Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Терехов Арсений Константинович

Предсказание времени жизни строк кода

Курсовая работа

Научный руководитель:
к. т. н., доцент Брыксин Т. А.

Санкт-Петербург
2019

Оглавление

Введение

За последнее время появилось много различных подходов к выявлению ошибок в исходном коде программного обеспечения. В частности, некоторые из них для каждого файла предсказывают количество дефектов, найденных в нём за 6 месяцев после определенного коммита в системе контроля версий [2]. Другие классифицируют исходный код на правильный и склонный к дефектам [3].

Большинство из этих подходов основаны на методах машинного обучения, потому что дефекты не являются чем-то строго детерминированным, как, например, ошибки компиляции. Поэтому в статьях всегда поднимается вопрос о том, как определить, что такое дефект и с помощью какой информации можно разметить файлы, исходный код или даже целые модули двумя классами. Например, некоторые подходы используют системы отслеживания ошибок, такие как Jira и Bugzilla, чтобы связать ошибку с временем и местом её появления [2]. Также есть решения, которые сами синтезируют неправильный код, таким образом получая негативные примеры из положительных [3].

Тем не менее, понятия дефекта программного обеспечения и качества исходного кода остаётся размытым. Поэтому мы хотим предложить новый подход, который основан на идее о связи между временем жизни строк кода и количеством ошибок в нём. Чем дольше живут строки, то есть остаются неизменными, тем код лучше — то есть вероятность того, что в нем есть ошибки, меньше. При этом ошибка понимается в широком смысле — это не только то, что приводит к дефекту всей программы в целом, но и плохое качество кода.

Таким образом, инструмент, который будет предсказывать оставшееся время жизни строк кода поможет разработчикам отслеживать потенциально ошибочные места в исходном коде. С помощью него можно будет выявить опасные места в программе, которые соответствуют предсказаниям с наименьшим временем жизни. При этом время на поиск ошибок существенно сократится, потому что инструмент позволит отделить потенциально дефектные места исходного кода от исправных.

В конечном итоге, использование такого инструмента приведёт к увеличению среднего времени жизни кода, что повлечёт улучшение его качества, уменьшение ошибок в нём и повысит скорость разработки программного обеспечения.

Постановка задачи

Целью данной курсовой работы является создание инструмента, который поможет разработчикам программного обеспечения увеличить качество и среднее время жизни их кода с помощью отображения результатов предсказаний оставшегося времени жизни строчек кода. Для достижения цели были поставлены задачи в двух направлениях: получение предсказаний и разработка инструмента.

Для разработки модели, которая будет предсказывать оставшееся время жизни строк кода, были поставлены следующие задачи:

1. Исследовать подходы к выявлению дефектов программного обеспечения.
2. Формализовать понятие времени жизни строки кода.
3. Определить признаки строк кода.
4. Собрать датасет для обучения модели.
5. Обучить на полученном датасете модель и оценить результаты предсказаний.
6. Выявить лучшие признаки.

Получив решающую функцию, требуется встроить её в плагин IntelliJ IDEA [1], который будет отображать результаты предсказаний. Так как эта решающая функция должна вычисляться во время использования плагина, необходимо в это же время собирать данные для её вычисления. Также время вычисления функции для строк одного файла должно быть разумным, поэтому некоторые сложно вычислимые признаки,

которые могут быть собраны при обучении модели, использовать для предсказаний в реальном времени не представляется возможным.

Таким образом, для разработки плагина были поставлены следующие задачи:

1. Сделать функциональность отображения результатов через графический интерфейс IntelliJ IDEA.
2. Сделать автоматическое вычисление признаков строк.
3. Внедрить решающую функцию в плагин.

1. Основные понятия

В этой главе приведены определения, использованные в данной работе.

1.1. Метрики качества

Метрика качества - это показатель правдоподобности или ошибки предсказаний, получаемых решающей функцией. Для определения конкретных метрик обозначим за y значения целевой переменной, а за $\hat{y}$ предсказания модели. Так же за n обозначим размер выборки. Для задачи регрессии используются следующие метрики:

1. Среднеквадратическая ошибка (RMSE, Root Mean Square Error)

$$\mathrm{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{1}$$

2. Средняя абсолютная ошибка (MAE, Mean Absolute Error)

$$\mathrm{MAE}(y, \hat{y}) = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n} \tag{2}$$

Для задачи бинарной классификации используется метрика AUC (Area under the ROC curve).

$$\mathrm{AUC} = \text{площадь под ROC-кривой} \tag{3}$$

Значение AUC соответствует доле правильно упорядоченных классификатором пар разных классов.

1.2. Показатели важности признаков

Для отбора признаков используются различные показатели того, какой вклад вносит каждый признак в предсказания модели и их качество. В этой работе используется показатель permutation importance. Для его вычисления для конкретного признака модель сначала обучается на исходной обучающей выборке. Дальше выбирается некоторая

метрика качества, которая вычисляется на исходной тестовой выборке и на выборке, полученной из тестовой перемешиванием значения данного признака. В итоге значение permutation importance равно разнице этих двух значений метрики. При этом отрицательное значение permutation importance соответствует ухудшению качества предсказаний после перемешивания признака, а положительное улучшению. Первый случай означает то, что данный признак вносит положительный вклад в предсказание, а второй соответствует отрицательному.

2. Обзор существующих решений

На тему методов машинного обучения в области программной инженерии написано огромное множество статей. Хотя и существует много направлений исследований в данной области, но большинство из них так или иначе основаны на извлечении различной информации об исходном коде.

Сравнение классических подходов к предсказанию ошибок в исходном коде было написано в статье Марко Д'Амброса [2]. Он сравнил 6 подходов, основанных на метриках, показывающих сложность исходного кода, его изменений, количества предыдущих ошибок и других факторах. Значением целевой переменной было количество ошибок в период шести месяцев после определенного коммита репозитория. Результатом этой статьи является вычисление корреляции между факторами и количеством ошибок. Из его работы можно вынести то, что метрики изменений, которые, например, отражают различную информацию про опыт автора и текущие время жизни кусков кода, имеют хорошую корреляцию с дефектами в программном обеспечении.

Статья, которая больше всего подходит к текущей задаче по целевой переменной - это диссертация Пер Нордфорса [4], в которой он обучал многослойную нейронную сеть и случайный лес для предсказания того, сколько проживёт данный кусок кода. Время его жизни измерялось в количестве коммитов, после которых кусок кода будет удалён либо модифицирован. Также введенное им понятие куска кода соответствует не одной, а объединению нескольких последовательных строк. Все обучение основывалось на следующих факторах: текстовое представление исходного кода с помощью bag-of-words [5], статический анализ кода и метаинформация про изменение.

Для валидации результата проводилось обучение модели на 5 разных репозиториях. В качестве метрики качества использовалась RMSE (формула 1). Она вычислялась для предсказаний полученной модели и наивного решения, представляющим из себя среднее арифметическое. Доля RMSE для обоих решающих функций от RMSE наивного реше-

ния для каждого репозитория составляла от 0.7 до 0.9. Это означает то, что в среднем ошибка предсказаний полученных моделей составляет 80 процентов от ошибки наивного решения.

В этой статье есть много полезного, из неё, например, можно выделить признаки для обучения. Но в ней есть и определенные недостатки. Во-первых, система контроля версий позволяет вычислить большую информацию про изменения строк, чем Пер Нордфорс использовал в своей статье. Также предскания вычисляются не для одной строчки кода, а для целого непрерывно добавленного куска строк, называемого hunk. Такой подход позволяет делать предсказания для только что добавленных в коммите строк, но не позволяет быстро вычислять целевое значение для остальных. Это связано с тем, что разбиение файла на такие куски кода требует от системы контроля версий отката изменений назад по истории, что может занимать довольно долгое время. Кроме этого, в этой статье не произведён отбор признаков, проведение которого позволит понять, какие из признаков играют самую главную роль, а какие можно не учитывать.

Таким образом, для данной работы из статьи Пер Нордфорса можно взять ряд полезных приёмов, но в чистом виде его подход не применим для создания требуемого инструмента.

3. Ход исследования

3.1. Решающая функция

Решающая функция - это отображение из пространства признаков $\mathbb{R}^n$ в множество ответов. Для задачи регрессии ответом является число из $\mathbb{R}$, а для задачи бинарной классификации число из $\{0, 1\}$. Пусть дан репозиторий и конкретный его слепок, полученный после коммита c, а также строка кода s, у которой есть свой номер и файл, в котором она находится. Обозначим за $factors(c, s) \in \mathbb{R}$ вектор факторов, кодирующий строку кода s, всё ещё живую или только что появившуюся после c. Тогда мы хотим получить решающую функцию:

$$f : factors(c, s) \to \mathbb{R}$$

которая строке ставит в соответствие её время оставшейся жизни, то есть время в днях до того, как эта строка будет удалена или изменена. В частности, рефакторинг кода считается его удалением и добавлением нового. Так же нас интересует задача классификации строк код на два класса. Первый и них представляют строки, которые проживут больше одного года, вторые - меньше одного года. Формально она выглядит примерно так же, как и для задачи регрессии:

$$f : factors(c, s) \to \{0, 1\}$$

Здесь под единицей принимается первый класс, под нулём второй.

3.2. Инструменты

Код для получения искомой решающей функции пишется на языке python. Для работы с машинным обучением используются следующие библиотеки:

- GitPython [1] - библиотека для работы с системой контроля версий git. Она представляет из себя обёртку над её командами. Кроме

[1] www.gitpython.readthedocs.io

этого с помощью этой библиотеки можно делать запросы явно - результатом является строка, при этом ровно такая же, как если написать эту команду в командной строке. После этого полученную строку можно разобрать и извлечь всю необходимую информацию.

- scikit-learn [2] - библиотека для машинного обучения. В ней реализовано множество решающих функций, их алгоритмов обучения, а так же алгоритмы отбора факторов и валидации результата.
- Библиотека eli5 [3] - используется для вычисления permutation importance для выделения значимых признаков
- NumPy [4] и Pandas [5] - библиотеки для работы с данными. Удобное средство для работы с датасетами.

Для получения информации о потенциальных ошибках в исходном коде используется статический анализатор кода PMD. Он позволяет получить около 300 различных видов инспекций, разбитых на 8 групп. С помощью него можно получить различные предупреждения об ошибках в строках кода и использовать это как признаки.

3.3. Сбор датасета

Данные собираются из репозитория IntelliJ Comunity [1] на гитхабе. В нём порядка 50000 java файлов и 200000 коммитов. Коммиты извлекаются только из ветки master. Для сбора датасета выбирается слепок репозитория, соответствующий некоторому моменту времени в прошлом и множество файлов, из строк которых будет собираться датасет.

[2] www.scikit-learn.org
[3] www.eli5.readthedocs.io
[4] www.numpy.org
[5] www.pandas.pydata.org

3.3.1. Факторы

Для кодирования строки вектором признаков в $\mathbb{R}$ используются три вида факторов.

- **Признаки исходного кода** рассматривают файл, как текст. Они, например, показывают есть ли то или иное ключевое слово языка в строке, количество строк в файле, относительный номер строки и прочее.
- **Метрики изменений** показывают метаинформацию об изменениях, полученную из git. Например, сколько раз менялся файл, в котором содержится данная строка, текущее время жизни строки, опыт автора и прочее. Основная часть признаков получается с помощью команды *git annotate*, которая для каждой строки определенного файла выдаёт её автора и дату коммита добавления строки.
- **Факторы статического анализа** получаются из запуска анализатора PMD на полном проекте. После этого появляется сопоставление сообщений о возможных ошибках к строкам кода.

Более подробное описание всех используемых факторов приводится в таблицах 1 и 2. Для сборки признаков выбирается определенный коммит, делается откат изменений к этому коммиту и после этого вычисляются все признаки.

3.3.2. Целевая переменная

Для подсчёта времени жизни строк кода был написан алгоритм, который итерируется от начального коммита, который соответствует рассматриваемому слепку репозитория, до конечного, который был загружен в репозиторий через один год. На каждой итерации поддерживается сопоставление изначальных строк кода к строкам, полученных после применения текущего коммита, тем самым позволяя отследить, какая именно строка была удалена или изменена. После всех итераций

Таблица 1: Признаки строк: метрики изменений и факторы исходного кода

Номер	Признак	Вид
1-49	Наличие ключевых слов: abstract, continue, for, new, switch, assert, default, package, synchronized, boolean, do, if, private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case, enum, instanceof, return, transient, catch, extends, int, short, try, char, final, interface, static, void, class, finally, long, strictfp, volatile, const, float, native, super, while	Исходный код
49-50	Номер строки, номер строки относительно длины файла	
51-53	Длина строки, отношение длины строки к самой длиной/средней строки в файле	
53	Количество строк в файле	
54	Глубина вложения строки	
55-58	Текущее время жизни строки, минимальное/среднее/максимальное время жизни строки в файле	Метрики изменения
59-61	Отношение текущего времени жизни строки к минимальному/среднему/ максимальному времени жизни строки в файле	
62	Среднее время жизни соседних строк	
63	Количество коммитов автора строки	
64	Число строк у автора текущей строки в файле	
65-67	Минимальное/среднее/максимальное время жизни строк автора текущей строки	
68	Количество коммитов, из строк которых состоит файл	
69	Количество различных авторов строк в файле	
70-72	Отношение минимального/среднего/ максимального количества строк с одинаковым автором к длине файла	
73-75	Отношение минимального/среднего/ максимального количества строк с одинаковым коммитом появления к длине файла	

Таблица 2: Признаки строк: статический анализ кода

76-337	Выдал ли PMD сообщение о возможной ошибке конкретного типа (262 вида предупреждений)	Статический анализ
345	Суммарное количество предупреждений в строке по одной из групп (8 вид групп)	
353	Суммарное количество предупреждений в файле по одной из групп (8 вид групп)	

получается множество строк, которые были удалены или изменены за этот период, и множество строк, которые остались без изменений. Стоит отметить, что этот процесс занимает достаточно долгое время и может работать несколько часов.

3.4. Модель обучения

В качестве решающей функции как и для классификации, так и для регрессии был выбран случайный лес, потому что он обладает следующими хорошими свойствами:

- Высокая обобщающая способность
- Не склонен к переобучению
- Нелинейная функция, способная выразить довольно сложные зависимости
- На этапе обучения модели и получений предсказаний можно выполнять вычисления паралельно

3.5. Оценка качества предсказаний

Для измерения качества предсказаний собирается два датасета - обучающий и отложенный. Обучающий датасет разделяется на обучающую и тестовую выборку в отношении 3 к 1. Тестовая выборка, так же как и отложенная, не участвуют в обучении модели - на них оцениваются результаты предсказаний.

Для того, чтобы результат был валидным, очень важно следить за тем, чтобы в отложенный датасет не попала информация про строки из тренировачного датасета, иначе полученный результат будет не репрезентативным. Нужно подобрать контрольный датасет так, чтобы он максимально симулировал запросы предсказаний, на которые должна отвечать наша решающая функция при испльзовании плагина. Поэтому тренировочный датасет собирается по слепку репозитория некоторого года в прошлом, а контрольный из более позднего. При этом значения целевой переменной собираются по одному и тому же периоду.

Помимо этого, обычно разделение выборки на 2 части происходит случайным образом, чтобы в обучающей и проверяющей части было схожее распределение объектов. В нашем случае, используя подобный подход, строки одного и того же файла могут попасть и в обучающую, и в тестовую выборку. В ходе исследования было выяснено, что подобный подход негативно влияет на результаты предсказаний на отложенной выборке, хотя и показывает очень хорошие результаты на проверяющей. Это случается из-за того, что значения целевой переменной для строк одного файла почти одинаковые. Поэтому признаки, которые могут неявно закодировать файл, выходят на первый план и таким образом модель показывает хорошие результаты на выделенной четверти обучающего датасета, но плохие на контрольной выборке. Решением этой проблемы стало перемешивание датасета не по самим строкам, а по целым файлам. Тогда строки одного файла оказываются только в одной выборке, при этом случайное распределение строк по выборкам сохраняется.

В качестве метрики качества для регрессии была выбрана RMSE (формула 1), а для классификации - AUC (формула 3). Так же для наглядности и простоты интерпретации для регрессии приведена MAE (формула 2).

3.6. Результаты предсказаний

В таблице 3 приведены результаты предсказаний для регрессии и классификации. Обучающий датасет был вычислен из слепка репозитория IntelliJ community 2017 года, а отложенный из 2018.

Таблица 3: Результаты предсказаний

Целевая функция	Тестовая выборка (2017)	Отложенная выборка (2018)	Тип
RMSE	79.967	98.830	Регрессия
MAE	67.8	70.935	
AUC	0.772	0.644	Классификация

Результаты означают, что в среднем на отложенной выборке предсказания оставшегося времени жизни строк кода отличаются от истинных значений примерно на 3 месяца, а на тестовой примерно на 2 месяца. Что касается классификации, AUC эквивалентна вероятности того, что классификатор присвоит больший вес положительному классу. Базовое значение AUC равняется 0.5 и соответствует случайной классификации. Так как и на тестовой и на отложенной выборке значения AUC больше, чем базовое, то это показывает по крайней мере то, что построенная таким образом модель обладает предсказательной способностью и действительно есть корреляция между выбранными признаками строки и оставшимся временем её жизни.

3.7. Отбор признаков

Для улучшения качества предсказаний в ходе работы применялся отбор признаков. Существует много различных подходов, из которых был выбран *permutation importance*. Время вычисления этого показателя важности признака относительно мало и является независимым от количества признаков, в отличие от forward selection и backward elimination [6], которые на больших датасетах могут вычисляться очень долго. Так же этот признак основывается не на обучающей выборке,

[6] www.gerardnico.com/data$_m$*ining*/*stepwise*$_r$*egression*

как, например, feature importance [7], а на результатах предсказаний на тестовой выборке. Кроме того permutation importance подходит для любых решающих функций и любых метрик качества, что делает его применимым к нашей задаче. В итоге получилось выявить самые сильные признаки для предсказаний времени жизни строк кода для репозитория IntelliJ IDEA. Самые лучшие признаки приведены в таблице 4 в порядке уменьшения важности. BestPractices inspections и CodeStyle inspections - это суммарное количество инспекций в файле строки определенного вида. BestPractices - это инспекции, которые побуждают следовать правилам хорошего тона. Например, они возникают, когда локальная переменная объявлена, но не используется, или когда класс объявлен абстрактным, но не содержит ни одного абстрактного метода. Инспекции вида CodeStyle возникают при нарушении стиля кода. Например, когда не соблюдается соглашении об именовании классов или когда у нестатического класса не объявлен ни один конструктор.

Таблица 4: Лучшие признаки по показателю permutation importance

1	BestPractices
2	Текущее среднее время жизни строк кода в файле
3	Количество коммитов автора строки
4	Code Style
5	Количество строк автора строки в файле

[7] www.blog.datadive.net/selecting-good-features-part-iii-random-forests/

4. Разработка плагина IntelliJ IDEA

Плагин предоставляет следующие возможности. По запросу пользователя для конкретного файла для каждой строки отображаются предсказания в виде трёх цветов на левой вертикальной панели (Gutter Bar) редактора. При этом происходит вычисление признаков строк для данного файла и значений решающей функции.

В плагине не происходит обучения модели, так как это бы заняло очень много времени для больших проектов. Например, сбор датасета из репозитория IntelliJ IDEA занимает примерно 10 часов, а обучение случайного леса около 30 минут. Поэтому уже обученный случайный лес встраивается в плагин, что позволяет не требовать у пользователя много времени на сбор датасета.

4.1. Структура плагина

Структура плагина состоит из следующих компонент:

- Действия - это классы, обрабатывающие события. Они позволяют осуществлять взаимодействие между пользователем и плагином.
- Специальные классы для хранения и извлечения признаков строк для файлов. Они вычисляют значения признаков, обращаясь к исходному коду и системе контроля версий.
- Сервисы - классы, соответствующие шаблону проектирования singleton. Они нужны для кеширования признаков.

4.1.1. Графический интерфейс

Графический интерфейс разработанного плагина отображается на левой боковой панели редактора. Эта панель раскрашивается в три цвета - красный, желтый и зелёный. Красный означает, что строка проживёт меньше месяца, желтый - от месяца до года и зелёный - больше года. Графический интерфейс показан на рисунке 1.

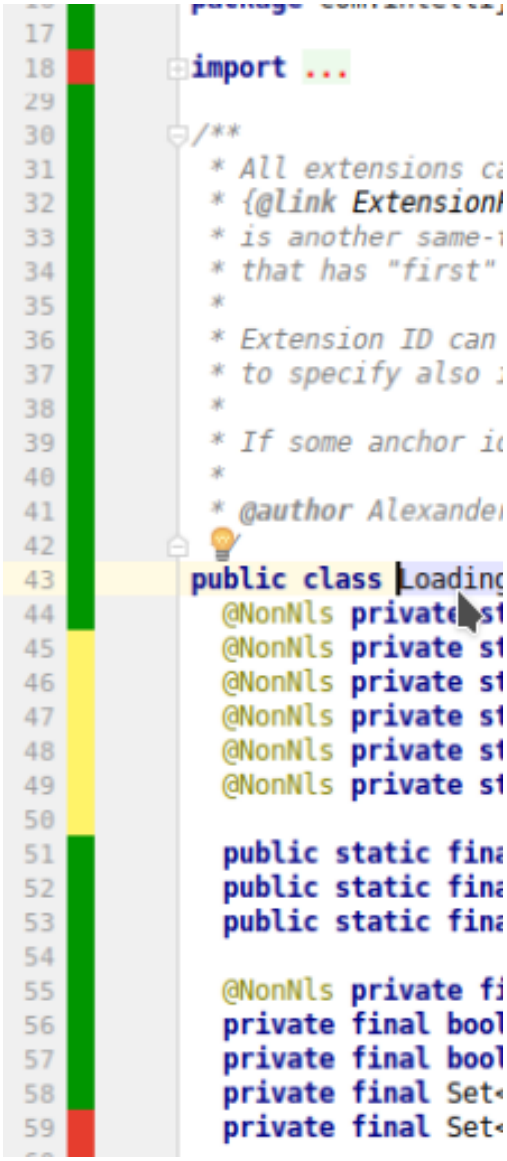

Рис. 1: Графический интерфейс

IntelliJ IDEA предоставляет возможность обращаться к боковой панели, зарегистрировав класс действия (action) в конфигурационном файле. Так, основной класс, который отвечает за отображение результатов, реализует интерфейс TextAnnotationGutterProvider. Он реализует несколько методов, связанных с заданием шрифта и цвета текста, а также два метода, которые как раз должны отображать результаты предсказаний. Они принимают на вход номер строки и экземпляр класса редактора. Эти два метода самые основные, так как именно с них начинается вычисление признаков для строк текущего файла и результатов предсказаний.

4.1.2. Вычисление факторов и их кеширование

Признаки, которые вычисляются в плагине, можно разделить на две категории - признаки исходного кода и признаки, полученные из системы контроля версий. За их вычисление и хранение отвечают два класса, экземпляры которых соответствуют конкретным файлам.

IntelliJ IDEA предоставляет возможность обращаться к файлам как к тексту. Для вычисления факторов исходного кода используются встроенные в стандартную библиотеку функции работы со строками.

Работа с системой контроля версий осуществляется с помощью библиотеки git4idea. Используя некоторые её методы из репозитория извлекается такая же информация, как и при консольной команде git annotate. Из полученной информации вычисляется большинство признаков, связанных с системой контроля версий, такие, как, например, текущее время жизни строки и количество коммитов, из которых состоит файл.

Такое извлечение признаков из репозитория выполняется относительно долго, так как системе контроля версий приходится откатываться назад по истории коммитов. Так же git не предоставляет функциональности извлечения информации только для одной строки в файле. Так как вызывать эту операцию для каждой строки файла дорого, было решено использовать кеширование признаков по файлам.

За механизм кеширования отвечают два сервиса, соответствующие шаблону проектирования singleton. Они хранят в себе глобальную хеш-таблицу, ключами которой являются файлы, а значениями экземпляры классов, которые хранят в себе вычисленные значения признаков. Когда поступает запрос на вычисление признаков строки, сначала в кеше ищется соответствующий файл, и, если не находится, то кеш пополняется новым элементом. Такой подход в разы увеличивает скорость работы плагина.

Весь процесс работы плагина отображен в диаграмме активности (рисунок 2).

После того, как пользователь нажимает кнопку в контекстном ме-

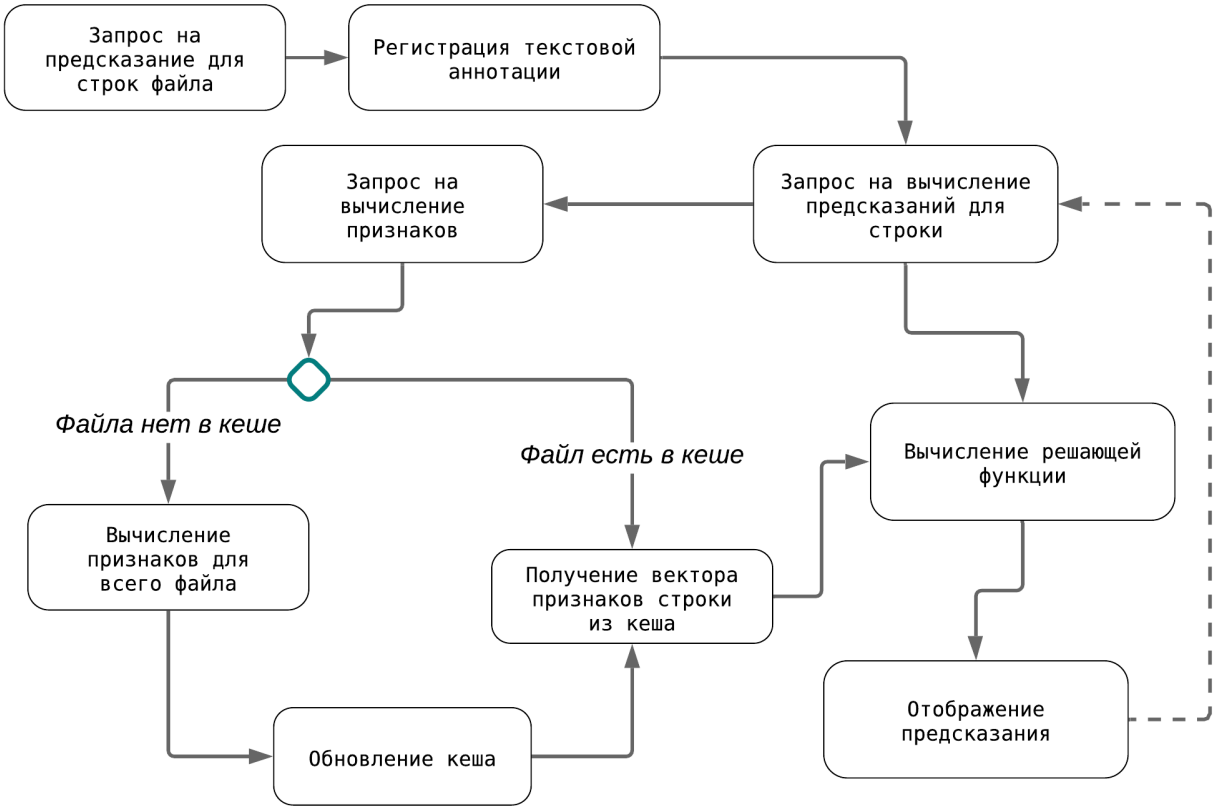

Рис. 2: Диаграмма

ню, в текущем редакторе регистрируется текстовая аннотация, которая для каждой строки посылает запрос на вычисление предсказаний. Для каждого запроса проверяется, есть ли текущий файл в кеше, и если есть, то признаки строки берутся из кеша. Иначе признаки вычисляются для всего файла и кешируются. С помощью полученных признаков строки вычисляется значение решающей функции и полученный результат предсказаний отображается с помощью графического интерфейса.

4.2. Встраивание решающей функции

Случайный лес представляет из себя ансамбль деревьев решений. Каждое такое дерево является бинарным. В каждой вершине дерева находится предикат, который по признаку определяет, в какое поддерево нужно спуститься объекту.

Для генерации java кода, соответствующего обученной модели с помощью языка python, использовался уже готовый скрипт. При запуске скрипта каждое дерево случайного леса преобразуется в java-класс, у которого есть всего один статический метод. Этот метод состоит из обилия конструкций if-else, которые соответствуют предикатам вершин, и возвращает предсказание одного решающего дерева. Так же после преобразования создаётся класс случайного леса, который вызывает статические методы у классов решающих деревьев, после чего их усредняет.

Заключение

В ходе работы были получены следующие результаты:

1. Был написан код для автоматической сборки датасета для произвольного слепка репозитория.
2. Собраны четыре датасета из слепков репозитория IntelliJ IDEA, состоящие из 353 признаков.
3. Получена решающая функция для задач регрессии и классификации времени жизни строк кода в репозитории IntelliJ community.
4. Вычисленны результаты прдесказаний для полученных моделей.
5. Было вычислено значение permutation importance для полученных решающих функций и собранных датасетов.
6. Написан плагин для IntelliJ IDEA, красиво отображающий результаты предсказаний.

В дальнейшем планируется использовать более продвинутые статические анализаторы, такие, как, например, инспекции IntelliJ IDEA, добавлять новые признаки и улучшать результаты предсказаний.

Список литературы

[1] JetBrains. IntelliJ IDEA. — URL: https://www.jetbrains.com/idea/.

[2] Marco D'Ambros Michele Lanza Romain Robbes. An Extensive Comparison of Bug Prediction Approaches. — 2010.

[3] Michael Pradel Koushik Sen. Deep Learning to Find Bugs. — 2017.

[4] Nordfors Per. Prediction of code lifetime. — 2017.

[5] Wikipedia. Bag-of-words. — URL: https://en.wikipedia.org/wiki/Bag-of-words_model.