Manual for DependencyVectors 2.0

Sebastian Pado
Computational Linguistics
Saarland University
pado@coli.uni-sb.de

September 2, 2005

This is a semi-technical document on how to use the DEPENDENCYVECTORS software package to produce dependency-based vector spaces. The concept of dependency spaces and ideas as to their applications can be found in [1] and [2]. The installation is described in the `README` file; implementation details can be found in the `javadoc` documentation (see `README`).

Contents

1 Licence

2 Using DEPENDENCYVECTORS

2.1 Architecture

The DEPENDENCYVECTORS software package contains three programs, corresponding to three steps of constructing a dependency-based semantic space model.

1. `ExtractBasisElements`: Choice of basis elements. Different from traditional word-based semantic space models, which usually use the most frequent context words, DEPENDENCYVECTORS models can use any set of basis elements (BEs). These have to be extracted (together with their frequencies) from the corpus in the first step. Additionally, `ExtractBasisElements` extracts frequencies for the target words and the total number of paths which are necessary for step 3.

2. `ExtractSpace`: Construction of semantic space. This constructs an actual DEPENDENCYVECTORS semantic space model from a corpus, given targets and basis elements.

3. `LLT`: Log-likelihood transformation. This performs a log likelihood transformation on a semantic space (optional).

It is recommended to call the three programs through the shell scripts in `bin/`. They communicate through files; the flow of information can be seen in Figure 1. All files in the middle row are stored in the directory specified by the option `targetdir` and are named according to a naming convention (see Table 1) that reflects the fact that the two main parameters of the extraction are the context specification and path value functions. Currently, change of the basis mapping function is only possible by recompiling; the default is the . All programs support the concurrent processing of files with diffent parameters (see below).

2.2 Using `ExtractBasisElements`

This program extracts basis elements. Since there may be (too) many, if a lexical basis mapping (like the default) is used, `ExtractBasisElements` supports *purging*, i.e. the regular

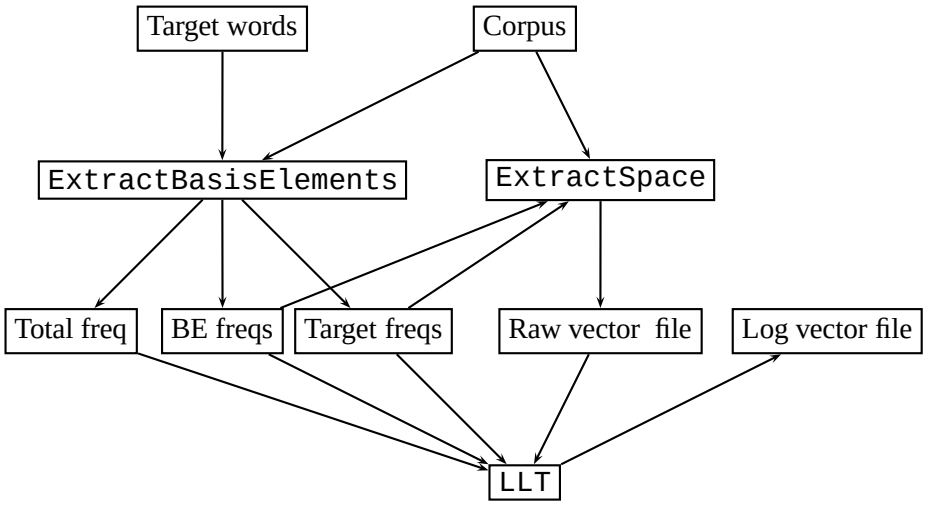

Figure 1: Flow of Information in DEPENDENCYVECTORS

File	Naming convenion
Target frequencies	`cs_pv.targets`
BE frequencies	`cs_pv.bes`
Total frequency	`cs_pv.total`
Raw vectors	`cs_pv.vectors`
Log-transformed vectors	`cs_pv.vectors`

Table 1: Naming convention for files in `targetdir` (`cs` is name of context specification, `pv` name of path value function).

removal of the most infrequent basis elements. This is implemented as follows: if the list contains more than `maxsize/ratio` elements, the list is shortened to `maxsize` elements. This guarantees that at every point, `maxsize` *reliable* basis elements are available.

–corpus <name> : Location of corpus (default: System.in)

–cutoff <int> : Ratio of basis elements to be deleted in each purge (default: 0.3)

–help : Display help and exit.

–log <file> : Destination for the log file (default: log.txt)

–maxsize <int> : Desired number of basis elements (default: 10000)

–spec <file> : Name of file with context specification (default: all). The standard DEPENDENCYVECTORS distribution comes with four specs:
`contextspec_{minimal,medium,rich,wide}.txt`

–targets <file> : File containing target words

–targetdir <file> : Data directory, used for writing output files (see Figure 1).

–plain | –length | –oblique | –oblength : Path value functions provided by the standard DEPENDENCYVECTORS distribution (default: all)

If either `-spec` or the path value function is omitted, the extraction is run for all available values of that parameter, resulting in different file sets in the target directory. Example call for `ExtractBasisElements`:

```
zcat bnc.parsetrees.gz | bin/ExtractBasisElements.sh
 --targets targetsfile --targetdir data/bnc/
 --plain --spec contextspec_minimal.txt
```

Note that after `ExtractBasisElements` is a good opportunity to sort out unwanted basis elements according to frequency or any other criterion; see the definition of the frequency file format below.

2.3 Using `ExtractSpace`

This program extracts semantic spaces, given sets of targets and basis elements. It supports incremental space building (to obtain learning curves) by using the `-every` option. The central data structure is implemented in two different ways. The default implementation is faster, but uses much memory. For the concurrent extraction of multiple spaces, I recommend using the `-small` option which may be slower, but saves memory[1].

–corpus <name> : Location of corpus (default: System.in)

[1] Note that the maximum size of processes on 32-bit machines is limited to about 1.8 GB

–every <num> : Incrementally write semantic space every <num> corpus bytes to file `targetdir/cs_pv.vectors.<index>`

–help : Display help and exit.

–log <file> : Destination for the log file (default: log.txt)

–small : Uses Semantic Space class with small memory footprint (recommended for concurrent extracting of multiple spaces, but possibly slower)

–spec <file> : Name of file with context specification (default: all). The standard DEPENDENCYVECTORS distribution comes with four specs: `contextspec_{minimal,medium,rich,wide}.txt`

–targetdir <file> : Data directory, used for reading input files and writing output files (see Figure 1).

–plain | –length | –oblique | –oblength : Path value functions provided by the standard DEPENDENCYVECTORS distribution (default: all)

–nofrequencies : Do not complain about missing frequencies in basis element and target files (see below).

If either `-spec` or the path value function is omitted, the extraction is run for all available values of that parameter, resulting in different file sets in the target directory. Example call for `ExtractSpace`:

```
zcat bnc.parsetrees.gz | bin/ExtractSpace.sh
 --targetdir data/bnc/ --every 10000000
 --plain --spec contextspec_minimal.txt
```

Having individual target and basis element files for different parametrisations may seem unnecessary. However, recall that these files also contain frequencies (computed, e.g., by `ExtractBasisElements`). These typically vary between parameterisations and are necessary for computing the log likelihood. ExtractSpaces itself can run without this frequency information, but this is disabled by default as a security measure. Specify `-nofrequencies` if you want to enable this mode.

2.4 Using LLT

This program performs a log-likelihood transformation on a semantic spaces. In keeping with the arguments of the other programs, the name of the vector file is also specified by the context spec and path value functions, which again allows for concurrent processing.

–help : Display help and exit.

–log <file> : Destination for the log file (default: log.txt)

–spec <file> : Name of file with context specification (default: all). The standard DEPENDENCYVECTORS distribution comes with four specs:
`contextspec_{minimal,medium,rich,wide}.txt`

–targetdir <file> : Data directory, used for reading input files and writing output files (see Figure 1).

–plain | –length | –oblique | –oblength : Path value functions provided by the standard DEPENDENCYVECTORS distribution (default: all)

Example call for `LLT` to convert spaces for all context specifications:

```
bin/LLT.sh --targetdir data/bnc/ --plain
```

3 File formats

This section lists the formats of the various files.

3.1 Targets file

One word (target) per line, everything else ignored. No comments allowed.

3.2 Target/BE frequency file

Every line contains two tab-separated tokens: (1), the target or basis element, and (2), its frequency. No comments allowed.

3.3 Total path frequency file

Just one line, containing the string representation of a Double.

3.4 Vectors file

The first line contains a tab-separated list of all basis elements. All following lines contain first the target that is represented by that line, then a colon, and then the vector as a space-separated list of Doubles.

3.5 Context specification file

The context specifications are stored in external files. Lines that begin with # are treated as comments and ignored, as are empty lines. Lines that specify path templates contain one or more edge templates, which are separated by `=`. Each edge template is a five-tuple, separated by colons (`:`). The first and second tokens specify the lemma and part of speech of the source node. The third token is the dependency relation label of the edge. The fourth and fifth tokens are the part of speech and lemma of the target node. For every of these tokens, users can specify the asterisk (`*`), which will match everything. For examples, see the provided context specification files.

4 Modifications and extensions

If you implement any modifications or extensions to the DEPENDENCYVECTORS package, I would be very interested in hearing from you and merging the new code into the main development branch. If you have any trouble in understanding what's going on, let me know.

4.1 Using another parser

This should (only) require extending or replacing the `Corpus` class, which can at the moment only parse MINIPAR-analysed corpora.

4.2 Using a different basis mapping function

This requires rewriting (only) the `Parameters.basisMapping` method.

4.3 Adding new context specifications

This requires putting the new context specification files (format see above) into the directory `lib/contextspecs` and adding the name to the `Parameters.allCSFiles` array.

4.4 Adding new path value functions

This requires extending the `Parameters.pathValue` method.

5 Version history

- Version 1.0 (September 2002): First version.
- Version 1.1 (July 2003)
  - New feature: `LLT` for log-likelihood transformation.
  - New feature: Support for MINIPAR "full parses" format.
  - Doc: First PS documentation.
- Version 2.0 (November 2004)
  - Doc: Thorough documentation of code.
  - Doc: Revised PS documentation.
  - Implementation: Complete refactorisation of classes.
  - Implementation: All programs use `targetdir` now.
  - Implementation: Introduction of packaging scheme and `Makefile`.

References

[1] S. Pado and M. Lapata: Constructing Semantic Space Models from Parsed Corpora. Proceedings of ACL-03, Sapporo.

[2] S. Pado and M. Lapata: Dependency-based semantic space models. Submitted.