SE 465 Software Testing, Quality Assurance, and Maintenance
Project/Lab B Detailed Description of Feature Collection, Version 1

Instructor: Lin Tan
Lab Instructor: Thibaud Lutellier
Project Lead: Song Wang
Release Date: March 6, 2018

Data for Online Defect Detection Model

Table 1: Evaluated projects for change-level defect prediction. **Lang** is the programming language used for the project. **LOC** is the number of the line of code. **First Date** is the date of the first commit of a project, while **Last Date** is the date of the latest commit. **Changes** is the number of changes collected in this work. **TrSize** is the average size of training data on all runs. **TSize** is the average size of test data on all runs. **ABR** is the average buggy rate. **NR** is the number of runs for each subject.

Project	Lang	LOC	First Date	Last Date	Changes	TrSize	TSize	ABR (%)	# NR
PostgreSQL	C	289K	1996-07-09	2011-01-25	89K	1,232	6,824	27.4	7
Xorg	C	1.1M	1999-11-19	2012-06-28	46K	1,756	6,710	14.7	6
JDT	Java	1.5M	2001-06-05	2012-07-24	73K	1,367	6,974	20.5	6
Lucene	Java	828K	2010-03-17	2013-01-16	76K	1,194	9,333	23.6	8
Jackrabbit	Java	589K	2004-09-13	2013-01-14	61K	1,118	8,887	37.4	10

This dataset is collected from five open-source projects, i.e., PostgreSQL, Xorg, Jdt (from Eclipse), Lucene, and Jackrabbit. They are large and typical open source projects covering operating system, database management system. These projects have enough change history to build and evaluate change-level defect prediction models. Table **??** shows the evaluated projects for change-level defect prediction. Each project have more multiple runs, each run contains a training dataset and a test dataset, which are collected during different time periods. For example, PostgreSQL has 7 runs, you can build and train you model on each of the 7 runs, and use the average performance to evaluate your prediction models. The LOC and the number of changes in Table **??** include only source code (C and Java) files[1] and their changes.

Each folder of each project contains both the training and test datasets. Each change has an unique change id and the label (buggy or clean). We have already collected the source code for each change, which is stored in **patch.zip**.

How to Collect Features

Bag-of-Words: Take the first change in the training data of folder 0 from Lucene, i.e., **9007** as an example. The corresponding source code of this change is **lucene-9007.patch** in **/proj-skeleton/data/patch.zip**.

```
diff --git a/solr/src/java/org/apache/solr/handler/component/QueryComponent.java b/solr/src/java/org/
    apache/solr/handler/component/QueryComponent.java
index 0415f29..96f3893 100644
--- a/solr/src/java/org/apache/solr/handler/component/QueryComponent.java
+++ b/solr/src/java/org/apache/solr/handler/component/QueryComponent.java
@@ -217,14 +217,8 @@ public class QueryComponent extends SearchComponent
```

[1] We include files with these extensions: .java, .c, .cpp, .cc, .cp, .cxx, .c++, .h, .hpp, .hh, .hp, .hxx and .h++.

```
            for (String groupByStr : funcs) {
              QParser parser = QParser.getParser(groupByStr, ”func”, rb.req);
              Query q = parser.getQuery();
-             SolrIndexSearcher.GroupCommandFunc gc;
-             if (groupSort != null) {
-               SolrIndexSearcher.GroupSortCommand gcSort = new SolrIndexSearcher.GroupSortCommand();
-               gcSort.sort = groupSort;
-               gc = gcSort;
-             } else {
-               gc =  new SolrIndexSearcher.GroupCommandFunc();
-             }
+             SolrIndexSearcher.GroupCommandFunc gc = new SolrIndexSearcher.GroupCommandFunc();
+             gc.groupSort = groupSort;

              if (q instanceof FunctionQuery) {
                gc.groupBy = ((FunctionQuery)q).getValueSource();
```

You can collect the **Bag-of-Words** features for each change by using the code of each change we provided. Specifically, you need first to tokenize each change by filtering unnecessary symbols, then collect all the frequent tokens (e.g., the frequency is larger than 3) from all the changes from a folder (both the training and test datasets). Finally, building the bag-of-words model for changes.

Useful information about the **Bag-of-Words** model can be found here:

`https://en.wikipedia.org/wiki/Bag-of-words_model`

Code Complexity: To collect the code complexity of a change, you need the file versions before and after a change. You can find the the commit id of each change in folder **/proj-skeleton/data**. For example, **lucene_commit_change.csv** contains the commit id of each change id from project lucene. Taking the **9007** as an example, you could obtain the commit that made this change, i.e., **9535bb795f6d1ec4c475a5d35532f3c7951101da**.

```
 9007,9535bb795f6d1ec4c475a5d35532f3c7951101da, solr/.../QueryComponent.java
```

With the commit id, you could obtain the file versions before and after a change (e.g., **9007**). After that, you could use Understand (available on ecelinux machines) to collect these code metrics.

Characteristic Vector: The process of collecting this type of features is similar to that of **Code Complexity**. The only difference is that after obtaining the file versions before and after a change (e.g., **9007**), you need to apply Deckard.