软件分析

机器学习与软件分析

熊英飞
北京大学
2017

从软件分析到机器学习

- 需要把软件分析问题转换成机器学习问题
- 软件分析问题
  - 输入：程序源代码、文档或其它软件制品
  - 输出：取决于问题
    - 程序是否有特定类型的缺陷
    - 变量可能指向的位置
    - ……
- 机器学习问题
  - 输入：特征向量，训练集
  - 输出：分类中的特定类别
- 转换方法：
  - 程序源代码等 => 特征向量
  - 分类 => 软件分析问题的输出
  - 训练集收集方法

程序源代码等 => 特征向量

- 主要挑战：
  - 程序并非简单的向量结构
  - 同一程序可以有多种不同写法
  - 软件制品中包含大量自定义标识符
- 解决方法：
  - 针对问题设计特征向量
  - 特征向量提取过程对写法不同的程序有一定包容能力
  - 特征向量通常不依赖于自定义标识符

分类 => 软件分析问题的输出

- 主要挑战：
  - 大量软件分析问题都不可能用简单模型解决
    - 预测停机问题？
    - 预测程序中是否有内存泄露？
  - 大量软件分析问题并不是简单分类问题
    - 指向分析——类别太多，不利于机器学习算法学习
- 解决方案：
  - 解决传统方法不容易解决，但机器学习容易解决的问题
  - 解决问题中的一些关键点而不是整体问题

训练集收集

- 主要挑战
  - 软件分析问题常常难以人工计算输出
- 解决方案
  - 从软件历史数据等信息中寻找输出
  - 考虑可以自动计算的输出
    - 利用机器学习来加速

本节内容

- 缺陷预测问题
- 测试排序问题
- 静态分析加速问题
- 代码补全问题
- 树卷积神经网络
- 自然语言到程序的转换

缺陷预测

缺陷预测

- 输入：给定一个程序
- 输出：程序中可能有缺陷的文件/方法，按优先级排列

- 主要作用：决定测试的顺序
  - 传统静态分析不容易解决
  - 不能完全解决问题，但是比随机强

程序 => 特征向量

- 常见代码级别特征：

Metric Name	Description (applies to method level)
fanIN	Number of methods that reference a given method
fanOUT	Number of methods referenced by a given method
localVar	Number of local variables in the body of a method
parameters	Number of parameters in the declaration
commentTo CodeRatio	Ratio of comments to source code (line based)
countPath	Number of possible paths in the body of a method
complexity	McCabe Cyclomatic complexity of a method
execStmt	Number of executable source code statements
maxNesting	Maximum nested depth of all control structures

与人常犯的错误有关，与代码本地特性无关，包容代码的不同写法

表格来自：Giger, D'Ambros, Pinzger, Gall. Method Level Bug Prediction. ESEM'12

程序历史 => 特征向量

Metric Name	Description (applies to method level)
methodHistories	Number of times a method was changed
authors	Number of distinct authors that changed a method
stmtAdded	Sum of all source code statements added to a method body over all method histories
maxStmtAdded	Maximum number of source code statements added to a method body for all method histories
avgStmtAdded	Average number of source code statements added to a method body per method history
stmtDeleted	Sum of all source code statements deleted from a method body over all method histories
maxStmtDeleted	Maximum number of source code statements deleted from a method body for all method histories
avgStmtDeleted	Average number of source code statements deleted from a method body per method history
churn	Sum of $stmtAdded - stmtDeleted$ over all method histories
maxChurn	Maximum *churn* for all method histories
avgChurn	Average *churn* per method history
decl	Number of method declaration changes over all method histories
cond	Number of condition expression changes in a method body over all revisions
elseAdded	Number of added else-parts in a method body over all revisions
elseDeleted	Number of deleted else-parts from a method body over all revisions

表格来自：Giger, D'Ambros, Pinzger, Gall. Method Level Bug Prediction. ESEM'12

训练集

- 从软件项目历史中提取
- 一个提交为修复提交，如果满足如下条件
  - 修改的行数小于一个阈值，且
  - 带有fix, repair等关键字
  - 或者包含某个issue id
- 所有修改了的方法/文件被认为是有缺陷的
  - 分类问题：方法/文件是否有缺陷
  - 回归问题：方法/文件是否多少缺陷

最新论文预测效果

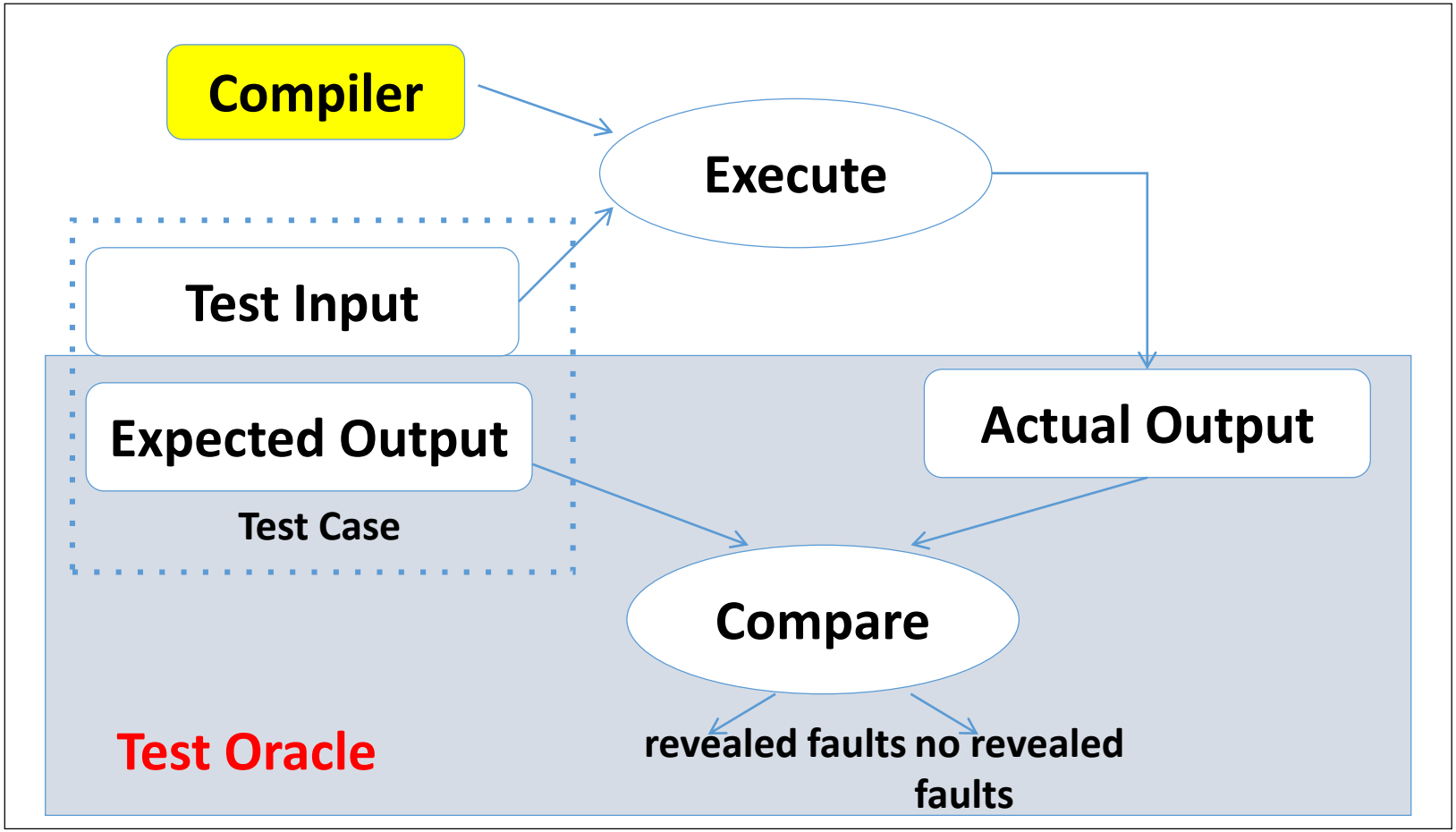
- Automatically Learning Semantic Features for Defect Prediction. Song Wang, Taiyue Liu and Lin Tan. In the International Conference on Software Engineering. Acceptance Rate: 19% (101/530)

- PROMISE数据集结果
  - 项目内平均64.1，最高94.2
  - 项目间平均56.8，最高97.9

测试排序问题

机器学习解决关键难点

Compiler Testing
--- guaranteeing compiler quality



Software Testing Process

随机编译器测试

- 测试输入：随机程序生成工具CSmith
  - 大型公司往往开发内部的随机程序生成工具
- 测试预言：
  - RDT: 用两个不同的编译器执行测试输入，看结果是否一致
  - DOL: 用编译器的不同优化级别，看结果是否一致
  - EMI: 把原程序变异成等价程序，看结果是否一致

随机编译器测试的问题

- 随机测试的效率很低
- RDT花了3年在GCC和LLVM上检查出来325个Bug
- EMI花了11个月在GCC和LLVM上检查出来147个Bug

如何加速编译器测试？

- 先跑能发现Bug的测试程序
- 先跑时间短的测试程序

- 假设程序发现错误的能力为C，运行时间为t，则C/t越大的程序越先执行

- 如何快速知道C和t？
  - 用机器学习预测

程序=>特征向量

- 特征选择
  - 基本思路：复杂的程序编译时容易出错，且运行时间可能长
  - 存在特征
    - 是否存在某个类型的语句
    - 是否存在某个类型的变量
    - ……
  - 使用特征
    - 程序中变量被读取的次数
    - 程序中指针被引用的次数
    - 程序中指针被比较的次数
    - ……

训练集

- 在旧版本的GCC和LLVM上发现Bug的测试程序和没有发现Bug的测试程序
- 各自1000个，保持数量平衡

用机器学习预测C和t

- 预测C
  - 采用线性SVM模型
- 预测t
  - 采用高斯过程回归——一种回归模型
- 特征选择：预先采用特征选择来排除无效特征
- 运行结果：绝大多数时候加速30%-60%

静态分析加速问题

Hakjoo Oh, Hongseok Yang, Kwangkeun Yi: Learning a strategy for adapting a program analysis via bayesian optimisation. OOPSLA 2015: 572-588

机器学习解决关键难点

自动生成训练集

复习：敏感性选择

1. a=100;
2. if(a>0)
3. a=-a;
4. b=a+1;

流敏感符号分析

$a_1 = 正$

$b_1 = b_0$

$a_2 = a_1$

$b_2 = b_1$

$a_3 = \neg a_2$

$b_3 = b_2$

$a_4 = a_3 \sqcap a_2$

$b_4 = a_3 \sqcap a_2 + 正$

流非敏感符号分析

$a = a \sqcap 正 \sqcap \neg a$

$b = b \sqcap a + 正$

敏感性实际可以混合

1. a=100;
2. if(a>0)
3. a=-a;
4. b=a+1;

流敏感符号分析

$a_1 = 正$
$b_1 = b_0$
$a_2 = a_1$
$b_2 = b_1$
$a_3 = \neg a_2$
$b_3 = b_2$
$a_4 = a_3 \sqcap a_2$
$b_4 = (a_3 \sqcap a_2) + 正$

流非敏感符号分析

$a = a \sqcap 正 \sqcap \neg a$
$b = b \sqcap a + 正$

部分流敏感分析

$a_1 = 正$
$a_2 = a_1$
$a_3 = \neg a_2$
$a_4 = a_3 \sqcap a_2$
$b = b \sqcap (a_3 \sqcap a_2) + 正$

速度高于流敏感分析
精度高于流非敏感分析

如何为变量选择敏感性？

- 敏感性越高
  - 结果越精确
  - 花费时间越长
- 希望能尽可能提高结果精度，减少时间开销
- 机器学习决定每个变量的精确度

程序=>特征向量

- 为每个变量生成一个特征向量

#	Features
1	local variable
2	global variable
3	structure field
4	location created by dynamic memory allocation
5	defined at one program point
6	location potentially generated in library code
7	assigned a constant expression (e.g., x = c1 + c2)
8	compared with a constant expression (e.g., x < c)
9	compared with an other variable (e.g., x < y)
10	negated in a conditional expression (e.g., if (!x))
11	directly used in malloc (e.g., malloc(x))
12	indirectly used in malloc (e.g., y = x; malloc(y))
13	directly used in realloc (e.g., realloc(x))
14	indirectly used in realloc (e.g., y = x; realloc(y))
15	directly returned from malloc (e.g., x = malloc(e))
16	indirectly returned from malloc
17	directly returned from realloc (e.g., x = realloc(e))
18	indirectly returned from realloc
19	incremented by one (e.g., x = x + 1)
20	incremented by a constant expr. (e.g., x = x + (1+2))
21	incremented by a variable (e.g., x = x + y)
22	decremented by one (e.g., x = x - 1)
23	decremented by a constant expr (e.g., x = x - (1+2))
24	decremented by a variable (e.g., x = x - y)
25	multiplied by a constant (e.g., x = x * 2)
26	multiplied by a variable (e.g., x = x * y)
27	incremented pointer (e.g., p++)
28	used as an array index (e.g., a[x])
29	used in an array expr. (e.g., x[e])
30	returned from an unknown library function
31	modified inside a recursive function
32	modified inside a local loop
33	read inside a local loop

机器学习和训练集

- 方法1【非原论文内容】：
  - 给定一组程序，首先对所有变量进行流非敏感分析
  - 对每个变量，单独提升其精度为流敏感，测试分析时间的增量和结果准确度的增量
  - 准确度增量/时间增量 > 阈值 为正例，否则为负例
  - 可以用任意的机器学习模型
- 方法2【论文内容】：
  - 假设决策函数是线性函数
  - 用一个启发式搜索决定所有的线性函数的参数值
  - 以限定时间内达到的精确度作为Fitness函数
  - 相当于自己做了一个学习算法

实验结果

	Training				Testing								
	FI	FS	partial FS		FI		FS			partial FS			
Trial	prove	prove	prove	quality	prove	sec	prove	sec	cost	prove	sec	quality	cost
1	6,383	7,316	7,089	75.7 %	2,788	48	4,009	627	13.2 x	3,692	78	74.0 %	1.6 x
2	5,788	7,422	7,219	87.6 %	3,383	55	3,903	531	9.6 x	3,721	93	65.0 %	1.7 x
3	6,148	7,842	7,595	85.4 %	3,023	49	3,483	1,898	38.6 x	3,303	99	60.9 %	2.0 x
4	6,138	7,895	7,599	83.2 %	3,033	38	3,430	237	6.2 x	3,286	51	63.7 %	1.3 x
5	7,343	9,150	8,868	84.4 %	1,828	28	2,175	577	20.5 x	2,103	54	79.3 %	1.9 x
TOTAL	31,800	39,625	38,370	**84.0 %**	14,055	218	17,000	3,868	**17.8 x**	16,105	374	**69.6 %**	**1.7 x**

Table 4. Effectiveness of our method for flow-sensitivity. prove: the number of proved queries in each analysis (FI: flow-insensitivity, FS: flow-sensitivity, partial FS: partial flow-sensitivity). quality: the ratio of proved queries among the queries that require flow-sensitivity. cost: cost increase compared to the FI analysis.

在FI和FS之间达到较好权衡

代码补全的统计模型

为代码设计的语言模型

Probabilistic CFG

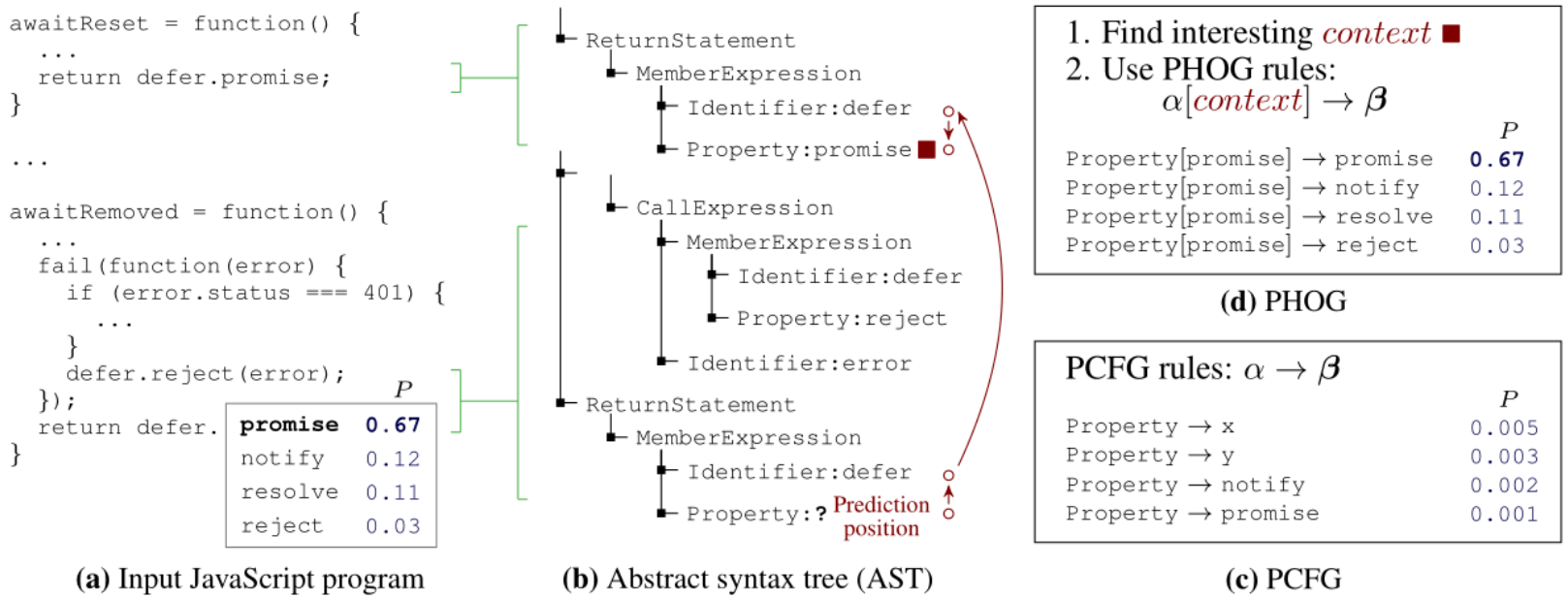
- 给每个推导式附一个概率
  - A ->B C　　0.8
  - -> C　　0.2
  - B -> X “,” B　　0.5
  - -> X　　0.5
- 同一个非终结符出发的推导式概率和为1
- 训练方法：
  - $\frac{\text{所有}AST\text{树上该推导式出现次数}}{\text{该推导式左边终结符出现次数}}$
- 可以用作代码补全

PHOG

- PHOG=Probabilistic Higher Order Grammar
- PCFG相当于用于预测的特征只有当前非终结符
- 成功的预测需要提取更多特征，称为Context

- 相关论文：
  - PHOG: Probabilistic Model for Code. Pavol Bielik, Veselin Raychev, Martin Vechev. ACM ICML 2016
  - Learning Programs from Noisy Data. Veselin Raychev, Pavol Bielik, Martin Vechev, Andreas Krause. ACM POPL 2016
  - Code Completion with Statistical Language Models. Veselin Raychev, Martin Vechev, Eran Yahav. ACM PLDI 2014

PHOG示例：javascript方法补全



提取同一个对象上的上一次操作作为Context

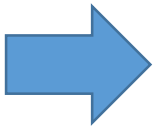
PHOG定义

- 非终结符[Context] -> 符号1 符号2 … 符号n　概率
- 同时存在一个函数，从已有的代码中提取Context
- 同一个非终结符和同一个Context的推导式概率和为1
- 训练方法：
  - $$\frac{\text{所有}AST\text{树上该推导式在该}context\text{下的出现次数}}{\text{该推导式左边终结符在该}context\text{下的出现次数}}$$

更复杂的Context示例

- 基于变量之前的n次使用情况预测下一个API调用

```
Stream s = new FileStream(“c:\book.txt”);
Reader r = new TextReader(s);
r.read();
s.?();
r.?();
```



s的context：
<new FileStream, ret>
<new TextReader, 1>

r的context：
<new TextReader, ret>
<read, 0>

变量使用位置编号：ret = 0.method(1, 2, 3, …)

Context定义语言

- 把Context获取过程定义为从当前结点出发的AST树的遍历

```
   Ops ::=  ϵ | Op Ops
    Op ::= WriteOp | MoveOp
WriteOp ::= WriteValue | WritePos | WriteAction
MoveOp ::= Up | Left | DownFirst | DownLast | PrevDFS |
           PrevLeaf | PrevNodeType | PrevActor
```

- 之前的示例可以写作

```
Left   PrevActor WriteAction WritePos
       PrevActor WriteAction WritePos
```

Left：左边的兄弟结点（即补全的目标对象）
PrevActor：前一次该变量的使用
WriteAction：使用时调用的方法
WritePos：该Actor在AST树中的位置

图统计模型

- PHOG的Context仍然是序列结构
- 代码的本质是图结构，能否在统计时把图结构利用起来？

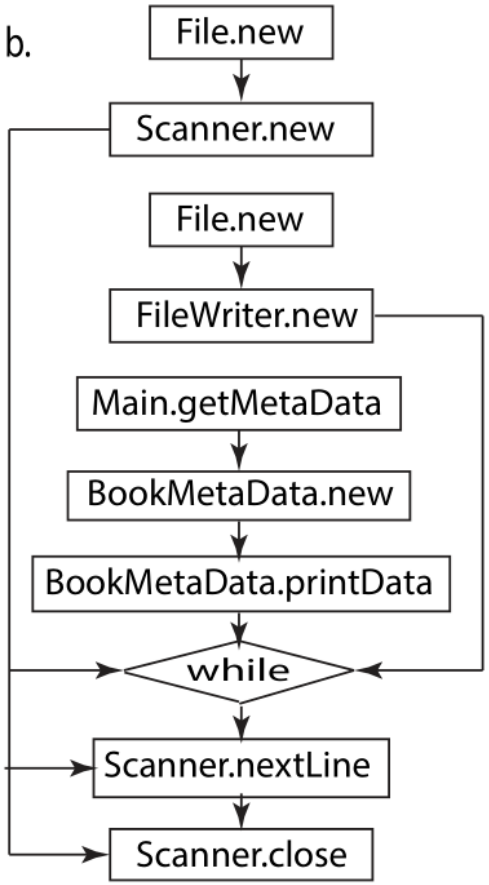
- 相关论文：
  - Nguyen, Anh Tuan, and Tien N. Nguyen. "Graph-based statistical language model for code." ICSE 2015.
  - Nguyen, Tung Thanh, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. "Graph-based mining of multiple object usage patterns." FSE 2009

API使用图Groum

a.

```
1  File bookFile = new File("books.txt");
2  Scanner bookSc = new Scanner(bookFile);
3
4  File authorFile = new File("authors.txt");
5  FileWriter authorFW= new FileWriter
6                                  (authorFile);
7  BookMetaData metaData =
8           getMetaData("bookMetaData.txt");
9  metaData.printData();
10
11 while ( ) {
12     bookSc.nextLine();
13 }
14
15 bookSc.close();
```

b.



- 结点是API调用或者if, while
- 边为数据依赖或者控制依赖

图上的API调用补全问题

- 在图上加上一个新节点，问该结点为哪个API调用的时候概率最大
  - 即寻找G’，最大化P(G’|G)，其中G’比G多一个API调用节点
- 经典机器学习问题：样本不足以支持取样
- 解决方案：
  - 朴素贝叶斯+n-gram

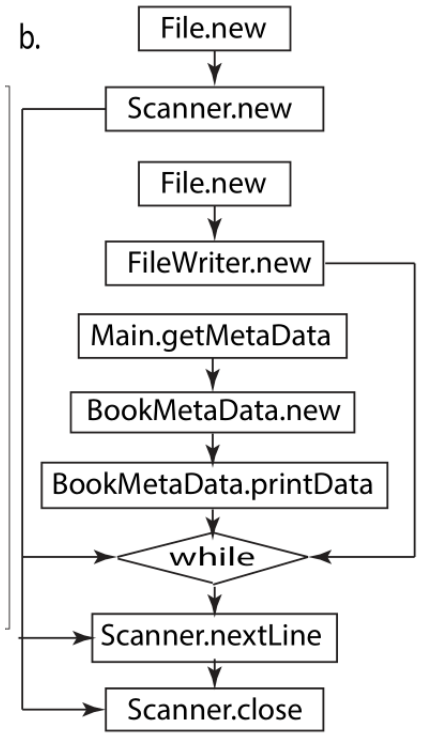
基于图的API调用补全

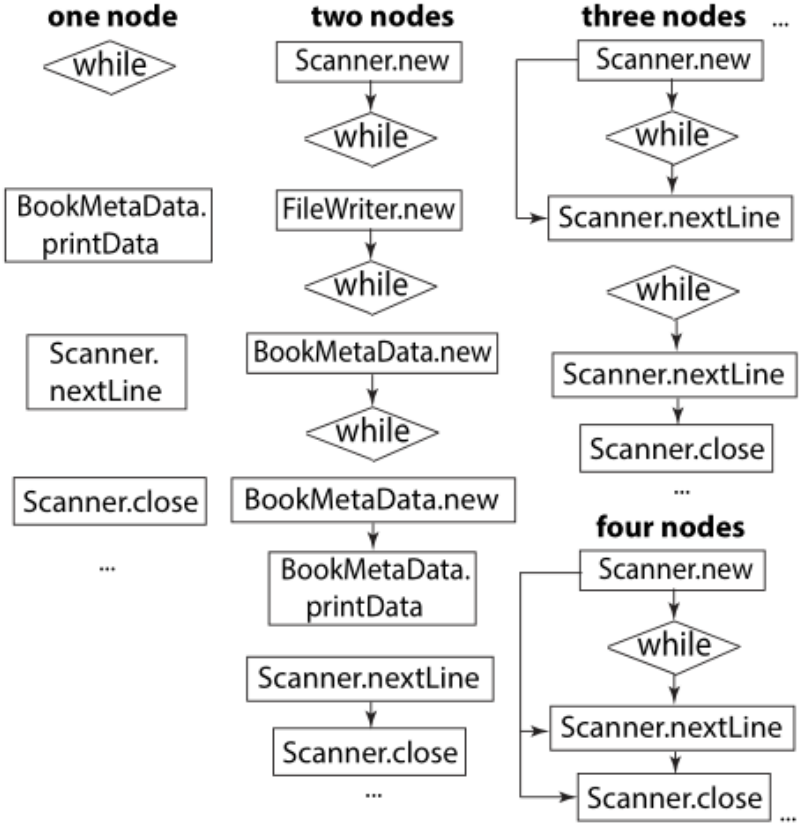
a.

```
1  File bookFile = new File("books.txt");
2  Scanner bookSc = new Scanner(bookFile);
3
4  File authorFile = new File("authors.txt");
5  FileWriter authorFW= new FileWriter
6                                  (authorFile);
7  BookMetaData metaData =
8           getMetaData("bookMetaData.txt");
9  metaData.printData();
10
11 while ( ) {
12     bookSc.nextLine();
13 }
14
15 bookSc.close();
```

1. 寻找m个距离补全点最近的节点，为上下文节点
   - 距离用文本上的字符数衡量
   - 令m=4，则包括
     - BookMetaData.printData
     - While
     - Scanner.nextLine
     - Scanner.close

基于图的API调用补全





2. 找到图上所有节点数<=n的子图（仅考虑包含节点间所有边的子图），选出仅包含上下文节点的子图，形成集合G

基于图的API调用补全

- 对于集合G中的每一个子图和每一个API，形成新的子图h
- 用朴素贝叶斯求概率
  - $P(h|G) = \frac{P(G|h)P(h)}{P(G)}$
  - $P(G|h) \approx P(g_1|h)P(g_2|h) \dots P(g_n|h)$
- 其中
  - $P(g_i|h) \approx \frac{\text{同时包含}g_i\text{和}h\text{的方法的数量}}{\text{包含}h\text{的方法的数量}}$
  - $P(h) \approx \frac{\text{包含}h\text{的方法的数量}}{\text{所有方法的数量}}$

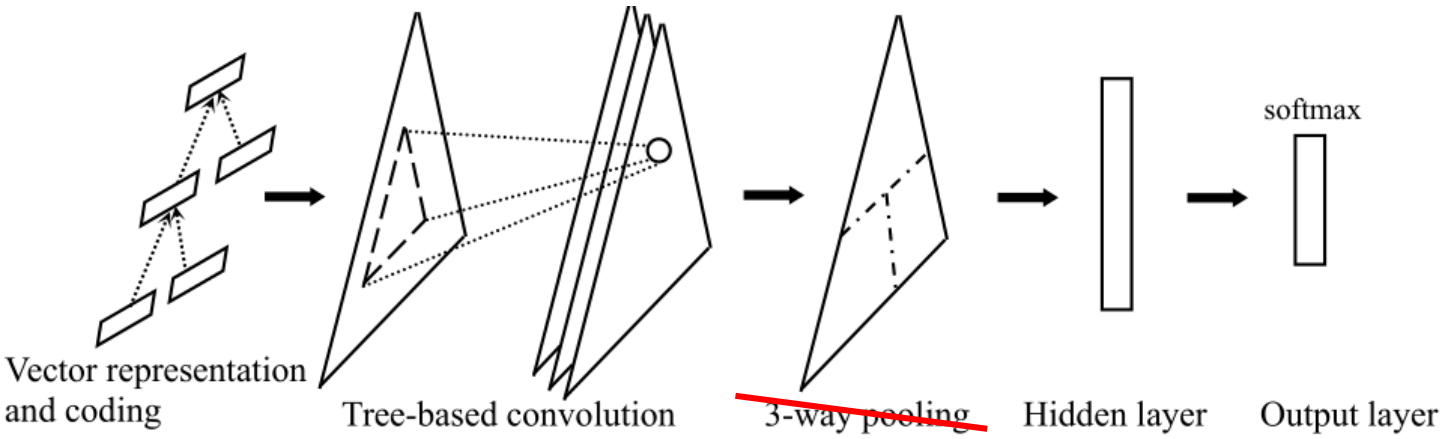
树卷积神经网络

为代码设计的机器学习通用模型

Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin.
"Convolutional neural networks over tree structures for programming language processing." AAAI 2016.

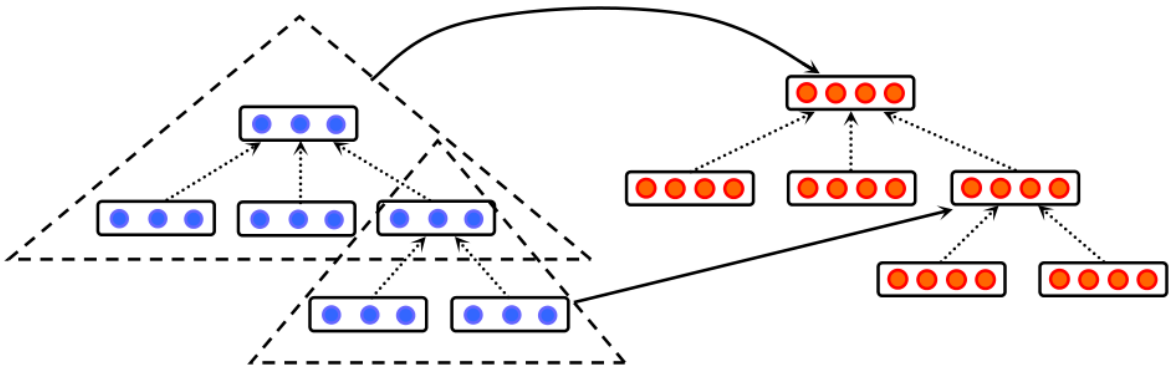
树卷积神经网络

- 在AST树上做卷积



1-way pooling

树卷积



$$\boldsymbol{y} = \tanh\left(\sum_{i=1}^{n} W_{\text{conv},i} \cdot \boldsymbol{x}_i + \boldsymbol{b}_{\text{conv}}\right)$$

- 主要问题：给定深度为d的窗口，n的大小不固定，怎么办？
- 解决方法：通过对固定参数线性加权来产生任意长度的参数列表
  - $\mathrm{W}_{conv,i} = \eta_i^t W^t + \eta_i^l W^l + \eta_i^r W^r$
  - $\eta_i^t = \frac{d_i - 1}{d - 1}$，其中$d_i$为节点i的深度
  - $\eta_i^r = (1 - \eta_i^t)\frac{p_i - 1}{n - 1}$，其中$n$为节点i所有兄弟（含节点i自己）的数量，$p_i$为节点i的在兄弟中位置
  - $\eta_i^l = (1 - \eta_i^t)\,(1 - \eta_i^r)$

树卷积神经网络效果

- 104道POJ编程题目，每道题500个答案
- 让神经网络自动分类每个答案应该属于哪个题目
- 准确率：94.0%

非代码分析示例：API调用序列查询

非代码上的机器学习问题

Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, Sunghun Kim, Deep API Learning, FSE 2016

API调用序列查询问题

$DocumentBuilderFactory.newInstance$

↓

$DocumentBuilderFactory.newDocumentBuilder$

↓

$DocumentBuilder.parse$

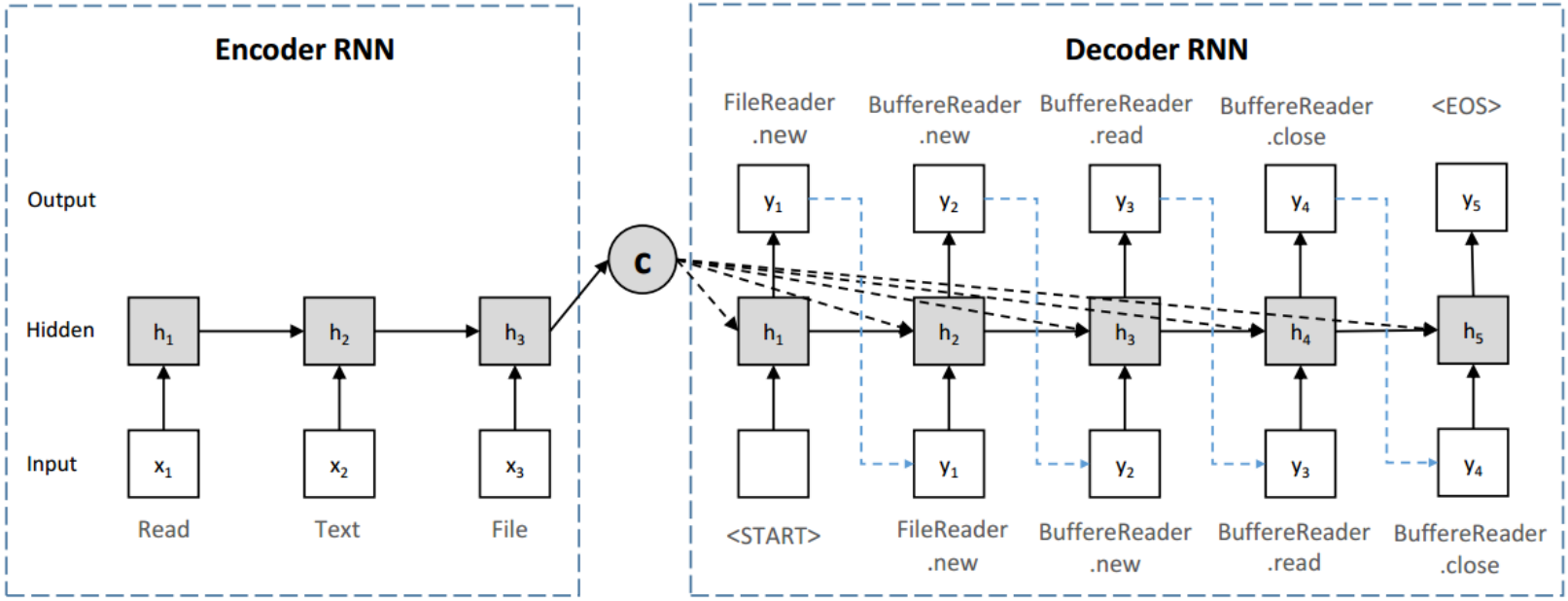
训练集收集

```
/***
 * Copies bytes from a large (over 2GB) InputStream to an OutputStream.
 * This method uses the provided buffer, so there is no need to use a
 * BufferedInputStream.
 * @param input the InputStream to read from
 *  . . .
 * @since 2.2
 */
public static long copyLarge(final InputStream input,
   final OutputStream  output, final byte[] buffer) throws IOException {
   long count = 0;
   int n;
   while (EOF != (n = input.read(buffer))) {
      output.write(buffer, 0, n);
      count += n;
   }
   return count;
}
```

API sequence: InputStream.read → OutputStream.write
Annotation: copies bytes from a large inputstream to an outputstream.

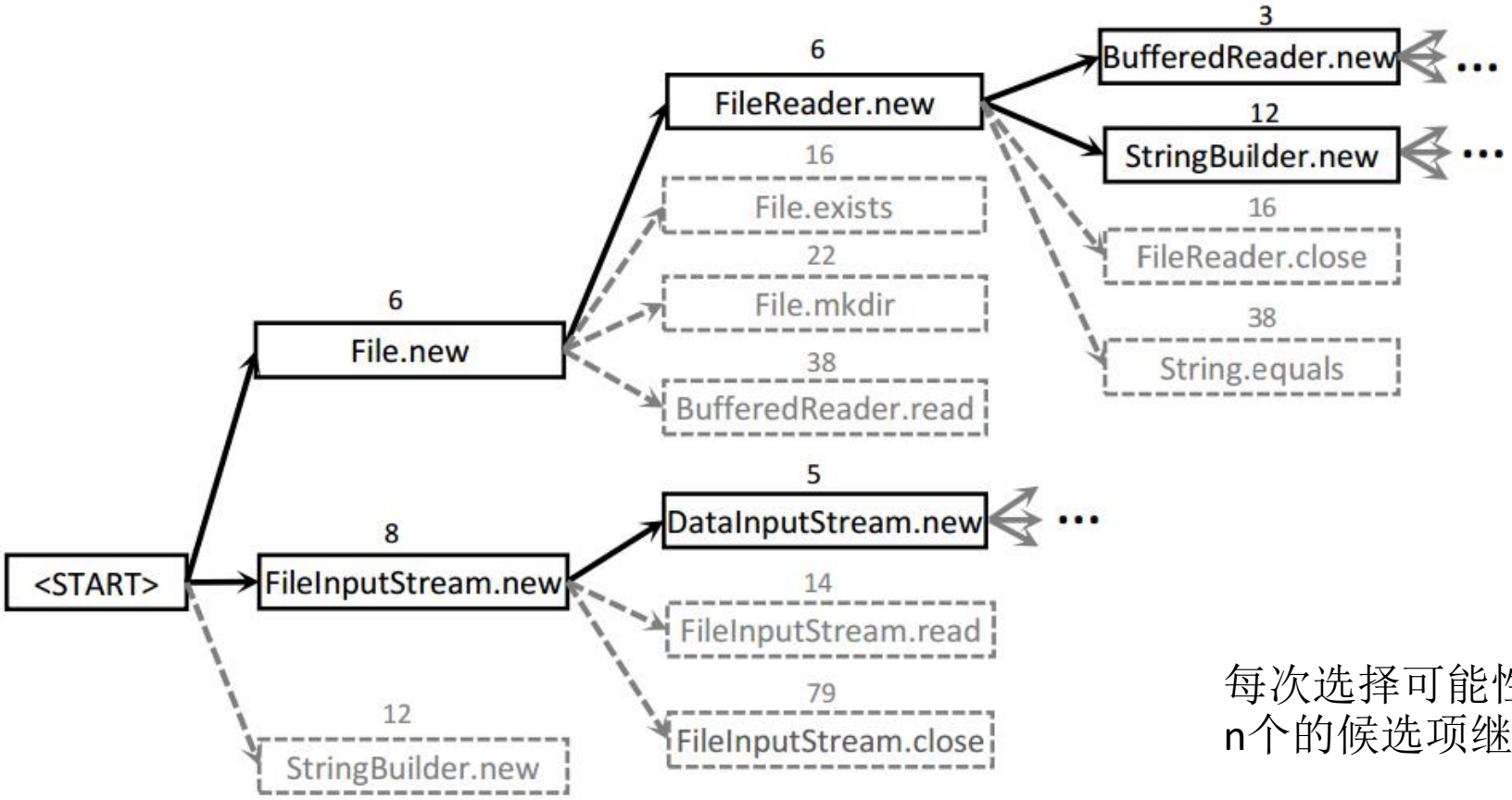
- 从控制流提取API调用序列
- 循环中的语句只考虑一次
- 注释第一句作为查询
- API方法为一个预先定义好的大小固定的集合

RNN翻译模型



- 利用一个单独的神经网络将单词映射成向量（称作Word Embedding）

基于Beam搜索的生成



每次选择可能性最大的n个的候选项继续展开

实验效果

- 串的匹配用BLEU指标来衡量
  - $BLEU = BP \prod_{n=1}^{N} \frac{\text{出现在参考输出中的}n-gram\text{的数量}}{\text{输出中}n-gram\text{的数量}}$
  - N为所考虑的n-gram的最大值
  - BP为较短序列的惩罚，令c为输出序列的长度，r为参考序列的长度，则 $BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$
  - 易见BLEU为0-1之间的数，越大越好
- 实验结果
  - BLEU=54.42%，之前的方法都不到20%