконання якої використовується кожен з них. Потоки

використовуються для невеликих завдань, тоді як процеси використовуються для більш важких завдань – в основному виконання програм.

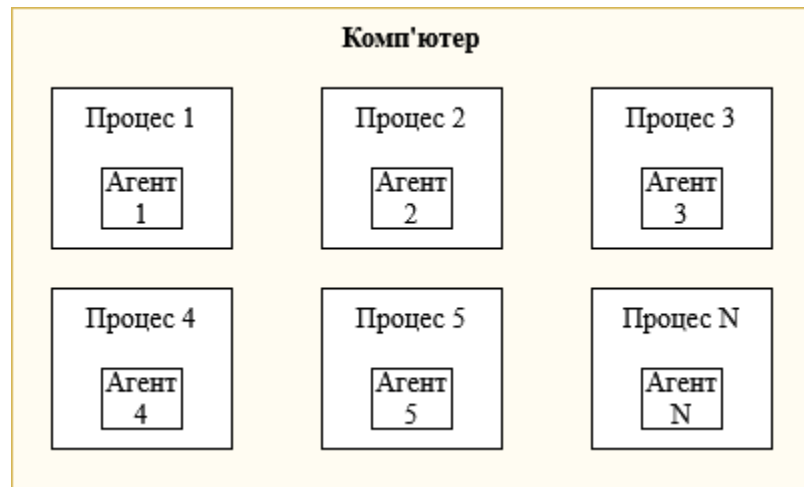


Рис 2.2. Діаграма розгортання МАС в різних процесах [власна розробка]

Інша відмінність між потоком і процесом полягає в тому, що потоки в рамках одного процесу мають однаковий адресний простір (також називається віртуальний адресний простір), тоді як різні процеси – ні. Це дозволяє потокам читати і записувати в однакові структури даних та змінні, а також полегшує зв'язок між потоками. Зв'язок між процесами - також відомий як IPC, або міжпроцесовий зв'язок – досить складний та ресурсомісткий.

З іншого боку, у випадку розміщення МАС на одному комп'ютері, існує обмеження не лише на пам'ять, але й на ресурси процесора. Відповідно, якщо необхідно в один момент часу виконувати кількість операцій, які значно перевищують можливості процесора, навіть з урахуванням великої кількості ядер, то потрібно використовувати комп'ютерний кластер або суперкомп'ютер.

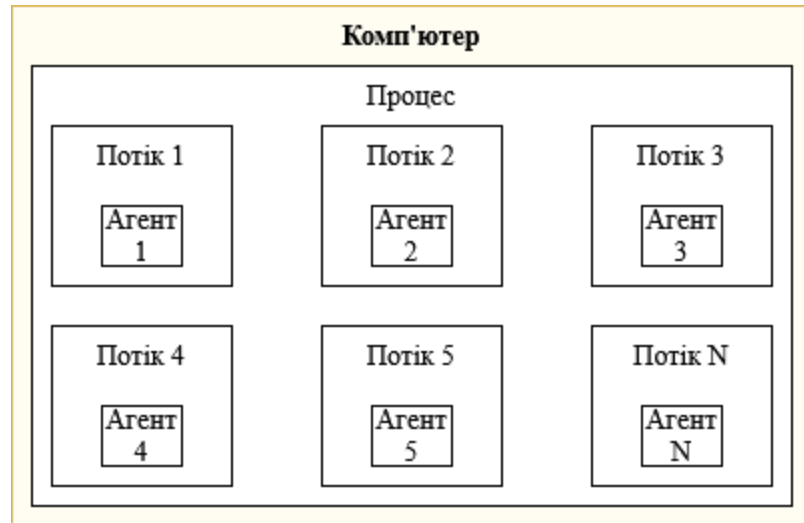


Рис 2.3. Діаграма розгортання МАС в різних потоках одного процесу [власна розробка]

Вищезазначений огляд дозволяє описати конкретні можливості для вдосконалення швидкості роботи системи, елементи системи, які можна оптимізувати. Наприклад, використовуючи змішаний підхід – поєднання розподілених обчислень з розпаралелюванням роботи в межах одного комп'ютера – можна досягти додаткових покращень показників швидкості обробки даних. Або у випадку, коли багатопроцесна система має додаткові затримки, можна перейти до багатопоточного варіанту.

При побудові власної моделі мультиагентної системи варто звернути увагу на роботу [22], в якій при дослідженні були визначені ряд параметрів навантаження кожного з агентів. В цьому експерименті були визначені наступні параметри:

- *Обчислювальне навантаження агента.* Воно представляє рівень робочого навантаження обчислень у кожного агента. Це робоче навантаження генерується комбінацією циклів та розрахунків. При вимірюванні оптимального значення цього параметра можна взяти



приклад з цієї ж роботи – обсяг робочого навантаження контролюється випадковим числом.

- *Розмір повідомлення.* Це розмір повідомлень, переданих під час кожного процесу спілкування. Під час експерименту він також може визначатись випадковим числом, як у випадку з обчислювальним навантаженням.
- *Тривалість передачі повідомлень.* Це тривалість, що минула між двома обмінами повідомленнями. Він визначає, як часто здійснюється зв'язок між двома робочими агентами. На підставі попередніх спостережень за поведінкою агентів (в дослідженні використовувались агенти, які виконують обчислення фінансового типу) у мережі Ethernet, тривалість проміжних повідомлень в даній роботі обмежили 10-ма секундами для всіх робочих агентів в системі. Проте, такий час буде достатньо довгим при обробці поточкових даних.
- *Кореляція між обчисленнями та комунікацією.* Зберігаючи послідовну поведінку агента, в дослідженні співвідносяться обчислення та зв'язок агента, наприклад, агент з більш високим рівнем обчислювального навантаження, при комунікації містить повідомлення великого розміру, і навпаки. Це досягається в системі шляхом використання того самого випадкового числа для генерації робочого навантаження та розміру повідомлення.

При проєктуванні програмних систем та при побудові їх архітектур використовується таке поняття, як точка зору (англ. – viewpoint). В цілому, це поняття є достатньо інтуїтивним, але важливим є чітке виділення точок зору, з яких необхідно проаналізувати отриману архітектуру. Моделювання МАС слід розглядати з різних взаємодоповнюючих точок зору, щоб мати справу з його складністю. Одна з перших пропозицій, методологія ААП/BDI [22]

розглядає дві точки зору: зовнішню та внутрішню. Зовнішня точка зору розглядає агентів як складні об'єкти (з власними цілями, відповідальністю, послугами та інформацією) та зовнішні взаємодії, що узгоджується з класичним уявленням про агентів як автономних об'єктах, що взаємодіють із своїм оточенням.

Внутрішня точка зору враховує елементи, необхідні для конкретної архітектури агента, наприклад, набір переконань, цілей та планів. Ця методологія керується розробкою та уточненням моделей для кожного виду: спочатку розглядається зовнішня точка зору; потім, внутрішня точка зору; пізніше зовнішні моделі отримують зворотний зв'язок, і процес продовжується, поки не буде отримано достатньо деталей реалізації. Мета зовнішньої точки зору полягає у визначенні ієрархії класу агента (модель агента) та набору взаємозв'язків між агентами (модель взаємодії). Вони будуються в чотири етапи, а саме:

- Визначення ролей у домені програми.
- Для кожної ролі визначення пов'язаних з нею обов'язків, а також послуг, що надаються та використовуються для виконання цих обов'язків.
- Для кожної послуги визначається взаємодія, пов'язана з наданням послуги. Це дозволяє визначити контрольні відносини між агентами.
- Уточнюється ієрархія агента, наприклад, рефакторинг, композиція та агрегація. Цей процес призводить до призначення функціоналу (послуг) агентам та об'єднань (взаємозв'язків послуг та взаємодії) між ними.

Методологія розробки цих моделей починається з урахування послуг, що надаються агентом, та пов'язаних з ними подій та взаємодій. Вони визначають цілі, а аналіз полягає в розбитті їх на підцілі, що призводить до

виявлення планів. Це узагальнено у два етапи. Першим є аналіз засобів досягнення цілей. Це складається з розбиття цілі на підцілі та дії для різних контекстів, в яких має бути досягнута мета. Цей процес застосовується неодноразово до підцілей. Другим етапом є побудова переконання системи, аналізуючи контекст та умови, які контролюють виконання діяльності. Також, в даній роботі [23] були визначені декілька точок зору, які варто врахувати при проєктуванні:

- *Точка зору агента*, яка описує обов'язки агента щодо завдань та ролей. Вона також включає в себе контроль агента та визначає його цілі та тематичні стани, необхідні під час виконання.
- *Точка зору організації*, яка визначає архітектуру системи. Структурні відносини не обмежуються ієрархіями між ролями. Ці структури делегуються спеціалізованим елементам, які називаються групами. В моделі організації є також відносини влади між групами, організаціями та агентами. Функціональність організації виражається за допомогою робочих процесів, які демонструють асоціації споживачів/виробників між завданнями, а також – розподіл відповідальності за їх виконання та ресурси, пов'язані з кожним із них.
- *Точка зору на середовище*, яка визначає сенсори та ефектори агентів. Вона також визначає наявні ресурси, а також уже існуючі агенти та додатки, наприклад, застарілі системи, бази даних, веб-інформаційні системи.
- *Точка зору на завдання та цілі*, на яку сильно впливає модель BDI та принцип раціональності Ньюелла. Основна мета - обґрунтувати виконання завдань з точки зору цілей. Вона також забезпечує розподіл завдань і цілей. Що стосується першого, існують спеціалізовані

взаємозв'язки, що деталізують, яка інформація потрібна, щоб вважати мету вирішеною чи невдалою. Нарешті, ця точка зору також надає детальну інформацію про завдання в системі на низькому рівні та описує, які ресурси потрібні під час виконання, які програмні модулі використовуються протягом всього процесу, а які вхідні та вихідні дані.

- *Точка зору взаємодії*, яка описує, як координується залучення до процесу агентів. Це робить крок далі, ніж діаграми послідовності UML, оскільки відображає мотивацію взаємодії та її учасників. Він також включає інформацію про психічний стан, необхідний кожному агенту протягом всієї взаємодії, а також про завдання, що виконуються в процесі. Це дозволяє нам на проєктному рівні обґрунтувати, чому агент бере участь у взаємодії та чому це має продовжуватися.

Враховуючи вищезазначену інформацію в подальших розділах буде описана архітектура мультиагентної системи для обробки текстових потоків даних, будуть враховані поради щодо методологій побудови даних систем, а також – щодо властивостей, які характеризують розподілені системи.

## **2.2. Модель МАС для вирішення задачі автоматичної побудови словника предметної області, взаємодія агентів, мова спілкування між агентами**

З метою апробації моделі обробки великих даних буде вирішено задачу автоматичної побудови словника предметної області, що детально була описана в першому розділі дисертації. На рис 2.4 базово зображена мультиагентна система для вирішення даної задачі. Вона складається щонайменше з дев'яти агентів, деякі з яких можуть існувати в декількох екземплярах. Цими агентами є – агент інтерфейсу, менеджменту, запиту, результату, два агенти даних, два агенти для саме вирішення задачі та агент

для періодичної перевірки стану системи. В подальшому у даному розділі буде детально описана робота цієї моделі.

Ця модель є новим внеском в теорію, оскільки вона є коректною, повною та не є надлишковою. Це доведено під часу аналізу даної моделі з використанням UML-діаграм, де представлені різні можливі випадки роботи даної моделі та прописані дії кожного агента. Діаграми викладені в цьому розділі.

У запропонованій моделі мультиагентній системі агент пошуку та фільтрації даних відіграє ключову роль в інтеграції та попередній обробці інформації. Його функціонал передбачає здійснення активного пошуку релевантних даних у мережі Інтернет, у розподілених файлових сховищах або приймання поточних даних за зовнішніми запитами. Агент виконує інтелектуальну фільтрацію отриманої інформації, застосовуючи визначені критерії достовірності та релевантності, що дозволяє суттєво знизити ймовірність надходження некоректних або нерелевантних даних у систему. Завдяки використанню механізмів попередньої валідації, агент забезпечує передачу відфільтрованих даних до агента менеджменту для подальшої структуризації та обробки. Такий підхід дозволяє не лише оптимізувати процес обробки великомасштабних поточних даних, але й підвищити загальну ефективність функціонування мультиагентної системи в умовах динамічного інформаційного середовища.

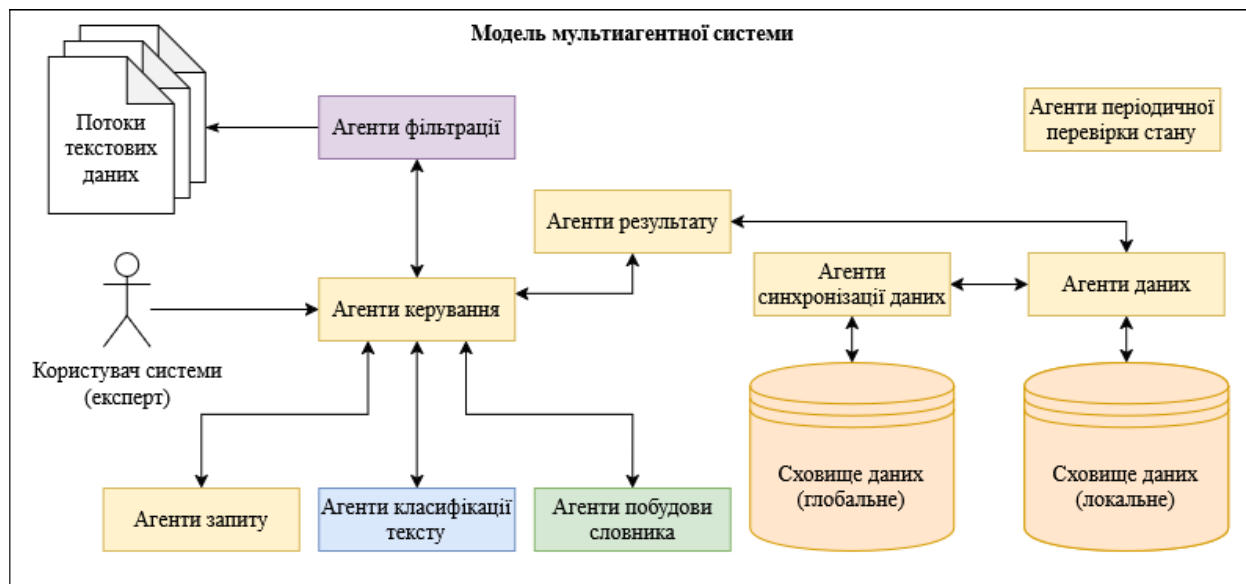


Рис. 2.4. Загальний вигляд моделі мультиагентної системи для аналізу текстових даних [власна розробка]

Агент менеджменту, спілкуючись з агентом запиту, дізнається про деталі даного запиту та отримує «завдання» на виконання для досягнення необхідного результату. Цей агент поєднаний майже з усіма агентами, але може існувати в декількох програмних екземплярах (декількох процесах), як зображено на рис. 2.5. У випадку, якщо необхідно розмістити агентів на одному обчислювальному вузлі (комп'ютері), то розміщення відбувається в різних процесах, як вказано на рис. 2.5 а). Альтернативою може бути розміщення на різних вузлах, це вказано на рис. 2.5 б). Агент менеджменту може бути використаний для взаємодії з користувачем (яким може бути людина або інша програма, яка звертається до цього агента). Також, даний агент надає відповідь назад користувачу після виконання роботи при наявності зовнішніх запитів до системи.

Варто зазначити, що подібне розгортання може стосуватись не лише агентів менеджменту, а й всіх інших, тому що кількість вхідних даних може

неперервно збільшуватись, і у випадку досягнення обмежень в обчислювальних потужностях іншими агентам необхідно буде створювати двійників та передавати задачі на них.

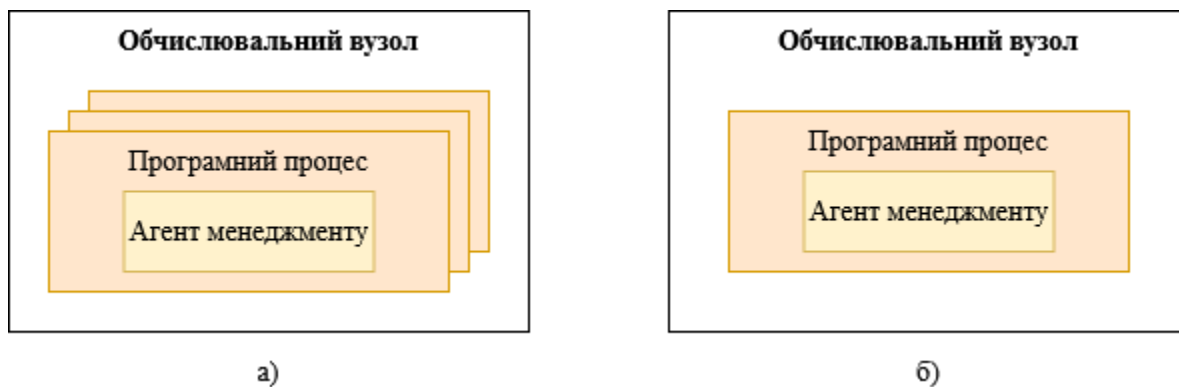


Рис. 2.5. Можливості розгортання агента менеджменту [власна розробка]

Що стосується агенту результату – він потрібен для перетворення результату у формат, придатний для збереження в базі даних або у звичайному файловому сховищі (неважливо в контексті даної роботи). Оскільки, існує ціль якомога швидше звільнити агентів власне обробки даних, то необхідно передати задачу на агента результату.

На діаграмі зображені агенти для роботи власне зі сховищами даних, причому виділено два типи агентів. Мається на увазі, що на кожному обчислювальному вузлі має бути певний набір даних для роботи – це дані, які агент отримує в якості результату, а ще – це дані, з якими агент саме працює. Як приклад можна навести наступне: агент має займатись побудовою словника предметної області, але в той самий час агенти на інших вузлах теж працюють над цією задачею. Тому, відповідно, необхідно агенту доводитись працювати з локальною копією даних, а не зі спільним поєднаним в єдине ціле результатом. Для досягнення синхронізації агенти даних спілкуються час від часу та повідомляють про оновлення, які необхідно буде переслати між

вузлами. Це потрібно робити періодично, щоб не виникало великої десинхронізації, бо це приведе до погіршення точності кінцевих результатів.

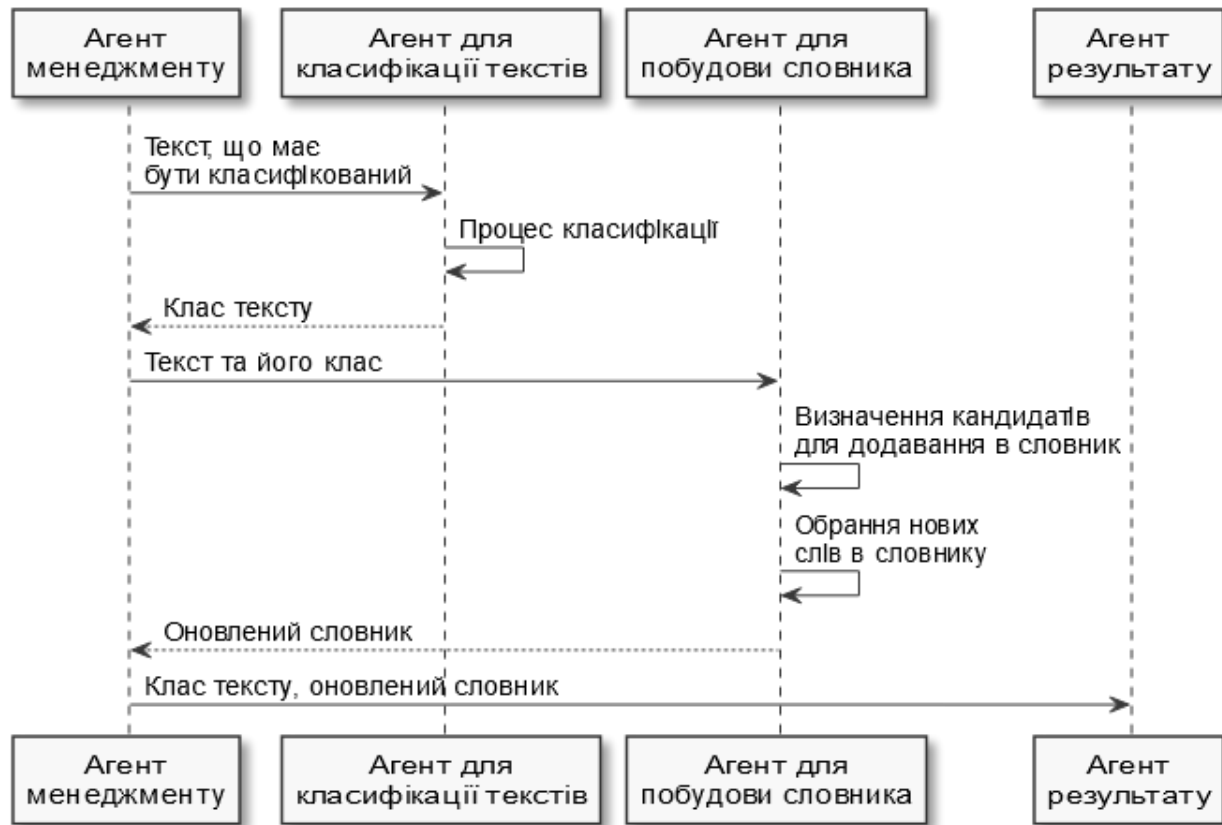


Рис. 2.6. Діаграма послідовностей взаємодії агентів при аналізі текстів  
[власна розробка]

Агент для класифікації тексту, а також – агент для побудови словника на даній діаграмі зображені лише схематично, оскільки більш детальна їх робота, а також – архітектура описані в наступних розділах дисертації. Проте, як можна побачити з архітектури, обидва агенти поєднані з агентом менеджменту, відповідно, саме він їм передаватиме задачі. Приклад такої послідовності можна побачити на діаграмі послідовностей, що зображена на рис. 2.6. Основним є те, що чітко виділені зони відповідальностей кожного агенту. Один з них відповідає за визначення класу текстів, другий – за власне



формування словнику, і тому немає необхідності в їх постійній взаємодії, і, як наслідок – завантаженні каналу зв'язку.

В даній системі також зображений агент для перевірки стану системи. Може статись так, що для якоїсь комбінації вхідних даних агент обробки даних буде в стані нескінченного циклу, наприклад, або буде завершений з помилкою. Іншими словами – не зможе виконувати свої функції. Для цього випадку агент перевірки стану системи час від часу питає в агентів про їх статус робіт. Чи довго вони стоять без завантаження певною задачею, чи може якісь з них перестають відповідати. В такому випадку необхідно буде сповістити агента менеджменту про проблему, і тоді цей агент створить або альтернативну задачу, або повідомить про помилку агента перевірки стану, тоді останній вже має залишити всю необхідно інформацію для подальшого аналізу людиною в файловому сховищі. На основі цієї інформації систему можна буде покращувати в майбутньому, що буде сприяти її розвитку.

Крім власне взаємодії на високому рівні, варто ще описати формат повідомлень та взаємодії агентів. Проаналізувавши альтернативні варіанти було обрано за засіб взаємодії мову комунікації агентів FIPA ACL. Загальний протокол комунікації зображена на рис. 2.7.

Форум стандартів агентів FIPA [24] зосереджується на конкретних протоколах зовнішньої взаємодії та служб платформи, а не на поведінці внутрішнього агента, оскільки останні не є легкодоступними та спостережуваними, на відміну від взаємодії із зовнішніми агентами. Крім того, відповідні алгоритми для опису внутрішньої поведінки різних типів агентів важко стандартизувати, оскільки вони часто є поєднанням конкретних проблем, специфіки додатків і потребують того, щоб організації залишалися

приватними, а не публічними. Модель FIPA (взаємодія агента), яку часто називають FIPA-ACL, насправді є скоріше набором протоколів взаємодії агента (AIPS), а не єдиною декларативною мовою спілкування агента. AIPS містить кілька різних семантичних протоколів для спілкування агентів, включаючи: процес взаємодії, комунікативні акти, логіку вмісту та онтології вмісту. AIPS також визначає кілька різних синтаксичних протоколів для визначення структури повідомлень, їх кодування та їх транспортування повідомлень. Вони пояснюються більш докладно наступним чином.

```
"Query ADI-11" : {  
  "performative"      : "query-ref", //FIPA00037  
  "sender"            : "agent-identifier :name aid-02", //FIPA00061  
  "receiver"          : "set (agent-identifier :name aid-11)", //FIPA00061  
  "reply-to"          : null, //FIPA00061  
  "content"           : "(all ?x (can-be-set ?y aid-11))", //FIPA00008 WIP  
  "language"          : "fipa-sl", //FIPA00008  
  "encoding"          : null, //FIPA00007  
  "ontology"          : "fipa-acl", //FIPA00008  
  "protocol"          : null, //FIPA00025  
  "conversation-id"   : null, //FIPA00061  
  "reply-with"        : "query-12", //FIPA00061  
  "in-reply-to"       : null, //FIPA00061  
  "reply-by"          : null //FIPA00061  
}
```

Рис. 2.7. Формат спілкування агентів із зазначенням специфікації, яка описує відповідну секцію [власна розробка]

Протокол CA Communicative Act є суттю моделі FIPA-ACL і визначає спілкування як сукупність різних СА, заснованих на теорії актів мовлення [25].

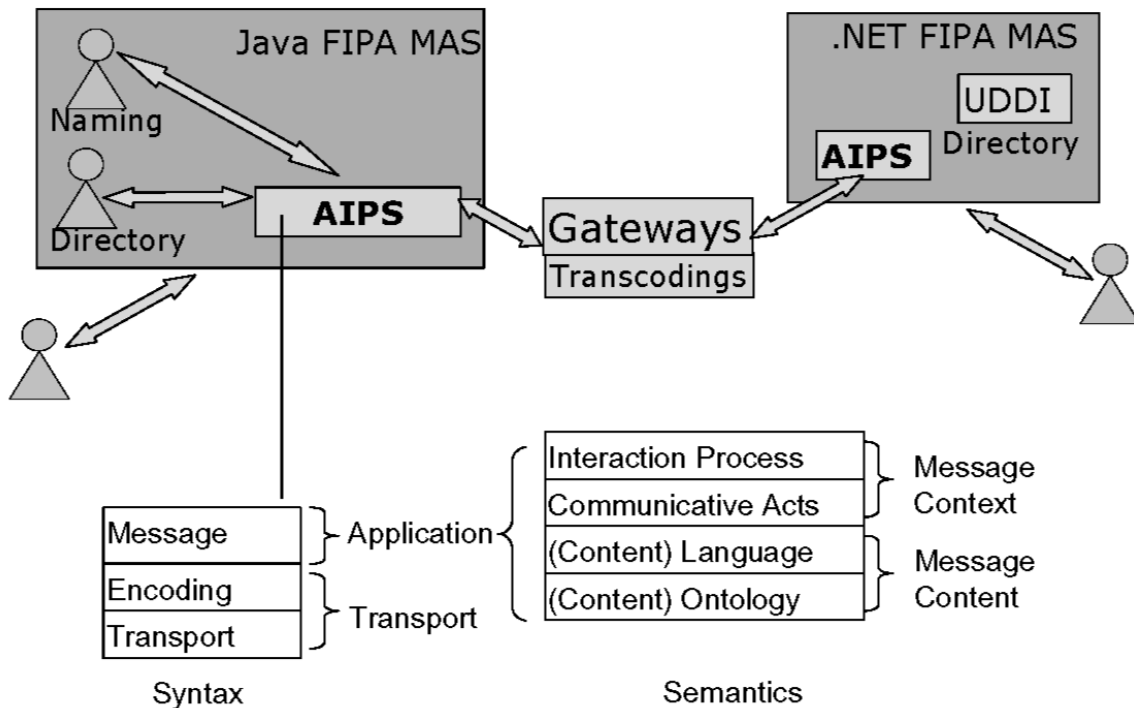


Рис. 2.8. Загальна схема взаємодії агентів за стандартом FIPA [26]

Метадані для сприяння комунікації агента, як правило, включають імена відправника та одержувача, прив'язані до мережових адрес, тип повідомлення (тип FIPA Communicative Act), тип взаємодії (процесу), частиною якого він є, використовувана онтологія та логіка вмісту, тайм-аути для відповідей та ідентифікаторів повідомлень, щоб визначити інші повідомлення, на які він відповідає. Протокол кодування дозволяє обміну повідомленнями використовувати декілька кодувань, таких як Stings, XML та бітову ефективність, остання розроблена для використання через мережеві лінії з меншою пропускну здатністю.

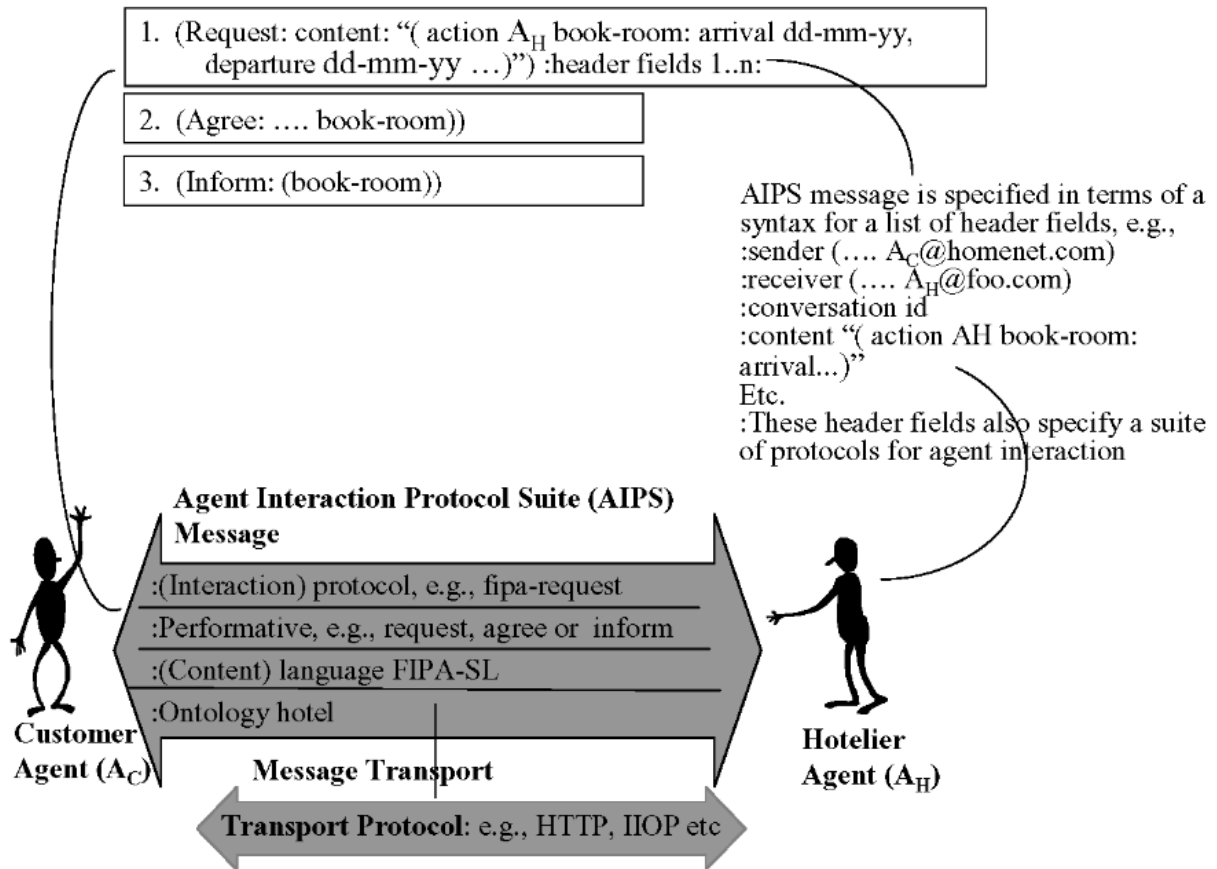


Рис. 2.9. Список секцій повідомлення при взаємодії агентів [26]

Специфікація FIPA FIPA00084 визначає використання кількох надійних асинхронних протоколів передачі повідомлень для обміну повідомленнями, наприклад, асинхронний HTTP [27]. Семантична мова (SL) використовується для визначення семантики для FIPASA як логіки психічних установок і дії, оформлені в модальній мові першого порядку з ідентичністю; див. Специфікацію бібліотеки комунікативного акту FIPA [27] для деталей цієї логіки. Для того, щоб ЦС планувалося або передбачалося відправником, повинні бути вказані як передумови (причини, за якими обрано діяння), так і умови (після), які повинні бути виконані після завершення дії.

Опис повідомлень для різних агентів наведений в додатку Б. Нижче в таблиці вказані приклади актів комунікації двох агентів. Як можна помітити, у запиті необхідно вказувати агенту якого типу йде запит, ідентифікатор запиту (для подальшого отримання відповіді), сам вміст запиту, мову та онтологію.

|  |
|--|
| <pre>"Query ADI-19" : {<br/>  "performative"      : "query-ref",<br/>  "sender" : "agent-identifier :name aid-02",<br/>  "receiver" : "set (agent-identifier :name aid-19)",<br/>  "content" : "(all ?x (can-be-set ?y aid-19))",<br/>  "language" : "fipa-sl",<br/>  "ontology" : "fipa-acl",<br/>  "reply-with" : "query-13"<br/>}</pre>                             |
| <pre>"AID-11 response" : {<br/>  "performative"      : "inform",<br/>  "sender" : "agent-identifier :name aid-11",<br/>  "receiver" : "set (agent-identifier :name aid-02)",<br/>  "content" : "(( = (all ?x (can-be-set ?y aid-11)) (set (2, 3, 7, 8)) ))",<br/>  "language" : "fipa-sl",<br/>  "ontology" : "fipa-acl",<br/>  "in-reply-to" : "query-12"<br/>}</pre> |

Рис. 2.10. Приклади формату спілкування агентів [власна розробка]

В даному підрозділі була описана мультиагента система для аналізу текстових даних без детального опису власне роботи агентів для класифікації текстів та побудови словника предметної області. Це описується в наступних розділах. Також, було описано мову для спілкування агентів, що базується на використанні стандарту FIPA, а приклад її опису знаходиться в додатку. Проте, мультиагентна система має бути ще програмно реалізована з використанням сучасних засобів. Виникає ряд питань, як саме обрати мову програмування, а також – програмну бібліотеку або фреймворк для цього. Опис та порівняльний такого інструментарію наведений в наступному розділі.

### **2.3. Аналіз та порівняння мов програмування та програмних бібліотек для розробки моделі МАС при обробленні слабоструктурованих текстових даних.**

На сьогоднішній день існує декілька реалізацій мультиагентних систем для різних мов програмування. Деякі з них є застарілими та частково втратившими підтримку, інші – активно розвиваються і можуть бути доповнені користувачем самостійно, у випадку потреби. В даному розділі розглянуті основні бібліотеки та їх особливості, недоліки та переваги. Як результат даного розділу буде рекомендація стосовно практичного застосування бібліотек для проведення подальшого дослідження.

В першу чергу розглянуті мови та бібліотеки, що з'явилися ще на початку двохтисячних років. Роботою, в якій проведено аналіз цих бібліотек являється [28]. Проте, не всі бібліотеки повторно описані, а лише ті, які на сьогоднішній день ще існують та описані на офіційних ресурсах.

Описуючи підходи для створення мультиагентних систем їх варто поділити на декілька груп, а саме:

- з використанням декларативних мов;
- з використанням імперативних мов;
- з використанням інтегрованих середовищ для розробки;
- з використанням агентних платформ та фреймворків;
- гібридні підходи.

Більшість досліджень в агентно-орієнтованих мовах програмування базується на декларативних підходах. Існує багато декларативних рішень, більшість з яких ґрунтуються на логіці. Суто імперативні мови незвичні в літературі агентів, оскільки по суті вони не підходять для вираження абстракцій високого рівня, пов'язаних із дизайном систем агентів. З іншого

боку, орієнтовані на агенти мови програмування, як правило, забезпечують легку інтеграцію із (застарілим) кодом, написаним імперативними мовами. Цікаво, що характеристики архітектур базового агента визначають, що часто доцільніше використовувати інтерпретатори, а не компілятори [28].

Що стосується інтегрованих середовищ для розробки – існуючі середовища забезпечують базову підтримку управління проектами, створення, редагування файлів та побудови і запуску систем, але не вдаються для підтримки складних функцій у всіх цих категоріях. Крім того, жодна з IDE агента не розкриває аспекти рефакторингу та тестування застосувань агента. Однією з причин цього є те, що, крім системи LivingSystems, усі середовища розроблені з нуля і, відповідно, не покладаються на вже існуючі технології. Взагалі, підтримка IDE для розробки систем, заснованих на агентах, є досить слабкою, і існуючі інструменти агентів не пропонують того самого рівня зручності користування, яка в є сучасних об'єктно-орієнтованих мовах [28].

Різні підходи, згадані в ході цього опитування, свідчать про те, що ще потрібно зробити багато роботи. Серед основних викликів, з якими стикається дослідницьке співтовариство, є:

- Концепція та розробка спеціалізованих засобів виправлення помилок, зокрема для мов когнітивних агентів;
- Інтеграція інструментальних засобів у існуючі IDE, а не починаючи з нуля;
- Розділення MAC-фреймворків з агентських платформ, так що кожен фреймворк може бути використаний для розгортання систем на різноманітних платформах;

- Поширення парадигми програмування МАС, щоб програмісти могли краще зрозуміти його основи, а також практичні характеристики [28].

Серед декларативних мов на сьогоднішній день ще існує така мова, як DALI [28]. Більше того, програмний код та документація викладені на github [29]. DALI – це мова для логічного програмування, призначена для виконання специфікації логічних агентів. Вона використовує «рупорні» речення, а її семантика заснована на моделях Least Herbrand. Розробники мови мають намір забезпечити конструкції для повторення реактивності та проактивності агента за допомогою правил. Агент DALI – це логічна програма, яка містить повторно активні правила, події та дії, спрямовані на взаємодію із зовнішнім середовищем. Реактивна та активна поведінка агента DALI викликається кількома видами подій: зовнішніми, внутрішніми, теперішніми та минулими. Усі події та дії мають відмітку часу, щоб записати, коли вони відбулися. Нові синтаксичні сутності, тобто предикати, пов’язані з подіями та проактивністю, позначаються спеціальними поштовими індексами. Коли подія відбувається у «зовнішньому світі» агента, агент може це сприйняти і вирішити реагувати. Реакція визначається реактивним правилом, яке враховує цю зовнішню подію. Внутрішні події визначають поведінку агента DALI, роблячи його активним незалежно від навколишнього середовища та дозволяючи йому маніпулювати та переглядати свої знання.

Більш відомою та й досі актуальною на сьогоднішній день є мультиагентна платформа JADE. JADE (англ., Java Agent DEvelopment Framework) – це платформа Java для розробки розподілених мультиагентних додатків. Він представляє агентне проміжне програмне забезпечення, що надає набір доступних та простих у використанні послуг та декілька графічних інструментів для налагодження та тестування. Однією з основних цілей



платформи є підтримка взаємодії, суворо дотримуючись специфікацій FIPA, що стосуються архітектури платформи, а також комунікаційної інфраструктури. Більше того, JADE дуже гнучкий і може бути адаптований для використання на пристроях з обмеженими ресурсами, таких як КПК та мобільні телефони. JADE широко використовувався протягом останніх років багатьма академічними та промисловими організаціями. Як приклад було використано JADE для побудови агентної системи для підтримки прийняття рішень в центрах трансплантації органів [28].

Звісно, дана платформа є актуальною, має наявні на офіційній сторінці вихідні коди, підтримує стандарт FIPA, тощо. Тому її можна використовувати для досліджень й на сьогоднішній день. Проте, не дивлячись на наявність вихідних кодів в даній платформі, перезібрати та додати власні зміни в платформу є лише теоретично можливим, для цього необхідно окремо контактувати з розробниками платформи. Це додає затримки при проведенні власних експериментів, і тому є проблемою. Також, проблемою є певна «важкість» даної платформи з точки зору розгортання на кінцевому обчислювальному вузлі віртуальної машини Java, а після цього – й самої JADE. І це варто враховувати при початку використання платформи.

Однією з таких бібліотек є PADE. PADE - це основа для розробки, виконання та управління мультиагентними системними в середовищах розподілених обчислень. PADE повністю написаний на мові Python і використовує бібліотеки Twisted для здійснення зв'язку між мережевими вузлами. PADE є відкритим кодом на умовах ліцензії MIT і розроблений Smart Grids Group (GREI) у кафедрі електротехніки Федерального університету Сеари, Бразилія. Будь-хто, хто хоче внести свій внесок у проєкт PADE, може

це зробити. Кожен можете завантажити, виконати, протестувати та надіслати розробникам відгук про функціональність PADE [30].

Додатково до вищезазначених переваг варто додати й те, що в PADE можна задавати FIPA-стандартизований обмін повідомленнями, що, як було показано в попередніх розділах, є надважливим з точки зору розвитку даного наукового напрямку та інтеграцією з іншими дослідженнями. Також, наявність актуальної документації є великим плюсом при користування бібліотекою.

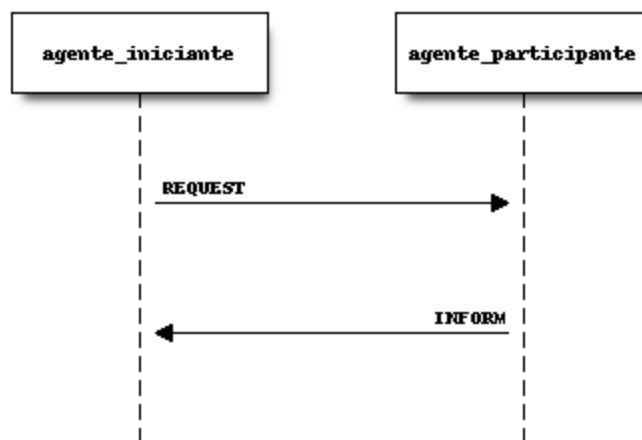


Рис. 2.11. Діаграма послідовностей для FIPA-запитів з використанням PADE [30]

Останньою бібліотекою, яку варто зазначити, є бібліотека osBrain. osBrain - це загальний мультиагентний системний модуль, написаний на Python і розроблений OpenSistemas. Агенти працюють незалежно як системні процеси та спілкуються між собою за допомогою передачі повідомлень. osBrain використовує ØMQ для ефективного та гнучкого передавання повідомлень між агентами. Вона також використовує Pyro4 для полегшення конфігурації та розгортання складних систем [31].

Як зазначається на офіційному сайті, Python був обраний як мова для швидкого прототипування та за наявності величезної екосистеми аналізу даних. Він був збережений для остаточної роботи вихідного коду, створеного з ним. Поява osBrain була наслідком низки кроків, які були зроблені в процесі розробки:

- Виділення агентів; створення окремих системних процесів, щоб уникнути спільної пам'яті та будь-яких проблем, пов'язаних із багатопотоковою розробкою.
- Реалізація передачі повідомлень; використання гнучкої бібліотеки ØMQ.
- Простота конфігурації / розгортання; використовуючи дуже зручний, добре впроваджений та задокументований пакет Pyto4.
- Відокремлення від торгової платформи; що почалося як базова архітектура для реалізації платформи автоматизованої торгівлі в режимі реального часу, а в кінцевому підсумку стала загальною архітектурою мультиагентних систем.

Проаналізувавши вищезазначені мови, платформи та бібліотеки, рекомендовано для вирішення задачі прототипування системи для обробки великих аналізів текстових даних використовувати бібліотеку PADE через наступні переваги:

- безкоштовне використання;
- відкриті вихідні коди бібліотеки;
- декларація зі сторони розробників про можливість розширення функціоналу бібліотеки;
- наявність можливості надсилати FIPA-стандартизовані повідомлення між агентами;

- можливість розгортання на декількох обчислювальних вузлах;
- актуальний стан програмних кодів бібліотеки, підтримка її з боку розробників;
- використання мови програмування Python, що дає можливість простої інтеграції з бібліотеками для інтелектуального аналізу даних.

#### **2.4. Опис моделі MAC та запитів на мові FIPA ACL для вирішення задачі.**

В даному розділі описується архітектура підсистеми для побудови словника предметної області. Причому, архітектура саме мультиагентної системи, з поясненням роботи кожного типу агентів. На рис 2.12. зображена високорівнева архітектура. Також в цьому розділі розглядається також дана система з точки зору розгортання в реальному середовищі.

На діаграмі зображені чотири основні типи агентів, а саме – агент менеджменту вхідних запитів, агент розрахунку змін до словника, агент голосування та агент приймання кінцевого рішення. Агенти голосування за своїм основним завданням одночасно також можуть бути агентами приймання кінцевого рішення, якщо вони роблять це в локальному середовищі.

Агент менеджменту вхідних запитів отримує текст та інформацію про його клас, що вже були розраховані попередньо підсистемою. Даний агент має визначити оптимальний по завантаженості обчислювальний вузол (у випадку розподіленої реалізації даної системи). На цьому вузлу знаходиться або створюється новий агент для розрахунку змін до словника. В загальному випадку пропонується створення / використання цих агентів з різними методами відповідно до попередньо розрахованих ймовірностей створення. Справа в тому, що визначення словника виключно одним методом може

призвести до загальної одноманітності термінів в ньому. Проте, виходячи з цього, виникає питання, чи не стане результат занадто «суб'єктивним».

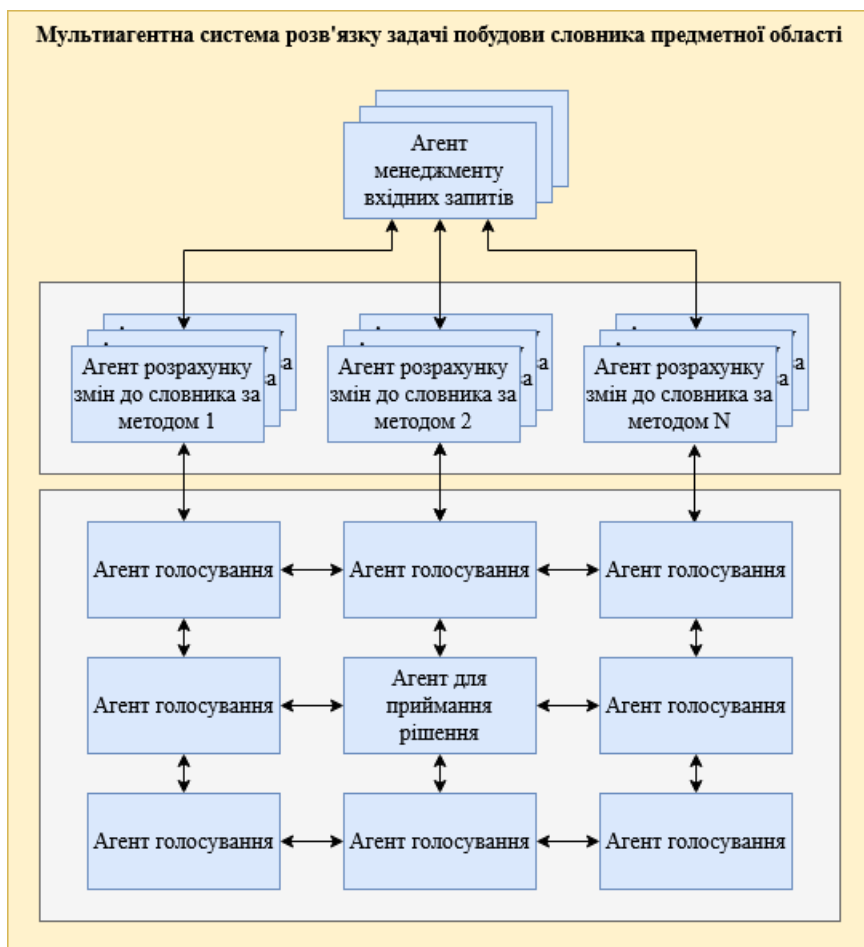


Рис 2.12. Загальний вигляд підсистеми побудови словника [власна розробка]

Як спосіб вирішення даної проблеми пропонується створювати декілька агентів. При цьому, якщо завчасно буде визначено експертами-мовознавцями, що один з методів дає кращий результат, тоді цей метод має бути використаний частіше. Як початкове значення ймовірностей пропонується виставити значення 0.5.

Вже після розрахунку нового словника, а точніше – нового локального словника для даного агента, необхідно ці результати поєднати в один новий

словник. Для цього і потрібні агенти голосування та приймання рішень. Як буде показано далі – робота цих агентів залежить від їх розміщення в системі. Що ж стосується їх роботи – агенти між собою спілкуються, обмінюються локальними словниками та визначають нових кандидатів для додавання в словник. Це робиться методами голосування, більше деталей про які описано в наступних розділах.

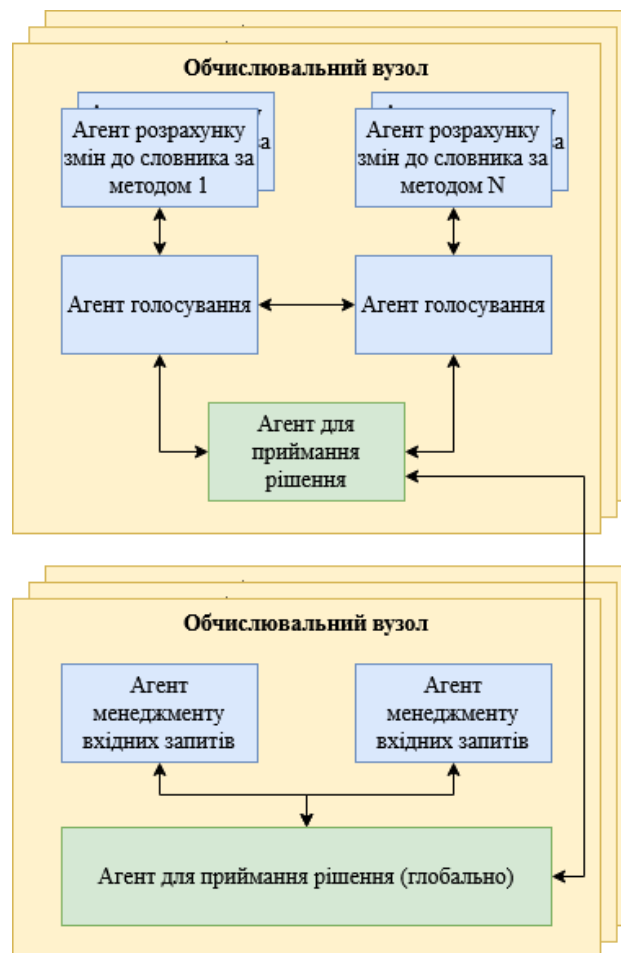


Рис 2.13. Запропонований варіант розгортання системи для обробки великих масивів даних та зменшення затримок міжпроцесної взаємодії [власна розробка]

На рис. 2.13. зображений можливий варіант розгортання системи. Можливим він є через те, що в залежності від завантаження – вигляд може

бути змінений. В попередніх розділах було вказано, що агенти можуть працювати в різних програмних процесах та програмних потоках, а ще – знаходитись на різних обчислювальних вузлах. В цьому ж випадку передбачається, що для суттєвого зменшення затримок при передачі даних варто помістити агентів для розрахунку змін до словника, агентів голосування для локального прийняття рішень на один вузол – задля зменшення затримок передачі даних.

Але, оскільки в базу даних має бути занесений новий варіант словника, то необхідно мати агента, що приймає рішення глобально, коли локальні вже завершили роботу – вони разом з глобальним утворюють таку саму мережу для прийняття рішень, як і було описано раніше для одного обчислювального вузла. Далі через агентів менеджменту після прийняття остаточного рішення цей агент надсилає результат назад, і він йде до агентів, які відповідають за зберігання інформації у постійній пам'яті.

Новий екземпляр словника для предметної області має бути переданий всім іншим агентам, і для цього агенти після зберігання інформації (це є дуже важлива дія) – повідомляють решті агентів про оновлення словників. Таким чином, система продовжує самовдосконалення та користується результатами власної роботи в подальшому.

Для спілкування агентів цього виду потрібно мати стандартизований протокол – мову. Так само, як і для всієї системи, використовується мова спілкування ACL, основа якої описана раніше. Також, в таблиці 2.1 описані приклади запитів для голосування. Приклади запитів для деяких інших агентів вказані в додатку. Даний розділ лише описує інформацію, необхідну для голосування агентів.

Таблиця 2.1. Приклади запитів для голосування агентів

|   |
|---|
| <pre> "Query Voting-20" : {   "performative" : "query-ref",   "sender" : "agent-identifier :name aid-02",   "receiver" : "set (agent-identifier :name aid-19)",   "content" : "(all ?x (can-be-set ?y aid-19))",   "language" : "fipa-sl",   "ontology" : "fipa-acl-voting",   "reply-with" : "query-20" } </pre>                                   |
| <pre> "Voting-20 response" : {   "performative" : "inform",   "sender" : "agent-identifier :name aid-voting-1",   "receiver" : "set (agent-identifier :name aid-02)",   "content" : "(( = (all ?x (can-be-set ?y aid-11)) (set (4, 3, 2, 7)) ))",   "language" : "fipa-sl",   "ontology" : "fipa-acl-voting",   "in-reply-to" : "query-20" } </pre> |

В наступних підрозділах описані методи голосування, прийняття рішень та власне аналізу текстових даних на предмет визначення словника. В загальному ж, вони взаємодіють саме таким чином, як було описано в даному розділі.

Відповідно до запропонованих даних передбачається, що можна створити мультиагентну систему, яка міститиме в собі не лише один класифікатор, а й можливість перевірити правильність виконання у випадку недостатнього рівня класифікації з використанням одного з методів. На рис. 2.14. зображена загальна модель даної системи.

Дана система складається з чотирьох типів агентів, при чому, деякі з них можуть існувати в декількох екземплярах. Перш за все інформацію отримує агент менеджменту вхідних запитів, які оцінюють завантаженість системи та можливість надсилання агентів на обчислювальні вузли у випадку розподіленої системи. Дані агенти в першу чергу надсилають текст до агента,



який буде класифікувати тексти з використанням нейронної мережі (наприклад, методом RNN).



Рис 2.14. Модель мультиагентної системи для класифікації текстових даних  
[власна розробка]

Ця нейронна мережа має високу точність класифікації згідно з попередніми дослідженнями. Проте, існує ймовірність, що в результаті роботи системи з новими текстами, агент RNN може отримати низький рівень класифікації, у порівнянні з навчальною вибіркою. Пропонується визначити, якщо агент RNN отримує результат менший на 90% стосовно належності тексту до одного з класів, тоді цей агент передає на аналіз тексту людиною з метою донавчання системи.

Необхідність такого рішення впливає з того, що якість отриманого словника має високий пріоритет для даної системи, відповідно, варто максимально точно класифікувати текст. Коли цей агент отримує результат,

то він робить класифікацію власним методом. Якщо клас підтверджується (порівняння йде з класом, який мав найвищий рівень класифікації у попереднього агента), тоді текст передається в роботу останньому агенту. Якщо ж трапиться так, що агент визначає інший клас, тоді цей текст передається до агента, котрий буде зберігати тексти для подальшого аналізу людиною-експертом.

Коли ж текст успішно класифіковано, тоді він передається останньому агенту разом з отриманим класом. Цей агент передає в роботу текст до підсистеми автоматичної побудови словника, яка буде описана в наступному розділі даної дисертації.

## **2.5. Методи голосування для визначення кінцевих словників в розподіленому середовищі.**

Існує ряд підходів для голосування агентів, проте одним з найкраще визначених є підхід, описаний в роботі [32]. Тому, описані в ній основні кроки, використані і в даній роботі. Згідно з [32], процедура проведення голосування наступна:

- комітет засідає, а голова відкриває засідання;
- учасник подає пропозицію;
- інший учасник аналізує пропозицію;
- учасники обговорюють подання;
- голова закликає тих, хто «за» подання, проголосувати;
- голова закликає тих, хто «проти» подання, проголосувати;
- подання приймається або ні згідно з встановленими правилами голосування комітету.

В контексті дорадчої асамблеї ми зв'яжемо комітет із самою асамблеєю, а учасників комітету – з учасниками асамблеї. Нарада учасників, на якій мають

прийматися рішення, буде називатися сесією: за час тривалості асамблеї може бути кілька сесій. У наших програмах електронного голосування, оскільки вузол прийняття рішень є агентом, то термін «агент» буде використовуватися в подальшому аналізі і для «учасника».

Цей неофіційний опис дає основні етапи процедури голосування. Щоб визначити, хто наділений повноваженнями (у тому сенсі, який буде чітко визначений пізніше) робити кожен крок, ми ідентифікуємо ряд підмножин набору агентів. Ми тут детально не розглядаємо як визначається набір учасників; хоча ми зазначаємо, що це можна зробити різними способами. Наприклад, це може бути зроблено за замовчуванням (коли представник на асамблеї автоматично відноситься до тієї чи іншої групи); за кваліфікацією (наявність якогось сертифіката або вміння чи здатності забезпечує класифікацію) або за призначенням (тобто, коли розподіл агентів до набору учасників визначається якимось іншим протоколом).

Підмножинами набору агентів відповідно до ролей, які можуть займати агенти, є:

- **виборець**, тобто ті агенти, які мають право голосу;
- **заявник**, ті агенти, які уповноважені подавати клопотання;
- **відрядник** (виступаючий за проєкт), ті агенти, які уповноважені на другу дію;
- **голова**, ті агенти, які кваліфіковані для проведення процедури; один з яких у будь-який час буде призначений фактичним головою і тим самим наділений повноваженнями проводити процедуру;

- **відстежувач**, ті агенти, які повинні бути проінформовані про дії інших, зокрема про те, як вони віддають свої голоси (за чи проти), та про результати голосування (відбулися чи ні).

Таблиця 2.2. Дії в протоколі голосування [32]

| Дії                   | Текстовий опис                         |
|-----------------------|--|
| open_session (Ag,S)   | Агент Ag починає сеанс S               |
| close_session(Ag,S)   | Агент Ag закриває сеанс S              |
| propose(Ag,M)         | Агент Ag пропонує дію M                |
| second(Ag,M)          | Агент Ag пропонує другу дію M          |
| open_ballot(Ag,M)     | Агент Ag починає голосування на дію M  |
| close_ballot(Ag,M)    | Агент Ag закриває голосування за дію M |
| vote(Ag,M,aye)        | Агент Ag голосує за дію M              |
| vote(Ag,M,nay)        | Агент Ag голосує проти дію M           |
| revoke(Ag,M)          | Агент Ag відкликає свій голос на дію M |
| abstain(Ag,M)         | Агент Ag утримується від руху M        |
| declare(Ag,M,carried) | Агент Ag заявляє, що рух M здійснив    |

|                           |  |
|---------------------------|--|
| declare(Ag,M,not_carried) | Агент Ag заявляє, що рух M не здійснив |
|---------------------------|--|

В таблиці 2.2. описані основні етапи голосування, а саме – можливі дії агентів. Також, у випадку помилкових результатів можуть бути введені санкції проти конкретного агента, але це не є обов’язковим і в рамках даного дослідження цей момент упускається.



Рис 2.15. Голосування агентів за зміни в словнику [власна розробка]

Але для вирішення поставленої задачі простого голосування «за» та «проти» недостатньо. Тобто потрібно ранжувати можливі варіанти – голосувати за зміни які мають бути застосовані до методів оловування. Відповідно, загальна процедура голосування описана вище, а далі – пропонується модифікація методу Шульце, яка дозволить провести коректне ранжування.

Основна ідея методу Шульце полягає в тому, що сила непрямого порівняння “альтернатива  $a$  проти альтернативи  $b$ ” – це сила найсильнішої частини  $a \equiv c(1), \dots, c(n) \equiv b$  від альтернативи  $a \in A$  альтернатива  $b \in A \setminus \{a\}$  і що сила шляху є силою  $(N[c(i), c(i+1)], N[c(i+1), c(i)])$  його найслабшого зв'язку  $(i), c(i+1)$ .

Метод Шульце визначається наступним чином. Шлях від альтернативи  $x \in A$  до альтернативи  $y \in A \setminus \{x\}$  - це послідовність альтернатив  $c(1), \dots, c(n) \in A$  з такими властивостями:

1.  $x \equiv c(1)$
2.  $x \equiv c(n)$
3.  $n \in \mathbb{N}$  з  $2 \leq n \leq \infty$
4. Для будь якого  $i = 1, \dots, (n-1)$ :  $c(i+1) \in A \setminus \{c(i)\}$

Силою шляху  $c(1), \dots, c(n) \in \min_D \{ (N[c(i), c(i+1)], N[c(i+1), c(i)]) \mid i = 1, \dots, (n-1) \}$ . Іншими словами – як і зазначено вище – це сила найслабшого зв'язку.  $P_D[a, b] \in \mathbb{N}_0 \times \mathbb{N}_0$  це сила найсильнішого шляху з альтернативи  $a \in A$  до альтернативи  $b \in A \setminus \{a\}$  [33]. Бінарне відношення  $O$  в  $A$  визначене як наступне:  $ab \in O : \Leftrightarrow P_D[a, b] >_D P_D[b, a]$ . Як результат – набором  $S$  є набір потенційних переможців  $S := \{ a \in A \mid \forall b \in A \setminus \{a\}: b a \notin O \}$ .

Приклад голосування між агентами з використанням методу Шульце показаний в таблицях 2.3 та 2.4. Таблиця парних преференцій показує, скільки разів кожне слово (рядок) було обрано вище іншого слова (стовпець) у голосах агентів. Сила шляху між парами слів визначається не лише прямими порівняннями, а й через транзитивні зв'язки. Наприклад, якщо "спорт" >

"команда" та "команда" > "грати", тоді "спорт" > "грати". Видно, що в таблиці 2.4 результат переваги слова "спорт" над "грати" змінився.

Таблиця 2.3. I крок методу Шульце: розрахунок парних преференцій

|         | спорт | команда | грати | м'яч |
|---------|-------|---------|-------|------|
| спорт   | 0     | 3       | 2     | 3    |
| команда | 1     | 0       | 3     | 2    |
| грати   | 2     | 1       | 0     | 3    |
| м'яч    | 1     | 2       | 1     | 0    |

Таблиця 2.4. II крок методу Шульце: розрахунок матриці сили шляхів

|         | спорт | команда | грати | м'яч |
|---------|-------|---------|-------|------|
| спорт   | 0     | 3       | 3     | 3    |
| команда | 2     | 0       | 3     | 3    |
| грати   | 2     | 2       | 0     | 3    |
| м'яч    | 1     | 2       | 2     | 0    |

На основі матриці сили шляхів визначаються фінальні позиції кандидатів:

1. "спорт": виграє у всіх інших.
2. "грати": сильніший за "команда" та "м'яч".
3. "команда": слабший за "спорт" та "грати".
4. "м'яч": слабший за всіх інших.

В наступних абзацах описано переваги використання саме методу Шульце для вирішення поставленої задачі при розробці моделі мультиагентної системи. Цей метод є потужним алгоритмом для визначення найкращих кандидатів серед множини можливих варіантів. Його основна перевага

полягає у тому, що він дозволяє не просто підрахувати голоси, а враховує взаємні преференції кожного елемента, що дає змогу отримати узгоджений результат. Це особливо важливо в системах, де рішення приймається на основі колективного аналізу, наприклад, при формуванні оновленого словника в мультиагентній системі.

Простий підхід до голосування, такий як підсумовування балів, має кілька недоліків. Наприклад, якщо одне слово отримало більше голосів, ніж інші, це не означає, що воно є найкращим вибором. Таке слово може набрати більшість тільки через локальну популярність у певних голосах, але бути слабшим у загальному контексті. Метод Шульце дозволяє оцінити силу кожного слова в порівнянні з іншими словами та визначити його реальну перевагу.

Однією з ключових особливостей методу є використання парних порівнянь. Замість того, щоб просто рахувати кількість голосів, алгоритм аналізує, наскільки часто один кандидат домінує над іншим у множині голосувань. Таким чином, якщо якесь слово стабільно перемагає у більшості парних порівнянь, воно отримує вищий рейтинг, навіть якщо загальна кількість голосів у нього менша.

Ще однією важливою перевагою є врахування транзитивних преференцій. Якщо слово А краще за В, а В краще за С, метод Шульце враховує цей зв'язок і може вивести, що А також повинно бути кращим за С, навіть якщо ці два кандидати безпосередньо не порівнювалися. Такий підхід дозволяє вибудовувати стабільну структуру голосування, усуваючи парадокси, які можуть виникати у звичайному голосуванні. Крім того, метод Шульце є стійким до змін у кандидатах. У випадку, якщо певний кандидат



буде виключений із голосування, результат не зміниться кардинально, оскільки алгоритм продовжить аналізувати залишкові зв'язки між іншими словами. У звичайному голосуванні видалення одного кандидата може повністю змінити результати, що може призвести до нестабільних рішень. Метод також ефективно вирішує проблему циклічних преференцій. У класичних системах голосування може виникнути ситуація, коли вибір агентів суперечливий: один агент голосує за порядок  $A > B > C$ , інший  $B > C > A$ , а ще один  $C > A > B$ . В результаті утворюється цикл без очевидного переможця. Метод Шульце знаходить найкращі варіанти, аналізуючи всі можливі шляхи переваг.

Завдяки цим характеристикам метод Шульце є найкращим вибором для визначення найбільш релевантного оновленого словника. Він дозволяє не лише узгодити вибір агентів, а й сформувати більш стабільний та надійний набір слів, що відповідає загальним тенденціям в оброблюваних текстах.

Отже, в даному розділі були описані загальний процес голосування в мультиагентних системах і метод Шульце, який доцільно використовувати саме до випадку ранжування нового набору слів в словнику, визначена його модифікація під дану конкретну задачу.

## **2.6. Висновки.**

- В даному розділі описані загальні підходи для проєктування мультиагентних систем а також – визначена методологія розробки таких систем, описані загальні точки зору. Як обґрунтовано в Розділах 3 та 4 це є доцільно для ефективної побудови моделі МАС оброблення слабоструктурованих текстових даних.

- На основі запропонованої моделі для аналізу великих масивів даних розроблена прикладна модель для вирішення задачі автоматичної побудови словника предметної області. Теоретична модель з розділу 2.2 адаптована під практичну реалізацію. Описана робота агентів, проаналізована робота моделі з використанням діаграм послідовностей та розгортання UML.
- Розроблений протокол спілкування агентів з використанням стандарту FIPA та мови комунікації агентів ACL, опис кожного акту наведений в додатку.
- Проведений порівняльний аналіз програмних бібліотек та мов для розробки мультиагентних систем, для задачі оброблення слабоструктурованих текстових даних обрано бібліотеку PADE, переваги детально описані в розділі 2.3.
- Запропонована модель мультиагентної системи для вирішення задачі автоматичної побудови словника предметної області, була внесена модифікація до загальної моделі мультиагентної системи обробки великих наборів текстових даних. Для комунікації агентів був розширений набір запитів на мові ACL, які стосуються вирішення поточної задачі.
- В результаті роботи кожного програмного агента з'являється локальний словник, який необхідно в подальшому об'єднати з локальними словниками інших агентів. Відповідно, були досліджені методи голосування, в тому числі – методи голосування агентів та запропонований результуючий метод, який і реалізований в даному дослідженні.

## **РОЗДІЛ 3 МЕТОДИ ФІЛЬТРАЦІЇ, КЛАСИФІКАЦІЇ ТА АНАЛІЗУ ПОТОКІВ ТЕКСТОВИХ ДАНИХ В МОДЕЛІ МАС.**

В даному розділі розглядається задача багатокласової класифікації поточкових текстових даних, що означає, що на вхід системи подаються постійно нові і нові тексти, які варто класифікувати. В подальшому ці тексти, як зазначено в попередньому розділі, передаються на рішення задачі автоматичної побудови словника предметної області.

В першу чергу розглянуті теоретичні відомості та визначені напрямки для вдосконалення та пропозиції роботи моделі. Це буде описано в розділі 3.1. Розділ 3.2 описує запропоновану модифікацію фільтра Блума для можливості багатокласової фільтрації потоків текстових даних. В розділі 3.3 розглядається розроблена модель нейронної мережі для задачі класифікації текстів.

В розділі 3.4 розглянуті основні теоретичні відомості, що стосуються саме методів побудови словників, їх недоліки, переваги та можливості для вдосконалення. Для вирішення задачі автоматичної побудови словника, на вхід системи подається текст та його клас. Це означає, що підсистема, описана в даному розділі, має виконати ряд задач: видати відповідному агенту завдання, створити нових агентів у випадку високої завантаженості, агентом має бути визначене локальне оновлення до словника даної області, а після комунікації агентів – мають бути запропоновані зміни в загальний словник.

### **3.1.      Задача фільтрації, класифікації та автоматизованої побудови словників при обробці потоків текстів.**

У більшості традиційних алгоритмів передбачається, що аналізу піддається якась база даних. Тобто коли виникає потреба в даних, вони є в наявності. У цьому розділі приймається інше припущення: дані надходять у

вигляді одного або декількох потоків (наприклад, проактивні агенти шукають дані та мають швидко визначити, чи варто їх надіслати до системи для подальшої обробки) і, якщо не обробити або не зберегти їх негайно, то вони пропадуть назавжди. Більш того, ми припускаємо, що дані надходять так швидко, що практично нереально зберегти їх всі в активному сховищі (традиційній базі даних), та обробляти пізніше, коли буде зручно.

Всі алгоритми обробки потоків мають на увазі те чи інше узагальнення даних. Ми почнемо з питання про те, як зробити корисну вибірку з потоку і як відфільтрувати потік, виключивши більшість «небажаних» елементів. Потім ми покажемо, як оцінити число різних елементів в потоці, використовуючи набагато менше пам'яті, ніж треба було б для зберігання списку всіх надійшовших елементів [12].

Інший підхід до узагальнення потоку полягає в дослідженні тільки «вікна» фіксованого розміру, що містить останні  $n$  елементів, де  $n$  зазвичай велике число. Якщо потоків багато і / або  $n$  велике, то ми не зможемо зберегти вікна цілком для кожного потоку, тому узагальнювати доведеться навіть вікна. Ми вирішимо фундаментальну проблему підрахунку приблизного числа одиниць у вікні бітового потоку, використовуючи набагато менше пам'яті, ніж треба було б для зберігання всього вікна.

За аналогією з системою управління базами даних ми можемо розглядати потоковий процесор як свого роду систему управління даними. У систему може входити будь-яке число потоків. Кожен потік може постачати елементи за своїм розкладом; і темп передачі, і типи даних можуть бути різні, часом між надходженням елементів одного потоку теж не обов'язково однакову. Той факт, що швидкість надходження елементів потоку не

контролюється системою, відрізняє обробку потоків від обробки даних з СУБД, коли система контролює швидкість читання даних з диска і, отже, може не турбуватися про те, що дані загубляться, поки вона буде виконувати [12].

Перш ніж переходити до обговорення алгоритмів, розглянемо, які обмеження накладаються при роботі з потоками. Перш за все, потоки часто приносять елементи дуже швидко. Ми повинні обробляти елементи в реальному масштабі часу, потім такої можливості вже не буде (хіба що звернутися до архівного сховища). Тому важливо, щоб алгоритм обробки потоку виконувався цілком в оперативній пам'яті, взагалі або майже не звертаючись до зовнішніх пристроїв [12].

Тому багато завдань, які стосуються поточкових даних, було б легко вирішити, якщо є достатньо пам'яті, але, коли потрібно обробляти потоки реальної частоти на машині реального розміру, завдання стає важкою, і для її вирішення доводиться винаходити нові методи. Ось два загальних зауваження про поточкові алгоритми, які варто мати на увазі при читанні цієї глави:

- часто можна набагато ефективніше отримати наближене, а не точне рішення;
- як і в розділі 3, можуть виявитися корисними різні методи, пов'язані з хешем. Взагалі кажучи, ці методи вносять корисну випадковість в поведінку алгоритму, що дозволяє отримати наближену відповідь, дуже близьку до істинної.

Очевидний підхід полягає в тому, щоб генерувати випадкове ціле число від 0 до 9 при отриманні кожного пошукового запиту. Кортеж зберігається тоді і тільки тоді, коли це число дорівнює 0. У цьому випадку буде збережена в середньому  $1/10$  всіх запитів кожного користувача. Через статистичні

особливості дані будуть трохи зашумлені, але якщо користувач відправляє багато запитів, то згідно із законом великих чисел частка його збережених запитів буде близька до  $1/10$ . Однак при такій схемі ми отримаємо невірну відповідь на питання про середню кількість повторюваних запитів для кожного користувача. Припустимо, що в минулому місяці користувач відправив  $s$  запитів по одному разу,  $d$  запитів по два рази і жодного запиту більше двох разів. Якщо взяти вибірку, що містить  $1/10$  всіх запитів даного користувача, то ми побачимо в ній очікувані  $s/10$  запитів, надісланих по разу. З  $d$  запитів, відправлених двічі, в вибірку потрапить тільки  $d/100$ ; ця величина дорівнює  $d$ , помножене на ймовірність того, що обидва запити потраплять до вибірки обсягом  $1/10$ . З запитів, які зустрілися в повному потоці двічі,  $18d/100$  будуть присутні рівно один раз. Зауважимо, що  $18/100$  - це ймовірність того, що один з двох запитів виявиться в обраній  $1/10$  частини потоку, а інший - в невибраній частині, що становить  $9/10$ .

Правильна відповідь на питання про частку повторюваних пошукових запитів -  $d/(s + d)$ . Однак на основі вибірки ми отримуємо відповідь  $d/(10s + 19d)$ . Зауважимо, що  $d/100$  запитів зустрічаються у вибірці двічі, а  $s/10 + 18d/100$  - один раз. Отже, частка запитів, які зустрілися в вибірці двічі, дорівнює  $d/100$ , поділене на  $d/100 + s/10 + 18d/100$ , тобто  $d/(10s + 19d)$ . За жодних позитивних значеннях  $s$  і  $d$  не може мати місце рівність  $d/(s + d) = d/(10s + 19d)$  [12].

Відзначимо, що ми не зберігаємо користувача в комірку, там взагалі не зберігаються дані. Хеш-функція використовується як генератор випадкових чисел, але має важливу додаткову властивість: якщо застосувати її до одного і того ж користувача кілька разів, то всякий раз будемо виходити те ж саме «випадкове» значення. Таким чином, ніде не зберігаючи ознаку входить-не

входить, ми можемо реконструювати її при отриманні кожного запиту від даного користувача. Більш загально, ми можемо отримати вибірку, що містить частку запитів користувачів, що дорівнює будь-якому раціональному числу  $a/b$ . Для цього потрібно хешувати імена користувачів в  $b$  комірки з номерами від 0 до  $b - 1$  і включати запит до вибірки, якщо хеш-код менше  $a$  [12].

Важливий процес, який часто застосовується до потоків – фільтрація. Ми хочемо відбирати тільки кортежі, що задовольняють деякому критерію. Кортежі що пройшли перевірку передаються потоком іншому процесу, інші відкидаються. Якщо критерієм є обчислювальна властивість кортежу (наприклад, перша компонента менше 10), то зробити фільтрацію легко. Завдання ускладнюється, коли критерій має на увазі пошук в деякій множині, особливо, якщо постійно не поміщається в оперативній пам'яті. У цьому розділі ми обговоримо метод «фільтра Блума», що дозволяє виключити більшість кортежів, що не відповідають критерію.

Фільтр Блума складається з наступних частин:

1. Масив  $n$  біт, спочатку рівних 0.
2. Набір хеш-функцій  $h_1, h_2, \dots, h_k$ . Кожна функція відображає значення «ключів» в  $n$  комірок, відповідних  $n$  бітів масиву.
3. Множина  $S$ , що містить  $m$  ключів.

Призначення фільтру Блума - пропускати всі елементи потоку, ключі яких належать  $S$ , і відкидати більшість елементів з ключами, що не належать  $S$ . Ініціалізувавши бітовий масив, зануляємо всі біти. Застосуємо до кожного ключу в  $S$  кожен з  $k$  хеш-функцій. Встановимо в 1 біти, номери яких збігаються з  $h_i(K)$  для деякої хеш-функції  $h_i$  і деякого ключа  $K$  з  $S$  [12].

При обробці ключа  $K$ , що надійшов з потоку, перевіряємо, що значення всіх бітів з номерами  $h_1(K)$ ,  $h_2(K)$ , ...,  $h_k(K)$  рівні 1. Якщо це так, пропускаємо елемент. Якщо хоча б один біт дорівнює 0, то ключ  $K$  не може належати  $S$ , тому відкидаємо елемент.

Якщо значення ключа належить  $S$ , то елемент, безумовно, буде пропущений фільтром Блума. Але елемент може пройти і тоді, коли його ключ відсутній в  $S$ . Нам потрібно зрозуміти, як виразити ймовірність хибно позитивному результату у вигляді функції від довжини бітового масиву  $n$ , кількості  $m$  елементів множини  $S$  і кількості хеш-функцій  $k$  [12].

В якості моделі скористаємося киданням дротиків в мішені. Нехай є  $x$  мішеней і  $y$  дротиків. Кожен дротик з однаковою ймовірністю вражає будь-яку мішень. Яке математичне сподівання кількості мішеней, уражених хоча б один раз, після кидання всіх дротиків:

- Вірогідність, що даний дротик не влучить у дану мішень, дорівнює  $(x - 1) / x$ .
- Вірогідність, що жоден з  $y$  дротиків не влучить у дану мішень, дорівнює  $((1 - x) / x)^y$ . Цей вислів можна переписати у вигляді  $(1 - 1 / x)^{x(y / x)}$ .
- Скориставшись апроксимацією  $(1 - \varepsilon)^{1 / \varepsilon} = 1 / e$  для малих (див. Розділ 1.3.5), робимо висновок, що вірогідність того, що жоден з  $y$  дротиків не влучить у дану мішень, дорівнює  $e^{-y / x}$ .

Це правило можна застосувати і до більш загальної ситуації, коли множина  $S$  містить  $m$  елементів, масив складається з  $n$  бітів і є  $k$  хеш-функцій. Кількість мішеней  $x = n$ , а кількість дротиків -  $y = km$ . Отже, ймовірність того, що якийсь біт залишиться рівним 0, дорівнює  $e^{-km / n}$ . Ми хочемо, щоб частка нулів була досить великою, інакше ймовірність того, що елемент, який не



належить  $S$ , хоча б один раз буде хешований в 0, виявиться занадто маленькою, а, значить, буде занадто багато хибно позитивних результатів. Наприклад, можна вибрати кількість хеш-функцій  $k$  рівним  $n / t$  або менше. Тоді ймовірність отримати 0 не менше  $e^{-1}$ , або 37%. У загальному випадку ймовірність хибно позитивного результату, тобто ймовірність отримати одиничний біт, дорівнює  $(1 - e^{-km/n})^k$ .

Відзначимо, що при читанні потоку необов'язково зберігати елементи. В оперативній пам'яті потрібно зберігати тільки одне ціле число на кожну хеш-функцію - поточну максимальну хвостову довжину елементів потоку для даної функції. Якщо обробляється всього один потік, то можна використовувати мільйони хеш-функцій, а це набагато більше, ніж необхідно для отримання близької оцінки. І лише якщо одночасно обробляється багато потоків, то обсяг оперативної пам'яті починає обмежувати кількість функцій, асоційованих з кожним. Втім, на практиці кількість хеш-функцій обмежена, скоріше, часом обчислення хеш-кодів елементів потоку [12].

У класифікації тексту ми отримуємо опис  $d \in X$  документа, де  $X$  – простір документа;  $i$  фіксований набір класів  $C = \{c_1, c_2, \dots, c_J\}$ . Класи також називаються категоріями або мітками. Як правило, класифікація виконується людиною для потреб програми, як у прикладах Китаю та документах, які говорять про багатоядерні комп'ютерні чіпи вище. Маємо навчальний набір  $D$  з позначеними документами  $(d, c)$ , де  $(d, c) \in X \times C$ . Наприклад:  $(d, c) = (\text{Beijing joins the World Trade Organization, China})$  для документа з одним реченням (Пекін приєднується до Світової організації торгівлі) та класу (або ярлика). Потім, використовуючи метод навчання або алгоритм навчання, ми хочемо вивчити класифікатор або функцію класифікації  $\gamma$ , яка відображає документи в класи:  $\gamma: X \rightarrow C$ . Цей тип навчання називається навчанням «з вчителем»,

оскільки керівник (людина, яка визначає класи та позначає навчальні документи) виконує роль вчителя, який керує процесом навчання. Позначаємо контрольований метод навчання через  $\Gamma$  і записуємо  $\Gamma(D) = \gamma$ . Метод навчання  $\Gamma$  приймає навчальний набір  $D$  як вхідний сигнал і повертає вивчену функцію класифікації  $\gamma$ .

Для аргументації мотивації використання фільтру Блума в комбінації разом з нейронною мережею було проведено ряд досліджень, включаючи наступне [49]. Додатково, нижче приведено графік середнього часу обробки тексту з використанням кожного з цих методів.

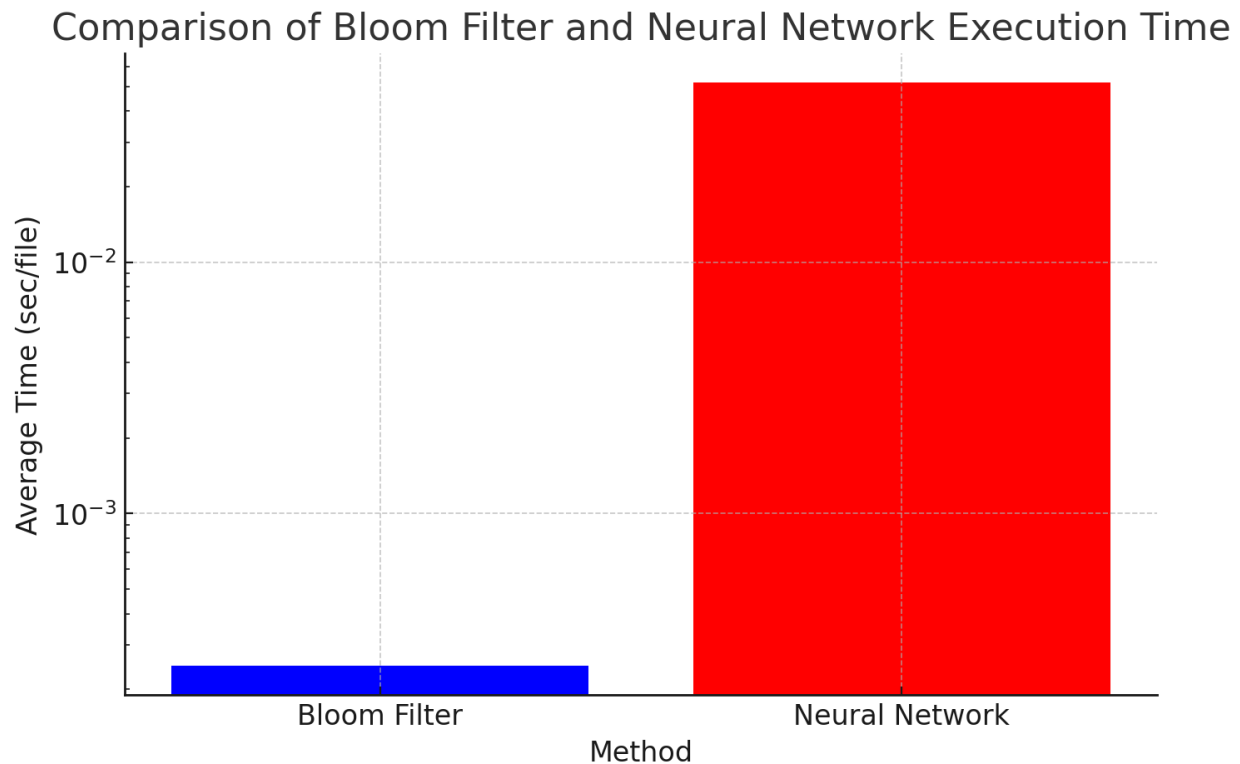


Рис. 3.1. Порівняння часу фільтрації та класифікації тексту з використанням фільтру Блума і нейронної мережі відповідно [власна розробка]

На графіку видно, що час фільтрації файлу є значно нижчим, ніж час класифікації. Відповідно, мотивація використання комбінованого підходу є обґрунтованою.

Варто зазначити, що фільтр Блума має недолік – він застосовується лише для бінарної класифікації, але має й перевагу – може працювати з текстовими даними. Відповідно, в даному розділі вирішені дві задачі:

- модифікація фільтру Блума для можливості багатокласової класифікації текстів;
- використання нейронних мереж як швидкого класифікатора для багатокласової класифікації текстів;
- використання методів для автоматизованої побудови словників предметної області.

### **3.2. Модифікація фільтру Блума для швидкої багатокласової фільтрації текстів.**

#### **3.2.1. Опис модифікації моделі фільтру Блума для багатокласової фільтрації.**

Основною метою є створення моделі з використанням фільтру Блума для інтелектуальної обробки поточкових текстових даних, а саме – для вирішення задачі багатокласової класифікації текстів, які надходять у реальному часі. Оцінка моделі відбувається за точністю класифікації, швидкістю навчання моделі, кількістю використаної пам'яті та швидкістю видачі результату класифікації. Складові частини Фільтру Блума:

1. Масив  $n$  біт, спочатку рівних 0.
2. Набір хеш-функцій  $h_1, h_2, \dots, h_k$ , кожна з яких відображає значення «ключа» в  $n$  комірок, відповідно до  $n$  бітів масиву.
3. Множина  $S$ , що містить  $m$  ключів.

Призначення фільтру Блума – пропускати всі елементи потоку, ключі яких належать  $S$ , і відкидати більшість елементів з ключами, що не належать  $S$  [12].

Спочатку створюємо бітовий масив, обнуливши всі біти. Застосуємо до кожного ключу в  $S$  кожен з  $k$  хеш-функцій. Встановимо в 1 біти, номери яких збігаються з  $h_i(K)$  для деякої хеш-функції  $h_i$  і деякого ключа  $K$  з  $S$  [12].

При обробці ключа  $K$ , що надійшов з потоку, перевіряємо, що значення всіх бітів з номерами  $h_1(K)$ ,  $h_2(K)$ , ...,  $h_k(K)$  рівні 1. Якщо це так, то пропускаємо елемент. Якщо хоча б один біт дорівнює 0, то ключ  $K$  не може належати  $S$ , тому відкидаємо елемент [12].

Додавання елемента в фільтр відбувається наступним чином - для значення вираховується кілька різних хешів, після чого виставляються відповідні біти в бітовому полі. Перевірка на входження здійснюється схожим чином - спочатку вираховуються хеші для значення, після чого перевіряється, чи виставлені відповідні біти. Так як різні значення можуть мати однакові хеш-коди, то можливі хибнопозитивні спрацьовування. Але якщо хеш-коди різні, то, однозначно, значення теж різні, тому помилково-негативних спрацьовувань буде менше зі збільшення кількості хеш-функцій.

В загальному випадку ймовірність отримання хибнопозитивного результату можна обчислити за формулою:

$$(1 - e^{-km/n})^k \text{ [12]}$$

З формули видно, що для зменшення ймовірності отримання хибнопозитивного результату потрібно збільшити кількість хеш-функцій  $k$  або кількість бітів масиву  $n$ .

Нижче описаний фільтр, який складається з 16 біт і працює з двома хеш-функціями.

Додається слово "Football", перша функція дає хеш код, рівний 25425, а друга 894346520. Так як за умовою доступно всього 16 біт, тому знаходиться значення по модулю 16. Перший хеш рівний 1, а другий - 8. На відповідних позиціях виставляється значення «1».

Для видалення одного значення доведеться перераховувати хеші всіх інших значень до того часу, поки хоча б одне зі значень не дасть ті ж біти, або поки не перерахуємо всі значення.

Потім перевіряється, чи проходить слово "Nation" даний фільтр. Вираховується для цього слова хеш-значення, для прикладу визначимо, що вони рівні 3 і 7 відповідно. Біт 3 не виставлений (дорівнює нулю), тому однозначно, цей рядок не входить у фільтр. Візьмемо інше слово "Sky", його хеші мають значення 1 і 8, хоча даного значення у фільтрі немає, фільтр пропустить дане слово, що і дає нам хибнопозитивне значення.

Нашою задачею є класифікація текстів у вигляді потоків даних. Для прикладу будемо виконувати класифікацію на 2 класи з подальшим розширенням. Також буде показано спосіб, що дозволяє вдосконалити модель в режимі потокової обробки даних.

Першим етапом є навчання. Для цього візьмемо по 1 тексту за еталон до кожного класу на якому буде проведено перше навчання моделі. Навчання буде відбуватися у 2 етапи:

1. Попередня обробка тексту
2. Побудова фільтра Блума

Попередня обробка тексту складається з наступних етапів:

1. Переведення всіх слів в нижній реєстр
2. Видалення чисел
3. Видалення знаків пунктуації
4. Видалення символів розділення слів
5. Видалення стоп-слів та слів, які не мають змістового навантаження
6. Токенізація
7. Стеммінг
8. Лематизація

Після проведення попередньої обробки ми отримаємо масив слів без зайвих символів та у простій формі. Наступним етапом є створення фільтру Блума використавши декілька хеш-функцій. Після виставлення бітів в 1 для кожного слова з масиву для кожної хеш-функції, навчання моделі можна вважати завершеним.

Варто не використовувати надмірну кількість хеш-функцій, тому що із збільшенням кількості хеш-функцій збільшується кількість одиниць, які будуть виставлені в масиві бітів, а це збільшує кількість хибнопозитивних результатів. Якщо всі біти будуть рівні одиниці, то дана модель не буде працювати, бо кожне слово буде давати позитивний результат.

Далі необхідно виконати навчання моделі, в даному випадку, для двох класів. Можна створити необхідну кількість класів використавши допустиму кількість пам'яті. Але варто пам'ятати й про хибнопозитивні значення.

Так, наприклад, якщо доступно 100 Мб оперативної пам'яті і необхідно створити 4 класи при розмірі обробленого масиву в 10 000 слів. Тобто дані

будуть міститись у 800 000 бітів, які розділені на 4 класи по 200 000 біт у кожному. Використавши 3 хеш-функції ми отримаємо наступне значення у відсотках хибнопозитивного результату для кожного зі слів:

$$(1 - e^{-km/n})^k = (1 - e^{-3 * 10\,000 / 200\,000})^3 = (1 - 0.861)^3 = 0.003, \text{ тобто } 0.3\%$$

Але якщо при тих самих умовах потрібно розбити на 40 класів, то вийде 20 000 бітів на кожен клас і ми отримаємо наступні результати:

$$(1 - e^{-km/n})^k = (1 - e^{-3 * 10\,000 / 20\,000})^3 = (1 - 0.223)^3 = 0.469, \text{ тобто } 46.9\%$$

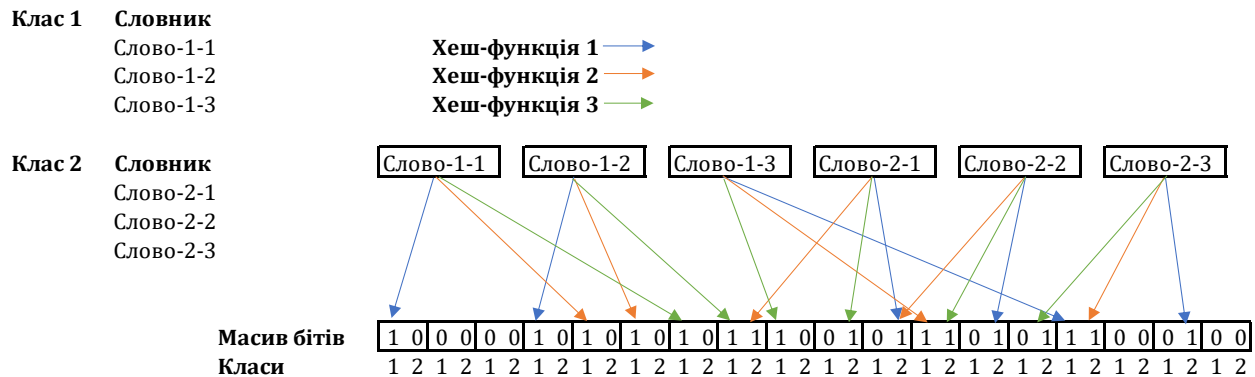


Рис 3.2. Схематичне представлення модифікації фільтра Блума [власна розробка]

Як видно з прикладів всі параметри фільтра Блума тісно пов'язані і необхідно доцільно розраховувати можливості системи. Після навчання обох класів ми отримаємо два фільтра Блума для кожного класу. Наступною задачею є логіка приналежності вхідного тексту до відповідного класу. Одним з методів для класифікації тексту до одного з класів є підрахунок відсотка слів з вхідного тексту, у яких значення всіх хеш-функцій дорівнює «1». Даний вираз, позначимо його  $P$ , можна представити у наступному вигляді:

$$P = \frac{\sum_{i=1}^m h_1(S[i])h_2(S[i]) \dots h_k(S[i])}{m} \text{ (формула 3.1.)},$$

де  $S$  – масив слів,  $m$  – кількість слів,  $h$  – хеш-функція.

Робота моделі складається з наступних етапів:

1. Попередня обробка вхідного тексту
2. Розрахунок  $P$  для кожного з класів
3. Порівняння  $P$  кожного з класів
4. Видача результату

Попередня обробка вхідного тексту відбувається таким самим чином, як і при навчанні моделі. Розрахунок  $P$  для кожного з класів може відбуватися паралельно, що значно прискорить роботу класифікатора. На виході ми отримаємо результат приналежності даного тексту до одного з класів або, при рівності  $P$ , рівну приналежність до обох класів. Схема роботи моделі представлена на рисунку 3.3.

Завершальним етапом у побудові системи є обробка даних у реальному часі. Для вирішення цієї задачі доцільно використати чергу повідомлень, до якої вхідні дані будуть поступати з різних ресурсів та записуватись у чергу. А сервіс для класифікації буде звертатися до черги повідомлень та брати всі непрочитані текстові дані та класифікувати їх. З повною схемою роботи системи можна ознайомитись в роботі [34].

В першу чергу навчається модель для кожного з класів на наборі текстових даних. На виході буде масив бітів фільтра Блума, які будуть використовуватися системою для підрахунку кількості слів, які проходять даний фільтр. Донавчання моделі буде відбуватися лише у випадку, коли вхідний текст належить до одного з класів та задовольняє критерію належності



класу та не буде відбуватися, коли вхідний текст однаково належить кожному з класів. Так як у даному випадку неможливо точно визначити, модель якого з класів донавчати.

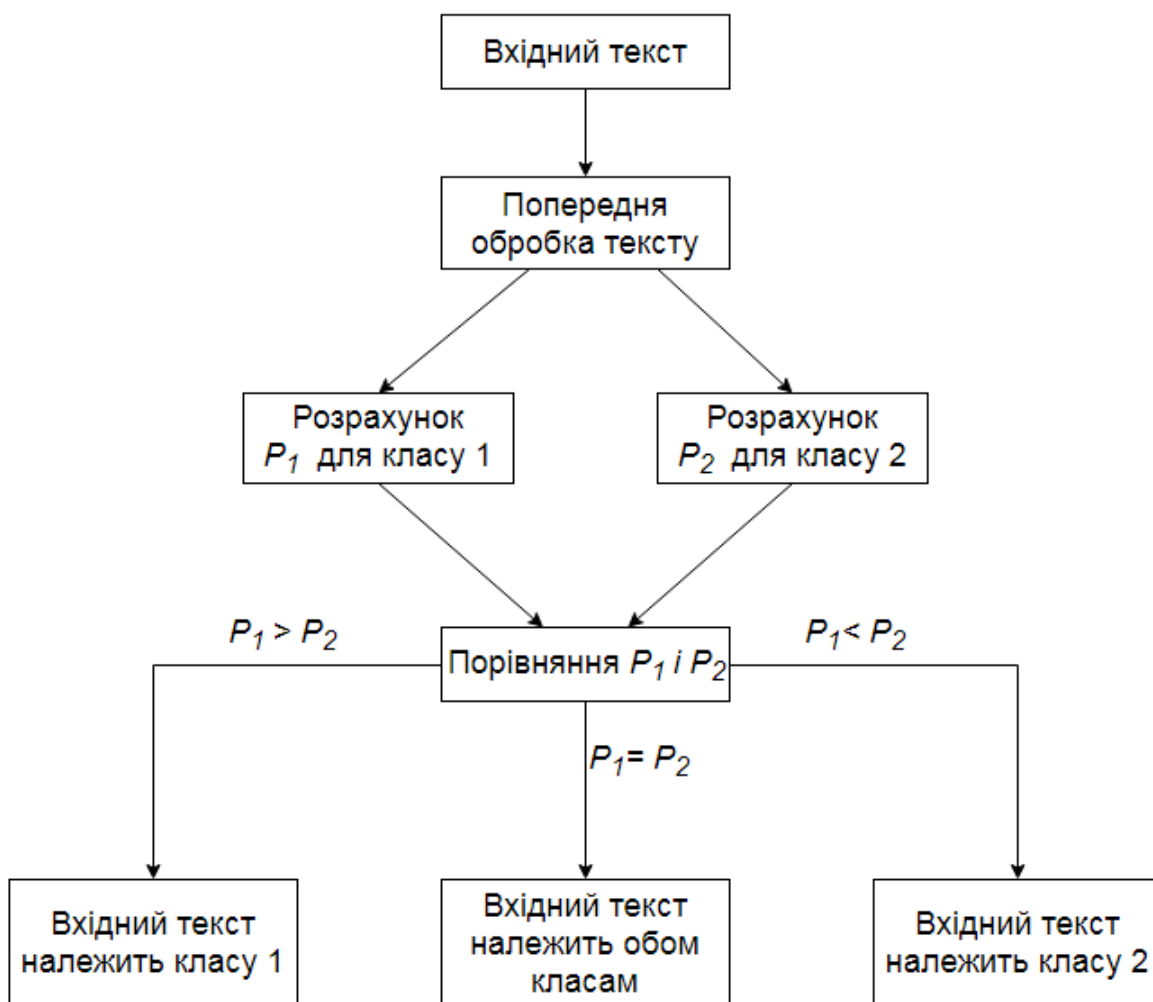


Рис 3.3. Алгоритм роботи системи [власна розробка]

Фільтри Блума мають значні переваги над префіксними деревами, особливо у задачах обробки великих обсягів даних, таких як класифікація тексту чи перевірка належності слів до словника. Насамперед, вони набагато ефективніше використовують пам'ять, оскільки потребують лише фіксований розмір бітового масиву та кілька хеш-функцій, незалежно від кількості слів чи

їхньої довжини. Крім того, фільтри Блума ідеально підходять для задач класифікації, оскільки забезпечують просту реалізацію перевірки членства у множині, тоді як префіксне дерево може ускладнити розв'язання таких задач через необхідність обробки перекриття слів чи логіки обходу дерева. Ще однією перевагою є здатність фільтра Блума масштабуватися для роботи з великими словниками без значного збільшення пам'яті, оскільки його розмір залишається сталим, тоді як префіксне дерево потребує значних оптимізацій для підтримки ефективності. Нарешті, фільтри Блума дозволяють регулювати баланс між використанням пам'яті та допустимим рівнем хибнопозитивних результатів через налаштування розміру бітового масиву та кількості хеш-функцій, що неможливо у випадку префіксних дерев, які не можуть жертвувати точністю заради ефективності. Проте, якщо для задачі критично важлива точна відповідність із нульовою ймовірністю хибнопозитивних результатів, або якщо потрібно виконувати специфічні операції, такі як автозавершення чи пошук за префіксами, префіксні дерева можуть бути кращим вибором, особливо для невеликих словників чи у разі, коли обмеження пам'яті не є вирішальними.

Перевірка членства у фільтрі Блума виконується за сталий час  $O(k)$ , де  $k$  — кількість хеш-функцій, що фактично можна вважати близьким до  $O(1)$ . У той же час, пошук у префіксному дереві має складність  $O(\log_2 N)$ , де  $N$  — кількість вузлів, або  $O(m)$ , де  $m$  — довжина слова. Це робить префіксні дерева менш придатними для роботи з великими словниками, оскільки час пошуку зростає зі збільшенням словника або довжини слів.

Крім швидкості, фільтр Блума є значно більш ефективним у використанні пам'яті. Він потребує лише фіксований розмір бітового масиву, тоді як пам'ять, потрібна для префіксного дерева, зростає пропорційно до

кількості слів та символів, адже зберігаються всі шляхи дерева. Формули для оцінки пам'яті  $M$  виглядають так:

$$M = \frac{-n \ln(p)}{\ln 2^2} \text{ (формула 3.2).}$$

де  $p$  — прийнятний рівень хибнопозитивних результатів. Враховуючи час пошуку та пам'ять, фільтр Блума стає вигіднішим, коли  $N$  (кількість слів у словнику) зростає до значень, за яких логарифмічний час пошуку в префіксному дереві перевищує сталий час фільтру Блума, а обсяг пам'яті, потрібний для дерева, стає значно більшим за розмір бітового масиву. Для великих словників і задач, де допускається невелика ймовірність хибнопозитивних результатів, фільтр Блума є кращим вибором, забезпечуючи оптимальний баланс між швидкістю і пам'яттю.

Підводячи підсумки, нижче описаний модифікований алгоритм застосування фільтру Блума в даній моделі мультиагентної системи.

### **Крок 1: Ініціалізація структури**

1.1. Створення бітового масиву: Фільтр Блума ініціалізується як масив із  $m$  елементів, де кожен елемент є цілим числом (наприклад, байт або  $N$ -бітове значення), а не одним бітом.

1.2. Визначення хеш-функцій: Обирається  $k$  незалежних хеш-функцій:  $h_1, h_2, \dots, h_k$ , кожна з яких приймає слово як вхід і повертає індекс у діапазоні від 0 до  $m-1$ .

1.3. Задається кількість класів: Фільтр підтримує  $C$  класів. Для позначення класів у кожному елементі використовується бітове представлення (наприклад, для 4 класів: 0001, 0010, 0100, 1000).

### **Крок 2: Навчання фільтра**

2.1. Формування вхідного словника: Кожен елемент навчального словника має формат {слово: клас}, де клас позначається бітовим кодом відповідного класу.

2.2. Хешування слова: Для кожного слова  $w$  з навчального словника обчислюються його хеші  $h_1(w)$ ,  $h_2(w)$ , ...,  $h_k(w)$ , де кожен хеш повертає індекс у масиві.

2.3. Оновлення бітового масиву: У позиціях  $h_1(w)$ ,  $h_2(w)$ , ...,  $h_k(w)$  бітові значення елемента оновлюються за допомогою бітової операції АБО (OR), щоб додати бітове представлення класу.

### **Крок 3: Перевірка належності тексту до класу**

3.1. Хешування слів тексту: Для кожного слова  $w$  у тексті, що перевіряється, обчислюються його хеші  $h_1(w)$ ,  $h_2(w)$ , ...,  $h_k(w)$ .

3.2. Перевірка бітів для класу: Для кожної позиції у бітовому масиві, отриманій з хеш-функцій слів тексту, перевіряється, чи встановлено біти, що відповідають класу, що перевіряється.

3.3. Підрахунок відповідностей: Підраховується кількість слів у тексті, для яких на всіх хеш-позиціях встановлено біти, що відповідають класу.

3.4. Порівняння з порогом (threshold): Якщо загальна кількість слів у тексті, що відповідають класу, перевищує заданий поріг  $threshold$ , текст вважається таким, що належить до класу.

3.5. Підсумковий результат: Для кожного класу обчислюється кількість відповідних слів у тексті. Текст може належати одному або кільком класам залежно від перевищення порогового значення.

Рис. 3.4. Алгоритм застосування фільтру Блума в запропонованій моделі MAC [власна розробка]

### **3.2.2. Оптимізація параметрів фільтра Блума за допомогою генетичного алгоритму**

Генетичний алгоритм (ГА) є ефективним методом оптимізації в задачах, де простий повний перебір параметрів є надто довгим на заданих обчислювальних потужностях. У випадку налаштування мультиагентної системи для автоматизованої побудови словника предметної області кількість можливих конфігурацій (розмір фільтра Блума, кількість хеш-функцій, тип хешування, порогові значення класифікації тощо) зростає експоненційно. Повний перебір вимагав би перевірки всіх можливих комбінацій, що є непрактичним для реального застосування через значні витрати часу та обчислювальних ресурсів.

Натомість генетичний алгоритм дозволяє ефективно знаходити оптимальні параметри завдяки принципам природного відбору, мутації та кросоверу. Використовуючи початкову популяцію можливих рішень, ГА ітеративно вдосконалює їх шляхом відбору найкращих особин, комбінування їхніх характеристик та введення випадкових змін, що забезпечує баланс між експлуатацією найкращих поточних рішень та дослідженням нових можливостей. Такий підхід особливо ефективний у задачах, де функція пристосованості є складною або багатовимірною, а пошук глобального оптимуму неможливо виконати через високі обчислювальні витрати.

Додатковою перевагою ГА є його здатність знаходити близько-оптимальні рішення навіть у разі неповного покриття всіх можливих комбінацій параметрів. Це дозволяє швидко адаптувати систему до нових умов, що є критично важливим у динамічному середовищі обробки текстових даних. Таким чином, застосування генетичного алгоритму виправдане в контексті даного дослідження, оскільки дозволяє оптимізувати параметри

фільтра Блума та інших компонентів системи при значному скороченні обчислювальних витрат порівняно з методами повного перебору або випадкового пошуку.

Генетичний алгоритм (ГА) використовується для оптимізації мультикласового фільтра Блума, що застосовується для класифікації текстів. Фільтри Блума є ефективними з точки зору використання пам'яті та часу, однак вони схильні до появи хибнопозитивних результатів, коли об'єкт помилково визначається як частина множини. У контексті мультикласового фільтра Блума додатковою складністю є потенційна поява хибнонегативів через використання множин для зберігання класів. Завдання алгоритму — мінімізувати ці похибки, забезпечуючи високу точність класифікації та адаптивність до різних текстових наборів.

Оскільки характеристики фільтра Блума можна представити у вигляді функції дискретної змінної, то використання даного типу алгоритмів є доцільним в даній роботі. Генетичний алгоритм базується на принципах природного відбору, де конфігурації фільтра Блума представляються як "особини". Під час ітерацій алгоритм генерує нові покоління конфігурацій шляхом оцінки "придатності", відбору найкращих рішень, схрещування "батьків" для отримання "нащадків" і внесення випадкових мутацій для підтримання різноманітності. В результаті ГА знаходить оптимальні параметри, які дозволяють ефективно використовувати фільтр Блума для класифікації текстів.

#### **Кроки генетичного алгоритму**

##### **Ініціалізація:**

- Генерація початкової популяції конфігурацій із випадковими значеннями таких параметрів:

- Розмір таблиці (size).
- Кількість хеш-функцій (num\_hashes).
- Тип хеш-функції (hash\_function).
- Поріг класифікації (threshold).
- Додаткові параметри для хеш-функцій (hash\_args).

#### **Оцінка:**

Кожна конфігурація оцінюється за допомогою функції придатності (фітнес-функції):  $Fi = A_i - FP_i - FN_i$ , де:

- $A_i$ : точність класифікації,
- $FP_i$ : кількість хибнопозитивних результатів,
- $FN_i$ : кількість хибнонегативних результатів.

#### **Відбір:**

- Сортуювання конфігурацій за значенням функції придатності.
- Вибір 50% найкращих конфігурацій для наступного покоління.

#### **Схрещування:**

- Випадкове комбінування параметрів "батьків" для створення "нащадків":
  - size, num\_hashes, hash\_function, threshold, hash\_args обираються випадково з атрибутів одного з батьків.

#### **Мутація:**

- З невеликою ймовірністю (наприклад, 10%) змінюються параметри:
  - size: змінюється в допустимому діапазоні.
  - num\_hashes: обирається нове випадкове значення.
  - hash\_function: обирається інший тип хеш-функції.
  - threshold: змінюється в межах допустимого діапазону.
  - hash\_args: змінюється значення параметра.

#### **Ітерація:**

- Повторення процесів оцінки, відбору, схрещування та мутації до досягнення термінальної умови:
  - Задана кількість поколінь.
  - Конвергенція значення функції придатності.

#### **Запропонований підхід:**

- Використання генетичного алгоритму для оптимізації мультикласового фільтра Блума, що розширює можливості цієї структури для текстової класифікації.
- Розробка адаптивної функції придатності (фітнес-функції), яка враховує баланс між точністю, хибнопозитивними та хибнонегативними результатами.
- Інтеграція різних хеш-функцій і їх параметрів у процес оптимізації.
- Забезпечення гнучкості та ефективності через генетичну еволюцію параметрів, що дозволяє адаптуватися до різних текстових наборів і доменів.

Рис. 3.5. Генетичний алгоритм для оптимізацій параметрів фільтра Блума в запропонованій моделі МАС [власна розробка]

### **3.3. Інтеграція нейронної мережі у модель мультиагентної систему для класифікації поточкових текстів.**

На сьогоднішній день є актуальним проведення досліджень у області машинного аналізу даних на базі нейромережових та ймовірнісних класифікаторів, що надають можливість перевести аналіз даних [35]. Слід зазначити високий рівень поставленого завдання, яке можна віднести до області побудови штучного інтелекту (artificial intelligence, AI), і крім того, вказати на його значну практичну цінність. Зауважимо, що на сьогоднішній день за умов розширення інструментальної бази не розроблено універсальну методологію побудови ефективних алгоритмів машинного аналізу текстових даних у режимі потокової обробки, що вказує на *актуальність даного дослідження*.

Аналіз, що було проведено у попередньому розділі, надав можливість побудувати ефективні схеми нейромережових класифікаторів, провести аналіз точності їх роботи та часу, що витрачається при обробці текстових блоків



фіксованого формату та розміру. У якості вхідних даних було застосовано текстові матеріали інтернет-видання “British Broadcasting Corporation” представлені на відкритому мережевому ресурсі [35]. Всі моделі класифікаторів проходили навчання на 1725 зразках зазначеної бази даних, а наступні 500 зразків використовувались для оцінки точності визначення одного з п’яти класів, що відповідає текстовому блоку. Обчислювальний ресурс робочої станції може бути описано через наступні показники: (i) центральний процесор 13th Gen Intel(R) Core(TM) i5-13500H 2.60 GHz, оперативна пам’ять 32 ГБ / 3200 МГц. Детальний опис даного дослідження наведений в роботі [35].

Модель глибокої нейромережі (deep neural network, DNN), що була побудована у ході експериментального дослідження складається з наступних складових: набір вхідних даних, що визначається через кількість класів, навчальних зразків та тестових зразків; токенизатор для попередньої сегментації вхідних даних відповідно середній довжині речення та загального об’єму словника; модель для розподіленого представлення слів GloVe; блок вирівнювання послідовностей (ембеддінг); модель нейромережі, що характеризується архітектурою, кількістю нейронів у кожному з прихованих шарів, функцією активації, що застосовується на кожному з прихованих шарів, оптимізатором, та функцією втрат; блок аналізу точності і швидкості класифікації даних.

Результати експериментальних досліджень наведено у табл. 1 і 2. Дослідження вказало на пріоритет застосування сигмоїдної функції активації для більшості типів архітектури DNN, у тому разі комбінації сигмоїди і ReLU. Посеред оптимізаційних алгоритмів оптимальні результати було отримано для алгоритму «Adam»; це є характерним саме для DNN, у той час як для

неглибоких нейронних мереж (shallow neural networks, SNT) оптимальні результати показує метод стохастичного градієнта (stochastic gradient descent, SGD).

*Таблиця 3.1. Оптимізація класифікатора текстових блоків на базі глибокої нейронної мережі з одним та двома повнозв'язними шарами*

| №     | функція активація | кількість епох | алгоритм оптимізації | повнозв'язні шари | кількість нейронів у шарі | точність класифікації |
|-------|-------------------|----------------|----------------------|-------------------|---------------------------|-----------------------|
| D-1.1 | сігмоїда          | 20             | Adam                 | 1                 | 5                         | 94,2%                 |
| D-1.2 | сігмоїда          | 20             | Adam                 | 2                 | 32                        | 95,5%                 |
|       | сігмоїда          |                |                      |                   | 5                         |                       |
| D-1.3 | ReLU              | 20             | Adam                 | 2                 | 32                        | 93,8%                 |
|       | сігмоїда          |                |                      |                   | 5                         |                       |

При побудові класифікатора на базі рекурентної нейронної мережі (recurrent neural networks, RNN) було використано такі механізми як вентильний рекурентний вузол (gated recurrent unit, GRU) та довга короткострокова пам'ять (long short-term memory, LSTM).

Достатній рівень точності було отримано лише для алгоритмів LSTM (табл. 3 і 4), причому експериментальне дослідження показало оптимальну розмірність ядра, як значення, збільшення якого призводить до зростання часу обробки даних, але не впливає на точність класифікації.

Таблиця 3.2. Оптимізація класифікатора LSTM

| №     | функція активація | Кількість епох | алгоритм оптимізації | розмірність LSTM | точність класифікації |
|-------|-------------------|----------------|----------------------|------------------|-----------------------|
| R-2.1 | Softmax           | 20             | Adam                 | 300              | 95%                   |
| R-2.2 | Softmax           | 20             | Adam                 | 64               | 93,2%                 |
| R-2.3 | ReLU              | 20             | Adam                 | 64               | 22%                   |
| R-2.4 | сігмоїда          | 20             | Adam                 | 64               | 95.6%                 |

Також в роботі власного авторства [35] була проведена експериментальна оцінка ефективності роботи алгоритмів класифікації, що базуються на згорткових нейромережах (convolutional neural network, CNN). На рівні аналітичного рішення задачі оптимізації процесу класифікації текстових блоків даний тип класифікаторів розглядався як ефективний. Згідно загальної схеми результат роботи кожної зі згорток мав передаватися до наступної до отримання шаблону, що, в залежності від розміру ядра згортки, міг являти групи суміжних слів (словосполучення і окремі вирази). Тим не менш, практичне дослідження показало порівняно низьку точність CNN-класифікаторів.

### **3.4. Опис роботи агентів побудови словника предметної області та обґрунтування вибору методу TF-IDF.**

В даному підрозділі описується загальна модель автоматичної побудови словників за наявними текстами, адаптація якої до розподіленого середовища описана в іншому розділі. Перш ніж будувати мультиагентну систему (або

точніше – підсистему, в контексті розв’язку загальної задачі), варто дослідити, які саме моделі і методи лежать в основі. Необхідно зробити порівняльний аналіз, вказати їх переваги та недоліки, можливості застосування. Відповідно, було проведено експеримент для визначення методів, які лежатимуть в основі моделі MAC.

На етапі попереднього дослідження визначено наступні моделі і методи, які можуть вирішити дану задачу: модель Word2Vec [36] для векторизації документа, метод частотного аналізу корпусу текстових даних TF-IDF, алгоритми розрахунку важливості слів TextRank та RAKE. Їх зображення в контексті всієї системи зображено на рис 3.6.

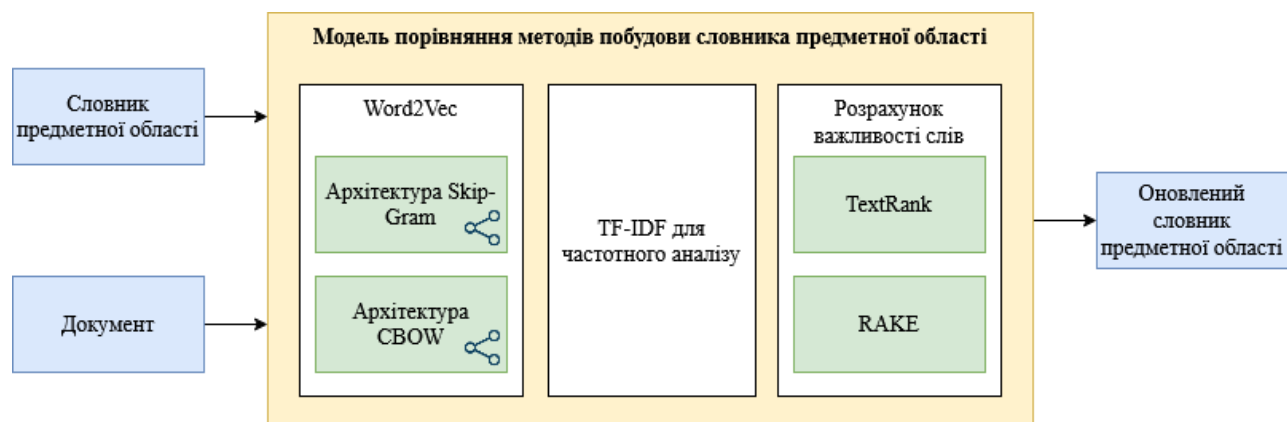


Рис 3.6. Загальний вигляд моделі побудови словника предметної області  
[власна розробка]

Як можна побачити з даного рисунку, вхідними даними в систему є не лише поточний текст, який вона має обробляти, але й поточний словник предметної області. В середині є три альтернативних методи, які зображені великими прямокутниками, при чому – в першому методі може бути дві альтернативні архітектури нейронної мережі, а в третьому підході – ще два альтернативних методи. Виходом системи є оновлений словник предметної

області. В наступних трьох розділах буде детально описано кожен з цих етапів. Метою даного розділу є аналіз існуючих альтернативних рішень. В подальшому – ці рішення мають бути порівняні та обраний один основний метод для загального вирішення задачі.

Ключовий метод, який розглядається в даному розділі, – це частота термінів, зворотна для частоти документа (англ. Term Frequency Inverse Document Frequency, скорочено – TF-IDF). Цю схему зважування можна класифікувати як статистичну процедуру, хоча її безпосередні результати мають детермінований характер. Хоча TF-IDF є відносно старою схемою зважування, вона проста і ефективна, що робить її популярною відправною точкою для інших, новіших алгоритмів [37].

TF-IDF працює шляхом визначення відносної частоти слів у конкретному документі порівняно з оберненою часткою цього слова у всьому корпусі документа. Інтуїтивно це обчислення визначає, наскільки дане слово відповідає певному документу. Слова, поширені в окремій чи невеликій групі документів, мають, як правило, більші значення TF-IDF, ніж звичайні слова, наприклад артиклі чи прийменники. Офіційна процедура впровадження TF-IDF має деякі незначні відмінності щодо всіх його застосувань, але загальний підхід працює наступним чином. Враховуючи колекцію документів  $D$ , слово  $w$  та окремий документ  $d \in D$ , обчислюється:

$$w_d = f_{w,d} * \log \left( \frac{|D|}{f_{w,D}} \right), \text{ (формула 3.3)}$$

де  $f_{w,d}$  дорівнює кількості разів, коли  $w$  з'являється в  $d$ ,  $|D|$  - розмір корпусу, і  $f_{w,D}$  дорівнює кількості документів, в яких  $w$  відображається в  $D$ . Існує кілька

різних ситуацій, які можуть виникнути тут для кожного слова, залежно від значень  $f_{w,d}$ ,  $|D|$  та  $f_{w,D}$ .

Припустимо, що  $|D| \sim f_{w,D}$ , тобто розмір корпусу приблизно дорівнює частоті  $w$  над  $D$ . Якщо  $1 < \log \left( \frac{|D|}{f_{w,D}} \right) < c$  для деякої дуже малої константи  $c$ , то  $w_d$  буде меншим, ніж  $f_{w,d}$  але все ще позитивний. Це означає, що  $w$  є відносно поширеним у всьому корпусі, але все ще має певне значення протягом  $D$ . Наприклад, це може бути так, якби TF-IDF досліджував слово "Ньютон" над підручником з фізики. Більш важливий для нас такий результат можна було б очікувати від слова «Об'єднані» у корпусі документів ООН. Це також стосується надзвичайно поширених слів, таких як артиклі, займенники та прийменники, які самі по собі не мають відповідного значення у запиті (якщо користувач не має документів, що містять такі загальновживані слова). Таким чином, такі загальновживані слова отримують дуже низький бал TF-IDF, що робить їх по суті незначними при пошуку.

Нарешті, припустимо,  $f_{w,d}$  є великим і  $f_{w,D}$  є малим. Тоді  $\log \left( \frac{|D|}{f_{w,D}} \right)$  буде досить великим, і тому  $w_d$  також буде великим. Це той випадок, який є найбільш важливим в даному контексті, оскільки слова з високим  $w_d$  означають, що  $w$  - важливе слово в  $d$ , але не поширене в  $D$ . Цей термін  $w$  має велику дискримінаційну силу. Тому, коли запит містить це  $w$ , повернення документа  $d$ , де  $w_d$  велике, дуже ймовірно задовольнить користувача.

Код для TF-IDF елегантний своєю простотою. Враховуючи запит  $q$ , що складається з набору слів  $w_i$ , ми обчислюємо  $w_{i,d}$  для кожного  $w_i$  для кожного документа  $d \in D$ . Найпростішим чином це можна зробити, пробігаючи колекцію документів та зберігаючи поточну суму  $f_{w,d}$  та  $f_{w,D}$ . Після цього ми

можемо легко розрахувати  $w_{i,d}$  відповідно до математичних рамок, представлених раніше. Як тільки всі  $w_{i,d}$  знайдені, ми повертаємо набір  $D^*$ , що містить документи  $d$ , таким чином, щоб максимізувати таке рівняння:

$$\sum_i w_{i,d} \rightarrow \max. \text{ (формула 3.4.)}$$

Або користувач, або система можуть довільно визначити розмір  $D^*$  до ініціювання запиту. Також документи повертаються у порядку зменшення відповідно до рівняння зазначеного вище. Це традиційний метод реалізації TF-IDF. Хоч даний метод і виглядає досить просто, проте це не зменшує його ефективність. В подальших розділах буде проведений порівняльний аналіз цього методу з альтернативними.

TextRank - це графічна модель ранжування, яка може бути використана для різноманітних програм обробки природних мов, де знання, отримані з цілого документа, використовуються для прийняття місцевих рішень щодо ранжування / відбору. Основна ідея TextRank - витягнути графік з тексту документа, використовуючи текстові фрагменти як вершини. Що становить авертекс, залежить від завдання, до якого застосовується алгоритм [38].

Одиницями, що підлягають ранжуванню, є послідовності однієї або декількох лексичних одиниць, вилучених з тексту, і вони представляють вершини, додані до графіку тексту. Будь-яке відношення, яке можна визначити між двома лексичними одиницями, є потенційно корисним зв'язком (ребром), який можна додати між двома такими вершинами. Використовуються співвідношення спільності, що контролюється відстанню між входженнями слів: дві вершини пов'язані, якщо їх відповідні лексичні одиниці співіснують у вікні максимум слів, де може бути встановлено десь від 2 до 10 слів. Зв'язки спільної взаємодії виражають зв'язки між синтаксичними елементами, і,

подібно до семантичних зв'язків, знайдених корисними для завдання неоднозначності сенсу слова, вони представляють показники згуртованості для даного тексту [39].

Алгоритм вилучення ключових слів TextRank повністю не контролюється і діє наступним чином. По-перше, текст токенізується та коментується частиною мовних тегів - кроком попередньої обробки, необхідним для застосування синтаксичних фільтрів. Щоб уникнути надмірного збільшення розміру графіка, додаючи всі можливі комбінації послідовностей, що складаються з більш ніж однієї лексичної одиниці (ngrams), розглядаються лише окремі слова, як кандидати на додавання до графіка, при цьому ключові слова зі словосполученнями з часом реконструюються в дописі – це фаза обробки.

Далі на графік додаються всі лексичні одиниці, які проходять синтаксичний фільтр, і додається ребро між тими лексичними одиницями, які співіснують у вікні слів. Після побудови графіка (неорієнтований незважений графік) оцінка, пов'язана з кожною вершиною, встановлюється як початкове значення 1, а алгоритм ранжування, описаний у розділі 2, запускається на графіку протягом декількох ітерацій, поки він не сходиться - як правило, для 20- 30 ітерацій, на порозі 0,0001.

Після отримання остаточної оцінки для кожної вершини на графіку, вершини сортуються в зворотному порядку їх оцінки, а верхні вершини рейтингу зберігаються для подальшої обробки. Хоча може бути встановлено будь-яке фіксоване значення, яке зазвичай варіюється від 5 до 20 ключових слів (наприклад, (Turney, 1999) обмежує кількість ключових слів, витягнутих за допомогою його системи GenEx, до п'яти), можна використовувати й інший,



більш гнучкий підхід, який визначає кількість ключових слова на основі розміру тексту.

Під час подальшої обробки всі лексичні одиниці, вибрані як потенційні ключові слова за допомогою алгоритму TextRank, позначаються в тексті, а послідовності сусідніх ключових слів згортаються в ключове словосполучення [39].

RAKE (англ., Rapid Automatic Keyword Extraction) [40], тобто швидке автоматичне вилучення ключових слів, використовує стоп-лист для пошуку ключових слів-кандидатів. Будь-яка послідовність слів, що з'являються між двома словами зі стоп-списку та / або розділовими знаками, позначаються як ключові слова-кандидати. Потім частота та значення ступеня важливості кожного слова, так звана частота слова – це загальна кількість його повторень у списку ключових слів-кандидатів. Ступінь слова – це загальна кількість слів, з якими воно з'являється, у списку ключових слів кандидатів. Потім кожному слову присвоюється оцінка ступеня у відповідності до частоту. Сукупний бал кожного ключового слова-кандидата обчислюється шляхом підсумовування балів слів, які воно містить.

Також в цьому розділі представлено узагальнення спроб вирішити проблему значущості слова шляхом аналізу загальноприйнятих методів ранжування слів, виявлення морфологічної та синтаксичної подібності між словами, а в кінці – побудови багатозначного агента системи для вирішення проблеми побудови словника предметної області. Детальне дослідження описано в роботі [47].

Побудова автоматизованої системи, яка вирішує цю проблему, допоможе пришвидшити створення теоретичного багажу для нових областей

знань, які ще не уклали єдиної термінології, а також допоможе дослідникам краще зрозуміти нові галузі знань. Створення такого словника, навіть із погано визначеною онтологією, буде суттєвою допомогою у вирішенні цих проблем.

Теоретичні дослідження включають вивчення підходів, що використовуються при аналізі текстових даних, а саме - у пошуку ключових слів, пошуку семантичної подібності слів, а також у задачах створення короткого опису документа. На основі цього дослідження було обрано цільову групу методів, які будуть використовуватися в майбутньому. Порівняння існуючих методів, а саме: TF-IDF, RAKE, TextRank та моделей нейронної мережі Word2Vec. Методи були розділені на дві основні категорії відповідно до кількості документів, які вони обробляють: документ (RAKE, TextRank) і ціле тіло (TF-IDF, Word2Vec) [39][41]. Порівняння методів TF-IDF та Word2Vec було проведено на двох наборах корпусів текстових даних та базових словників предметних областей: фізики та біології. Параметри цих наборів наведені в таблицях 3.3, 3.4.

Таблиця 3.3. Параметри корпусу документів (домен фізики)

| Характеристики             | Значення       |
|----------------------------|----------------|
| Кількість документів       | 11 документів  |
| Кількість символів         | 27701 символів |
| Кількість слів             | 5732 слів      |
| Розмір в байтах            | 27100 байт     |
| Розмір словника в N-грамах | 135 N-грам     |

Таблиця 3.4. Параметри корпусу документів (домен біології)

| Характеристики             | Значення       |
|----------------------------|----------------|
| Кількість документів       | 6 документів   |
| Кількість символів         | 43105 символів |
| Кількість слів             | 12527 слів     |
| Розмір в байтах            | 62311 байт     |
| Розмір словника в N-грамах | 44 N-грам      |

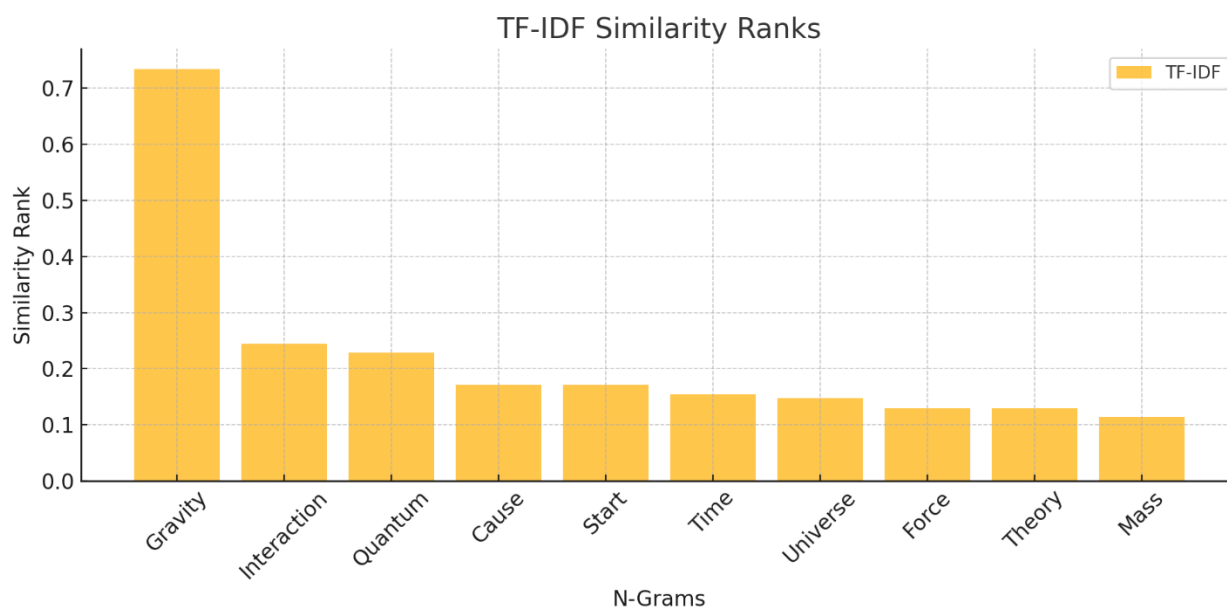


Рис 3.7. Результати методу TF-IDF (домен фізики) [власна розробка]

Практична частина експерименту включає проєктування та розробку системи на основі методу, обраного за результатами попереднього етапу. Цей етап включає як теоретичну розробку системи, так і аналіз існуючих рішень та архітектур для роботи з даними та практичну реалізацію. Були розглянуті

архітектури Карра та Lambda, а також системи розшифрування на основі мультиагентних систем [42][43][44]. Останнім етапом дослідження є аналіз результатів та спроба пояснити їх з точки зору використовуваних алгоритмів та архітектурних підходів до створення системи. Створено план подальших досліджень, що включає можливості вдосконалення системи.

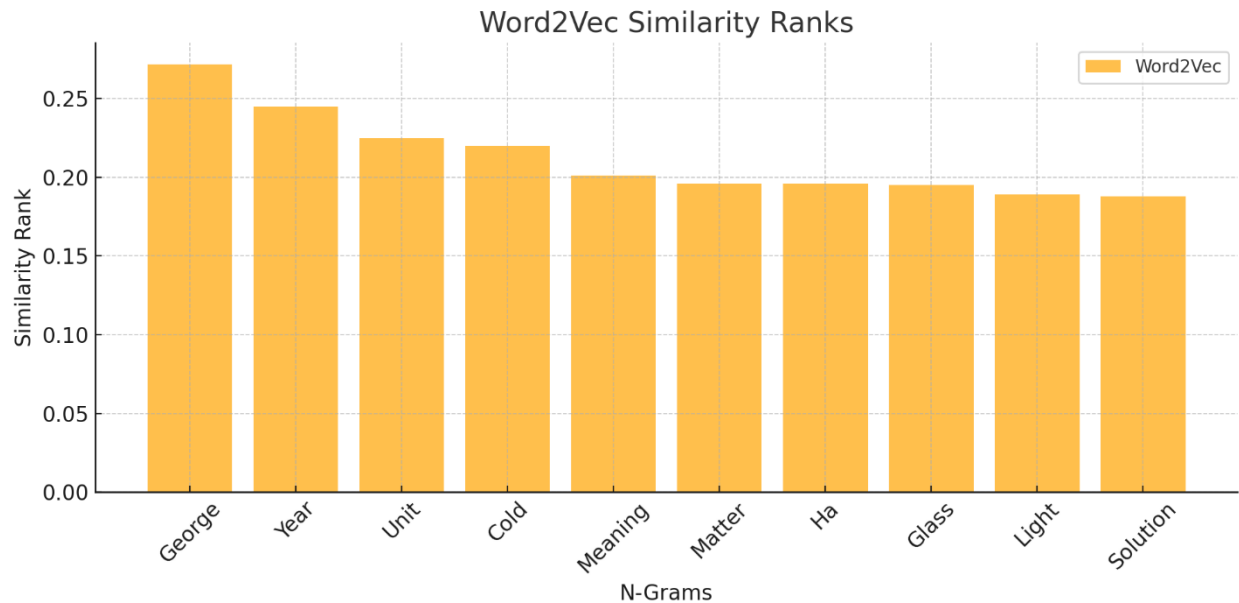


Рис 3.8. Результати методу Word2Vec (домен фізики) [власна розробка]

На етапі порівняння методів були отримані наступні результати: ключові слова, витягнуті за допомогою TF-IDF, та слова, подібні до тих, що містяться у словнику предметної області, похідному від Word2Vec, охоплюють досить широку онтологію. Однак N-грами, отримані методом TF-IDF, відносяться до галузі фізики з вищим значенням рангу (найважливішим є перше слово, через його високий ранг). Подібні результати були отримані в області біології (хоча це судження є суто евристичним, оскільки ранг подібності – це значення відносно слів в одному алгоритмі). Результати відображені графічно на наступних рисунках.

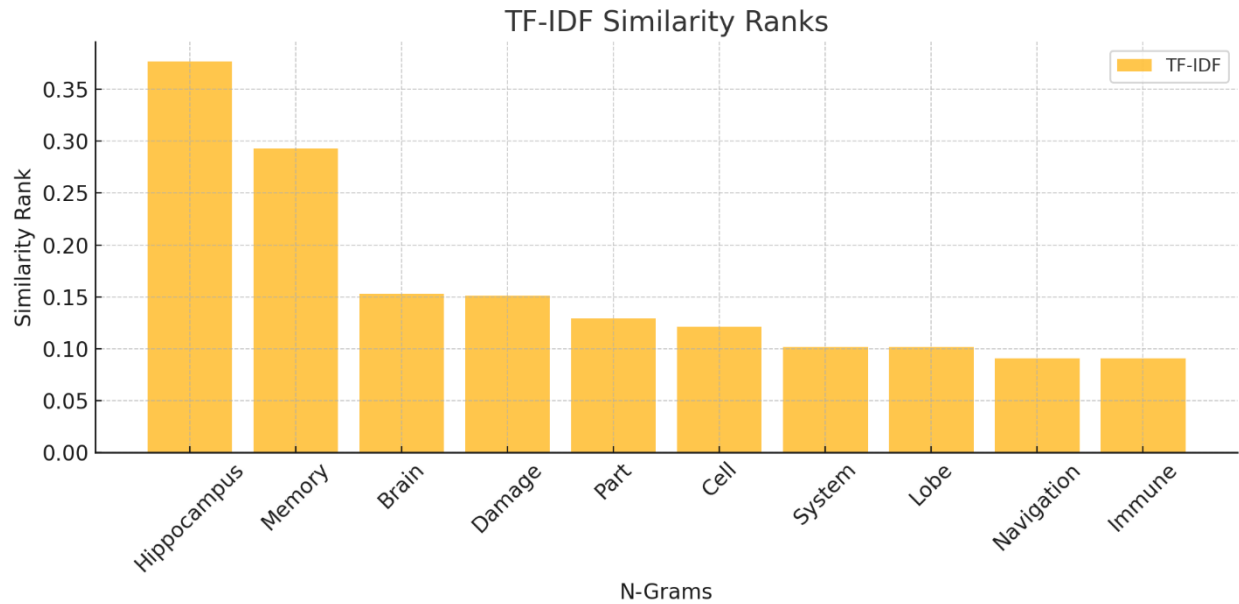


Рис 3.9. Результати методу TF-IDF (домен біології) [власна розробка]

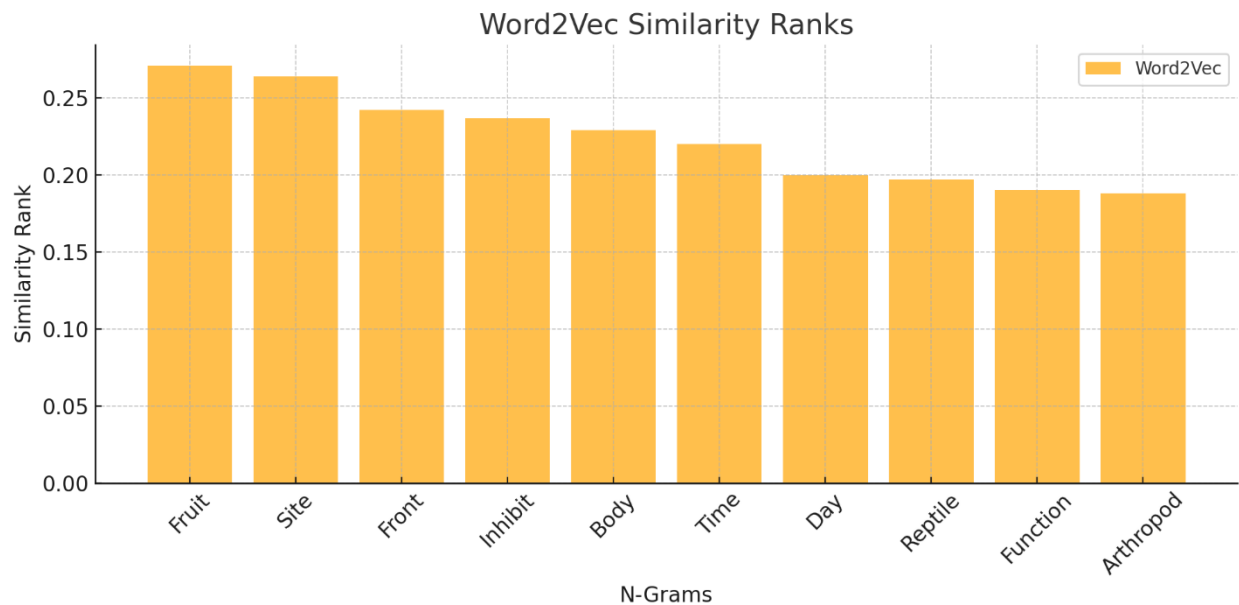


Рис 3.10. Результати методу Word2Vec (домен біології) [власна розробка]

Після порівняння та вибору методу на його основі було запропоновано архітектуру підсистеми MAC. Методи з використанням MAC досить ефективні [45]. На цьому етапі дослідження система досить проста і має наступний вигляд, показаний на рис. 3.11.

Основним компонентом у цій системі є агент (рис. 3.12), який виконує функцію обробки корпусу даних залежно від набору ключових слів, які йому надав агент планувальника.

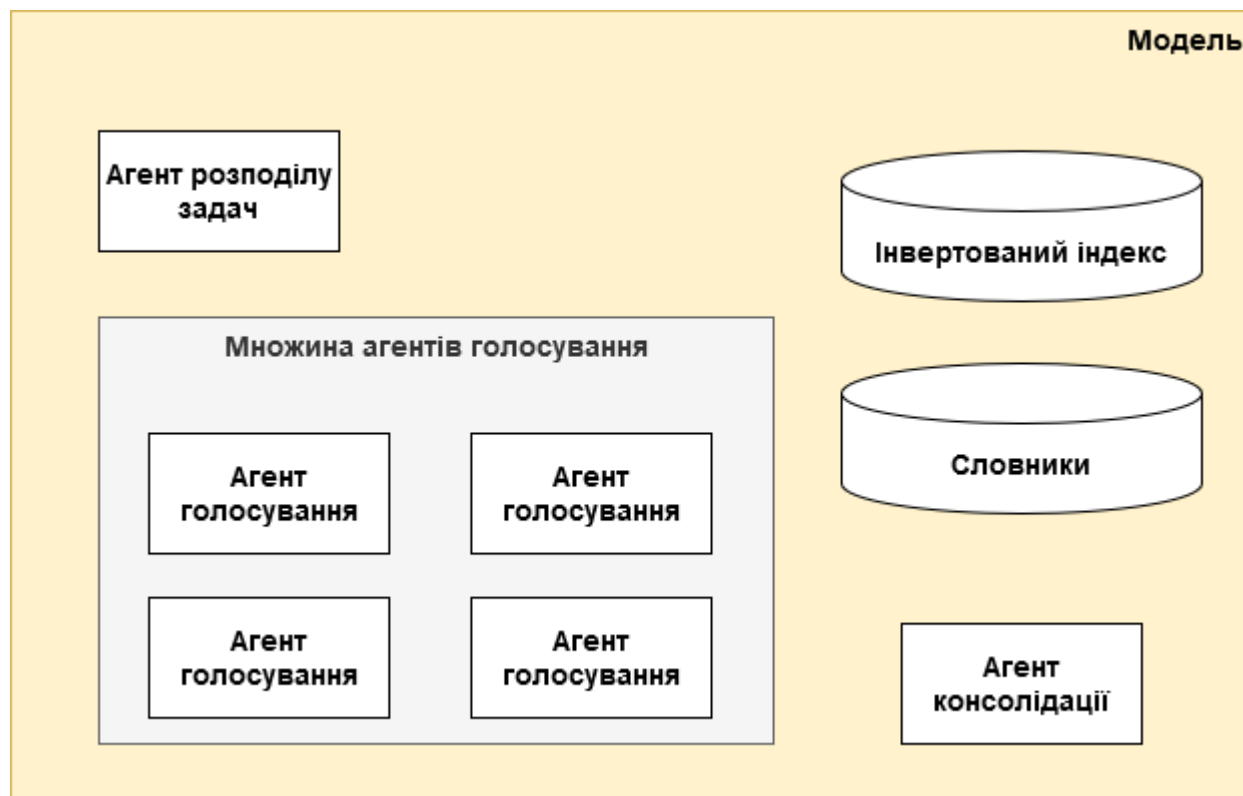


Рис 3.11. Модель агентів системи [власна розробка]

Кожен агент вводить певну кількість слів із базового словника, за допомогою якого він відбирає документи, які сформують його власне тіло для обробки за допомогою TF-IDF. Агент обробляє корпус і публікує власний набір слів-кандидатів, які слід додати до основного словника. Останній етап у системі - закріплення результатів роботи агентів (списків кандидатських слів) методом голосування Шульце та подальше додавання обраних голосуванням слів до основного словника [33].

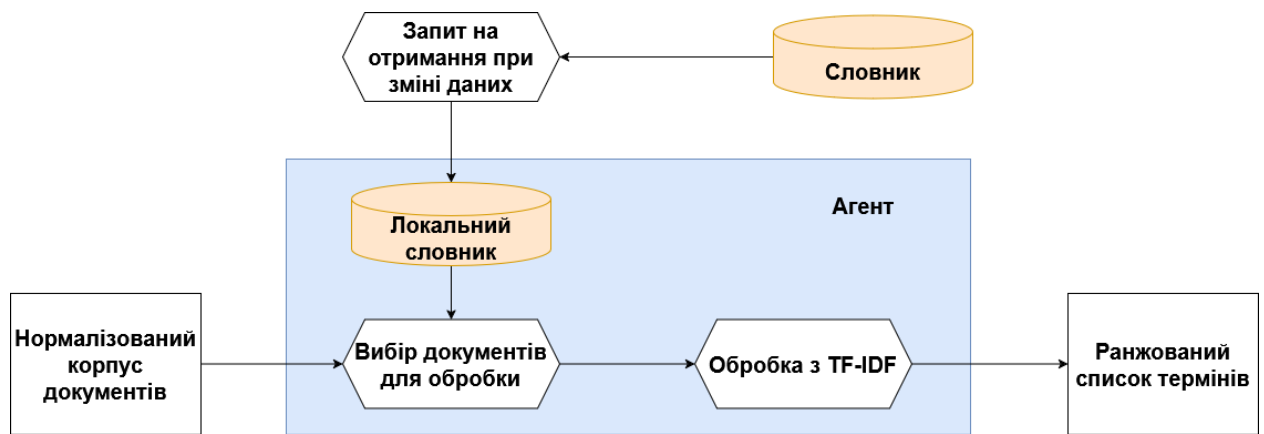


Рис 3.12. Внутрішня структура агента [власна розробка]

В експерименті були використані корпуси даних з таблиць 1, 2. Кожен агент отримує однакову кількість слів зі словника (якщо це можливо), і кількість агентів у системі дорівнює кількості слів у словнику предметної області, прийнятих як сума за модулем кількості слів агента для голосування. Кожен агент пропонує 10 слів для додавання до словника (голосування). Шульце виграє 4 слова в процесі голосування. Результати роботи системи з такою конфігурацією такі:

1. Для фізичного тіла обрано: «гравітація», «взаємодія», «квант», «старт».
2. Для біологічного тіла вибрано: «мозок», «хребетний», «структура», «розмір».

З цього прикладу ми можемо підрахувати, що точність системи становить близько 75%, оскільки 3 з 4 слів можна віднести до відповідних предметних областей.

На основі дослідження можна зробити висновок, що методи, засновані на частотному аналізі з використанням TF-IDF, демонструють високий результат точності пошуку слів, пов'язаних із загальною предметною областю, у структурах даних, що мають низьку семантичну в'язкість (словники).

В загальному випадку, для голосування можна обирати комбінацію з декількох методів. Наприклад, TF-IDF, який показав найкращий результат для даного набору, може мати вищий коефіцієнт врахування його результату [47]. Проте, в даному дослідженні достатньо зупинитись на методі TF-IDF з адаптацією під розподілене середовище, реалізованою з використанням методу Шульце.

Кожен агент розраховує локальну частоту термінів перед голосуванням методом Шульце, і саме це є ключовою особливістю модифікованого TF-IDF у розподіленому середовищі. У класичному TF-IDF значення TF (term frequency) визначається на рівні всього корпусу текстів, а IDF (inverse document frequency) враховує глобальну частоту термінів у всіх документах. Однак у даній системі, де агенти працюють незалежно та мають обмежений доступ до повного корпусу текстів, розрахунок здійснюється у два етапи. На першому етапі кожен агент обчислює локальні значення TF-IDF для своєї підмножини текстів, використовуючи власні обчислювальні ресурси. Це дозволяє виконувати обчислення паралельно, зменшуючи час обробки великих обсягів текстових потоків. Після цього кожен агент формує список ключових термінів, які він вважає релевантними для оновлення словника. На другому етапі агенти обмінюються своїми кандидатами через механізм голосування за допомогою методу Шульце. Цей метод дозволяє визначити найбільш значущі слова, беручи до уваги колективний вибір агентів, що працювали на різних частинах корпусу текстів. Важливо зазначити, що під час голосування враховується не лише частотність термінів, а й взаємозв'язки між словами у різних підмножинах текстів. Таким чином, слова, які отримали високі значення локального TF-IDF у різних агентів, матимуть вищий пріоритет при оновленні словника.



Щодо глобального IDF, він у даній системі оновлюється поступово: на кожній ітерації система враховує нові тексти, додає нові слова та знижує вагу термінів, які рідко з'являються або втрачають актуальність. Такий підхід дозволяє адаптивно оновлювати модель без необхідності зберігати весь корпус текстів та здійснювати централізований перерахунок IDF. Таким чином, модифікований TF-IDF у нашій системі дозволяє ефективно працювати в умовах розподілених обчислень, забезпечує баланс між локальними та глобальними частотами термінів і використовує механізм голосування для більш точної побудови словника.

### **3.5. Висновки.**

- В даному розділі були розглянуті методи класифікації потокових текстових даних великого розміру, були визначені основні критерії, за якими варто оцінювати роботу таких систем.
- Була запропонована модифікація фільтру Блума, яка дозволяє класифікувати одночасно декілька текстових класів. Дана модифікація була реалізована програмно, доведена її ефективність, низький час класифікації та висока точність.
- Запропоновано підхід до оптимізації параметрів фільтру Блума з використанням генетичних алгоритмів. Визначено фітнес-функцію та параметри для модифікації фільтру, а також – механізми мутації та вибору нащадків.
- Був проведений аналіз різних архітектур нейронних мереж на предмет часу класифікації та її точності. Дані архітектури були порівняні між собою та визначений оптимальний варіант, який дає точність класифікації 96% та вище.

- Проаналізовані сукупності методів для автоматичної побудови словника предметної області, порівняні результати їх роботи в залежності від різних можливих архітектур та комбінацій методів. Запропонована модифікація методу TF-IDF для можливості його роботи в розподіленому середовищі.
- Як результат попередніх пунктів – була запропонована модель мультиагентної системи, яка вирішує задачу класифікації потокових текстових даних. Дана модель була реалізована, а результати свідчать про її можливість класифікувати потоки текстових даних.

## **РОЗДІЛ 4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АПРОБАЦІЯ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗОВАНОГО СТВОРЕННЯ СЛОВНИКІВ.**

Опис результатів практичних досліджень, середовища для розробки та опису програмної системи наведений в розділах 4.1-4.3. Вказані параметри, використані у даному підході, його недоліки та нюанси, які необхідно враховувати при використанні даної системи в майбутньому, враховуючи особливості мови програмування, роботи методів, тощо. Наведені загальні висновки з точки зору практичної реалізації системи.

### **4.1. Формулювання експерименту побудови тестової мультиагентної системи для автоматизованої побудови словника предметної області.**

Метою є перевірити коректність і кількісні показники точності автоматизованої мультиагентної системи для побудови словника предметної області та класифікації текстів, враховуючи, що початковий словник задається людиною.

#### **Етапи експерименту**

#### **Попередня підготовка**

#### **Формування початкового словника:**

- Початковий словник створюється вручну на основі експертних знань для кожної предметної області. Людина визначає базовий набір ключових термінів, які є релевантними для конкретної категорії.
- Словники зберігаються у вигляді Python-словників, де ключами є назви категорій (наприклад, "sport", "tech"), а значеннями — списки відповідних термінів.

- Наприклад:

```
class_dicts = {  
    "sport": ["game", "win", "team", "player", "goal"],  
    "tech": ["technology", "software", "hardware", "digital", "mobile"],  
    ...  
}
```

Рис. 4.1. Приклад базових словників [власна розробка]

### **Створення тестового корпусу:**

- Текстові файли розташовуються у локальній файловій системі в структурах директорій, що відповідають класам (sport, tech, business, entertainment, politics).

### **Налаштування системи**

#### **Автоматизація процесу:**

- Після початкового завдання словника система автоматизовано виконує пошук, фільтрацію, класифікацію та оновлення словника на основі вхідних текстів. Це знижує потребу у подальшому втручанні людини.

#### **Мультикласовий фільтр Блума:**

- Використовується для зберігання та швидкого пошуку ключових термінів. Початкові терміни додаються до фільтра на основі заданого вручну словника.

#### **Нейронна мережа:**

- Використовується для класифікації текстів після їх фільтрації. Модель тренується один раз на основі початкового набору даних.

#### **Модель мультиагентної системи:**

- Автоматизує взаємодію між компонентами:

- Агент пошуку знаходить текстові файли у файловій системі.
- Агент фільтрації виділяє тексти, що містять терміни зі словника.
- Агент класифікації визначає категорію тексту.
- Агент менеджменту оновлює словник, додаючи нові терміни на основі аналізу.

## **Проведення експерименту**

### **Етап 1: Початковий словник**

- Сформувати базові словники для кожної категорії (sport, tech тощо).
- Додати ключові терміни до мультикласового фільтра Блума через метод add.

### **Етап 2: Автоматизована обробка даних**

- Запустити мультиагентну систему:
  - Агент пошуку та фільтрації перевіряє відповідність текстів ключовим термінам, обробляє текстові файли з директорій.
  - Агент класифікації визначає категорії текстів із використанням нейронної мережі.
  - Агенти побудови словника працюють над конструюванням наступної версії словника, приймаючи участь у голосуванні та спільному прийнятті рішень.
  - Агент менеджменту оновлює словник, додаючи нові терміни, якщо їхня частота у фільтрованих текстах перевищує заданий поріг.
  - Інші агенти виконують задачі, відповідно до описаних кроків в попередніх розділах.

### Етап 3: Оновлення словника

- Перевірити (експертною оцінкою), чи нові терміни, додані до словника, є релевантними, та чи не додаються нерелевантні терміни.

### Оцінка результатів

#### Початковий словник:

- Оцінити, чи правильно система працює з вручну заданим базовим набором термінів.

#### Автоматизація:

- Перевірити, наскільки ефективно система автоматично додає нові терміни до словника.

#### Точність:

- Оцінити точність класифікації текстів (precision, recall, F1-score).

### Інструменти та бібліотеки

- **PADE** для мультиагентної системи.
- **Фільтр Блума:** *bitarray*, *mmh3* для управління даними.
- **Нейронна мережа:** *TensorFlow*, *Keras* для побудови *LSTM*.
- **Токенізація:** *nlk* для попередньої обробки тексту.
- **Локальна файлова система:** *os* для роботи з файлами.
- **Логування:** *logging* для запису результатів.

### Очікувані результати

- **Точність:** Початковий словник забезпечує основу для високої точності класифікації.
- **Адаптивність:** Система автоматично вдосконалює словник, без подальшого втручання людини.
- **Продуктивність:** Система підтримує потокову обробку текстів.

## **4.2. Вирішення задачі фільтрації та класифікації для відповідних агентів**

### **4.2.1. Характеристики текстових даних**

Тексти, використані для тестування, належать до різних предметних областей, включаючи бізнес, розваги, політику, спорт і технології. Кожна категорія характеризується унікальними термінами та частотним розподілом слів, що дозволяє оцінити ефективність алгоритмів класифікації. Загальна кількість слів у текстах коливається від 73 тисяч до понад 111 тисяч у різних категоріях, що забезпечує надійну базу для аналізу [48].

Тексти в категорії "бізнес" містять 97 748 слів із середньою довжиною слова 6.18 символів, що свідчить про специфічність термінології. Найбільш уживані слова, такі як "said", "us" і "market", вказують на інформаційний характер матеріалів, пов'язаних із компаніями, ринками та економічними подіями.

Категорія "розваги" відрізняється яскравими термінами, такими як "film", "music" і "show", що відображає акцент на культурних подіях і творчості. Загальна кількість слів становить 73 129, середня довжина слова — 5.89 символів. Частота базових слів, таких як "said" і "best", підкреслює орієнтацію на оглядові матеріали.

Тексти в категорії "політика" мають найбільшу кількість слів — 104 390, із середньою довжиною слова 6.20 символів. Найпоширеніші терміни, такі як "government", "party" і "election", підкреслюють тематичну спрямованість на політичні процеси, вибори та діяльність уряду.

Категорія "спорт" містить 90 959 слів із середньою довжиною слова 5.88 символів. Популярні терміни, такі як "game", "win" і "world", акцентують увагу на спортивних змаганнях, перемогах і міжнародних подіях.

Категорія "технології" є найбільш насиченою за кількістю слів — 111 843, зі середньою довжиною слова 6.13 символів. Найчастіше зустрічаються терміни "technology", "people" і "mobile", що вказує на зміст, орієнтований на новітні розробки, використання гаджетів і вплив технологій на суспільство [48].

--- Overall Analysis for Directory: business ---

Total Words: 97748

Unique Words: 11208

Most Common Words: [('said', 1680), ('us', 813), ('year', 637), ('mr', 600), ('would', 464), ('also', 440), ('market', 425), ('company', 415), ('new', 415), ('growth', 384)]

Total Characters: 603859

Average Word Length: 6.18

-----  
--- Overall Analysis for Directory: entertainment ---

Total Words: 73129

Unique Words: 11234

Most Common Words: [('said', 825), ('film', 753), ('best', 588), ('music', 432), ('also', 398), ('us', 369), ('year', 368), ('one', 362), ('show', 328), ('new', 321)]

Total Characters: 430429

Average Word Length: 5.89

-----  
--- Overall Analysis for Directory: politics ---

Total Words: 104390

Unique Words: 10771

Most Common Words: [('said', 2240), ('mr', 1678), ('would', 1065), ('labour', 760), ('government', 730), ('people', 623), ('blair', 573), ('party', 569), ('election', 565), ('also', 452)]



```

Total Characters: 647258
Average Word Length: 6.20
-----
--- Overall Analysis for Directory: sport ---
Total Words: 90959
Unique Words: 10176
Most Common Words: [('said', 941), ('game', 476), ('england', 459), ('first',
437), ('win', 415), ('would', 411), ('world', 377), ('last', 376), ('one', 355), ('two',
351)]
Total Characters: 535153
Average Word Length: 5.88
-----
--- Overall Analysis for Directory: tech ---
Total Words: 111843
Unique Words: 11648
Most Common Words: [('said', 1567), ('people', 960), ('also', 537), ('new',
517), ('mr', 509), ('technology', 504), ('one', 497), ('would', 481), ('could', 473),
('mobile', 467)]
Total Characters: 686072
Average Word Length: 6.13
-----

```

Рис. 4.2. Опис тестових даних [власна розробка]

Ці характеристики демонструють різноманітність тематичних областей і забезпечують якісну базу для оцінки алгоритмів аналізу текстів.

#### **4.2.2. Визначення параметрів модифікованого фільтра Блума з використанням генетичних алгоритмів.**

Генетичний алгоритм (ГА) у цьому коді застосовується для оптимізації параметрів мультикласового фільтра Блума, який використовується для класифікації текстів. Основна мета — знайти оптимальну конфігурацію фільтра, яка забезпечує високу точність, мінімізує хибнопозитивні та хибнонегативні результати, і є ефективною з точки зору ресурсів. Генетичний

алгоритм моделює процес еволюції: кожна конфігурація (особина) оцінюється за функцією придатності, що враховує точність класифікації, кількість хибнопозитивних і хибнонегативних результатів.

Алгоритм починається з генерації початкової популяції випадкових конфігурацій, які характеризуються розміром таблиці, кількістю хеш-функцій, порогом класифікації та додатковими параметрами. На кожній ітерації конфігурації оцінюються за допомогою функції придатності. Найкращі конфігурації відбираються для створення нового покоління через схрещування, де нові "нащадки" отримують параметри від "батьків". Для підтримання генетичної різноманітності додається етап мутації, під час якого окремі параметри випадково змінюються.

У результаті ГА поступово оптимізує параметри фільтра Блума. Код дозволяє експериментально визначити найкращу конфігурацію, яка забезпечує баланс між точністю, продуктивністю та обсягом пам'яті. Цей підхід особливо корисний для автоматизації налаштування фільтра в умовах великої кількості параметрів і текстових даних.

Частини описаного вище генетичного алгоритму наведені в наступних фрагментах коду.

```
# Evaluate fitness
fitness_scores = []
for config in population:
    bloom_filter = MultiClassBloomFilter(size=config['size'],
num_hashes=config['num_hashes'],      hash_function=config['hash_function'],
**config['hash_args'])
```

```

        process_dictionaries(class_dicts, bloom_filter)
        total_accuracy = 0
        total_false_positives = 0
        total_false_negatives = 0
        for class_label, directory in zip(class_dicts.keys(), directories):
            accuracy, false_positives, false_negatives =
calculate_metrics(directory, bloom_filter, class_label,
threshold=config['threshold'])
            total_accuracy += accuracy
            total_false_positives += false_positives
            total_false_negatives += false_negatives
            fitness_score = total_accuracy / len(class_dicts) -
total_false_positives - total_false_negatives
            fitness_scores.append((fitness_score, config))

# Selection
fitness_scores.sort(reverse=True, key=lambda x: x[0])
sorted_population = [config for _, config in fitness_scores]
population = sorted_population[:population_size // 2] # Select top half

```

Рис. 4.3. Фрагмент коду генетичного алгоритму [власна розробка]

На завершальному етапі генетичний алгоритм визначив найкращу конфігурацію для фільтра Блума, яка забезпечує оптимальний баланс між точністю, продуктивністю та використанням ресурсів. Ця конфігурація включає розмір таблиці в 1751 елемент, використання 9 хеш-функцій із реалізацією `mmh3_hash`, поріг класифікації 0.002064, що мінімізує хибнопозитивні результати, і фактор хеш-функції 1.8767. Такий набір

параметрів забезпечує ефективну роботу фільтра, адаптованого до класифікації текстових даних.

```
vyaremenko@VadymLaptop:~/workspace/personal/phd/phd_dissertation/
programs/Ubuntu/BloomFilter$ python3 genetic_optimization.py

[nltk_data] Downloading package punkt to /home/vyaremenko/nltk_data...
[nltk_data] Package punkt is already up-to-date!

Generation 1
Generation 2
Generation 3
Generation 4
Generation 5
Generation 6
Generation 7
Generation 8
Generation 9
Generation 10

Best Configuration: {'size': 578, 'num_hashes': 6, 'hash_function': <function
custom_hash at 0x7fefb2cbc820>, 'threshold': 0.0020645016231201248,
'hash_args': {'factor': 1.83671536047641}}
```

Рис. 4.4. Ітерації генетичного алгоритму [власна розробка]

Визначена конфігурація вказує на значення, які були найкращими серед усіх тестованих. Зокрема, велика кількість хеш-функцій і оптимальний розмір таблиці сприяли зменшенню ймовірності колізій, що підвищило загальну точність. Порогове значення класифікації було адаптоване для забезпечення балансу між релевантністю й ризиком хибнопозитивних результатів. Цей результат підтверджує ефективність генетичного алгоритму для автоматизації

процесу налаштування складних структур даних, таких як мультикласовий фільтр Блума.

#### 4.2.3. Тестування фільтру Блума для агентів фільтрації даних.

Базовий мультикласовий фільтр Блума був налаштований із параметрами: розмір таблиці — 578, кількість хеш-функцій — 6, кастомна хеш-функція `custom_hash`, а поріг класифікації встановлено на рівні 0.0020645016231201248. Його основною метою було забезпечення швидкої перевірки відповідності текстів до визначених класів із використанням ефективної структури даних. Ця реалізація класифікатора демонструє хороші результати для більшості категорій, забезпечуючи значну швидкість обробки при відносно високій точності. Результати наведені в таблиці нижче.

```
vyaremenko@VadymLaptop:~/workspace/personal/phd/phd_dissertation/
programs/Ubuntu/BloomFilter$ python3 filtration_agent.py

[nltk_data] Downloading package punkt to /home/vyaremenko/nltk_data...
[nltk_data] Package punkt is already up-to-date!

Configuration:
  Table size: 578
  Number of hash functions: 6
  Hash function: custom_hash
  Threshold: 0.0020645016231201248
  Accuracy for sport: 0.78
  False positives for sport: 107
  False negatives for sport: 3
  Total number of files for sport: 511
  Accuracy for tech: 0.70
```

|  |
|--|
| False positives for tech: 119                |
| False negatives for tech: 0                  |
| Total number of files for tech: 401          |
| Accuracy for business: 0.77                  |
| False positives for business: 116            |
| False negatives for business: 2              |
| Total number of files for business: 510      |
| Accuracy for entertainment: 0.81             |
| False positives for entertainment: 74        |
| False negatives for entertainment: 0         |
| Total number of files for entertainment: 386 |
| Accuracy for politics: 0.75                  |
| False positives for politics: 106            |
| False negatives for politics: 0              |
| Total number of files for politics: 417      |

Рис. 4.5. Налаштування фільтру Блума [власна розробка]

За результатами класифікації, точність для різних класів коливалася від 70% до 81%. Категорії "sport", "business", "entertainment" показали точність на рівні 77-81%, що свідчить про ефективність збережених у фільтрі термінів. Категорія "tech" мала нижчу точність — 70%, що може бути пов'язано з більшим рівнем хибнопозитивів (119 випадків). Хибнопозитивні результати виявились суттєвим викликом для фільтра, особливо в категоріях з великою кількістю схожих термінів, що перекриваються з іншими класами.

Загалом, базовий класифікатор справляється з визначенням текстових класів у короткий час, проте потребує вдосконалення, щоб мінімізувати хибнопозитивні результати, які впливають на загальну продуктивність

системи. Це створює основу для подальшої оптимізації через генетичний алгоритм або розширення словників для покращення класифікаційних показників.

Проте, також важливим є тестування не лише на предмет класифікацій, а й на предмет фільтрації даних. Скажімо, якщо задати фільтру Блума характеристики для навчання на 3 класах, а два класи обрати як «шум», результати описані в наступному абзаці.

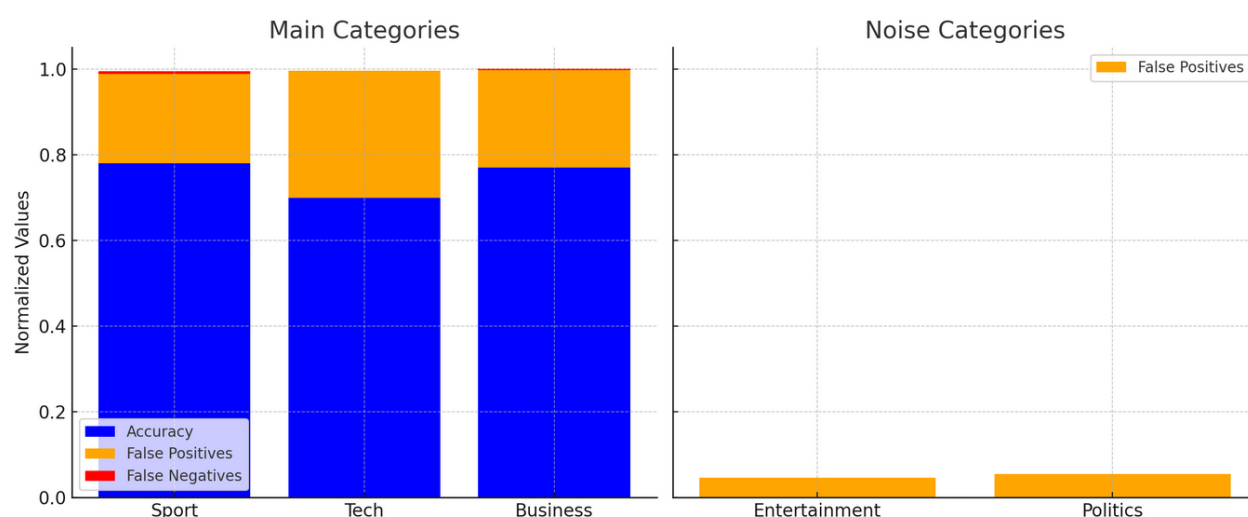


Рис. 4.5. Графічне представлення результатів налаштування фільтру Блума [власна розробка]

Результати тестування мультикласового фільтру Блума показують, що система ефективно обробляє тексти основних категорій та демонструє прийнятний рівень фільтрації шумових даних. Для шумових категорій "entertainment" і "politics" кількість хибнопозитивних результатів становила 18 із 386 файлів (4.7%) та 23 із 417 файлів (5.5%) відповідно.

Таке зниження хибнопозитивних результатів досягається завдяки оптимізації словників, точності параметрів фільтру, а також використанню

метрики множинної перевірки (ensemble voting). Попри це, є можливість подальшого вдосконалення системи через розширення специфічності словників та налаштування хеш-функцій для кращого розподілу значень у таблиці фільтру.

#### **4.2.4. Навчання та тестування нейронної мережі для агентів класифікації даних.**

Підготовка даних починається зі зчитування текстів із трьох категорій: "business", "tech", і "sport", загальною кількістю 2225 документів. Кожен текст позначається відповідним класом. Для навчання було використано попередньо навчені векторні представлення слів із GloVe (Global Vectors for Word Representation), які містять 400 000 векторів. Вектори були фільтровані, щоб використовувати лише ті слова, які зустрічаються в текстах, і створено матрицю вбудовувань розміром 15 749. Лише два слова з текстів не мали векторів у GloVe, що свідчить про високу якість відповідності.

Після розділення даних на навчальну та тестову вибірки (80% на навчання та 20% на тестування) тексти були токенізовані, перетворені на послідовності індексів та доповнені до максимальної довжини. Модель базується на двонаправленому шарі LSTM із використанням вбудованої матриці GloVe, що не оновлюється під час навчання. Протягом 20 епох навчання було досягнуто значного зменшення функції втрат із початкового значення 1.39 до 0.12. Остаточна точність на навчальній вибірці досягла 97.46%, а на тестовій — 95.68%, що свідчить про високу здатність моделі до узагальнення та класифікації текстів.



### Особливості моделі:

- **Архітектура:** Двонаправлений шар LSTM із 64 нейронами, шар Dropout для регуляризації, два повнозв'язані шари (16 нейронів для проміжного шару та 3 для вихідного шару відповідно до кількості класів).
- **Гіперпараметри:** Використовувався алгоритм оптимізації Adam із швидкістю навчання 0.001. Навчання тривало 20 епох із моніторингом точності та втрат на валідаційній вибірці.
- **Результати:** Остаточна точність на тестовій вибірці склала **95.68%**, втрата — 0.2089, що свідчить про високий рівень якості класифікації. Модель збережена у форматі HDF5 для подальшого використання.

```
vyaremenko@VadymLaptop:~/workspace/personal/phd/phd_dissertation/
programs/Ubuntu/Classifier$ python3 classifier.py
...
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument
`input_length` is deprecated. Just remove it.
  warnings.warn(
W0000 00:00:1732976629.442050 58762 gpu_device.cc:2344] Cannot
dlopen some GPU libraries. Please make sure the missing libraries mentioned
above are installed properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices...
Epoch 1/20
56/56 ————— 29s 483ms/step - accuracy: 0.4144 - loss:
1.3927 - val_accuracy: 0.6360 - val_loss: 1.0295
Epoch 2/20
56/56 ————— 26s 461ms/step - accuracy: 0.7058 - loss:
0.8219 - val_accuracy: 0.6157 - val_loss: 1.0270
Epoch 3/20
56/56 ————— 26s 458ms/step - accuracy: 0.7736 - loss:
0.7321 - val_accuracy: 0.8764 - val_loss: 0.4006
Epoch 4/20
```

56/56 ————— 25s 452ms/step - accuracy: 0.8936 - loss:  
 0.3855 - val\_accuracy: 0.8899 - val\_loss: 0.3530  
 Epoch 5/20  
 56/56 ————— 26s 456ms/step - accuracy: 0.8277 - loss:  
 0.5160 - val\_accuracy: 0.8270 - val\_loss: 0.5580  
 Epoch 6/20  
 56/56 ————— 25s 452ms/step - accuracy: 0.8542 - loss:  
 0.4628 - val\_accuracy: 0.7730 - val\_loss: 0.6241  
 Epoch 7/20  
 56/56 ————— 25s 447ms/step - accuracy: 0.7249 - loss:  
 0.7861 - val\_accuracy: 0.8831 - val\_loss: 0.3707  
 Epoch 8/20  
 56/56 ————— 25s 454ms/step - accuracy: 0.9122 - loss:  
 0.3009 - val\_accuracy: 0.8225 - val\_loss: 0.5128  
 Epoch 9/20  
 56/56 ————— 25s 447ms/step - accuracy: 0.8584 - loss:  
 0.4475 - val\_accuracy: 0.8989 - val\_loss: 0.2949  
 Epoch 10/20  
 56/56 ————— 26s 469ms/step - accuracy: 0.9382 - loss:  
 0.1983 - val\_accuracy: 0.9281 - val\_loss: 0.2673  
 Epoch 11/20  
 56/56 ————— 25s 449ms/step - accuracy: 0.9323 - loss:  
 0.2312 - val\_accuracy: 0.7551 - val\_loss: 0.7388  
 Epoch 12/20  
 56/56 ————— 32s 583ms/step - accuracy: 0.7930 - loss:  
 0.6234 - val\_accuracy: 0.8607 - val\_loss: 0.4628  
 Epoch 13/20  
 56/56 ————— 25s 452ms/step - accuracy: 0.8962 - loss:  
 0.3509 - val\_accuracy: 0.9079 - val\_loss: 0.3225  
 Epoch 14/20  
 56/56 ————— 25s 447ms/step - accuracy: 0.9197 - loss:  
 0.2845 - val\_accuracy: 0.8090 - val\_loss: 0.5692  
 Epoch 15/20  
 56/56 ————— 25s 454ms/step - accuracy: 0.8627 - loss:  
 0.4281 - val\_accuracy: 0.9011 - val\_loss: 0.3360  
 Epoch 16/20  
 56/56 ————— 25s 445ms/step - accuracy: 0.9347 - loss:  
 0.2123 - val\_accuracy: 0.9281 - val\_loss: 0.2632

```

Epoch 17/20
56/56 ————— 25s 454ms/step - accuracy: 0.9513 - loss:
0.1781 - val_accuracy: 0.9281 - val_loss: 0.2392
Epoch 18/20
56/56 ————— 26s 457ms/step - accuracy: 0.9531 - loss:
0.1638 - val_accuracy: 0.9371 - val_loss: 0.2480
Epoch 19/20
56/56 ————— 26s 465ms/step - accuracy: 0.9710 - loss:
0.1217 - val_accuracy: 0.9458 - val_loss: 0.2149
Epoch 20/20
56/56 ————— 26s 467ms/step - accuracy: 0.9746 - loss:
0.1162 - val_accuracy: 0.9568 - val_loss: 0.2089
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
Model saved to text_classification_model_bilstm.h5
14/14 ————— 2s 149ms/step - accuracy: 0.9289 - loss:
0.2534
Test Accuracy: 0.9567963132216

```

Рис. 4.6. Навчання LSTM [власна розробка]

Процес навчання демонструє ефективність використання попередньо навчених векторів GloVe та моделі LSTM у задачах класифікації текстів із різних категорій. Досягнута висока точність і низька втрата показують, що модель здатна розпізнавати тонкі відмінності між текстами різних класів.

Приклад тестування нейронної мережі наведений далі. Під час тестування класифікатор демонструє здатність правильно визначати клас тексту, використовуючи модель, навчану на LSTM, і токенізатор, створений у процесі підготовки даних. Програма завантажує попередньо збережену модель та токенізатор, токенізує тестовий текст і перетворює його у векторне подання для моделі. Класифікація здійснюється шляхом обчислення ймовірностей

належності тексту до кожного з класів, після чого виводиться клас із найвищою ймовірністю разом із розподілом ймовірностей. У представленому прикладі модель із впевненістю 99.27% віднесла текст до класу "політика", демонструючи високу точність класифікації. Такий підхід забезпечує автоматизацію та простоту перевірки моделі на нових текстах, підтверджуючи її ефективність і готовність до практичного використання.

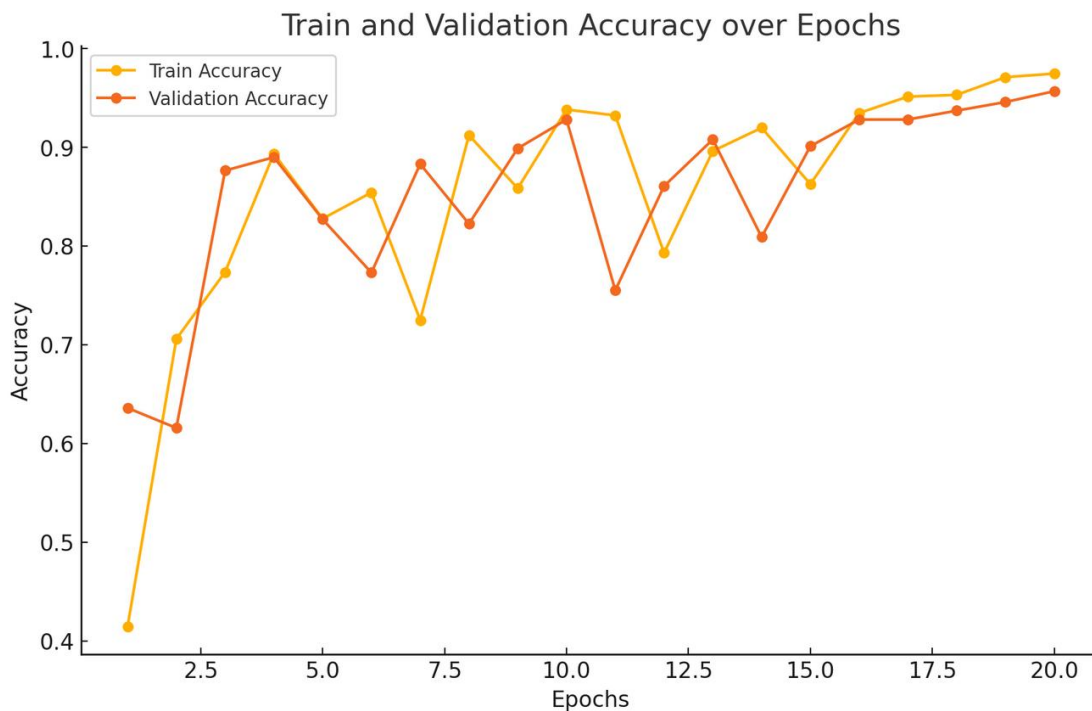


Рис. 4.7. Графічне зображення процесу навчання LSTM [власна розробка]

```
vyaremenko@VadymLaptop:~/workspace/personal/phd/phd_dissertation/
programs/Ubuntu/Classifier$ python3 test_classifier.py
Skipping registering GPU devices...
WARNING:absl:Compiled the loaded model, but the compiled metrics
have yet to be built. `model.compile_metrics` will be empty until you train or
evaluate the model.
Model loaded successfully.
Tokenizer loaded successfully.
```

|   |
|---|
| 1/1 ————— 0s 354ms/step<br>The file belongs to class: 3<br>Class probabilities: [1.5619693e-03 2.1407674e-03 9.5622422e-04] |
|---|

Рис. 4.7. Приклад роботи LSTM [власна розробка]

#### **4.3. Комплексне тестування побудованої мультиагентної системи: ітераційне вдосконалення словників.**

Методика побудови та уточнення словника заснована на використанні мультиагентної системи, де кожен агент виконує локальний аналіз текстів, генерує кандидатні словники, а результати їх роботи комбінуються за допомогою методу голосування Шульце. Цей підхід забезпечує ефективне врахування різних точок зору (агентів), що дозволяє створювати узгоджений і релевантний до задачі словник.

Програма реалізує ітеративний процес, який складається з наступних етапів:

- 1. Розподіл файлів між агентами:** Папка із текстами ділиться на групи файлів, які обробляються 5 агентами. Кожен агент отримує по 10 файлів для обробки за одну ітерацію.
- 2. Обробка текстів агентами:** Кожен агент аналізує свої тексти, використовуючи метод TF-IDF, та визначає топ-10 ключових слів, що утворюють локальний словник агента.
- 3. Голосування за допомогою методу Шульце:** Зібрані локальні словники агентів разом із поточним узгодженим словником беруть участь у голосуванні. Метод Шульце забезпечує вибір найбільш впливових слів, враховуючи попарні порівняння між кандидатами.
- 4. Уточнення словника:** На основі результатів голосування та ваг слів (важливості) відбувається оновлення узгодженого словника. Слова, які

втратили релевантність, з часом видаляються за допомогою механізму зменшення ваги.

Механізм ваги слів грає ключову роль в адаптації словника до нових даних. Вага слів у базового словника зменшується після кожної ітерації, якщо слово не з'являється в кандидатних словниках агентів. Це дозволяє поступово виключати менш релевантні терміни, забезпечуючи динамічну та актуальну модель словника.

**Загальний опис результатів** Результати тестування мультиагентної системи в області "спорт" показують динамічну еволюцію словника через кілька ітерацій. Початковий базовий словник включав загальні слова, такі як "football", "game", "time", "winter", "show". У процесі роботи агенти пропонували локальні кандидатні словники, що дозволило уточнювати початковий набір слів за допомогою методу голосування Шульце. Завдяки механізму ваги слів і зменшенню їх важливості слова, які втрачали релевантність, поступово виключалися.

**Ітерація 1** На першому етапі агенти сформували свої локальні словники на основі перших 50 текстів. Більшість запропонованих слів були пов'язані з футбольними матчами та змаганнями: "cup", "england", "game", "said". Результуючий словник включив слова "said", "england", "cup", "france", "play", які найчастіше з'являлися в кандидатних наборах. Ці слова отримали високі ваги, що відобразило їх релевантність для оброблених текстів.

Таблиця 4.1. Ітерація 1 конструювання словників предметної області

| Агент 1  | Агент 2  | Агент 3   | Агент 4  | Агент 5  | Результат  |
|--|--|---|--|--|--|
| beat,<br>celtic,<br>connor,<br>everton,<br>final,<br>game,<br>mcallister,<br>rangers,<br>second,<br>seed | cup,<br>england,<br>france,<br>injury,<br>play, said,<br>team,<br>time,<br>williams,<br>year | campese,<br>england,<br>operation,<br>play,<br>players,<br>rugby, said,<br>sculthorpe,<br>world, year | arsenal,<br>chelsea,<br>cup, just,<br>league,<br>liverpool,<br>said, think,<br>wales,<br>world | cyprus,<br>england,<br>faroe,<br>france,<br>ireland,<br>israel,<br>republic,<br>said,<br>switzerland,<br>win | football:1.62,<br>game:3.33,<br>time:3.33,<br>winter:1.62,<br>show:1.62,<br>said:3.33,<br>england:3.33,<br>cup:3.33,<br>france:2.43,<br>play:3.33,<br>roddick:2.70,<br>team:2.70,<br>year:2.70 |

**Ітерація 2** Під час другої ітерації агенти аналізували наступні 50 текстів. У результаті слова, що стосувалися міжнародних спортивних подій, таких як "roddick", "team", "year", увійшли до оновленого словника. Базовий набір було значно змінено, а релевантність початкових слів, як-от "winter", почала зменшуватися через недостатнє використання.

Таблиця 4.2. Ітерація 2 конструювання словників предметної області

| Агент 1   | Агент 2   | Агент 3  | Агент 4  | Агент 5  | Результат  |
|---|---|--|--|--|--|
| cup,<br>henman,<br>match,<br>roddick,<br>rusedski,<br>said, team,<br>time,<br>world, year | brazil,<br>final,<br>game,<br>jose, play,<br>roddick,<br>said,<br>second,<br>team,<br>tevez | ac, cup,<br>gerrard,<br>harriers,<br>liverpool,<br>new, parry,<br>said,<br>steven,<br>united | bates,<br>club,<br>dallaglio,<br>england,<br>game,<br>play, said,<br>set, white,<br>year | chelsea,<br>england,<br>game, half,<br>henson,<br>nations,<br>said, time,<br>united,<br>williams | football:1.46,<br>game:3.90,<br>time:3.90,<br>winter:1.46,<br>show:1.46,<br>said:3.90,<br>england:3.90,<br>cup:3.00,<br>france:2.19,<br>play:3.00,<br>roddick:2.43,<br>team:2.43,<br>year:3.33,<br>win:2.70,<br>world:2.70 |



**Ітерація 3** На третьому етапі були помітні зміни, пов'язані з домінуванням слів "win", "world", "england". Це свідчить про те, що тематика текстів поступово зміщувалася до міжнародних чемпіонатів і перемог. Механізм ваги забезпечив збереження найважливіших слів, таких як "game" і "time", які залишалися релевантними.

Таблиця 4.3. Ітерація 3 конструювання словників предметної області

| Агент 1   | Агент 2  | Агент 3  | Агент 4  | Агент 5   | Результат   |
|---|--|--|--|---|---|
| england,<br>game,<br>half, irish,<br>italy,<br>players,<br>said,<br>spain,<br>spanish,<br>try | athens,<br>britain,<br>goal,<br>ireland,<br>lead,<br>liverpool,<br>new,<br>points,<br>radcliffe,<br>said | best,<br>cardiff,<br>game,<br>liverpool,<br>olympic,<br>said,<br>seconds,<br>time, win,<br>world | ferrero,<br>home,<br>league,<br>match,<br>mirza,<br>round,<br>said, win,<br>world,<br>year | australian,<br>game,<br>hewitt,<br>international,<br>open, rugby,<br>said, wales,<br>win, world | football:1.31,<br>game:4.41,<br>time:4.41,<br>winter:1.31,<br>show:1.31,<br>said:4.41,<br>england:4.41,<br>cup:2.70,<br>france:1.97,<br>play:2.70,<br>roddick:2.19,<br>team:3.09,<br>year:3.90,<br>win:3.33,<br>world:3.33,<br>half:2.70,<br>rugby:2.70 |

**Ітерація 4** Агенти почали знаходити більше слів, пов'язаних із регбі та командними змаганнями: "rugby", "half", "team". Ці слова увійшли до словника завдяки їх високій популярності в текстах. Слово "rugby" отримало вагу, яка підняла його до основного списку.

Таблиця 4.4. Ітерація 4 конструювання словників предметної області

| Агент 1   | Агент 2  | Агент 3  | Агент 4  | Агент 5  | Результат   |
|---|--|--|--|--|---|
| edinburgh,<br>game,<br>glasgow,<br>half, home,<br>rugby, said,<br>smith,<br>squad,<br>start | england,<br>game, half,<br>new,<br>rugby, said,<br>team,<br>wales, win,<br>zealand | arsenal,<br>chelsea,<br>club,<br>drugs, old,<br>said,<br>season,<br>wenger,<br>world, year | chelsea,<br>club, cup,<br>fa, mido,<br>play,<br>rooney,<br>said, team,<br>time | arsenal,<br>cup,<br>ferguson,<br>game,<br>giggs,<br>players,<br>said, time,<br>united,<br>wenger | football:1.18,<br>game:4.87,<br>time:4.87,<br>winter:1.18,<br>show:1.18,<br>said:4.87,<br>england:4.87,<br>cup:2.43,<br>france:1.77,<br>play:2.43,<br>roddick:1.97,<br>team:3.68,<br>year:4.41,<br>win:3.90,<br>world:3.90,<br>half:2.43,<br>rugby:3.33 |

**Ітерація 5** На п'ятій ітерації ключовими стали слова, що пов'язані з тренерами та гравцями, наприклад "players", "coach". Вага слів "game" і "time" залишалася стабільно високою, що свідчить про їхню постійну важливість у текстах.

Таблиця 4.5. Ітерація 5 конструювання словників предметної області

| Агент 1  | Агент 2  | Агент 3   | Агент 4   | Агент 5   | Результат  |
|--|--|---|---|---|--|
| england,<br>game,<br>players,<br>robinson,<br>rugby,<br>said, time,<br>win, world,<br>year | conte,<br>good,<br>jones,<br>lions,<br>match,<br>rangers,<br>said,<br>smith,<br>world,<br>year | birmingham,<br>club, cup,<br>injury, new,<br>newcastle,<br>play, said,<br>souness,<br>world | chelsea,<br>collins,<br>great,<br>mcilroy,<br>race, said,<br>team, win,<br>world,<br>year | england,<br>hoddle,<br>juninho,<br>manager,<br>players,<br>said, time,<br>win,<br>wolves,<br>year | football:1.06,<br>game:5.28,<br>time:5.28,<br>winter:1.06,<br>show:1.06,<br>said:5.28,<br>england:5.28,<br>cup:2.18,<br>france:1.59,<br>play:2.18,<br>roddick:1.77,<br>team:3.31,<br>year:4.87,<br>win:4.41,<br>world:4.41,<br>half:2.19,<br>rugby:3.00,<br>ireland:2.70 |

**Ітерація 6** Слова, пов'язані з Ірландією та регіональними змаганнями, такі як "ireland", "rugby", стали помітними на цьому етапі. Слово "cup" почало втрачати свою вагу, оскільки його популярність зменшувалася.

Таблиця 4.6. Ітерація 6 конструювання словників предметної області

| Агент 1   | Агент 2   | Агент 3  | Агент 4  | Агент 5   | Результат  |
|---|---|--|--|---|--|
| final, gara, ireland, italy, kelly, penalty, scotland, second, set, try | british, champion, england, goal, match, said, time, win, world, year | bellamy, leinster, munster, new, players, said, squad, ulster, win, year | england, france, game, good, ireland, just, redknapp, said, season, sullivan | break, france, gara, got, hodgson, ireland, players, said, time, year | football:0.96, game:5.65, time:5.65, winter:0.96, show:0.96, said:5.65, england:5.65, cup:1.97, france:1.43, play:1.97, roddick:1.59, team:2.98, year:5.28, win:4.87, world:4.87, half:1.97, rugby:2.70, ireland:3.33, williams:2.70, players:2.70 |

**Ітерація 7** Додаткові терміни, наприклад "players", "williams", увійшли до словника завдяки значній частотності в текстах. Водночас слова з меншою важливістю, такі як "show", були майже повністю виключені з результуючого списку.

Таблиця 4.7. Ітерація 7 конструювання словників предметної області

| Агент 1   | Агент 2  | Агент 3  | Агент 4   | Агент 5  | Результат   |
|---|--|--|---|--|---|
| final, open, round, russia, said, second, seed, williams, win, year | cup, england, france, game, ireland, players, rugby, said, scotland, world | added, coach, conte, game, ireland, jones, said, team, time, win | driscoll, england, gara, greek, half, ireland, line, said, try, villa | italy, jones, liverpool, new, parry, players, try, wales, williams, year | game:5.99, time:5.99, said:5.99, england:5.99, cup:1.77, france:1.29, play:1.77, roddick:1.43, team:2.68, year:5.65, win:5.28, world:5.28, half:1.77, rugby:2.43, ireland:3.00, williams:2.43, players:2.43 |

**Ітерація 8** На цьому етапі словник стабілізувався навколо ключових понять, таких як "game", "time", "win", "world". Слово "england" зберегло високу вагу, що свідчить про його важливість у текстах.

Таблиця 4.8. Ітерація 8 конструювання словників предметної області

| Агент 1   | Агент 2  | Агент 3  | Агент 4   | Агент 5   | Результат   |
|---|--|--|---|---|---|
| athens,<br>club,<br>england,<br>france,<br>game, half,<br>like, play,<br>said, year | club, cup,<br>football,<br>game,<br>glazer,<br>new, said,<br>team,<br>united,<br>world | adriano,<br>england,<br>iaaf, ll,<br>llewellyn,<br>said, time,<br>ve, wales,<br>year | england,<br>francis,<br>game, got,<br>greene,<br>johansson,<br>media,<br>robinson,<br>said, win | holmes,<br>just, new,<br>old,<br>olympic,<br>said, time,<br>title,<br>world, year | game:6.29,<br>time:6.29,<br>said:6.29,<br>england:6.29,<br>cup:1.59,<br>france:1.16,<br>play:1.59,<br>roddick:1.29,<br>team:2.41,<br>year:5.99,<br>win:5.65,<br>world:5.65,<br>half:1.59,<br>rugby:2.18,<br>ireland:2.70,<br>williams:2.19,<br>players:3.09,<br>chelsea:2.70,<br>arsenal:2.70 |

**Ітерація 9** Словник відображав слова, які були пов'язані з англійськими футбольними клубами, наприклад "chelsea", "arsenal". Вага слова "rugby" зменшилася, відображаючи зниження його частотності в аналізованих текстах.

Таблиця 4.9. Ітерація 9 конструювання словників предметної області

| Агент 1  | Агент 2   | Агент 3  | Агент 4  | Агент 5   | Результат   |
|--|---|--|--|---|---|
| connors,<br>federer,<br>game,<br>james,<br>number,<br>players,<br>said,<br>smith,<br>tennis,<br>time | chelsea,<br>france,<br>game,<br>jones,<br>liverpool,<br>said, stade,<br>time,<br>wales, win | arsenal,<br>burnley,<br>game,<br>nations,<br>players,<br>round,<br>said, shot,<br>time, want | bellamy,<br>castaignede,<br>england,<br>indoor,<br>johnson,<br>play, said,<br>season,<br>world, year | arsenal,<br>chelsea,<br>england,<br>final,<br>henry,<br>open, said,<br>win,<br>world,<br>year | game:6.56,<br>time:6.56,<br>said:6.56,<br>england:6.56,<br>cup:1.43,<br>france:1.05,<br>play:1.43,<br>roddick:1.16,<br>team:2.17,<br>year:6.29,<br>win:5.99,<br>world:5.99,<br>half:1.43,<br>rugby:1.97,<br>ireland:2.43,<br>williams:1.97,<br>players:3.68,<br>chelsea:2.43,<br>arsenal:2.43 |

**Ітерація 10** Передостання ітерація підтвердила релевантність слів "said", "game", "time", які залишилися центральними. Менш популярні слова, такі як "roddick", були остаточно виключені, що дозволило створити компактний і точний словник.

Таблиця 4.10. Ітерація 10 конструювання словників предметної області

| Агент 1  | Агент 2   | Агент 3   | Агент 4   | Агент 5   | Результат  |
|--|---|---|---|---|--|
| boss,<br>lions,<br>open,<br>players,<br>said,<br>second,<br>time, win,<br>world,<br>year | charlton,<br>game,<br>indoor,<br>left,<br>record,<br>said,<br>season,<br>time, win,<br>year | game,<br>gerrard,<br>liverpool,<br>mourinho,<br>parry, said,<br>steven,<br>think, time,<br>year | england,<br>game, good,<br>injury, just,<br>premiership,<br>said, think,<br>white, year | cup,<br>england,<br>league,<br>moya,<br>robinson,<br>ruddock,<br>rugby,<br>said, time,<br>wales | game:6.80,<br>time:5.90,<br>said:6.80,<br>england:6.80,<br>cup:1.29,<br>france:0.94,<br>play:1.29,<br>roddick:1.05,<br>team:1.95,<br>year:6.56,<br>win:5.39,<br>world:5.39,<br>half:1.29,<br>rugby:1.77,<br>ireland:3.08,<br>williams:1.77,<br>players:3.31,<br>chelsea:2.19,<br>arsenal:2.19,<br>break:2.70 |



**Ітерація 11** Завершальна ітерація, сформований кінцевий словник для обмеженого набору тестових даних. Частина агентів не працювала через відсутність задач.

Таблиця 4.11. Ітерація 11 конструювання словників предметної області

| Агент 1  | Агент 2   | Агент 3    | Агент 4    | Агент 5    | Результат  |
|--|---|------------|------------|------------|--|
| break,<br>england,<br>game,<br>ireland,<br>nadal,<br>point,<br>roddick,<br>said, serve,<br>set | america,<br>athletics,<br>big, chunk,<br>compete,<br>training,<br>walker,<br>year | Без роботи | Без роботи | Без роботи | game:6.80,<br>time:5.90,<br>said:6.80,<br>england:6.80,<br>cup:1.29,<br>play:1.29,<br>roddick:1.05,<br>team:1.95,<br>year:6.56,<br>win:5.39,<br>world:5.39,<br>half:1.29,<br>rugby:1.77,<br>ireland:3.08,<br>williams:1.77,<br>players:3.31,<br>chelsea:2.19,<br>arsenal:2.19,<br>break:2.70 |

Результати тестування показали ефективність мультиагентного підходу для вирішення поставленої комплексної задачі та уточнення словника. Кінцевий словник містив найбільш релевантними слова для текстів предметної області "спорт", демонструючи адаптивність та здатність відкидати менш важливі терміни. Після 10 ітерацій результати співпали з прямим проходженням по всім файлам.

На рисунку 4.7. показано, як змінюється кількість слів у словнику протягом кількох ітерацій. Спочатку словник збільшується, коли вводяться нові слова, але пізніше він стабілізується, оскільки менш релевантні слова видаляються, а залишаються лише найважливіші терміни. В моделі можливі два способи вибору кількості елементів в словнику: з використанням порогового значення (автоматизований з використанням експерта для задання цього значення), та з використанням відносного значення до попереднього елементу словнику (автоматизований з використанням задання числа епсілон).

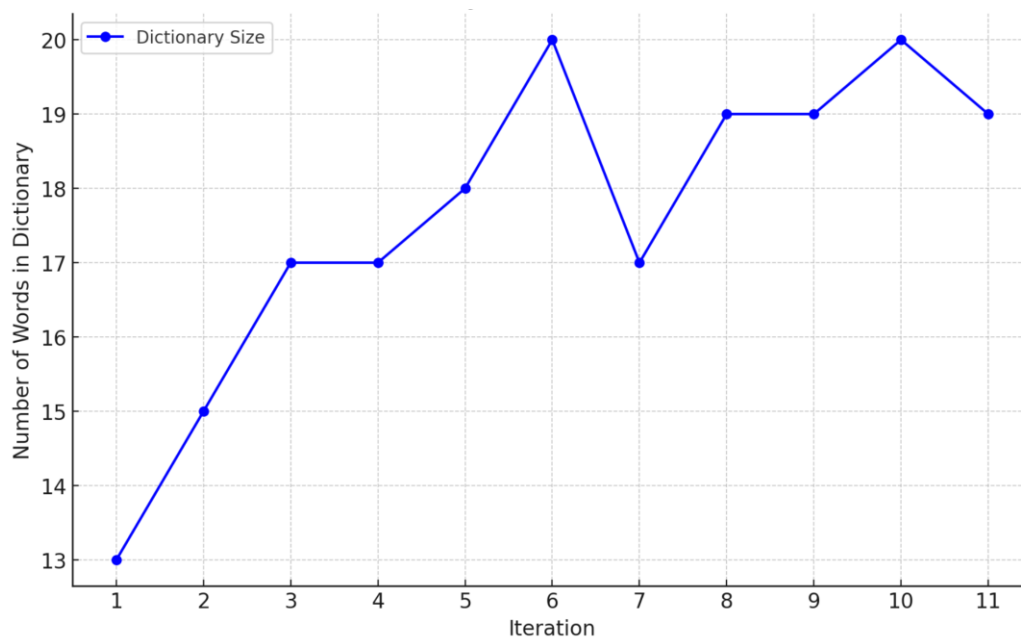


Рис. 4.7. Залежність розміру словника від ітерації [власна розробка]

На наступному графіку відображені результати самовдосконалення роботи системи фільтрації починаючи з ітерації 5 та закінчуючи кінцевим словником. Дані порівнюються з початковою ітерацією, де слова були задані експертом. Як можна помітити, вдосконалення словника впливає на точність фільтрації, що доводить потенціал використання механізму зворотного зв'язку в даній моделі мультиагентної системи.

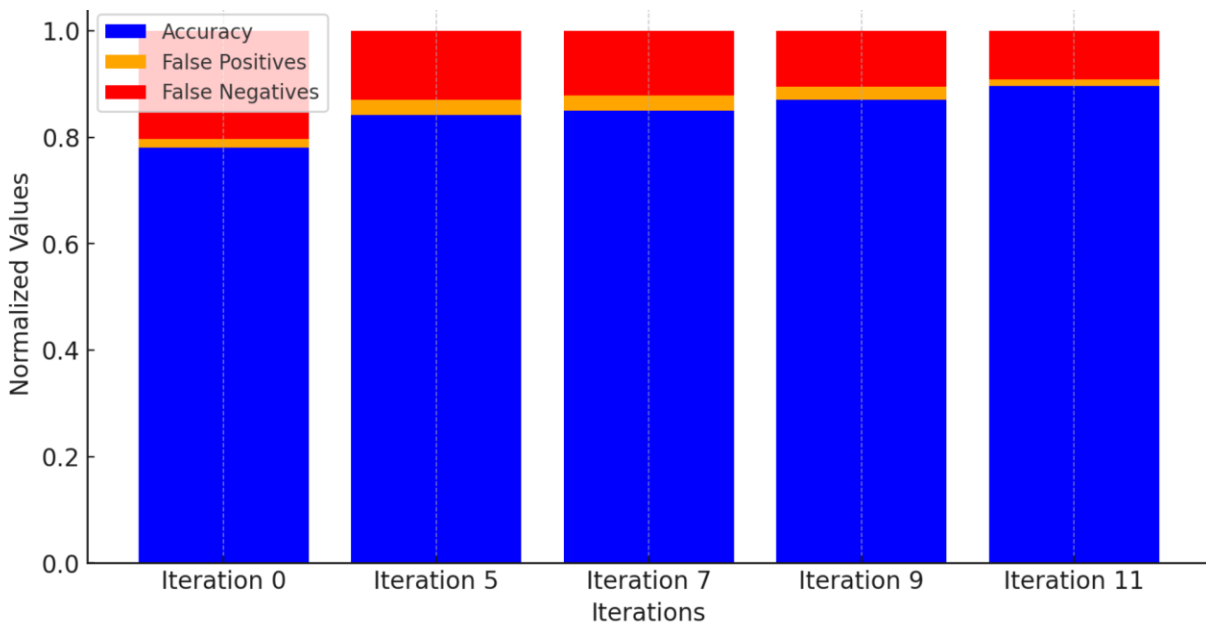


Рис. 4.8. Ітераційне покращення результатів фільтрації одного класу з використанням оновлених словників предметних областей [власна розробка]

#### 4.4. Висновки.

- Основою практичної роботи стала мультиагентна система, яка дозволяє автоматизовано створювати та оновлювати словник предметної області. Завдяки використанню TF-IDF для локального аналізу текстів агентами та голосування Шульце для об'єднання їх результатів, система здатна адаптуватися до змін у текстовому корпусі. Такий підхід забезпечив ефективну інтеграцію локальних словників-кандидатів від агентів в узгоджений словник, який оновлювався на кожній ітерації.
- Для оптимізації параметрів фільтра Блума, який використовується для попередньої фільтрації текстів, було реалізовано генетичний алгоритм. Генетичний алгоритм дозволив знайти найкращі конфігурації, такі як розмір таблиці, кількість хеш-функцій і поріг прийняття, шляхом ітеративного відбору, схрещування та мутації параметрів. Це значно підвищило ефективність фільтрації, зменшивши кількість хибнопозитивних результатів.
- Модифікований фільтр Блума був реалізований для швидкої перевірки належності текстів до певної предметної області. У процесі роботи він продемонстрував високу ефективність у відсіюванні нерелевантних текстів. Завдяки динамічному оновленню словника, система змогла підтримувати високу точність фільтрації навіть у випадку збільшення обсягу даних.
- Для отримання точного класу відфільтрованного тексту була реалізована модель нейронної мережі на основі двонаправленого LSTM. Модель досягла високої точності класифікації, підтвердивши можливість її використання для аналізу текстів.

- Поєднання мультиагентного підходу, фільтра Блума та нейронної мережі продемонструвало високу ефективність для автоматизованої роботи з текстовими даними. Кінцевий результат показав, що система здатна адаптуватися до нових текстових даних, підтримувати релевантність словника та забезпечувати високу точність класифікації. Це відкриває перспективи для масштабування підходу на більші текстові корпуси та інші предметні області.
- Всі вищезазначені вдосконалення реалізовані програмно з використанням бібліотеки PADE, було зроблені числові вимірювання результатів, чим було доведена ефективність запропонованого підходу.

## ВИСНОВКИ

У цьому дослідженні розроблено нову модель мультиагентної системи для автоматизованої побудови словника предметної області, яка може ефективно вирішити проблему обробки неперервних потоків слабоструктурованих текстових даних. Використовуючи властивості фільтрів Блума та нейронних мереж, реалізовано процес фільтрації та ефективної класифікації вхідних текстів. Цей підхід також включає процес спільного прийняття рішень, у якому кілька агентів пропонують зміни до словника, а потім голосують за найкраще рішення для кожної ітерації вдосконалення словника. Результати експериментів демонструють ефективність нашого підходу до створення точних і актуальних словників у динамічному середовищі, яке постійно змінюється. Це дослідження має важливі наслідки для додатків, які потребують обробки потокових текстових даних, наприклад, як у задачах обробки природної мови. За результатами роботи визначена наукова новизна, описана в подальших абзацах:

- **Запропоновано** модель мультиагентної системи, яка поєднує модифікований фільтр Блума, нейронну мережу для класифікації текстів, мультиагентний підхід для побудови та оновлення словників і механізм голосування методом Шульце з використанням методу TF-IDF, що дозволяє автоматизувати процес створення словників предметної області в умовах потокової обробки текстових даних.
- **Запропоновано** модифікацію класичного фільтра Блума, який відрізняється тим, що він забезпечує швидке виявлення релевантних текстів і виконання їх попередньої класифікації, що забезпечує значне

зменшення обсягу необроблених даних на наступних етапах системи та підвищує ефективність роботи в умовах обробки поточкових даних.

- **Запропоновано** модифікацію методу TF-IDF в розподіленому середовищі для вирішення задачі побудови словника предметної області, яка відрізняється застосуванням адаптованого методу Шульце для використання у мультиагентних системах при голосуванні між агентами щодо оновлення доменних словників, що забезпечує ухвалення рішень на основі колективного аналізу текстових даних.
- **Запропоновано** метод оптимізації параметрів хешування під час налаштування фільтру Блума з використанням генетичного алгоритму для застосування у задачі багатокласової фільтрації поточкових текстових даних для підвищення точності їх попередньої класифікації.

Розроблена система дозволяє автоматизувати побудову та оновлення словників предметної області шляхом обробки поточкових текстових даних і може бути адаптована до впровадження в існуючі системи. Вхідними параметрами розробленої моделі є потік текстів різних класів і початковий словник одного або декількох класів, вихідними ж даними є оновлений словник предметної області. Спочатку модифікований фільтр Блума відсіює нерелевантні тексти, після чого нейронна мережа класифікує дані з високою точністю. Далі агенти аналізують тексти, формують кандидатні терміни та голосують за оновлення словника методом Шульце. Оновлений словник інтегрується у фільтр Блума, що забезпечує динамічну адаптацію системи до нових даних. Користувач може переглядати статистику, коригувати словник і використовувати його для інформаційного пошуку, аналізу текстів, NLP-завдань або інших задач інтелектуальної обробки текстів.

Механізм системи зворотного зв'язку, який передає оновлений словник назад у фільтр Блума, що дозволяє динамічно адаптувати систему до нових даних і підвищити точність фільтрації текстів, що забезпечує постійну актуальність системи в умовах змінного середовища даних та підвищує якість отриманих словників. Модель мультиагентної системи має відкриту будову та низькі вимоги до обчислювальних ресурсів за рахунок зменшення обсягу переданих даних і розподілений характер обробки, що забезпечує гнучкість та можливість ефективного масштабування, а в результаті – економію коштів. Запропоноване рішення може застосовуватися в задачах аналізу текстових потоків, моніторингу та інформаційного пошуку у різних сферах, в тому числі – кібербезпеки, фінансів, соціальних мереж тощо.



## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wooldridge M. *An Introduction to MultiAgent Systems 2nd Edition* / Michael Wooldridge., 2009. – 488 с. – (2).
2. Bellifemine F. *Developing Multi-Agent Systems with JADE* / F. Bellifemine, D. Greenwood, G. Caire., 2007. – 304 с. – (1).
3. Hassani H. *Text Mining in Big Data Analytics* / H. Hassani, C. Beneki, S. Unger, M. Mazinani, M. Yeganegi. – 2020. – *Big Data and Cognitive Computing*. – 4(1). – DOI: 10.3390/bdcc4010001.
4. Agrawal R. *A Detailed Study on Text Mining Techniques* / R. Agrawal, M. Dhingra. – 2013. – *International Journal of Soft Computing and Engineering (IJSCE)*. – ISSN: 2231-2307. – Volume-2, Issue-6, January 2013.
5. Talib R. *Text Mining: Techniques, Applications and Issues* / R. Talib, M. K. Hanif, S. Ayesha, F. Fatima. – 2016. – *International Journal of Advanced Computer Science and Applications (IJACSA)*. – Vol. 7, No. 11.
6. Ennals R. *Highlighting Disputed Claims on the Web* / R. Ennals, B. Trushkowsky, J. M. Agosta. – 2010. – *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. – Association for Computing Machinery, New York, NY, USA. – C. 341–350. – DOI: 10.1145/1772690.1772726.
7. Tsytarau M. *Survey on Mining Subjective Data on the Web* / M. Tsytarau, T. Palpanas. – 2011. – *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. – C. 478–514.
8. Mowafy M. *An Efficient Classification Model for Unstructured Text Document* / M. Mowafy, A. Rezk, H. M. El-Bakry. – 2018. – *American Journal of Computer Science and Information Technology*. – ISSN: 2349-3917. – Vol. 6, No. 1:16.
9. Chen M. *Big Data: A Survey* / M. Chen, S. Mao, Y. Liu. – 2014. – *Mobile Networks and Applications*. – Vol. 19, No. 2. – C. 171–209. – DOI: 10.1007/s11036-013-0489-0.
10. Ganz J. *Extracting Value from Chaos* / J. Ganz, D. Reinsel. – 2011. – *IDC Analyze the Future*. – Видавник: IDC.
11. Jordan M. I. *Machine Learning: Trends, Perspectives, and Prospects* / M. I. Jordan, T. M. Mitchell. – 2015. – *Science*. – Vol. 349, No. 6245. – C. 255–260. – DOI: 10.1126/science.aaa8415.
12. Leskovec J. *Mining of Massive Datasets 2nd Edition* / J. Leskovec, A. Rajaraman, J. D. Ullman. – Cambridge University Press, 2014. – ISBN-10: 9781107077232, ISBN-13: 978-1107077232. – 2nd Edition.

13. Yaremenko V. *An Approach for Data Clustering CURE Algorithm Implementation Using the MapReduce Technology* / V. Yaremenko. – 2017. – ESC “IASA” NTUU “KPI” – 19th International Conference SAIT 2017, Kyiv, Ukraine, May 22-25. – C. 204.
14. Polozniuk K. *Neural Networks and Monte-Carlo Method Usage in Multi-Agent Systems for Sudoku Problem Solving* / K. Polozniuk, V. Yaremenko. – 2020. – *Technology Audit and Production Reserves*. – №6/2 (56). – C. 38–41.
15. Camara M. *A Multi-Agent System with Reinforcement Learning Agents for Biomedical Text Mining* / M. Camara, O. Bonham-Carter, J. Jumadinova. – 2015. – *BCB '15: Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*. – C. 634–643. – DOI: 10.1145/2808719.2812596.
16. Poniszewska-Maranda A. *Retrieval and Processing of Information with the Use of Multi-Agent System* / A. Poniszewska-Maranda, L. Gebel. – 2016. – *Journal of Applied Computer Science*. – Vol. 24, No. 2. – C. 17–37.
17. Ottens K. *A Multi-Agent System for Building Dynamic Ontologies* / K. Ottens, M.-P. Gleizes, P. Glize. – 2007. – *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07)*. – DOI: 10.1145/1329125.1329399.
18. Twardowski B. *Multi-Agent Architecture for Real-Time Big Data Processing* / B. Twardowski, D. Ryzko. – 2014. – *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*. – DOI: 10.1109/wi-iat.2014.185.
19. Chavan G. S. *A Survey of Various Machine Learning Techniques for Text Classification* / G. S. Chavan, S. Manjare, P. Hegde, A. Sankhe. – 2014. – *International Journal of Engineering Trends and Technology*. – Vol. 15, No. 6. – C. 288–292. – DOI: 10.14445/22315381/IJETT-V15P255.
20. Veríssimo P. *Distributed Systems for System Architects* / P. Veríssimo, L. Rodrigues. – 2001. – *Advances in Distributed Computing and Middleware*. – DOI: 10.1007/978-1-4615-1663-7.
21. Korte T. *Supercomputing vs. Distributed Computing: A Government Primer* / T. Korte. – January 5, 2014. – Режим доступа до ресурсу: <https://datainnovation.org/2014/01/supercomputing-vs-distributed-computing-a-government-primer>
22. Chow K.-P. *On Load Balancing for Distributed Multiagent Computing* / K.-P. Chow, Y.-K. Kwok. – 2002. – *IEEE Transactions on Parallel and Distributed Systems*. – Vol. 13, No. 8. – C. 787–801. – DOI: 10.1109/tpds.2002.1028436.

23. Gómez-Sanz J. *Methodologies for Developing Multi-Agent Systems* / J. Gómez-Sanz, J. Pavón. – 2004. – *Journal of Universal Computer Science*. – Vol. 10, No. 4. – С. 359–374. – Submitted: 17/10/03, Accepted: 2/2/04, Appeared: 28/4/04.
24. Poslad S. *Protecting What Your Agent Is Doing* / S. Poslad, P. Charlton, M. Calisti. – 2001.
25. Searle J. R. *Speech Acts: An Essay in the Philosophy of Language* / J. R. Searle. – 1969. – Cambridge: Cambridge University Press.
26. Poslad S. *Specifying Protocols for Multi-Agent Systems Interaction* / S. Poslad. – 2007. – *ACM Transactions on Autonomous and Adaptive Systems*. – Vol. 2, No. 4. – С. 15–es. – DOI: 10.1145/1293731.1293735.
27. FIPA Standard. – 2002. – Режим доступу до ресурсу: <http://www.fipa.org/repository/standardspecs.html>.
28. Bordini R. H. *A Survey of Programming Languages and Platforms for Multi-Agent Systems* / R. H. Bordini, L. Braubach, M. Dastani. – 2006. – *Informatica*. – Vol. 30. – С. 33–44.
29. DALI Framework. – Режим доступу до ресурсу: <https://github.com/AAAI-DISIM-UnivAQ/DALI>.
30. PADE Framework. – Режим доступу до ресурсу: <https://pade.readthedocs.io/en/latest/>.
31. osBrain Module. – Режим доступу до ресурсу: <https://osbrain.readthedocs.io/en/stable/about.html>.
32. Pitt J. *Voting in Multi-Agent Systems* / J. Pitt, L. Kamara, M. Sergot, A. Artikis. – 2006. – *Computer Journal*. – Vol. 49. – С. 156–170.
33. Schulze M. *The Schulze Method of Voting* / M. Schulze. – 2018. – DOI: 10.48550/arXiv.1804.02973.
34. Яременко В. С. Підхід до використання фільтра Блума для багатокласової класифікації текстових даних в режимі реального часу / В. С. Яременко, Д. Ю. Будьонний. – 2019. – *Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво"*. – №6. – С. 153–159.
35. Yaremenko V. S. *Application of Neural Network Algorithms and Naive Bayes for Text Classification* / V. S. Yaremenko, W. S. Rogoza, V. I. Spitkovskyi. – 2021. – *Journal of Theoretical and Applied Information Technology*. – Vol. 99, No. 1. – ISSN: 1992-8645, E-ISSN: 1817-3195. – С. 125.
36. Rong X. *Word2vec Parameter Learning Explained* / X. Rong. – 2016. – arXiv:1411.2738v4 [cs.CL], 5 Jun 2016. – Режим доступу до ресурсу: <https://arxiv.org/abs/1411.2738v4>.

37. Ramos J. *Using TF-IDF to Determine Word Relevance in Document Queries* / J. Ramos. – 2003.
38. Petasis G. *Identifying Argument Components through TextRank* / G. Petasis, V. Karkaletsis. – 2016. – *Proceedings of the 3rd Workshop on Argument Mining*. – Berlin, Germany, August 7–12. – C. 94–102. – ©2016 Association for Computational Linguistics.
39. Mihalcea R. *TextRank: Bringing Order into Texts* / R. Mihalcea, P. Tarau. – *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. – 2004. – Режим доступа до ресурсу: <https://aclanthology.org/W04-3252.pdf>.
40. Pay T. *An Ensemble of Automatic Keyword Extractors: TextRank, RAKE and TAKE* / T. Pay, S. Lucci, J. L. Cox. – 2019. – *Computación y Sistemas*. – Vol. 23, No. 3. – C. 703–710. – ISSN: 2007-9737. – DOI: 10.13053/CyS-23-3-3234.
41. Rose S. R. *Automatic Keyword Extraction from Individual Documents* / S. R. Rose, D. Engel, N. Cramer, W. Cowley. – 2010. – *Text Mining*. – DOI: <http://doi.org/10.1002/9780470689646.ch1>.
42. Twardowski B. *Multi-Agent Architecture for Real-Time Big Data Processing* / B. Twardowski, D. Ryzko. – 2014. – *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*. – Vol. 3. – C. 333–337. – DOI: <http://doi.org/10.1109/wi-iat.2014.1858>.
43. Kiran M. *Lambda Architecture for Cost-Effective Batch and Speed Big Data Processing* / M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja. – 2015. – *2015 IEEE International Conference on Big Data (Big Data)*. – C. 2785–2792. – DOI: <http://doi.org/10.1109/bigdata.2015.73640829>.
44. Singh K. *Big Data Ecosystem: Review on Architectural Evolution* / K. Singh, R. Behera, J. Mantri. – 2019. – *Advances in Intelligent Systems and Computing*. – C. 335–345. – DOI: [http://doi.org/10.1007/978-981-13-1498-8\\_30](http://doi.org/10.1007/978-981-13-1498-8_30).
45. Aref M. M. *A Multi-Agent System for Natural Language Understanding* / M. M. Aref. – 2003. – *IEMC'03 Proceedings. Managing Technologically Driven Organizations: The Human Side of Innovation and Change (IEEE Cat. No.03CH37502)*. – C. 36–40. – DOI: <http://doi.org/10.1109/kimas.2003.1245018>.
46. Amdahl G. *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities* / G. Amdahl. – 2007. – *Reprinted from the AFIPS Conference Proceedings, Vol. 30 (Atlantic City, N. J., Apr. 18–20)*. – *IEEE Solid-State Circuits Newsletter*. – Vol. 12. – C. 19–20. – DOI: <http://doi.org/10.1109/n-ssc.2007.4785615>.

47. Yaremenko V. *Development of a Multi-Agent System for Solving Domain Dictionary Construction Problem* / V. Yaremenko, O. Syrotiuk. – 2020. – *Technology Audit and Production Reserves*. – №4/2 (54). – С. 27–30.
48. BBC Full Text Document Classification Dataset. – Режим доступу до ресурсу: <https://www.kaggle.com/datasets/shivamkushwaha/bbc-full-text-document-classification>.
49. Hryshchenko O. A Comparative Analysis of Text Data Classification Accuracy and Speed Using Neural Networks, Bloom Filter and Naive Bayes / O. Hryshchenko, V. Yaremenko. – 2021. – *Technology Audit and Production Reserves*. – №5/2 (61). – P. 6–8. <https://doi.org/10.15587/2706-5448.2020.208400>
50. Kungurtsev O. B., Mileiko I. I., Novikova N. O. Technology for Automated Construction of Domain Dictionaries with Special Processing of Short Documents / O. B. Kungurtsev, I. I. Mileiko, N. O. Novikova. – 2024. – *Radio Electronics, Computer Science, Control*. – №4. – P. 148. <https://doi.org/10.15588/1607-3274-2023-4-14>
51. Ren W., Zhang H., Chen M. A Method of Domain Dictionary Construction for Electric Vehicles Disassembly / W. Ren, H. Zhang, M. Chen. – 2022. – *Entropy*. – №24(3). – P. 363. <https://doi.org/10.3390/e24030363>

## **ДОДАТОК А. СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ.**

### **Зарубіжні видання, що індексуються у наукометричній базі Scopus (Q4)**

1. Yaremenko V., Rogoza W., Spitkovskiy V. Application of neural network algorithms and naïve bayes for text classification. Journal of Theoretical and Applied Information Technology. 2021. Vol.99. No 1. P. 125-134. ISSN 1606-3716.

### **Українські фахові видання категорії «Б» (спеціальність 122)**

2. Яременко В. С. Огляд наявних мультиагентних систем для задач інтелектуального аналізу даних. Вчені записки Таврійського національного університету імені В. І. Вернадського. 2018. Том 29 (68), №3. С. 47–55.

3. Яременко В. С., Будьонний Д. Ю. Підхід до використання фільтра Блума для багатокласової класифікації текстових даних в режимі реального часу. Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво". 2019. №6. С. 153–159.

4. Яременко В. С., Худяков А. С. Модель мультиагентної системи для семантичного аналізу текстів. Міжвузівський збірник наукових праць "Наукові нотатки". 2019. №68. С. 152–156.

5. Yaremenko V., Syrotiuk O. Development of a multi-agent system for solving domain dictionary construction problem. Technology audit and production reserves. 2020. №4/2 (54). P. 27–30.

6. Hryshchenko O. A comparative analysis of text data classification accuracy and speed using neural networks, Bloom filter and Naive Bayes / O. Hryshchenko, V. Yaremenko. // Technology audit and production reserves. – 2021. – P6-8. - №5/2(61).

7. Яременко В. С., Тарасенко М. В. Порівняльний аналіз програмних бібліотек для класифікації текстових даних із використанням штучних нейронних мереж. Вчені записки ТНУ імені В.І. Вернадського. Серія: технічні науки. 2019. Том 30 (89), №3. С. 214–218.

8. Poloziuk K., Yaremenko V. Neural networks and Monte-Carlo method usage in multi-agent systems for sudoku problem solving. Technology audit and production reserves. 2020. №6/2 (56). P. 38-41.

## ДОДАТОК Б. Приклад протоколу спілкування агентів мовою ACL

### Протокол спілкування Agent Manager:

```
{

//2
"Query InterfaceAgent-ManagementAgent":{
  "performative" : "request",
  "sender"       : "agent-identifier :name InterfaceAgent",
  "receiver"     : "set (agent-identifier :name ManagementAgent)",
  "content"      : //"((process the text with id ))",
  "language"     : "fipa-sl",
  "ontology"     : "fipa-acl",
  "reply-with"   : "interface-management-query"
},

"ManagementAgent response": {
  "performative" : "inform",
  "sender"       : "agent-identifier :name ManagementAgent",
  "receiver"     : "set (agent-identifier :name InterfaceAgent)",
  "content"      : "((text id received))",
  "language"     : "fipa-sl",
  "ontology"     : "fipa-acl",
  "in-reply-to"  : "interface-management-query"
},

//3
"Query ManagementAgent-RequestAgent":{
  "performative" : "request",
  "sender"       : "agent-identifier :name ManagementAgent",
  "receiver"     : "set (agent-identifier :name RequestAgent)",,
  "content"      : "((specify task for text))",
  "language"     : "fipa-sl",
  "ontology"     : "fipa-acl",
  "reply-with"   : "management-request-query"
},

"RequestAgent response": {
  "performative" : "inform",
  "sender"       : "agent-identifier :name RequestAgent" ,
  "receiver"     : "set (agent-identifier :name ManagementAgent)",,
  "content"      : "((send to classification))",
  "language"     : "fipa-sl",
  "ontology"     : "fipa-acl",
  "in-reply-to"  : "management-request-query"
},

//4
"Query ManagementAgent-TextClassificationAgent":{
  "performative" : "request",
  "sender"       : "agent-identifier :name ManagementAgent",
  "receiver"     : "set (agent-identifier :name TextClassificationAgent)",,
  "content"      : "((classify the text))",
  "language"     : "fipa-sl",
  "ontology"     : "fipa-acl",
  "reply-with"   : "management-classification-query"
},
}
```



```

    "TextClassificationAgent response": {
        "performative" : "request",
        "sender"       : "agent-identifier :name TextClassificationAgent",
        "receiver"     : "set (agent-identifier :name ManagementAgent)",
        "content"      : "((send text id))",
        "language"     : "fipa-sl",
        "ontology"     : "fipa-acl",
        "in-reply-to"  : "management-classification-query",
        "reply-with"   : "classification-management-response"
    },

    "Query ManagementAgent-TextClassificationAgent textSending":{
        "performative" : "inform",
        "sender"       : "agent-identifier :name ManagementAgent",
        "receiver"     : "set (agent-identifier :name TextClassificationAgent)",
        "content"      : //"((text id is))",
        "language"     : "fipa-sl",
        "ontology"     : "fipa-acl",
        "reply-with"   : "management-text-sending",
        "in-reply-to"  : "classification-management-response"
    },

    "TextClassificationAgent classifiedText": {
        "performative" : "inform",
        "sender"       : "agent-identifier :name TextClassificationAgent",
        "receiver"     : "set (agent-identifier :name ManagementAgent)",
        "content"      : //"((text is classified as ))",
        "language"     : "fipa-sl",
        "ontology"     : "fipa-acl",
        "in-reply-to"  : "management-text-sending"
    },

    //5
    "Query ManagementAgent-VocabularyBuilderAgent":{
        "performative" : "request",
        "sender"       : "agent-identifier :name ManagementAgent",
        "receiver"     : "set (agent-identifier :name VocabularyBuilderAgent)",
        "content"      : "((build vocabulary))",
        "language"     : "fipa-sl",
        "ontology"     : "fipa-acl",
        "reply-with"   : "management-builder-query"
    },

    "VocabularyBuilderAgent response": {
        "performative" : "request",
        "sender"       : "agent-identifier :name VocabularyBuilderAgent",
        "receiver"     : "set (agent-identifier :name ManagementAgent)",
        "content"      : "((send text id and class))",
        "language"     : "fipa-sl",
        "ontology"     : "fipa-acl",
        "in-reply-to"  : "management-builder-query",
        "reply-with"   : "builder-management-response"
    },

    "Query ManagementAgent-VocabularyBuilderAgent textSending":{
        "performative" : "inform",
        "sender"       : "agent-identifier :name ManagementAgent",
        "receiver"     : "set (agent-identifier :name VocabularyBuilderAgent)",
        "content"      : //"((text id. class))",
        "language"     : "fipa-sl",
        "ontology"     : "fipa-acl",
        "reply-with"   : "management-vocabulary-get-text",

```



```

        "in-reply-to" : "builder-management-response"
    },
    "VocabularyBuilderAgent readyVocabulary": {
        "performative" : "inform",
        "sender" : "agent-identifier :name VocabularyBuilderAgent",
        "receiver" : "set (agent-identifier :name ManagementAgent)",
        "content" : //"((vocabulary is build. vocabulary id is ))",
        "language" : "fipa-sl",
        "ontology" : "fipa-acl",
        "in-reply-to" : "management-vocabulary-get-text"
    },
    //6
    "Query ManagementAgent-ResultAgent":{
        "performative" : "request",
        "sender" : "agent-identifier :name ManagementAgent",
        "receiver" : "set (agent-identifier :name ResultAgent)",
        "content" : //"((process results: text class, vocabulary id))",
        "language" : "fipa-sl",
        "ontology" : "fipa-acl",
        "reply-with" : "management-result-query"
    },
    "ResultAgent response": {
        "performative" : "inform",
        "sender" : "agent-identifier :name ResultAgent",
        "receiver" : "set (agent-identifier :name ManagementAgent)",
        "content" : "((results are processed))",
        "language" : "fipa-sl",
        "ontology" : "fipa-acl",
        "in-reply-to" : "management-result-query"
    },
}

```

## Протокол спілкування Result Agent:

```

{
    //7
    "Query ResultAgent-DataAgent":{
        "performative" : "request",
        "sender" : "agent-identifier :name ResultAgent",
        "receiver" : "set (agent-identifier :name DataAgent)",
        "content" : "((save results into local storage: text class, vocab id))",
        "language" : "fipa-sl",
        "ontology" : "fipa-acl",
        "reply-with" : "result-data-query"
    },
    "DataAgent response": {
        "performative" : "inform",
        "sender" : "agent-identifier :name DataAgent",
        "receiver" : "set (agent-identifier :name ResultAgent)",
        "content" : "((results are saved into local storage))",
        "language" : "fipa-sl",
        "ontology" : "fipa-acl",
        "in-reply-to" : "result-data-query"
    }
}

```