Санкт-Петербургский государственный университет

Прикладная математика и информатика
Нелинейная динамика, информатика и управление

Горгулов Павел Олегович

Модель N-грамм для поиска нечетких повторов в «плоских» текстах

Бакалаврская работа

Научный руководитель:
д. т. н., профессор Кознов Д. В.

Рецензент:
ст. преп. Луцив Д. В.

Санкт-Петербург
2018

Saint Petersburg State University

Applied Mathematics and Computer Science
Nonlinear Dynamics, Computer Science and Control

Pavel Gorgulov

N-gram model in searching for fuzzy repetitions in “flat” texts

Graduation Thesis

Scientific supervisor:
professor Dmitry Koznov

Reviewer:
senior lecturer Dmitry Luciv

Saint Petersburg
2018

Оглавление

Введение

Многообразие современного программного обеспечения (ПО) создает необходимость детального информационного сопровождения его процессов. Техническая документация является неотъемлемым элементом любого продукта ПО и на протяжении жизненного цикла продукта подвергается регулярным модификациям. Подобные изменения повышают сложность структуры документации, ее объем, что делает дальнейшее сопровождение документации трудоемким процессом.

В ходе жизненного цикла документации активно используется копирование отдельных фрагментов, что приводит к накапливанию дублированного текстового материала. Таким образом, документация становится избыточной и, как следствие, снижается качество сопровождения ПО. Данная проблема ставит важную задачу — поиск и рефакторинг повторов фрагментов текста. Однако сложность заключается в том, что повторы могут быть не идентичными, представлены с разной степенью детализации, то есть каждый раз при копировании фрагмента, он подвергался редактированию. Поиск таких неточных повторов представляет особую сложность.

Решению этой проблемы посвящен проект DocLine [1, 2, 3]. Основная цель проекта — создание инструмента для разработки и сопровождения технической документации с повторами. В рамках проекта DocLine разрабатывается инструмент Duplicate Finder [4] для поиска неточных повторов в тексте и отслеживания их распределения по документу. Инструмент включает в себя алгоритм поиска по образцу и алгоритм экспресс оценки, позволяющий быстро определить неточные повторы в документе.

Также разработан алгоритм [5] для поиска неточных повторов в документации ПО с использованием современных методов обработки естественного языка, что позволяет найти рациональное решение проблемы поиска неточных повторов. Одним из таких методов является модель N-gram [6], которая позволяет находить семантически корректные повторы. Но у данного алгоритма имеются следующие недостатки

и ограничения:

- большое количество ложных срабатываний, нахождение неправильных или нерелевантных групп неточных повторов,
- нахождение групп, непригодных для дальнейшего рефакторинга.

Обращение к методам и алгоритмам обработки естественных языков (Natural Language Processing, NLP) [7], реализованным в виде инструментов и библиотек, позволяет доработать существующий алгоритм.

Постановка задачи

Цель данной работы заключается в доработке алгоритма поиска неточных повторов на основе N-грамм в документации программного обеспечения для проекта Duplicate Finder с применением методов обработки естественных языков. В связи с этим были сформулированы следующие задачи.

1. Изучение методов и стандартных алгоритмов обработки естественных языков, изучение инструментов и библиотек.
2. Выбор и обоснование алгоритмов и инструментов для улучшения существующего алгоритма поиска неточных повторов в Duplicate Finder.
3. Реализация решения.
4. Проведение экспериментов, анализ результатов, выводы.

1. Обзор

В этом разделе рассмотрим виды документации программного обеспечения, явление повторов в документации и проблемы, связанные с ними. Опишем существующий алгоритм в Duplicate Finder. Также опишем методы обработки естественных языков.

1.1. Документация программного обеспечения и повторы

Документация является важным аспектом разработки программного обеспечения. Понимание исходных текстов осложняется спецификой компьютерных или информационных систем, разнообразием языков: программирования, сценарных, предметно-ориентированных. Одна из известных попыток решения проблемы сопровождения исходного текста документацией была предпринята Д. Кнут в парадигме "Грамотное программирование" [8]. Д. Кнут озвучил важную проблему — исходный текст должен "понять" не только компьютер, но и человек.

Документацию можно разделить на несколько частей, таких как документация требований, архитектурная документация, техническая, пользовательская, маркетинговая [9, 10, 11]. Документация включает исходный код, учебные элементы, примеры, коды ошибок, где это необходимо. Может быть представлена в разных форматах: PDF, Word, XML и его разновидности. Документация облегчает поддержку, понимание, использование программного продукта. Такие задачи предполагают детальное описание продукта, и на данный момент остается актуальной дискуссия о необходимом количестве документации. В. Ройс в своей статье о водопадной модели [12] писал, что для разработки многомиллионного программного продукта, должно быть не менее 1000 страниц документации. При этом имелась в виду архитектурная документация.

Итак, при разнообразии и обширности документации программного обеспечения зачастую используется техника copy/paste. В результате

чего появляется большое количество повторов, которые необходимо отслеживать для дальнейшей поддержки документации. Наличие повторов в документации влияет на скорость восприятия текста и повышает его понятность. Принято считать, что качественная документация отличается унифицированностью повторов при описании идентичных аспектов, для этого сводится к минимуму употребление синонимов. Некоторые фрагменты при копировании подвергаются редактированию, затем скопированный фрагмент текста вставляется с изменениями. Такое понятие носит название неточный повтор [13] и набор таких фрагментов — это группа неточных повторов [14]. Поиск неточных повторов представляет собой отдельную задачу, для решения которой было придумано множество алгоритмов, позволяющих сопоставлять фрагменты текста не точно, а с некоторым количеством несовпадений между ними.

1.2. Существующий алгоритм поиска неточных повторов

Для повышения качества поиска следует применять новые методы извлечения информации, чтобы обеспечить надлежащий анализ в документации по программному обеспечению. Для этой цели подходят методы обработки естественного языка. Одним из таких методов является модель N-грамм. Он был положен в основу алгоритма для поиска неточных повторов в инструменте Duplicate Finder [4, 5].

Текст рассматривается как набор предложений. Для каждого предложения модель N-грамм включает в себя все последовательности (N-граммы), состоящие из n слов, где каждое слово непосредственно соответствует слову в исходном предложении. Поэтому каждый N-грамм является подстрокой соответствующего предложения. Например, если мы хотим обнаружить тот факт, что два предложения похожи, мы сравниваем их множества N-грамм. Одним из наиболее распространенных инструментов программирования для практического использования N-граммной модели является Natural Language Toolkit (NLTK) [15]. Он предоставляет ряд стандартных лингвистических операций и реализо-

ван в Python.

Рассмотрим подробнее алгоритм поиска неточных повторов, его полная спецификация представлена на листинге 1. Алгоритм извлекает предложения из текста и строит 3-граммное множество для каждого из них. После этого рассматриваются существующие группы и выбирается наиболее подходящая группа, которая содержит предложения и 3-граммное множество этих предложений. Критерий добавления повторов в группу такой, что если в группе содержится по крайней мере половина 3-граммного множества предложения из текста, предложение добавляется в данную группу, а множество 3-грамм дополняет множество группы. В случае, если такая группа не будет найдена, создается новая группа со взятым предложением. В итоге, алгоритм возвращает группы, содержащие два и более предложения. Эти группы образуют группы неточных повторов.

Опишем алгоритм детально. Ниже кратко объяснены основные функции и формальная спецификация, представленная в листинге 1.

- *intersect(A, B)* функция возвращает элементы, содержащиеся во множестве A и B.
- *size(A)* возвращает количество элементов, содержащихся в множестве A.
- *sent* — это массив предложений из текста документа.
  - *sent[i].nGrams* — это 3-граммное множество i-ого предложения.
- *groupSent* — это предложение из группы.
- *groups* — это набор неточных повторов.

```
1  for i = 1 to size(sent) do
2      i ← sent_i
3      bestOverlap ← 0
4      bestGroup ← NULL
5      for j = 1 to size(groups) do
6          curGroup ← groups_j
7          curIntersect ← intersect(curSent.nGrams, curGroup.nGrams)
8          curOverlap ← size(curIntersect)/size(curSent.nGrams)
9          if curOverlap > bestOverlap then
10             bestOverlap ← curOverlap
11             bestGroup ← curGroup
12         end
13     end
14     if bestOverlap < 0.5 then
15         create new group newGroup
16         newGroup.nGrams += curSent.nGrams
17         newGroup.sent += curSent
18     end
19     else
20         bestGroup.nGrams += curSent.nGrams
21         bestGroup.sent += curSent
22     end
23 end
24 for G in groups such that size(G) ← 1 do
25     groups -= G
26 end
27 return groups
```

Листинг 1: Алгоритм поиска неточных повторов

Прокомментируем листинг 1 построчно.

1. **Строки 1-22:** основной цикл документа, который выполняет итерации по всем предложениям документа.
2. **Строки 5-13:** цикл выбора лучшей группы. Для каждой группы:
   - 2.1. **Строка 7:** пересечения множества 3-грамм текущего предложения из текста и 3-граммного множества группы.
   - 2.2. **Строка 8:** вычисляем отношение мощности данного пересечения к мощности 3-граммного множества предложения.
   - 2.3. **Строка 9-12** если текущая группа является лучшей из обработанных, мы запоминаем ее.

3. **Строка 14:** условие проверки отношения.

    3.1. **Строки 15-17:** если отношение меньше 0.5, мы создаем новую группу и добавляем предложение в нее.

    3.2. **Строка 19, 20:** иначе, мы добавляем предложение в найденную группу.

4. **Строки 23-25:** удаляем группы, содержащие одно предложение.

1.3. Методы обработки естественных языков

Обработка естественных языков занимает значительное место в существующих системах анализа текста. Под обработкой естественных языков понимается решение задач, связанных с распознаванием речи, синтаксическим и семантическим анализом текста, машинным переводом и генерацией текстов [16].

Выделяют несколько групп методов обработки естественных языков по соответствующим задачам:

- синтаксические,
- семантические,
- естественная речь.

Синтаксические методы обработки основаны на морфологической организации естественного языка и строении единиц исходного текста. К группе синтаксических методов относят лемматизацию, стемминг, удаление пунктуации, выделение элементов текста. Лемматизация и стемминг — это методы нормализации, то есть приведения словоформ к одной форме. Лемматизация выделяет морфемы и преобразует словоформу в начальную форму — лемму [17]. Сложность данного метода зависит от морфологии языка. Стемминг работает проще: выделяется основа словоформы вне зависимости от морфологии [18]. Важным алгоритмом является лексическая декомпозиция, которая выделяет элементы в тексте — токены. Соответственно методы выполняющие данную

задачу называются токенайзерами [19]. Удаление пунктуации связано с этими методами, так как пунктуационные знаки являются единицами предложения.

Семантические методы занимаются выделение смысла фрагментов в тексте, соотношением контекста и обозначений в тексте. К таким методам относятся определение части речи слов, выделение имен нарицательных в исходном тексте, связь слов в предложении, определение тональности текста. Методы из данных категорий могут переплетаться. В основе этих методов в большинстве своем лежат методы машинного обучения. Так, например, метод кластеризации может быть применен к распознаванию частей речи. Также к семантическим методам определяют генерацию естественного языка — преобразование информации из баз данных в естественный язык [20].

К задаче естественной речи относят большую группу методов распознавания речи, такие как сегментация речи [21], то есть определения границы между словами. В основе таких методов часто используют скрытую марковскую модель [22].

2. Архитектура решения

2.1. Изменение существующего алгоритма

При рассмотрении алгоритма поиска неточных повторов, псевдокод которого представлен на листинге 1, можно заметить, что критерий добавления повторов в группу нуждается в доработке. При увеличении группы, увеличивается вероятность попадания в группу предложения, никак не связанного семантически с содержащимися предложениями в группе.

Исходя из существующей проблемы, доработаем критерий таким образом, чтобы увеличилась точность при добавлении предложения в группу. То есть решение о добавление предложения в группу принимается только если предложение соотносится с имеющимися в группе предложениями.

Для этого рассмотрим документ D как набор предложения. Пусть G — группа неточных повторов, содержащая предложения g_i, а st — предложение, взятое из D. ${g_i}^3$ и st^3 — множество 3-грамм предложений g_i и st соответственно. Тогда назовем k точностью совпадений предложений. Число k такое, что $1/2 \leq k \leq 1$, так как при k близком к нулю, предложения будут с небольшой долей похожести, что нас не интересует. Оптимально рассматривать случай, когда $k \geq 1/2$, и при выполнении следующего условия, добавить предложение st в группу G:

$$\frac{|\bigcap_{i=1}^{n} {g_i}^3 \cap st^3|}{|st^3|} \geq k, \tag{1}$$

где $|\bigcap_{i=1}^{n} {g_i}^3 \cap st^3|$ — это мощность пересечения множеств 3-грамм предложений из группы G и предложения из D, а $|st^3|$ — это мощность множества 3-грамм предложения st.

Таким образом, если при пересечении всех 3-граммных множеств каждого предложения из группы, в этом пересечение содержится как минимум половина 3-граммное множество предложения из текста, то

данное предложение добавляется в группу. Данное изменение критерия обеспечивает более высокую точность соответствия предложений в группе неточных повторов и позволяет избавиться от множества 3-грамм группы за ненадобностью.

С такими изменениями (1) алгоритм будет выглядеть следующим образом (Листинг 2).

```
1  for i = 1 to size(sent) do
2      i ← sent_i
3      bestOverlap ← 0
4      bestGroup ← NULL
5      for j = 1 to size(groups) do
6          curGroup ← groups_j
7          curIntersect ← curSent.nGrams
8          for groupSent in curGroup.sents do
9              curIntersect ← intersect(curIntersect, groupSent.nGrams)
10         end
11         curOverlap ← size(curIntersect)/size(curSent.nGrams)
12         if curOverlap > bestOverlap then
13             bestOverlap ← curOverlap
14             bestGroup ← curGroup
15         end
16     end
17     if bestOverlap < k then
18         create new group newGroup
19         newGroup.sent += curSent
20     end
21     else
22         bestGroup.sent += curSent
23     end
24 end
25 for G in groups such that size(G) ← 1 do
26     groups -= G
27 end
28 return groups
```

Листинг 2: Модернизированный алгоритм поиска неточных повторов

Опишем внесенные изменения в существующий алгоритм поиска неточных повторов, представленный на листинге 1.

1. **Строка 7:** множество 3-грамм предложения из текста, кандидата на добавление.
2. **Строки 8-10:** пересечение каждого множества 3-грамм предложений из группы и предложения из текста.

3. **Строки 11-15:** действия аналогичны Алгоритму 1. Вычисляем отношение мощности конечного пересечения и мощность множества 3-грамм предложения из текста, затем запоминаем лучшую группу.
4. **Строки 17-22:** условие проверки отношения. Если отношение меньше k, создаем новую группу и добавляем предложение в нее, иначе — добавляем предложение в найденную группу.

2.2. Средства предварительной фильтрации текста

Из многочисленных методов обработки естественных языков для задачи поиска неточных повторов в тексте представляют значение синтаксические методы. Были выбраны методы токенизации, удаление пунктуации и стоп-слов, нормализации: лемматизации и стемминга. Данные методы позволяют устранить следующие проблемы при поиске неточных повторов.

- Избыточное употребление пунктуационных знаков.
- Использование слов, не влияющих на смысловое содержание предложения.

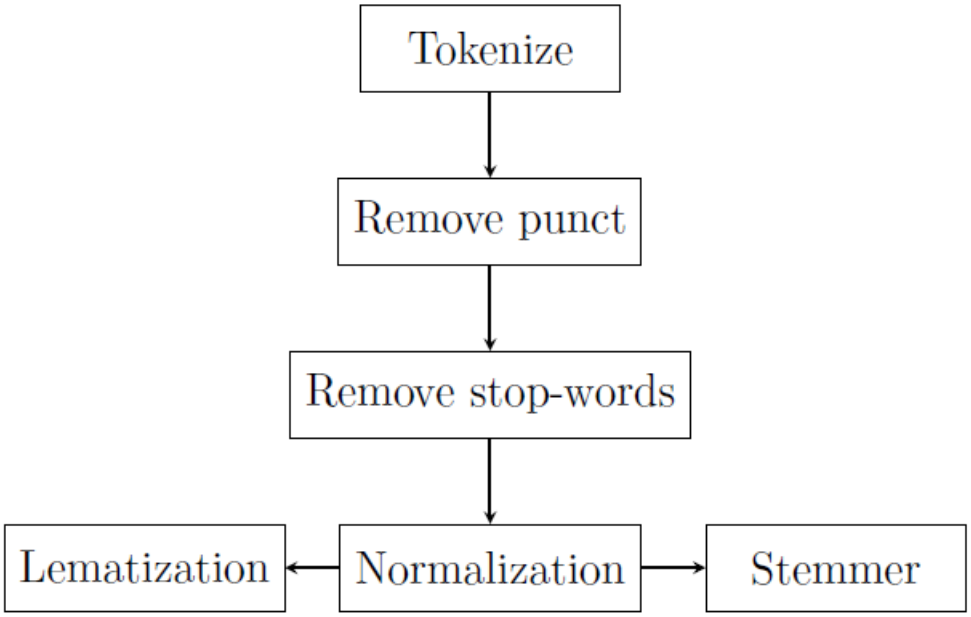

Рис. 1: Последовательность применения методов.

- Проблема омонимии.
- Морфологические вариации.

Знаки пунктуации не влияют на поиск повторов, в то же время они могут усложнить поиск, если в тексте документа их находится очень много. Идентично с так называемыми стоп-словами [23]. Слова, удаление которых не влияет на смысловое содержание текста.

Омонимия — это явление, когда разные по значению слова совпадают по написанию в отдельных грамматических формах. Повторы, в которых содержатся омонимы семантически некорректны.

Проблема морфологических вариаций так же создает трудности при задаче поиска повторов. Слова в контексте одного нечеткого повтора могут изменяться по падежам, иметь другое склонение, множественное или единственное число.

Следующие этапы предобработки текста в последовательности их применения показаны на рис. 1.

Самым первым этапом предобработки текста является распознавание лексических единиц текста. Для этого используется токенизатор. Исходный текст разбивается на токены предложений, затем внутри каждого предложения также слова заменяются токенами. Как правило, токены совпадают со словоформами, но все равно используется термин "токен", а не "слово", так как под токеном могут пониматься отдельные морфемы. Например, после нормализации.

В документации используется общепринятый набор знаков. Кроме того, применение в документации пунктуационных знаков особенно обширно в связи с формальными оборотами и установленными правилами оформления.

Основная задача пунктуации выделить синтаксический и интонационный строй текста для упрощения восприятия устной и письменной речи. Но при поиске повторов нет необходимости в перечисленных выше задачах. Более того, изобилие пунктуационных знаков в тексте при поиске неточных повторов, может расцениваться как совпадение элементов при пересечении, что приведет к некорректному результату.

К категории стоп-слов относят слова, не влияющие на семантическую составляющую текста, такие как предлоги, междометия, частицы, но необходимые для нормального восприятия текста, его читабельности. Стоп-слова характеризуются высокочастотным использованием в тексте и такие слова ”загрязняют”, ”зашумляют” текст, в результате чего трудно выделить семантическую составляющую отдельно взятого предложения. Рассмотрим на примере в (Таблице 1). Как видно, оба предложения исходного текста являются неточным повтором от ”Doctor Science”, но повтор не будет распознан из-за использования стоп-слов. Удалив мешающие слова, получится, что оба предложения будут распознаны как неточный повтор. Особенно употребление стоп-слов характерно для документации, в которой в большом количестве содержатся вводные слова и формальные речевые обороты соответствующие стилю.

Исходный текст	*She has been a Doctor of Science since 2009.* *As it thought she is a Doctor of Science.*
После удаления стоп-слов	*has Doctor Science 2009.* *Thought Doctor Science.*

Таблица 1: Пример удаления стоп-слов

Для корректного поиска неточных повторов необходимо избавиться от грамматической информации о слове, оставив смысловое значение. При использовании лемматизации слово приводится к лемме — начальной словарной форме слова. Для имени существительного лемма — форма единственного числа, именительного падежа, для глагола — инфинитив.Каждый токен словоформы заменяется токеном, в основе которого лежит лемма.

Для англоязычных текстов используется алгоритмическая лемматизация. Существует список морфем, таких как ”-s”, ”-ty” ”-ly” и другие, которые выделяются и в дальнейшем удаляются. Это видно в (Таблице 2).

Для русскоязычных текстов эта задача намного сложнее, так как русский язык имеет сложную систему морфологического словообразования, отличную от английской парадигмы. Пример работы лемматизатора для русских слов можно увидеть в (Таблице 3).

Исходный текст	*The charity concert starts approximately at 5 pm.*
Лемматизация	*The chari concert start approximate at 5 pm.*

Таблица 2: Лемматизация английских слов

Исходный текст	*Красивая мама красиво мыла раму*
Лемматизация	*Красивый мама красиво мыть рама*

Таблица 3: Лемматизация русских слов

При стемминге семантически схожие словоформы приводятся к общей основе. Данный метод базируется на том, что количество морфем в языке ограничено. При обработке текста используя стеммер, получаем токены, где под токеном определяется основа слова.

По сравнению с лемматизатором стеммер прост в реализации как для английского языка, так и для русского. Также скорость выполнения стемминга заметно выше по сравнению с лемматизацией. В русском языке слова имеют сложную морфологическую изменяемость, поэтому основа слова не всегда совпадает с корнем. Но в рамках документации это не является ключевым фактором, так как большинство таких слов используются в литературных текстах.

В настоящее время в большинстве случаев используется стеммер Портера [24]. Данный стеммер показал эффективную работу, которая измерялась соотношением количества словоформ исходного текста и основ слов к словам, оставшимся после работы стемминга [25].

3. Реализация

Разберем существующие программные инструменты выбранных нами методов обработки естественных языков. Алгоритм поиска неточных повторов (Листинг 1) изначально реализован на языке программирования Python и разбивка предложения на множество 3-грамм взята из пакета nltk [15], поэтому все инструменты методов NLP выбраны для этого языка.

Реализацию токенизации предоставляют два пакета библиотек обработки естественного языка nltk и Spacy [26]. Также существует библиотека морфологического анализатора pymorphy2 [27]. Данные инструменты работают идентично друг другу, поэтому была выбрана библиотека tokenize из пакета nltk, чтобы уменьшить импорт сторонних библиотек.

Удаление пунктуационных знаков происходит благодаря стандартной функции *"filter"* из Python. Функция получает набор знаков, которые задаются самостоятельно и исключаются из текста документа. Передача собственного множества знаков позволяет удалить специальные символы, как "{", "}", "|" и так далее. Эти символы отсутствуют в методе из nltk, при этом часто встречаются в документации.

Списки стоп-слов вариативны и применяются в соответствии с разными задачами обработки естественного языка. Эти списки выложены в открытый доступ обычными пользователями, а также существуют стандартные наборы стоп-слов в nltk_data [28] и spacy. Удаление стоп-слов происходит аналогично удалению пунктуации. Только вместо набора пунктуационных знаков, функция получает набор стоп-слов. Этот набор взят из библиотеки corpus в nltk_data. Данный набор покрывает большинство стоп-слов как в английском, так и в русском языке. Также есть возможность добавить свой набор стоп-слов характерный для специфической документации. Исключение из массива текста широкоупотребляемых и семантически незначительных конструкций обеспечит поиск по важным смысловым элементам документации.

В nltk отсутствует лемматизатор для русского языка, поэтому был

использован pymystem3 [29]. Этот модуль содержит морфологический анализатор, в том числе лемматизатор, для русского языка Mystem 3.0 [30] от компании Yandex.

В библиотеке stem из пакета nltk существует стемминг Портера. Именно из stem взята реализация стемминга для англоязычных текстов.

4. Эксперименты

4.1. Данные для экспериментов

В качестве данных для проведения экспериментов были взяты четыре документа из открытого источника, также был взят коммерческий документ:

- Linux Kernel documentation (LKD), 892 KB in total [31];
- Zend Framework documentation (Zend), 2924 KB in total [32];
- DocBook 4 Definitive Guide (DocBook), 686 KB in total [33];
- Version Control with Subversion (SVN), 1810 KB in total [34];
- Commercial project user guide (CProj), 164 KB in total.

К данным документам был применен существующий алгоритм поиска неточных повторов и модернизированный алгоритм поиска неточных повторов. Результатом работы обоих алгоритмов является набор групп неточных повторов.

4.2. Пример работы улучшенного алгоритма поиска неточных повторов

Рассмотрим на листинге 3, листинге 4, листинге 5 примеры групп неточных повторов, найденные модернизированным алгоритмом в документе "Linux Kernel documentation" (список групп неточных повторов представлен в Приложении А).

Как видно, модернизированный алгоритм поиска неточных повторов действительно находит группы неточных повторов.

```
1 Each function and struct member has a short description which is marked with an [XXX] identifier.
2 Each struct member has a short description which is marked with an [XXX] identifier.
3 Each function has a short description which is marked with an [XXX] identifier.
4 Each function has a short description which is marked with an [XXX] identifier.
```

Листинг 3: Группа неточных повторов

1 If the checks fail to pass the function returns a non zero error code.
2 This returns a non zero error code on failure.

Листинг 4: Группа неточных повторов

1 The instance is allocated and zeroed by the driver, possibly as part of a larger structure, and registered with a call to drm_crtc_init with a pointer to CRTC functions.
2 The instance is allocated and zeroed by the driver, possibly as part of a larger structure.

Листинг 5: Группа неточных повторов

4.3. Анализ результатов

Сформулируем оценочные вопросы.

- **Вопрос 1:** Как изменились результаты при изменении критерия добавления предложения в группу?
- **Вопрос 2:** Как повлияли на результаты используемые методы предварительной фильтрации текста?
- **Вопрос 3:** Какие количественные изменения между модернизированным алгоритмом и существующим?

При использовании нового критерия, в группах неточных повторов пропали некорректные предложения, включенные в группу. Например, в листинге 6 с новым критерием последнее предложение не войдет в группу, так как оно не является корректным по отношению к уже содержащимся в группе.

1 qc-amp;gt;complete_fn() callback is the asynchronous path used by normal SCSI translated commands and qc-amp;gt;waiting is the synchronous (issuer sleeps in process context) path used by internal commands.
2 qc-amp;gt;complete_fn() callback is used for completion notification.
3 qc-amp;gt;complete_fn() callback is invoked.
4 ap-amp;gt;active_tag and qc-amp;gt;tag are poisoned.

Листинг 6: Группа неточных повторов, найденная существующим алгоритмом

Удаление пунктуации в конечном результате исключило группы с большим содержанием пунктуационных знаков. Как правило, предложения таких групп состоят из ”$”, ”#”, ”*” и других знаков. При удалении стоп-слов можно увидеть, что пропали группы, состоящие из формальных подписей к таблицам, рисункам и прочему. Также стало гораздо меньше групп, состоящих из одного большого и одного небольшого

предложения. Следующий пример 7 показывает группу, отсутствующую при поиске с использованием удаления стоп-слов.

```
1 Planes (struct drm_plane) A plane represents an image source that can be blended with or
  overlayed on top of a CRTC during the scanout process.
2 The result is then blended with or overlayed on top of a CRTC.
```

Листинг 7: Группа неточных повторов, найденная существующим алгоритмом

Нормализация слов текста привнесла существенные изменения в результат. Исключились некорректные группы, такие как показаны в листинге 8. Также появились новые группы, одна из них представлена в листинге 9.

```
1 That&#39;s just a whitelist, used to reject peripherals not supported with a given Linux OTG host.
2 That&#39;s what uio_pdrv_genirq does.
3 EXPORT_NO_SYMBOLS; That&#39;s all!
```

Листинг 8: Некорректная группа неточных повторов

```
1 It is recommended to use the name of your kernel module for this.
2 I recommend using the name of your module for this.
```

Листинг 9: Новая группа неточных повторов

Сформируем численные оценки и статистики. В Таблицах 4 и 5 представлены количественные результаты работы двух алгоритмов. Результаты сравнивались по трем параметрам:

- общее количество найденных групп неточных повторов,
- среднее количество предложений в группах,
- среднее количество слов в группах.

Из таблиц видно, что в четырех документах из пяти снизилось общее количество групп неточных повторов. Это объясняется тем, что при использовании предварительной фильтрации текста уменьшаются ложные срабатывания. При этом существуют небольшие изменения, касающееся содержания среднего количества предложений и слов в группе, то есть содержание большинства групп не претерпело больших изменений.

Document	Количество групп	Среднее кол-во предложений в группе	Среднее кол-во слов в группе
LKD	189	3.14	124.97
Zend	576	2.58	59.63
DocBook	82	2.23	85.54
SVN	343	2.9	93.98
Cproj	72	3.76	106.29

Таблица 4: Статистические данные работы существующего алгоритма

Document	Количество групп	Среднее кол-во предложений в группе	Среднее кол-во слов в группе
LKD	166	2.85	111.2
Zend	482	2.38	57.18
DocBook	62	2.2	82.23
SVN	334	2.68	94.26
Cproj	73	3.41	101.05

Таблица 5: Статистические данные работы модернизированного алгоритма

В целом, можно сделать вывод, что модернизированный алгоритм находит более корректные группы неточных повторов по сравнению с существующим алгоритмом поиска неточных повторов.

Заключение

В рамках выпускной квалификационной работы были достигнуты следующие результаты.

1. Изучены методы и алгоритмы обработки естественных языков.
2. Были выбраны библиотеки corpus, stem, tokenize из пакета NLTK для получения списка стоп-слов, реализации стеммера и токенайзера соответственно. Также была выбрана библиотека pymystem3 для реализации лемматизатора.
3. Реализован модернизированный алгоритм поиска неточных повторов.
4. На основе проведенных экспериментов сделан анализ полученных результатов. Анализ показал, что доработанный алгоритм с использованием методов обработки естественных языков находит релевантные и более корректные группы неточных повторов, пригодные для дальнейшего рефакторинга.

Список литературы

[1] Minchin, L. Refactoring the Documentation of Software Product Lines / L. Minchin, K. Romanovsky, D. Koznov // Lecture Notes in Computer Science. — 2011. — Vol. 4980. — P. 158–170.

[2] Кознов, Д.В. Поиск клонов при рефакторинге технической документации / Д.В. Кознов, А.В. Шутак, М.Н. Смирнов, М.А. Смажевский // Компьютерные инструменты в образовании. — 2012. — no. 4. — P. 30–40.

[3] Романовский, К.Ю. DocLine: метод разработки документации семейств программных продуктов / К.Ю. Романовский, Д.В. Кознов // Программирование. — no. 4. — P. 1–13.

[4] Duplicate Finder. — URL: `http://www.math.spbu.ru/user/kromanovsky/docline/duplicate-finder-en.html`.

[5] Кантеев, Л.Д. Обнаружение неточно повторяющегося текста в документации программного обеспечения / Л.Д. Кантеев, Ю.О. Костюков, Д.В. Луцив, Д.В. Кознов, М.Н. Смирнов // Труды Института системного программирования РАН. — 2017. — Vol. 29, no. 4. — P. 303–314.

[6] Broder, A.Z. Syntactic clustering of the web / A.Z. Broder. — 1997. — Vol. 29. — P. 1157–1166.

[7] Chowdhury, G. Natural Language Processing / G. Chowdhury // Annual Review of Information Science and Technology. — 2003. — Vol. 37. — P. 51–89.

[8] Knuth, Donald E. Literate programming / Donald E. Knuth // The Computer Journal. British Computer Society. — 1984. — P. 97–111.

[9] Earle, Ralph H. User preferences of software documentation genres / Ralph H. Earle, Mark A. Rosso, Kathryn E. Alexander // Proceedings

of the 33rd Annual International Conference on the Design of Communication. — No. 46. — 2015.

[10] Garousi, G. Usage and usefulness of technical software documentation: An industrial case study / G. Garousi, V. Garousi-Yusifoglu, G. Ruhe, J. Zhi, M. Moussavi, B. Smith // Information and Software Technology. — 2015. — Vol. 57. — P. 664–682.

[11] van Heesch, U. A documentation framework for architecture decisions / U. van Heesch, A. Avgeriou, R. Hilliard // The Journal of Systems and Software. — 2011.

[12] Royce, W.W. Management the development of large software systems / W.W. Royce // Proceedings of the 9th international conference on Software Engi-neering. — 1987. — P. 328–338.

[13] Bassett, P.G. The Theory and Practice of Adaptive Reuse / P.G. Bassett // Pro-ceedings of SIGSOFT Software Engineering Notes. — 1997. — P. 2–9.

[14] Koznov, D.V. Detecting Near Duplicates in Software Documentation / D.V. Koznov, D.V. Luciv, G.A. Chernishev, A.N. Terekhov // ArXiv EPrint. — 2017.

[15] Natural Language Toolkit Documentation. — URL: `https://www.nltk.org/`.

[16] Когаловский, М.Р. Перспективные технологии информационных систем / М.Р. Когаловский. — 2003. — Р. 288.

[17] Liu, H. BioLemmatizer: A lemmatization tool for morphological processing of biomedical text / H. Liu, T. Christiansen, W. A. Baumgartner, K. Verspoor // Journal of Biomedical Semantics. — 2012. — Vol. 3.

[18] Lovins, J. B. Development of a Stemming Algorithm / J. B. Lovins // Mechanical Translation and Computational Linguistics. — 1968. — Vol. 11. — P. 22–31.

[19] Bumbulis, P. RE2C: A more versatile scanner generator / P. Bumbulis, D. D. Cowan // ACM Letters on Programming Languages and Systems. — 1993. — Vol. 2. — P. 70–84.

[20] Perera, R. Recent Advances in Natural Language Generation: A Survey and Classification of the Empirical Literature / R. Perera, P. Nand // Computing and Informatics. — 2017. — Vol. 36. — P. 1–32.

[21] Badecker, W. Morphological Parsing and the Perception of Lexical Identity: A Masked Priming Study of Stem Homographs / W. Badecker, A. Mark // Journal of Memory and Language. — 2002. — P. 125–144.

[22] Baum, L. E. An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model for Ecology / L. E. Baum, J. A. Eagon // Bulletin of the American Mathematical Society. — 1967. — Vol. 73. — P. 360–363.

[23] Flood,. Historical note: The Start of a Stop List at Biological Abstracts / Flood, J. Barbara // Journal of the American Society for Information Science. — 2016.

[24] Porter, M.F. An algorithm for suffix stripping / M.F. Porter // Program. — 1980. — P. 130–137.

[25] Яцко, В.А. Алгоритмы и программы автоматической обработки текста / В.А. Яцко // Вестник ИГЛУ. — 2012. — Р. 150–160.

[26] Industrial-Strength Natural Language Processing. — URL: `https://spacy.io/`.

[27] pymorphy2. — URL: `https://github.com/kmike/pymorphy2`.

[28] Natural Language Toolkit Data. — URL: `http://www.nltk.org/nltk_data/`.

[29] pymystem. — URL: `https://pypi.python.org/pypi/pymystem3/0.1.0`.

[30] Yandex Mystem 3.0. — URL: https://tech.yandex.ru/mystem/.

[31] Linux Kernel Documentation, snapshot on Dec 11, 2013. — URL: https://github.com/torvalds/linux/tree/master/Documentation/DocBook/.

[32] Zend PHP Framework documentation, snapshot on Apr 24, 2015. — URL: https://github.com/zendframework/zf1/tree/master/documentation.

[33] DocBook Definitive Guide, snapshot on Apr 24, 2015. — URL: http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/.

[34] SVN Book, snapshot on Apr 24, 2015. — URL: http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/.

Приложение А. Примеры групп неточных повторов

В данном приложении представлены примеры групп неточных повторов, найденных в документе Linux Kernel documentation. Данные 15 примеров были выбраны из 166 групп и показаны в листингах ниже.

```
1 If the checks fail to pass the function returns a non zero error code.
2 This returns a non zero error code on failure.
```

Листинг 10: Группа неточных повторов

```
1 You need to define the copy and silence callbacks for the data transfer.
2 Thus you need to define the copy and silence callbacks as well, as in the cases above.
```

Листинг 11: Группа неточных повторов

```
1 page callback This callback is optional too.
2 This callback is optional.
```

Листинг 12: Группа неточных повторов

```
1 Another note is that this callback is non-atomic (schedulable).
2 Note that this callback is now non-atomic.
```

Листинг 13: Группа неточных повторов

```
1 Note that this and prepare callbacks may be called multiple times per initialization.
2 Also, the callback may be called multiple times, too.
```

Листинг 14: Группа неточных повторов

```
1 Buffer management details will be described in the later section Buffer and Memory Management.
2 The detailed will be described in the later section Buffer and Memory Management.
3 Some examples will be explained in the later section Buffer and Memory Management, too.
```

Листинг 15: Группа неточных повторов

```
1 The third argument (index, 0 in the above) is the index of this new pcm.
2 The third argument is the index of this component.
```

Листинг 16: Группа неточных повторов

```
For accessing to the PCM layer, you need to include &amp;lt;soundpcm.h&amp;gt; first.
The definition of runtime instance is found in &amp;lt;soundpcm.h&amp;gt;.
The API is provided in &amp;lt;soundpcm.h&amp;gt;.
```

Листинг 17: Группа неточных повторов

```
The ALSA interfaces like the PCM and control APIs are defined in other
&amp;lt;soundxxx.h&amp;gt; header files.
The control API is defined in &amp;lt;soundcontrol.h&amp;gt;.
```

Листинг 18: Группа неточных повторов

```
pci directory This directory and its sub-directories hold the top-level card modules for PCI
soundcards and the code specific to the PCI BUS.
isa directory This directory and its sub-directories hold the top-level card modules for ISA
soundcards.
```

Листинг 19: Группа неточных повторов

```
File modification time is not updated by this request.
File modification time is not updated by this request.
File modification time is not updated by this request.
USBDEVFS_DISCARDURBTBS File modification time is not updated by this request.
USBDEVFS_DISCSIGNALTBS File modification time is not updated by this request.
USBDEVFS_REAPURBTBS File modification time is not updated by this request.
USBDEVFS_REAPURBNDELAYTBS File modification time is not updated by this request.
```

Листинг 20: Группа неточных повторов

```
The ioctl parameter is an integer holding the number of the interface (bInterfaceNumber from
descriptor).
The ioctl parameter is an integer holding the number of the interface (bInterfaceNumber from
descriptor); File modification time is not updated by this request.
The ioctl parameter is ignored.
```

Листинг 21: Группа неточных повторов

```
The simple style is to make only control requests; some devices don&#39;t need more complex
interactions than those.
USBDEVFS_CONTROLIssues a control request to the device.
```

Листинг 22: Группа неточных повторов

```
It can work with any device compliant to PCI 2.3 (circa 2002) and any compliant PCI Express
device.
All devices compliant to PCI 2.3 (circa 2002) and all compliant PCI Express devices should support
these bits.
```

Листинг 23: Группа неточных повторов

```
1 You should have received a copy of the GNU General Public License along with this program; if
  not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
  02111-1307 USA For more details see the file COPYING in the source distribution of Linux.
2 You should have received a copy of the GNU General Public License along with this program; if
  not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
  02111-1307 USA For more details see the file COPYING in the source distribution of Linux.
3 You should have received a copy of the GNU General Public License along with this program; if
  not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
  02111-1307 USA For more details see the file COPYING in the source distribution of Linux.
4 You should have received a copy of the GNU General Public License along with this program; if
  not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
  02111-1307 USA For more details see the file COPYING in the source distribution of Linux.
5 You should have received a copy of the GNU General Public License along with this program; if
  not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
  02111-1307 USA For more details see the file COPYING in the source distribution of Linux.
```

Листинг 24: Группа неточных повторов