How to discover functional dependencies

Panagiotis Mandros

This pdf contains instructions on how to discover functional dependencies from mixed data. FoDiscovery.jar is the executable with all dependencies attached.

1 Valid data formats

- arff
- xarf (see https://bitbucket.org/realKD/realkd/wiki/model/data/xarf)
- csv
  - with or without header
  - it automatically infers attribute type (for full control, it is advised to declare attribute types with arff or xarf)

2 Shell arguments for the two discovery algorithms

Only the obligatory commands are required for the code to run. The remaining arguments are set to their default values.

2.1 Obligatory for both algorithms

- -DATASET
  - filepath for dataset (e.g., user/data/dataset.arff)
- -OUTPUTFOLDER
  - folder to write output file

2.2 Optional for both algorithms

- -TARGET
  - the index of the target variable (starting from 1)
  - default is the last column
- -K
  - number of results to return (e.g., top-10)
  - default is 1
- -OPT
  - which bounding function to use for pruning
  - options are

    - MON for $\bar{f}_{\text{mon}}$
    - SPC for $\bar{f}_{\text{spc}}$
    - CHAIN to prune with both (first try with $\bar{f}_{\text{mon}}$, and if it fails, try again with $\bar{f}_{\text{spc}}$)
  - default is CHAIN

- -DISCTYPE
  - which discretization technique to use for discretizing ordinal attributes
  - options are
    - COP for discretizing variables with constrained optimal partition, based on reliable mutual information (supervised)
    - EF for discretizing variables with equal-frequency, based on reliable mutual information (supervised)
    - PRE pre-discretize data independently of each other with equal-frequency (unsupervised)
  - default is COP

- -L
  - maximum number of bins for discretization (PRE will pre-discretize in exactly that many equal-frequency bins)
  - default is 5

- -C
  - parameter to multiply the maximum number of bins for initial equal-frequency bins (COP only)
  - default is 2

- -NUM_BINS_TARGET
  - bins for equal-frequency discretization of the target if it is ordinal
  - default is 10

2.3 For greedy algorithm only

- -BEAMWIDTH
  - the size of the beam, i.e., how many of the top candidates to expand in each level
  - default is 5

2.4 For branch-and-bound algorithm only

- -ALPHA
  - alpha-approximation to use
  - default is 1, which yields the optimal solution

3 Executing the code

The code creates an output file in the -OUTPUTFOLDER folder, with the dataset name and parameters used. The file contains various statistics of the search, and the top-k functional dependencies discovered. Every dependency contains the entropy of the target, the mutual information, the expected mutual information under the null, the reliable fraction of information score, and the uncorrected reliable fraction of information score.

4 Examples of executing the code

Consider the dataset australian.arff (from KEEL data repository) located in this folder.

4.1 Single dataset

Branch-and-bound The following command executes the branch-and-bound algorithm for top-1 solution, with constrained optimal partition, maximum number of bins $l = 10$, $c = 3$, and $\alpha = 1$. (copy pasting will result in a line break)

```
nohup java -cp FoDiscovery.jar de.mpiinf.fodiscovery.singleexp.FoOPUS -DATASET australian.arff
-OUTPUTFOLDER exampleOutput/ -ALPHA 1 -K 1 -L 10 -C 3 -DISCTYPE COP &
```

Greedy The following command executes the greedy algorithm for top-1 solution, with supervised equal-frequency, maximum number of bins $l = 10$, and beam width 10.

```
ohup java -cp FoDiscovery.jar de.mpiinf.fodiscovery.singleexp.FoBeam -DATASET australian.arff -
OUTPUTFOLDER exampleOutput/ -BEAMWIDTH 10 -K 1 -L 10 -DISCTYPE EF &
```

4.2 Multiple datasets

The implementation offers the option to run multiple experiments using the classes FoOPUSs and FoBeams that can run multiple OPUS and Beam experiments respectively. It requires two obligatory arguments:

- -INPUT
  - a file that contains one single experiment per line, using the same arguments as above
- -OUTPUTFOLDER
  - output folder to write the output files (one for every experiment in the input file))

Example Let us assume we want to run two times the branch-and-bound algorithm to discover dependencies with different approximation guarantees, e.g., $\alpha = 0.8, \alpha = 0.5$, and supervised equal-frequency in maximum $l = 10$ bins. We create an input file named australianDifferentAlphasExperiment.txt and insert the two lines

```
-DATASET australian.arff -ALPHA 0.5 -DISCTYPE EF -L 10
-DATASET australian.arff -ALPHA 0.8 -DISCTYPE EF -L 10
```

We run the experiment with the following command:

```
nohup java -cp FoDiscovery.jar fodiscovery.multipleexp.FoOPUSs -INPUT australianDifferentAlpha-
sExperiment.txt -OUTPUTFOLDER exampleOutput/ &
```

which creates two files in the output folder, one for each run.