Knowledge Acquisition for Next Generation Statement Map

Author: Eric Nichols, eric@ecei.tohoku.ac.jp

instances2matrix.py

`instances2matrix.py`: creates a matrix of co-occurence counts between relation pattern * arguments in mongodb from input instances

Usage

```
Usage: instances2matrix.py [options] [<instance_files>]

Options:
-h, --help            show this help message and exit
-c COLLECTION, --collection=COLLECTION
                      collection name
-d DB, --database=DB  database name
-o HOST, --host=HOST  mongodb host machine name.
                      default: localhost
-p PORT, --port=PORT  mongodb host machine port number.
                      default: 27017
```

Instances

Format

Instances have the following tab-delimited format:

- `score`: score representing weight * co-occurence count for instance
- `loc`: giving source and location of instance
- `rel`: containing relation pattern
- `argc`: giving argument count

- `argv`: tab-delimited list of arguments as strings

Example

```
1.0\treverb_clueweb_tuples-1.1.txt:30:10-11\tARG1 acquired ARG2\t2\Google\tYouTube
```

Co-occurence Matrix

Format

The co-occurence matrix collection has the following fields:

- `rel`: relation pattern
- `arg1`: first argument
- ...
- `argn`: nth argument
- `score`: score for rel * args tuple

Naming Scheme

Instances of differing argument count are stored in separate mongodb collections with names formatted as `<collection>_<argc>`. E.g. if a collection `clueweb` has instances with argument counts of 1, 2, and 3, then the following collection would be created:

- `clueweb_1`
- `clueweb_2`
- `clueweb_3`

Indexing

It is indexed for fast look up of rel, args, and (rel,args) tuples.

matrix2pmi.py

`matrix2pmi.py`: caches co-occurence frequencies and discounted PMI between relation patterns and argument tuples into a matrix stored in mongodb

Usage

```
Usage: matrix2pmi.py [options] [database] [collection]

Options:
-h, --help            show this help message and exit
-o HOST, --host=HOST  mongodb host machine name.
                      default: localhost
-p PORT, --port=PORT  mongodb host machine port number.
                      default: 1979
-s START, --start=START
                      specify calculation to start with
                      1 or F_i: instance tuple frequencies
                      2 or F_p: relation pattern
frequencies
                      3 or F_ip: instance*pattern co-
occurence frequencies
                      4 or pmi_ip: instance*pattern
discounted PMI score
                      default: F_i
```

Caches Created

Creates 4 frequency/score caches in the form of mongodb collections:

1. `<matrix>_F_i`: instance tuple frequencies

2. `<matrix>_F_p`: relation pattern frequencies

3. `<matrix>_F_ip`: instance*pattern co-occurence frequencies

4. `<matrix>_pmi_ip`: instance*pattern Pointwise Mutual Information score discounted to account for bias toward infrequent events following [1]

Pointwise Mutual Information

Pointwise mutual information between argument instances and relation patterns is defines following [2] as:

```
(1) PMI(i,p) = log( F(i,p) / F(i)*F(p) )
```

where

```
(2) F(i) = the frequency of argument instance i
(3) F(p) = the frequency of relation pattern p
(4) F(i,p) = the co-occurence frequency of argument
instance i and relation pattern p
```

Discounted PMI

Pointwise Mutual Information is known to be biased toward infrequent events. Pantel and Ravichandran [1] compensate by multiplying PMI by a "discounting factor" that is essentially a smoothed co-occurence frequency multiplied by a smoothed frequency of the argument instance or the relation patter, whichever is lesser.

```
(5) discount(i,p) = (F(i,p) / F(i,p)+1) * (min(F(i),F(p))
/ min(F(i),F(p))+1)
(6) discountedPMI(i,p) = PMI(i,p) * discount(i,p)
```

espresso.py

`espresso.py`: an implemenatation of the Espresso bootstrapping algorithm

Usage

```
    Usage: espresso.py [options] [database] [collection]
[rel] [seeds]

    Options:
      -h, --help            show this help message and
exit
```

```
      -o HOST, --host=HOST  mongodb host machine name.
default: localhost
      -p PORT, --port=PORT  mongodb host machine port
number. default: 27017
      -s START, --start=START
                            iteration to start with.
default: 1
      -t STOP, --stop=STOP  iteration to stop at. default:
2
```

Caches Created

Creates 2 caches of bootstrapped instances and patterns for the target relation:

1. `<matrix>_<rel>_esp_i`: bootstrapped instances for
2. `<matrix>_<rel>_esp_p`: bootstrapped patterns for

Bootstrapping

Bootstrapping starts with seed instances and alternates between promoting new patterns and instances following the Espresso bootstrapping algorithm [2].

1. retrieve promoted instances/patterns
2. rank by reliability score
3. keep top 10 promoted instances/patterns
4. bootstrap patterns/instances using promoted instances/patterns

Reliability Score

Candidate patterns and instances are ranked by reliability score, which reflects the pointwise mutual information score between a promoted pattern/instance and the set of instances/patterns that generated it.

```
     (1) r_i(i,P) = sum( dpmi(i,p)*r_p(p) / max_pmi ) /
```

```
len(P)
                     for p in P

    (2) r_p(P,i) = sum( dpmi(i,p)*r_i(i) / max_pmi ) /
len(I)
                     for i in I
```

where dpmi is Discounted Pointwise Mutual Information [1]. r_i and r_p are recursively defined with r_i=1.0 for the seed instances.

References

[1] Patrick Pantel and Deepak Ravichandran. Automatically Labeling Semantic Classes. HLT-NAACL 2004.

[2] Patrick Pantel and Marco Pennacchiotti. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. ACL 2006.