> 7 минут выступление + 3 минуты демонстрация и ответы на вопросы

Слайд 1

Приветственные слова.

Слайд 2

Одной из актуальных проблем разработки программного обеспечения является наличие дефектов, то есть логических ошибок в коде программы, приводящих к снижению качества продукции. Особенно это становится актуальным при большом объеме текста программы. Затраты на выявление и устранение дефектов могут составлять до 80% от общей стоимости ПО. При этом чем раньше будет обнаружен дефект, тем меньше ущерба будет нанесено разработчику и эксплуатанту ПО. Существует множество способов и методов для обнаружению дефектов. В данной работе предлагается использовать новое решение – методы машинного обучения, которые на основе ранее написанного кода позволят дать вероятностную оценку нахождения дефекта. На данном слайде представлены цель и задачи данной работы.

Слайд 3

На следующем слайде представлена классификация разновидностей дефектов по характеристическим признакам.

Слайд 4

Для обучения моделей обнаружения дефектов ПО во многих статьях используются классические алгоритмы машинного обучения, решающие задачу классификации. В результате объекты помечаются как положительные (имеются дефекты) и отрицательные (дефекты отсутствуют). Для оценки качества работы полученных моделей

используются такие метрики, как accuracy, precision, recall и F-мера. Их формулы представлены на следующем слайде.

Слайд 5

Градиентный бустинг показал наивысший результат. Можно сделать вывод, что его применение является наиболее выгодным для рассматриваемой задачи.

Слайд 6

В разрабатываемом методе предполагается, что на вход системе подается код, состоящий из блоков с функциями, каждая из которых занимает не более 2000 строк. Система путем статического анализа с использованием алгоритма градиентного бустинга определяет и размечает в коде блоки, подверженные дефектам.

Слайд 7

Данный метод включает в себя несколько этапов. На данном слайде можно увидеть детализированную IDEF0-диаграмму. Первым этапом код, для которого нужно определить наличие дефектов, делится на блоки, содержащие функции.

Слайд 8

Следующим этапом является подсчет метрик кода Маккейба и Холстеда для каждого блока. Данные метрики вы можете видеть на экране.

Слайд 9

Обучение модели градиентного бустинга происходит на данных из репозитория PROMISE. Данный набор представляет собой вычисленные метрик кода для модулей, написанных на C++. Градиентный бустинг является разновидностью ансамблей деревьев решений. Деревья в нем обучаются последовательно, при этом каждое новое учитывает ошибки предыдущих, за счет чего достигается большая точность. На данном слайде

вы видите схему данного алгоритма. Первое дерево строится обычным образом, обучаясь на входных данных. Каждое последующее – путем оптимизации функции потерь. Ансамбль строится до тех пор, пока не будет достигнут критерий остановки – достигнуто заданное количество итераций.

Слайд 10

Для решения исследуемой задачи используется логарифмическая функция потерь. На каждом шаге значение данного выражения необходимо минимизировать. Для этого в качестве целевой переменной при построении очередного дерева используется антиградиент – значение, противоположное градиенту.

Слайд 11

Итоговая модель градиентного бустинга представляет композицию деревьев. Сумма предсказаний каждого дерева является вероятностью, с которой во входной программе имеется дефект. Однако такой подход может привести к переобучению. Для решения этой проблемы вводят специальный параметр, называемый темпом обучения.

Слайд 12

Вычисленные значения метрик для блоков поступают обученной модели, где происходит классификация по наличию или отсутствию дефекта. Данный набор необходимо классифицировать на каждом из деревьев. По формуле, представленной на предыдущем слайде считается общий прогноз. Итог работы – определение наличия дефекта во входной программе и соответствующая вероятность его присутствия. Последним этапом метода является разметка: необходимо графически отметить в коде вероятности наличия дефектов для каждого блока.

Слайд 13

Реализованное программное обеспечение имеет многомодульную структуру. Каждый модуль соответствует определенному этапу разработанного метода.

Слайд 14

Система обнаружения дефектов ПО представляет собой приложение. Пользователь имеет возможность загрузить файл с кодом на C++ либо ввести его вручную. Данный код размечается разными цветами в зависимости от вероятности нахождения в функции дефектов.

Слайд 15

Для оценки точности обучающая выборка делится на тренировочную и тестовую. Тестовая часть составляет 20% от общего количества. На каждой итерации считается значение функции потерь, что позволяет понять, правильно ли идет обучение. Видно, что график имеет гиперболическую форму и на более обученной модели становится линейным. Это говорит о том, что обучение проходит верно, функция потерь постепенно становится ниже и ближе к концу выходит на плато.

Слайд 16

Основным средством анализа точности является подсчет рассмотренных ранее метрик: accuracy, precision, recall и f-мера. Видно, что разработанный метод является довольно точным, все значения метрик близки к единице и результатам существующих реализаций. Стоит отметить, что accuracy и recall превышают другие показатели. В рамках рассматриваемой задачи это является преимуществом, так как лучше отнести сомнительный код к дефективным, чем пропустить и не обратить внимания пользователя на него.

Слайд 17

Для оценки времени происходит генерация функций с различным количеством строк, каждая из которых состоит из операторов и операндов. Для данной функции считаются метрики и происходит предсказание результата моделью. Даже для максимальных значений количества строк время показывает довольно низкие значения. График имеет линейный характер, что говорит о прямой зависимости между временем выполнения и количество строк в функции.

Слайд 18

В результате выполнения выпускной квалификационной работы была достигнута цель и решены все поставленные задачи.

Перейдем к демонстрации разработанного ПО.