Reproducibility package

This package contains code to repeat the research done for

On the Evaluation of Unsupervised Outlier Detection: Measures, Datasets, and an Empirical Study by G. O. Campos, A. Zimek, J. Sander, R. J. G. B. Campello, B. Micenková, E. Schubert, I. Assent and M. E. Houle to appear in Data Mining and Knowledge Discovery

Supplementary Material Homepage

Prerequisites

1. Lots of CPU power

   On a 24 cores Intel Xeon X5650 @ 2.67 GHz with 128 GB RAM, rerunning all the experiments took:

   ```
   "make primary": Wall-clock time: 22:05:29
   "make eval":    Wall-clock time: 00:15:33
   ```

   Generating all the plots (on a single core) needs additional time. Depending on your CPU, it may be inbetween of 400 to 600 CPU hours.

2. Disk space

   You will need about 50 GB of disk space. More, if you add additional data or methods.

3. Software prerequisites

   - Java 8, for example OpenJDK 8, in a recent version
   - Python, with numpy and scipy for evaluation
   - R, for visualization, with the following packages (only on one host) `install.packages(c("dplyr", "ggplot2", "RColorBrewer", "fields", "car"))`
   - GNU Make, for running the Makefile
   - Linux. Most probably other operating systems will not work. We used Debian Linux and Ubuntu Linux.

4. Data

   From the project page, download the `literature` and `semantic` archives, and extract them into the folders `input/literature` and `input/semantic`.

   You can add additional files in the `input`, but you may need to adjust the scripts. Currently, only uncompressed `.arff` files will automatically be processed, and they must only have numerical columns, except for an optional `id` and a mandatory `outlier` column.

5. ELKI

   The ELKI version we used for our experiments is included in this package (a 0.7.0 pre-release, without the visualization add-on).

   You can however download a later version of ELKI

   Results with other ELKI versions may change, and you may need to adjust the scripts appropriately!

Repeating everything

The first (and most expensive) step is to repeat all outlier detectors.

The scripts *try* to use locking to allow parallel computation on multiple machines, if you have shared storage (e.g. NFS) with reliable file locking and low latency, it *may* be possible to use multiple hosts to build the results. You can then invoke the `make primary` command on each node.

```
make primary
```

This command will start computing all outlier detectors, using all available CPU cores. If you only want to use a single core, you can instead use this:

```
make SOFTFAIL=1 all-primary
```

When all primary results have been built, you can run the analysis procedures:

```
make eval
```

Again, this *should* be able to run in parallel, using multiple hosts and CPUs.

Finally, generate the figures on a single host using

```
make plot
```

This command should *not* be used in parallel on multiple machines. The runtime on a single core should be less than one hour. It will probably work if you parallelize on a single host via `make -j4 plot` to use 4 cores.

If you only have a single host (and enough time), you should be able to build everything by calling

```
make primary eval plot
```

Adding new methods

To add a new method to the analysis, we suggest that you add the method to ELKI first. There is a Tutorial on the ELKI Wiki on implementing a new outlier detection method. We use the class `ComputeKNNOutlierScores` to compute the results, so if you add your method to this class, build a new ELKI package with `mvn package`, modify the `Makefile` to use your new ELKI package, and rerun everything, your new method should appear in the results. For the plots, add it to `scripts/shared.R`, too.

Implementation notes

With `SOFTFAIL=1`, rules will *not* fail if the file is currently locked by a different host. This is only acceptable *if* it is not a hard prerequisite for a target. Thus, do *not* use this for the later stages of the build.

Recovering from failure. If some computation fails (for example due to parallelization clashes or other software failures), you may need to recover by manually removing lock files (ending in `.lock`, located either in the `tmp` folder, or next to the output file) or incomplete result files (for example `.raw`). Finished result files will be compressed into `.gz` files.

Except for the input data, scripts, and the ELKI binary, you can remove and regenerate results if in doubt.