Сравнительный анализ алгоритмов вычисления текстовых метрик для документации программного обеспечения

Леденева Екатерина Юрьевна

Группа: 16.Б09-мм
Научный руководитель: д. т. н., профессор Д. В. Кознов
Рецензент: руководитель команды аналитики ООО "Интеллиджей Лабс" Н. И. Поваров

СПбГУ

9 июня 2020 г.

Примеры JavaDoc-комментариев

Из библиотеки Gson

```
/**
* convenience method to get this element as a primitive double.
*
* @return get this element as a primitive double.
* @throws NumberFormatException if the value contained is not a valid double.
*/
@Override
public double getAsDouble() {
    ...
```

```
/**
* convenience method to get this element as a boolean value.
*
* @return get this element as a primitive boolean value.
*/
@Override
public boolean getAsBoolean() {
    ...
```

Постановка задачи

- Цель — сравнительное исследование существующих алгоритмов вычисления сходства текстовых фрагментов для задачи определения сходных JavaDoc-комментариев в Java-приложениях
  - Изучение предметной области и выбор алгоритмов для исследования
  - Разработка и программная реализация инфраструктуры эксперимента
  - Проведение эксперимента и анализ результатов

Выбранные алгоритмы

- Longest Common Subsequence (LCS)
- Cosine Similarity (COS)
- Locality-Sensitive Hashing (LSH)
- Levenshtein Distance (LEV)

Модель эксперимента

- Данные (dataset)
  - 2638 пар JavaDoc-комментариев, сгенерированных из групп комментариев к нескольким известным Java-проектам (Slf4J, Mockito, GSON, JUnit)
- Классификация пар
  - Мера сходства — результат работы одного из алгоритмов
  - Подбор порога с помощью логистической регрессии

Процесс обработки (Pipeline)

- Нормализация — лемматизация, удаление пунктуации и стоп-слов (опционально)
- Сегментация
  - Алгоритмы применяется к одноимённым JavaDoc-тегам, получается набор чисел $d_i \in [0, 1]$
  - Результаты взвешенно суммируются: $d = \frac{\sum_i d_i \cdot l_i}{\sum_i l_i}$, где l_i — сумма длин тега в паре

Метрики

- J-K-fold кросс-валидация
  - F-мера
  - Accuracy
- ROC AUC
- Медиана и 95-й перцентиль для оценки быстродействия

Вопросы экспериментального исследования

1. Лучшая конфигурация процесса обработки
2. Лучший алгоритм
3. Оценка быстродействия
4. Сравнение с результатами статьи Soto и др. об использовании различных алгоритмов для поиска фрагментов переиспользования в DITA-документации*

*Axel J. Soto и др. «Similarity-Based Support for Text Reuse in Technical Writing». В: *Proceedings of the 2015 ACM Symposium on Document Engineering*. DocEng '15. ACM, 2015, с. 97—106.

Лучшая конфигурация процесса обработки и лучший алгоритм

Pipelines \ Алгоритмы	LCS	COS	LEV	LSH
Только алгоритм	96.77%±0.06%	92.85%±0.12%	92.12%±0.05%	92.02%±0.58%
Частичная нормализация	96.77%±0.09%	93.24%±0.03%	88.36%±0.14%	91.52%±0.01%
Полная нормализация	95.88%±0.04%	92.87%±0.04%	89.05%±0.11%	84.86%±0.02%
Сегментация	96.08%±0.09%	95.18%±0.05%	96.50%±0.06%	92.25%±0.09%
Сегм., частичная норм.	96.19%±0.09%	94.95%±0.11%	95.51%±0.08%	91.44%±0.07%
Сегм., полная норм.	96.04%±0.06%	94.95%±0.06%	95.77%±0.08%	83.65%±0.05%

Таблица: F-мера

- Лучшая конфигурация
  - LCS и LSH — в исходном виде
  - COS и LEV — с сегментацией и без нормализации
- Лучший алгоритм — LCS

Оценка быстродействия

Pipelines \ Алгоритмы	LCS	COS	LEV	LSH
Только алгоритм	0.259	0.199	1.863	5.467
Частичная нормализация	9.610	9.631	10.591	14.638
Полная нормализация	10.474	10.084	11.138	13.829
Сегментация	6.170	6.355	7.744	36.986
Сегм., частичная норм.	13.208	13.472	15.517	42.175
Сегм., полная норм.	14.378	15.062	16.564	43.988

Таблица: 95-й перцентиль времени работы в мс*

- Самый быстрый алгоритм — COS, самый медленный — LSH
- Нормализация и сегментация на порядок замедляют работу алгоритмов

*Компьютер: AMD Ryzen 5 3500U (2.1 ГГц, ОЗУ 8 Гб)

Сравнение со статьёй Soto и др.

- Лучший алгоритм — LCS, худший — LSH (и у нас, и в статье)
- LSH у нас оказался самым медленным, а в статье — самым быстрым

Результаты

- Изучена предметная область (проблема поиска неточных повторов в документации ПО, подход Duplicate Finder)
- Выбраны алгоритмы для исследования
  - Longest Common Subsequence (LCS), Cosine Similarity (COS), Levenshtein Distance (LEV), Locality-Sensitive Hashing (LSH)
- Разработана и реализована инфраструктура эксперимента
  - Собран набор из 2638 пар JavaDoc-комментариев
  - Создан процесс обработки из нормализации и сегментации
  - Оценено качество и быстродействие классификации
  - Реализация на языке Python с использованием библиотек NLTK, SciPy, scikit-learn, difflib, datasketch
- Выполнен эксперимент и проанализированы результаты
  - Лучшим алгоритмом оказался LCS
  - Процессы обработки не дали хороших результатов, но более чем на порядок увеличили время работы
  - Результаты частично совпали с результатами Soto и др.