Санкт-Петербургский государственный университет

Математика и Механика

Математическое обеспечение и
администрирование информационных систем

Профиль системного программирования

Коновалова Ирина Михайловна

# Поиск и кластеризация нечетких повторов в документации программного обеспечения

Магистерская выпускная работа

Научный руководитель:
ст. преп., к.ф.-м.н. Д.В. Луцив

Рецензент:
менеджер проектов ООО «СКЗ»
Н.А. Пенкрат

Санкт-Петербург
2020

SAINT PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software and Administration of Information Systems

Software Engineering

Konovalova Irina

# Detection and Clustering of Near Duplicates in Software Documentation

Master's graduation thesis

Scientific supervisor:

PhD, Senior lecturer D. Luciv

Reviewer:

LLC 'Computer Vision Systems'

Project manager, N. Penkrat

Saint Petersburg

2020

# Содержание

# Введение

Документация играет значительную роль при создании и сопровождении программных продуктов. В крупных и долгосрочных проектах она является важным средством передачи знаний пользователям и разработчикам о принципах и деталях работы разрабатываемого ПО. На протяжении десятелетий представители как науки, так и индустрии, признавали это, но наряду с важностью документации они часто констатировали её недооцененность и проблемность, как актива программного проекта. В 1970-х Ф. Брукс в книге «Мифический человеко-месяц» [1] отмечает дефицит внимания к документации в софтверных проектах. На рубеже XX и XXI вв. отечественные исследователи отмечают проблемы с качеством документации и с недостаточно ответственным отношением к ней при разработке ПО [2, 3, 4]. Эта тенденция сохраняется и в настоящее время [5, 6, 7], поэтому тематика улучшения качества документации важна для программной инженерии.

Созданием документации обычно занимается целая команда разработчиков (программистов, технических писателей, архитекторов), и в течение жизненного цикла продукта в неё вносится множество изменений. Стандартизация данного процесса, в основе которого лежит переиспользование и унификация фрагментов текста, упрощает поддержку документации в актуальном состоянии. В то же время, копирование и последующее частичное изменение текста приводят к накоплению ошибок и затрудняет дальнейшее использование документации и модификацию её взаимосвязанных частей.

Результат такого изменения документации — копирования фрагментов текста и не систематизироанного изменения их копий — характеризуется термином «нечеткие повторы». Поиск, отслеживание и улучшение текста таких повторов позволяет лучше следовать правилам стандартизации и повышает общее качество текста [8]. Сопроводительная документация современных

программных продуктов — это документы, исчисляемые десятками и сотнями мегайбайт [9, 10, 11], поэтому задача автоматического поиска нечетких повторов для дальнейшего переиспользования и редактирования является актуальной.

Некоторые существующие работы, посвященные поиску нечетких повторов, основываются на методах поиска клонов в программном обеспечении [8, 5, 6]. Также есть работы, основанные на алгоритмах из области анализа естественных языков [12]. Количество ложных срабатываний описанных в этих работах алгоритмов достаточно велико, и может достигать 40%. Такие значения говорят о том, что применение данных алгоритмов в индустрии возможно лишь в полуавтоматическом режиме, так как они всё ещё требуют вмешательства человека для экспертизы полученных результатов. Кроме того, как отмечалось в упомянутых работах, автоматический поиск повторов в произвольном тексте не может обеспечить их осмысленность, и правильное определение границ осмысленного участка текста — очень сложная для решения в общем виде задача.

В довершение, при работе с документацией недостаточно только найти повторы в тексте, необходимо также их должным образом сгруппировать, так как это значительно облегчит последующий анализ результатов поиска и в итоге поможет разработчикам в улучшении существующей документации.

Некоторые виды документации (обычно это справочная документация в таких форматах, как, например DITA [13] или Javadoc [14]) исходно разбиты на небольшие осмысленные фрагменты, которые могут повторяться точно или приблизительно. Это позволяет решенить задачи поиска и группировки осмысленных повторов. При работе с документацией такого вида рациональным для группировки повторов видится использование кластеризации — механизма идентификации классов объектов в коллекции таким образом, чтобы похожие объекты были назначены одному и тому же классу.

Упомянутые выше исследования повторов в документации программного

обеспечения [5, 6] продолжаются, и полученные в данной магистерской работе результаты призваны облегчить разработку и оценку качества новых подходов к обнаружению и анализу дублирующихся данных в документации программного обеспечения, а также обеспечить автоматизацию тестирования создаваемых алгоритмов на тестовых коллекциях документов.

В данной работе проводится обзор векторных представлений текста, алгоритмов кластеризации и способов оценки их качества для задачи поиска нечетких повторов в документации, после чего формулируются вопросы экспериментального исследования и разрабатывается методика проведения эксперимента для апробации применения рассмотенных методов в задаче поиска повторов. Структура дальнейшей работы строится в соответствии с вопросами исследования: рассматриваются векторные представления текстовых повторов, метрики оценки качества кластеризации и алгоритмы кластеризации повторов.

# Цель работы

Целью работы является исследование возможности применения различных методов обработки текста и алгоритмов кластеризации для задачи поиска неточных повторов в документации программного обеспечения. Для достижения этой цели были сформулированы следующие задачи:

1. Выбрать методы обработки текста на естественном языке и метрики для сравнения их применения на эталонной коллекции.

2. Выбрать алгоритмы кластеризации и метрики оценки их качества.

3. Создать программную среду для апробации создаваемых алгоритмов и разработать методику проведения экспериментов.

# 1 Обзор

## 1.1 Документация программного обеспечения

Существует много различных видов документации ПО и много различных способов ее классифицировать. Несколько таких способов перечислены в обзрной части работы [8]. Простая классификация, разделяющая документацию на справочную (reference) и описательную (narrative) была предложена в 2011 г. Д. Парнасом [15]. Как показано в упомянутых работах [5, 6, 7], при работе со справочной документацией ценной оказывается информация о повторах в ней: поскольку справочная документация описывает много сущностей одной природы (классов, методов и т.д.). Повторы в документации являются предметом также и данной работы, поэтому в качестве данных для апробации предложенных в ней подходов и методик используется справочная документация.

Распространенным видом справочной документации программных проектов является встроенная API-документация, состоящая из фрагментов текста, описывающих различные сущности программного кода — методы, атрибуты, классы, пакеты и т.д. в виде специальным образом отформатированных комментариев. По встроенной документации можно генерировать справочники в различных форматах (PDF, HTML и т.д.). Интегрированные среды разработки (IDE) также поддерживают работу со встроенной документацией — показывают ее во всплывающих подсказках, используют при автодополнении и т.д. Существуют разные технологии и инструменты для раобты с документацией, встроенной в исходный код программ на разных языках. Перечислим некоторые из них — известные и популярные.

Одним из первых инструментов, генерирующих документацию по комментариям в исходном коде, является *mkd*, разработанный в 1989 г. для UNIX-подобных систем [16] (фр.). Mkd поддерживал документирование программ

на разных языках — C/C++, Basic, Fortran, Pascal, Python и многих других. Инструмент извлекал много информации из структуры самой программы, при этом коментарии к сущностям программы можно было указывать в свободной форме.

Средство документирования API для Java — *Javadoc* — появилось в 1995 году вместе с языком Java [14, 17]. Javadoc поддерживает простое текстовое описание сущностей кода (с форматированием HTML), так и специальные директивы, например, относящие те или иные слова к определенному параметру метода, как в примере ниже на листинге 1.1.

```
/**
 * Create a new line attributes with the
 * specified line cap, join and width.
 *
 * @param width the line width
 * @param cap the line cap style
 * @param join the line join style
 */
public LineAttributes(float width, int cap, int join) {
        this(width, cap, join, SWT.LINE_SOLID,
             null, 0, 10);
}
```

Листинг 1: Пример комментария Javadoc

Поддержка JavaDoc интегрирована во многие IDE. При генерации справочников по умолчанию используется HTML, но возможно также генерировать PDF, RTF и некотрые другие форматы. Технология JavaDoc породила множество аналогичных для других языков — JavaScript, Scala, Kotlin и т.д.

Для языка Python был создан ряд инструментов для работы со встроенной документацией, основанных на использовании специального вида комментариев — *Python DocStrings*. Простейший из них — *PyDoc* [18] — позволяет документировать код с использованием простого языка разметки *epytext*.

Epytext почти не имеет средств форматирования, но поддреживает директивы, аналогичные директивам Javadoc. Также поддерживается работа с более продвинутым форматом, использующим язык разметки *reStructuredText*.

Аналогичные средства документирования есть и для языка Go — это система *GoDoc* [19]. GoDoc не предлагает явных правил для форматирования и директив в документации, а подразумевает лишь следование некоторым простым рекомендациям.

Интересным и развивающимся на протяжении уже более 20 лет проектом проектом является *Doxygen* [20]. Можно сказать, что Doxygen объединяет в себе возможности mkd и Javadoc: как mkd, он поддерживает встроенную документацию в исходном коде на разных языках программирования, и как Javadoc, позволяет выдавать документацию в различных конечных форматах. Формат комментариев Doxygen в целом аналогичен формату Javadoc.

В завершение обзора средств работы с документацией ПО отдадим должное интересной идее *Literate Programming* [21], предложенной проф. Д.Э. Кнутом в 1984 г. В противоположность документированию кода, Literate Programming предполагает встраивание работающих фрагментов кода в текст. Такой подход скорее ориентирован на научный мир, нежели на производственный, но тем не менее, для ряда языков программирования есть инструменты, его реализующие. Сейчас схожий подход используется в популярной сейчас ситеме Jupyter [22], позволяющей писать простые научные тексты и встраивать в них фрагменты кода на различных языках программирования.

В данной работе в качестве данных для анализа и апробации использована документация JavaDoc. Причиной такого выбора является то, что значительная часть доступной встроенной документации ПО с открытым исходным кодом использует именно этот формат. По данным GitHub, Inc., на 2019 г. Java является третьим по популярности языком программирования проектов, доступных на GitHub [23]. При этом многие открытые Java-проекты (такие, как, например, Eclipse) вышли из корпоративных экосистем, что обеспечило

им достаточно высокий уровень культуры документирования кода.

## 1.2 Векторные представления текста

Основная идея вычисления сходства текста заключается в определении векторов признаков документов и последующем расчете расстояния между этими признаками. Небольшое расстояние между признаками означает высокую степень сходства, в то время как большое расстояние означает низкую степень сходства.

Классической моделью текста является модель мешка слов (Bag of Words). Пусть $D$ множество документов в коллекции, $W$ словарь всех терминов, встретившихся в коллекции, $n_{dw}$ сколько раз слово $w$ встречается в документе $d$, $n_d$ число слов в документе $d \in D$. Каждый документ $d$ коллекции представляется в виде вектора длины $|W| : d = f_1^d, ..., f_{|W|}^d]^T$, где признак это частота использования слова в документе.

В последние годы широкое распространение получили различные векторные представления для слов, например word2vec [24], FastText[25] и другие. Эти модели получены путем обучения нейронной сети на большом корпусе данных, что позволяет отображать слова в векторное пространство небольшой размерности таким образом, что расстояние между ними тем меньше, чем ближе значения слов. Такой подход широко используется для классификации текста с использованием семантического сходства. Несмотря на то, что векторные представления слов, генерируемые нейронной сетью, хорошо отражают синтаксис и семантику, необходимо определить способ представления целого предложения в виде вектора. В 2018 году компания Google представила BERT[26] — предварительно обученную нейронную сеть, которая на данный момент занимает лидирующее место среди моделей векторного представления слов и показывает лучшие результаты при обработке естественного языка на таких задачах как GLUE[27], MultiMLI, а также во многих приклад-

ных приложениях, например, таких как определение настроения текста[28]. В статье[29] было показано, что модель BERT можно усовершенствовать, если увеличить время обучения, использовать больший набор данных и обучать на более длинных последовательностях. Авторы представили новую модель и назвали ее RoBERTa. Они показали, что данная модель превосходит BERT на ряде задач. Тем не менее, использование BERT и RoBERTa приводит к значительным вычислительным затратам. В 2019 году были представлены новые модели — Sentence-Bert и Sentence-RoBERTa[30], они представляют из себя модификацию предварительно обученных сетей BERT и RoBERTa соответственно. Для получения этих векторных представлений использовались сиамские и triplet нейронные сети для обновления весов модели таким образом, чтобы полученные векторные представления предложений можно было сравнивать с помощью косинусого сходства. Векторные представления, полученные с помощью SBert и SRoberta, позволяют искать и сравнивать предложения за значительно более короткое время, чем это делает Bert. Другая модель векторного представления предложений Google Sentence Encoder (USE)[31] так же была обучена и оптимизирована для текста, превышающего длину слова (предложения, фразы или короткие абзацы). Так как обучение происходило с использованием различных источников данных, это делает данную векторную модель применимой для широкого спектра задач по пониманию естественного языка[32].

## 1.3 Мера схожести векторов

После того, как фрагменты текста представлены в векторном виде необходимо выбрать функцию расстояния между ними. Есть много метрик для измерения сходства текста, например, косинусная мера, коэффициент подобия Жакара и коэффициент корреляции. Однако для данных в разреженном и многомерном пространстве, например при кластеризации документов, ко-

синусное сходство используется чаще. Это также часто используемый показатель сходства при информационном анализе текста и в информационном поиске[33].

Косинусное сходство измеряет косинус угла между векторами. Для двух векторов $x$ и $y$, каждый из которых имеет длину $n$, косинусное сходство вычисляется следующим образом:

$$cos(i, j) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

## 1.4 Алгоритмы кластеризации

Сравнительный анализ с использованием набора данных реального мира представлен в нескольких работах [34, 35]. В [34] сравнительный анализ методов кластеризации был выполнен в контексте задачи распознования речи, а в [35] использовали набор данных экспрессии генов рака.

В литературе было предложено много разных типов методов кластеризации. Несмотря на такое разнообразие, некоторые методы используются чаще [36]. Кроме того, многие из обычно используемых методов определяются в терминах схожих допущений относительно данных (например, k-means и k-medoids) или учитывают аналогичные математические концепции (например, матрицы подобия для спектральной или графовой кластеризации) и, следовательно, должны обеспечить аналогичную производительность в типичных сценариях использования. Поэтому в ходе работы будут рассмотрены алгоритмы кластеризации из разных семейств методов. Было предложено несколько таксономий для организации множества различных типов алгоритмов кластеризации в семейства. В то время как некоторые таксономии классифицируют алгоритмы на основе их целевых функций, другие нацелены на конкретные структуры, желаемые для полученных кластеров (например, иерархические). Наиболее известными классами алгоритмов кластеризации являются: алгоритмы, основанные на разбиении (например, k-means), иерар-

хические алгоритмы (алгомеративная кластеризация), алгоритмы кластеризации на основе плотности (dbscan).

**K-means++**

Алгоритм k-means[37] широко используется исследователями [36]. Помимо низких вычислительных затрат, алгоритм может обеспечить хорошие результаты во многих практических сценариях, таких как обнаружение аномалий [38] или сегментация изображений [39].

Получая в качестве входного значения количество кластеров, алгоритм k-means работает следующим образом:

- Выбирается $k$ точек в качестве начальных центроидов
- Формируются кластеры, путем объединения всех точек с ближайшим центроидом
- Пересчитываются центроиды для каждого кластера
- Алгоритм завершается, когда на какой-то итерации не происходит изменения внутрикластерного расстояния

У данного алгоритма есть ряд ограничений. Основным ограничением является необходимость задать необходимое количество кластеров. А так же то, что результат кластеризации может сильно зависеть от выбора количества кластеров и начальной инициализации центроидов. Для выбора наилучшего начального расположения центроидов был предложен алгоритм k-means++[40]. Данный алгоритм выбирает центроиды следующим образом:

- Первый центроид выбирается случайным образом
- Следующим выбранным центроидом является тот, который находится дальше всего от выбранного в настоящий момент
- Выбор продолжается до тех пор, пока не будут получены k центроидов

### Спектральная кластеризация

Спектральные алгоритмы [41, 42] основаны на методе уменьшения размерности при проецировании данных на собственные вектора матрицы попарных расстояний (матрицы подобия), соответствующие k наибольшим по модулю собственным значениям. После этого применяется алгоритм, схожий с k-means. Одним из недостатков спектрального метода является дорогостоящий процесс вычисления собственных векторов матрицы подобия.

Первым шагом алгоритма является построение матрицы подобия $A \in R^{N \times N}$, где значение в j-й строке и k-м столбце указывает на сходство между векторами j и k. Затем собственные значения и собственные вектора матрицы используются для разделения данных в соответствии с заданным критерием.

Рассмотрим работу алгоритмы более формально. При заданном наборе данных $x_1, ..., x_n$ и некотором понятнии сходства $s_{ij} \geq 0$ между всеми парами данных $x_i$ и $x_j$, цель кластеризации состоит в том, чтобы разделить данные на несколько групп, таким образом, чтобы точки в одной группе были похожими, а точки в разных группах не похожи друг на друга. Данные представляются в виде графа подобия $G = (V, E)$. Каждая вершина этого графа представляет точку данных $x_i$. Две вершины соединяются, если сходство между соответствующими точками $x_i$ и $x_j$ является положительным или превышает определенный порог. Таким образом, задача кластеризации может быть переформулирована следующим образом: необходимо найти такое разбиение графа, чтобы ребра между различными группами имели очень малые веса (это равносильно тому, что точки в разных кластерах отличны друг от друга), а ребра в одной группе имели большой вес (это равносильно тому, что точки в одном кластере похожи друг на друга).

### Affinity propagation

Алгоритм Affinity propagation[43] основан на идее "передачи сообщений"между объектами выборки и оценивании того, насколько один объект может слу-

жить центром или представителем для другого. На вход алгоритму подается матрица сходства между всеми объектами данных, в которой значение $s(i,k)$ показывает, насколько объект с индексом $k$ подходит для того, чтобы быть представителем объекта $i$. Диагональные значения матрицы $s(k,k)$, показывают априорную вероятность того, что данный объект должен быть выбран в качестве представителя. Есть два вида сообщений, которыми обмениваются объекты. Они задаются с помощью матриц ответственности и доступности. Ответственность (responsibility) $r(i,k)$ показывает насколько хорошо объект $k$ подходит для того, чтобы быть представителем объекта $i$, учитывая при этом других потенциальных представителей данного объекта. Доступность (availability) $a(i,k)$ показывает насколько правильным для объекта $i$ является выбор объекта $k$ в качестве своего представителя, учитывая при этом информацию о том, насколько данный объект считается подходящим представителем для других объектов. При инициализации обе эти матрицы содержат только нули, после чего происходит обмен значениями по следующим правилам:

$$r(i,k) \leftarrow s(i,k) - \max_{k',k' \neq k} \{a(i,k') + s(i,k')\}$$

$$a(i,k) \leftarrow \min \left\{ 0, r(k,k) - \sum_{i',i' \notin \{k,i\}} \max \{0, r(i',k)\} \right\}$$

$$a(k,k) \leftarrow \max \left\{ 0, \sum_{i',i' \neq k} r(i',k) \right\}$$

Алгоритм работает в три этапа:

- Обновление матрицы responsibility
- Обновление матрицы availability
- Рассмотрение доступных решений путем сочитания матриц responsibility и availability

- Алгоритм завершается,если матрицы перестали обновляться или достигнуто максимальное количество итераций.

**Агломеративная кластеризация**

Иерархические алгоритмы кластеризации были предложены в статье [44]. В результате работы алгоритма создается кластерная иерархия (дендрограмма). Дендрограмма позволяет отобразить набор объектов данных в дерево кластеров путем многоуровневого вложенного разбиения. Кластеризация объектов данных получается путем ограничения дендрограммы на нужном уровне, после чего все связанные компоненты образуют кластера. В иерархической кластеризации выделяют агломеративные и дивизионные методы. В алгоритме агломеративной иерархической кластеризации изначально каждый элемент принадлежит соответствующему отдельному кластеру. После чего происходит поэтапное объединение наиболее похожих элементов. Основой для работы этого метода кластеризации является матрица схожести. Для объединения кластеров необходимо выбрать правило вычисления расстояния между ними. Существуют различные методы вычисления расстояния:

- Метод ближайшего соседа (single-link) Сходство между двумя кластерами определяется как сходство между их наиболее похожими элементами (ближайшими соседями).

- Метод наиболее удаленного соседа (complete-link) Сходство между двумя кластерами определяется как сходство между их наиболее разными элементами.

- Попарное среднее (average-link) Сходство определяется как среднее расстояние между элементами в одном кластере и элементами в другом (т.е. все пары, состоящие из представителей разных кастеров).

Для данного метода кластеризации есть несколько вариантов реализации. Уровень построенной иерархии можно регулировать, если задать алгоритму

количество кластеров в которые нужно объеденить объекты, либо определить пороговое значение для метрик сходства между двумя кластерами выше которого кластеры не будут объединяться.

### DBSCAN

Идея алгоритма кластеризации DBSCAN[45] состоит в объединении в кластер плотно сгрупированных точек и отбрасывании "шумовых"точек, которые расположены в удалении от остальных. Как и в методе иерархической кластеризации здесь используется матрица схожести. Количество кластеров определяется автоматически. В алгоритме используются следующие понятия:

$\epsilon$ - максимальный радиус окрестности

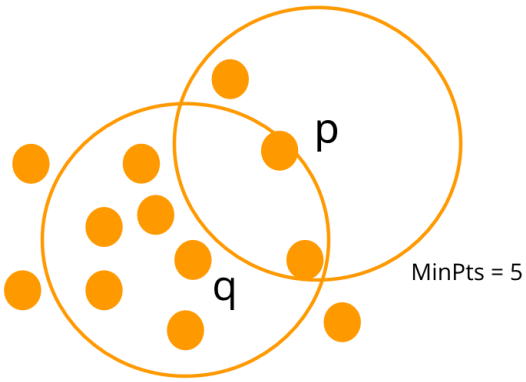
$MinPts$ - минимальное количество точек в $\epsilon$-окрестности точки

$\epsilon$-окрестность точки $q$: $N_{eps}(q) = \{p \in D : dist(p, q) \leq \epsilon\}$

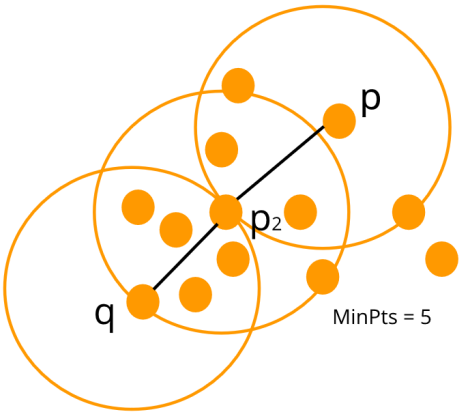
Точка называется корневой, если $|N_{eps}(q)| \geq MinPts$.

Говорят, что точка $p$ непосредственно плотно-достижима из точки $q$, если $p \in N_{eps}(q)$ и $q$ является корнывой точкой.

Говорят, что точка $p$ плотно-достижима из точки $q$, если существуют такие точки $p_1, ..., p_n$, где $p_1 = q$, $p_n = p$, что точка $p_{i+1}$ непосредственно плотно-достижима из $p_i$ при всех $i = 1, ..., n - 1$.




a) Точка $p$ непосредственно плотно-достижим из точки $q$




b) Точка $p$ плотно-достижима из точки $q$

Говорят, что точка $p$ плотно-связана с точкой $q$, если существует точка

$o$, такая, что и $p$, и $q$ плотно-достижимы из $o$.

Алгоритм DBSCAN работает следующим образом:

- Произвольно выбирается точка $p$
- Выбираются все точки, которые являются плотно-достижимыми из $p$
- Если выбранная точка $p$ является корневой, то образуется кластер
- Если выбранная точка $p$ является граничной точкой (то есть она находится в кластере, но не является корневой) и нет плотно-достижимых из $p$ точек, то переходим к следующей точке
- Алгоритм завершается, когда все точки просмотрены.

## 1.5 Методы оценки качества кластеризации

Задача оценки качества кластеризации непроста, поскольку результат кластеризации, оптимальный с точки зрения запрограммированной модели, может оказаться совершенно неприемлемым с точки зрения экспертов предметной области.

Существуют внутренние и внешние метрики для оценки качества работы алгоритмов кластеризации [46]. Внутренние оценки качества позволяют сравнивать результаты кластеризации в отсутствие дополнительной информации об истинных классах объектов, т.е. исходя только из имеющихся данных об объектах выборки и построенной кластеризации. Внешние оценки качества опираются на информацию об истинных классах объектов. Предполагается, что эта информация не является вычисляемой из самих данных, ее получают посредством ручной классификации.

На практике алгоритмы кластеризации используются для автоматизации процесса поиска. Несмотря на то, что люди могут осуществлять поиск достаточно хорошо, этот процесс происходит медленно по сравнению со

скоростью компьютера. В случае кластеризации текста эксперты предметной области опираются на свои специализированные знания и на понимание естественного языка, поэтому наилучшая мера качества будет основана на субъективных суждениях человека. Один из способов добиться этого - попросить людей судить о качестве результатов напрямую. Однако это дорогостоящий и трудоемкий процесс, и для каждого опробованного алгоритма (или варианта одного алгоритма) потребуется новый набор субъективных суждений. Альтернативой этому способ заключается в том, чтобы один раз попросить людей сгруппировать набор данных в то, что они считают правильным набором кластеров. Идея состоит в том, что предполагаемые пользователи алгоритма будут ожидать, что алгоритм разделит данные на такие же кластера. Таким образом, разбиение, полученное с помощью экспертов предметной области, рассматривается как идеальная кластеризация (эталонное разбиение), а качество оценивается на основании некоторой степени того, насколько метки кластеров, созданные различными алгоритмами, соответствуют эталону.

Существуют различные внешние метрики для оценивания кластеризации[47], они основаны на следующих идеях: подсчет пар элементов, сопоставление множеств, а также использование информационно-теоретической взаимной информации.

### Метрики, основанные на подсчёте пар

Рассмотрим метрики, основанные на подсчете пар. Возможны 4 варианта распределения данных:

- TP (true positive) - количество пар, которые в обоих разбиениях принадлежат одному и тому же кластеру

$$TP = |(x_i, x_j) : y_i = y_j, \widetilde{y}_i = \widetilde{y}_j|$$

где $y_i$ это истинный номер кластера, к которому принадлежит элемент из пары и $\widetilde{y}_i$ это номер кластера, который был получен в результате

работы алгоритма.

- FP (false positive)- количество пар, которые лежат в разных эталонных кластерах, но распознаны как дубликаты

$$FP = |(x_i, x_j) : y_i \neq y_j, \widetilde{y}_i = \widetilde{y}_j|$$

- FN (false negative)- количество пар, которые являются дубликатами, но не распознаны как таковые

$$FN = |(x_i, x_j) : y_i = y_j, \widetilde{y}_i \neq \widetilde{y}_j|$$

- TN (true negative)- количество пар, которые в обоих разбиениях принадлежат разным кластерам

$$TN = |(x_i, x_j) : y_i \neq y_j, \widetilde{y}_i \neq \widetilde{y}_j|$$

Рассмотрим метрики, основанные на сопоставлении пар, используя введенные выше обозначения:

- Точность (Precision)

$$\frac{TP}{TP + FP}$$

- Полнота (Recall)

$$\frac{TP}{TP + FN}$$

- F-мера

$$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

- Rand Index [48]

  Rand Index отражает близость друг к другу похожих текстов. Он оценивает насколько много из тех пар элементов, которые в эталоне находились в одном кластере, и тех пар элементов, которые находились в

разных кластерах (не привязываясь к меткам кластеров), сохранили это состояние после кластеризации алгоритмом.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- Fowlkes and Mallows (FM) [49]

$$\sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}}$$

**Метрики, основанные на сопоставлении множеств**

Этот класс метрик основывается на предположении о существовании однозначного соответствия между эталоном и полученным разбиением. Для определения метрик введем понятия точности и полноты соответсвия между алгоритмическими кластерами и эталонными, а так же их гармоническое среднее F-меру. Пусть $N$ - количество объектов, $C = \{c_1, c_2, ..., c_k\}$ - кластера, найденные алгоритмом, $L = \{l_1, l_2, ..., l_j\}$ - кластера в эталонном разбиении,

$$Precision(C_i, L_j) = \frac{\mid C_i \cap L_j \mid}{\mid C_i \mid}$$

$$Recall(C_i, L_j) = \frac{\mid C_i \cap L_j \mid}{\mid L_i \mid}$$

$$F_{measure}(C_i, L_j) = \frac{2 \times Precision(C_i, L_j) \times Recall(C_i, L_j)}{Precision(C_i, L_j) + Recall(C_i, L_j)}$$

- Purity [50, 33]

  При вычислении этой метрики сначала каждый полученный алгоритмический кластер сопоставляется эталонному, с которым он имеет наибольшее перекрытие. После этого вычисляется правильность данного сопоставления. Алгоритмический кластер считается чистым, если он содержит объекты из одного и только одного эталонного кластера. И

наоборот, кластер считается нечистым, если он содержит объекты из разных эталонных кастеров.

$$Purity = \sum_i \frac{\mid C_i \mid}{N} \max_j Precision(C_i, L_j)$$

- Van Rijsbergen's F-measure [51, 52]

$$F = \sum_i \frac{\mid L_i \mid}{N} \max_j F(L_i, C_j)$$

**Метрики, использующие информационно-теоретическую взаимную информацию**

Этот подход к сравнению кластеров берет свое начало в теории информации и основан на понятии энтропии. Энтропия кластера [53, 52] отражает то, как элементы распределяются внутри каждого кластера, энтропия кластеризации вычисляется путем усреднения энтропии всех кластеров:

$$Entropy = -\sum_j \frac{n_j}{n} \sum_i P(i,j) \times \log_2 P(i,j),$$

где $P(i,j) = \frac{n_{ij}}{n_j}$ - вероятность найти элемент из эталонного кластера $i$ в алгоритмическом кластере $j$ (число объектов в кластере $j$, принадлежащих эталону $i$, по отношению к общему числу элементов в кластере $j$), $n_{ij}$ - число элементов в кластере $j$, принадлежащих эталонному кластеру, $i$, $n_j$ - количество элементов в кластере $j$, $n$ - общее количество элементов.

Хаотичное разбиение (то есть отсутствие кластеров) соответствует максимальной величине энтропии, хорошее кластерное решение соответствует минимуму энтропии. Недостатком приведённой метрики является то, что наибольшие значения будут достигаться при числе кластеров, равном числу элементов.

Другой метрикой, основанной на понятии энтропии является Mutual Information (MI) [54]. MI измеряет, сколько информации одна случайная переменная может рассказать о другой.

### Метрики BCubed

Метрики BCubed Precision и Recall были предложены в статье [55]. Эта группа метрик предлагает взглянуть на кластеризацию с точки зрения одного отдельного элемента. Точность элемента показывает, сколько элементов в одном алгоритмическом кластере находится так же и в одном эталонном. Полнота элемента показывает сколько элементов из такого же эталонного кластера находятся в алгоритмиическом кластере.

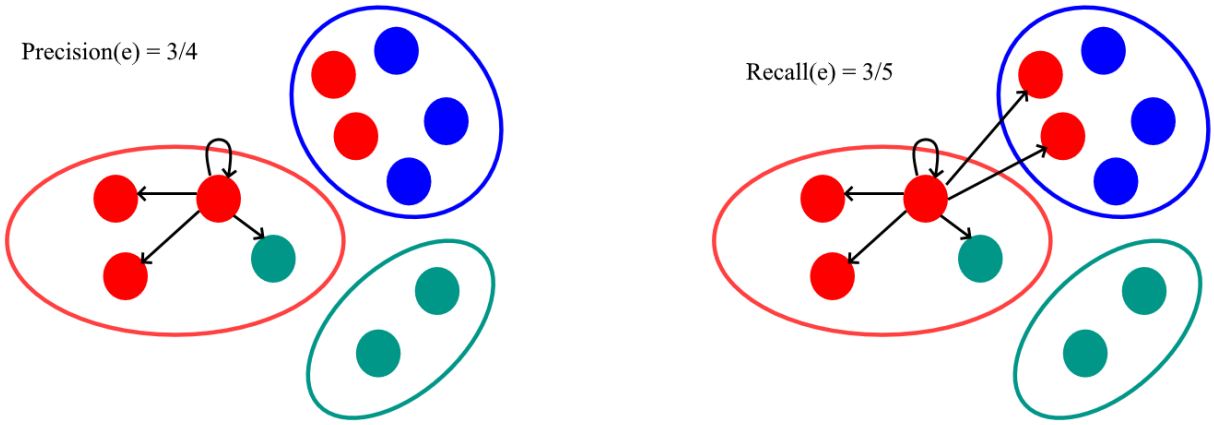



Рис. 1: Пример вычисления точности и полноты
для одного элемента кластера

С точки зрения пользователя BCubed показывает эффективность системы кластеризации, когда после обращения к одному элементу пользователь может изучить остальные элементы в этом кластере. Если элемент имеет высокое значение полноты BCubed, то пользователь найдет большинство связанных с ним элементов, не покидая кластер. Если элемент имеет высокую точность BCubed, то пользователь не будет находить в этом кластере элементов из других кластеров.

Точность и полнота для всей кластеризации определяется путем усреднения по элементам:

$$Precision(PBC) = \frac{1}{N} \sum_{elem} Precision(elem)$$

$$Recall(RBC) = \frac{1}{N} \sum_{elem} Recall(elem)$$

Среднее гармоническое точности и полноты:

$$F(BCubed) = \frac{2 \times PBC \times RBC}{PBC + RBC}$$

**Внутренние метрики оценки качества**

Внутренние метрики позволяют оценить качество кластеризации, основываясь только на наборе данных. Оптимальное число кластеров обычно определяют с использованием внутренних метрик.

- Коэффициент силуэта

В статье [56] было проведено сравнение большого количества внутренних метрик оценки качества и показано, что метрика Silhouette (коэффициент силуэта) является одной из метрик, показавших наилучшие результаты и при этом превосходство этой метрики статичтически значимо (тест Шаффера с уровнем значимости 10%)

Эта метрика показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров. Сначала силуэт определяется отдельно для каждого элемента. a – среднее расстояние от данного элемента до элементов из того же кластера, b – среднее расстояние от данного элемента до элементов из ближайшего кластера (отличного от того, в котором лежит сам элемент). Тогда силуэтом данного элемента называется величина:

$$S = \frac{b - a}{\max(a, b)}$$

Силуэтом выборки называется средняя величина силуэта объектов данной выборки. Данная величина лежит в диапазоне [-1,1]. Значения, близкие к -1, соответствуют варианту кластеризации с высокой дисперсией, значения,

близкие к нулю, говорят о том, что кластеры пересекаются и накладываются друг на друга, значения, близкие к 1, соответствуют "плотным"четко выделенным кластерам. Таким образом, чем больше силуэт, тем более четко выделены кластеры, и они представляют собой компактные, плотно сгруппированные облака точек.

## 1.6 Свойства метрик

В статье [57] разбирается ряд принципов, их корректность и то, как различные классы внешних метрик удовлетворяют этим принципам. Примеры свойств, приведенные на рисунках 2, 3, 4, 5 взяты из этой статьи.

- Однородность кластеров

  Значение метрики качества должно уменьшаться при объединении в один кластер двух эталонных.

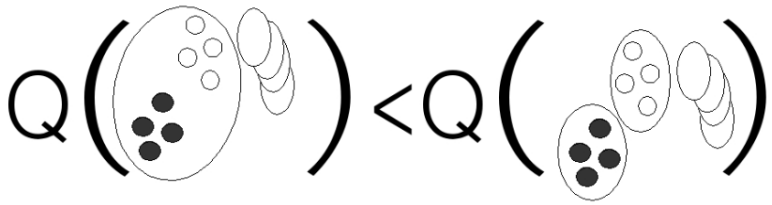



Рис. 2: Однородность кластеров

- Полнота кластеров

  Значение метрики качества должно уменьшаться при разделении эталонного кластера на части.

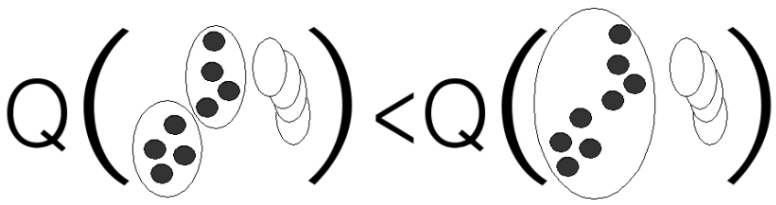



Рис. 3: Полнота кластеров

- Rag Bag

  Поместить новый нерелевантный обоим кластерам элемент в разупорядоченный кластер менее вредно, чем поместить этот элемент в чистый кластер.

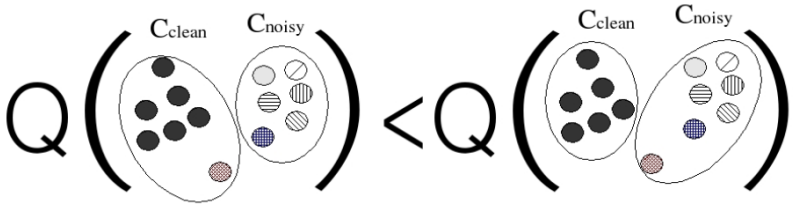



Рис. 4: Rag Bag

- Size vs Quantity

  Значительное ухудшение кластеризации большого числа небольших кластеров должно обходиться дороже небольшого ухудшения кластеризации в крупном кластере.

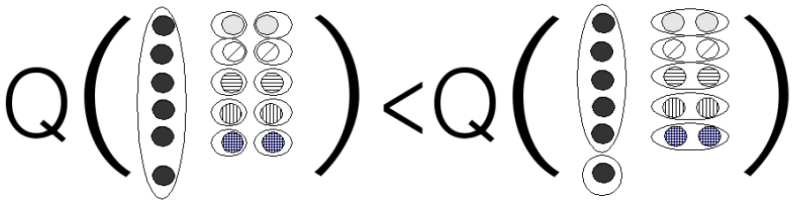



Рис. 5: Size vs Quantity

# 2 Вопросы исследования и методика проведения экспериментов

## 2.1 Вопросы исследования

В работе были рассмотрены следующие вопросы исследования:

- **Вопрос исследования 1** (векторные представления): *Какой способ векторного представления позволяет лучше всего находить неточные повторы в Javadoc документации?* Важной метрикой при нахождении дубликатов является семантическое сходство фрагментов текста. Существуют разные подходы для вычисления сходства между двумя документами, они используют лексическое сопоставление, лингвистический анализ или семантические функции. Повторы в Javadoc документации представляют собой короткие текстовые фрагменты, поэтому сравнение их с помощью лексического сопоставления может показать хорошие результаты. Но, возможно, это не самый эффективный метод. Для этого были рассмотрены различные методы векторного представления текста и проведено сравнение полученных результатов с помощью F-меры.

- **Вопрос исследования 2** (выбор метрики оценки качества кластеризации): *Какие из метрик оценки качества целесообразно применять для сравнения качества алгоритмов кластеризации?* В случае поиска повторов в документации программного обеспечения не все текстовые фрагменты являются дубликатами, поэтому при кластеризации образуется большое количество одноэлементных кластеров. Еще одной особенностью данной задачи кластеризации является то, что большинство кластеров, не являющихся одноэлементными, состоят из малого количества элементов (2-3 элемента на кластер). Это вносит существенные ограничения на использование большинства общепринятых внешних

метрик оценки качества кластеризации.

- **Вопрос исследования 3** (алгоритм кластеризации): *Какой из алгоритмов кластеризации наиболее применим к задаче поиска неточных повторов?* Необходимо дать оценку апробированным алгоритмам кластеризации, используя выбранные метики, а так же оценить алгоритмы по производительности и удобству использования на практике.

## 2.2 Выбор инструментальных средств

Для реализации и проведения экспериментов был выбран язык Python, популярный при решении задач в областях data science и анализа текстов на естественных языках. Благодаря наличию большого количества различных фреймворков и библиотек (pandas, scikit-learn, Tensorflow и другие) Python удобен для применения в задачах анализа данных. Разработка модели эксперимента проводилась в Jupyter Notebook. Формат файлов-блокнотов `.ipynb` соответствует принципу Literate Programming: позволяет хранить в одном файле программный код, комментарии, графики и изображения, что заметно облегчает анализ полученных результатов. При этом так же доступна возможность обращаться к коду во внешних модулях Python (например, в других `.py`-файлах).

## 2.3 Методика проведения экспериментов

В рамках выпускной квалификационной работы магистра целью проведения эксперимента является выбор алгоритма кластеризации, лучше всего подходящего для задачи поиска неточных повторов в документации программного обеспечения. Люди могут осуществлять поиск повторов достаточно хорошо, при этом они опираются на свои знания предметной области и на понимание естественного языка, но это трудоемкая задача, требующая

большого количества времени. Задача алгоритмов кластеризации автоматизировать процесс поиска и разбить данные на кластера таким же образом, как это сделал бы человек. Методика эксперимента представлена на рисунке 6.

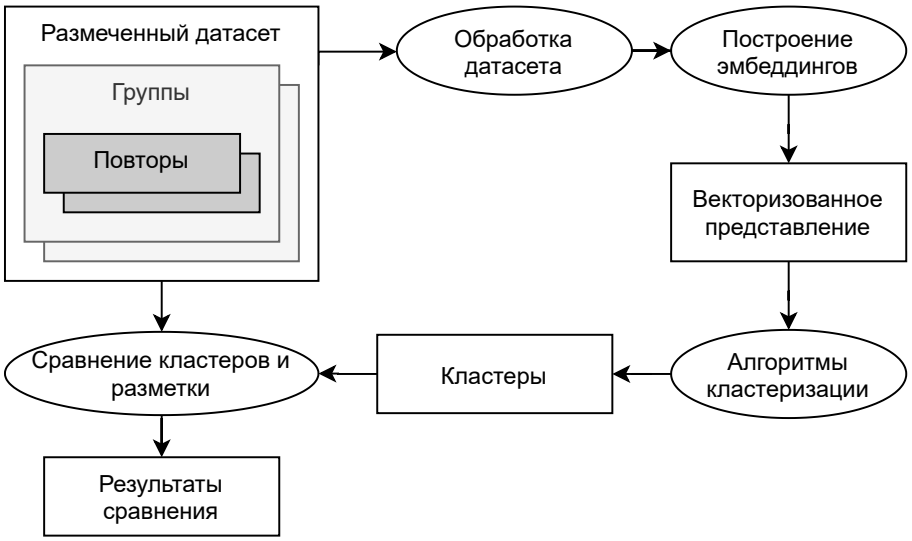



Рис. 6: Структура экспериментов

Модель экспериментов по оценке качества кластеризации включает в себя следующие этапы:

1. Выбор документа для анализа.
2. Предобработка документа – перевод слов в нижний регистр, удаление пунктуации, лемматизация и удаление стоп слов.
3. Получение векторного представления для кажого текстового фрагмента.
4. Запуск алгоритма кластеризации.
5. Оценка алгоритма путем сравнения результатов кластеризации с эталонным разбиением.

## 2.4 Программная среда

Для апробации моделей векторного представления и алгоритмов кластризации была создана программная среда, реализующая описанную выше методику.

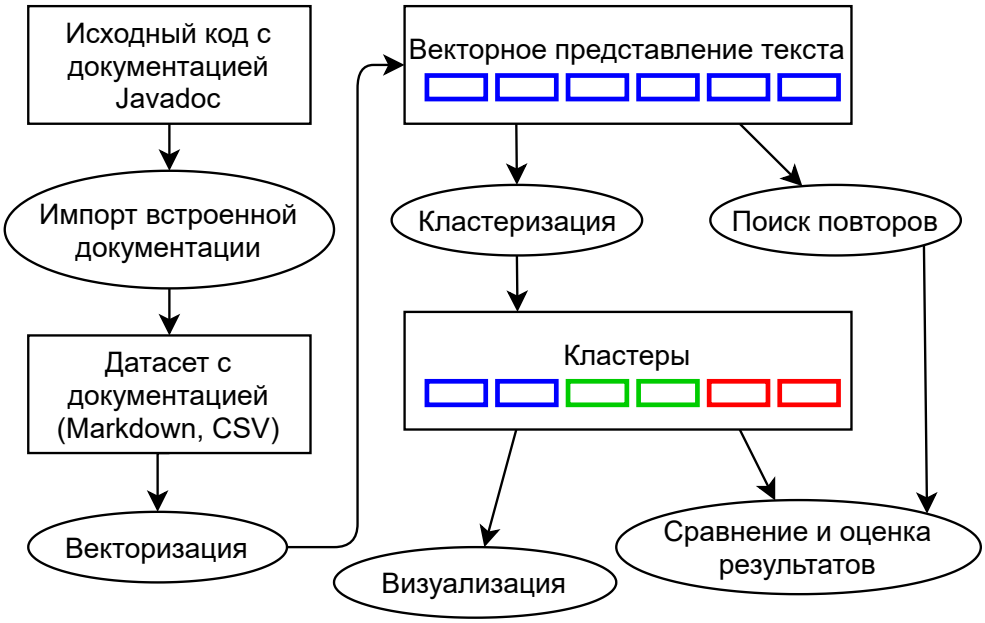



Рис. 7: Программная среда

Исходный Javadoc файл преобразуется в формат .csv, где каждый текстовый фрагмент представлен отдельной строкой. После этого файл обрабатывается в одно из векторных представлений (BoW, USE, S-Bert, S-Roberta) и осуществляется поиск повторов по косинусному сходству, либо кластеризация их на группы, с последующей оценкой результатов работы алгоритмов. Для алгоритмов кластеризации, в которых перед началом работы необходимо задать количество кластеров $k$, возможно построение графика, показывающего изменение значения коэффициента силуэта при различном значении $k$. Это помогает пользователю определиться с выбором и добиться лучшего результата кластеризации. Визуализация с использованием инструмента Duplicate Finder [5] помогает наглядно оценить разбиения, полученные при кластеризации и проанализировать возможные причины возникновения ошибок в работе алгоритмов. Данную программную среду так же можно использовать для поиска неточных повторов в неразмеченных данных.

# 3 Векторные представления повторов

Для сравнения векторных представлений текста использовался эталонный датасет, состоящий из документов Slf4J, Mockito, GSON, JUnit. Общее количество текстовых фрагментов в документах, а также количество повторов в них представлены в Таблице 1. Задачей алгоритма было найти все фрагменты текста, которые похожи между собой. Для каждого документа было известно какие фрагменты текста действительно являются повторами. Разметка документов осуществлялась экспертами предметной области. Для определения схожести двух фрагментов текста использовалось косинусное сходство. В рамках текущего исследования были протестированы следующие векторные представления: BoW, USE, SBert, SRoberta.

| | Slf4j | Mockito | GSON | JUnit |
|---|---|---|---|---|
| Общее количество текстовых фрагментов | 225 | 594 | 442 | 223 |
| Количество повторов | 133 | 134 | 71 | 223 |

Таблица 1: Документы эталонной коллекции

Метрикой оценки качества была выбрана F-мера [33], которая является средним гармоническим между точностью (precision) и полнотой (recall).

$$F_{measure} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Для поиска дубликатов использовался стандартный подход с определением пороговой функции подобия [58]. То есть, если значение косинусной метрики между двумя фрагментами текста превышает пороговое значение, то эти фрагменты определяются как повторы.

Порог $h$, в результате которого алгоритм принимает решение, выбирался следующим образом:

$$F_{measure} = arg \max_{h \in [0.5:1]} [F(h)]$$

Результаты аппробации представлены в Таблице 2.

| | Slf4j | Mockito | GSON | JUnit |
|---|---|---|---|---|
| BoW | 0.892 | 0.880 | 0.901 | 0.881 |
| USE | 0.887 | 0.878 | 0.895 | 0.877 |
| S-Bert | 0.836 | 0.742 | 0.885 | 0.871 |
| S-Roberta | 0.885 | 0.878 | 0.897 | 0.879 |

Таблица 2: Результаты апробации: F-мера

Проанализируем полученные значения. Как видно из Таблицы 2, значение F-меры для векторного представления BoW является высоким и превосходит значения F-меры для других векторных представлений. Вероятная причина этого закючается в том, что повторы в Javadoc имеют небольшой объем и насыщены специальной терминологией, в связи с этим модели, обученные на других корпусах, могут не содержать в словаре необходимых слов.

Рассмотрим применение векторных представлений на примерах фрагментов текста из различных документаций программного обеспечения. Пример 3 взят из статьи Soto [59] об использовании различных метрик для поиска фрагментов переиспользования в технической документации.

| Фрагмент 1 | Фрагмент 2 |
|---|---|
| Adding work packages to an iteration. Iterations can group work packages with the same due dates. At least one iteration must exist. Group all work packages due for the same milestone into one iteration. In My Products, expand the desired product. Click the desired release. Click Iterations. Select the box around the iteration to which the work package belongs. Work packages that are not yet added to an iteration are shown in Unscheduled. All work packages in the selected iteration appear. Drag and drop the work package into the desired iteration. The work package is now part of the selected iteration. | Adding an iteration. Iterations can group your work packages by due dates. Know the due date for your iteration. Use iterations to easily track a group of work packages with similar due dates. In My Products, expand the desired product. Click the desired release. Click Iterations. Click Create new iteration. Enter a name and description for the iteration. Set the start and end dates. Click Save. The iteration is set and ready for work packages. If you need to edit an iteration, click the iteration name. The iteration details window opens for editing. |

Таблица 3: Пример фрагментов текста из работы [59]

Это два фрагмента текста из DITA-документации. Авторы статьи попросили оценить экспертов предметной области и технических писателей данные фрагменты на предмет возможного переиспользования. Трое из четырех экспертов подтвердили, что данные фрагменты передают одну и туже идею, но с несколько иной формулировкой. Применим различные методы векторного представления и рассчитаем косинусное расстояние между этими фрагмента-

ми. Результат представлен в Таблице 4. Как видно из данного примера, для DITA-документации предобученные модели S-Bert и S-Roberta показывают результат лучше, чем BoW. Это связано с тем, что количество общих слов в двух фрагментах текста невелико. При этом, даже не смотря на наличие технических терминов, эти нейронные сети смогли распознать семантическое сходство фрагментов.

| | BoW | USE | S-Bert | S-Roberta |
|---|---|---|---|---|
| cosine sim | 0.771 | 0.711 | 0.901 | 0.881 |

Таблица 4: Результаты апробации на DITA-документации

Следующий пример взят из UNIX Man Pages — документации по системным инструментам и программным библиотекам UNIX-подобных систем. Подробнее о том, как эта документация форматируется, можно прочитать в обзоре работы [8].

| Фрагмент 1 | Фрагмент 2 |
|---|---|
| Causes the file information and file type evaluated for each symbolic link to be those of the file referenced by the link, and not the link itself. See NOTES. | when showing file information for a symbolic link, show information for the file the link references rather than for the link itself. |

Таблица 5: Пример фрагментов текста из UNIX Man Pages

Несмотря на то, что инструменты (`ls` и `find`), к которым относятся фрагменты докментации, формально не зависят друг от друга (и их документация тоже составлялась независимо), они являются частью общей UNIX-экосистемы. Поэтому одинаковые опции командной строки для этих инструментов часто имеют одинаковый смысл. В итоге для обоих инструментов в нашем примере по-разному сформулировано описание опции `-L`, но смысл в обоих случаях один: опция предписывает при обходе файловой системы сле-

дует разыменовывать символические ссылки и анализировать сущности, на которые они указывают, а не ссылки сами по себе. Данный пример показывает, что векторные представления S-Bert и S-Roberta определяют сходство между фрагментами лучше методов BoW и USE.

| | BoW | USE | S-Bert | S-Roberta |
|---|---|---|---|---|
| cosine sim | 0.699 | 0.721 | 0.854 | 0.827 |

Таблица 6: Результаты апробации для UNIX Man Pages

Рассмотрим пример неточного повтора из Javadoc документации.

| Фрагмент 1 | Фрагмент 2 |
|---|---|
| Adds to errors if this method: is not public, or takes parameters, or returns something other than void, or is static (given isStatic is false), or is not static (given isStatic is true). | Adds to errors if any method in this class is annotated with annotation, but: is not public, or takes parameters, or returns something other than void, or is static (given isStatic is false), or is not static (given isStatic is true) |

Таблица 7: Пример фрагментов текста из jave-doc

| | BoW | USE | S-Bert | S-Roberta |
|---|---|---|---|---|
| cosine sim | 0.957 | 0.896 | 0.915 | 0.910 |

Таблица 8: Результаты апробации для Javadoc

Большинство повторов в рассматриваемой Javadoc документации имеют подобный вид. Из-за того, что лишь малая часть повтора является вариативной, BoW хорошо определяет неточный повтор.

Основываясь на полученных результатах, для кластеризации неточных повторов в Javadoc документации было выбрано векторное представление текста в виде BoW.

# 4 Метрики оценки качества кластеризации

Выбор метрики оценки качества работы алгоритмов кластеризации зависит от поставленной задачи и набора исследуемых данных. Особенностью задачи кластеризации неточных повторов в документации программного обеспечения является наличие большого количества одноэлементных кластеров, так как не все текстовые фрагменты явяются дубликатами, а так же в целом небольшой размер большинства групп (3-4 элемента).

Рассмотрим возможность применения метрик, основанных на подсчете пар. Из-за наличия кластеров, состоящих из одного элемента, при подсчте метрик этого класса TN(True Negative) подавляет влияние TP, FP и FN. Для того, чтобы снизить роль TN предполагается учитывать не все фрагменты текста, а только те, которые хотя бы в одном разбиении являются чьими-то дубликатами.

*Пример 0.* Одинаковые разбиения:

$$[\mathrm{S}_1, S_2], S_3, S_4, S_5, ..., S_n$$
$$[\mathrm{S}_1, S_2], S_3, S_4, S_5, ..., S_n$$

Тогда TP = 1, FP = FN = TN = 0, P = R = F-measure = 1

*Пример 1.* Разные разбиения:

$$[\mathrm{S}_1, S_2], S_3, S_4, S_5, ..., S_n$$
$$[\mathrm{S}_1, S_3], S_2, S_4, S_5, ..., S_n$$

Тогда TP = 0, FP = FN = TN = 1, P = R = F-measure = 0

*Пример 2.*

$$[\mathrm{S}_1, S_2], S_3, [S_4, S_5], ..., S_n$$
$$[\mathrm{S}_1, S_3], S_2, [S_4, S_5], ..., S_n$$

Тогда TP = FP = FN = 1, TN = 7 , P = R = F-measure = 0.5

Рассмотрим возможность применения метрик, основанных на сопоставлении множеств. Для этой группы метрик в работах [53, 60, 57] была выделена следующая проблема: поскольку каждый алгоритмический кластер оценивается лишь тем эталонным кластером, которому принадлежит больше объектов, изменения в других кластерах не будут обнаружены. Кроме того, значение метрики *Purity* будет высоким, если каждый объект находится в еденичном кластере. А так же один и тот же алгоритмический кластер может быть сопоставлен нескольким эталонным. Из-за данных ограничений использование метрик этого класса в рассматриваемой задаче не оправдано.

Рассмотрим выполнение свойств однородности, полноты, Rag Bag и Size vs Quantity для метрик. В таблице 9 обозначение $F-$ означает F-меру, основанную на подсчете пар. При расчете метрик, основанных на подсчете пар, учитываются только элементы, которые хотя бы в одном разбиении являются дубликатами.

| | Однородность | Полнота | Rang Bag | Size vs Quantity |
|---|---|---|---|---|
| F-мера | ✓ | ✓ | × | × |
| Rand | ✓ | ✓ | × | × |
| FM | ✓ | ✓ | × | × |
| Entropy | ✓ | × | × | × |
| MI | ✓ | × | × | × |
| FBCubed | ✓ | ✓ | ✓ | ✓ |

Таблица 9: Свойства внешних метрик

В задаче кластеризации повторов в документации программного обеспечения большинство кластеров содержат малое количество элементов, таким образом, особенно важным является выполненение свойства Size vs Quantity. Как видно из таблицы 9 данное свойство выполняется только для метрики FBCubed. Таким образом, для оценки качества кластеризации была выбрана FBCubed метрика.

# 5 Алгоритмы кластеризации повторов

Кластеризация данных является важной задачей в машинном обучении, она помогает находить естественные группировки в данных. Обычно в кластеризации нет единого идеального решения проблемы, но алгоритмы стремятся минимизировать определенный математический критерий (который варьируется между алгоритмами). При этом результат работы алгоритмов является неоднозначным: по одному и тому же набору данных могут быть построены разные варианты разбиений, которые не всегда будут удовлетворять требованиям эксперта. В этом случае становится актуальным процесс взаимодействия пользователя и программной среды, который бы позволял добиться желаемых результатов. Для сравнения были выбраны алгоритмы kmeans++, спектральный, affinity propagation, агломеративный с указанием количества кластеров и с указанием порогового значения, DBSCAN.

Для алгоритмов кластеризации, которые требуют указания количества кластеров перед началом своей работы (k-means, спектральная, агломеративная), реализована возможность оценки экспертом результатов работы алгоритма при различных параметрах, с последующим выбором оптимального значения. Для осуществления этого процесса строится график, показывающий значение коэффициента силуэта при разном заданном количестве кластеров.

Для тестирования алгоритмов кластеризации использовался датасет, состоящий из документов Slf4J, Mockito, GSON, JUnit. Разметка документов осуществлялась экспертами предметной области. Каждому фрагменту текста присвоен уникальный номер группы, в котором он находится. Фрагментам, которые не являются повторами, присвоен индивидуальный отрицательный номер группы, таким образом каждый из них образует одноэлементный кластер. На рисунке 10 представлено описание используемого датасета.

| | Slf4j | Mockito | GSON | JUnit |
|---|---|---|---|---|
| Количество символов | 97421 | 290177 | 174228 | 207454 |
| Количество слов | 15574 | 40651 | 25639 | 29287 |
| Общее количество текстовых фрагментов | 225 | 594 | 442 | 223 |
| Количество повторов | 133 | 134 | 71 | 223 |
| Количество групп | 22 | 29 | 18 | 78 |

Таблица 10: Документы эталонной коллекции

В таблице 11 приведены усредненные максимальные значения метрики качества FBCubed для имеющегося набора данных. Эти данные были получены путем перебора параметра $k$ (количества кластеров), а также подбором оптимальных параметров для каждого алгоритма. Для проведения эксперимента использовались спектральный, алгомеративный и k-means алгоритмы, так как они позволяют задавать количество кластеров. Данные значения можно считать потенциальным максимумом для данных алгоритмов.

| | kmeans++ | Спектральная | Агломеративная |
|---|---|---|---|
| FBCubed | 0.88 | 0.86 | 0.91 |

Таблица 11: Максимальное значение FBCubed

Так как в реальной задаче для документов не существует информации об истинном разбиении повторов на группы, то возникают трудности с подбором необходимых параметров. Одним из подходов для решения этих проблем является оптимизация внутренней метрики качества (коэффициента Silhouette). В таблице 12 приведены усредненные значения метрики FBCubed, полученные при подборе параметров с помощью оптимизации коэффициента силуэта.

| | kmeans++ | Спектральная | Агломеративная (k) |
|---|---|---|---|
| FBCubed | 0.86 | 0.85 | 0.88 |

Таблица 12: FBCubed при подборе параметров с помощью коэффициента силуэта

Алгоритмы Affinity Propagation, агломеративный с указанием порогового значения и DBSCAN определяют количество кластеров автоматически. Подбор оптимальных параметров для этих алгоритмов осуществлялся с помощью поиска максимального значения метрики FBCubed. Для Affinity propagation такими параметрами являются $preference$, значение на диагонали матрицы $S$, и $damping$, коэффициент затухания, введеный во избежание численных колебаний при обмене сообщениями. Для агломеративного алгоритма настраиваемым параметром является пороговое значение, для алгоритма DBSCAN $eps$ и $minPts$. В таблице 13 представлено среднее максимальное значение метрики FBCubed и среднее значение коэффициента силуэта при кластеризации.

| | Affinity | Агломеративная (порог) | DBSCAN |
|---|---|---|---|
| FBCubed | 0.91 | 0.92 | 0.90 |
| Силуэт | 0.31 | 0.37 | 0.37 |

Таблица 13: FBCubed при подборе параметров с помощью коэффициента силуэта

Приведенное сравнение демонстрирует превосходство алгоритмов, которые при запуске не трубуют указания количества кластеров. При этом удобство их использования так же значительно выше, так как нет необходимости несколько раз запускать алгоритмы для поиска лучшего разбиения. Лучшим алгоритмом для задачи оказался агломеративный алгоритм, использующий для группировки текстовых фрагментов пороговое значение. В ходе экспериментов было установлено, что для лучшей кластеризации косинусное сход-

ство между двумя векторами должно быть больше 0.8 (косинусное расстояние в таком случае будет 0.2), а в качестве метода вычисления растояния между кластерами должен быть использован метод попарного среднего.

Исследование полученных разбиений повторов показало, что большинство из повторов, которые были неправильно кластеризованы алгоритмами являются неоднозначными и их сложно отнести к какому-то одному классу. Пример подобной неоднозначности для фрагментов текста из документа Mockito представлен в Приложении. В связи с этим, несмотря на то, что полученные разбиения удовлетворяют внешней метрике качества, в последующих работах предполагается рассмотреть алгоритмы нечеткой класстеризации, в которых одному объекту может быть назначено несколько кластеров.

Для алгоритмов, показавших наибольшее значение метрики FBCubed (Afinity Propagation, агломеративная кластеризация по пороговому значению и DBSCAN) была проведена оценка времени их работы на документах проекта Apache Commons Collections и проекта Eclipse SWT. Данные документы не являются размеченными, они были выбраны, так как по объему превышают имеющиеся в эталонной коллекции. В таблице 14 представлено описание документов, а в таблице 15 представлено время работы (в секундах) алгоритмов кластеризации. Конфигурация вычислительного устройства: Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.5GHz 8GB RAM (свободно 7.67GB).

| | Apach | SWT |
|---|---|---|
| Количество символов | 1520997 | 342807 |
| Количество слов | 198035 | 53030 |
| Общее количество текстовых фрагментов | 3648 | 1454 |

Таблица 14: Документы для тестирования времени работы алгоритмов

| | Apach | SWT |
|---|---|---|
| Affinity | 905.3 | 7.1 |
| Агломеративная | 30.2 | 3.7 |
| DBSCAN | 4.1 | 0.4 |

Таблица 15: Время работы (в секундах)

Как видно из таблицы, время работы алгоритма Affinity Propagation существенно дольше, чем у других алгоритмов. Для автоматической кластеризации повторов в рамках текущей работы лучшими алгоритмами были признаны алгоритм агломеративной кластеризации по пороговому значению, который показал лучшее значение FBCubed метрики на эталонной коллекции, и алгоритм DBSCAN, время работы которого существенно ниже, чем у других алгоритмов, что является актуальным при обработке документаций крупных проектов.

# Заключение

В рамках выпускной квалификационной работы были получены следующие результаты:

1. Произведена оцененка различных методов векторного представления текста на естественном языке применительно к задаче поиска повторов в документации ПО.

2. Произведено сравнение различных алгоритмов кластеризации.

3. Создана программная среда и разработана методика проведения эксперимента, позволяющая проводить апробацию создаваемых алгоритмов.

# Список литературы


[1] Brooks Jr F. P. The mythical man-month (anniversary ed.). Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Терехов А.Н., Терехов А.А. Автоматизированный реинжиниринг программ. Издательство Санкт-Петербургского государственного университета, 2000.

[3] Липаев Владимир Васильевич. Документирование сложных программных средств. Москва: СИНТЕГ, 2005.

[4] Шалыто А.А. Новая инициатива в программировании. Движение за открытую проектную документацию // Информационно-управляющие системы. 2003. № 4. С. 52–56.

[5] Duplicate finder toolkit / D. Luciv, D. Koznov, G. Chernishev et al. // Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. 2018. P. 171–172.

[6] Интерактивный поиск неточных повторов в документации программного обеспечения / Д.В. Луцив, Д.В. Кознов, А.А. Шелиховский [и др.] // Программирование. 2019. № 6. С. 55–66.

[7] Extraction of Archetype from Near Duplicates in Software Documentation / D. Koznov, D. Luciv, G. Chernishev et al. // 2019 Actual Problems of Systems and Software Engineering (APSSE). Moscow, Russia: IEEE, 2019. . P. 126–130.

[8] Луцив Дмитрий Вадимович. Поиск неточных повторов в документации программного обеспечения. диссертация на соискание учёной степени кандидата физико-математических наук: СПбГУ. 2018.

[9] Linux Kernel Documentation. URL: https://www.kernel.org/doc/Documentation/ (online; accessed: 04.05.2020).

[10] .NET Documentation. URL: https://docs.microsoft.com/en-us/dotnet/ (online; accessed: 04.05.2020).

[11] Python Documentation. URL: https://docs.python.org/3.3/ (online; accessed: 04.05.2020).

[12] Обнаружение неточно повторяющегося текста в документации программного обеспечения / Л.Д. Кантеев, Ю.О. Костюков, Д.В. Луцив [и др.] // Труды Института системного программирования РАН. 2017. Т. 29, № 4. С. 303–314.

[13] Day D., Priestley M., Schell D. Introduction to the Darwin Information Typing Architecture // IBM corporation. 2005.

[14] Javadoc Guide. URL: https://docs.oracle.com/en/java/javase/13/javadoc/javadoc.html (online; accessed: 04.05.2020).

[15] Parnas D. L. Precise documentation: The key to better software // The Future of Software Engineering. Springer, 2011. P. 125–148.

[16] Mkd (Extracteur de documents) [фр.]. 2016. . URL: https://webstore.iec.ch/preview/info_isoiec26514%7Bed1.0%7Den.pdf (online; accessed: 04.05.2020).

[17] Kramer D. API documentation from source code comments: a case study of Javadoc // Proceedings of the 17th annual international conference on Computer documentation. 1999. P. 147–153.

[18] PyDoc. URL: http://epydoc.sourceforge.net/ (online; accessed: 04.05.2020).

[19] Pike R. The Go programming language // Talk given at Google's Tech Talks. 2009.

[20] Laramee R. S. Bob's Concise Introduction to Doxygen: Tech. Rep.: : Technical report, The Visual and Interactive Computing Group, 2011.

[21] Knuth D. E. Literate programming // The Computer Journal. 1984. Vol. 27, no. 2. P. 97–111.

[22] Jupyter Notebooks-a publishing format for reproducible computational workflows. / T. Kluyver, B. Ragan-Kelley, F. Pérez et al. // ELPUB. 2016. P. 87–90.

[23] GitHub, Inc. Octoverse 2019. URL: `https://octoverse.github.com/` (online; accessed: 04.05.2020).

[24] Distributed representations of words and phrases and their compositionality / T. Mikolov, I. Sutskever, K. Chen et al. // Advances in neural information processing systems. 2013. P. 3111–3119.

[25] Enriching word vectors with subword information / P. Bojanowski, E. Grave, A. Joulin et al. // Transactions of the Association for Computational Linguistics. 2017. Vol. 5. P. 135–146.

[26] Bert: Pre-training of deep bidirectional transformers for language understanding / J. Devlin, M.-W. Chang, K. Lee et al. // arXiv preprint arXiv:1810.04805. 2018.

[27] Glue: A multi-task benchmark and analysis platform for natural language understanding / A. Wang, A. Singh, J. Michael et al. // arXiv preprint arXiv:1804.07461. 2018.

[28] Sun C., Huang L., Qiu X. Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence // arXiv preprint arXiv:1903.09588. 2019.

[29] Roberta: A robustly optimized bert pretraining approach / Y. Liu, M. Ott, N. Goyal et al. // arXiv preprint arXiv:1907.11692. 2019.

[30] Reimers N., Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks // arXiv preprint arXiv:1908.10084. 2019.

[31] Universal sentence encoder / D. Cer, Y. Yang, S.-y. Kong et al. // arXiv preprint arXiv:1803.11175. 2018.

[32] Universal sentence encoder for English / D. Cer, Y. Yang, S.-y. Kong et al. // Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2018. P. 169–174.

[33] Manning C. D., Raghavan P., Schütze H. Introduction to information retrieval. Cambridge university press, 2008.

[34] Comparison of clustering methods: A case study of text-independent speaker modeling / T. Kinnunen, I. Sidoroff, M. Tuononen et al. // Pattern Recognition Letters. 2011. Vol. 32, no. 13. P. 1604–1617.

[35] Clustering cancer gene expression data: a comparative study / M. C. de Souto, I. G. Costa, D. S. de Araujo et al. // BMC bioinformatics. 2008. Vol. 9, no. 1. p. 497.

[36] Top 10 algorithms in data mining / X. Wu, V. Kumar, J. R. Quinlan et al. // Knowledge and information systems. 2008. Vol. 14, no. 1. P. 1–37.

[37] Lloyd S. Least squares quantization in PCM // IEEE transactions on information theory. 1982. Vol. 28, no. 2. P. 129–137.

[38] Münz G., Li S., Carle G. Traffic anomaly detection using k-means clustering // GI/ITG Workshop MMBnet. 2007. P. 13–14.

[39] Dhanachandra N., Manglem K., Chanu Y. J. Image segmentation using K-means clustering algorithm and subtractive clustering algorithm // Procedia Computer Science. 2015. Vol. 54. P. 764–771.

[40] Arthur D., Vassilvitskii S. k-means++: The advantages of careful seeding: Tech. Rep.: : Stanford, 2006.

[41] Von Luxburg U. A tutorial on spectral clustering // Statistics and computing. 2007. Vol. 17, no. 4. P. 395–416.

[42] A survey of kernel and spectral methods for clustering / M. Filippone, F. Camastra, F. Masulli et al. // Pattern recognition. 2008. Vol. 41, no. 1. P. 176–190.

[43] Frey B. J., Dueck D. Clustering by passing messages between data points // science. 2007. Vol. 315, no. 5814. P. 972–976.

[44] Johnson S. C. Hierarchical clustering schemes // Psychometrika. 1967. Vol. 32, no. 3. P. 241–254.

[45] A density-based algorithm for discovering clusters in large spatial databases with noise. / M. Ester, H.-P. Kriegel, J. Sander et al.

[46] Jain A. K., Dubes R. C. Algorithms for clustering data. Prentice-Hall, Inc., 1988.

[47] Wagner S., Wagner D. Comparing clusterings: an overview. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.

[48] Hubert L., Arabie P. Comparing partitions // Journal of classification. 1985. Vol. 2, no. 1. P. 193–218.

[49] Fowlkes E. B., Mallows C. L. A method for comparing two hierarchical clusterings // Journal of the American statistical association. 1983. Vol. 78, no. 383. P. 553–569.

[50] Zhao Y., Karypis G. Criterion functions for document clustering: Experiments and analysis. 2001.

[51] Van Rijsbergen C. J. Foundation of evaluation // Journal of documentation. 1974.

[52] Karypis M. S. G., Kumar V., Steinbach M. A comparison of document clustering techniques // TextMining Workshop at KDD2000 (May 2000). 2000.

[53] Meilă M. Comparing clusterings by the variation of information // Learning theory and kernel machines. Springer, 2003. P. 173–187.

[54] Vinh N. X., Epps J., Bailey J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance // The Journal of Machine Learning Research. 2010. Vol. 11. P. 2837–2854.

[55] Bagga A., Baldwin B. Entity-based cross-document coreferencing using the vector space model // Proceedings of the 17th international conference on Computational linguistics-Volume 1 / Association for Computational Linguistics. 1998. P. 79–85.

[56] An extensive comparative study of cluster validity indices / O. Arbelaitz, I. Gurrutxaga, J. Muguerza et al. // Pattern Recognition. 2013. Vol. 46, no. 1. P. 243–256.

[57] A comparison of extrinsic clustering evaluation metrics based on formal constraints / E. Amigó, J. Gonzalo, J. Artiles et al. // Information retrieval. 2009. Vol. 12, no. 4. P. 461–486.

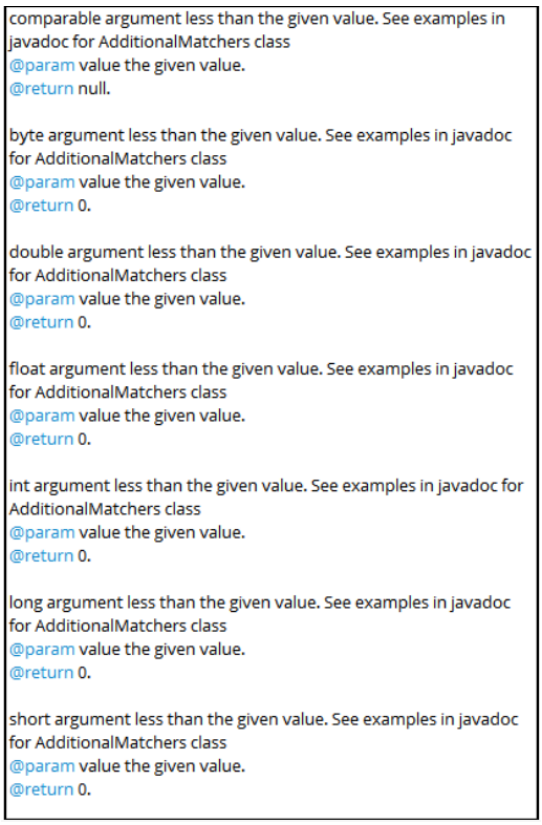
[58] Comparison of different reconstruction algorithms for three-dimensional ultrasound imaging in a neurosurgical setting / D. Miller, C. Lippert, F. Vollmer et al. // The International Journal of Medical Robotics and Computer Assisted Surgery. 2012. Vol. 8, no. 3. P. 348–359.

[59] Similarity-based support for text reuse in technical writing / A. J. Soto, A. Mohammad, A. Albert et al. // Proceedings of the 2015 ACM Symposium on Document Engineering. 2015. P. 97–106.

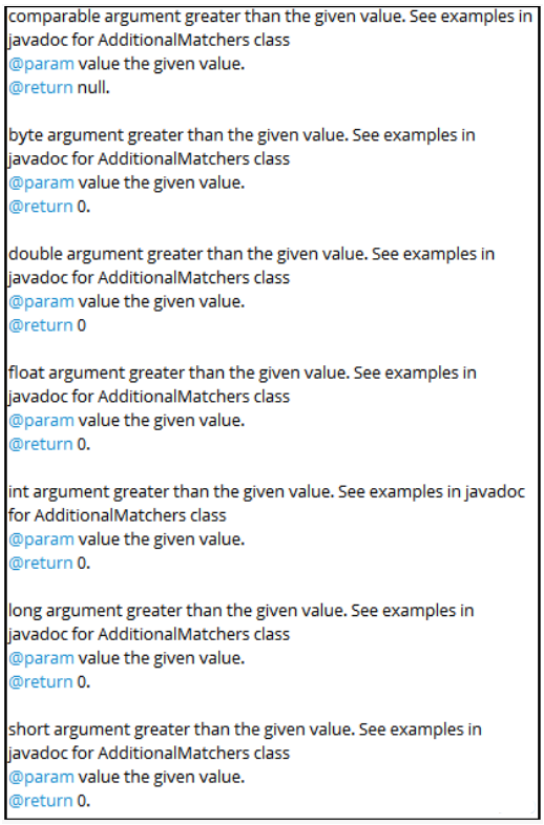
[60] Rosenberg A., Hirschberg J. V-measure: A conditional entropy-based external cluster evaluation measure // Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL). 2007. P. 410–420.

# Приложение

В документе Mockito экспертами были выделены две группы, представленные на рисунке.




a) Первая группа повторов




b) Вторая группа повторов

Обе группы состоят из 7 фрагментов текста. Отличиям в данных фрагментах заключаются в первом слове (тип аргумента), а так же в словах greater/less и 0/null. Альтерантивными способами объеденить данные фрагменты в группы являются:

- объединение в 7 групп по разным типам, по два повтора в каждой
- объединение в две группы (большую и маленькую) по 0/null

Алгоритмы относят все 14 фрагментов к одной группе. Человек, глядя на смысл написанного, пользуется критериями, отличными от объективных критериев, порождаемых метриками. Эксперты подтвердили, что данное разбиение может быть неоднозначным.