ВІДОМОСТІ ПРО АВТОРА

Євген Купріянов – кандидат філологічних наук, доцент кафедри інтелектуальних комп'ютерних систем Національного технічного університету «Харківський політехнічний інститут».

Наукові інтереси: прикладна лінгвістика, комп'ютерна лексикографія, науково-технічний переклад.

INFORMATION ABOUT THE AUTHOR

Yevhen Kupriianov – Candidate in Philology, Associate Professor at the Intelligent Computer Systems Department, National Technical University "Kharkiv Polytechnic Institute".

Research interests: applied linguistics, computer lexicography, translation of scientific and technical information.

УДК: 004.436.4:81'373.423

ПРОГРАМНА WSD СИСТЕМА ДЛЯ ВСТАНОВЛЕННЯ ЗНАЧЕНЬ ОМОНІМІВ

Ярослав ШЕВЧУК (Ізмаїл, Україна)

e-mail: yarolegovich@gmail.com

ШЕВЧУК Ярослав. ПРОГРАМНА WSD СИСТЕМА ДЛЯ ВСТАНОВЛЕННЯ ЗНАЧЕНЬ ОМОНІМІВ

В статті представлена розробка програмного забезпечення способу точного вибору значення слова з наявних омонімів з урахуванням семантичного контексту на прикладі англійської мови. Наукова новизна полягає у створенні авторської програми розпізнавання омонімів, складовими якої є проектування математичного визначення; розробка алгоритму; планування архітектури проекту; розробка системи та тестування програмного забезпечення.

Ключові слова: омоніми, семантичний контекст, алгоритм, архітектура проекту.

SHEVCHUK Yaroslav. WSD SOFTWARE FOR HOMONYM MEANING RECOGNITION

Optimization of the cognitive function of language in the architecture of the systems of processing speech phenomena in the process of working with text structures is one of the topical issues of modern computer linguistics, which determines the relevance of the research. The article presents the software for precisely selecting the meaning of a word from existing homonyms, taking into account the semantic context of the Russian and English language. Scientific novelty consists in creation of author's program of recognition of homonyms, the components of which is the design of mathematical definition; development of the algorithm; planning of project architecture; system development and software testing. Methodology of the research is based on the methods of mathematical modeling: methods of Michael Lesk, Dictionary and knowledge-based methods; supervised, semi-supervised and minimally supervised methods. The implementation of the developed program was carried out using the unsupervised method, which is based on the assumption that similar word values are contained in similar contexts, and therefore the meanings of words can be separated from the text by clustering the occurrences of words using a certain degree of context. The architecture of the project with the "unsupervised" vector model algorithm is based on the modular principle (REST interface, Interactor, Feature extractor, WordNet, Classifier, Knowledge base, Database), each of which contains a certain part of the functionality in order to minimize the dependencies between the modules. The modules are fully compatible with the design architecture and encapsulate the implementation details at the package level. To implement the classifier, an algorithm of cosine coefficients was used. The basic implementation of the system is performed in Java programming language. The project class diagram is generated using the CodeIris dependency analysis tool. The server was implemented using Rapidoid technology that runs on asynchronous buffer classes from the java.nio package. Testing has shown that the total execution time of the program's WSD system for setting the value of homonyms is about 400 milliseconds, and virtually everything goes to finding the base vectors.

Key words: homonyms, semantic context, algorithm, project architecture.

Комп'ютерна лінгвістика є відносно молодою науковою галуззю, яка динамічно розвивається і потребує удосконалення програмного забезпечення для обробки мовних явищ, зокрема програмних інструментів вибору точного значення слова. Оптимізація когнітивної функції мови в архітектурі систем обробки мовних явищ у процесі роботи з текстовими структурами є одним з актуальних питань сучасної комп'ютерної лінгвістики. Вилучення інформації з тексту за допомогою комп'ютерних алгоритмів може ускладнитися, якщо в реченні присутні слова-омоніми – різні за значенням, але однакові у написанні одиниці мови. Word Sense Disambiguation (WSD) як проблема комп'ютерної лінгвістики (від англ. *word* – слово; *sense* – смисл; *disambiguation* – усунення конфліктів, неоднозначностей) – означає сукупність операцій, скерованих на віднаходження механізмів вирішення питань лексичної багатозначності в процесі автоматичної обробки текстів; розробку програмного забезпечення лінгвістичної обробки мови для процесів пошукової оптимізації, при перекладі, підвищенні релевантності видачі адекватного значення слова тощо.

Метою дослідження є розробка програмного забезпечення способу точного вибору значення слова з наявних омонімів з урахуванням семантичного контексту на прикладі англійської мови. Досягнення мети передбачає розв'язання наступних завдань: проектування математичного визначення; розробка алгоритму; планування архітектури проекту; розробка системи; тестування програмного забезпечення.

Наукова новизна дослідження полягає у розробці інноваційного програмного забезпечення для встановлення значень омонімів. Архітектура проекту з алгоритмом векторної моделі класу "*unsupervised"* розроблена за модульним принципом (REST інтерфейс, Interactor, Feature extractor, WordNet, Classifier, Knowledge base, Database), кожен з

яких містить певну частину функціоналу з метою мінімізації залежності між модулями. Модулі достеменно відповідають спроектованої архітектурі і інкапсулюють деталі реалізації на рівні пакетів. Для реалізації класифікатора був використаний алгоритм косинусних коефіцієнтів. Базова реалізація системи виконана на мові програмування Java. Діаграма класів проекту згенерована за допомогою інструменту для аналізу залежностей CodeIris. Сервер був реалізований з використанням технології Rapidoid, що працює на асинхронних буферних класах вводу-виводу з пакета java.nio. Мовним матеріалом дослідження стали лексичні одиниці з оригіналів класичних творів англійської літератури.

Дослідження проводилося з опертям на методи математичного моделювання з урахуванням *методів Майкла Леска*, скерованого на вирішення завдання зіставлення слова і смислу в електронних словниках, виходячи з контексту, в яких вживалися слова та похідних від нього методів *Dictionary and knowledge-based methods,* що покладаються на словники, тезауруси та лексичні бази знань; *supervised, semi-supervised and minimally supervised methods*, наглядова, наполовину чи мінімально контрольована методика яких ґрунтується на контексті, в якому використовуються ті чи інші лексично багатозначні слова; та реалізацією розробленої програми за допомогою *unsupervised method*, який є найменш розробленим і вживаним у WSD програмуванні. В його основі лежить припущення, що подібні значення слів містяться у схожих контекстах, а відтак смисли слів можуть бути виокремлені з тексту шляхом кластерізації входжень слів з використанням певного ступеню контексту, що в науці отримала визначення індукція або дискримінація смислу контексту.

Програма може бути використана в інформаційних пошукових системах, автоматизованому текстовому редакторі, при класифікації і кластеризації текстів, вилученні термінів і ключових слів, контент-аналізі, машинному перекладі.

Електронний тезаурус WordNet, створений в 1985-х р. у Принстонському університеті для англійської мови, складається з іменників, дієслів, прикметників, прислівників та інших частин мови, згрупованих у набори синонімічних множин, або когнітивних синонімів (synsets), що взаємопов'язані за допомогою концептуально-семантичних та лексичних відносин. [1]. Цей ресурс відноситься до класу лексичних онтологій, який знаходиться у вільному доступі в Інтернеті, на основі якого було проведено безліч експериментів у галузі інформаційного пошуку. WordNet охоплює приблизно 155 000 різних лексем і словосполучень, організованих у 117 000 понять або сукупностей синонімів, а загальна кількість пар «лексема-значення» налічує 200 000. Мережа WordNet як корисний інструмент для обчислювальної лінгвістики та обробки природної мови є вільно доступною для завантаження. Он-лайн версія WordNet постійно допрацьовується і оновлюється. За її принципом створено бази даних інших мов світу, включаючи російську і українську [2].

Перевагами WordNet є розробка лексикону як бази груп слів синонімів, пов'язаних між собою залежністю узагальнення. Наприклад, слова *рух, хід, переміщення, пересування* можуть перебувати в одній синонімічній множинності, а *занепад, падіння, зниження* – в іншій, при цьому для них визначається спільна узагальнююча множинність значень, що включає в себе поняття *зміна, заміна*.

Структурним представленням такого роду залежностей є математична модель графа. Об'єкт граф G являє собою пару невпорядкованих множин (V, E), де V – множина вершин графа, а E множина залежностей або ребер в графі. Ребром графа назвемо впорядковану пару вершин (v_i, v_j), з чого виходить, що граф є орієнтованим. Вершиною в даному графі виступає синонімічна множинність значень, а ребро позначає залежність «узагальнення», тобто ребро (v_i, v_j) означає, що множина v_j включає в себе слова-узагальнення слів множини v_i. Для зручності обробки вершинам графу присвоєно цілочисельні ідентифікатори (на малюнку вище: 1, 2, 3). Таким чином, представлений вище граф описується як:

$$G = \{\{1, 2, 3\}, \{(2,1), (3,1)\}\}$$

Крім того, synset містить коротке визначення ("gloss") і, в більшості випадків, одне або декілька коротких речень, що ілюструють використання членів синонімічних множин. Форми слів з кількома різними значеннями представлені в якомога більшій кількості різних посилань. Таким чином, кожна пара, що має формоутворююче значення в WordNet, є унікальною.

Для комп'ютера визначення значення слова омоніму є проблематичним, навіть коли відомий контекст його використання. Для побудови коректної комп'ютерної системи, що вирішує цю проблему потрібно:

1. Дати однозначне визначення проблеми. Це робиться або шляхом складання специфікації системи, де перераховуються всі вимоги до неї, або шляхом математичного моделювання і формулювання проблеми. В даному випадку через специфіку проблеми був обраний другий підхід.
2. Підібрати алгоритми для нетривіальних компонентів системи і описати їх в достатніх деталях для реалізації на обраній мові програмування.
3. Розробити архітектуру системи реалізації алгоритму.
4. Реалізувати систему.

Математичне визначення. Математичне визначення як опосередковане практичне дослідження проблеми, є важливим етапом розробки математичної моделі як необхідної компоненти програмної системи. За допомогою математичних методів представимо бачення ідеального об'єкту на етапі його змістового моделювання:

1. Універсальна множина D усіх слів у мові.
2. WordNet граф, що надалі позначатиметься як WN.
3. Функція s(w), що повертає множину усіх синонімічних множин слова w.

$$s(w) = \{ v, v \in WN \mid w \in v \}$$

4. Множина O – є множиною слів омонімів. Той факт, що слово входить у більш ніж одну синонімічну множину, а відтак може приймати різні значення, будемо вважати показником омонімії

$$O = \{ w, w \in D \mid |s(w)| > 1 \}, O \subset D$$

5. Множина W – включає в себе усі слова, що не є омонімами:

$$W = D \setminus O, W \subset D$$

6. Функція Syn(w) повертає множини усіх синонімів w:

$$Syn(w) = \cup s(w) \setminus O$$

Множини, що повертаються функцією Syn(w) визначено як об'єднання синонімічних множин, до яких входить w, за виключенням слів омонімів, зокрема й самого w.

7. Заміни на Функція A(w) повертає множину усіх слів, які не є омонімами і входять до множин об'єднання усіх прямих узагальнень синонімічних множин, які включають в себе w:

$$A(w) = \cup\{ v_j, v_j \in \cup \{ (v_i, v_j) \in E \mid v_i \in s(w) \} \} \setminus O$$

8. Функція W'(w) повертає об'єднання множин усіх синонімів слова w і множин прямих узагальнень цих слів. Дану множину будемо вважати множиною слів-замінників слова w. Необхідність включати у цю множину прямі слова-замінники обумовлена тим, що слово може не мати синонімів для одного із своїх значень:

$$W'(w) = S(w) \cup A(w), \forall w \in O$$

9. Залежність E (word, meaning) і функція m(w), що повертає значення слова w:

$$m(w) = \prod_{meaning}(y_{word = w}(E)), \forall w \in W$$

Областю значень функції m(w) є множина W. Якщо перекласти вираження реляційної алгебри у матеріальний світ, то залежність E – це тлумачний словник, в якому надано визначення тільки тих слів, що мають одне, не залежне від контексту значення. А функція m(w) – процес пошуку значення слова w у словнику E.

10. Множина C слів, що утворюють контекст вживання слова. Простим прикладом контексту є речення, що включає в себе слово-омонім. Наприклад, для омоніма bank, використаного В. Шекспіром у реченні («Юлій Цезар»): *Tiber trembled underneath her **banks*** ми вважатимемо контекстом множину C = { Tiber, trembled, underneath, her }.

11. Функція P(w, C), що визначає імовірність вживання слова w у даному контексті C. Ця частина є ключовою і найскладнішою в реалізації. Від вибору алгоритму для обчислення імовірності будуть залежати такі ключові характеристики програми як коректність,

продуктивність і вартість реалізації. Імовірні підходи, а також їх переваги і недоліки описано в наступній секції.

12. Функція f(w, C) для находження найбільш імовірного слова-замінника для w в даному контексті C:

$$f(w, C) = \arg\max \{ P(w', C), w' \in W'(w) \}, w \in O$$

Насамкінець, процес визначення омоніму у певному контексті його використання, в рамках описаної математичної моделі, може бути виражений як:

$$m(w, C) = m(f(w, C)), w \in O$$

Тепер математична модель слугує однозначній специфікації задачі. Потрібно створити програмну систему, яка отримуючи на вхід слово і контекст, в якому воно вживається, буде повертати значення цього слова.

Представлена математична модель також надає уявлення про те, які операції опрацювання даних мають бути реалізовані у програмі.

Алгоритм вирахування імовірності. Вектор представляє собою упорядкований набір компонентів одного типу. У фізиці вектора використовуються для представлення величин, що мають величину і напрям, таких як швидкість, однак все це є лише згодою. Вектор може мати необмежену кількість компонентів, значення яких можуть нести будь-який смисл. Розширимо окреслену вище математичну модель, вводячи поняття вектора контексту.

Надамо кожному слову з множини D унікальний ідентифікатор i ∈ [1, |D|], тоді вектором контексту слова w є вектор c ∈ $N^{|D|}$, с |D| компонентами, що виражені натуральними числами. Компонент на позиції ***i*** є рівним одиниці, коли слово з ідентифікатором ***i*** може бути використаним в одному контексті зі словом w або нулю в противному випадку. Контекст, в якому слово буде вживане з найбільшою вірогідністю, будемо називати базовим вектором контексту слова w.

У лінійній алгебрі існує операція скалярного добутку векторів, що визначається як сума добутку компонентів двох векторів:

$$a * b = \sum_{i=1} a_i b_i$$

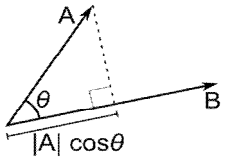
Результатом операції є число, а не новий вектор. З теореми косинусів виводиться, що скалярний добуток рівний добутку довжин векторів, помноженому на косинус кута між векторами:

$$a * b = |a| * |b| * \cos(\angle ab)$$

Наслідок цього часто використовується для обчислення косинуса кута між векторами:

$$\cos(\angle ab) = (a * b) / (|a| * |b|)$$

Згідно з визначенням контекстного вектору, його компоненти належать множині натуральних чисел, а відтак знаходяться у першому квадранті координатної площини, з чого виходить, що можливе значення кута між векторами знаходиться в інтервалі [0, 90], а областю визначення функції є проміжок [0, 1], що відповідає діапазону можливих значень імовірності події. Значення косинусу кута між векторами можна розглядати як міру схожості векторів, що не залежать від довжини векторів. Такий метод прийнято називати методом косинусного коефіцієнту схожості:



Тоді наша функція імовірності вживання слова w в контексті C може перетворити множину C в вектор контексту й обчислювати косинус кута між ним і базовим вектором контексту слова w. Базовий вектор має бути перед-обчислений для кожного слова, значення якого ми бажаємо визначити, на даний момент допустимо, що існує функція B(w), що повертає базовий вектор контексту свого аргументу.

Перетворення контекстної множини в вектор контексту виконується просто: якщо слово з ідентифікатором ***i*** входить у множину C, то компонент вектору на позиції ***i*** рівний одиниці, а у протилежному випадку – нулю. Якщо ця операція виконується функцією T(C), тоді функція імовірності вживання слова w в контексті C може бути записана як:

$$P(w, C) = \cos(B(w), T(C))$$

Одним із варіантів обчислення базового вектору контексту є обробка текстового масиву значних розмірів. Оскільки кількість слів, які часто вживаються одне з одним, неймовірно замала відносно усіх слів у мові – вектор контексту слова, що повністю відповідає математичному визначенню, буде дуже розрядженим, через що буде не доцільним реалізовувати таку структуру даних на практиці. Її зберігання буде займати дуже багато пам'яті, більшість компонентів не будуть вносити будь-яку інформацію і обчислення косинусного коефіцієнту буде потребувати більше процесорного часу. Найбільш практичним рішенням буде збереження вектора фіксованого розміру довжиною, наприклад в 50 елементів, куди увійде 50 найбільш часто уживаних слів.

Таким чином, у псевдокоді алгоритм обчислення базового вектору контексту може бути описано таким чином:

```
computeBaseContextVector(w, n):
  ocurrenceMap <- {}
  for sentence : text
    if sentence.contains(w)
      for word : sentence
        if word != w
          ocurrenceMap[word]++
  sortDescByValue(ocurrenceMap)
  return ocurrenceMap.keys().take(n)
```

де w – слово, для якого ми обчислюємо базовий контекстний вектор розміру n. Практично усі мови програмування представляють реалізацію асоціативних масивів, яка може бути використана як ocurrenceMap, а операцію sortDescByValue можна виконати за допомогою сортування або додавання елементів до черги пріоритетів.

Плюсами векторної моделі є простота розуміння, реалізації і розпаралеллювання обчислення. Володіючи достатньо великою базою даних, отриманою при обробці різноманітних текстів, модель цілком здатна давати правильні відповіді у більшості випадків. Алгоритм відноситься до класу "*unsupervised"*. Це означає, що текстові дані, що оновлюють базу знань алгоритму, не несуть ніякої мета-інформації, а алгоритм сам намагається виявити внутрішні взаємозв'язки, залежності в них.

Найбільшим мінусом моделі є те, що усі слова в ній вносять однаковий внесок до результатируючої імовірності. Наприклад, при розгляді речення з роману Джейн Остін «Гордість та упередження»:

Every disposition of the ground was good; and she looked on the whole scene, the river, the trees scattered on its banks and the winding of the valley, as far as she could trace it, with delight.

очевидно, що слово river має бути сильним маркером того, що омонім banks вживаний у значенні берегів ріки, а не банків. У векторній моделі внесок слова river до результату дорівнює внеску будь-якого іншого слова у реченні. Через цю властивість можливі приклади, на яких алгоритм буде обробляти текстовий масив некоректно.

Альтернативний алгоритм, що не має проблеми відсутності ваги слів, може бути побудований на базі байєсівських імовірнісних мереж. Байєсівські мережі є скерованим зваженим ациклічним графом, що представляє загальний розподіл імовірності на множини випадкових величин. Дана модель робить можливим статистичні висновки на основі залежностей вершин у графі. Кожна вершина графа у байєсівських мережах представляє собою випадкову величину, а спрямована грань (v_i, v_j) – вплив вершини v_i на вершину v_j.

Основоположне припущення про умовну незалежність випадкових величин у байєсівських мережах описується так:

$$X : (X \perp U \setminus De(X) \mid Pa(X))$$

де U – універсальна множина випадкових величин, De(X) – множина усіх потомків X, а Pa(X) – множина усіх величин, що впливають на X. Дане вираження означає, що випадкова величина X є умовно незалежною від своїх потомків, коли відомі величини, що впливають на X. Тоді загальний вірогідний розподіл кожної випадкової величини X у байєсівських

мережах може бути представлений як добуток умовних ймовірностей усіх величин окремо, враховуючи імовірність впливаючих на них величин:

$$P(x_1, \ldots, x_n) = \prod_{i=1} P(x_i \mid Pa(x_i))$$

При використанні байєсівських мереж в якості класифікатора потрібно обчислити імовірність даного слова-замінника, приймаючи до уваги значення, що залишилися у мережі:

$$P(w' \mid C) = P(U)/P(C) \propto P(U) = \prod_{u \in U} P(u \mid Pa(u))$$

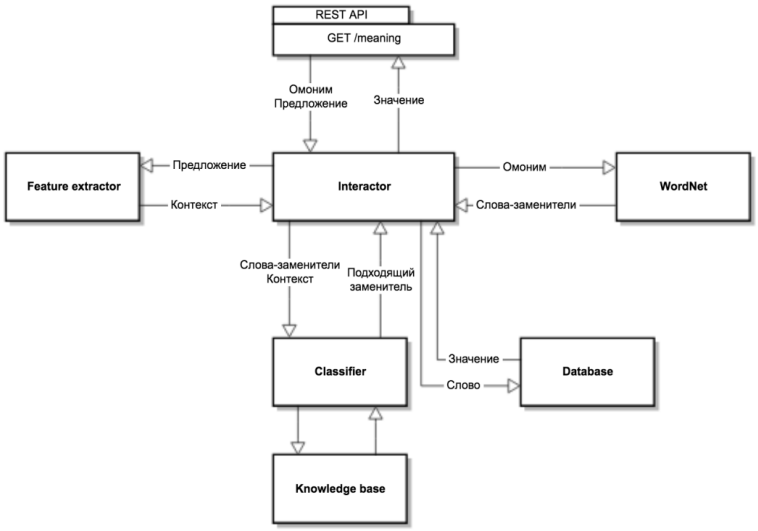
Недоліком класифікатора на основі байєсівських мереж є складність його реалізації і тренування. Перед тим, як показувати результати, класифікатор має бути натренованим на великому наборі маркованих даних, що є прикладом алгоритму "*supervised by teacher",* тобто потребуючого навчання "з вчителем". У випадку розгляду проблеми мета-інформацією, що поступає на вхід алгоритму при оновленні бази знань, є правильне значення слова-омоніма у даному контексті. Ручне маркування даних є тривалим, дорогим і складним процесом.

Архітектура і базова реалізація системи. Під архітектурою проекту розуміється опис його частин та їх взаємодії. Планування архітектури є одним із ключових етапів циклу розробки. Помилки, що виявлені на стадії проектування виправити набагато простіше, ніж ті, які виявляються у процесі розробки, що вимагає відповідального ставлення до цього етапу.

Для опису архітектури, система розбивається на модулі, кожний з яких містить у собі певну частину функціоналу. Далі визначаються інтерфейси модулів, через які вони будуть взаємодіяти між собою і кінцевим користувачем. Деталі реалізації кожного модуля на етапі планування архітектури не розглядаються.

В архітектуру програмної системи має бути закладена розширюваність з метою створення можливості її доопрацювання у випадку появи нових вимог. Впровадження найбільш імовірного функціоналу має вимагати найменшу кількість зусиль – наприклад, якщо первісна система отримує свої аргументи з командного рядка, з ростом затребуваності проекту може знадобитися винести його у хмарний сервіс. Це буде просто зробити, якщо модуль вводу даних буде ізольований від решти компонентів програми. Один із принципів підходу до написання програм SOLID фокусується на такій вимозі: елементи мають бути відкритими до розширення і закриті до модифікації, – тобто зміни у програмі повинні робитися за допомогою додавання нових компонентів, а не переписування старих. Плюсами модульної архітектури також є тестуємість і можливість повторного використання компонентів (Див. Рис. 1).

Рис. 1. Архітектура програмної WSD системи для встановлення значень омонімів.



На Рис. 1 представлена архітектура програмної WSD системи для встановлення значень омонімів, яка побудована таким чином, щоб мінімізувати залежності модулів один від одного. Система складається з таких модулів:

1. **REST інтерфейс**, представлений як приклад модуля введення даних. Система отримує запити на класифікацію за протоколом обміну даними HTTP. Запит включає в себе

слово-омонім і речення, в якому дане слово використовується. Після вилучення цих даних з тіла запиту, вони передаються в модуль Interactor:

```
public class Server {

    public static void main(String[] args) {
        On.get("/meaning")
                .plain((ReqHandler) req -> {
                    Resp response = req.response();
                    String word = req.param( name: "word");
                    Set<String> context = req.attr( name: "context");
                    if (word == null || context == null) {
                        response.code(400);
                    } else {
                        String meaning = Interactor.getDefault().inferMeaning(word, context);
                        if (meaning == null) {
                            response.code(500);
                        } else {
                            response.code(200);
                            response.body(meaning.getBytes(Charsets.UTF_8));
                        }
                    }
                    return response;
                });
    }
}
```

2. **Interactor**. Даний модуль слугує сполучною ланкою між модулями. На вхід він отримує омонім і речення, в якому омонім використовується. Далі Interactor за допомогою інших модулів трансформує вхідні дані та у разі успіху повертає значення омоніму.

3. **Feature extractor**. Модуль, що використовується модулем Interactor для трансформації речення в об'єкт контексту. В процесі з речення можуть бути видалені розділові знаки або слова, що не несуть важливої інформації, такі як займенники, а само речення може бути трансформовано у список слів.

4. **WordNet**. Модуль, що використовується модулем Interactor для трансформації слова-омоніму в його слова-замінники, які не є омонімами. WN графа, визначеного у математичній моделі проблеми:

```
public class WordNet {

    private Digraph digraph;
    private Map<String, List<Integer>> nounToSynset = new HashMap<>();
    private Map<Integer, String> idToNouns = new HashMap<>();

    public WordNet(String synsets, String hypernyms) {
        ensureNotNull(synsets, hypernyms);
        processVertices(readAll(synsets));
        processEdges(readAll(hypernyms));
        ensureRooted();
        ensureNoCycles();
    }

    private void processVertices(String synsets) {
        int maxVertex = 0;
        for (String synset : synsets.split( regex: "\n")) {
            String[] tokens = synset.split( regex: ",");
            int id = Integer.parseInt(tokens[0]);
            String[] nouns = tokens[1].split( regex: " ");
            idToNouns.put(id, tokens[1]);
            for (String noun : nouns) {
                noun = noun.trim();
                nounToSynset.computeIfAbsent(noun, n -> new ArrayList<>()).add(id);
            }
            maxVertex = Math.max(maxVertex, id);
        }
        digraph = new Digraph( V: maxVertex + 1);
    }

    private void processEdges(String hypernims) {
        for (String hypernim : hypernims.split( regex: "\n")) {
            if (hypernim.isEmpty()) continue;
            String[] tokens = hypernim.split( regex: ",");
            int hyponimId = Integer.parseInt(tokens[0]);
            for (int i = 1; i < tokens.length; i++) {
                int hypernimId = Integer.parseInt(tokens[i]);
                digraph.addEdge(hyponimId, hypernimId);
            }
        }
    }
```

5. **Classifier**. Класифікатор, який отримує на вхід контекст визначеного формату і список слів, що потребують перевірки на вживаність у даному контексті. Закінчивши обробку, класифікатор повертає найбільш прийнятне у даному контексті слово.

```
private String classify(@NonNull Set<String> labels, @NonNull UseContext useContext) {
    String result = null;
    double maxCos = Double.MIN_VALUE;
    for (String label : labels) {
        UseContext baseContext = knowledgeBase.getBaseContextFor(label);
        Pair<int[], int[]> vectors = buildVectors(useContext, baseContext);
        final double vec1Length = Math.sqrt(dot(vectors.first, vectors.first));
        final double vec2Length = Math.sqrt(dot(vectors.second, vectors.second));
        final double cos = dot(vectors.first, vectors.second) / (vec1Length * vec2Length);
        if (cos > maxCos) {
            result = label;
            maxCos = cos;
        }
    }
    return result;
}

private Pair<int[], int[]> buildVectors(UseContext baseContext, UseContext useContext) {
    Set<String> dictionary = union(baseContext, useContext);
    final int vectorDimension = dictionary.size();
    final int[] baseVec = new int[vectorDimension];
    final int[] useVec = new int[vectorDimension];
    int index = 0;
    for (String word : dictionary) {
        baseVec[index] = baseContext.contains(word) ? 1 : 0;
        useVec[index] = useContext.contains(word) ? 1 : 0;
        index++;
    }
    return new Pair<>(baseVec, useVec);
}

private double dot(int[] vec1, int[] vec2) {
    int result = 0;
    for (int i = 0; i < vec1.length; i++) {
        result += vec1[i] * vec2[i];
    }
    return result;
}
```

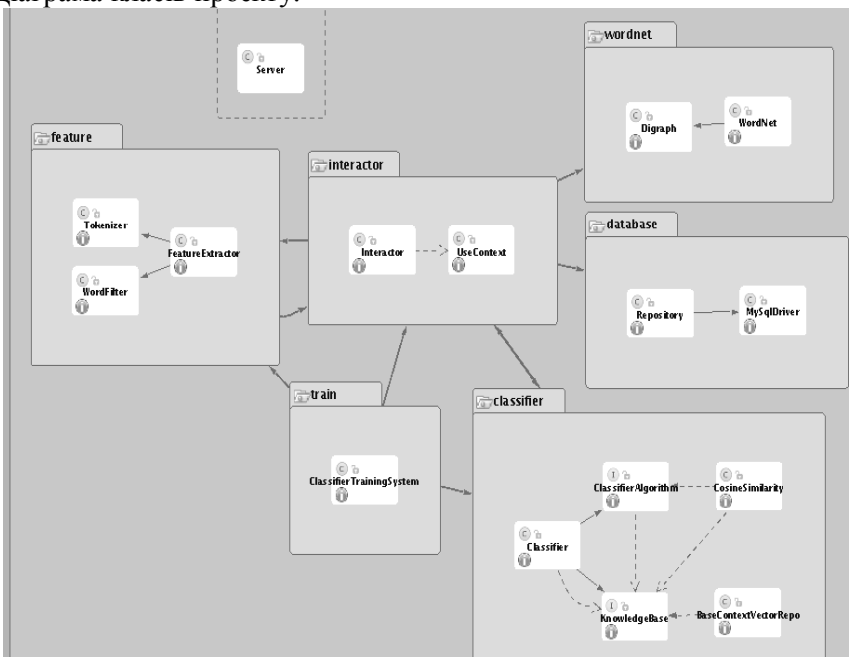
6. **Knowledge base**. Цей модуль є набором даних, на основі яких класифікатор робить свій вибір. Внутрішній устрій буде прямо залежати від алгоритму, що використовується для класифікації. Наприклад, при використанні методу косинусного коефіцієнту схожості, Knowledge base буде містити базу даних базових векторів контексту для різних слів.

7. **Database**. Проста база даних, яка приймає слово як аргумент і повертає його словникове визначення. Пряма реалізація залежності E (word, meaning), визначеної у математичній моделі проблеми.

Базова реалізація системи, модулі якої достеменно відповідають спроектованій архітектурі та інкапсулюють деталі реалізації на рівні пакетів, виконана на мові програмування Java. Алгоритм косинусних коефіцієнтів був використаний для реалізації класифікатора. Результати його роботи наведено в секції з його описом.

Діаграма класів проекту згенерована за допомогою інструменту для аналізу залежностей CodeIris. В системі присутній модуль train, який був добавлений для ініціалізації базових векторів контексту. В модулі Classifier визначено інтерфейси ClassifierAlgorithm и KnowledgeBase для простішої заміни реалізації (Див. Рис. 2):

Рис. 2. Діаграма класів проекту.



Сервер був реалізований з використанням технології Rapidoid, що працює на асинхронних буферних класах вводу-виводу з пакета java.nio.

При тестуванні програма обробляла два твори класичної англійської літератури: «Аліса в країні чудес» Л. Керола і «Великий Гетсбі» Ф. С. Фіцджералда». Під час обробки в Knowledge base зберігалися контекстні вектори для трьох синонімів омоніма *ball*, які мали різні значення: *dance, sphere, wad*. Кожний вектор міг мати не більш 50 компонентів.

Нижче представлені отримані вектори:

Wad: [coat, watching, tighten, under, shoulder, back, saw, muscle, tom]

Sphere: [wish, sensible, up, consider, quit, quitting, marrying, nephew, myself, brought, been, own, good]

Dance: [join, could, wont, very, people, made, himself, from, said, did, do, come, talked, sure, so, about, surprised, evening, before, must, tell, next, sort, every, amusement, meryton, can, ball, whiting, room, till, sent, less, we, couples, feeling, particular, depended, nearer, curious, england, said, alice, young, quite, only, whether]

після чого програма визначала, яке значення слово ball має в контексті даного, випадково взятого речення із твору Ф. С. Фіцджералда «Великий Гетсбі»:

Bingley had soon made himself acquainted with all the principal people in the room; he was lively and unreserved, danced every dance, was angry that the ***ball*** *closed so early, and talked of giving one himself at Netherfield*

Результат програми був таким:

dance: 0.1499

sphere: 0.0

wad: 0.0

Загальний час виконання становив близько 400 мілісекунд і практично весь він пішов на находження базових векторів.

Висновки. Для реалізації поставленої проблеми – оптимізації існуючих програмних систем для встановлення значень омонімів, було розроблено два алгоритми:

- алгоритм векторної моделі класу "*unsupervised"* з функціями самостійного виявлення внутрішніх взаємозв'язків і залежностей аналізованих омонімів;
- алгоритм на базі байєсівських імовірнісних мереж, що є прикладом алгоритму "*supervised by teacher"*.

В роботі продемонстровані переваги архітектури алгоритму векторної моделі (Див. Рис. 1), серед яких: легко адаптувати програму для використання в іншій сфері, для чого достатньо додати новий модуль вводу даних, який буде працювати з модулем Interactor; можними є зміна сховища даних, для чого достатньо замінити модуль бази даних; при експлуатації може бути використана будь-яка технологія, що дозволяє зберігати пари *ключ-значення*; легко замінити алгоритм: деталі реалізації класифікатора не мають зовнішніх залежностей в системі; модуль Feature extractor може бути легко використаним при наповненні бази знань класифікатора, оскільки він не залежить від будь-яких модулей на кшталт бази даних або REST інтерфейсу; кожний модуль за виключенням модуля Interactor може бути легко протестований окремо, оскільки взаємодіє з системою виключно через свій публічний інтерфейс.

Базова реалізація системи виконана на мові програмування Java. Її модулі точно відповідають спроектованій архітектурі, інкапсулюють деталі реалізації на рівні пакетів. Діаграма класів проекту згенерована за допомогою інструменту для аналізу залежностей CodeIris (Див. Рис. 2). В системі присутній модуль train, який був добавлений для ініціалізації базових векторів контексту. В модулі Classifier визначено інтерфейси ClassifierAlgorithm и KnowledgeBase для простішої заміни реалізації. Сервер був реалізований з використанням технології Rapidoid, що працює на асинхронних буферних класах вводу-виводу з пакета java.nio.

Клас Tokenizer в модулі Feature extractor реалізує алгоритми розбиття тексту на речення і речень на слова, зокрема видалення розділових знаків. WordFilter слугує для видалення з тексту слів, які не несуть смислового навантаження, таких як артиклі "a", "the" або займенники "he", "she" й ін.

Клас UseContext в модулі Interactor представляє собою уніфіковане дя всіх модулів проекту уявлення контексту використання омоніму. Оскільки усі модулі взаємодіють з модулем Interactor, це не вносить додаткових залежностей.

Для збереження словника використовується система керування базою даних MySql. Ця деталь реалізації прихована від решти проекту в середині класу Repository. Для реалізації класу WordNet була додана структура даних, яка не представлена стандартною бібліотекою мови Java – орієнтований граф, клас класс Digraph у проекті. Окрім орієнтованого графа, клас WordNet містить дві хеш-таблиці, які надають можливість швидко отримати список цілочисельних ідентифікаторів вершин графа за словом, а також отримати список слів за ідентифікатором вершини графа. WordNet ініціалізується з текстових файлів, що представлені у відкритому доступі Принстонського університету. Тестування засвідчило, що загальний час виконання програмної WSD системи для встановлення значення омонімів становить близько 400 мілісекунд, і практично весь йде на знаходження базових векторів.

БІБЛІОГРАФІЯ

1. WordNet. Electronic resource. Access mode: https://wordnet.princeton.edu/
2. WordNet (Ukraine). Electronic resource. Access mode: https://dl.acm.org/citation.cfm?id=2958315

REFERENCES

3. WordNet. Electronic resource. Access mode: https://wordnet.princeton.edu/
4. WordNet (Ukraine). Electronic resource. Access mode: https://dl.acm.org/citation.cfm?id=2958315

ВІДОМОСТІ ПРО АВТОРА

Ярослав Шевчук – бакалавр програмної інженерії, магістрант-філолог Ізмаїльського державного гуманітарного університету.

Наукові інтереси: прикладна лінгвістика.

INFORMATION ABOUT THE AUTHOR

Yaroslav Shevchuk – bachelor of Software Engineering, graduate philologist at the Izmail State University of Humanitees.

Scientific interests: applied linguistics.