S-CAT切分

由于原数据集中每个测试文件的数据量较大，同时为了便于针对性对测试数据进行总结，我们决定对原测试文件进行切分，将全部众包测试工作者的测试event数据切分成一条条合适长度的测试event序列，一条合适长度的event序列定义如下。

合适长度n的event序列：一个入口event+其他event

一个入口event定义为：打开应用程序、进入新界面等标志一项测试开始始的event

其他event定义为：除入口event之外的event

原文件为168个测试人员的测试文件组成的大长度event序列，我们将切分划为2个部分，第一步为基于相邻时间差的中等长度event序列的切分，切分后得到377个基于大时间切分的中等长度event序列，第二步为基于FP-growth明显event类型的小长度序列切分，切分后得到993个基于事件类型的小长度event序列，经过以上两步切分后过滤，可以得到814份合理长度event序列集合。

**SYMBOL DEFINITION**

|  |  |
| --- | --- |
| SYMBOL | DEFINITION |
| t | 设置的相邻event时间差 |
| n | 合理序列长度n |
| min\_sup | 最小支持度阈值 |
| D | 事务数据集 |
| N | 树节点Node |
| FP-Tree | 构建完成的FP树 |
| F | 频繁项集合 |
| P | 去除p之后的频繁项集合 |
| αf | 频繁项 |
| β | 路径节点组合 |

第一步：基于相邻event时间差的中等长度event序列的切分

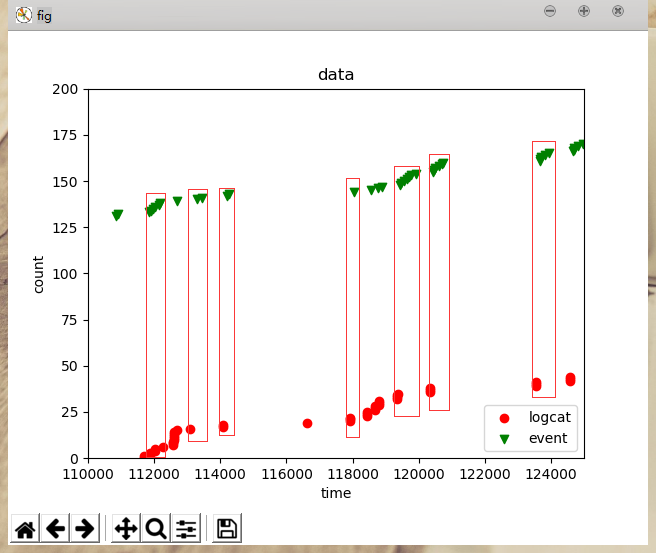


图1

图1是一份经过时间标注的众包测试人员的数据文件，将预处理后的event数据和logcat数据按照系统时间的顺序在图中标注出来，由上图可以看出文件中包含多段event序列，每段event序列之间有明显的时间差，基于该项特征我们考虑基于时间差对数据进行切分。切分算法为算法1。

算法1:基于相邻eventevent时间差切分

输入：预处理后的event数据集eventFiles

输出：经时间差算法切分后的中等长度event序列middleList

1. middleList = []
2. For eventList in eventFiles:
3. event\_middle\_sequence = []
4. For i in range(0,eventList.length-1):
5. time\_difference = eventList[i+1].SyscTime-eventList[i].SyscTime
6. If time\_difference<t:
7. event\_middle\_sequence.append(eventList[i])
8. Else:
9. event\_middle\_sequence.append(eventList[i])
10. If event\_middle\_sequence.length>n:
11. middleList.append(event\_middle\_sequence)
12. event\_middle\_sequence = []

在时间差参数设置方面，我们统计了各个event时间差频次的最大最小值。

|  |  |  |
| --- | --- | --- |
| 时间差  出现频次i | 频次大于i的前20高频的时间差最小值（单位：ms） | 频次小于等于i的前20低频的时间差最大值（单位：ms） |
| 1 | 448 | 21396 |
| 2 | 371 | 10001 |
| 3 | 319 | 9992 |
| 4 | 292 | 1984 |
| 5 | 281 | 1253 |

对于合理长度参数的设置，我们做了以下工作考虑用event的系统时间参数与logcat对应，寻找logcat出现‘Error'，’Warning'的最短最长序列，从而定义有意义测试序列的长度。

基于时间序列的切分后

|  |  |  |  |
| --- | --- | --- | --- |
| event总数量 | 事件序列包含‘E'，’W‘的logcat最短event长度 | <3数量 | >=3数量 |
| 452 | 3 | 75 | 377 |

由以上数据可以看出包含与事件序列对应logcat中含有“Warning”，“Error”的最短长度为3，由于“Warning”，“Error”是强显性的bug标识，所以我们设置合理序列长度为包含“W”，“E”的最短事件序列长度，即n=3.

基于上述数据和安卓系统日志生成规则我们将时间差t设置为10s（1000ms），合理序列长度n最小值设置为3，通过以上算法我们由168个测试人员的测试文件组成的大长度event序列切分得到了452个测试人员的测试文件组成的大长度event序列。为了证明于相邻时间差的中等长度event序列的切分算法的合理性，我们通过FP-growth算法来证明。

**算法：FP-Tree构造算法**

输入：事务数据集 D，最小支持度阈值 min\_sup

输出：FP-Tree

(1) 　　扫描事务数据集 D 一次，获得频繁项的集合 F 和其中每个频繁项的支持度。对 F 中的所有频繁项按其支持度进行降序排序，结果为频繁项表 L ;

(2) 　　创建一个 FP-Tree 的根节点 T，标记为“null”;

(3) 　　for 事务数据集 D 中每个事务 Trans do

(4) 　　　　对 Trans 中的所有频繁项按照 L 中的次序排序;

(5) 　　　　对排序后的频繁项表以 [p|P] 格式表示，其中 p 是第一个元素，而 P 是频繁项表中除去 p 后剩余元素组成的项表;

(6) 　　　　调用函数 insert\_tree( [p|P], T );

(7) 　　end for

**insert\_tree( [p|P], root)**

(1) 　　if root 有孩子节点 N and N.item-name=p.item-name then

(2) 　　　　N.count++;

(3) 　　Else

(4) 　　　　创建新节点 N;

(5) 　　　　N.item-name=p.item-name;

(6) 　　　　N.count++;

(7) 　　　　p.parent=root;

(8) 　　　　将 N.node-link 指向树中与它同项目名的节点;

(9) 　　end if

(10)　　if P 非空 then

(11)　　　　把 P 的第一项目赋值给 p，并把它从 P 中删除;

(12)　　　　递归调用 insert\_tree( [p|P], N);

(13)　　end if

**算法：FP-Growth(FP-Tree, α);**

输入：已经构造好的 FP-Tree，项集 α（初值为空），最小支持度 min\_sup;

输出：事务数据集 D 中的频繁项集 L;

(1) 　　L 初值为空

(2) 　　if Tree 只包含单个路径 P then

(3) 　　　　for 路径 P 中节点的每个组合（记为 β） do

(4) 　　　　　　产生项目集 α∪β，其支持度 support 等于 β 中节点的最小支持度数;

(5) 　　　　　　return L = L ∪ 支持度数大于 min\_sup 的项目集 β∪α

(6)　　 else　　　　//包含多个路径

(7) 　　　　for Tree 的头表中的每个频繁项 αf do

(8) 　　　　　　产生一个项目集 β = αf∪α，其支持度等于 αf 的支持度;

(9) 　　　　　　构造 β 的条件模式基 B，并根据该条件模式基 B 构造 β 的条件 FP- 树 Treeβ;

(10) 　　　　　　if Treeβ≠Φ then

(11) 　　　　　　　　递归调用 FP-Growth(Treeβ, β);

(12) 　　　　　　end if

(13) 　　　　end for

(14)　　end if

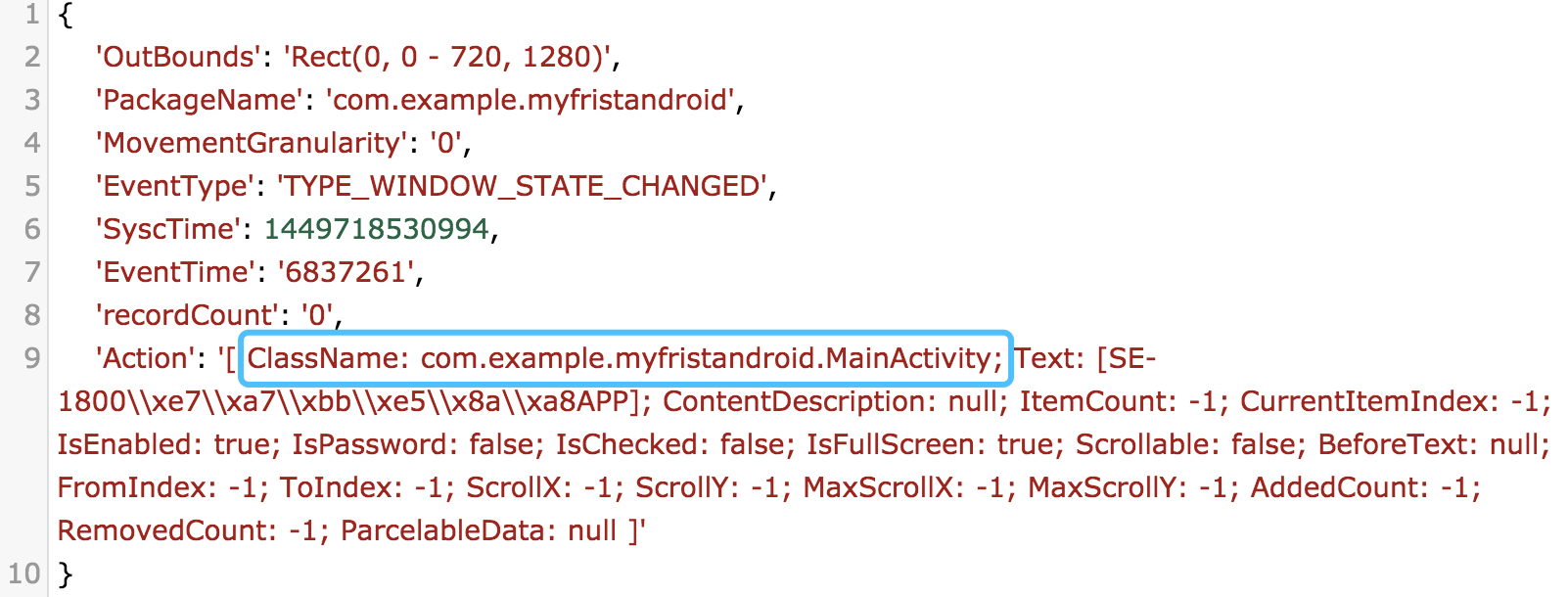
在基于相邻时间差的中等长度event序列的切分合理性证明中，我们选择event类型EventType为特征项，通过比较切分前后的明显特征来评判切分的合理性，取最小支持度为150，实验结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类型 | 切分前 | | 切分后 | | 准确率 |
| 全部序列数量 | 168 | | 377 | |
| TYPE VIEW CLICKED的序列数量 | 168 | 100% | 343 | 91.0% | 91.0% |
| TYPE WINDOW STATE CHANGED的序列数量 | 168 | 100% | 327 | 86.7% | 86.7% |
| TYPE WINDOW CONTENT CHANGED的序列数量 | 168 | 100% | 366 | 97.1% | 97.1% |
| TYPE VIEW SCROLLED的序列数量 | 157 | 93.5% | 229 | 60.7% | 64.9% |

由以上可以看出，切分前的最高前三项频繁event类型（TYPE VIEW CLICKED，TYPE WINDOW STATE CHANGED，TYPE WINDOW STATE CHANGED）在切分后仍排在前三项，且相对准确率均在85%以上，从而证明了基于相邻时间差的中等长度event序列的切分算法的合理性。

第二步：基于FP-growth明显event类型的小长度序列切分

在实验一的基础上，对TYPE VIEW CLICKED，TYPE WINDOW STATE CHANGED，TYPE WINDOW CONTENT CHANGED这三种高频eventType做进一步挖掘频繁项，选取了action中的className作为第二特征。



结合EventType和ClassName双特征基于FP-growth明显event类型的小长度序列切分，取最小支持度为200，实验结果如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EventType | ClassName | 数量 | 总数量 | 占比 |
| TYPE\_WINDOW\_CONTENT\_CHANGED | android.widget.FrameLayout | 303 | 377 | 80.4% |
| TYPE\_WINDOW\_CONTENT\_CHANGED | com.example.myfristandroid.MainActivity | 301 | 377 | 79.8% |
| TYPE\_WINDOW\_STATE\_CHANGED | com.example.myfristandroid.graphWebShow | 211 | 377 | 56.0% |
| TYPE\_VIEW\_CLICKED | android.widget.RelativeLayout | 203 | 377 | 53.8% |

由以上结果得，应考虑TYPE\_WINDOW\_STATE\_CHANGED：com.example.myfristandroid.MainActivity和TYPE\_WINDOW\_CONTENT\_CHANGED：android.widget.FrameLayout两类event作为切分标识进行切分。通过https://developer.android.com/training/basics/firstapp/creating-project了解到TYPE\_WINDOW\_STATE\_CHANGED：com.example.myfristandroid.MainActivityevent为应用程序主Activity，即应用程序入口点，当应用程序运行时，系统会启动此activity的实例并加载布局，因此我们选择TYPE\_WINDOW\_STATE\_CHANGED：com.example.myfristandroid.MainActivity作为入口event对已经过时间差切分的序列进行二次切分，得到993份小长度的event序列。

对于合理长度参数的设置，我们做了以下工作考虑用event的系统时间参数与logcat对应，寻找logcat出现‘Error'，’Warning'的最短最长序列，从而定义有意义测试序列的长度。

基于fpgrowth的切分后

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 总数量 | 时间包含‘E'，’W‘的logcat最短event长度 | 时间包含‘E'，’W‘的logcat最长event长度 | <2数量 | >38 | 2-38 |
| 993 | 2 | 39 | 151 | 28 | 814 |

由以上数据可以看出包含与事件序列对应logcat中含有“Warning”，“Error”的最短长度为3，由于“Warning”，“Error”是强显性的bug标识，所以我们设置合理序列长度为包含“W”，“E”的最短事件序列长度，即n∈[2,38]。对其进行合理长度event事件过滤（n∈[2,38]），得到814份合理长度event序列，至此，序列切分工作已基本完成。下面将结合logcat对合理长度event序列的分析和总结。