Android bugs processed by Weka

- We created csv files
- We ran unsuccessfully command

  ```
  $ java weka.core.converters.CSVLoader \
      ANDROID_PLATFORM_BUGS.csv > ANDROID_PLATFORM_BUGS.arff
  ```

- so we design our own script
- We tried tried to convert strings attributes to vectors for 1000 most important words. Unfortunately, following command crashes.

  ```
  $ java weka.filters.unsupervised.attribute.StringToWordVector \
      -i android-platform-bugs.arff -o android_plat-bugs-Vectors.arff
  ```

- We found out that the script works if there only the string attributes so we generated *android-plat-strings.arff* and *android-plat-except-strings.arff*
- We split them manually on the data part and the header, which describes types of attributes. The headers we simply concatenated. We applied following commands on the data.

  ```
  nl -s " " -b a android-plat-except-strings.data # number lines
  nl -s " " -b a android-plat-strings.data # number lines
  sed s:\(......\) :\1 a : -f *except* > number-and-prefix-a
  sed s:\(......\) :\1 b : -f *strings* > number-and-prefix-b
  cat *strings* *except*> mixed.data
  sort -n data > sorted.data
  sed s:\n...... b :: -f sorted.data> joined-related-lines
  sed s:...... a :: -f joined-related-lines > without-prefix
  cat mixed-head without-prefix > android-plat-bugs2vectors.arff
  ```

- Next day we realised, that weka crashed because the name of attributes *status*, *type* or *id* were also generated by **StringToWordVector** command, so we had had multiple attributes with the same names. So instead of splitting in previous steps we just rename our attributes.
- Finally, we could run our example. We experienced that Weka requires much less memory if is launch from command line, so we had to find out parameters for command line first.
  - We launch Weka
  - Switched to Explorer
  - Set up *Component* as label attribute for training and testing
  - Choose 10 fold cross validation
  - Choose Support classifier e.g. Support Vector Machines (SMO in Weka)
  - Copy parameters generated by GUI to command line
- At the end we can ran a Weka command for training, testing and validating one of data mining algorithms

Weka data mining algorithms

In our first experiment we tried to use Support Vector Machines(SMO) and Decision Trees to guess *Component* attribute. *Component* attribute is a nominal attribute and define in which part of Android the bug was discovered. Possible values are *Browser*, *Dalvik*, *Framewok*, etc.

As we mentioned before, we generated the command via Weka *Explorer* GUI application and for validating we have chosen 10 fold cross validation, because it's standard validation method.

```
$ java weka.classifiers.functions.SMO \
 -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K \
 "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0" \
 -t sample-vectorised.arff -c 7> smo.out
```

We got precision of 79% on sample containing 4000 items with following accuracy and confusion matrix. Actually, as you can see in the confusion matrix the results seems quite good because most of the misclassified samples are from category null. In the second table we compared the accuracy of the same algorithm with and without null values.

Confusion matrix

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	classified as
45	9	1	1	1	0	1	0	0	0	0	0	0	0	0	0	a = Browser
5	2138	34	52	19	0	9	10	9	15	3	0	1	0	0	0	b = null
0	65	17	12	1	0	2	6	1	4	1	0	0	0	0	0	c = Framework
0	75	6	38	4	0	0	6	2	4	0	0	0	0	0	0	d = Applications
0	47	1	2	54	0	0	0	0	1	0	0	0	0	0	0	e = Tools
0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	f = User
1	18	1	0	1	0	32	0	0	0	0	0	1	0	0	0	g = Dalvik
1	38	4	5	0	0	0	18	0	0	0	0	0	0	0	0	h = Google
0	35	5	3	0	0	0	0	2	0	0	0	0	0	0	0	i = GfxMedia
0	33	3	10	1	0	0	2	0	15	0	0	0	0	0	0	j = Device
0	15	2	0	0	0	1	1	1	3	2	0	0	0	0	0	k = System
0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	l = Build
0	14	2	0	2	0	0	0	0	1	0	0	4	0	0	0	m = Web
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	n = Market
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o = Android
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	p = Docs

We reran the experiment without null values for target *Component* attribute. See difference below.

Algorithms results

Type	Algorithm	Sample	Time	Accuracy
Decision Trees	Weka J48	4000 with nulls	3:36:03	77,4%
Decision Trees	Weka J48	4000 without nulls	3:14:25	62,2%
Support Vector Machines	Weka SMO	4000 with nulls	43:03	79%
Support Vector Machines	Weka SMO	4000 without nulls	44:56	64.8%

Classification of bugs by developer

In Another experiment we tried to deduce which owner should care about the bug. At first we ran the experiment with null values and got following results of Stratified 10 folds cross-validation. In this experiment we used only *SMO*, because from our prior knowledge Support Vector Machines should perform better than Decision trees and we have at this point confirmed it using both Weka and Rapid Miner.

Experiment results

Type	Algorithm	Sample	Time	Accuracy
Support Vector Machines	Weka SMO	4000 with nulls	58:23	77.9%
Support Vector Machines	Weka SMO	4000 without nulls	1:03:41	55.1%