Санкт-Петербургский Государственный Университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Черняев Арсений Витальевич

Применение методов машинного обучения в системах отслеживания ошибок программного обеспечения

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:
Иванов Д. А.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Arsenii Cherniaev

Application of machine learning algorithms to bug tracking systems

Bachelor's Thesis

Scientific supervisor:
professor Andrey Terekhov

Reviewer:
Dmitry Ivanov

Saint-Petersburg
2016

Оглавление

1. Введение

При создании современного программного обеспечения зачастую возникает потребность в использовании специальных средств, позволяющих программистам контролировать ход разработки, получать отзывы от пользователей и поддерживать связь с другими членами команды, особенно в больших распределенных проектах.

Одним из таких средств является система отслеживания ошибок. Эта система позволяет команде разработчиков учитывать и контролировать ошибки, найденные в ходе эксплуатации программного обеспечения. Каждый, кто имеет доступ к системе, может создать специальный запрос, затем внутри команды этот запрос будет обработан, будет выделен человек или команда, ответственные за решение поставленной задачи или исправление допущенных исключительных ситуаций. Запрос получит уникальный номер, состояние, символизирующее статус работы, и может служить местом обсуждения проблемы внутри команды или между командой разработчиков и пользователем.

В связи с выше сказанным, в системах отслеживания ошибок можно выделить ряд прикладных задач, позволяющих упростить процесс работы создателей программного обеспечения. Один из примеров - задача объединения сообщений в кластеры для нахождения дубликатов.

Отдельного упоминания заслуживает задача определения ответственного за устранение неисправности, описанной в сообщении. В больших проектах возможны ситуации, когда ежедневно к отслеживанию добавляются тысячи новых ошибок и предложений к улучшению, требующих назначения ответственного. В особенности это касается отчетов об исключительных ситуациях - во многих крупных проектах имеется возможность автоматической отправки такого типа сообщений, из-за чего система отслеживания ошибок будет подвергаться огромной нагрузке. Также стоит отметить, что данные отчеты генерируются автоматически и очень часто не содержат дополнительной информации от пользователя. Ручное определение разработчика в таком случае может занимать еще больший объем времени, даже в случае, если человек,

обрабатывающий запросы, обладает достаточными знаниями о всех деталях разрабатываемого программного обеспечения.

Таким образом, можно сделать заключение, что необходима автоматизация данного процесса, позволяющая переложить большую часть работы с разработчика на машину. Методы машинного обучения, позволяющие решать многие задачи классификации и кластеризации, могут быть эффективными и для проблемы классификации отчетов об исключительнх ситуациях.

2. Постановка задачи

Цель данной дипломной работы заключается в разработке модуля, предназначенного для автоматизации назначения разработчика, ответственного за исправление допущенной при эксплуатации программного обеспечения исключительной ситуации для проекта ReSharper в системе отслеживания ошибок YouTrack. Для этого необходимо выполнить следующие задачи:

- Провести подготовку данных. На данном этапе необходимо выбрать способ преобразования отчета в наиболее подходящий для дальнейшего анализа вид, а также сформировать набор данных для обучения.

- Выбрать наиболее подходящий метод классификации. Для этого необходимо сравнить различные алгоритмы обучения на подготовленных данных проекта, а также некоторые многоуровневые модели предсказаний в выбранной для этого метрике.

- Реализовать модуль, решающий проблему автоматизации назначения разработчика, для сервера, занимающегося обработкой отчетов об исключительных ситуациях, присылаемых из системы отслеживания ошибок.

3. Исходные данные

В качестве набора данных используются отчеты об исключительных ситуациях проекта ReSharper, расположенного в системе отслеживания ошибок YouTrack за все время его существования. Данный набор содержит 5974 размеченных отчета (они помечены как исправленные, и для них известен ответственный за них разработчик), а также примерно 34000 уникальных неразмеченных отчетов.

Каждый отчет состоит из:

- Заголовок сообщения - содержит информацию о типе исключительной ситуации и сопровождающем ее тексте;
- Стек вызовов;
- Версия продукта, в которой произошел сбой;
- Дата появления проблемы;
- Информация о разработчике, ответственном за исправление;
- Текущий статус проблемы;
- Дополнительная информация о версии продукта, в которой проблема будет исправлена;
- Подсистема - в проекте ReSharper подсистема обозначает собой определенную функциональность, обособленную от всех остальных. Определение подсистемы, в которой возникла исключительная ситуация, может помочь в определении ответственного разработчика.

4. Обзор существующих работ

Методы машинного обучения применялись к области классификации сообщений об ошибках и раньше. Большинство подходов основывались на принципе рассмотрения задачи как более общей задачи классификации текста, применялись различные классификаторы: наивный байесовский классификатор [3] [6], машины опорных векторов [1] [3] [18], решающие деревья [3] [18].

Проблема применимости данных подходов к конкретной задаче заключается в том, что они основываются на идее обработки текста сообщения, который в случае отчетов об исключительных ситуациях является менее информативным: многие элементы выборки имеют один и тот же текст, но при этом относятся к абсолютно разным разработчикам. Основным источником для определения сути проблемы в таком случае является стек вызовов, неизменно прилагающийся к отчету. По этой причине рассмотрение другого подхода к подготовке данных может улучшить результаты.

Также стоит отметить, что данные исследовательские работы в большинстве своем не рассматривают возможности применения ансамблей классификаторов, которые могут увеличить точность предсказания [2], а также не рассматривают возможность игнорирования предсказания в случае низкой степени уверенности классификатора, что позволит уменьшить вероятность ошибки за счет небольшого количество данных, которые придется классифицировать вручную.

Помимо методов, основанных на применении алгоритмов машинного обучения, стоит также рассмотреть идею, придуманную ранее внутри команды ReSharper. Ее суть заключалась в применении голосующего алгоритма по отношению к строкам стека вызовов. Задавался набор правил вида "префикс строки стека" - "разработчик". Рассматривались все строки стека вызовов, если строка начиналась с определенного префикса, то счетчик соответствующего разработчика увеличивался на 1. Набравший достаточно много голосов разработчик назначался ответственным.

Этот подход не дал достаточной точности и поэтому не использовался. Помимо этого, стоит отметить, что данный подход недостаточно автоматизирован - при появлении нового разработчика в команде необходимо добавлять новые правила. Также, возможны ситуации, когда несколько разработчиков работают над схожей областью, и поэтому деление по префиксам строки может быть недостаточно точным.

4.1. Методы оценки точности

Для оценки точности работы классификаторов на тестовой выборке использовались различные подходы:

1. Оценивание общего количества верно предсказанных элементов выборки:

$$\text{Общая точность} = \frac{\text{Количество верно предсказанных}}{\text{Общее количество предсказаний}}$$

   Данная оценка является самой простой и чаще других возможных оценок используется в существующих работах в данной области.

   Однако стоит отметить, что она плохо отражает точность предсказания для каждого конкретного класса, поэтому возможны ситуации, когда, например, хорошая степень предсказания часто встречающего класса нивелирует низкую точность для объектов редкого класса. Поэтому данная оценка вычислялась, в основном, для сравнения с существующими подходами.

2. Рассмотрение усредненной по всем классам F_1-меры [14].

   Для каждого класса c рассматривались оценки точности Precision и Recall:

$$Precision_c = \frac{\text{Количество верных предсказаний элементов класса}c}{\text{Количество предсказаний принадлежности классу}c}$$

$$Recall_c = \frac{\text{Количество верных предсказаний элементов класса}c}{\text{Количество элементов, принадлежащих классу}c}$$

$$F_{1,c} = 2\frac{Precision_c Recall_c}{Precision_c + Recall_c}$$

$$F_1 = \frac{\sum_{c \in C} F_{1,c}}{|C|}$$

Данная оценка гораздо более явно позволяет определять степень точности классификации для каждого класса и поэтому использовалась как основная оценка для сравнения классификаторов между собой.

5. Используемая методология

Далее идеет краткий обзор методов машинного обучения, используемых в данной работе.

Многие из перечисленных ниже алгоритмов были выбраны, т.к. показали хорошие результаты в задачах классификации текстов, частным случаем которой можно считать рассматриваемую задачу [1] [3] [6] [8] [10]. Также, к рассмотрению были добавлены популярные алгоритмы, использующие ансамбли классификаторов, поскольку их использование зачастую позволяет улучшить результаты базовых алгоритмов.

Для решения поставленных задач использовались реализации методов машинного обучения библиотек Scikit-learn [13] и Xgboost [5] для Python. Данные язык и библиотеки были выбраны в силу возможности быстрой разработки прототипа и большого количества вспомогательных сторонних библиотек для исследования.

Scikit-learn содержит большое количество оптимизированных алгоритмов обучения и подготовки данных, включая возможность работы с разреженными матрицами, а библиотека Xgboost содержит показывающую высокие результаты реализацию метода градиентного бустинга.

5.1. Логистическая регрессия

Логистическая регрессия - пример линейного классификатора [20].

Данный алгоритм классификации использует сигмоидную функцию $g(x) = \frac{1}{1+e^{-x}}$ в качестве функции активации и позволяет дать вероятностную оценку принадлежности объекта каждому классу. В [8] данный метод описан подробнее; также, упомянутая статья описывает применение алгоритма к задаче классификации текстов.

5.2. Наивный байесовский классификатор

Наивный байесовский классификатор - вероятностный классификатор. Основан на применении теоремы Байеса с дополнительным пред-

положением о независимости различных свойств рассматриваемой модели.

Основным вопросом при использовании классификатора является следующий: какому закону подчиняются плотности распределения случайных величин, характеризующих значение каждого свойства объекта? Существует большое количество подходов, основанных на различных предположениях, например: подчиненности случайных величин нормальному закону распределения или предположении, что вектора объектов характеризуют частоты встречания [6].

Традиционно считается, что наивные байесовские классификаторы хорошо показывают себя в задачах классификации текстов, поэтому в рамках данной дипломной работы они были рассмотрены в первую очередь.

5.3. Машина опорных векторов

Данный линейный классификатор [20] основывается на принципе построения оптимальных гиперплоскостей, разделяющих предположительно линейно разделяемые данные. Такие гиперплоскости будут максимально удалены от всех элементов выборки и тем самым наиболее явно разделять пространство на классы. В случае линейно неразделяемых данных рассматривается возможность применения некоторой функции-ядра, переводящей элементы в пространство, где данные могут быть разделяемыми. Подробнее о данном методе можно прочесть в [9].

5.4. Метод ближайших соседей

Метод ближайших соседей - метрический алгоритм, его суть заключается в рассмотрении нескольких ближайших в некоторой метрике соседей с последующим выбором доминирующего среди них класса [19]. Основной вопрос заключается в выборе оптимального количества рассматриваемых соседей - небольшое количество соседей создаст слишком большую чувствительность к шумам, в то время как слишком большое

количество ухудшит предсказание путем вырождения функции в константу. Также важен выбор наиболее подходящей метрики для измерения расстояния.

Помимо этого, стоит упомянуть различные модификации, использующие, например, веса объектов, обратно пропорциональные расстоянию до классифицируемого объекта.

5.5. Решающие деревья

Решающие деревья - класс алгоритмов классификации, позволяющий получать предсказание для объекта за счет последовательности ответов на иерархически организованную систему вопросов (выстроенную в виде дерева) [17].

Каждый внутренний узел в дереве представляет собой некоторую функцию f, которая действует над пространством, содержащем элементы выборки, и принимает одно из конечного числа значений, задающих переход к соответствующему потомку. Каждый лист в дереве - это метка определенного класса.

Функция f может представлять собой, например, простое пороговое условие над некоторым свойством или же некоторую их конъюнкцию.

Существует несколько реализаций решающих деревьев, отличающихся методами построения дерева. Примерами могут служить ID3, C4.5 и CART [17]. В данной работе рассматривался алгоритм CART.

5.6. Случайный лес

Случайный лес - это алгоритм классификации, основанный на принципе использования ансамбля нескольких решающих деревьев для достижения ими большей точности [4].

Основные шаги алгоритма следующие:

Классификаторы (решающие деревья) обучаются независимо друг от друга. Каждый классификатор обучается на некотором подмножестве элементов обучающей выборки (выбираемом случайным образом с повторением), причем алгоритм его обучения изменен: на каждом этапе

разбиения при обучении выбирается наиболее подходящее свойство не из всех n свойств, а из некоторого количества произвольно выбранных, с целью построения более разнообразных деревьев.

Затем классификаторы независимо друг от друга делают предсказания о входном элементе, и класс, за который проголосовало больше всего классификаторов, становится предсказанием итогового классификатора.

5.7. Методы градиентного бустинга

Методы градиентного бустинга позволяют использовать ансамбль классификаторов для увеличения точности предсказания, в качестве итоговой вероятности принадлежности объекта классу используется взвешенная сумма вероятностей принадлежности объекта классу каждого классификатора.

Основной принцип градиентного бустинга заключается в постепенном наращивании композиции классификаторов:

$$g_i(x) = g_{i-1}(x) + \alpha_i h(x, \beta_i)$$

Здесь $h(x, \beta_i)$ - базовый классификатор с параметрами β_i, α_i - коэфициент взвешенной суммы. Используя методику градиентного спуска, можно постепенно подбирать параметры так, чтобы минимизировать некоторую выбираемую функцию потерь.

Частным случаем метода градиентного бустинга может служить алгоритм AdaBoost [7], использующий экспоненциальную функцию потерь или же алгоритм Extra Gradient Boosting библиотеки xgboost [5]. Именно эти два метода рассматривались в данной работе.

5.8. Expectation-Maximization

Технику Expectation-Maximization можно применить к задаче классификации с частичным обучением [10].

В первую очередь по размеченным данным строится наивный бай-

есовский классификатор, затем все неразмеченные данные получают предсказания о вероятности принадлежности к каждому классу. Затем полученные вероятностые показатели используются для создания нового классификатора. Процесс продолжается до тех пор, пока не образуется стабильный классификатор.

6. Эксперименты

6.1. Подготовка данных

В качестве данных рассматривались подготовленные данные проекта ReSharper. Была собрана статистика по всем разработчикам, когда-либо участвовавшим в исправлении неисправностей в проекте, среди них были выбраны те, кто в данный момент являются разработчиками в команде. Из этого набора людей были удалены люди, данных для которых почти нет. Оставшиеся разработчики рассматривались как множество классов для классификации. Их количество составило 28 человек.

В качестве размеченных данных использовались отчеты об исключительных ситуациях, исправленные кем-то из отобранных разработчиков. Размер полученной выборки - 5974 примера.

Для тестирования различных подходов к классификации размеченные данные каждый раз разбивались случайным образом, на 80% происходило обучение классификатора, оставшиеся 20% использовались для тестирования корректности. Данная процедура повторялась 10 раз, затем результаты тестирования усреднялись, полученные величины рассматривались как результат работы классификатора.

Важным шагом для подготовки данных являлся способ их преобразования в числовые вектора. Было решено рассматривать задачу как задачу классифкации текста, были рассмотрены возможности сбора информации об отчете при помощи:

- Обработка стека вызовов - обработка строк стека вызовов одним из следующих способов:
  - Использование строки стека вызовов без списка параметров, как единого слова, добавляемого в документ.
    Пример: ”Namespace.Class.Method(parameters)” ->
    ”Namespace.Class.Method”.

– Разбиение каждой строки стека вызовов без параметров на отдельные слова, добавляемые в документ.

  Пример: ”Namespace.Class.Method(parameters)” -> ”Namespace”, ”Class”, ”Method”.

- Обработка текста сообщения - из текста отчета удалялись символы, отличные от элементов алфавита, в силу большого количества специфичных для области IT-индустрии слов было решено не использовать стемминг. Полученный набор слов добавлялся в документ, соответствующий отчету.

Были рассмотрены различные способы подготовки строковых данных:

- Хэширование строк: к каждому слову в документе применяется некоторая хэш-функция, принимающая значения от 0 до некоторого N. Тогда документ можно представить как вектор длины N+1, значение в i-ой ячейке которого говорит о том, сколько различных слов имеют значение хэш-функции, равное i.

  Данный подход позволяет свободно регулировать длину векторов, однако полученные вектора свойств будут неинформативны, в силу неоднозначности хэш-функции, таким образом, пропадает возможность анализа наиболее подходящих для рассматриваемой задачи свойств.

- Подход Bag-of-the-Words: из всех слов во всех документах составляется словарь, далее документу ставится в соответствие вектор длины размера словаря, показывающий, какие слова (и, возможно, сколько раз) встречаются в данном документе.

  Данный подход делает возможным анализ важности тех или иных свойств для обучения, однако при этом возникают дополнительные проблемы, связанные с обработкой векторов большой размерности.

Для учета степени популярности того или иного слова использовалась статистика TF-IDF [11], прямо пропорциональная количеству вхождений слова в документ и обратно пропорциональная общему количеству документов, содержащих рассматриваемое слово:

$$TF - IDF(t, d) = \frac{\text{количество вхождений t в d}}{\text{количество слов в d}} \times$$

$$\times \log(\frac{\text{Общее число документов}}{\text{Число документов, содержащих t}})$$

Для уменьшения размерности числовых векторов и увеличения точности вследствие отсечения "шумовых" свойств, в случае рассмотрения подхода Bag-of-the-Words, были рассмотрены 3 различных методики, дающие для каждого свойства некоторую числовую оценку его важности, благодаря которой затем выбираются несколько самых значимых:

- Mutual Information - это один из способов определения степени зависимости некоторого набора классов C от слова t [15].

  Предположим, что A_c - событие "встретился документ класса c", B - событие "встретился документ, содержащий слово t. Тогда данная статистика вычисляется следующим образом:

  $$MI(t, C) = \sum_{c \in C} P(A_c, B) \log_2(\frac{P(A_c, B)}{P(A_c)P(B)} + \sum_{c \in C} P(A_c, \bar{B}) \log_2(\frac{P(A_c, \bar{B})}{P(A_c)P(\bar{B})}$$

- Тест χ^2 - данный тест позволяет оценивать степень уверенности в независимости встречания некоторого слова t и класса c. Данная величина вычисляется следующим образом:

  $$\chi^2(t, c) = \frac{(Expected(t, c) - Observed(t, c))^2}{Expected(t, c)}$$

  Здесь $Observed(t, c)$ - то, сколько раз слово t встречается в документах класса c, $Expected(t, c)$ - сколько раз t должно встречаться в документах класса , при условии, что величины, характеризующие появление слова t в некотором документе и появление доку-

мента класса c, являются независимыми.

Слова, обладающие малым значением данной статистики по всем классам, можно отбросить как вероятно не зависящие от классов. Подробнее о данном методе можно посмотреть в [16].

С точки зрения статистики применение данного теста не всегда корректно, однако в реальных задачах классификации его применение увеличивает точность предсказания.

- Использование решающих деревьев.

  Данный подход заключается в передаче всей выборки некоторому классификатору, основанному на решающих деревьях. После построения модели, можно рассмотреть, какие свойства использовались при ветвлении дерева. Предполагается, что свойство, рассмотренное в узле решающего дерева, является более важным для классификации. Для создания более надежной статистики можно рассматривать ансамбль деревьев и учитывать ветвления во всех деревьях. В итоге для каждого свойства будет известно, сколько раз оно встречалось в узлах деревьев. Библиотека xgboost [5] содержит реализацию данного подхода.

В качестве итогового способа подготовки данных было решено остановиться на методе, использующем стек вызовов с разбиением на отдельные слова и заголовок сообщения без использования стемминга, с последующем применением подхода Bag-of-the-Words и уменьшением размерности при помощи решающих деревьев, так как при этом подходе удалось добиться наибольшей точности при тестировании - максимальный результат, показанный каким-либо из выбранных классификаторов (см. 6.2), при данном подходе был выше прочих (см. Таблица 2, Таблица 3, Таблица 4, Таблица 5). Также, стоит отметить, что методика Bag-of-the-Words позволяет получить более наглядное представление свойств, а решающие деревья сильнее всего уменьшают размерность числовых векторов (с 20674 до 1970 элементов), что увеличило скорость работы.

На заключительном этапе данные были нормализованы (сведены в интервал [0; 1]), т.к. многие алгоритмы машинного обучения чувствительны к масштабированию данных.

Также, рассматривались возможности учета даты создания, комментариев к отчету или версии продукта, а также использования N-грамм (композиций последовательных слов), однако добиться увеличения точности предсказания при этом не получилось.

Интересным подходом, не рассмотренном в данной работе, является использование истории отчета. В [3] используется понятие графа пересылок - хранимой для каждого отчета истории принадлежности разным разработчикам. Его анализ позволяет редактировать результат предсказания базового классификатора.

К сожалению, недостаток большинства данных проекта ReSharper заключается в пустой либо некорректной истории. Это вызвано миграцией проекта из одной системы отслеживания ошибок в другую, в ходе которой данные были утеряны. Поэтому применение каких-либо методик, основанных на анализе истории, является невозможным.

6.2. Применение методов машинного обучения

К подготовленным данным были применены следующие методы машинного обучения с учителем:

- Логистическая регрессия;
- Гауссовский наивный байесовский классификатор;
- Метод ближайших соседей;
- Машина опорных векторов с линейным ядром;
- Решающее дерево (CART);
- Случайный лес;
- AdaBoost с решающими деревьями в качестве базовых классификаторов;

- Метод Extra Graident Boosting библиотеки xgboost.

Библиотека scikit-learn взята как источник реализаций всех алгоритмов, кроме последнего. В Таблице 1 приведены результаты классификации, которых удалось достичь на данных, подготовленных при помощи метода, выбранного в предыдущем параграфе.

Метод	Total Accuracy, %	F_1
Логистическая регрессия	74	0.63
Наивный байесовский классификатор	64	0.43
Метод ближайших соседей	58	0.39
Машина опорных векторов с линейным ядром	74	0.61
Решающее дерево (CART)	62	0.52
Случайный лес	73	0.62
AdaBoost	69	0.53
Extra Gradient Boosting	74	0.56

Таблица 1: Результаты применения методов машинного обучения к задаче классификации на разработчиков

При этом среднеквадратичные отклонения общей точности для всех методов не превосходили 0.5%, а среднеквадратичные отклонения усредненной F_1-меры не превосходили 0.005.

Метод логистической регрессии показал наилучшие результаты (и высокую стабильность - очень небольшие среднеквадратичные отклонения), в дальнейшем его было решено использовать как базовый классификатор, улучшения которого мы будем искать.

Сравнение данного метода с некоторыми другими работами в данной области, использующими машинное обучение, предоставлено на Рис. 1. Сравнение произведено на основе измерения общей точности, т.к., как уже отмечалось в 4.1, это наиболее общий и часто встречающийся метод измерения точности для работ в данной области. Однако, стоит заметить, что данное сравнение не совсем корректно, т.к. рассматриваемые классификаторы зачастую работают с различными

данными, а в данной работе, ввиду рассмотрения отчетов об исключительных ситуациях, содержащих стек вызовов, еще и с данными более конкретной структуры.

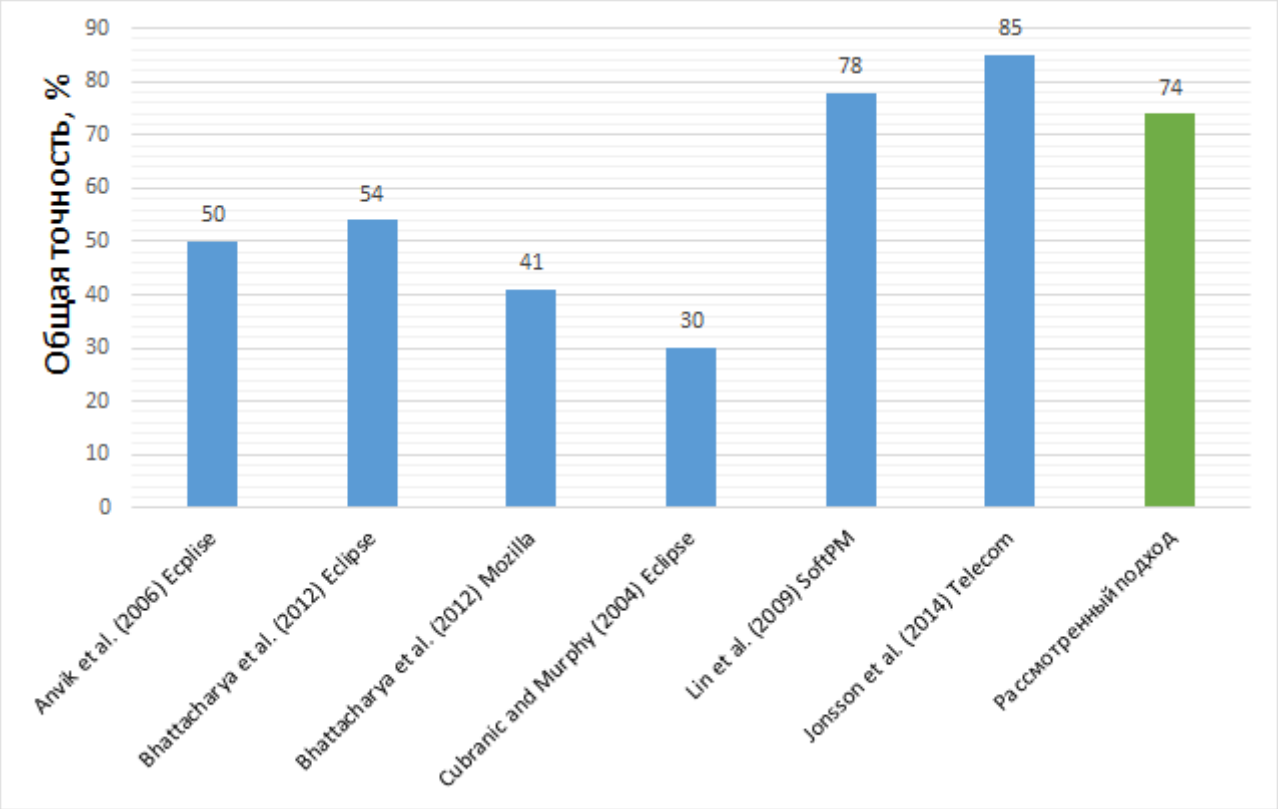

Рис. 1: Сравнения результатов классификаторов, описанных в других работах данной области, с используемым методом логистической регрессии

В расположенных ниже таблицах находятся максимальные результаты, достигнутые при применении алгоритмов классификации к основным комбинациям методов подготовки данных. На их основе и был выбран итоговый вариант преобразования:

Метод	F_1
Использование строк стека вызовов целиком	0.53
Использование строк стека вызовов целиком + Обработка текста сообщения	0.54
Разбиение строк стека вызовов на слова	0.53
Разбиение строк стека вызовов на слова + Обработка текста сообщения	0.54

Таблица 2: Максимальное значение F_1-меры, которой удалось добиться при каждом из рассмотренных способов преобразования при использовании техники хэширования слов (Максимальное значение хэша при максимумах - 7000)

Метод	F_1
Использование строк стека вызовов целиком	0.60
Использование строк стека вызовов целиком + Обработка текста сообщения	0.61
Разбиение строк стека вызовов на слова	0.61
Разбиение строк стека вызовов на слова + Обработка текста сообщения	0.62

Таблица 3: Максимальное значение F_1-меры, которой удалось добиться при каждом из рассмотренных способов преобразования при использовании техник Bag-of-the-Words и Mutual Information (Выбираемое количество свойств при максимумах - 7500)

Метод	F_1
Использование строк стека вызовов целиком	0.60
Использование строк стека вызовов целиком + Обработка текста сообщения	0.61
Разбиение строк стека вызовов на слова	0.61
Разбиение строк стека вызовов на слова + Обработка текста сообщения	0.61

Таблица 4: Максимальное значение F_1-меры, которой удалось добиться при каждом из рассмотренных способов преобразования при использованиии техник Bag-of-the-Words и Теста χ^2 (Выбираемое количество свойств при максимумах - 7000)

Использование строк стека вызовов целиком	0.62
Использование строк стека вызовов целиком + Обработка текста сообщения	0.61
Разбиение строк стека вызовов на слова	0.61
Разбиение строк стека вызовов на слова + Обработка текста сообщения	0.63

Таблица 5: Максимальное значение F_1-меры, которой удалось добиться при каждом из рассмотренных способов преобразования при использованиии техники Bag-of-the-Words и с использованием решающих деревьев для уменьшения размерности

6.3. Двухэтапная классификация с использованием подсистем проекта

6.3.1. Нахождение зависимостей между разработчиками

При эксплуатации прототипа было замечено, что в большом количестве случаев, даже при ошибке предсказания, классификатор указывал на разработчика, работающего вместе с истинным разработчиком над одной и той же подсистемой.

Поскольку отчеты об исключительных ситуациях имеют очень мало информации о подсистемах из-за автоматической генерации, зачастую невозможно оценить, действительно ли подсистема, указанная в отчете, курируется обоими разработчиками (предсказанным и настоящим).

Однако для проверки замеченной закономерности была собрана статистика о том, какие разработчики над какими проектами работали. Эта статистика показала, что в случае ошибки (26% от общего числа) в 23% от общего числа тестовых элементов предсказанный и настоящий разработчики будут иметь пересечения по подсистемам. Поскольку разработчик работает обычно не более, чем над 4-5 подсистемами, это дает неплохой шанс, что предсказанный разработчик быстро и верно перенаправит отчет нужному.

6.3.2. Классификация по подсистемам

Интересной идеей для анализа является построение составного алгоритма, сперва определяющего подсистему, а затем определяющего человека среди сильно суженного круга разработчиков.

Для этого был проведен анализ подсистем, некоторые подсистемы, очень близкие по сути, были объединены в одну. Их количество составило 52 подсистемы.

Также подготавливались данные по исправленным отчетам, в которых была указана подсистема. Размер выборки данных составил 4724 отчета.

Было рассмотрено два подхода:

1. Использование голосующего алгоритма.

   Было решено отойти от идеи машинного обучения и рассмотреть идею голосования по заранее заданным правилам. Так как подсистемы в проекте обновляются гораздо реже, то и изменение правил надо проводить реже. Было решено ввести правила вида ”Подстрока” - ”Подсистема”. При обходе текста отчета об исключительной ситуации каждое вхождение подстроки давало голос за соответствующую подсистему. Однако точность такого подхо-

да в терминах общей точности составила меньше 40 процентов, поэтому от данной методики было решено отказаться.

2. Использование методов машинного обучения.

   Данные обрабатывались схожим с классификацией по разработчикам образом. Результат применения методов машинного обучения предоставлен в Таблице 6.

Метод	Total Accuracy, %	F_1
Логистическая регрессия	74	0.64
Наивный байесовский классификатор	61	0.50
Метод ближайших соседей	60	0.54
Машина опорных векторов с линейным ядром	76	0.63
Решающее дерево (CART)	70	0.52
Случайный лес	75	0.64
AdaBoost	75	0.61
Extra Gradient Boosting	77	0.62

Таблица 6: Результаты применения методов машинного обучения к задаче классификации на подсистемы

Среднеквадратичные отклонения при измерении общей точности составили не более 0.5% для всех методов. Среднеквадратичные отклонения при измерении усредненной F_1-меры составили не более 0.005 для всех методов. Лучших результатов в рассматриваемой метрике удалось достичь при использовании метода логистической регрессии.

Поскольку такой подход позволил добиться относительно неплохой точности предсказания, было решено собрать данные о разработчиках, сгруппированные по подсистемам и построить алгоритм, состоящий из двух частей: сперва при помощи метода логистической регрессии определялась подсистема, затем по собранным отчетам для данной подсистемы, имеющим ответственного разработчика, строился классификатор, определяющий разработчика в заданной подсистеме.

В качестве классификаторов для второго этапа было опробовано

несколько методов, однако лучших результатов удалось добиться при использовании машин опорных векторов, причем общая точность составила 24%, усредненная F_1-мера - 0.15.

Такой плохой результат связан, впервую очередь, с малым количеством отчетов об исключительных ситуациях, размеченных одновременно по подсистемам и разработчикам, а также с тем, что для многих отчетов определение подсистемы весьма неоднозначно, и выбирая подсистему, довольно часто исключался необходимый человек.

6.4. Использование частичного обучения для классификации по разработчикам

Было замечено, что помимо относительно небольшого набора размеченных данных проект содержит большое количество данных, не содержащих меток об исправляющем их разработчике или же содержащем их, но не имеющем подтверждения тому, что метка назначена правильно (примерно 34000).

Было решено применить методы частичного обучения, а именно метод Expectation-Maximization, неплохо показавший себя в задаче классификации текстов [10]. Для этого использовалась реализация, содержащаяся в библиотеке upclass для языка R [12].

80% процентов размеченных данных и все неразмеченные использовались для обучения классификатора, тестирование проводилось на оставшихся 20% размеченных данных.

При помощи метода Expectation-Maximization удалось достичь общей точности 50% при значении усредненной F_1-меры в 0.35. В связи с долгим обучением и недостаточной точностью от этого метода было решено отказаться.

6.5. Использование векторов вероятностей для улучшения предсказания

В данной секции рассматриваются результаты предсказаний классификатора не в виде меток разработчиков, а в виде векторов вероятностей, показывающих, с какой степенью уверенности классификатор относит отчет к каждому из рассматриваемых классов разработчиков.

В первую очередь, была собрана статистика, показывающая, в каком проценте случаев настоящий разработчик попадал в список из K разработчиков с максимальными вероятностями. Результаты предоставлены в Таблице 7:

K	Частота попаданий, %
1	74
2	79
3	85
4	88
5	91

Таблица 7: Процентное соотношение количества попаданий правильного разработчика в K лучших по мнению классификатора

6.5.1. Уточнение результата предсказания

Для K = 3 настоящий разработчик попадал в список из наиболее вероятных разработчиков в 85% случаев. Было опробовано несколько идей улучшения результата предсказания:

1. Было решено попробовать улучшить результат предсказания метода логистической регресии, основываясь на статистике для каждой тройки разработчиков.

   Опробованная идея заключалась в следующем: предположим, что для каждых 3 разработчиков есть статистика, собранная в ходе эксплуатации алгоритма, каждый элемент которой состоит из степеней уверенности классификатора в принадлежности отчета

каждому из трех разработчиков, а также информации о том, кто из этих трех разработчиков (если он есть среди них) ответственен за отчет на самом деле.

При эксплуатации прототипа было замечено, что довольно популярные для предсказателя разработчики часто становятся результатом неверного предсказания с небольшим отрывом. Возможно, можно было бы по имеющейся статистике построить небольшой классификатор, который разделил бы трехмерное пространство вероятностей так, чтобы уменьшить вероятность выбора этих людей.

Выбор K = 3 обуславливался балансом прироста общей точности и размером статистики (для бóльших значений K статистика для каждых K разработчиков содержит меньше элементов). Размеченные данные разбивались на 3 части:

- 60% данных использовались для обучения базового классификатора, который затем должен был быть улучшен;
- 20% данных, наравне с первыми 60%, использовались для построения обозначенной ранее статистики;
- на оставшихся 20% проводилось тестирование подхода.

Для совершения предсказания проводились следующие операции:

(a) Построенный базовый классификатор на основе логистической регрессии предсказывал какие разработчики и с какой степенью уверенности могут быть ответственны;

(b) Если для трех разработчиков с наибольшей степенью уверенности нет достаточной статистики, то в качестве окончательного результата возвращался разработчик с максимальной степенью уверенности;

(c) В противном случае, по статистике строился небольшой классификатор (методом логистической регрессии), который предсказывал, кого из троих выбрать.

Результат тестирования подхода:

- Усредненная F_1-мера: 0.63;
- Общая точность: 72%.

Таким образом, применение данного подхода не дало улучшения, и от него решено было отказаться.

2. Игнорирование некоторого набора разработчиков в случаях, когда их вероятности довольно близки к вероятностям претендентов.

   Данная идея основа на предположении, похожем на описанное в пункте 1, однако является менее автоматизированной. Можно выбрать несколько разработчиков, для которых тестирование алгоритма показало, что они слишком часто становятся результатами неверного предсказания (в силу, например, большого количества примеров из тренировочной выборки, принадлежащих им). Если для некоторого отчета степень уверенности для первого разработчика из этого списка отличается от степени уверенности для второго по популярности разработчика на величину, меньшую некоторого порога, то первый разработчик отбрасывается, в качестве предсказания возвращается второй.

   Был выбран набор разработчиков, для которых соотношение количества предсказанных на них элементов к числу реальных отчетов, принадлежащих им, было бы больше коэффициента 1.2. Были рассмотрены различные пороги, однако не удалось найти порог, при котором наблюдалось бы улучшение. Причиной данной проблемы может быть тот факт, что применение методики больше искажает предсказание для часто упоминающихся разработчиков, нежели помогает исправить результаты не принадлежащих им сообщений.

6.5.2. Классификатор с неопределенностью

Заключительным этапом стало рассмотрение возможности отказываться от предсказания. Не всегда целесообразно давать предсказания всем отчетам об исключительных ситуациях в системе, так как в случае неверного предсказания разработчик потратит дополнительное время, пытаясь осознать, верно ли, что это сообщение предназначается ему.

В связи с этим было решено установить порог предсказания - если степень уверенности классификатора меньше этого порога, то имеет смысл не предсказывать разработчика для данного сообщения и позволить, например, руководителю команды принять окончательное решение о судьбе отчета. На Рис. 2 показана зависимость количества допущенных предсказаний от порога.

В ходе тестирования был выбран порог - 0.55, что позволило снизить вероятность ошибки с 26% от общего размера данных, до 16% при общем количестве проигнорированных отчетов в 17%.

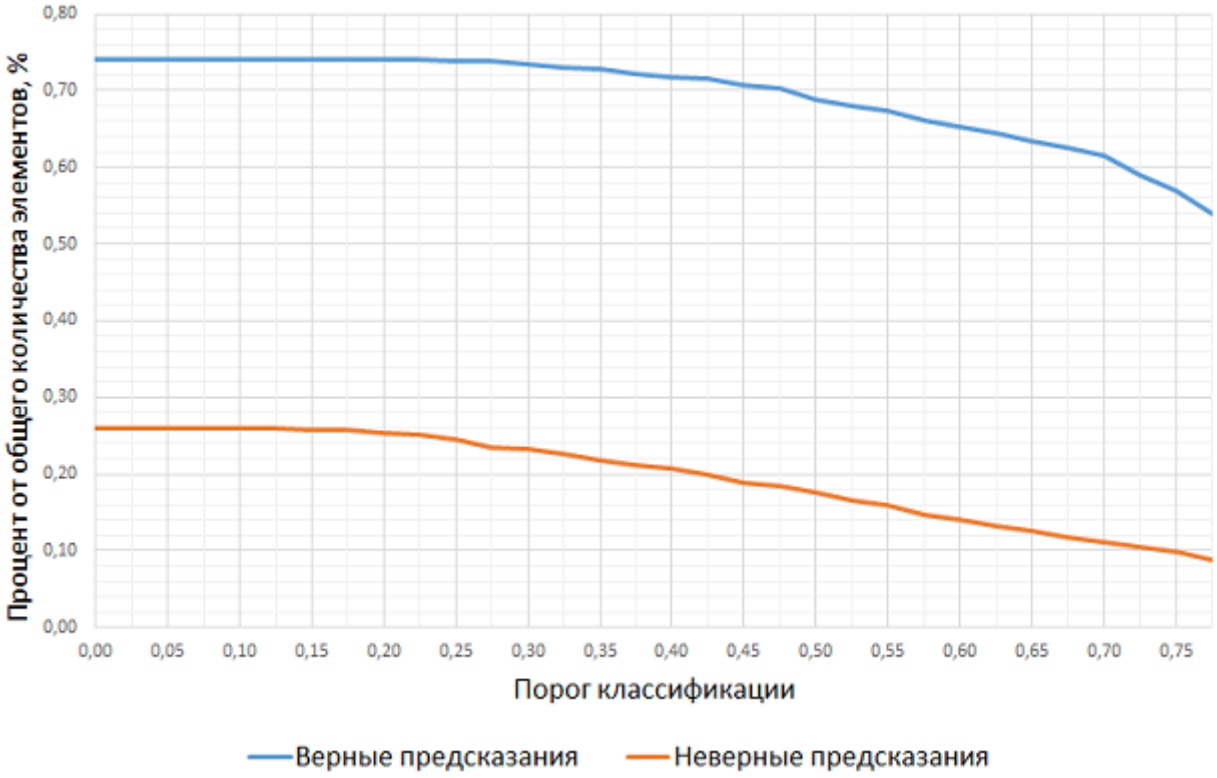

Рис. 2: Зависимости общего количества верных и неверных предсказаний от выбора порога классификации

7. Заключение

В данной работе была рассмотрена задача автоматического назначения ответственного за отчет об исключительных ситуациях проекта ReSharper. Были рассмотрены различные способы преобразования данных, была рассмотрена возможность применения различных классификаторов, в том числе и составных. Были проведены сравнение и анализ результатов.

Исходя из полученных данных, был выбран классификатор на основе метода логистической регрессии, реализованного в библиотеке scikit-learn, обладающий лучшими показателями усредненной по всем классам F_1-меры, выбранной в качестве оценки точности - при помощи него удалось достить значения в 0.63. При этом данный классификатор также показал лучшие результаты при измерении процента правильно предсказанных отчетов - он верно определял разработчика в 74% случа-

ев, что является хорошим результатом в рассматриваемой области (см. Рис. 1), а также новым результатом для более конкретной задачи классификации отчетов об исключительных ситуациях, показывающим, как уточнение области может влиять на способ подготовки данных и применяемые алгоритмы классификации.

На основе данного метода был реализован анализатор, позволяющий определять разработчика, который вероятнее всего ответственен за данный отчет об ошибке. Данный компонент встраивается в сервер, занимющийся обменом сообщений с системой отслеживания ошибок.

Для уменьшения вероятности выдачи неверного предсказания было добавлено игнорирование результатов предсказания, обладающих недостаточно высокими вероятностными показателями. Это позволило снизить вероятность ошибки до 16% от общего размера тестовой выборки при общем размере количества проигнорированных элементов в 17%.

Также, была добавлена возможность предсказания нескольких наиболее вероятных, по мнению классификатора, разработчиков с их числовыми оценками вероятностей. Так, процент попаданий настоящего разработчика в список из трех наиболее подходящих для тестовой выборки равен 85%, поэтому данная статистика может использоваться как дополнительная информация для команды разработчиков.

Литература

[1] Anvik J., Hiew L., Murphy G.C. Who should fix this bug? // ICSE. — 2006. — P. 361–370.

[2] Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts / L. Jonsson, M. Borg, D. Broman et al. // Empirical Software Engineering. — 2014.

[3] Bhattacharya P., Neamtiu I., Shelton C.R. Automated, highly-accurate, bug assignment using machine learning and tossing graphs // The Journal of Systems and Sofstware. — 2012. — Vol. 85, no. 10. — P. 2275–2292.

[4] Breiman L. Random forests // Machine learning. — 2001. — Vol. 45, no. 1. — P. 5–32.

[5] Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. — 2016.

[6] Cubranic D., Murphy G.C. Automatic bug triage using text categorization // SEKE. — 2004. — P. 92–97.

[7] Freund Y., Schapire R.E. A Short Introduction to Boosting // Journal of Japanese Society for Artificial Intelligence. — 1999. — P. 771–780.

[8] Genkin A., Lewis D.D., Madigan D. Sparse Logistic Regression for Text Categorization. — 2004.

[9] Gunn S.R. Support Vector Machines for classification and regression // Technical report, University of Southampton, Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science. — 1998.

[10] McCallum A., Nigam K., Mitchell T. Semi-supervised text classification using EM // MIT Press. — 2006.

[11] Ramos J. Using TF-IDF to Determine Word Relevance in Document Queries // Technical report, Department of Computer Science, Rutgers University. — 2003.

[12] Russell N., Cribbin L., Murphy T.B. Updated Classification Methods using Unlabeled Data. — 2013. — URL: `https://cran.r-project.org/web/packages/upclass/upclass.pdf` (online; accessed: 11.03.2016).

[13] Scikit-learn: Machine Learning in Python / F. Pedregosa, G. Varoquaux, A. Gramfort et al. // Journal of Machine Learning Research. — 2011. — Vol. 12. — P. 2825–2830.

[14] Sokolova M., Lapalme G. A systematic analysis of performance measures for classification tasks // Information Processing and Management 45. — 2009. — P. 427–437.

[15] A Study on Mutual Information-based Feature Selection for Text Categorization / Y. Xu, G. Jones, J. Li et al. // Journal of Computational Information Systems. — 2007. — P. 1007–1012.

[16] Yang Y., Pedersen J. O. A comparative study on feature selection in text categorization // ICML-97, 14th International Conference on Machine Learning. — 1997. — P. 412–420.

[17] A comparative study of decision tree ID3 and C4.5 / B. Hssina, A. Merbouha, H. Ezzikouri, M. Erritali // International Journal of Advanced Computer Science and Applications. — 2014. — P. 13–19.

[18] An empirical study on bug assignment automation using Chinese bug data / Z. Lin, F. Shu, Y. Yang et al. // 3rd International Symposium on Empirical Software Engineering and Measurement. — 2009. — P. 451–455.

[19] Воронцов К.В. Лекции по метрическим алгоритмам классификации. — 2008. — URL: `http://www.ccas.ru/voron/download/MetricAlgs.pdf` (online; accessed: 03.02.2016).

[20] Воронцов К.В. Лекции по линейным алгоритмам классификации. — 2009. — URL: http://www.machinelearning.ru/wiki/images/6/68/voron-ML-Lin.pdf (online; accessed: 29.12.2015).