OLED Manual

November 2, 2018

1 Install

- Clone of download the source code:
  `git clone https://github.com/nkatzz/OLED`
- `cd OLED/install-scripts`
- `./install.sh`
  You will need sudo priviledges to run the install script. The whole process will take some time to complete.
- Update your `LD_LIBRARY_PATH`, `PATH` and `PYTHONPATH` variables as shown in Figure 1. To do so, add the lines presented in Figure at the end of your `~/.profile` (or your `~/.bashrc`) file. Don't forget to modify paths shown in 1 to match your own path to `OLED` and to log-off/log-on again for the updates to the variables to take effect.

2 Run

We'll next do a test run to make sure that everything is ok. We'll use the CAVIAR[1] dataset for human activity recognition. The CAVIAR dataset consists of a number of videos of actors performing a set of activities. Manual annotation (performed by the CAVIAR team) provides ground truth for two activity types. The first type corresponds to simple events and consists of the activities of a person at a

[1] http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/

```
LD_LIBRARY_PATH=/home/nkatz/OLED/external_dependencies/lpsolve55:$LD_LIBRARY_PATH

export LD_LIBRARY_PATH

export PATH=$PATH:/home/nkatz/OLED/external_dependencies/LoMRF/target/universal/
LoMRF-0.5.5-SNAPSHOT/bin:/home/nkatz/OLED/external_dependencies/clingo/clingo-4.5.4-source/build
/release

export PYTHONPATH=$PYTHONPATH:/home/nkatz/OLED/external_dependencies/clingo/clingo-4.5.4-source/
build/release/python
```

Figure 1: Set paths properly at `/Home/.bashrc` or `/Home/.profile`.

certain video frame/time point, such as *walking*, or *standing still*. The second activity type corresponds to complex events and consists of activities that involve more than one person, e.g. two people *meeting each other*, or *moving together*. The goal is to recognize complex events as combinations of simple events and additional contextual knowledge, such as a person's direction and position.

2.1 Download and Prepare the Data.

First, install MongoDB to your computer. We'll use mongo to store the data and consume it in a simulation of a streaming setting. Mongo is very easy to install, just google for specific instructions for your particular Linux distribution.

```
# From within /oledhome execute the following commands in a terminal:
# Download the data:
wget http://users.iit.demokritos.gr/~nkatz/oled/caviar-data.zip

# Download the background knowledge for the particular learning task:
wget http://users.iit.demokritos.gr/~nkatz/oled/caviar-bk.zip

# Unzip
unzip caviar-data.zip
unzip caviar-bk.zip
cd caviar-data

# Import training set into Mongo:
mongoimport --db caviar-train --collection examples --file caviar-train.json

# Import testing set into Mongo:
mongoimport --db caviar-test --collection examples --file caviar-test.json

# Clean-up (you may want to inspect the data first):
rm -r caviar-data
rm caviar-data.zip
rm caviar-bk.zip

# Make sure everything is ok (you should see the newly-created dbs 'caviar-train' and 'caviar test'):
mongo
show dbs
```

Figure 2: Download CAVIAR data and import it into Mongo.

Next, download the the data and import it into Mongo. Open a terminal and execute the commands shown in Figure 2. You'll download two datasets, a fragment of CAVIAR that will be used for training (`caviar-train`) and another that will be used for testing (`caviar-test`). Subsequently, you'll import each dataset into mongo, clean-up (remove the downloaded data) and make sure that the new dbs were created as required.

Before cleaning up, you may wish to open the downloaded json files and inspect the data. You'll see that each entry contains three fileds: `'annotation'`, `'narrative'` and `'time'`. The `'annotation'` filed contains instances of the target complex events we wish to learn definitions for. In this example, we're learning a definition for two persons *meeting each other*, therfore, `'annotation'` consists of instances of the *meeting* complex event. The `'narrative'` field consists of instances of simple events (*walking*, *active*, *inactive* and so on) and spatiotemporal knowledge, such as persons' coordinates and direction. Each entry is a "batch" of data, i.e. it consists of annotation and narrative found within a temporal interval/window. The entry's `'time'` field is the first time point observed in the batch and it serves as a key to refer to that particular batch/entry.

Next, take a look at the background knowledge that you downloaded (in `/caviar-bk`). It consists of three files, `ec`, `bk` and `modes`. `ec` contains the axioms of the Event Calculus. This file is optional in the general case, it is only necessary if you want to learn using the Event Calculus in the backgound knowledge (but it is necessary for the particular test OLED run that we are about to execute). `bk` is the file where the user defines some domain-specific knowledge. For the purposes of this example, `bk` contains code for calculating Euclidean distances between persons from their coordinates. Such distances are produced at learning time. All content of the `ec` and the `bk` is Answer Set Programming (ASP) code that `Clingo`, OLED's main reasoning component, is able to execute. Please consult the guide provided by the `Clingo` team[2]. The user may use any ASP code in the `bk` file to define tasks that will be executed by `Clingo` at learning time, e.g. generating extra, non-input knowledge for the input data, or performing numerical computations via embedded Lua or Python scripts.

The `modes` file contains some language bias, i.e. restrictions to the syntax of predicates that may be used to form rules. OLED uses *mode declarations* as a language bias (see the paper that comes with the source code, or any ILP-introductory textbook for details). in addition to the lanhuage bias, the `modes` file contains

[2]Available from `https://potassco.org/`, at the documentation site

a declaration of annotation predicate signatures (via the `examplePattern/1` predicate) and input predicate signatures (via `inputPredicate/1`). The latter is the signature specification of all predicates used to pass data to the input and it is used to extract entity types during the learning process. The modes file is necessary for all learning tasks.

We are now ready to perform learning with OLED. cd to your `/oledhome` folder and run the following in a terminal:

```
java -cp oled-0.1.jar app.runners.OLEDDefaultRunner \
--inpath=<PATH TO caviar-bk FOLDER> --delta=0.00001 \
--prune=0.8 --target=meeting --train=caviar-train \
--saveto=<PATH TO SOME LOCATION>/theory.lp
```

To see all available command line arguments do `java -cp oled-0.1.jar -help`. The learnt hypothesis will be saved in `/oledhome/theory.lp`. You may evaluate this theory on the test set as follows:

```
java -cp oled-0.1.jar app.runners.OLEDDefaultRunner \
--inpath=<PATH TO caviar-bk FOLDER> --target=meeting \
--test=caviar-test --evalth=<PATH TO theory.lp>
```