УДК 004.912:811.161.2

Глибовець А. М., Точицький В. В.

АЛГОРИТМ ТОКЕНІЗАЦІЇ ТА СТЕМІНГУ ДЛЯ ТЕКСТІВ УКРАЇНСЬКОЮ МОВОЮ

У статті досліджено проблематику застосування алгоритмів токенізації (лематизації) та стемінгу для обробки документів українською мовою як початкових етапів для аналізу текстових документів. Аналіз текстів – доволі поширена задача, що виникає в різноманітних завданнях, і на першому етапі початкової обробки завжди використовується стемінг або лематизація. Для більшості мов цю проблему давно розв'язано й існує багато різноманітних реалізацій. Але для української мови немає реалізації цих алгоритмів. Для написання лематизатора потрібно мати багато додаткових інструментів, оскільки він спирається на лексикографічний аналіз, тоді як реалізація стемера не потребує додаткових бібліотек. У цій статті ми зосередилися на створенні стемера для української мови. Було проаналізовано наявні алгоритми і запропоновано адаптацію та реалізацію для української мови. Результатом є готовий модуль мовою PHP, що перебуває у вільному доступі та може використовуватися в проектах, пов'язаних з аналізом текстів українською мовою.

Ключові слова: text mining, токенізація, лематизація, стемінг, семантичний аналіз текстів.

Вступ

Розглянемо задачу адаптації алгоритмів токенізації та стемінгу для документів, написаних українською мовою, і знаходження оптимального підходу до початкового етапу автоматизованого аналізу текстів.

Інтелектуальний аналіз тексту (ІАТ, англ. *text mining*) – напрям інтелектуального аналізу даних (англ. *Data Mining*) та штучного інтелекту, метою якого є отримання інформації з колекцій текстових документів, ґрунтуючись на застосуванні ефективних методів машинного навчання та обробки природної мови. Щороку прогрес у цій сфері зростає швидкими темпами. Сучасне програмне забезпечення все більше потребує готових рішень для обробки текстів та вдосконалення своїх систем. Найбільша складність виникає під час роботи з природною мовою або з текстами без чіткої структури контенту [1].

Ключовими завданнями ІАТ є категоризація текстів, пошук інформації, обробка змін у колекціях текстів, а також розроблення засобів представлення інформації для користувача. Для того щоб було зручно працювати з текстом, потрібно перевести його в представлення мови програмування та обробити для подальшої роботи [2].

Для того щоб використати готові реалізації подібних алгоритмів, потрібно проаналізувати велику кількість систем та бібліотек. Більшість із них можуть працювати з найпопулярнішими мовами, зокрема російською.

Для української мови все набагато складніше: є поодинокі незавершені роботи (наприклад, «Вільний алгоритм стемінгу для української мови» http://www.senyk.poltava.ua/projects/projects.html) або комплексні системи, проте без потрібного функціоналу (наприклад, «LanguageTool» https://www.languagetool.org/development/). Тому виникає необхідність реалізувати власні бібліотеки для обробки та аналізу текстів.

Базові поняття

Для лексичного аналізу існує процес розбиття тексту на елементарні одиниці – токени. Такий процес називається *токенізацією* і є зазвичай початковим етапом обробки текстів, адже дає змогу працювати зі словом як з окремою сутністю, при цьому знаючи його контекст [3].

Зазвичай лексичний аналіз відбувається на рівні слів. Проте іноді буває важко визначити, що мається на увазі під «словом». Часто виникають різні задачі, наприклад: варто чи не варто включати знаки пунктуації або пробіли в результуючий список лексем; усі суміжні лексеми є частиною одного токена, наприклад, число з пробілами (123 456 789) або крапками (123.456).

Для прикладу візьмемо такий уривок тексту:

LiveArt (грудень 2014 – листопад 2015)

Посада: Front-end розробник

Опис проекту: веб-дизайнер для створення макетів різних товарів на замовлення на друк

Технології:

- TypeScript
- Knockout.js
- PHP5
- JavaScript
- Bootstrap3
- JavaScript
- jQuery

Обов'язки:

- постійна підтримка та рефакторинг
- розробка під клієнта

Для токенізації тексту можна використати кілька різних підходів:

1. Розбиття на основі пробілів.

Регулярний вираз: '/[\pZ\pC]+/u'

Результат:

LiveArt,грудень,2014,листопад,2015,Посада,Front-end,розробник,Опис,проекту,веб-дизайнер,створення,макетів,різних,товарів,замовлення,друк,Технології,TypeScript,Knockout.js,PHP5,JavaScript,Bootstrap3,JavaScript,jQuery,Обов'язки,постійна,підтримка,рефакторинг,розробка,клієнта

2. Розбиття на основі пробілів та розділових знаків.

Регулярний вираз:
'/([\pZ\pC]*)([^\pP\pZ\pC]+|.) ([\pZ\pC]*)/xu'

Результат:

LiveArt,грудень,2014,листопад,2015,Посада,Front,end,розробник,Опис,проекту,веб,дизайнер,створення,макетів,різних,товарів,замовлення,друк,Технології,TypeScript,Knockout,js,PHP5,JavaScript,Bootstrap3,JavaScript,jQuery,Обов'язки,постійна,підтримка,рефакторинг,розробка,клієнта

3. Розбиття на основі набору регулярних виразів, які будуть застосовувати до вхідного тексту ітеративно. Такий підхід є частково комбінацією двох попередніх.

Тестування цих способів для різних текстових файлів виявило, що найшвидше та найоптимальніше працюватиме перший варіант, який враховує лише пробіли.

Проаналізувавши результати токенізації за розділовими знаками, помітили, що деякі ключові слова розбиті на окремі частини і далі їх можна втратити при семантичному аналізі (наприклад: Knockout,js). Тому для вдосконалення роботи варто зробити циклічний обхід по отриманих токенах для видалення непотрібних символів (нормалізація) та стоп-слів або з'єднання сусідніх токенів за певними правилами. Під нормалізацією мається на увазі обробка отриманих токенів і видалення їх із результуючого масиву, якщо вони не є лемами, що несуть семантичний зміст. Приблизний список стоп-слів можна знайти за посиланням https://goo.gl/7vmeHl.

Стемінг (англ. *stemming*) – це процес скорочення слова до основи шляхом відкидання допоміжних частин, як-от закінчення чи суфікс. Результати стемінгу іноді дуже схожі на визначення кореня слова, але його алгоритми базуються на інших принципах. Тому слово після обробки алгоритмом стемінгу (стематизації) може відрізнятися від морфологічного кореня слова. Стемінг застосовується в лінгвістичній морфології та в інформаційному пошуку. Багато пошукових систем використовують стемінг для встановлення синонімічних зв'язків, якщо в них збігаються форми після стематизації. Цей процес називають злиттям.

Алгоритм стемінгу Мартіна Портера набув значного поширення та став де-факто стандартним алгоритмом стемінгу для англійської мови. Оригінальна версія стемера була призначена для англійської мови і була реалізована на мові BCPL. Згодом Портер створив проект «Snowball» і, використовуючи основну ідею алгоритму, написав стемер для поширених індоєвропейських мов, зокрема для російської. Для адаптації алгоритму Мартіна Портера за основу варто взяти псевдокод для російської мови [4], адже мови схожі між собою. Що ми й зробили.

Адаптація алгоритму стемінгу для української мови

Алгоритм не використовує баз основ слів, а лише, застосовуючи послідовно ряд правил, відсікає закінчення і суфікси, ґрунтуючись на особливостях мови, у зв'язку з чим працює швидко, але не завжди безпомилково. Алгоритм працює з окремими словами. Це означає, що контекст, у якому вжито слово, – невідомий.

Проаналізувавши існуючі рішення для слов'янських мов і взявши за основу алгоритм Портера, було запропоновано використовувати такий адаптований алгоритм стемінгу для української мови:

1. Визначимо класи закінчень та суфіксів, які будемо перевіряти у слові за допомогою регулярних виразів:

a. Дієприслівник

PERFECTIVEGROUND = /((ив|ивши|ившись))$/

b. Інфінітив

INFINITIVE = /(ти|учи|ячи|вши|ши|ати|яти|ючи|)$/

c. Рефлексивне дієслово

REFLEXIVE = /(с[яьи])$/

d. Прикметник

ADJECTIVE = /(ими|ій|ий|а|е|ова|ове|ів|є|їй|єє|еє|я|ім|ем|им|ім|их|іх|ою|йми|іми|у|ю|ого|ому|ої)$/

e. Дієприкметник

PARTICIPLE = /(ий|ого|ому|им|ім|а|ій|у|ою|ій|і|их|йми|их)$/

f. Дієслово

VERB = /(сь|ся|ив|ать|ять|у|ю|ав|али|учи|ячи|вши|ши|е|ме|ати|яти|є)$/

g. Іменник

NOUN = /(а|ев|ов|е|ями|ами|еи|и|ей|ой|ий|й|иям|ям|ием|ем|ам|ом|о|у|ах|иях|ях|ы|ь|ию|ью|ю|ия|ья|я|і|ові|ї|ею|єю|ою|є|еві|ем|єм|ів|їв|\'ю)$/

h. RVRE – це частина після першого голосного, або кінець слова, яка не містить голосних

RVRE = /^(.*?[аеиоуюяіїє])(.*)$/

i. Словотвірні частини

DERIVATIONAL = /[^аеиоуюяіїє][аеиоуюяіїє]+[^аеиоуюяіїє]+[аеиоуюяіїє].*сть?$/

2. Далі виконати:

- Якщо слово має ознаки інфінітива – вийти з алгоритму.
- Знайти ознаки дієприслівника та видалити зі слова, перейти до наступного кроку. В іншому випадку знайти ознаки рефлексивного дієслова та видалити їх. Потім видалити ознаки прикметника, дієслова або іменника та перейти до наступного кроку.
- Якщо слово закінчується на ***і***, видалити його.
- Пошук словотвірних частин. Видалити, якщо такі є.
- Видалити подвоєння та м'який знак.

Якщо застосувати цей алгоритм до наведеного вище зразка тексту, отримаємо такі токени: liveart, груден, 2014, листопад, 2015, посад, front-end, розробник, опис, проект, веб-дизайнер, створен, макет, різн, товар, замовлен, друк, технології, typescript, knockout.js, php5, javascript, bootstrap3, javascript, jquery, обов'язк, постійн, підтримк, рефакторинг, розробк, клієнт.

Якщо порівняти ідеальний варіант основи слова та результат написаного алгоритму стемінгу для певної вибірки українських слів (див. таблицю), то помітно, що стемінг не завжди коректно враховує суфікс, наприклад: «ортогональний» стало «ортогональн», хоча мало би бути «ортогонал». Також помилку можна побачити в прислівниках, наприклад: «восени» стало «восен», хоча мало би бути «восени».

Таблиця. **Порівняння між ідеальним варіантом основи слова та результатом написаного алгоритму стемінгу**

Слово	Основа	Результат стемінгу
ти	ти	ти
весна	весн	весн
міський	міськ	міськ
підводний	підводн	підводн
швидкий	швидк	швидк
бігати	бігати	бігати
безпритульними	безпритул	безпритульн
ортогональний	ортогонал	ортогональн
повільне	повіл	повільн
цивільним	цивіл	цивільн
дивними	дивн	дивн
критикувати	критикувати	критикувати
блиск	блиск	блиск
восени	восени	восен
бюро	бюро	бюро
перемагати	перемагати	перемагати
танцюючи	танцюючи	танцюючи
швидко	швидко	швидко
тут	тут	тут
авіадиспетчер	авіадиспетчер	авіадиспетчер

Реалізація алгоритму

```
/**
* @paramstring $word
* @returnstring
*/
publicstaticfunctionstemWord(string$word): string
{
$stem= mb_strtolower($word);
do{
//checkifinfinitive
$m = preg_replace(self::$INFINITIVE, '', $word);
if(strcmp($m, $word) !== 0) {
$stem= $word;
continue;
    }
//init
preg_match_all(self::$RVRE, $stem, $p);
if(!$p) {
break;
    }
if(empty($p[2]) || empty($p[1][0])) {
break;
    }
$start= $p[1][0];
$RV = $p[2][0];
//STEP 1
```

```
$m = preg_replace(self::$PERFECTIVEGROUND, '', $RV);
if(strcmp($m, $RV) === 0) {
$RV = preg_replace(self::$REFLEXIVE, '', $RV);
$m = preg_replace(self::$ADJECTIVE, '', $RV);
if(strcmp($m, $RV) === 0) {
$RV = preg_replace(self::$PARTICIPLE, '', $RV);
        } else{
$RV = $m;
$m = preg_replace(self::$VERB, '', $RV);
if(strcmp($m, $RV) === 0) {
$RV = preg_replace(self::$NOUN, '', $RV);
            } else{
$RV = $m;
            }
          }
      } else{
$RV = $m;
      }
//STEP 2
$RV = preg_replace('/i$/u', '', $RV);
//STEP 3
if(preg_match(self::$DERIVATIONAL, $RV)) {
$RV = preg_replace('/ість?$/u', '', $RV);
      }
//STEP 4
$m = preg_replace('/ь?$/u', '', $RV);
if(strcmp($m, $RV) === 0) {
$RV = preg_replace('/ейше?/u', '', $RV);
$RV = preg_replace('/нн$/u', 'н', $RV);
      } else{
$RV = $m;
      }
$stem= $start. $RV;
  } while(false);
return$stem;
}
```

Висновки

Проаналізувавши класичні підходи до аналізу текстів у сфері text mining, ми адаптували алгоритм Портера та реалізували вільно доступний модуль мовою PHP для токенізації та стемінгу української мови. Практичне тестування показало, що для токенізації добре спрацьовує просте розбиття по пробілах із подальшою нормалізацією токенів. Для стемінгу ми використали підхід, який відсікає закінчення та суфікси, максимально наближуючи слово до його кореня. Кращого результату можна досягти через лематизацію, але таке рішення потребує окремого вивчення мови та великих ресурсів.

Отримана початкова обробка тексту в цій статті дає змогу працювати з токенами для отримання семантичних зв'язків у текстах, написаних українською мовою. Стемер розроблено у вигляді модуля, що може бути швидко інтегрований до проекту та доступний для використання за посиланням: https://packagist.org/packages/tochytskyi/ukrstemmer.

Список літератури

1. Глибовець М. М. Інтелектуальні мережі : навч. посіб. / М. М. Глибовець, А. М. Глибовець, М. В. Поляков. – Дніпропетровськ : Нова ідеологія, 2014. – 464 с.
2. Пошук інформації / А. В. Анісімов, А. М. Глибовець, М. М. Глибовець, А. С. Шабінський. – Київ : НаУКМА, 2015. – 284 с.
3. Manning C. D. An Introduction to Information Retrieval / Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. – Cambridge University Press, 2009. – P. 22–36.
4. Snowball Stemmer [Електронний ресурс]. – Режим доступу: http://snowball.tartarus.org/algorithms/russian/stemmer.html.

A. Glybovets, V. Tochytskyi

TOKENIZATION AND STEMMING ALGORITHMS FOR THE UKRAINIAN LANGUAGE

The article focuses on the problem of adaptation of tokenization and stemming algorithms for documents written in the Ukrainian language and finding an optimal approach to the initial stage of automated text analysis.

Intelligent Text Analysis (IAT) is the direction of Data Mining and Artificial Intelligence, whose purpose is to retrieve information from collections of text documents based on the application of effective methods of machine learning and the processing of natural language. Every year, progress in this area is growing at a rapid pace. Modern software increasingly needs ready-made solutions for word processing and enhancement of their systems. The greatest difficulty is encountered when working with natural language or with texts without a clear content structure.

The key tasks of IAT: categorization of texts, information search, processing of changes in text collections, and development of means of representation of information for the user. In order to work comfortably with the text, one needs to translate it into the presentation of the programming language and process it for further work.

For the Ukrainian language, there are some unfinished works (for example, "Free algorithm for stigma for the Ukrainian language" http://www.senyk.poltava.ua/projects/projects.html) or complex systems, but without the necessary functional (for example, "LanguageTool" https://www.languagetool.org/development/). Therefore, there is a need to implement their own libraries for processing and analyzing texts.

After analyzing the classic approaches to the text mining text analysis, we adapted Porter's algorithm and implemented a freely available PHP language module for tokenization and emulation of the Ukrainian language. Practical testing has shown that for tokenization, a simple breakdown into spaces with the subsequent normalization of tokens works well. For stemming, we used an approach that cuts the end and suffix, bringing words to its root as closely as possible. A better result can be achieved through lemmatization, but such a solution requires a separate study of the language and large resources.

The resulting initial processing of the text in this article allows one to work with tokens to obtain semantic links in texts written in Ukrainian. The stemmer designed as a module that can be quickly integrated into the project and available for use at the link https://packagist.org/packages/tochytskyi/ukrstemmer.

Keywords: text mining, tokenization, lematization, stemming, semantic text analysis.

Матеріал надійшов 12.09.2017

УДК 519.85,519.172

Стецюк П. І., Ляшко В. І., Бардадим Т. О.

ВЛАСТИВОСТІ КВАДРАТИЧНОЇ ЗАДАЧІ ПРО МАКСИМАЛЬНИЙ *k*-ПЛЕКС У НЕОРІЄНТОВАНОМУ ГРАФІ

У статті розглянуто властивості верхніх оцінок для квадратичної задачі про максимальний k-плекс у неорієнтованому графі. Проаналізовано зв'язок квадратичної задачі для 1-плекса з відомим формулюванням квадратичної задачі для знаходження максимальної кліки графа. Наведено лагранжеві двоїсті оцінки для найпростіших квадратичних задач та показано, що їх можна покращити при додаванні функціонально надлишкових обмежень.

Ключові слова: максимальний *k*-плекс, максимальна кліка, квадратична задача, лагранжева двоїста оцінка, функціонально надлишкове обмеження.

Вступ

Поняття *k*-плекса для неорієнтованого графа введено в [6] (*k* – деяке натуральне число). Якщо *k*=1, то *k*-плекс збігається з клікою (повним підграфом) графа. При $k > 1$ *k*-плекс є ослабленням поняття кліки графа і вимоги на включення вершини в *k*-плекс є слабшими, ніж вимоги на включення вершини в кліку. Ці поняття широко використовуються в соціології для виявлення та дослідження окремих підгруп населення, при кластеризації даних, для оптимізації інформаційних потоків у мережах тощо (див., напр., [2–5]). Слід зазначити, що зазвичай оптимізаційні задачі пошуку максимальних клік та *k*-плексів є NP-складними.

У статті буде проаналізовано зв'язок наведеного у [1] формулювання квадратичної задачі