**УДК 004.8**

**Л. І. Кужій[1], І-Р. Р. Хархаліс[1], А. М. Худий[2]**
[1]Львівський коледж Державного університету телекомунікацій,
[2]Національний університет "Львівська політехніка"

# ПОБУДОВА СИСТЕМИ ДІАЛОГУ СТУДЕНТ-ПК НА БАЗІ ПРИРОДНОЇ МОВИ ОБМІНУ З ВИКОРИСТАННЯМ СЕРЕДОВИЩА JAVA



**Проведено огляд і здійснено аналіз засобів обробки природної мови (ОПМ). Визначено глобальний критерій оцінювання ефективності засобів ОПМ. Подано рекомендації щодо доцільності використання засобів ОПМ.**

**Ключові слова: обробка природної мови, штучний інтелект, інженерія текстів, діалог студент–ЕОМ.**

**The review and analysis of natural language processing methods (NLP) was carried out. Definitely a global criterion for evaluating the effectiveness NLP funds. Recommendations on the expediency of the use NLP funds are given.**

**Key words: natural language processing, artificial intelligence, text engineering, student-computer dialogue.**

## Вступ

Задача обробки природної мови (ОПМ), попри свою важливість і актуальність, є надзвичайно складною. У галузі штучного інтелекту (Artificial Intelligence – AI) ця задача вважається AI-повною (AI-complete, AI-hard), оскільки її обчислювальна складність еквівалентна до складності вирішення головного завдання штучного інтелекту – створення комп'ютерів, настільки ж розумних, як і люди [1, 2]. Розв'язання задачі ОПМ потребує глибоких знань системи.

Через важливість і складність задачі оброблення природної мови сьогодні вже створено багато засобів і під час розроблення нового забезпечення варто оглянути здобутки, досягнуті у цій галузі, щоб ураховувати їх. В цьому сенсі засоби для мови Java становлять неабиякий інтерес, бо, крім штатних засобів для цієї мови, написано багато бібліотек, які надають потужну підтримку ОПМ і дають змогу подолати комплексні складні проблеми [6, 13, 14].

## Постановка проблеми

У роботі розглянуто й оцінено засоби оброблення природної мови (або теж Natural Language Processing – NLP). Для вибору оптимального засобу розв'язання задачі ОПМ необхідно визначити глобальний критерій оцінювання ефективності засобів ОПМ.

Щоб зорієнтуватися у класичних підходах до розв'язання задачі ОПМ та використанні типових прийомів і стандартів цієї галузі, в роботі проаналізовано продукти найбільших світових лідерів. До найуживаніших з них належать бібліотеки LingPipe, Stanford API, пакет Apache OpenNLP, фреймворки GATE і Apache UIMA [6, 13, 14].

## Аналіз останніх досліджень та публікацій

Бібліотеки LingPipe – один із найзріліших і широко використовуваних у промисловості набір java-інструментів для обробки тексту, який містить компоненти, здатні виконати практично будь-яке завдання. LingPipe забезпечує всебічну Java-документацію і навчальні програми через свій вебсайт, але це складні для розуміння матеріали. Найкраще LingPipe задовольняє вимоги досвідчених Java-розробників, які розуміють основні концепції ОПМ [16].

Бібліотека групи Stanford NLP приділяє багато уваги пошуку залежностей між словами, будуючи дерева синтаксичного аналізу. Бібліотека характеризується добре організованим API і широким використанням. Вона складається з наборів інструментів, які поширюються під загальною публічною ліцензією (General Public License – GPL) GNU, але які, загалом, не дозволено використовувати в комерційних застосунках.

Проект Apache OpenNLP складається із декількох бібліотек. Існує достатньо повна документація як для користувача, так і для розробника інтегрованих рішень. Цей пакет є хорошим вибором, який легко отримати і з яким легко працювати як досвідченим розробникам, так і початківцям [6].

GATE (General Architecture for Text Engineering) – загальна архітектура для інженерії текстів. Це написаний в Java пакет розробника для аналізу текстів. Цей фреймворк полегшує розроблення складних систем, забезпечуючи стандартну структуру коду програми, полегшуючи тим самим швидке створення готового програмного продукту.

UIMA (Unstructured Information Management Architecture) – архітектура для управління дослідженням неструктурованої інформації. Це загальний фреймворк, в якому вкладено засоби обробки природної мови – збірний програмний пакет і, одночасно, архітектура для аналізу неструктурованої інформації [4].

**Формулювання цілей статті**

Для того щоб максимально ефективно обробляти природну мову, система обробки повинна охоплювати шість рівнів аналізу [3]. За лінгвістичного підходу зазвичай виокремлюють чотири рівні: графематичний – виділення розділів, абзаців, речень тощо; морфологічний – визначення характеристик окремого слова; синтаксичний – встановлення синтаксичної залежності слів у реченнях. Четвертий рівень – аналіз семантики та прагматики – особливо складний, що зумовлено складністю людської мови [4]. Оскільки остаточна наша мета – побудова галузевої спеціалізованої бази знань у сучасному уніфікованому форматі, нас цікавить власне останній рівень. Для нього тепер у практиці зазвичай потрібно виконати такі завдання [6]:

- поділ тексту на частини (Finding Parts of Text);
- виділення речень (Finding Sentences);
- виділення (відокремлення) елементів тексту – токенізація (tokenization);
- визначення частин мови (Parts of Speech tagging – POS-tagging);
- видобування зв’язків (Extracting Relationships);
- пошук людей і речей (Finding People and Things);
- класифікація текстів і документів (Classifying Text and Documents).

За рангом важливості наведені задачі можна розподілити так, як це подано в табл. 1.

*Таблиця 1*

**Розподіл основних завдань за рангом важливості стосовно мети роботи**

| Завдання | Ранг важливості |
|---|---|
| Поділ тексту на частини | 7 |
| Виділення речень | 4 |
| Токенізація | 3 |
| Визначення частин мови | 2 |
| Видобування зв’язків | 1 |
| Розпізнавання іменованих об’єктів | 5 |
| Класифікація текстів | 6 |

Згадані вище засоби ОПМ по-різному пристосовані до розв’язання цих задач. Тому для отримання порівняльної оцінки тих чи інших засобів стосовно досягнення мети роботи варто зупинитися детальніше на особливостях як засобів, так і задач.

### Токенізація

Токенізацією (tokenization, інколи також – скануванням) називають процес розділення тексту на прості одиниці (токени – tokens) для подальшого аналізу. Це осмислені групи символів (напр., слова), що відповідають певним шаблонам [5]. Токенізація використовується як початковий крок у багатьох задачах ОПМ. Це фундаментальний і основний крок. Для його реалізації можна використати декілька базових класів Java, а також деякі API-інтерфейси ОПМ.

Перед токенізацією або під час її виконання часто використовують т. зв. нормалізацію. Це процес, який перетворює текст на одноріднішу послідовність [6].

Що стосується безпосередньо токенізації, то часто можна скористатись простою її технікою, що ґрунтується на основних Java-класах. Таку методику пропонує, наприклад, Java клас Scanner. Цей клас, однак, швидше за все, не справиться з розділенням словоскорочень, багатокрапок та інших ускладнень. Для виділення простих токенів непоганою може виявитися методика, основана на використанні методу indexOf () класу BreakIterator, а також методу split () класу String. Поза тим, можливості засобів комплекту Java SE SDK все ж обмежені й у них наявні також проблеми продуктивності.

Водночас токенізація з використанням API ОПМ надає широкий вибір методів виконання і можливостей зміни параметрів процесу. Бібліотека LingPipe підтримує декілька токенізаторів класом IndoEuropeanTokenizerFactory для токенізації “нормального” тексту з можливістю навчання для роботи з унікальним текстом. Класом TokenME можна створити специфічну модель токенізатора, здатну ідентифікувати різні символи пунктуації.

API Stanford NLP підтримує токенізацію декількома класами. Метод setTokenizerFactory () дає змогу змінювати токенізатори. За замовчуванням для токенізації введення використовується клас PTBTokenizer, що надає різні засоби для управління поведінкою токенізатора, а також різні варіанти відображення токенів. Клас DocumentPreprocessor спрощує токенізацію речення для простого тексту і XML-даних.

Apache OpenNLP для токенізації надає інтерфейс Tokenizer, який реалізується трьома класами: SimpleTokenizer, TokenizerME і WhitespaceTokenizer. Класи SimpleTokenizer і WhitespaceTokenizer виконують просту токенізацію тексту. Клас TokenizerME використовує статистичну модель максимальної ентропії (Maximum Entropy), що покращує якість токенізації, особливо таких джерел, як соціальні мережі, які використовують багато сленгу і спеціальних символів, наприклад смайликів [6].

Експертні оцінки щодо розв’язання задачі токенізації для засобів ОПМ, які ґрунтуються на наведеній інформації, подано в табл. 2. Ці оцінки основані на припущенні, що в цьому випадку не варто покладатися на прості методи токенізації, бо в текстах може бути багато специфічних термінів і абревіатур. Засоби GATE і UIMA, можливо, заслуговують на вищі оцінки, але як фреймворки їх не розглядатимемо на цьому етапі розроблення системи.

*Таблиця 2*

**Оцінка засобів ОПМ для токенізації**

| Токенізація – ранг важливості 3 | | | | | |
|---|---|---|---|---|---|
| Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE | UIMA |
| 0,5 | 1,0 | 0,9 | 1,0 | 0,5 | 0,5 |

### Розділення тексту на речення (SBD)

Java стандартно надає декілька методів визначення кінця речення. Можна використати регулярні вирази і метод getSentenceInstance () класу BreakIterator. Однак, коли текст ускладнюється, використання цих засобів стає проблематичним і краще звернутися до API-інтерфейсів ОПМ [6].

API ОПМ, котрі підтримують SBD, можуть бути основані на правилах, а також використовувати т. зв. моделі – спеціальні програми, попередньо навчені на відповідних (спеціалізованих) текстах. Щоб отримати найкращу модель, набори даних повинні бути якнайбільші.

Для оцінювання якості підготовки моделі визначають точність (Precision – P), повноту (відкликання, recall – R), а також міру ефективності $F_1$ (F-measure) [19]. Точність являє собою частку віднайдених релевантних об'єктів (пошуку чи аналізу), тоді як повнота – частку всіх знайдених об'єктів. F-міра, або також збалансована F-оцінка, поєднує точність і повноту й обчислюється як їх середнє гармонічне [19]:

$$F_1 = 2 \cdot P \cdot R/(P + R).$$

Використання техніки моделей широко підтримує бібліотека LingPipe. Для визначення меж речень у LingPipe передбачено цілу низку класів: HeuristicSentenceModel, IndoEuropeanSentenceModel, SentenceChunker тощо. Ці класи можуть використовувати додаткові правила токенізації, а також т. зв. чанкінг (chunking) – конкретизувати, які частини "фрагменти – chunks" тексту необхідно обробляти. Ці частини творять структуру, використання якої дає змогу підвищити точність аналізу семантики текстів.

Apache OpenNLP для визначення речень використовує модель з класу SentenceModel. За допомогою цієї моделі створюється екземпляр класу SentenceDetectorME і викликається метод sentDetect (), який повертає масив рядків, кожним елементом якого є речення. Модель SBD OpenNLP показує добрі результати як для простих, так і для складніших речень [6].

Stanford NLP може виконувати SBD, використовуючи різні підходи. Клас PTBTokenizer використовує правила для виконання SBD і має багато параметрів налаштування токенізації. Клас DocumentPreprocessor створює список речень у вигляді простого тексту або документа XML. Засоби Stanford NLP для виявлення речень навчені на наборі банку даних Penn і, отже, більше задовольняють вимоги у напрямку до формального english.

Оцінки засобів ОПМ для SBD наведені в табл. 3 і враховують обмеженість методів Java щодо аналізу складних речень. LingPipe, Stanford NLP і Apache OpenNLP дають змогу будувати якісні моделі та забезпечити високу точність і для простих, і для складних речень.

*Таблиця 3*

**Оцінка засобів ОПМ для SBD**

| Розділення тексту на речення – ранг важливості 4 | | | | | |
|---|---|---|---|---|---|
| Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE | UIMA |
| 0,4 | 0,9 | 1,0 | 0,9 | 0,5 | 0,5 |

**Пошук людей і речей (NER)**

Для оброблення фахових текстів галузі ця задача важлива через необхідність визначення змісту тексту, виявлення і оброблення семантичних відношень. Категоріями NER можуть бути імена людей, географічні назви, назви організацій, час, URL-адреси тощо. Процес NER містить у собі дві задачі: виявлення (пошук) об'єктів і їх класифікацію (Spotting) – призначення класів сутностям. До того ж NER може також знімати неоднозначність – омонімію (Disambiguation), фільтрувати (Filtering) нецікаві або мало пов'язані з метою об'єкти.

В Java для ідентифікації іменованих об'єктів можна використати списки "стандартних" об'єктів разом із регулярними виразами. В багатьох ситуаціях така методика дає змогу точно виділити потрібні об'єкти і цілком прийнятна для нашого випадку.

У бібліотеці LingPipe для пошуку іменованих об'єктів є клас RegExChunker. Його метод chunk () повертає об'єкт фрагментів Chunking, які використовують для пошуку сутностей.

Stanford API для розпізнавання іменованих сутностей використовує клас CRFClassifier. Цей клас реалізує так звану модель послідовності умовних випадкових полів (Conditional Random Field CRF).

У бібліотеці Apache OpenNLP задачу NER виконує метод find () класу NameFinderME, створеного, своєю чергою, за допомогою класу TokenNameFinderModel. Для навчання відповідної моделі використовують тренувальний файл, що містить мітки для демаркації сутностей. Отриману модель можна оцінити за допомогою класу TokenNameFinderEvaluator, який обчислює імовірність для об'єктів, ідентифікованих у тексті.

У підсумку варто зауважити, що для NER галузевих текстів можливо доволі ефективно використовувати широкий спектр засобів. Оцінки як штатних засобів Java, так і класів API для NER наведені в табл. 4 і є, загалом, близькими і високими.

*Таблиця 4*

**Оцінка засобів ОПМ для NER**

| Пошук людей і речей – ранг важливості 5 | | | | | |
|---|---|---|---|---|---|
| Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE | UIMA |
| 0,8 | 1,0 | 0,9 | 0,9 | 0,5 | 0,5 |

**Визначення частин мови (POS tagging)**

Частини мови визначають за допомогою POS-маркування (Parts of Speech tagging, POS tagging) – призначенням опису (тегу – tag**)** для токена або елемента тексту. Ті чи інші теги відповідають частинам мови, таким як іменник, дієслово, прикметник тощо. Наприклад, у Stanford NLP позначають іменники IN, дієслова NN, NNS тощо.

Загальний процес маркування складається із токенізації тексту, визначення можливих тегів і встановлення їх неоднозначності. Для ідентифікації POS використовують два основні підходи:

• оснований на правилах: коли тегувальник використовує словник і набір можливих тегів;

• стохастичний, оснований на моделях максимальної ентропії або на марковських моделях.

З бібліотеками LingPipe постачаються три моделі POS, наприклад, модель, яка ґрунтується на Brown Corpus. У LingPipe для тегування POS є метод tag (), доступний з різних класів. Він повертає екземпляр List об’єктів Tagging, які є словами та їх тегами. Клас HmmDecoder використовує метод tag () для визначення найімовірніших (перших, найкращих) тегів. Метод tagNBest () оцінює можливу розмітку і повертає ітератор п’яти різних передбажаних тегів. Клас ScoredTagging володіє методом оцінювання якості роботи. LingPipe класом MaxentTagger може обробляти специфічні скорочення (textese) [6]. POS tagging у бібліотеці LingPipe використовує багато методів нормалізації, уможливлюючи конвеєрне оброблення. Засоби класів StopWords і EnglishStopTokenizerFactory ідентифікують і видаляють стоп-слова. Для покращення якості процесу тегування POS використовується стемінг (клас PorterStemmerTokenizerFactory).

Стемінг і лематизація: ці процеси змінюють форму слів, зводячи їх до базової, початкової форми, до їх “коренів”. Стемінг (англ. stemming) – це процес визначення основи слова із відтинанням частин початку і/або закінчення токена, видаленням префіксів чи суфіксів, а остачу вважають основою (причому така основа не завжди збігається з коренем слова). Інструментом стемінгу є Porter Stemmer, який широко використовують для англійської мови. Porter Stemmer можна використати з LingPipe і OpenNLP. Існують також інші типи засобів стемінгу.

Стемінг, загалом, є доволі примітивною технікою, тому часто здійснюється з використанням лематизації. Лематизація (lemmatization) – процес приведення словоформи до леми (lemma) – її нормальної (початкової, основної, словникової) форми. Наприклад: для іменників – називний відмінок, однина; для прикметників – називний відмінок, однина, чоловічий рід; для дієслів – дієслово в інфінітиві тощо. Лематизація – процес складніший і тонший, ніж стемінг. Для визначення леми використовують словники форм (напр. словники WordNet) і морфологічні методики [12]. Прикладами засобів лематизації є класи StanfordCoreNLP і OpenNLPLemmatizer.

Алгоритми стемінгу, основані на лематизації, можуть забезпечувати дуже високу якість і мінімальний відсоток помилок. Остаточно лематизація дає іншу основу слова, ніж стемінг, і уможливлює точніший аналіз тексту. Точність – важливий для нас параметр, тому під час оцінювання засобів беруть до уваги наявність цього апарату.

У Stanford NLP для визначення частини мови призначений пакет Part_of_Speech Tagger, використання якого забезпечує дуже високу точність (понад 90 %). Для POS-тегування Stanford API підтримує клас MaxentTagger з використанням максимальної ентропії. Цей клас має різноманітні моделі, які можуть ефективно обробляти ускладнення в тексті, зокрема абревіатури типу textese. Для цього класу GATE також розробила свою модель, призначену для оброблення текстів твітера [6].

Тегери Stanford NLP використовують набір POS-тегів Penn Treebank. У комплекті з API постачаються моделі, основані на Wall Street Journal. Є можливість вибрати відповідну модель, використовуючи властивість pos. model.

Оброблення для визначення POS можна здійснити з використанням конвеєра – набором анотаторів tokenize, ssplit і pos, які, відповідно, токенізують, розділяють текст на речення, а потім шукають теги POS. У конвеєрі також є можливість здійснення лематизації анотатором lemma. Для лематизації передбачено також окремий клас StanfordLemmatizer.

Stanford API підтримує вирішення (розв'язання) кореференції (coreference resolution), тобто завдання знаходження усіх згадувань того самого об'єкта під різними іменами, важливе для правильної інтерпретації тексту. Кореференція виникає через проблеми співвідношення мислення і мови [17]. У разі вживання кореферентних виразів предмет перебуває у певному відношенні до тексту і до ситуації висловлювання. Алгоритми для вирішення кореференції повинні забезпечувати точність близько 75 % відповідно до оцінки точності та повноти [18]. Вирішення кореферентності реалізується класом StanfordCoreNLP, і також анотатором dcoref під час створення конвеєра. Метод get () класу annotation, використаний з аргументом CorefChainAnnotation. class, поверне екземпляр Map об'єктів CorefChain. Ці об'єкти міститимуть інформацію про кореференції, знайдені в реченнях. Клас CorefMention містить детальнішу інформацію про конкретну кореференцію [6].

OpenNLP для визначення частин мови надає декілька класів, які походять з класу POSModel і основані на наборі тегів Penn TreeBank. Для отримання основних тегів використовується клас POSTaggerME на основі максимальної ентропії. Тегер визначає тип тега на підставі самого слова і його контексту. Будь-яке задане слово отримує декілька пов'язаних з ним тегів. Метод topKSequences () поверне набір тегів POS, ґрунтуючись на їх передбажаній імовірності.

Для покращення тегування і уникнення неправильного застосунку тега до слова можна скористатися словником тегів класу POSDictionary, створивши його з відповідного XML-файла. В словнику теги відображаються, як, наприклад /NN/VBP/VB, тобто слово можливо інтерпретувати трьома різними способами. У разі POS тегінгу OpenNLP підтримується нормалізація і лематизація з використанням класу JWNLDictionary і конвеєра. Використання файлів цього словника істотно покращує ідентифікацію коренів. Крім того, можна завантажити із сайта і додати в проект вихідний код класу PorterStemmer [6].

Додаткову інформацію про структуру речення може дати використання чанкінгу (клас ChunkerME). Чанкінг передбачає розподіл речення на частини (блоки) і групування пов'язаних слів відповідно до їх типів. Ці блоки потім можуть бути анотовані тегами. OpenNLP є також одним з небагатьох пакетів, які підтримують вирішення кореференції.

Підсумкові оцінки засобів ОПМ для POS tagging наведені в табл. 5 і ґрунтуються на тому, що штатні засоби Java для визначення частин мови відсутні. Водночас засоби API ОПМ, використовуючи багату техніку, забезпечують високу точність визначення POS у складних сучасних умовах.

*Таблиця 5*

**Оцінка засобів ОПМ для POS tagging**

| Визначення частин мови – ранг важливості 2 | | | | | |
|---|---|---|---|---|---|
| Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE | UIMA |
| 0,0 | 1,0 | 1,0 | 1,0 | 0,5 | 0,5 |

**Класифікація тексту і документів**

Розрізняють текстові класифікацію і кластеризацію (clustering). В обох випадках визначають відповідність документа одній з декількох груп (категорій). Кластеризація пов'язана з ідентифікацією тексту без використання наперед визначених категорій. Класифікація використовує наперед визначені категорії. Під час класифікації мітки (теги) відомі наперед, а в разі кластеризації мітки (теги) наперед невідомі. Для нашого випадку як класифікація, так і кластеризація важливі для визначення семантики контексту – тем і підтем, розділів і підрозділів у тексті.

Найпоширеніші два методи класифікації тексту: на основі правил і контрольованого машинного навчання (Supervised Machine Learning, SML). Класифікація на основі правил використовує комбінацію слів та інших атрибутів, організованих навколо розроблених експертами правил. Вона може бути дуже ефективною, але для створення правил потрібно багато часу [6].

Контрольоване машинне навчання (SML) використовує колекцію анотованих навчальних документів для створення моделі – класифікатора (classifier). В нашому випадку варто орієнтуватися на обидва підходи, можливо, ефективнішими будуть все ж правила. Класифікація на основі правил теоретично може бути реалізована в Java, але створення правил займає багато часу. NLP API OpenNLP, Stanford API й LingPipe демонструють різні підходи до класифікації. LingPipe потребує уважнішого розгляду, оскільки він підтримує деякі задачі класифікації, зокрема загальну класифікацію тексту, використовуючи підготовані моделі. У разі загальної класифікації спочатку виконується навчання класу DynamicLMClassifier, а потім фактична класифікація методом classify (). LingPipe постачається з моделлю langid-leipzig. classifier, навченою для декількох мов.

OpenNLP підтримує процес класифікації через інтерфейс OpenNLP DocumentCategorizer, який реалізується класом DocumentCategorizerME. Цей клас класифікуватиме текст у наперед визначені категорії, використовуючи фреймворк максимальної ентропії. Категоризацію і класифікацію тексту підтримує також клас DoccatModel. Потрібно навчити модель, користуючись класом DocumentCategorizerME. Метод getCategory () класу DocumentCategorizerME повертає масив, кожний елемент якого містить ймовірність того, що текст належить до цієї категорії. Для визначення найкращої категорії можна використати метод getBestCategory.

Stanford NLP підтримує декілька класифікаторів для загальної класифікації в класі ColumnDataClassifier. Використання класифікаторів, підтримуваних API Stanford, може бути утрудненим [6].

Специфічний різновид класифікації – це аналіз настроїв (sentiment analysis). Він, зазвичай, пов’язаний з визначенням позитивного чи негативного ставлення стосовно конкретного продукту чи теми. Аналіз настроїв можна застосовувати до речення, пункту чи усього документа. Результат може бути як позитивним, так і негативним, або також рейтингом з використанням числових значень, наприклад, від 1 до 10. OpenNLP використовує тільки дві категорії – позитивну і негативну. Stanford NLP підтримує складнішу шкалу – дуже негативна – негативна – нейтральна – позитивна – дуже позитивна.

Щоб з’ясувати, які слова виражають типи настроїв, використовують лексикони почуттів – словники, які відображають сентиментальний зміст різних слів. Одним з таких лексиконів є The General Inquirer. Він містить 1915 слів, які вважають позитивними. Існують й інші лексикони, такі як MPQA Subjectivity Cues Lexicon. SentiWordNet – лексичний ресурс (семантичний тезаурус) для видобування суджень (опіній, оцінок) і аналізу емоцій. Для кожного із синсетів у WordNet SentiWordNet визначає оцінку з трьох значень – позитивність, негативність і об’єктивність [8].

Остаточну оцінку засобів ОПМ для класифікації наведено в табл. 6. Ця оцінка враховує, що для наших завдань (оброблення науково-технічних текстів) потреба аналізу емоцій видається не вельми важливою, хоча під час вирішення деяких конкретних завдань така потреба може обмежено виникати.

*Таблиця 6*

**Оцінка засобів ОПМ для класифікації**

| Класифікація тексту і документів – ранг важливості 6 | | | | | |
|---|---|---|---|---|---|
| Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE | UIMA |
| 0,4 | 1,0 | 0,8 | 0,9 | 0,5 | 0,5 |

**Видобування відношень (parsing)**

Визначення відношень між POS, фразами, пунктами тощо прийнято називати розбором – парсингом (parsing). Парсинг тісно пов'язаний з токенізацією, з визначенням частин мови і їх взаємовідносинами. На відміну від токенізації, парсинг повертає відношення, які існують в тексті, створюючи дерево розбору – ієрархічну структуру даних, яка відображає синтаксичну структуру речення, показує взаємозв'язок між граматичними елементами.

Існує багато можливих типів відношень, як-от Freebase, Resource Description Framework (RDF), DBPedia, WordNet. На особливу увагу заслуговує WordNet, – ієрархічно організована лексична база даних, широко використовувана в додатках ОПМ. Існує версія, локалізована до російської та української мов [9]. WordNet є прикладом онтологій – відображення буття у формалізованому вигляді, формальна структура, яка спрощує комп'ютерну обробку, даючи змогу застосункам розпізнавати семантичні відмінності [10]. У WordNet представлені не слова, а сенси (концепти) понять і кожному сенсу відповідає свій комплект слів – „синсет", які його позначають. Синсети зв'язані між собою за допомогою добре визначених семантичних відношень [11].

В Stanford NLP для синтаксичного аналізу речення призначений Java-пакет Stanford Parser. Він виконує імовірнісний аналіз з використанням імовірнісних контекстовільних граматик – PCFG (probabilistic context-free grammars). Клас LexicalizedParser (лексикалізований PCFGs-парсер) цього пакета забезпечує високу точність аналізу. Клас GrammaticalStructure (доступний також онлайн) визначає залежності слів, формує дерева синтаксичного аналізу, будуючи граф залежностей між елементами речення. Клас StanfordCoreNLP підтримує конвеєр з трьох анотаторів, в якому: tokenize – токенізує текст; ssplit – виділяє речення; parse – виконує синтаксичний аналіз тексту. Результат аналізу в Stanford NLP може відображатися з використанням графічного Java-інтерфейсу класу TreePrint, що надає метод роботи з деревами.

Парсер OpenNLP дає змогу виконати два типи синтаксичного аналізу – залежностей (Dependency) – взаємозв'язків між словами і фразової структури (Phrase structure) у реченні. Для розбору тексту використовують клас ParserTool. Його метод parseLine () повертає екземпляр Parser, що містить результати парсингу – парси (parses) у послідовності зростання їх імовірності. Метод getChildren () повертає масив об'єктів Parse. Використовуючи різні методи Parse, можна отримати теги і мітки для елементів тексту.

Оцінки засобів ОПМ для видобування відношень наведено в табл. 7. Їх встановили, враховуючи, що найбагатшим апаратом для парсингу володіє пакет Stanford Parser, який може забезпечити високу точність аналізу. Поза тим розбір засобами Java потребує великих зусиль і для цієї ситуації практично нездійсненний.

*Таблиця 7*

**Оцінка засобів ОПМ для видобування відношень**

| Видобування відношень – ранг важливості 1 | | | | | |
|---|---|---|---|---|---|
| Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE | UIMA |
| 0,0 | 0,7 | 1,0 | 0,8 | 0,5 | 0,5 |

У табл. 1–7 наведено оцінки засобів ОПМ стосовно їх спроможності (ефективності) вирішувати основні завдання оброблення природної мови, а також вказано оцінки (ранги) важливості цих завдань стосовно мети нашого дослідження відповідно до табл. 1.

Поза тим, важлива також можливість порівняльної оцінки цих засобів стосовно їх певних внутрішніх і зовнішніх властивостей, цікавих для нас з погляду користувача і розробника майбутньої системи. Серед цих властивостей – такі як складність освоєння, багатство функцій, наявність та потужність засобів для роботи із синтаксичними деревами, можливості оцінки точності, робота з моделями, підтримка конвеєрної обробки. Безперечно, важливі й використання стандартизації, можливості семантичного аналізу тощо.

**Оцінка засобів ОПМ для їх вибору**

Як бачимо, завдання вибору базового забезпечення ускладнене численними вимогами і властивостями, тому для прийняття рішення доцільно сформулювати та розв'язати задачу багатокритеріальної оцінки. Для цього застосовують багато методів, для аналізованого випадку придатний метод згортки критеріїв або метод комплексного критерію, його різновид – метод Гермеєра [20]. В цьому методі остаточне рішення приймають на підставі глобального критерію Q, а для його обчислення використовують функцію:

$$Q = \sum_i l_i(W_i) \cdot W_i, \qquad (1)$$

де $l_i$ – коефіцієнт значущості *i*-го (локального) показника якості (критерію $W_i$). Переважно при цьому дотримуються умови

$$\sum_i l_i(W_i) = 1.$$

Зазвичай $l_i$ визначають за допомогою методу експертних оцінок, що в цьому випадку цілком прийнятно. Оскільки на практиці вага критеріїв суттєво зменшується з їх рангом, для коректної оцінки доцільно використати квазігіперболічну залежність $l_i(W_i)$ [20]. Приймемо орієнтовно $l_i(W_i) = 1/(1+W_i)$. Користуючись правами експерта, виберемо остаточно для згортки критеріїв табл. 2–7 при *i* = $[\overline{1,6}]$: $l_i(W_i)$ = {0,4; 0,25; 0,2; 0,1; 0,04; 0,01}. Для отримання підсумкових висновків результати табл. 2–7 згруповано в табл. 8, причому в чисельниках продубльовано ненормовані значення (відповідно до табл. 2–7), а в знаменниках – нормовані $l_i(W_i) \cdot W_i$. Наведено також значення підсумкового критерію оцінки щодо розв'язання задач, обчисленого згідно з (1).

*Таблиця 8*

**Оцінка ефективності засобів ОПМ для розв'язання основних задач**

| Задача | Ранг | $l_i(W_i)$ | Java | LingPipe | Stanford NLP | Apache Open NLP | GATE і Apache UIMA |
|---|---|---|---|---|---|---|---|
| Видобування відношень | 1 | 0,4 | 0,0/0,0 | 0,7/0,28 | 1,0/0,4 | 0,8/0,32 | 0,5/0,2 |
| Визначення частин мови | 2 | 0,25 | 0,0/0,0 | 1,0/0,25 | 1,0/0,25 | 1,0/0,25 | 0,5/0,125 |
| Токенізація | 3 | 0,2 | 0,5/0,1 | 1,0/0,2 | 0,9/0,18 | 1,0/0,2 | 0,5/0,1 |
| Визначення речень | 4 | 0,1 | 0,4/0,04 | 0,9/0,09 | 1,0/0,1 | 0,9/0,09 | 0,5/0,05 |
| Пошук людей і речей | 5 | 0,04 | 0,8/0,032 | 1,0/0,04 | 0,9/0,036 | 0,9/0,036 | 0,5/0,02 |
| Класифікація | 6 | 0,01 | 0,4/0,004 | 1,0/0,01 | 0,8/0,008 | 0,9/0,009 | 0,5/0,005 |
| Підсумковий критерій оцінки щодо розв'язання задач | | | 0,5 | 0,87 | 1,37 | 0,986 | 0,475 |

Для порівняльної оцінки засобів ОПМ стосовно їх функціональних особливостей виберемо відповідні критерії та їх ранги, як подано в табл. 9. Для зважування критеріїв у табл. 9 за рангом приймемо *i*=$[\overline{1,5}]$: $l_i(W_i)$ = {0,45; 0,25; 0,15; 0,1; 0,05}.

Для остаточної згортки підсумкових критеріїв і отримання глобального критерію приймемо $l_i(W_i)$ = {0,6; 0,4}. Виконаємо розрахунки підсумкових та глобального критеріїв і подамо результати в табл. 10.

*Таблиця 9*

**Оцінка ефективності засобів ОПМ за функціональними особливостями**

| Функціональна особливість | Ранг | $l_i(W_i)$ | Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE і Apache UIMA |
|---|---|---|---|---|---|---|---|
| Складність освоєння | 1 | 0,45 | 1,0 / 0,45 | 0,5 / 0,225 | 0,9 / 0,405 | 0,9 / 0,405 | 0,8 / 0,36 |
| Робота із синтаксичними деревами | 2 | 0,25 | 0,0 / 0,0 | 0,5 / 0,125 | 1,0 / 0,25 | 0,8 / 0,2 | 0,5 / 0,125 |
| Функціональність | 3 | 0,15 | 0,5 / 0,075 | 0,9 / 0,135 | 1,0 / 0,15 | 0,9 / 0,135 | 0,6 / 0,09 |
| Стандартизація | 4 | 0,1 | 0,5 / 0,05 | 0,5 / 0,05 | 0,5 / 0,05 | 0,5 / 0,05 | 1,0 / 0,1 |
| Аналіз семантики | 5 | 0,05 | 0,0 / 0,0 | 0,7 / 0,035 | 0,8 / 0,04 | 0,7 / 0,035 | 1,0 / 0,05 |
| Підсумковий критерій оцінки за властивостями | | | 0,575 | 0,57 | 0,895 | 0,825 | 0,725 |

*Таблиця 10*

**Визначення глобального критерію оцінки ефективності засобів ОПМ**

| Підсумкові та глобальний критерії | $l_i(W_i)$ | Java | LingPipe | Stanford NLP | Apache OpenNLP | GATE і Apache UIMA |
|---|---|---|---|---|---|---|
| За розв'язанням основних задач | 0,6 | 0,5 / 0,3 | 0,87 / 0,572 | 1,37 / 0,872 | 0,986 / 0,5916 | 0,475 / 0,285 |
| За властивостями засобів ОПМ | 0,4 | 0,57 / 0,23 | 0,57 / 0,228 | 0,895 / 0,358 | 0,825 / 0,33 | 0,725 / 0,29 |
| Глобальний критерій | | 0,53 | 0,8 | 1,23 | 0,9216 | 0,575 |

**Висновки**

Для проведення оцінки засобів ОПМ доцільно встановити пріоритетні ранги основних задач ОПМ, виділивши насамперед видобування відношень (парсинг), визначення частин мови (POS-tagging), а також токенізацію. Такий розподіл рангів дає змогу виділити забезпечення, ефективне для подальшого видобування інформації.

Штатні засоби Java, наявні в стандартному дистрибутиві, отримали невисоку оцінку, оскільки істотно обмежені. Наприклад, здійснення парсингу засобами Java потребує великих зусиль і для розглянутої ситуації фактично недоцільне. Відсутні штатні засоби Java для визначення частин мови. Також в цьому випадку не варто покладатися на прості методи токенізації, доступні в Java, бо в реальних текстах іноді багато специфічних термінів і абревіатур. Тому штатні засоби Java можна використовувати лише як допоміжні.

Проведена оцінка підтверджує той факт, що найцікавішими з погляду досягнення мети нашої роботи є засоби LingPipe, Stanford NLP і Apache OpenNLP. Вони дають змогу будувати якісні моделі та забезпечити високу точність розв'язання пріоритетних для нас задач НЛП у складних сучасних умовах.

За підсумками оцінки (табл. 10) для розроблення системи варто орієнтуватися на засоби Stanford NLP, враховуючи їхні переваги під час розв'язання основних задач і за особливостями функціональності. Ця бібліотека володіє найрозвиненішим апаратом для розв'язання пріоритетної для нас задачі ОПМ – парсингу, забезпечуючи багатий функціонал і високу точність аналізу. Також на початкових етапах розроблення системи доцільно орієнтуватися на засоби Apache OpenNLP, враховуючи їхні переваги в простоті освоєння.

Засоби GATE і UIMA, правдоподібно, заслуговують на вищі оцінки, але як фреймворки їх не розглядають на цьому етапі, бо їх жорстка структура може завадити ефективному використанню для цього випадку, хоча їхні потужність, засоби аналізу семантики і стандартизації, безсумнівно, становлять інтерес.

*1. Хархаліс І.-Р. Структура системи спілкування студент – ЕОМ з використанням штучного інтелекту при вивченні дисциплін телекомунікаційного спрямування / Ігор-Роман Хархаліс, Галина Ничай, Наталія Охремчук // Вітчизняна наука на зламі епох: проблеми та перспективи розвитку: матер. XXV Всеукраїнської наук.-практ. інтернет-конференції: зб. наук. праць. – Переяслав-Хмельницький, 2016. – Вип. 25. – С. 156–160. 2. AI-повна задача [Електронний ресурс]. – Режим доступу: https://uk. wikipedia. org/w/index. php? title=AI-повна_задача. 3. Обробка природної мови [Електронний ресурс]. – Режим доступу: https://uk. wikipedia. org/w/index. php? title=Обробка природної мови. 4. Синтаксичний аналіз [Електронний ресурс]. – Режим доступу: https://uk. wikipedia. org/w/index. php? title=Синтаксичний_аналіз. 5. Лексичний_аналіз [Електронний ресурс]. – Режим доступу: https:/uk. wikipedia. org/w/index. php? title=Лексичний_аналіз. 6. Richard M. Reese / Natural Language Processing with Java // Birmingham – Mumbai Packt Publishing, 2015. – 262 с. 7. Що таке корпус? [Електронний ресурс]. – Режим доступу: http://www. mova. info/help1. html. 8. SentiWordNet [Електронний ресурс]. – Режим доступу: http://sentiwordnet. isti. cnr. it/. 9. Сайт проекту WordNet [Електронний ресурс]. – Режим доступу: http://wordnet. princeton. edu/. 10. Константинова Н. С. Онтологии как системы хранения знаний [Електронний ресурс] / Н. С. Константинова, О. А. Митрофанова // Санкт-Петербург, Санкт-Петербургский государственный университет. – 54 с. – Режим доступу: http://www. ict. edu. ru/ft/005706/68352e2-st08. pdf. 11. Михайлюк А. Формування лінгвістичної онтології на базі структурованого електронного енциклопедичного ресурсу / Антон Михайлюк, Олена Михайлюк, Олексій Пилипчук, Володимир Тарасенко [Електронний ресурс]. – Режим доступу: http://elibrary. kubg. edu. ua/2876/1/A_Mykhailiuk_MZK_3_IS. pdf. 12. Лемматизация [Електронний ресурс]. – Режим доступу: https://ru. wikipedia. org/w/index. Php? title=Лемматизация. 13. NLP Tools Java [Електронний ресурс]. – Режим доступу: http://www. phontron. com/nlptools. php. 14. Программное обеспечение для обработки естественного языка [Електронний ресурс]. – Режим доступу: https://ru. wikipedia. org/wiki/ index. php? title=Программное обеспечение для обработки естественного языка. 15. Lingpipe-blog [Електронний ресурс]. – Режим доступу: http://lingpipe-blog. com/. 16. Breck Baldwin Natural Language Processing with Java and LingPipe Cookbook / Breck Baldwin, Krishna Dayanidhi // Mumbai. Packt Publishing, 2014. – 312 p. 17. Гак В. Г. Семантическая структура слова как компонент семантической структуры высказывания / В. Г. Гак. – М.: Наука 1971. – С. 78–96. 18. Оценка классификатора (точность, полнота, F-мера) [Електронний ресурс]. – Режим доступу: http://bazhenov. me/blog/2012/07/21/classification-performance-evaluation. html. 19. Precision and recall [Електронний ресурс]. – Режим доступу: https://en. wikipedia. org/wiki/Precision_and_recall. 20. Основи багатокритеріальної оптимізації [Електронний ресурс]. – Режим доступу: http://kxtp. kpi. ua/common/shakhnovsky_-_multigoal_l10. pdf.*