ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет Санкт-Петербургская школа физико-математических и компьютерных наук

Богомолов Егор Олегович

ИСПОЛЬЗОВАНИЕ МОДЕЛИРОВАНИЯ ТЕМ В ЗАДАЧАХ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Выпускная квалификационная работа
по направлению подготовки 01.04.02 Прикладная математика и информатика
образовательная программа «Программирование и анализ данных»

Рецензент
к.ф.-м.н., доц.
Фильченков Андрей Александрович

Научный руководитель
к.т.н., доц.
Брыксин Тимофей Александрович

Санкт-Петербург
2021

Оглавление

Введение

Актуальность и релевантные работы

Область разработки программного обеспечения (Software Engineering, SE) активно развивается на протяжении последних десятилетий. В 2019 году число разработчиков в мире почти достигло 24 миллионов, а к 2024 ожидается увеличение до более чем 28 миллионов [1]. В связи с этим важной задачей является улучшение и оптимизация процесса разработки ПО: компании разрабатывают инструменты для программистов, внедряют новые производственные практики. Процессы разработки ПО и способы их упростить изучают в том числе и научные области эмпирических методов в программной инженерии (Empirical Software Engineering, ESE) и машинного обучения в программной инженерии (Machine Learning for Software Engineering, ML4SE).

В задачи, которыми занимается область ML4SE, входят автодополнение кода [2], поиск плагиата [3, 4], рекомендация рефакторингов [5, 6], рекомендация напарников для ревью кода [7, 8], рекомендация программистов для исправления ошибок [9, 10, 11, 12, 13], генерация кода или отдельных его элементов [14, 15, 16], и многие другие. Многие из перечисленных задач формулируются как рекомендация чего-либо разработчикам. Для применения подходов к решению этих задач на практике требуется высокая точность, но даже высокая точность на исторических данных не является гарантией полезности подхода для конечного пользователя. Так, например, было показано, что для систем рекомендации ревьюеров даже точные предсказания с точки зрения исторических данных могут не приносить никакой пользы программистам, хотя те и считают предсказания релевантными [17].

Одним из препятствий к получению пользы от моделей машинного обучения в SE является то, что зачастую их предсказания невозможно интерпретировать: модели действуют как черные ящики. Для решения этой проблемы требуется использование в моделях интерпретируемых факторов. Одним из возможных способом получить такие факторы из

большого массива данных является тематическое моделирование.

Тематическое моделирование (Topic Modeling, TM) — это область машинного обучения, позволяющая выделить часто встречающиеся темы в большом массиве неразмеченных данных. Обычно рассматриваются текстовые данные: веб-сайты, твиты, сообщения на форумах, новостные заметки. Также к текстовым данным можно отнести исходный код, что позволяет применять TM в области ML4SE. К применениям TM к исходному коду относятся суммаризация кода [18, 19, 20], локализация дефектов [21, 22], приоритезация выполнения тестов [23], рекомендация рефакторингов [5] и другие. В большинстве из них TM используется как самостоятельный инструмент для представления кода в более простом виде, анализа трендов в области, построения явных правил для решения задач. В то же время почти не исследованным остается использование тематического моделирования для извлечения факторов, которые затем могут быть использованы в других моделях. Также в существующих работах используются классические методы TM, никак не учитывающие специфику работы с исходным кодом.

Цель и задачи

Данная работа ставит целью создание подхода для извлечения интерпретируемых признаков из произвольного исходного кода на основе тематического моделирования и проверка его применимости в задачах разработки ПО. Для этого ставятся следующие задачи:

- Разработать подход для извлечения тем из кода произвольной гранулярности, учитывающий специфику работы с исходным кодом.
- Разработать способ получения интерпретируемых факторов на основе построенной тематической модели, представляющих:
  - произвольные фрагменты кода (методы, файлы, проекты);
  - области экспертизы отдельных разработчиков.

- Проверить применимость извлеченных факторов в следующих задачах SE:
  - поиск похожих проектов — проверка применимости факторов на уровне проектов;
  - рекомендация разработчиков для исправления ошибок — проверка применимости факторов на уровне кода программистов.

Достигнутые результаты

В рамках данной работы предложен алгоритм *Code2Topic* для построения тематической модели по корпусу исходного кода и дальнейшего преобразования произвольных фрагментов кода в набор факторов, представляющих распределение тем в данном фрагменте. *Code2Topic* учитывает особенности словаря, встречающегося в исходном коде, и работает на основе кластеризации векторных представлений токенов из кода. Алгоритм также включает в себя новый подход к наглядному отображению полученных тем при помощи дендрограмы агломеративной кластеризации. Использование дендрограмы позволяет упростить ручную разметку тем текстовыми описаниями, которые в свою очередь позволяют облегчить понимание тем конечными пользователеми.

Для проверки применимости *Code2Topic* на практике был реализован инструмент Sosed для поиска похожих репозиториев среди базы из 9 миллионов проектов с открытым кодом. Sosed вычисляет для проектов вектор распределения тем при помощи *Code2Topic* и затем определяет похожие проекты по косинусной близости или KL-дивергенции между распределениями. На данный момент Sosed работает с кодом на 16 наиболее популярных языках программирования, позволяет фильтровать проекты по популярности и языку программирования, обладает интерпретируемостью предсказаний благодаря использованию тем, размеченных текстовыми описаниями. Для оценки качества работы инструмента была проведена разметка релевантности top-5 предсказаний

инструмента для 92 репозиториев. Средняя релевантность предсказаний в top-5 оказалась равна 4,2, для top-1 — 4,67 из 5, что говорит о высоком качестве работы инструмента и применимости *Code2Topic* в задаче поиска похожих проектов.

Второй задачей для проверки применимости выступила рекомендация разработчиков для исправления ошибок. Для этого был разработан алгоритм *Dev2Topic*, адаптирующий *Code2Topic* для извлечения факторов, отражающих экспертизу разработчиков. Для этого он анализирует историю разработки репозиториев, в которых работал программист, извлекает из них код, принадлежащий ему, и строит зависимость распределения тем в его коде от времени. Использование факторов, извлеченных *Dev2Topic*, позволило улучшить точность рекомендации программистов для исправления ошибок на датасете из 9700 записей об ошибках, собранных в системе YouTrack компании JetBrains, с 68% до 75%. Таким образом, извлеченные факторы действительно отражают характеристики разработчиков и могут быть использованы в практических задачах.

По результатам оценки на двух задачах, работающих с кодом на разном уровне (на уровне проектов и на уровне истории разработки отдельных программистов) можно заключить, что *Code2Topic* и *Dev2Topic* позволяют достать факторы, которые улучшают результаты моделей по сравнению с существующими работами. Благодаря использованию тематического моделирования факторы также оказываются интерпретируемыми, что упрощает их применение на практике.

1 Обзор литературы

1.1 Тематическое моделирование исходного кода

Тематическое моделирование применяется в области ML4SE для решения разных задач. Наиболее популярными точками приложения являются суммаризация кода [24, 25, 26], поиск дефектов [21, 22], поиск реализации конкретной функциональности [27, 28]. Наиболее популярными использованными алгоритмами являются Latent Semantic Analysis (LSA) [29], Latent Dirichlet Allocation (LDA) [30] и Additive Regularization of Topic Models (ARTM) [31], а также их модификации.

1.1.1 Используемые алгоритмы тематического моделирования

Одним из популярных подходов к тематическому моделированию, используемых в ML4SE, является Latent Semantic Analysis (LSA) [29], также называемый Latent Semantic Indexing (LSI). В его основе лежит анализ матрицы совстречаемости слов и документов. В роли документов могут выступать как произвольные тексты (новости, сообщения на форумах), так и исходный код. На первом шаге алгоритма строится матрица M размером $N \times D$, где N это размер словаря, а D это общее число документов. Элемент матрицы $M_{d,w}$ равен числу раз, которые слово w встречается в документе d.

На втором шаге алгоритма к матрице M применяется сингулярное разложение (Singular Value Decomposition, SVD) [32]. Оно находит три матрицы U, Σ и V, такие что:

$$M = U\Sigma V^*,$$

Матрицы U и V унитарные и содержат сингулярные векторы M, а Σ — прямоугольная диагональная матрица размера $N \times D$ и содержит на диагонали сингулярные числа M. LSA рассматривает строки U и V как векторы, являющиеся латентными представлениями слов и документов. По свойствам сингулярного разложения для слов, которые входят в похожие документы (или документов, содержащих похожие

слова), косинусное расстояние между соответствующими векторами будет бóльшим, чем для непохожих слов или документов. На последнем шаге LSA применяется понижение размерности полученных векторов. Для этого алгоритм оставляет только компоненты векторов, которые соответствуют K наибольшим сингулярным числам, где K является заранее выбранным пользователем числом тем.

Каждая из K компонент векторов для слов и документов представляет из себя отдельную тему. Для интерпретации тем обычно выделяются слова или документы, в которых соответствующие компоненты являются максимальными. Одной из проблем такого подхода является то, что темы могут получиться плохо интерпретируемыми.

Другим популярным подходом является Latent Dirichlet Allocation (LDA) [30]. Как и LSA, он требует от пользователя задать количество тем K. В его основе лежит предположение о том, что документы создаются по следующему генеративному процессу: сперва из распределения Дирихле [33] генерируются параметры θ_i и φ_k для распределений тем в документах и слов в темах, соответственно. Затем для каждой позиции в документе генерируется тема $z_{i,pos}$ из мультиномиального распределения [33] с параметрами θ_i, и, наконец, слово из мультиномиального распределения с параметрами $\varphi_{z_{i,pos}}$. При обучении алгоритм подбирает параметры таким образом, чтобы вероятность сгенерировать имеющийся корпус документов была максимальной:

$$P(corpus) = \prod_{d \in D} \prod_{w \in d} p(w|d) = \prod_{d \in D} \prod_{w \in d} \sum_{t \in T} p(w|t)p(t|d),$$

где T это множество тем, а вероятности выражаются согласно описанному генеративному процессу.

Еще один алгоритм тематического моделирования ARTM [31] является обобщением предыдущего подхода. Вероятности $p(w|t)$ и $p(t|d)$ можно представить не в виде явных распределений (например, через распределение Дирихле и мультиномиальное как в LDA), а в виде двух матриц Θ и Φ, содержащих распределения тем в документах и слов в темах. Тогда задача оптимизации будет сформулирована как максими-

зация относительно Θ и Φ:

$$\log P(corpus) = \sum_{d \in D} \sum_{w \in d} n_{w,d} \log \sum_{t \in T} \varphi_{w,t} \theta_{d,t} \underset{\Theta,\Phi}{\longrightarrow} \max,$$

где $n_{w,d}$ обозначает число раз, которые слово w встратилось в документе d. Чтобы матрицы действительно задавали распределения, на них накладывают дополнительные ограничения:

$$\varphi_{w,t} \geq 0; \sum_{w \in W} \varphi_{w,t} = 1$$

$$\theta_{d,t} \geq 0; \sum_{d \in D} \theta_{d,t} = 1$$

Метод ARTM заключается в добавлении к имеющейся задаче оптимизации по Θ и Φ дополнительных слагаемых, отвечающих за регуляризацию:

$$\log P(corpus) = \sum_{d \in D} \sum_{w \in d} n_{w,d} \log \sum_{t \in T} \varphi_{w,t} \theta_{d,t} + R(\Theta, \Phi) \underset{\Theta,\Phi}{\longrightarrow} \max,$$

Выбранные регуляризаторы могут, например, поощрять более разреженные матрицы, которые соответствуют меньшему количеству тем для каждого слова или документа. Алгоритм LDA также выражается в терминах ARTM путем выбора соответствующего регуляризатора.

Минусом ARTM является большое количество гиперпараметров, для подбора которых требуется достаточно точная метрика качества, которой в задачах решаемых при помощи TM может и не быть.

1.1.2 Приложения моделирования тем в ML4SE

Наиболее популярным приложением TM в области ML4SE является суммаризация кода (code summarization). Данная задача ставит целью упростить чтение и понимание кода для разработчиков. Для этого исследователи предлагают алгоритмы для визуализации кодовой базы, упрощения навигации в ней, представления информации о функциональности кода в сжатом виде.

Кун и др. [24] предложили подход для визуализации программ в виде карты на плоскости. Элементы программы, такие как методы и классы, сперва были отображены в пространство высокой размерности при помощи LSA [29], а затем преобразованы в точки на плоскости при помощи многомерного шкалирования [34].

Лиу и др. [25] использовали тематическую модель отношений (Relational Topic Model, RTM) [35] для построения сети классов, связанных между собой. RTM представляет собой модификацию LDA, в которой наличие или отсутствие связи между документами выражается через косинусное расстояние между их распределениями тем. После построения RTM авторы определяют для каждого класса в проекте логически связанные с ним (даже если между ними нет прямой зависимости), что позволяет разработчикам быстро изучить интересующую часть кодовой базы.

Две другие работы также предлагают способы визуализировать кодовую базу в виде графа со связями. Такой способ призван помочь разработчикам быстро получить представление об устройстве большого проекта. TopicXP [26] строит граф тем, где связи между темами определяются как суммарное число вызовов между классами, относящимися к ним. ITMViz [20] визуализирует карту классов при помощи интерактивной модели тем (Interactive Topic Model, ITM) [36], еще одной модификации LDA.

Еще одной точкой приложения тематического моделирования в SE является локализация функциональности (feature location). Задача состоит в определении частей кода (например, методов или классов), которые реализуют интересующую функциональность. Такая проблема возникает при определении части кодовой базы, в которую требуется внести изменения при добавлении новой функциональности или исправлении ошибок.

Для решения этой задачи также были предложены несколько подходов, основанных на LSA и LDA. Пошиваник и др. [27] использовали LSA для сопоставления текстовых описаний в запросах на внесение новой функциональности и релевантных мест в кодовой базе. Затем авто-

ры строили решетку понятий по полученным результатам при помощи анализа формальных представлений (Formal Concept Analysis, FCA). В другой работе информация из LSA комбинировалась с анализом связей в коде и алгоритмами информационного поиска (такими как HITS [37] и PageRank [38]) для локализации функциональности в трех крупных проектах на Java [28].

На протяжении разработки и поддержки проектов разработчики активно используют рефакторинги кода. Они призваны исправлять технический долг, улучшать архитектуру проекта, исправлять несоответствия, возникшие в результате длительной работы над проектом. Для рекомендации мест, в которых требуется применить рефакторинги, исследователи также предлагали использовать ТМ. В работе Бавоты и др. [5] по коду строилась тематическая модель отношений на основе как текстовой, так и структурной информации, которая затем использовалась дальнейшей рекомендации рефакторингов переноса методов (между классами) и классов (между модулями): авторы предположили, что методы или классы выгодно переносить в части кодовой базы, которые лучше соответствуют их тематике.

1.2 Поиск похожих проектов

Задача поиска похожих проектов или репозиториев формулируется по аналогии с поиском похожих веб-сайтов или изображений в классических поисковых системах. Запросом является программный проект, обычно с доступным исходным кодом. Выдачей же являются проекты, схожие с ним по функциональности, теме, используемым технологиям. Системы для поиска похожих проектов могут помочь в быстром прототипировании, поиске альтернатив для использовании библиотек, поиске плагиата [39].

В предыдущих работах по поиску похожих репозиториев использовались разные источники данных. МакМиллан и др. [40] создали CLAN — подход, разработанный для поиска похожих приложений на Java путем анализа вызовов API. Авторы использовали Latent Semantic

Analysis [29] для подсчета векторных представлений проектов на основе матрицы совместной встречаемости проектов и вызовов API в них. Авторы определили похожесть двух приложений как косинусную близость между соответствующими векторами.

Помимо анализа исходного кода работы использовали данные, специфичные для платформ хостинга репозиториев (например, GitHub [41] и SourceForge [42]). В одной из работ для определения схожести проектов использовалась система тэгов, коротких описаний характеристик проекта, платформы SourceForge [42]. Авторы предложили способ задать веса для тэгов и подсчитывали схожесть проектов на основе пересечения их множеств тэгов с учетом весов. Жанг и др. [41] предложили измерять похожесть проектов на основе звезд на GitHub, которые были добавлены одним и тем же пользователем за короткий период времени, и анализу содержимого их файлов README.

Также проблема определения похожих проектов возникает в области анализа мобильных приложений [43, 44, 45, 46]. Основным отличием от обычных программных проектов является множество доступных данных: описания в магазинах приложений, изображения, скриншоты интерфейса, обзоры пользователей, размер скачиваемых файлов. В то же время исходный код приложений может не находиться в открытом доступе, так как приложения являются коммерческими.

1.3 Рекомендация разработчиков для исправления ошибок

Во многих программных проектах используются системы отслеживания ошибок. Они позволяют пользователям писать сообщения об ошибках, которые затем исправляются разработчиками. Для ускорения процесса исправления требуется быстро и правильно определить, кто из разработчиков должен быть ответственным за исправление конкретной проблемы. Для этого могут быть использованы модели машинного обучения, которые рекомендуют разработчика по информации, имеющейся в сообщении об ошибке.

В предыдущих работах, применяющих машинное обучение для решения данной задачи, она рассматривалась как мультиклассовая классификация, где классами выступают разработчики из фиксированного множества участников проекта. В работе Шокрипура и др. [47] была предложена модель для ранжирования разработчиков на основе подсчета tf-idf [48] терминов, встречающихся в их коде и описаниях ошибок. В последующих работах для анализа сообщений использовались нейросетевые методы.

В работе Ли и др. [11] было предложено использовать свёрточную нейронную сеть (Convolutional Neural Network, CNN) [49] для анализа сообщений об ошибках. Авторы предлагают переводить слова из описания ошибки и её названия в векторы при помощи эмбеддинга, затем применяют CNN с несколькими одномерными фильтрами для извлечения неявного представления анализируемого текста, и наконец делают предсказание по полученному вектору при помощи многослойного перцептрона.

Гуо и др. [13] развивают идею использования CNN для анализа сообщений, используя в сети эмбеддинги, предобученные при помощи Word2Vec [50], и добавляя нормализацию внутри батчей [51]. Также авторы предлагают учитывать информацию о том, как давно каждый программист участвовал в работе над репозиторием.

1.4 Выводы

Существующие работы, применяющие тематическое моделирование в SE, используют его для:

- анализа трендов в области: тематическое моделирование всех проектов на GitHub [52], определение тем, наиболее подверженных дефектам [21, 22];
- построения правил в явном виде или через RTM, которые затем используются для предсказаний: рекомендация рефакторингов [5], локализация функциональности [28, 27];

- представления информации о коде в сжатом виде: визуализация кодовой базы [20], построение графа связанных классов [26].

В то же время ТМ также может применяться как инструмент для извлечения факторов, которые затем могут быть использованы в моделях машинного обучения в произвольных задачах.

Используемые методы тематического моделирования напрямую перенесены из области NLP — они никак не учитывают специфику работы с исходным кодом. В то же время код значительно отличается от текстов на естественном языке: он обладает более богатым словарем [53], у него более сложная структура (код может быть представлен в виде абстрактного синтаксического дерева), слова в коде представляют из себя целые фразы, объединенные при помощи CamelCase или snake_case.

2 Тематическое моделирование исходного кода

По сравнению с текстами на естественном языке у исходного кода есть ряд особенностей, которые усложняют применение классических методов ТМ. Решить большинство проблем можно путем изменения процесса построения словаря, извлечения семантики слов при помощи использования *эмбеддингов* для отображения слов в векторное пространство и дальнейшего построения тем путём кластеризации полученных векторов. В данной главе обсуждаются трудности работы с кодом, предлагаются способы их решения, и в итоге описывается предложенный в данной работе алгоритм построения тем на основе эмбеддингов *Code2Topic*. Предложенный подход учитывает специфику работы с кодом и позволяет упростить интерпретацию тем.

2.1 Размер словаря в исходном коде

У словаря в исходном коде есть ряд особенностей. Во-первых, его размер значительно больше словаря, встречающегося в текстах на естественном языке [53]. Например, для корпуса из 14 тысяч проектов на Java размер словаря без дополнительной фильтрации составляет около 10 миллионов токенов (без комментариев и строковых литералов) [54]. Это вызвано тем, что токены в коде могут соответствовать не отдельным словам, а целым предложениям, объединенным через CamelCase (“$computeTopicModels$”) или snake_case (“$compute\_topic\_models$”). Также разработчики придумывают новые термины и используют слова из своих родных языков (зачастую отличных от английского). Такой большой размер словаря ведет к появлению большого количества слов с низкой частотой. Используемые алгоритмы тематического моделирования основаны на анализе матрицы совместной встречаемости, а значит получают информацию о токенах из множества документов, в которые они входят. Для редких слов такой способ получения информации будет менее надежным.

При дроблении токенов на сабтокены в соответствии с CamelCase и snake_case словарь для 14 тысяч проектов все ещё составляет около миллиона сабтокенов. Если же работать с несколькими языками программирования в масштабах десятков и сотен тысяч проектов, то даже число сабтокенов будет исчисляться миллионами. Также стоит отметить, что для интерпретации тем в большинстве методов TM пользователям предлагают идентифицировать их по наиболее популярным словам. Если слова являются сабтокенами, то разработчикам может быть сложнее понять смысл темы, поскольку без объединения в полноценные токены они сохраняют лишь часть смысла. В отдельных случаях это может повлиять на процесс тематического моделирования: к примеру, сабтокены “dot” и “net” могут встречаться по отдельности в проектах, связанных с интернетом, но в комбинации “dotNet” в рамках одного токена они указывают на использование конкретного фреймворка.

Для уменьшения числа токенов или сабтокенов также могут применяться лемматизация или стемминг [55], распространенные техники в области NLP. Стемминг использовался в предыдущих работах связанных с исходным кодом [52], но код после применения стемминга теряет часть информации: разные по смыслу токены после стемминга могут иметь одинаковую форму, тем самым их различие будет утрачено. Согласно существующим исследованиям стемминг сокращает словарь в исходном коде всего на 5% [54], что не очень выгодно на фоне потери информации. Лемматизация переводит слова в их начальную форму, тем самым объединяя разные формы одного слова в одно словарное. Применение лемматизации в исходном коде является возможным решением, но осложняется введением разработчиками новых терминов и использованием слов из разных языков.

В NLP для уменьшения размера словаря используют Byte-Pair Encoding (BPE) [56], алгоритм, позволяющий построить словарь произвольного размера без потери (или с заданной потерей) информации. Алгоритм инициализируется словарем, в котором словами являются все *символы*, которые встречаются в имеющемся корпусе данных. Если некоторые символы встречаются редко (например, из-за наличия ред-

ких иероглифов), то алгоритм может не использовать их, тем самым отбрасывая заданную часть информации. Затем BPE итеративно увеличивает словарь, объединяя в новое слово два наиболее часто встречающихся вместе среди уже имеющихся в словаре. Например, если в словаре уже есть слова “get” и “class”, а комбинация “getclass” встречается чаще всего среди всех пар слов, то они будут объединены в новое слово “getclass”. Таким образом, BPE позволяет построить словарь любого заданного размера.

После обучения BPE токены в тексте представляются в виде последовательности из нескольких частей, каждая из которых есть в построенном словаре. Это напоминает представление токенов в коде в виде нескольких сабтокенов, но части в BPE могут с ними не совпадать. Несмотря на то, что такой алгоритм позволяет построить словарь произвольного размера, который покрывает весь текст, и слова в котором имеют достаточно высокую частоту встречаемости, его невозможно напрямую использовать вместе с существующими алгоритмами тематического моделирования. Существующие алгоритмы ТМ предполагают, что для интерпретации будут использованы токены, которые относятся к конкретной теме. В то же время сабтокены в BPE могут не иметь явного смысла с точки зрения пользователя, что усложнит интерпретацию. Помимо этого один и тот же сабтокен в BPE может встречаться в токенах с совершенно разным смыслом, а значит встречаться в очень большом количестве документов. Такие сабтокены будут относиться сразу к большому количеству тем, но с небольшой вероятностью, что ухудшает качество полученных тем.

Использование токенизации при помощи BPE для работы с исходным кодом позволяет построить словарь нужного размера, но требует нового подхода к тематическому моделированию, который позволит представлять семантику целого токена, используя его разбиение на части. Возможным решением данной проблемы является использование векторных представлений слов, описанное в следующей подглаве.

2.2 Использование эмбеддингов для представления семантики токенов

Недостатком существующих подходов к тематическому моделированию является то, что они получают информацию о семантике токенов по множеству документов, в которых они встречаются. Эта проблема особенно остро возникает в исходном коде из-за значительных размеров словаря и использования специализированных терминов. Решить проблему извлечения семантики в подобных условиях можно при помощи использования *эмбеддингов*.

Эмбеддинги слов (word embeddings) или векторное представление слов — это группа методов машинного обучения, которые направлены на сопоставление словам численных векторов в пространстве низкой (по сравнению с размером словаря) размерности. В то время, как первые методы в этой области использовали для построения векторов матрицу совместной встречаемости в корпусе документов наподобие того, как это делалось в LSA (см. раздел 1.1.1), современные методы строят представления при помощи рассмотрения контекста вхождений слов в корпус.

Первым современным алгоритмом для построения векторных представлений является Word2Vec [50]. В его основе лежит рассмотрение контекстного окна — непосредственного окружения слова в тексте. Окно размера C обозначает, что контекст составляют $\frac{C}{2}$ слов до и после интересующего слова. Пример контекстного окна в предложении показан на Рисунке 2.1 Алгоритм Word2Vec имеет две модификации — CBOW (Continous Bag-Of-Words) и Skip-Gram.

Рис. 2.1: Пример контекстного окна ширины шесть внутри предложения.

При обучении Skip-Gram рассматривается задача предсказания слов из контекста по центральному слову. Центральному слову сопоставляется вектор при помощи матрицы эмбеддингов $W_{N\times D}$, где N это размер словаря, а D — желаемая размерность векторов. Затем по полученному вектору предсказываются слова из контекста путем умножения полученного вектора на еще одну матрицу $W'_{D\times N}$. К полученному вектору применяется операция Softmax, получая в результате вектор вероятности для всех слов встретиться в контексте центрального. Итоговая модель обучается при помощи алгоритма обратного распространения ошибки [57] как обычная нейронная сеть. Алгоритм CBOW работает таким же образом, но в ходе его работы по сумме векторов слов из контекста предсказывается центральное слово. Результатом работы алгоритма является обученная матрица W, строки которой являются векторами для слов.

Последующие алгоритмы развивают идею Word2Vec, интегрируя в векторы больше информации. FastText [58] представляет слово не в виде одного вектора, а в виде суммы эмбеддингов слова и его N-грамм (буквенных подстрок длины N) для N в заданном диапазоне (обычно от 3 до 6). Например, для слова “used” 3-граммами будут “<us”, “use”, “sed”, “ed>”, где “<” и “>” обозначают начало и конец слова. Такой подход позволяет дополнительно выучить векторные представления для часто встречающихся частей слов: окончаний, корней, приставок. Так, для окончания “ed>” вектор будет содержать информацию о том, что окончание обозначает прошедшее время. Благодаря обучению эмбеддингов N-грамм FastText может определять векторы для слов, которые не встречались в обучающей выборке — вектор для них будет равен сумме векторов входящих в них N-грамм. В случае, если для построения словаря по коду используется разбиение на сабтокены по CamelCase/snake_case, использование FastText имеет преимущество: для новых данных векторы сабтокенов можно будет построить по их N-граммам.

Другим алгоритмом построения эмбеддингов слов является алгоритм GloVe (Global Vectors) [59]. Авторы алгоритма объединили идеи

из подходов, строивших векторы по глобальной информации (совместной встречаемости слов и документов), и подходов, работающих с контекстными окнами. GloVe основан на построении матрицы логарифма совместной встречаемости пар слов и дальнейшей её факторизации. Хотя матрица содержит глобальную информацию о всем корпусе, совместная встречаемость определена локально для слов, находящихся в одном окне. Минусом GloVe является то, что алгоритм не позволяет построить векторы для слов, которые не встречались в тренировочном корпусе. Поскольку при использовании BPE для построения словаря новые сабтокены появиться не могут, GloVe можно использовать вместе с BPE. Данная комбинация показала хорошие результаты в области NLP [60].

Построив векторы для составных частей слов, нам требуется перейти к векторам для целых токенов, обладающих более понятным смыслом. Согласно существующим работам при использовании GloVe и BPE векторы для токенов можно определить как среднее векторов их составн[60]. При этом качество полученных векторов в задачах NLP сопоставимо с построением векторов напрямую использованием FastText. Таким образом, для построения векторных представлений токенов в корпусе кода требуется разбить токены на части при помощи BPE, обучить эмбеддинг для полученных сабтокенов алгоритмом GloVe и вычислить векторы для целых токенов усреднение векторы их частей.

2.3 От эмбеддингов к темам

Векторы слов, полученные в результате алгоритмов построения эмбеддингов, находятся близко в пространстве, если соответствующие слова часто встречаются в похожем контексте. В большинстве случаев это означает, что слова обладающие схожей семантикой имеют близкие векторы. С точки зрения тем, слова, относящиеся к конкретной теме, формируют в пространстве кластер близких векторов относительно косинусного расстояния, а значит их можно выделить при помощи алгоритмов кластеризации.

Выбор алгоритма кластеризации зависит от размера словаря токенов. В данной работе были испробованы несколько популярных подходов: DBSCAN [61], HDBSCAN [62], Spherical K-Means [63]. Для работы с миллионами токенов и косинусным расстоянием DBSCAN и HDBSCAN показали себя медленными и требующими ручной настройки гиперпараметров. Также оба алгоритма ожидают, что в кластеризуемом пространстве есть ярко выраженная кластерная структура. Для пространства эмбеддингов токенов это не выполнено: в то время как группы близких по теме токенов оказываются рядом, внутри группы их почти всегда можно разбить на более мелкие темы. Например, кластер, описывающий работу с текстовыми данными, может быть дальше разбит на кластеры операций со строками (“toLowerCase”, “filterAlphaNumeric”, и т.д.) и с отдельными символами (“findChars”, “alphaNumericCharacters”, и т.д.).

В условиях подобной структуры пространства хорошо себя показывает алгоритм Spherical K-Means [63]. Он базируется на алгоритме K-Means [64], но работает с косинусным расстоянием и выбирает векторы единичной длины как центры кластеров. Алгоритм требует предварительно задать количество тем K. Затем он выбирает K случайных векторов в качестве пробных центров кластеров и итеративно повторяет следующий процесс:

1. для всех векторов найти ближайший центр по косинусному расстоянию и отнести вектор к соответствующему кластеру;

2. для всех кластеров пересчитать их центр как нормализованное среднее относящихся к ним векторов.

Алгоритм останавливается, когда перемещение центров кластеров в рамках одной итерации не превышает пороговое значение. Результатом работы алгоритма является K кластеров, каждый из которых представлен центральным вектором. Для токенов можно определить кластер, к которому они относятся, найдя ближайший центр по косинусному расстоянию.

Минусом предложенного подхода к построению тем через кластеризацию является необходимость вручную задавать количество кластеров K. В рамках этой работы были рассмотрены способы автоматического выбора числа кластеров, например, gap statistic [65]. Такие подходы анализируют структуру внутри кластеров, например, среднее расстояние между элементами отнесенными к одному кластеру, и сравнивают её со случайной расстановкой центров кластеров. В случае, если в пространстве есть выраженная кластерная структура, после достижения правильного числа кластеров разность в средних расстояниях между обученным и случайным разбиением станет меняться заметно медленнее. Подобные техники оказались неприменимыми в случае кластеризации токенов, так как в пространстве отсутствует явная кластерная структура, и увеличение числа кластеров ведет к монотонному изменению в разности средних расстояний. Тем не менее, изменяя K, можно контролировать то, насколько "общими" будут темы, при этом сохраняя интерпретируемость.

2.4 Интерпретация тем

На предыдущих этапах работы алгоритма для корпуса кода были выделены темы из кода в виде кластеров в пространстве эмбеддингов токенов. По аналогии с существующими алгоритмами тематического моделирования для интерпретации их смысла можно использовать набор токенов, которые относятся к данной теме:

- обладающих наибольшей частотой в тренировочном корпусе: такие токены более просты для понимания, поскольку они менее специфичны из-за их большой частоты;
- наиболее близких центру кластера: такие токены лучше отражают специфику конкретной темы;
- случайно выбранных внутри кластера: в случае более общих тем такой набор токенов может лучше отображать их аспекты.

Поскольку для получения полноценного представления о теме кластера разработчику нужно изучить токены, представляющие его, процесс может быть упрощен, если каждому кластеру будет дано текстовое описание. Для этого требуется ручная разметка, которую должны проводить разработчики, обладающие экспертизой в соответствующих областях. При разметке разработчикам придется проанализировать токены, представляющие кластер, чтобы дать точное описание темы. В рамках разметки большого количества кластеров этот процесс становится трудозатратным. Чтобы упростить его, представить темы в более наглядном виде и одновременно решить проблему выбора числа кластеров, в данной работе предлагается воспользоваться агломеративной кластеризацией [66].

Так как центры кластеров также лежат в пространстве эмбеддингов слов, они также отражают семантическую информацию о кластерах. Значит, расстояние между центрами кластеров говорит о том, насколько соответствующие темы похожи друг на друга. Это наблюдение позволяет сделать агломеративную кластеризацию на множестве полученных тем, получив в результате бинарное дерево, называемое дендрограмой, отображающее их иерархическую структуру.

Агломеративная кластеризация работает следующим образом: изначально каждый вектор v помещается в отдельную группу c. Затем находится пара групп такая, что расстояние между минимально и объединяется в группу большего размера. Расстояние между группами можно задавать разными способами: как минимальное расстояние между их элементами, максимальное, среднее и т.д. На практике лучше всего себя показало использование среднего расстояния. Процесс продолжается, пока не останется всего одна группа. Процесс объединения групп можно отобразить в виде дендрограмы (бинарного дерева), где листьям соответствуют изначальные темы, а каждая вершина соответствует объединению тем в своём поддереве.

У представления построенных тем в виде дендрограмы есть ряд преимуществ. Во-первых, нарисованная дендрограма позволяет наглядно отобразить множество тем. Во-вторых, её можно использовать при раз-

метке словесных описаний для тем. Все вершины дендрограмы можно считать темами, при этом чем выше вершина лежит в дереве, тем более "общей" теме она соответствует. Этим фактом можно воспользоваться при разметке и попросить разработчиков давать текстовые описания не отдельному кластеру, обладающему достаточно узким смыслом, а бинарным делениям в дендрограме. При этом разработчик будет обладать текстовым описанием, которое уже дано текущей вершине и наборами токенов, представляющих две дочерние темы. В том случае, если разработчик понимает семантический смысл данного деления, процесс продолжается, тема разбивается на две, и разметка идёт дальше по дереву. Если же программист не может дать текстовое описание дочерним темам, поскольку больше не понимает их смысл, то дробление останавливается.

Таким образом одновременно с упрощением разметки можно решить проблему выбора количества тем: изначально задать K достаточно большим, а затем отрегулировать количество тем в ходе разметки текстовыми описаниями. Пример фрагмента дендрограмы, содержащий разметку кластеров текстовыми описаниями, показан на Рисунке 2.2. Еще одним плюсом подобной схемы разметки является то, что она гарантирует, что все извлеченные темы понятны человеку. Это важно при использовании тем в качестве интерпретируемых факторов для моделей, о котором рассказано в следующем разделе.

2.5 Извлечение факторов на основе тем

Предложенный алгоритм построения тем может сопоставить тему каждому токену в исходном коде. После этого произвольный фрагмент кода можно представить в виде распределения тем среди его токенов. Для этого его требуется токенизировать, вычислить векторы для полученных токенов путем усреднения векторов сабтокенов и определить кластер для каждого вектора. Результат можно представить в виде вектора из T чисел, где T — это итоговое количество тем. Компонента вектора k обозначает то, какую долю из всех токенов в фрагменте кода состав-

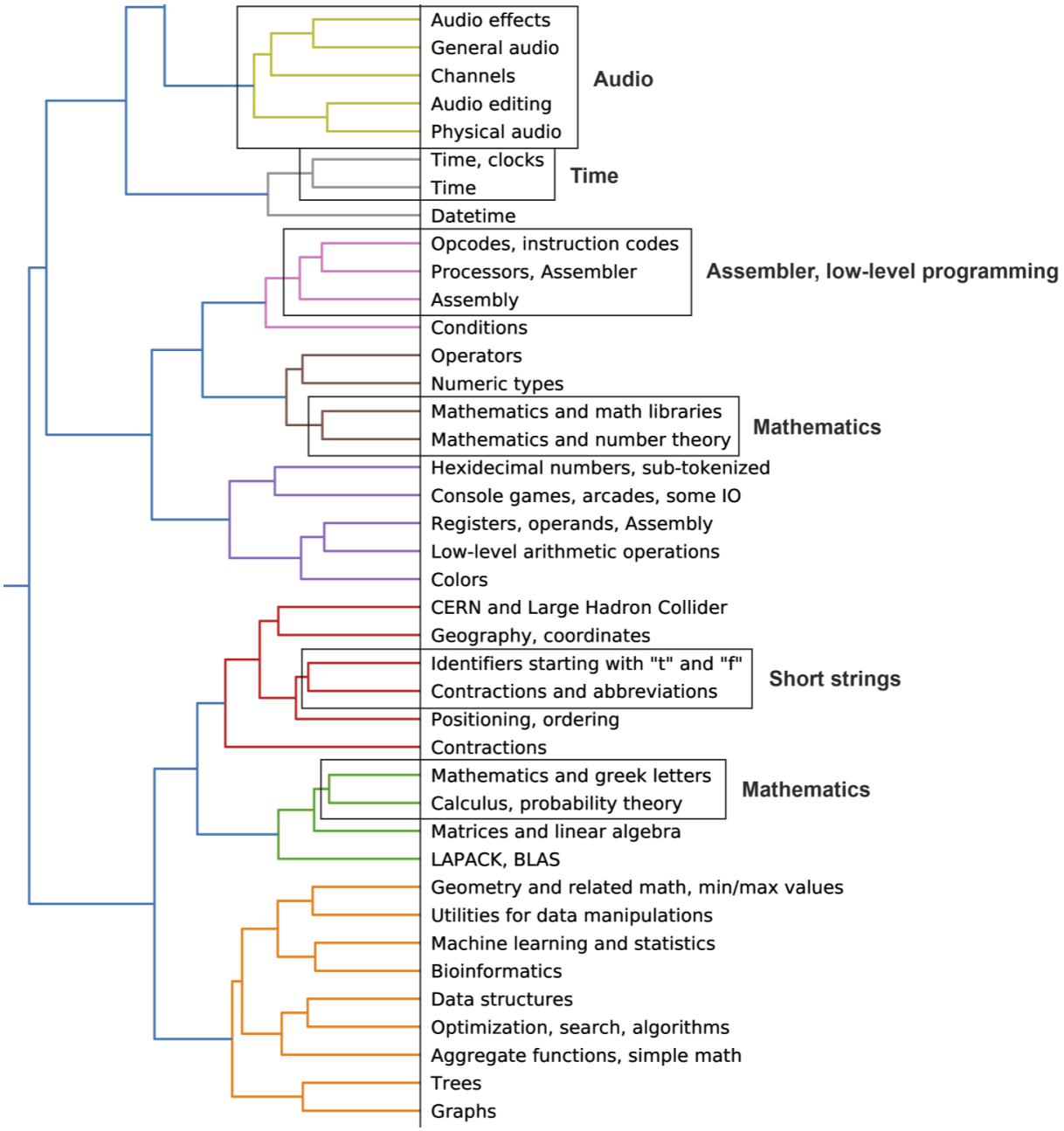

Рис. 2.2: Пример фрагмента дендрограмы для корпуса репозиториев с открытым исходным кодом. Прямоугольниками обведены группы кластеров, которые не были разбиты на более мелкие темы при разметке по дендрограме. Описания для листьев дендрограмы в прямоугольниках даны на основе токенов, представляющих их.

ляют токены из темы k. Полученный вектор для фрагмента кода затем можно использовать в качестве входных данных в других задачах.

2.6 Выводы

Предложенный алгоритм построения тем для исходного кода *Code2Topic* представлен на Рисунке 2.3. Алгоритм работает следующим образом:

1. Токенизировать корпус и разбить полученные токены на части при помощи BPE или разбиения на сабтокены по CamelCase или snake_case.

2. Обучить эмбеддинги в последовательностях полученных сабтокенов при помощи одного из существующих алгоритмов, например, FastText [58] или GloVE [59].

3. Подсчитать векторные представления токенов, усреднив эмбеддинги их частей.

4. Выделить кластеры в пространстве токенов, например, при помощи сферического K-Means [63], задав число кластеров достаточно большим.

5. Построить дендрограму кластеров при помощи агломеративной кластеризации [66].

6. При необходимости разметить кластеры текстовыми описаниями, двигаясь от корня дендрограмы к листьям.

Для применения факторов на основе тематического моделирования кода на практике требуется проделать следующие шаги:

1. Токенизировать корпус и разбить полученные токены на части при помощи BPE или разбиения на сабтокены по CamelCase или snake_case.

2. Подсчитать векторные представления токенов при помощи обученной модели эмбеддингов.

3. Определить темы для токенов, используя обученную модель кластеризации.

4. Для фрагмента кода вычислить распределение токенов по темам и получить вектор размера K.

5. Полученное распределение можно использовать в качестве факторов в последующей практической задаче.

6. При интерпретации предсказаний в практической задаче требуется выделить факторы, повлиявшие на ответ, и представить их текстовые описания (данные разметчиками или в виде избранных токенов).

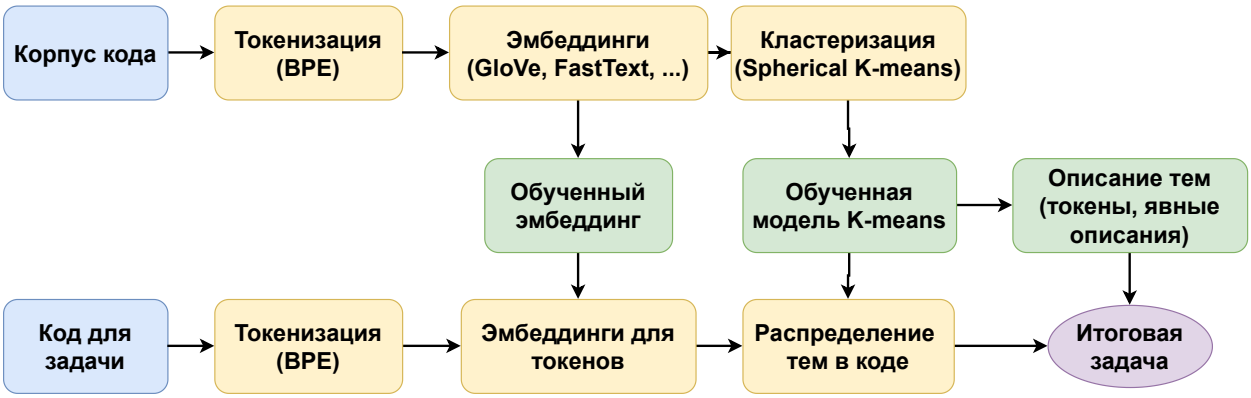

Рис. 2.3: Схема работы алгоритма *Code2Topic*.

В данной главе был предложен алгоритм построения тем, который учитывает специфику словаря, возникающую при работе с исходным кодом. Подход основан на использовании векторных представлений для отображения семантики токенов, составляющих код. Сами темы строятся на основе кластеризации эмбеддингов токенов, выделяя группы токенов схожих по тематике в векторном пространстве. Для построения эмбеддингов предлагается использовать алгоритмы FastText и GloVe, для кластеризации — Spherical K-Means. Обе стадии могут быть усовершенствованы одновременно с развитием техник построения эмбеддингов и кластеризации. Так, использование контекстуальных эмбеддингов, таких как ELMo [67], может позволить отразить многозначность слов, зависящую от контекста использования.

Помимо алгоритма построения тем был предложен способ их визуализации при помощи дендрограмы агломеративной кластеризации центров кластеров, который также упрощает разметку тем словесными описаниями. Одновременно с упрощением разметки использование дендрограмы позволяет решить проблему выбора количества тем, в то же время гарантируя их понятность для разработчиков. Темы, извлеченные предложенным алгоритмом, могут быть использованы в качестве интерпретируемых факторов в задачах ML4SE.

3 Sosed: инструмент для поиска похожих репозиториев

В главе 2 был описан алгоритм *Code2Topic*, предложенный в рамках данного исследования извлечения интерпретируемых факторов из исходного кода на основе нового подхода к тематическому моделированию. Для проверки его применимости на практике было выбрано две задачи, работающие с кодом разной гранулярности: поиск похожих проектов и рекомендация разработчиков для исправления ошибок. В данной главе описывается Sosed — разработанный в ходе данной работы инструмент для поиска похожих проектов, основанный на предложенном алгоритме для извлечения факторов из кода.

Поиск похожих проектов среди большого количества репозиториев с открытым исходным кодом может помочь в разных задачах разработки ПО: для быстрого прототипирования, поиска плагиата, суммаризации кода [39]. В то время, как современные поисковые системы позволяют искать похожие веб-страницы или похожие изображения, пока что не существует общепринятого подхода для поиска похожих программных проектов. Предыдущие работы в этой области использовали самые разные данные: вызовы API в Java [40], содержимое файлов README [41], реакции пользователей в виде звезд на GitHub [41], теги на платформе SourceForge [42].

Для решения этой задачи был разработан инструмент, называющийся Sosed, основанный на предложенном подходе к извлечению факторов из исходного кода через построение тем. Sosed доступен в открытом доступе на GitHub [68], статья про инструмент была представлен на конференции Automated Software Engineering 2020 [69]. Согласно проведенной вручную оценке качества, Sosed значительно опережает существующие подходы к поиску похожих проектов.

3.1 Обзор устройства инструмента Sosed

Схема работы инструмента представлена на Рисунке 3.1. Во-первых, для поиска похожих проектов требуется определить пространство поиска. Для этого был использован датасет из 9 миллионов репозиториев на GitHub, которые были собраны Марковцевым и коллегами [52]. Это самый большой существующий дедуплицированный датасет из программных проектов, который подходит для данной задачи. Данная работа является первым исследованием по поиску похожих репозиториев, оперирующей датасетом такого масштаба.

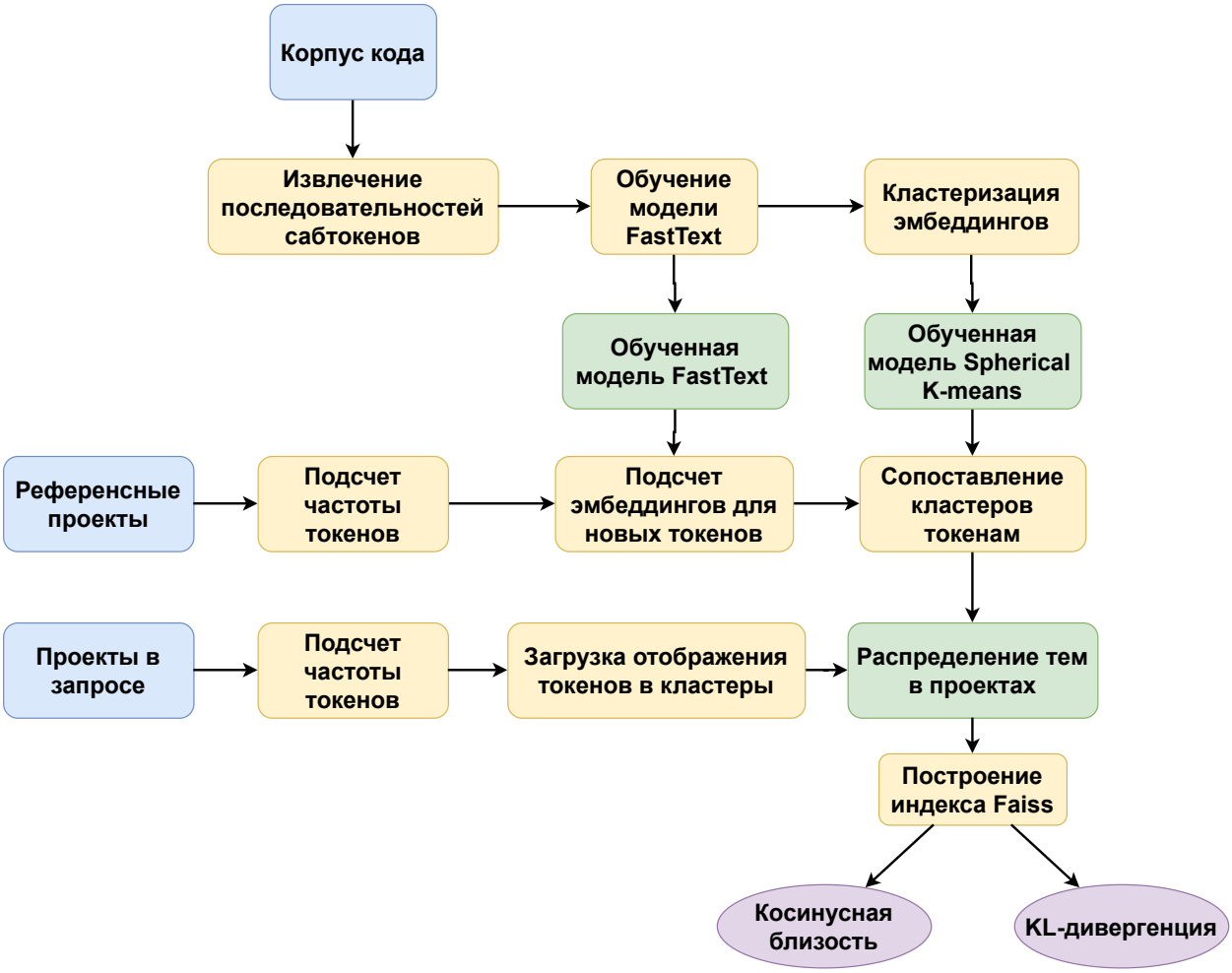

Рис. 3.1: Схема устройства инструмента для поиска похожих проектов.

Далее требуется преобразовать проекты в формат, подходящий для поиска. Для этого был использован алгоритм, описанный в главе 2. Векторные представления для сабтокенов были обучены при помощи алгоритма FastText [58] на большом корпусе исходного кода [70]. Затем

для полученных векторов были определены 256 кластеров при помощи алгоритма Spherical K-Means [63]. Наконец, для каждого проекта было вычислено распределение его токенов по кластерам, которые соответствуют темам. Чтобы темы было проще интерпретировать, была проведена разметка тем словесными описаниями по методике, представленной в разделе 2.4.

Наконец, для поиска похожих проектов использовались два подхода: поиск ближайших распределений по косинусной близости соответствующих векторов и по KL-дивергенции [71]. Оба подхода показали схожие результаты. Для ускорения поиска похожих распределений использовалась библиотека Faiss [72].

3.2 Пространство поиска

Датасет, предложенный Марковцевым и др. [52], содержит 9 миллионов репозиториев, собранных на GitHub в 2017 году. Для каждого репозитория датасет содержит множество всех токенов, уже разбитых на сабтокены, которые были выделены в файлах данного проекта. Датасет дедуплицирован: из него удалены все явные и неявные форки (forks). Форком называют проект, который копирует другой проект в определенный момент его истории и продолжает развиваться с этого места. Такие проекты зачастую являются почти полными дубликатами, поэтому в датасет добавляется только оригинальный проект. Неявными называются те форки, которые не помечены таковыми на GitHub, хотя и являются почти точной копией другого репозитория. Для выявления таких проектов авторы воспользовались алгоритмом на основе техники Locality-sensitive hashing (LSH).

Поскольку проекты в датасете были собраны в 2017 году, многие из них сильно изменились с того момента и даже могли быть удалены. Несмотря на это, использование данного датасета позволило обеспечить очень большое пространство поиска, а качество итоговых результатов говорит о том, что даже несмотря на изменения проекты остались релевантными. В будущем планируется обновить датасет, собрав актуаль-

ное множество проектов на GitHub и других платформах.

3.3 Векторные представления токенов

Для обучения векторных представлений сабтокенов был использован датасет последовательностей идентификаторов, извлеченных из 120 000 репозиториев на GitHub [70]. Он содержит последовательности токенов, разбитых на сабтокены, на примерно 200 языках программирования. Ввиду того, что исходный код в обоих используемых датасетах уже был разбит на сабтокены, для обучения эмбеддингов был выбран алгоритм FastText [58], детали работы которого приведены в разделе 2.2.

Согласно работам в области обработки естественного языка, сабтокены отображались в пространство размерности 100. Вместе с эмбеддингами для сабтокенов FastText также строит эмбеддинги для составляющих их N-грамм, что позволило вычислить векторы для сабтокенов, которые встречались только в большем датасете из 9 миллионов проектов. Результатом обучения модели стали векторы для 40 миллионов токенов.

Для того, чтобы пользователи могли использовать новые проекты в качестве запросов, необходимо уметь токенизировать и разбивать на сабтокены новые проекты таким же образом, как это было сделано в выбранных датасетах. Для этого был разработан еще один инструмент, называющийся Buckwheat, доступный как отдельный инструмент на GitHub [73]. Входом для инструмента является список путей к локальным директориям или ссылок на репозитории на GitHub. Итогом работы Buckwheat является список всех сабтокенов, найденных в проекте, и их количество.

Первым шагом токенизации является определение языка для всех файлов в каждом проекте. Для этого используется *enry* [74], языковой инструмент реализованный на Go, который применяет несколько стратегий для определения языка в каждом файле: название файла, расширение, анализ содержимого. *enry* поддерживает 382 языка программирования, отличается быстрой скоростью работы и не требует

информации из Git, что позволяет применять его к произвольному набору файлов.

Итогом работы *enry* является JSON файл со всеми найденными языками и списком файлов для каждого языка. Buckwheat оставляет из них те 16 языков, с которыми он может работать: C, C#, C++, Go, Haskell, Java, JavaScript, Kotlin, PHP, Python, Ruby, Rust, Scala, Shell, Swift и TypeScript. Этот набор языков был выбран для поддержки согласно статистике по популярности языков программирования [75]. В будущем возможна поддержка большего количества языков.

Следующий шаг токенизации — это извлечение идентификаторов из исходного кода: названий переменных, классов, функций и их аналогов. Для этого требуется токенизировать файл, проитерироваться по всем токенам и выделить токены, обладающие определенными типами (не включая литералы, комментарии и т.д.). Для этого Buckwheat использует два инструмента. 12 из 16 языков, включая 10 наиболее популярных, токенизируются при помощи *Tree-sitter* [76], библиотеки для парсинга, использующей грамматики языков для построения абстрактного синтаксического дерева (Abstract Syntax Tree, AST) по каждому файлу. Для четырёх оставшихся языков (Scala, Swift, Kotlin, Haskell) грамматики для *Tree-sitter* пока что недоступны, поэтому для их парсинга используется *Pygments* [77]. *Pygments* — это библиотека, предоставляющая набор лексеров, которые разбивают код на типизированные токены. Buckwheat затем фильтрует их, оставляя только токены, соответствующие идентификаторам.

Заключительным шагом токенизации является разбиение на сабтокены. Чтобы получить такие же сабтокены, что и в использованных датасетах [70, 52], токены разбиваются по CamelCase и snake_case, короткие сабтокены (до трех символов) склеиваются с последующими длинными (“HTMLLoader” разбивается как “html” и “loader”), и к длинным сабтокенам (более шести символам) применяется стемминг [78].

Для токенизации проекта Buckwheat извлекает идентификаторы, разбивает их на сабтокены для всех файлов, написанных на одном из 16 поддерживаемых языков. Сабтокенам затем сопоставляются векторы,

обученные алгоритмом FastText ранее.

3.4 Кластеризация эмбеддингов

Согласно алгоритму, описанному в главе 2, для построения тем была применена кластеризация векторных представлений, полученных при помощи FastText [72]. Поскольку количество токенов в датасете составляет около 40 миллионов, применение алгоритмов, таких как DBSCAN [61], требует слишком больших вычислительных затрат. Для кластеризации использовался Spherical K-Means [63]. Алгоритм работает так же, как и обычный K-Means [64], но работает с косинусными расстояниями вместо Евклидовых и выбирает центры кластеров на единичной сфере.

В качестве количества кластеров K было выбрано 256 вслед за предыдущими работами по тематическому моделированию в масштабах GitHub [52]. Использование автоматических алгоритмов для определения числа кластеров, таких как gap statistic [65], не дало результатов, поскольку разность между средним внутри-кластерным расстоянием для обученной и случайной модели остается монотонным при увеличении числа кластеров. В связи с этим для дальнейшего уточнения числа тем использовалась ручная разметка.

Чтобы упростить понимание тем полученных кластеров и сделать результат работы инструмента Sosed интерпретируемым, им были даны короткие текстовые описания при помощи пяти разработчиков. При разметке разработчикам были даны токены, представляющие каждый кластер: наиболее частотные токены и токены, ближайшие к центру кластера. Таблица 3.1 показывает пример из 12 размеченных кластеров, выбранных равновероятно из всего множества тем.

3.5 Поиск похожих проектов

На предыдущих шагах работы инструмента была обучена модель кластеризации, в которой каждый кластер представляется своим центральным вектором. Для каждого проекта затем подсчитывается вектор раз-

Таблица 3.1: Пример текстовых описаний и токенов с наибольшей частотой для 12 тем, выбранных случайно.

General games, character's state	**Low-level arithmetic operations**	**Random short identifiers**
who	vadd	gip
lev	vextract	mgq
ham	madd	pku
hungry	ifmt	xue
confused	neon	kue

2D Geometry	**Intercative DOM, nsIDOM, HTML**	**File operations**
vertex	htmle	file
mesh	ile	exists
vertices	aname	files
indices	atype	filename
vert	ody	directory

ML, Statistics	**Mathematics, Greek letters**	**Network, packets**
sum	prod	stats
weight	eps	cnt
dataset	fac	dropped
mean	rho	rts
weights	omega	collisions

Windows API	**WiFi, wireless drivers**	**Audio effects**
param	scan	play
iswindow	assoc	volume
idfrom	sta	sound
dlg	stype	pan
wnd	sdata	feedback

мерности K, в котором компонента C равна вероятности встретить кластера C среди токенов в файлах данного проекта. Таким образом были вычислены представления для 9 миллионов проектов, формирующих пространство поиска Sosed'a.

Для обработки новых проектов, поступающих в качестве запросов, их также требуется перевести в векторы распределений по темам. Для этого код всех файлов токенизируется так, как описано в разделе 3.3. Затем для токенов определяются кластеры при помощи обученной модели Spherical K-Means. Для ускорения работы используются предподсчитанные индексы кластеров для 40 миллионов известных токенов. Sosed не вычисляет векторные представления для токенов, не встретившихся ранее, по двум причинам. Во-первых, их число оказывается небольшим, поскольку датасет для поиска содержит 40 миллионов уникальных токенов. Во-вторых, поскольку датасет для поиска был собран в 2017 году, пропущенные токены могут соответствовать библиотекам и технологиям, которые появились после этапа сбора датасета. Для таких токенов использование эмбеддингов N-грамм из модели FastText может дать некорректный результат, неправильно отображающий их семантику.

Для поиска похожих проектов в референсном датасете реализованы два подхода, основанные на сравнении распределений тем. Первый из них это поиск ближайшего репозитория по косинусному расстоянию или же, что то же самое, обладающего максимальной косинусной близостью. Косинусная близость равна скалярному произведению между нормализованными векторами распределений тем для пары репозиториев:

$$cosine\_similarity(P_Q, P_R) = \sum_{c \in Clusters} \frac{P_Q(c)}{||P_Q||} \cdot \frac{P_R(c)}{||P_R||},$$

где P_Q и P_R — это распределения тем для проектов в запросе и в поисковом датасете соответственно. Вторым способом поиска похожих распределений является вычисление расстояния Кульбака-Лейблера [71] или KL-дивергенции. KL-дивергенция — это неотрицательный функционал, являющийся несимметричной мерой удаленности двух распределе-

ний. Если он равен нулю, то распределения совпадают почти наверное. В случае дискретных распределений, которыми являются интересующие нас распределения токенов в проектах по темам, KL-дивергенция может быть записана следующим образом:

$$D_{KL}(P_Q||P_R) = \sum_{c \in Clusters} P_Q(c) \log \frac{P_Q(c)}{P_R(c)}$$

Для поиска репозитория с наиболее похожим распределением требуется найти P_R, минимизирующий данный функционал. Преобразуем данную формулу:

$$\min_{P_R} \sum_{c \in Clusters} P_Q(c) \log \frac{P_Q(c)}{P_R(c)} =$$

$$\min_{P_R} \sum_{c \in Clusters} P_Q(c) \log P_Q(c) - P_Q(C) \log P_R(c) =$$

$$\min_{P_R} \sum_{c \in Clusters} -P_Q(C) \log P_R(c) =$$

$$\max_{P_R} \sum_{c \in Clusters} P_Q(C) \log P_R(c)$$

Последняя формула может быть представлена в виде скалярного произведения между нормализованным вектором P_Q и нормализованным логарифмом вектора P_R. Таким образом, как и в случае косинусной близости, задача поиска похожих репозиториев может быть сведена к максимизации скалярного произведения.

Для поиска векторов, максимизирующих скалярное произведение с заданным вектором, в инструменте Sosed используется библиотека Faiss [72]. Faiss сохраняет векторы в виде индексной структуры, к которой затем можно совершать запросы. Данная структура практически не требует лишней памяти, что позволяет эффективно искать среди 9 миллионов векторов.

3.6 Функциональность инструмента

С точки зрения пользователя Sosed является консольным приложением, написанным на языке Python, для работы которого требуется скачать репозиторий с инструментом на GitHub [68] и произвести настройку вводом нескольких команд, описанных в файле README. Также можно скачать Docker-версию инструмента, которая не требует дополнительной настройки. Функциональность инструмента не ограничивается непосредственной выдачей схожих по тематике проектов. Вывод инструмента является интерпретируемым благодаря использованию факторов на основе тем. Также при поиске допускается фильтрация по популярности проекта (числу звезд на GitHub) и основному языку.

Чтобы интерпретировать предсказания Sosed использует особенности скалярного произведения, на основании максимизации которого происходит поиск. Поскольку скалярное произведение равно сумме произведений для компонент векторов, для каждой темы можно определить вклад, который она вносит в итоговый результат. В случае косинусной близости вкладом темы C является произведение $P_R(C)P_Q(C)$, в случае KL-дивергенции — $P_Q(C)\log P_R(C)$. При передаче ключа "–explain" для каждого проекта в выдаче Sosed выведет список тем, которые внесли наибольший вклад в его решение, и токены из репозитория-запроса, которые относятся к данным темам. Пример работы с ключом "–explain" для проекта TensorFlow показан на Рисунке 3.2.

Также Sosed дает возможность фильтровать проекты в выдаче по количеству звезд на GitHub и основному языку. Для сбора этой информации был использован GHTorrent [79]— проект, занимающийся мониторингом всех событий, происходящих на GitHub. С его помощью была извлечена информация о количестве звезд и языке для 9 миллионов проектов, составляющих пространство поиска. Благодаря этому Sosed способен искать аналоги популярных библиотек в других языках. Например, при поиске проектов, похожих на TensorFlow[1], с фильтрацией

[1]https://github.com/tensorflow/tensorflow/

Query project: github.com/tensorflow/tensorflow

Output: github.com/dmlc/xgboost | similarity = 0.8710

Intersecting topics:

intersection = 0.21 | Data, data types, read-write |
input (139925), output (110979), data (81043), std (71129), from (17618)

intersection = 0.21 | Machine learning and statistics |
tensor (129095), tensorflow (45720), dataset (23191), outputs (15761), weights (15733)

intersection = 0.11 | Common programming words |
get (96002), type (86532), value (58794), string (57437), set (38336)

intersection = 0.04 | Data structures |
array (49665), index (34666), values (21846), all (21687), list (21408)

intersection = 0.04 | Memory management |
size (77894), ptr (67072), new (19412), memory (17057), allocator (6217)

Рис. 3.2: Пример объяснения рекомендации инструмента Sosed. В качестве запроса выступал проект TensorFlow, в качестве похожего проекта была найдена библиотека XGBoost. Для пяти тем, которые больше всего повлияли на результат, выведены их вклад, название темы и наиболее частые токены в репозитории-запросе, которые к ним относятся.

по языку Haskell на первом месте оказывается проект DNNGraph[2] — предметно-ориентированный язык для написания нейронных сетей.

3.7 Оценка качества работы инструмента Sosed

Единственным способом для оценки качества работы алгоритмов поиска похожих проектов в предыдущих работах являлся опрос разработчиков [40, 41, 42]. Для оценки качества работы инструмента Sosed был выбран набор из 94 проектов на GitHub, включающий в себя проекты с наибольшим числом звезд для разных языков. Результаты работы инструмента для этих проектов при поиске по косинусной близости и KL-дивергенции доступен на GitHub [68]. Например, среди пяти самых похожих проектов для TensorFlow лежат фреймворки для машинного обучения и глубокого обучения. Для проекта Bitcoin[3] Sosed нашел другие криптовалюты с открытым исходным кодом. Среди проектов, похожих на Python[4], лежит Brython[5], реализация Python для запуска в браузере.

В большинстве предыдущих работ предложенные подходы ориентированы на работу с Java-проектами, в то время как Sosed ищет похожие среди проектов на 16 языках. Также большая часть опубликованных ранее походов не имеет доступной реализации, поэтому прямое сравнение с ними не представляется возможным. Единственным инструментом, имеющим доступную реализацию и работающим с произвольными проектами, является Gazer [80], ищущий похожие проекты на основании пересечения множеств разработчиков, которые поставили проектам звезды на GitHub. Из-за использования информации о звездах инструмент может работать только со сравнительно популярными репозиториями на GitHub.

Чтобы получить численную оценку качества инструмента, были привлечены четверо разработчиков, которых просили оценить релевант-

[2]https://github.com/ajtulloch/dnngraph/
[3]https://github.com/bitcoin/bitcoin/
[4]https://github.com/python/cpython/
[5]https://github.com/brython-dev/brython/

ность предсказанных проектов для инструментов Sosed и Gazer от 1 до 5. Для оценки релевантности использовалась следующая инструкция:

- 5: Проект в выдаче релевантен — тема проекта совпадает с запросом (например, это две IDE или два языка программирования).
- 4: Проект в выдаче скорее релевантен — проекты относятся к одной доменной области (например, фреймворк для машинного обучения и реализация конкретной модели с его помощью), используют схожий стек технологий.
- 3: Релевантность проекта неоднозначна — может присутствовать сходство в доменной области или стеке используемых технологий, но есть и значительные различия.
- 2: Проект скорее не релевантен — тема и стек технологий значительно отличаются, но есть небольшие сходства.
- 1: Проект в выдаче совсем не релевантен — между проектами нет заметных сходств.

Для разметки репозиториев были подсчитаны средняя релевантность предсказаний в top-1 и в top-5. Для Sosed'a они равны 4,67 и 4,2 из 5, соответственно. Средняя релевантность в top-1 и top-5 для Gazer равна 4,3 и 3,7, соответственно. Это говорит о том, что разработчики считают проекты, которые предлагает Sosed, более релевантными, и их темы действительно совпадают с проектами в запросе. В связи с использованием информации о звездах на GitHub Gazer часто считает похожими репозитории, принадлежащие одной организации. Такие репозитории зачастую используют схожий стек технологий, но их тематика может заметно отличаться. Этим вызвано большое количество оценок 3 и 4 для предсказаний Gazer.

Поскольку Sosed допускает интерпретацию результатов выдачи, может работать с произвольными проектами на 16 языках и показывает высокую релевантность предсказаний при человеческой оценке, мож-

но заключить, что инструмент хорошо справляется с задачей поиска похожих проектов.

3.8 Выводы

Предложенный в данной работе подход к построению тем для исходного кода показал высокое качество для задачи рекомендации похожих проектов: по итогам человеческой оценки средняя релевантность для top-5 рекомендованных проектов оказалась равна 4,2 из 5, что значительно превышает результаты предыдущих работ. Предложенное решение было реализовано в качестве инструмента, называющегося Sosed и доступно в открытом доступе на GitHub [68]. Пространство поиска, используемое в инструменте, составляют 9 миллионов репозиториев, составлявших все уникальные репозитории на GitHub в 2017 году.

Предсказания инструмента являются интерпретируемыми: для каждого проекта в выдаче Sosed определяет темы, которые больше всего повлияли на решение. Темы в свою очередь имеют краткие текстовые описания, данные им вручную при помощи предложенного алгоритма разметки на основе анализа дендрограмы. Также инструмент может фильтровать проекты, среди которых осуществляется поиск, по популярности и основному языку. Это позволяет искать аналоги интересующих библиотек или проектов в других языках, что может быть полезно при смене языка программирования.

В то время, как предложенный подход использует только информацию из исходного кода, в него также можно интегрировать информацию из других источников, например, текстовых описаний. Для этого можно сконкатенировать вектор распределения тем в коде с векторами, содержащими другую информацию (например, аналогичное распределение тем в описаниях проектов). Также следующим важным шагом в развитии инструмента является сбор нового датасета референсных проектов.

Статья про предложенный инструмент была опубликована на конференции Automated Software Engineering 2020 (Core A) [69].

4 Рекомендация разработчиков для исправления ошибок

В главе 3 было описано применение предложенного в работе алгоритма *Code2Topic* для извлечения интерпретируемых факторов из кода в задаче поиска похожих проектов. В этой задаче факторы извлекались из файлов с исходным кодом и распределения тем строились на уровне целого проекта. Чтобы проверить применимость факторов для кода другой гранулярности, также было протестировано использование тематических факторов в задаче рекомендации программистов для исправления ошибок.

В крупных программных проектах для отслеживания ошибок, обнаруженных пользователями или разработчиками и пожеланий по улучшению проекта используются баг-трекеры или issue-трекеры. Примерами таких систем являются YouTrack, Jira, Bugzilla и многие другие. Процесс взаимодействия с подобными системами обычно выглядит следующим образом: пользователь (автор) создаёт текстовое описание ошибки (issue), затем находится разработчик (или же его находит инженер поддержки), который будет заниматься решением проблемы, в ходе обсуждения с пользователем уточняются детали ошибки, при необходимости ответственный за исправление программист меняется, пока наконец проблему не устранят. Для ускорения процесса исправления ошибок требуется как можно быстрее выбрать правильного разработчика для решения конкретного issue.

В данной главе описывается алгоритм *Dev2Topic*, расширяющий ранее предложенный в главе 2 алгоритм *Code2Topic* и позволяющий извлечь из истории разработки проектов интерпретируемые факторы, отражающие экспертизу каждого программиста. Полученные факторы затем используются в модели XGBoost, решающей задачу бинарной классификации для пар (issue, разработчик) и выдающей вероятность того, что разработчик подходит для решения конкретного issue. Использование полученных факторов позволило улучшить точность по сравнению с ранее разработанной моделью с 68% до 75%.

4.1 *Dev2Topic*: получение тематических факторов о разработчиках

Чтобы определить, какой разработчик должен быть назначен для решения issue, требуется представить экспертизу разработчиков в виде факторов, которые затем можно использовать в моделях машинного обучения. Для этого был предложен алгоритм *Dev2Topic*, являющийся адаптацией *Code2Topic* для работы с кодом отдельных программистов. *Dev2Topic* определяет области экспертизы программистов по фрагментам кода, который они писали в доступных для анализа проектах. Единственным требованием к анализируемым проектам является использование системы контроля версий.

Схема работы алгоритма *Dev2Topic* представлена на Рисунке 4.1. Алгоритм получает на вход множество проектов, которые будут проанализированы. Для каждого проекта из системы контроля версий извлекается история всех изменений (коммитов), которые программисты делали на протяжении разработки. Для каждого коммита определяется его автор и множество измененных файлов. Версии файлов после изменений составляют корпус кода, к которому затем применяется алгоритм *Code2Topic* для построения тематической модели. Изменения файлов в коммитах токенизируются, и из них выделяются токены, которые в рамках коммита добавил его автор. Поскольку *Code2Topic* сопоставляет темы отдельным токенам, для каждого автора можно вычислить распределение тем в его коммитах, сделанных до определенного момента времени. Результатом работы *Dev2Topic* является построение зависимости распределений тем от времени для каждого программиста, принявшего участие в разработке анализируемых проектов.

Dev2Topic был реализован в виде инструмента, доступного на GitHub [81]. На данный момент инструмент работает с системой контроля версий Git, поскольку она используется на наиболее популярных платформах хостинга репозиториев, таких как GitHub и GitLab. На вход инструмент получает список ссылок на проекты, которые затем скачиваются на диск. Затем при помощи библиотеки PyDriller [82],

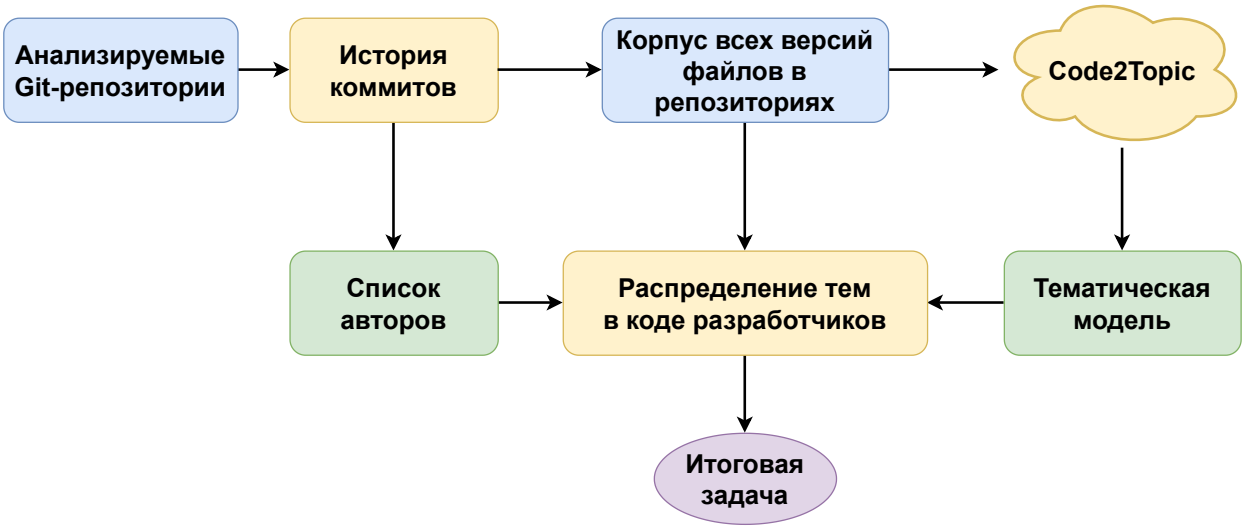

Рис. 4.1: Схема работы алгоритма *Dev2Topic*.

представляющей набор утилит для работы с Git-репозиториями в Python, он проходит по истории коммитов в основной ветке каждого репозитория и для каждого коммита выделяет следующую информацию:

- хеш коммита — строка, однозначно его определяющая в рамках проекта;
- никнейм и e-mail автора коммита;
- момент времени, в который коммит был сделан;
- пути до всех измененных файлов;
- номера строк, добавленных автором в новой версии каждого файла.

Версии файлов после каждого изменения записываются на диск в файлы "$projectName/commitStart/commitEnd/filePath\_fileName$", где $projectName$ — название проекта, $commitStart$ — первые два символа в хеше коммита, $commitEnd$ — оставшиеся символы в хеше коммита, $filePath$ — путь до измененного файла в проекте, в котором переходы между папками заменены на символ _. Такая структура позволяет избежать появления папок, содержащих десятки и сотни тысяч других файлов или папок, которые могли бы возникнуть для репозиториев,

содержащих сотни тысяч коммитов. Для каждого проекта также сохраняется словарь, содержащий в качестве ключей пары из никнейма и электронного адреса, которыми были подписаны коммиты, а в качестве значений — список кортежей из хеша коммита, момента времени отправки коммита, путей к измененным файлам и номеров строк, которые в них были добавлены.

После того, как на диске сохранены все измененные файлы, по ним строится тематическая модель при помощи *Code2Topic*. Для токенизации файлов используется Buckwheat [73], разработанная в рамках этой работы библиотека для токенизации исходного кода на 16 наиболее популярных языках программирования, подробно описанная в разделе 3.3. Итогом работы Buckwheat в данном случае становится файловая структура, совпадающая с той, что была дана на вход, но на этот раз содержимое всех файлов разбито на токены с сохранением положения токенов по строкам. Для деления токенов на сабтокены используется библиотека YouTokenToMe (YTTM) [83], эффективно реализующая алгоритм BPE. Для построения эмбеддингов сабтокенов используется алгоритм GloVe, а для построения эмбеддингов токенов — усреднение векторов для сабтокенов. Результатом работы *Code2Topic* является построенная тематическая модель на основе кластеризации эмбеддингов токенов, дендрограма тем и, при использовании ручной разметки, текстовое описание для каждой темы.

Следующим шагом является восстановление корректного множества авторов. В Git разработчики могут указывать разные никнеймы и электронные адреса (псевдонимы): менять их время от времени, использовать разные псевдонимы на рабочем и домашнем компьютере. Таким образом, разные псевдонимы могут соответствовать одному и тому же программисту, поэтому возникает необходимость определить группы коммитов, сделанных одним автором. Для этого в текущей версии инструмента реализован следующий алгоритм: сперва он удаляет слишком частые никнеймы и почты (например, “unknown”, “noreply@website.com”), затем строит двудольный граф никнеймов и электронных почт, где пара соединена ребром тогда, когда она встречалась

в рамках одного коммита. В полученном графе выделяются компоненты связности, которые и соответствуют отдельным авторам.

После того, как построена тематическая модель для токенов и определено корректное множество авторов, инструмент проходит по сохраненным ранее данным об изменениях, сделанных под каждым псевдонимом, подсчитывает распределение тем в коммитах на основе обученной тематической модели и токенизированных файлов, агрегирует эту информацию по всем псевдонимам одного автора и получает зависимость распределения тем в коде каждого программиста от времени.

Полученный вектор распределения тем, зависящий от времени, отражает экспертизу, которой обладает конкретный программист. Например, если в распределении преобладают темы, связанные с работой с графическими интерфейсами, то этот автор может быть хорошим кандидатом для решения проблем, связанных с ошибками в графической части приложения, но вряд ли должен заниматься проблемами, связанными с серверной частью.

4.2 Модель на основе *Dev2Topic* для рекомендации разработчиков

Большинство существующих подходов к рекомендации разработчиков для исправления ошибок решают задачу как мультиклассовую классификацию [13, 11]. Входными данными могут являться описание ошибки, ее название, автор, оставленные комментарии. Предсказываемыми классами являются разработчики, известные модели. Минусом такого подхода на практике является то, что в процессе развития проекта появляются новые разработчики, в связи с чем модель приходится обучать заново. Более того, поскольку такие модели обучаются на уже исправленных ошибках, для них существует проблема “холодного старта”: пока разработчик не исправил достаточное количество ошибок, модель не может получить информацию о нём, а значит не может рекомендовать его.

Альтернативным вариантом постановки задачи является бинарная

классификация: в этом случае модель получает на вход данные об issue (например, описание, информация об авторе) и о конкретном разработчике, а на выходе предсказывает вероятность того, что данный разработчик подходит для решения представленной проблемы. В зависимости от используемых факторов, представляющих программиста, такой подход может частично решить проблему холодного старта, поскольку информация о программисте может быть собрана даже без его активного участия в исправлении ошибок. Также бинарная постановка задачи значительно увеличивает количество данных, доступных для обучения, поскольку число пар (описание ошибки, программист) значительно превышает число описаний ошибок, но при этом датасет оказывается несбалансированным. В отличие от моделей мультиклассовой классификации, факторы, полученные *Dev2Topic*, можно напрямую использовать в моделях бинарной классификации.

Два подхода можно совместить: предварительно сделать фильтрацию программистов при помощи модели мультиклассовой классификации, тем самым сократив пространство поиска, а для выбранных программистов решить задачу бинарной классификации и выбрать наиболее подходящих.

В качестве бейзлайна для решения задачи в бинарной постановке была выбрана разработанная ранее в JetBrains модель на основе XGBoost [84], которая получает на вход следующие данные для пары из программиста и сообщения об ошибке:

- какой программист чаще всего исправлял ошибки, присланные автором данного сообщения;
- какая доля ошибок, исправленных данным программистом, была прислана автором данного сообщения;
- какую долю ошибок, присланных автором данного сообщения, исправил данный программист;
- какую долю ошибок с таким же приоритетом исправил данный программист;

- какую долю ошибок с таким же типом исправил данный программист;

- какое количество слов, характерных для данного программиста (чаще всего встречавшихся в исправленных им сообщениях об ошибках), содержит данное сообщение об ошибке.

В модель были добавлены факторы, отражающие экспертизу программистов в определенных темах, извлеченные ранее при помощи алгоритма *Dev2Topic*. Чтобы отразить информацию о содержимом ошибки, для каждой темы были вычислены связанные с ней слова из описаний и в качестве факторов добавлены их частоты. Для определения связанных с темой слов по тренировочной выборке для каждой пары из слова в описаниях и темы было подсчитано отношение среднего значения темы в распределении у решившего проблему программиста, когда слово присутствует в описании проблемы и когда не присутствует. Для каждой темы были выбраны 50 слов, для которых отношение слов было максимальным. К примеру, для темы описывающей работу с системами контроля версий одним из связанных слов стало слово “commit”.

4.3 Данные

Для проверки работоспособности предложенного подхода использовался исторический датасет из 9700 issue, открытых в системе YouTrack компании JetBrains в период с 2017 по 2020 год. Исправлением ошибок занималось 111 программистов. Для каждой issue в датасете представлена следующая информация: название, описание, идентификатор автора, время открытия, разработчик, который был назначен ответственным в момент закрытия issue. Датасет был разбит по времени на обучающую и тестовую выборку в отношении 90 к 10 соответственно.

Для извлечения факторов, отражающих экспертизу программистов был использован разработанный инструмент, описанный в разделе 4.1. С его помощью были обработаны все репозитории организации JetBrains

на GitHub[6], не являющиеся форками, что составило 438 проектов. Для каждого репозитория была проанализирована история коммитов и извлечены изменения, сделанные под каждым псевдонимом (пара из никнейма и электронной почты). Для определения псевдонимов, принадлежащих каждому из 111 разработчиков, занимавшихся разрешением issue в анализируемом датасете, я воспользовался внутренней системой JetBrains, позволяющей узнать адреса электронной почты, которые программисты указывали в своих коммитах. В результате для каждого из 111 программистов было получено распределение тем в его коде на момент публикации каждого из 9700 сообщений об ошибках.

4.4 Результаты

Для сравнения были выбраны модели, описанные в разделе 1.3: модель на основе tf-idf [47], более современный подход на основе CNN, использующий предобученные эмбеддинги [13], а также упрощенную версию последней модели, в которой CNN заменяется на многослойный перцептрон. Результаты сравнения предложенного подхода с другими моделями показаны в Таблице 4.1. Предложенная модель, представляющая собой модель XGBoost, получающую на вход признаки, использовавшиеся в бейзлайне, распределения тем в коде разработчика и частоты наиболее значимых слов для каждой темы в описании ошибки, показала улучшение в точности с 68% до 75%.

Модели на основе мультиклассовой классификации по описанию ошибки показали точность ниже, чем бейзлайн и его модификация, предложенная в данной работе. Важной для достижения высокого качества оказалась информация, связывающая автора сообщения об ошибке, программиста и содержание ошибки. Использование факторов, отражающих экспертизу программистов, позволило модели меньше опираться на то, какой разработчик чаще всего работал с автором сообщения, и чаще предлагать программиста, основываясь на его экспертизе относительно сути ошибки.

[6]JetBrains GitHub: https://github.com/JetBrains

Таблица 4.1: Сравнение моделей рекомендации разработчиков для исправления ошибок.

Модель	Точность (%)
Time TF-IDF [47]	42,3
Word2Vec + FCN [13]	44,7
Word2Vec + CNN [13]	52,6
JetBrains Baseline	68,4
JetBrains Baseline + *Dev2Topic*	**75,1**

4.5 Выводы

В рамках данной работы был предложен алгоритм *Dev2Topic*, позволяющий отразить экспертизу программистов в разных областях при помощи факторов, извлеченных на основе тематического моделирования. Алгоритм был реализован в качестве самостоятельного инструмента и доступен на GitHub [81]. Использование полученных факторов позволило улучшить точность в задаче рекомендации разработчиков для исправления ошибок с 68% до 75%. Факторы являются интерпретируемыми, что позволяет сделать результаты работы модели объяснимыми. Предложенный алгоритм построения факторов для разработчиков также может быть применен в других задачах, например, рекомендации напарников для ревью кода.

Заключение

В рамках данной работы были достигнуты следующие результаты:

- Разработан алгоритм *Code2Topic* для извлечения тем из исходного кода. Алгоритм учитывает особенности словаря, используемого в коде: большой размер словаря, использование новых терминов, склеивание целых фраз в токены через CamelCase и snake_case. *Code2Topic* извлекает из кода идентификаторы и разбивает их на сабтокены при помощи BPE. Затем для сабтокенов обучаются векторные представления при помощи алгоритмов GloVe или FastText. Векторы для токенов вычисляются усреднением векторов их частей. В пространстве токенов выделяются темы путем кластеризации векторов алгоритмом Spherical K-Means. Полученная модель сопоставляет темы идентификаторам, встречающимся в коде, благодаря чему для произвольного фрагмента кода можно определить распределение тем в нём. Распределение тем затем можно использовать для представления кода в моделях машинного обучения.

- Для упрощения интерпретации тем было предложено их представление в виде дендрограмы, строящейся путем агломеративной кластеризации центров полученных ранее кластеров. Использование дендрограмы также позволяет облегчить процесс разметки тем текстовыми описаниями и одновременно выбрать число тем: разметчикам предлагают давать описания не отдельным темам, а их бинарным делениям в дендрограме. Процесс деления кластеров останавливается, когда разметчик не может дать описание полученным дочерним темам. Тем самым количество тем определяется в процессе разметки, что гарантирует осмысленность всех полученных тем.

- На основе *Code2Topic* был предложен *Dev2Topic*, алгоритм, позволяющий представить экспертизу разработчиков в виде распределения тем в коде, который они писали в доступных для анали-

за проектах. Для этого анализируется история проектов: из нее извлекаются отдельные коммиты, для коммитов каждого автора определяются изменения, которые он внёс, а для изменений строится распределение темы при помощи подхода *Code2Topic*.

- Разработан Sosed, инструмент для поиска похожих репозиториев на основе предложенного алгоритма *Dev2Topic*. Sosed позволяет найти проекты, похожие на проект в запросе, среди множества из 9 миллионов проектов с исходным кодом на GitHub. Для этого он строит распределение тем в проектах и ищет похожие распределения по косинусной близости или KL-дивергенции. Помимо этого инструмент позволяет фильтровать выдачу по популярности и языкам. По результатам ручной разметки релевантности для 92 популярных проектов средняя релевантность выдачи Sosed'a в top-5 составляет 4,2, что говорит о высоком качестве рекомендаций.

- Предложенный подход для извлечения факторов, отражающих экспертизу программистов, был использован в задаче рекомендации разработчиков для исправления ошибок. Извлеченные факторы были использованы в модели XGBoost для улучшения точности рекомендации разработчика с 68% до 75% для датасета из 9700 багов из системы YouTrack.

Дальнейшая работа возможна в следующих направлениях:

- Предложенные алгоритмы *Code2Topic* и *Dev2Topic* можно улучшить, используя другие методики построения векторных представлений токенов. Например, использование контекстуальных эмбеддингов, таких как ELMo, может позволить решить проблему многозначности токенов, определяя точный смысл токена на основании контекста его использования.

- Предложенный инструмент для поиска похожих репозиториев можно улучшить, собрав актуальный датасет проектов для поиска на

GitHub и разработав способ обновления датасета в режиме реального времени. Это позволит итеративно обновлять пространство поиска по мере появления новых библиотек и технологий, что должно улучшить качество рекомендаций.

- *Code2Topic* можно также использовать для получения факторов из кода другой гранулярности: отдельных файлов или методов. Полезность получаемых факторов можно проверить в других задачах, связанных с исходным кодом: локализации дефектов, определении имен методов.

- Подход *Dev2Topic* можно применить в других задачах, требующих представления экспертизы разработчиков, например, рекомендации напарников для ревью кода.

Список литературы

[1] Corporation Evans Data. Worldwide Developer Population and Demographic Study 2020 Volume 2. — 2020. — URL: https://evansdata.com/reports/viewRelease.php?reportID=9.

[2] Hellendoorn Vincent J., Proksch Sebastian, Gall Harald C., Bacchelli Alberto. When Code Completion Fails: A Case Study on Real-World Completions // Proceedings of the 41st International Conference on Software Engineering. — ICSE '19. — IEEE Press, 2019. — P. 960–970. — URL: https://doi.org/10.1109/ICSE.2019.00101.

[3] Cosma Georgina, Joy Mike. An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis // IEEE Transactions on Computers. — 2012. — Vol. 61, no. 3. — P. 379–394.

[4] Burrows Steven, Tahaghoghi S., Zobel J. Efficient plagiarism detection for large code repositories // Software - Practice and Experience. — 2007. — Vol. 37. — P. 151–175.

[5] Bavota Gabriele, Oliveto Rocco, Gethers Malcom et al. Methodbook: Recommending Move Method Refactorings via Relational Topic Models // IEEE Trans. Softw. Eng. — 2014. — July. — Vol. 40, no. 7. — P. 671–694. — URL: https://doi.org/10.1109/TSE.2013.60.

[6] Kurbatova Zarina, Veselov Ivan, Golubev Yaroslav, Bryksin Timofey. Recommendation of Move Method Refactoring Using Path-Based Representation of Code // Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. — ICSEW'20. — New York, NY, USA : Association for Computing Machinery, 2020. — P. 315–322. — URL: https://doi.org/10.1145/3387940.3392191.

[7] Rahman Mohammad Masudur, Roy Chanchal K., Collins Jason A. CORRECT: Code Reviewer Recommendation in GitHub Based on Cross-Project and Technology Experience // 2016 IEEE/ACM 38th

International Conference on Software Engineering Companion (ICSE-C). — 2016. — P. 222–231.

[8] Hannebauer Christoph, Patalas Michael, Stünkel Sebastian, Gruhn Volker. Automatically Recommending Code Reviewers Based on Their Expertise: An Empirical Comparison // Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. — ASE 2016. — New York, NY, USA : Association for Computing Machinery, 2016. — P. 99–110. — URL: https://doi.org/10.1145/2970276.2970306.

[9] Xia Xin, Lo David, Wang Xinyu, Zhou Bo. Accurate developer recommendation for bug resolution // 2013 20th Working Conference on Reverse Engineering (WCRE). — 2013. — P. 72–81.

[10] Zhang Tao, Yang Geunseok, Lee Byungjeong, Lua Eng Keong. A Novel Developer Ranking Algorithm for Automatic Bug Triage Using Topic Model and Developer Relations // 2014 21st Asia-Pacific Software Engineering Conference. — Vol. 1. — 2014. — P. 223–230.

[11] Lee Sun-Ro, Heo Min-Jae, Lee Chan-Gun et al. Applying Deep Learning Based Automatic Bug Triager to Industrial Projects // Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. — ESEC/FSE 2017. — New York, NY, USA : Association for Computing Machinery, 2017. — P. 926–931. — URL: https://doi.org/10.1145/3106237.3117776.

[12] Sarkar Aindrila, Rigby Peter C., Bartalos Béla. Improving Bug Triaging with High Confidence Predictions at Ericsson // 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). — 2019. — P. 81–91.

[13] Guo Shikai, Zhang X., Yang X. et al. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network // Neural Processing Letters. — 2020. — Vol. 51. — P. 2589–2606.

[14] Alon Uri, Levy Omer, Yahav Eran. code2seq: Generating Sequences from Structured Representations of Code // CoRR. — 2018. — Vol. abs/1808.01400. — 1808.01400.

[15] Allamanis Miltiadis, Barr Earl T., Bird Christian, Sutton Charles. Learning Natural Coding Conventions // Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. — FSE 2014. — New York, NY, USA : Association for Computing Machinery, 2014. — P. 281–293. — URL: https://doi.org/10.1145/2635868.2635883.

[16] Yin Pengcheng, Deng Bowen, Chen Edgar et al. Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow // International Conference on Mining Software Repositories. — MSR. — ACM, 2018. — P. 476–486.

[17] Kovalenko Vladimir, Tintarev Nava, Pasynkov Evgeny et al. Does Reviewer Recommendation Help Developers? // IEEE Transactions on Software Engineering. — 2020. — Vol. 46, no. 7. — P. 710–731.

[18] Gelman Ben, Hoyle Bryan, Moore Jessica et al. A Language-Agnostic Model for Semantic Source Code Labeling // Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis. — MASES 2018. — New York, NY, USA : Association for Computing Machinery, 2018. — P. 36–44. — URL: https://doi.org/10.1145/3243127.3243132.

[19] McBurney Paul W., Liu Cheng, McMillan Collin, Weninger Tim. Improving Topic Model Source Code Summarization // Proceedings of the 22nd International Conference on Program Comprehension. — ICPC 2014. — New York, NY, USA : Association for Computing Machinery, 2014. — P. 291–294. — URL: https://doi.org/10.1145/2597008.2597793.

[20] Saeidi Amir, Hage Jurriaan, Khadka Ravi, Jansen Slinger. ITMViz:

Interactive Topic Modeling for Source Code Analysis. — 2015. — 05. — P. 295–298.

[21] Chen Tse-Hsun Peter, Thomas S.W., Nagappan M., Hassan Ahmed E. Explaining software defects using topic models. — 2012. — 06. — P. 189–198.

[22] Wang Yaojing, Yao Yuan, Tong Hanghang et al. Bug Localization via Supervised Topic Modeling // 2018 IEEE International Conference on Data Mining (ICDM). — 2018. — P. 607–616.

[23] Thomas Stephen W., Hemmati Hadi, Hassan Ahmed E., Blostein Dorothea. Static Test Case Prioritization Using Topic Models // Empirical Softw. Engg. — 2014. — February. — Vol. 19, no. 1. — P. 182–212. — URL: https://doi.org/10.1007/s10664-012-9219-7.

[24] Kuhn Adrian, Erni David, Loretan Peter, Nierstrasz Oscar. Software Cartography: Thematic Software Visualization with Consistent Layout // J. Softw. Maint. Evol. — 2010. — April. — Vol. 22, no. 3. — P. 191–210.

[25] Liu Xiangyue, Sun Xiaobing, Li Bin, Zhu Junwu. PFN: A novel program feature network for program comprehension // 2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS). — 2014. — P. 349–354.

[26] Savage Trevor, Dit Bogdan, Gethers Malcom, Poshyvanyk Denys. TopicXP: Exploring topics in source code using Latent Dirichlet Allocation // 2010 IEEE International Conference on Software Maintenance. — 2010. — P. 1–6.

[27] Poshyvanyk Denys, Gethers Malcom, Marcus Andrian. Concept Location Using Formal Concept Analysis and Information Retrieval // ACM Trans. Softw. Eng. Methodol. — 2013. — February. — Vol. 21, no. 4. — 34 p. — URL: https://doi.org/10.1145/2377656.2377660.

[28] Dit Bogdan, Revelle Meghan, Poshyvanyk Denys. Integrating Information Retrieval, Execution and Link Analysis Algorithms to Improve Feature Location in Software // Empirical Softw. Engg. — 2013. — April. — Vol. 18, no. 2. — P. 277–309. — URL: https://doi.org/10.1007/s10664-011-9194-4.

[29] Papadimitriou Christos H., Raghavan Prabhakar, Tamaki Hisao, Vempala Santosh. Latent Semantic Indexing: A Probabilistic Analysis // Journal of Computer and System Sciences. — 2000. — Vol. 61, no. 2. — P. 217–235. — URL: https://www.sciencedirect.com/science/article/pii/S0022000000917112.

[30] Blei David M., Ng Andrew Y., Jordan Michael I. Latent Dirichlet Allocation // J. Mach. Learn. Res. — 2003. — March. — Vol. 3, no. null. — P. 993–1022.

[31] Vorontsov Konstantin, Potapenko Anna, Plavin Alexander. Additive Regularization of Topic Models for Topic Selection and Sparse Factorization // Statistical Learning and Data Sciences / Ed. by Alexander Gammerman, Vladimir Vovk, Harris Papadopoulos. — Cham : Springer International Publishing, 2015. — P. 193–202.

[32] Klema V., Laub A. The singular value decomposition: Its computation and some applications // IEEE Transactions on Automatic Control. — 1980. — Vol. 25, no. 2. — P. 164–176.

[33] Kotz Samuel, Balakrishnan Narayanaswamy, Johnson Norman. Continuous Multivariate Distributions: Models and Applications, Volume 1, Second Edition. — 2000. — 01. — Vol. 1.

[34] Borg Ingwer, Groenen Patrick. Modern Multidimensional Scaling: Theory and Applications // Journal of Educational Measurement. — 2006. — 06. — Vol. 40. — P. 277 – 280.

[35] Chang Jonathan, Blei David. Relational Topic Models for Document Networks. // Journal of Machine Learning Research - Proceedings Track. — 2009. — 01. — Vol. 5. — P. 81–88.

[36] Hu Yuening, Boyd-Graber Jordan, Satinoff Brianna. Interactive Topic Modeling // Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. — Portland, Oregon, USA : Association for Computational Linguistics, 2011. — June. — P. 248–257. — URL: https://www.aclweb.org/anthology/P11-1026.

[37] Kleinberg Jon M. Authoritative Sources in a Hyperlinked Environment // J. ACM. — 1999. — September. — Vol. 46, no. 5. — P. 604–632. — URL: https://doi.org/10.1145/324133.324140.

[38] Brin Sergey, Page Lawrence. The Anatomy of a Large-Scale Hypertextual Web Search Engine // Computer Networks. — 1998. — Vol. 30. — P. 107–117. — URL: http://www-db.stanford.edu/~backrub/google.html.

[39] Mens Tom, Serebrenik Alexander, Cleve Anthony. Evolving Software Systems. — Springer Publishing Company, Incorporated, 2014. — ISBN: 364245397X.

[40] McMillan Collin, Grechanik Mark, Poshyvanyk Denys. Detecting Similar Software Applications // Proceedings of the 34th International Conference on Software Engineering. — ICSE '12. — IEEE Press, 2012. — P. 364–374.

[41] Zhang Yun, Lo David, Kochhar Pavneet Singh et al. Detecting similar repositories on GitHub. — 2017. — 02. — P. 13–23.

[42] Thung F., Lo D., Jiang L. Detecting similar applications with collaborative tagging // 2012 28th IEEE International Conference on Software Maintenance (ICSM). — 2012. — P. 600–603.

[43] Chen Ning, Hoi Steven, Li Shaohua, Xiao Xiaokui. SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning. — 2015. — 01.

[44] Linares-Vásquez M., Holtzhauer A., Poshyvanyk D. On automatically detecting similar Android apps // 2016 IEEE 24th International Conference on Program Comprehension (ICPC). — 2016. — P. 1–10.

[45] Li L., Bissyandé T. F., Klein J. SimiDroid: Identifying and Explaining Similarities in Android Apps // 2017 IEEE Trustcom/BigDataSE/ICESS. — 2017. — P. 136–143.

[46] Gonzalez Hugo, Stakhanova Natalia, Ghorbani Ali. DroidKin: Lightweight Detection of Android Apps Similarity. — Vol. 152. — 2014. — 09.

[47] Shokripour Ramin, Anvik John, Kasirun Zarinah M., Zamani Sima. A time-based approach to automatic bug report assignment // Journal of Systems and Software. — 2015. — Vol. 102. — P. 109–122. — URL: https://www.sciencedirect.com/science/article/pii/S0164121214002933.

[48] TF–IDF // Encyclopedia of Machine Learning / Ed. by Claude Sammut, Geoffrey I. Webb. — Boston, MA : Springer US, 2010. — P. 986–987. — ISBN: 978-0-387-30164-8. — URL: https://doi.org/10.1007/978-0-387-30164-8_832.

[49] Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks // Commun. ACM. — 2017. — May. — Vol. 60, no. 6. — P. 84–90. — URL: https://doi.org/10.1145/3065386.

[50] Mikolov Tomas, Chen Kai, Corrado Greg S., Dean Jeffrey. Efficient Estimation of Word Representations in Vector Space. — 2013. — URL: http://arxiv.org/abs/1301.3781.

[51] Ioffe Sergey, Szegedy Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proceedings of the 32nd International Conference on Machine Learning /

Ed. by Francis Bach, David Blei. — Vol. 37 of Proceedings of Machine Learning Research. — Lille, France : PMLR, 2015. — 07–09 Jul. — P. 448–456. — URL: http://proceedings.mlr.press/v37/ioffe15.html.

[52] Markovtsev Vadim, Kant Eiso. Topic modeling of public repositories at scale using names in source code // arXiv preprint arXiv:1704.00135. — 2017.

[53] Karampatsis Rafael-Michael, Babii Hlib, Robbes Romain et al. Big Code != Big Vocabulary: Open-Vocabulary Models for Source Code. — 2020. — 2003.07914.

[54] Babii Hlib, Janes A., Robbes R. Modeling Vocabulary for Big Code Machine Learning // ArXiv. — 2019. — Vol. abs/1904.01873.

[55] Porter M. F. An Algorithm for Suffix Stripping // Readings in Information Retrieval. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1997. — P. 313–316. — ISBN: 1558604545.

[56] Gage Philip. A New Algorithm for Data Compression // C Users J. — 1994. — February. — Vol. 12, no. 2. — P. 23–38.

[57] Kelley Henry J. Gradient theory of optimal flight paths // Ars Journal. — 1960. — Vol. 30, no. 10. — P. 947–954.

[58] Bojanowski Piotr, Grave Edouard, Joulin Armand, Mikolov Tomas. Enriching Word Vectors with Subword Information // Transactions of the Association for Computational Linguistics. — 2017. — Vol. 5. — P. 135–146.

[59] Pennington Jeffrey, Socher Richard, Manning Christopher. GloVe: Global Vectors for Word Representation // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). — Doha, Qatar : Association for Computational Linguistics, 2014. — October. — P. 1532–1543. — URL: https://www.aclweb.org/anthology/D14-1162.

[60] Heinzerling Benjamin, Strube Michael. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages // Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018). — Miyazaki, Japan : European Language Resources Association (ELRA), 2018. — May. — URL: https://www.aclweb.org/anthology/L18-1473.

[61] Ester Martin, Kriegel Hans-Peter, Sander Jörg, Xu Xiaowei. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. — KDD'96. — AAAI Press, 1996. — P. 226–231.

[62] McInnes Leland, Healy John. Accelerated Hierarchical Density Based Clustering // Data Mining Workshops (ICDMW), 2017 IEEE International Conference on / IEEE. — 2017. — P. 33–42.

[63] Hornik Kurt, Feinerer Ingo, Kober Martin, Buchta Christian. Spherical k-Means Clustering // Journal of Statistical Software. — 2012. — 09. — Vol. 50. — P. 1–22.

[64] Lloyd S. Least squares quantization in PCM // IEEE Transactions on Information Theory. — 1982. — mar. — Vol. 28, no. 2. — P. 129–137. — URL: https://doi.org/10.1109%2Ftit.1982.1056489.

[65] Tibshirani Robert, Walther Guenther, Hastie Trevor. Estimating the number of clusters in a data set via the gap statistic // Journal of the Royal Statistical Society: Series B (Statistical Methodology). — 2001. — Vol. 63, no. 2. — P. 411–423. — https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00293.

[66] Kaufman Leonard, Rousseeuw Peter. Finding Groups in Data: An Introduction to Cluster Analysis. — 2009. — 09. — ISBN: 9780470317488.

[67] Peters Matthew, Neumann Mark, Iyyer Mohit et al. Deep Contextualized Word Representations // Proceedings of the 2018 Conference of

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). — New Orleans, Louisiana : Association for Computational Linguistics, 2018. — June. — P. 2227–2237. — URL: https://www.aclweb.org/anthology/N18-1202.

[68] JetBrains Research GitHub: Sosed. — 2021. — URL: https://github.com/JetBrains-Research/sosed.

[69] Bogomolov Egor, Golubev Yaroslav, Lobanov Artyom et al. Sosed: A Tool for Finding Similar Software Projects // Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. — ASE '20. — New York, NY, USA : Association for Computing Machinery, 2020. — P. 1316–1320. — URL: https://doi.org/10.1145/3324884.3415291.

[70] Markovtsev Vadim. GitHub word2vec 120k. — https://data.world/vmarkovtsev/github-word-2-vec-120-k. — 2017.

[71] Kullback S., Leibler R. A. On Information and Sufficiency // Ann. Math. Statist. — 1951. — 03. — Vol. 22, no. 1. — P. 79–86. — URL: https://doi.org/10.1214/aoms/1177729694.

[72] Johnson Jeff, Douze Matthijs, Jégou Hervé. Billion-scale similarity search with GPUs // arXiv preprint arXiv:1702.08734. — 2017.

[73] JetBrains Research GitHub: Buckwheat. — 2021. — URL: https://github.com/JetBrains-Research/buckwheat.

[74] go-enry GitHub: enry. — 2021. — URL: https://github.com/go-enry/enry.

[75] The most popular languages of GitHub's pull requests, 1 quarter, 2020. — 2020. — URL: https://madnight.github.io/githut/#/pull_requests/2020/1.

[76] tree-sitter GitHub: tree-sitter. — 2021. — URL: https://github.com/tree-sitter/tree-sitter.

[77] Pygments: Python syntax highlighter. — 2021. — URL: https://pygments.org/.

[78] Porter Martin F. Snowball: A language for stemming algorithms. — 2001.

[79] Gousios Georgios. The GHTorrent dataset and tool suite // Proceedings of the 10th Working Conference on Mining Software Repositories. — MSR '13. — Piscataway, NJ, USA : IEEE Press, 2013. — P. 233–236. — URL: http://dl.acm.org/citation.cfm?id=2487085.2487132.

[80] Anvaka GitHub: Gazer. — 2021. — URL: https://github.com/anvaka/gazer/.

[81] Bogomolov Egor. egor-bogomolov GitHub: Dev2Topic. — 2021. — URL: https://github.com/egor-bogomolov/topic-modeling.

[82] Spadini Davide, Aniche Maurício, Bacchelli Alberto. PyDriller: Python framework for mining software repositories // Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018. — New York, New York, USA : ACM Press, 2018. — P. 908–911. — URL: http://dl.acm.org/citation.cfm?doid=3236024.3264598.

[83] VKCOM GitHub: YouTokenToMe. — 2021. — URL: https://github.com/VKCOM/YouTokenToMe.

[84] Chen Tianqi, Guestrin Carlos. XGBoost: A Scalable Tree Boosting System // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — KDD '16. — New York, NY, USA : Association for Computing Machinery, 2016. — P. 785–794. — URL: https://doi.org/10.1145/2939672.2939785.