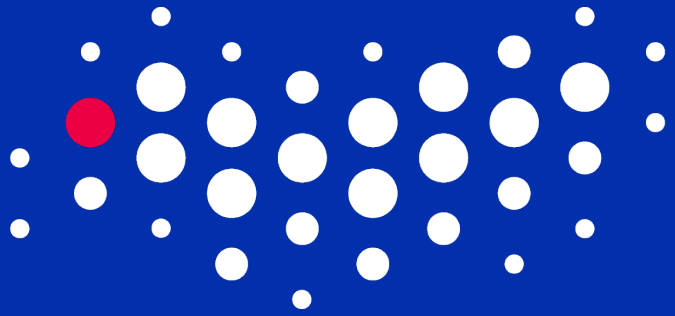


ITMO UNIVERSITY

Обнаружение проблем производительности в программах на языке программирования Kotlin с использованием статического анализа кода

Петухов Виктор

Кафедра Компьютерных технологий

Актуальность

- Проблемы в производительности программ могут возникают не только по вине программиста — пользователя языка программирования
- Часть проблем может заключаться в компиляторе языка — неоптимальной кодогенерации или некорректных оптимизациях
- Разработчикам языка Kotlin необходимо исследование, позволяющее выявить такие проблемы

Существующие решения по оценке производительности программ

- Профайлеры — инструменты для анализа производительности во время исполнения программы
- Инструменты, использующие статические анализаторы, основанные на эвристиках (например, performance issues в IDE)
- Performance-тесты (например, JMH — Java Microbenchmark Harness)

Постановка задачи

- Будем решать задачу обнаружения аномалий, уже поставленную в области машинного обучения
- Будем искать точечные аномалии
- Будем пробовать трактовать обнаруженные аномалии как проблемы производительности:
  - По причинам на стороне пользователя
  - По причинам на стороне компилятора (кодогенератора/оптимизатора)

Аргументация выбора алгоритмов машинного обучения

- Потенциальные проблемы производительности может повлечь не только объемный код
- В область наших интересов также попадает «нетипичный код» — редкое сочетание конструкций языка, приводящее к генерации нетипичного байт-кода, который может иметь проблемы производительности
- Итого: малоизученные места могут быть источником проблем производительности
- Для такой плохо формализуемой задачи алгоритмы машинного обучения оказываются более эффективными, чем классические алгоритмы

Научная новизна

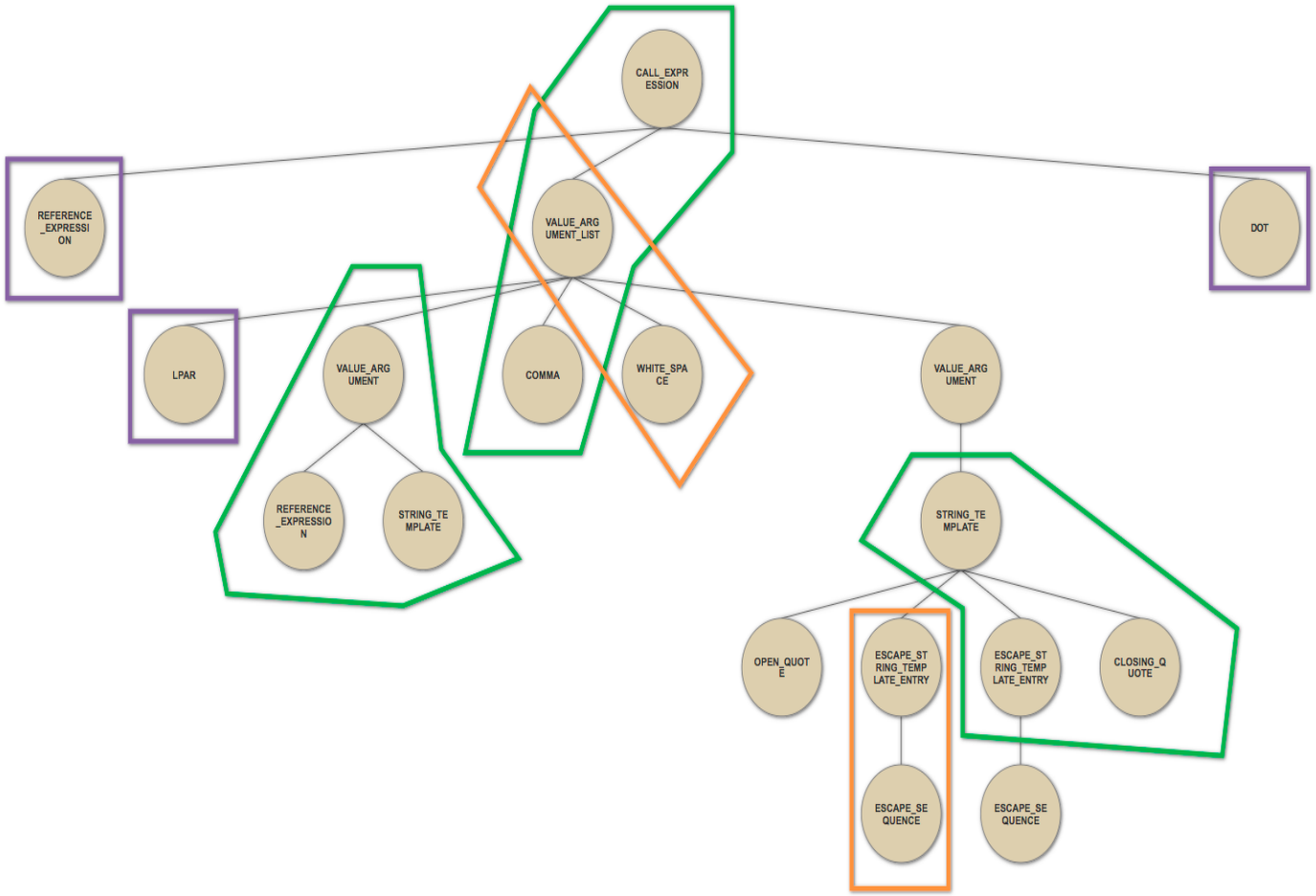
- ...заключается в анализе влияния компонентов компилятора на производительность полученных программ с применением алгоритмов машинного обучения

Сбор данных

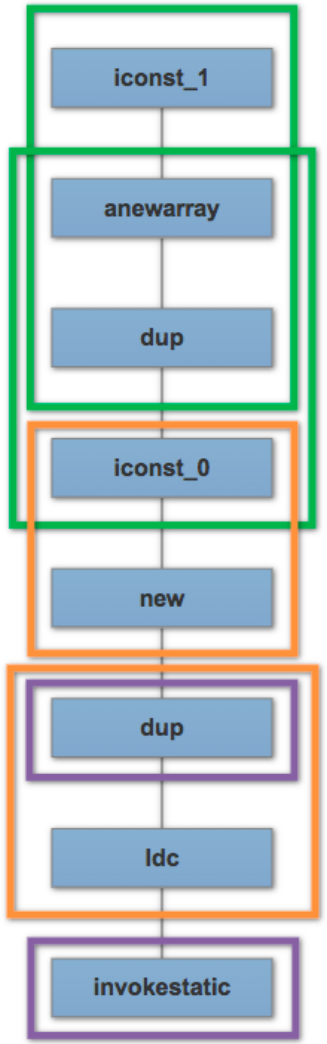
- В качестве набора данных для экспериментов были выбраны файлы с исходным кодом на Kotlin и JVM байт-кодом с ресурса Github
- Было собрано **930 тыс.** файлов с исходным кодом на Kotlin
- Было собрано **63 тыс.** файлов с JVM байт-кодом
- Файлы с JVM байт-кодом были сопоставлены файлам с исходным кодом

Подготовка данных. Задача факторизации

- Для алгоритмов машинного обучения требуются векторы — объекты с наборами признаков
- Встает задача факторизации — векторного представления дерева разбора и байт-кода
- Был выбран способ факторизации дерева путем разложения его на n-граммы (unigram, bigram и 3-gram)
- Для байт-кода (набора JVM-инструкций) был выбран аналогичный способ



Факторизация дерева разбора



Факторизация байт-кода

Результаты факторизации

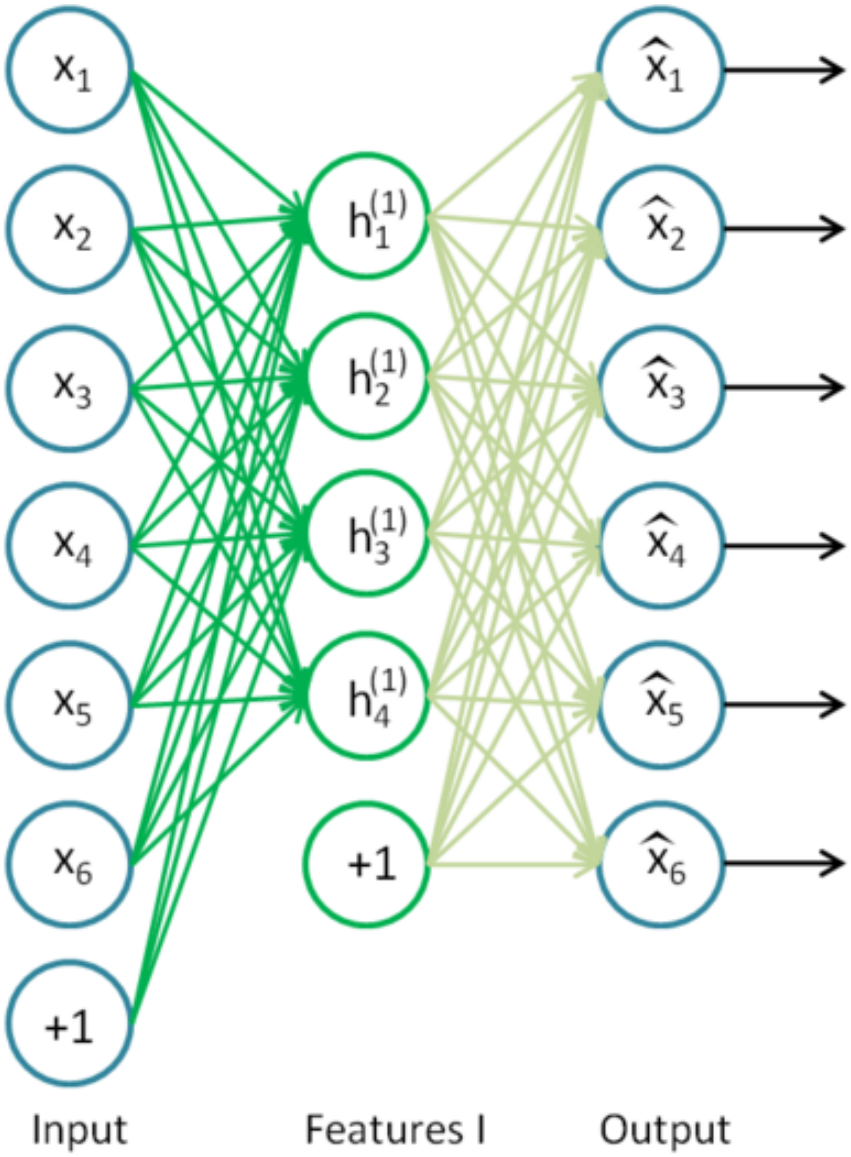
- В результате по набору деревьев разбора на Kotlin было извлечено **11 тыс.** n-грамм
- По набору JVM байт-кода было извлечено **110 тыс.** n-грамм
- Таким образом, были составлены следующие наборы данных:
  - По деревьям разбора — **930 тыс.** примеров, **11 тыс.** признаков
  - По JVM байт-коду — **63 тыс.** примеров, **110 тыс.** признаков

Методы решения задачи обнаружения аномалий

- Метод опорных векторов с одним классом (OC SVM)
- Методы, основанные на кластеризации (например, k-means)
- Метод k-ближайших соседей (k-nearest neighbor)
- Статистические методы
- Репликаторные нейросети, автоэнкодеры

Автоэнкодер. Обоснование выбора

- Для решения задачи был выбран автоэнкодер:
  - Обучение без учителя (примеров аномалий изначально нет)
  - Не двоичный выход — ошибку восстановления можно трактовать как «степень аномальности»
  - Встроенные механизмы сокращения размерности — можно формировать для этого отдельные слои в архитектуре сети: признаков очень много, нужно уметь сжимать информацию



- Простая архитектура — один скрытый слой
- Несколько экспериментов с коэффициентом сжатия **0.8** и **0.5**
- ап

Результаты

- По результатам работы автоэнкодера были вычислены евклидовы расстояния между входными и выходными векторами
- Расстояния трактовались как «степени аномальности»
- Аномалиями считались векторы, расстояния которых отклонялись от среднего на более чем 3-сигма или 5-сигма
- По 5-сигма было получено **1150** аномалий, по 3-сигма — **545**
- Полученные аномалии были сгруппированы по классам

```
class ConfigModel(val configs: Config) : ViewModel() {
    val ip = bind { configs.ipProperty }
    val dataBase = bind { configs.dataBaseProperty }
    val rootUser = bind { configs.rootUserProperty }
    val password = bind { configs.passwordProperty }
    val tableName = bind { configs.tableNameProperty }
    val entityName = bind { configs.entityNameProperty }
    val entityPackage = bind { configs.entityPackageProperty }
    val mapperPackage = bind { configs.mapperPackageProperty }
    val servicePackage = bind { configs.servicePackageProperty }
}
```

```
{
  "getIp" : [ "aload_0", "getfield", "areturn" ],
  "getDataBase" : [ "aload_0", "getfield", "areturn" ],
  "getRootUser" : [ "aload_0", "getfield", "areturn" ],
  "getPassword" : [ "aload_0", "getfield", "areturn" ],
  "getTableName" : [ "aload_0", "getfield", "areturn" ],
  "getEntityName" : [ "aload_0", "getfield", "areturn" ],
  "getEntityPackage" : [ "aload_0", "getfield", "areturn" ],
  "getMapperPackage" : [ "aload_0", "getfield", "areturn" ],
  "getServicePackage" : [ "aload_0", "getfield", "areturn" ],
  "getConfigs" : [ "aload_0", "getfield", "areturn" ],
  "<init>" : [ "aload_1", "ldc", "invokestatic", "aload_0", ... (4428 instructions)
}
```

```
public object CabalTokelTypes {
    val COLON                   : IElementType  = HaskellTokenType(":")
    val COMMA                   : IElementType  = HaskellTokenType(",")
    val COMMENT                 : IElementType  = HaskellTokenType("COMMENT")
    val OPEN_PAREN              : IElementType  = HaskellTokenType("(")
    val STRING                  : IElementType  = HaskellTokenType("string")
    val NUMBER                  : IElementType  = HaskellTokenType("number")
    val TAB                     : IElementType  = HaskellTokenType("TAB")
    val NEGATION                : IElementType  = HaskellTokenType("!")
    val COMMENTS                : TokenSet = TokenSet.create(END_OF_LINE_COMMENT, COMMENT)
    val WHITESPACES             : TokenSet = TokenSet.create(TokenType.WHITE_SPACE)
    val PROPERTY_KEY            : IElementType = CabalCompositeElementType("PROPERTY_KEY", ::PropertyKey)

   (and 78 more similar assignment statements)
```

```
{
  "<clinit>" : [ "ldc", "invokestatic", "putstatic", "new", "invokespecial", "return" ],
  "getCOLON" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getCOMMA" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getCOMMENT" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getOPEN_PAREN" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getSTRING" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getNUMBER" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getTAB" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getNEGATION" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getCOMMENTS" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getWHITESPACES" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)
  "getPROPERTY_KEY" : [ "getstatic", "dup", "ifnonnull", "new", "dup", "ldc_w", "ldc_w", "anewarray", "dup", ... (20 instructions)

  (and 78 more methods with similar bytecode)

  "<init>" : [ "aload_0", "invokespecial", "aload_0", "checkcast", "putstatic", "new", "dup", ... (846 instructions)
}
```

http://victor.am — список всех аномалий

big companion object

many safe calls

many root function definitions

many var or fun definitions

big and complex enums

big init method in bytecode

many similar call expressions

big static arrays or map

nested calls

big multiline strings

big code hierarchy

many square bracket annotations

long enumerations

long calls chain

many concatenations

many consecutive arithmetic expressions

many not null assertion operators

many case in when

complex or long logical expressions

many type reified params

many function arguments

many delegate properties

many inline functions

big constants set

many nested structures

many if statements

many generic parameters

big methods

many loops

many literal strings

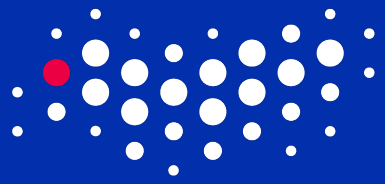
many assignment statements

Верификация результатов

- Для оценки качества результатов были привлечены эксперты — разработчики компилятора Kotlin
- Был разработан специальный веб-сервис, позволяющий давать оценку той или иной аномалии
- В результате планируется посчитать среднюю оценку по найденным аномалиям и тем самым оценить качество результатов
- Часть аномалий уже была включена в тесты на производительность компилятора Kotlin

Планы

- Запуск поиска аномалий на определенных срезах набора данных:
  - С заданными ограничениями на кол-во узлов / глубину в дереве разбора
  - С заданными ограничениями на объем JVM байт-кода
- Проведение экспериментов с оценкой влияния привнесенных в язык фичей (с перспективной разработки системы тестов на такой системе)
- Обучение классификатора на сгруппированном наборе аномалий для дальнейшей автоматической классификации и обнаружения новых классов
- Проведение аналогичного исследования в Kotlin/Native и Kotlin/JS



ITMO UNIVERSITY

Спасибо за внимание!

Петухов Виктор

Кафедра Компьютерных технологий

i@victor.am

Санкт-Петербург, 2018