Manual for DependencyVectors 2.4a

Sebastian Padó
Institut für Maschinelle Sprachverarbeitung
Stuttgart University
pado@ims.uni-stuttgart.de

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 By way of introduction: The `sample` directory

The directory `sample` contains a "minimal" running example for creating a semantic space: a two-sentence corpus in "full" format (`sample_corpus_full`) and triple format (`sample_corpus_triples`) as well as a set of two target words (targets). Executing either of the scripts `create_space.sh` or `create_space_triples.sh` will run the complete pipeline and create a space in the subdirectory vectors. These scripts can serve as templates for own calls of the DEPENDENCYVECTORS pipeline.

2.2 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 (boxes) 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. All programs support the concurrent processing of files with diffent parameters (see below).

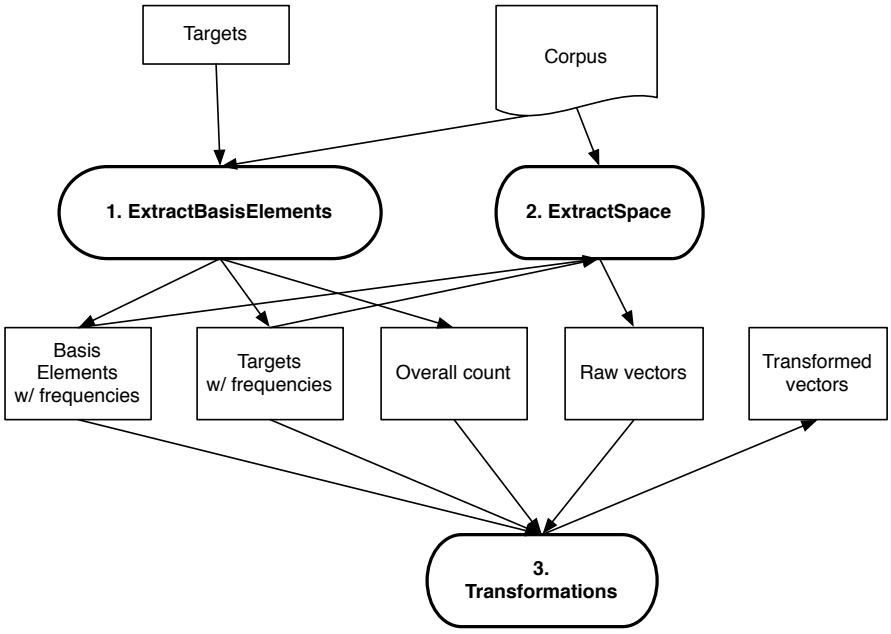

Figure 1: Flow of Information in DEPENDENCYVECTORS

File	Naming convenion
Target frequencies	`cs_pv.targets`
BE frequencies	`cs_pv.bes`
Total frequency	`cs_pv.total`
Raw vectors	`cs_pv.vectors`
Transformed vectors	`cs_pv.ll.vectors`

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

2.3 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 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`. The prefix `contextspec` is added automatically.

–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)

–basisMapping <bm> : Basis mapping function (no default). Currently available: `lex` (use words as dimensions of the semantic space) and `gramlex` (use word-dependency relation pairs as dimensions of the semantic space).

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.[1] Example call for `ExtractBasisElements`:

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

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.

[1] The list of all context specifications is read from the variable `Parameters.allCSFiles`, and the list of all path value functions from `Parameters.allValuations`.

2.4 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[2].

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

–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`. The prefix `contextspec` is added automatically.

–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).

–basisMapping <bm> : Basis mapping function (no default). Currently available: `lex` (use words as dimensions of the semantic space) and `gramlex` (use word-dependency relation pairs as dimensions of the semantic space).

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 minimal.txt --lex
```

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

Having individual target and basis element files for different parametrisations may seem unnecessary. However, recall that these files also contain frequencies (computed, for example, by `ExtractBasisElements`). These typically vary between parameterisations and are necessary for computing log likelihoods. Note that if you haven't computed the basis elements for all these parametrisations, the program will presumably fail while trying to open these files and give you surprising error messages.

To enable the extraction of semantic spaces with "raw counts", `ExtractSpaces` 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.

2.5 Using `LLT/PMT`

These program perform log-likelihood and pointwise mutual information transformations 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`.The prefix `contextspec` is added automatically.

–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
```

2.6 Example

The directory `sample` contains a "minimal" running example for creating a semantic space: a two-sentence corpus in "full" format (`sample_corpus_full`) and triple format (`sample_corpus_triples`) as well as a set of two target words (`targets`). Executing the script `create_space.sh` will run the complete pipeline and create a space in the subdirectory `vectors`.

old	new
:	_COLON_
\t	_TAB_
" (double quote)	(empty string)
\s (space)	_ (underscore)
`	_BACKTICK_

Table 2: String Replacements in Corpus, Target, and BE expressions

2.7 Details on String Handling

MINIPAR shows a somewhat unusual behaviour for parsers in that multi-word expressions can occur as single nodes; these nodes are labelled with strings including spaces. To correctly handle these cases, spaces (and a small number of other special characters) are replaced during processing. The replacements are given in Table 2. In addition, all words are lowercased, to generalise over the capitalisation of first words in sentenes. The replacements are defined (and can be altered) in the class `MiniparEncoding`. For technical reasons, it is not recommended that targets or basis elements contain spaces.

This replacement takes place both for corpora, and for the lists of basis elements and target words read from files. As consequence, target word lists written by DEPENDENCYVECTORS into the target data directory can differ from the original target word lists in terms of target normalisation. For example, *mull over* will be replaced by *mull_over*.

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.

Almost all modifications and extensions should be possible by modifying exactly one place in the code, since all frontends (`ExtractBasisElements`, `ExtractSpace`, and `LLT`) use the same backend libraries.

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 two changes. First, a new class must be written that implements the `BasisMapping` interface. See the top of `Parameters.java` for the two built-in basis mapping functions as examples. Second, a new case must be added to the (`if (mapping.equals(...))` conditional in the `Parameters()` constructor.

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. The filenames of context specification files must start with `contextspec_`.

4.4 Adding new path value functions

New path value functions can be added by extending the `if valuation.equals(X)` branch in the `Parameters.pathValue` method. The names of the new path value functions also have to be recorded in the `Parameters.allValuations` array. This makes them permissible command line arguments, and they can be called with `-X`.

5 Troubleshooting

5.1 Logfile

Each module records all important events, such as errors, in a file called `log.txt` in the `targetdir`. If the output does not look as expected, this file might be worth a look. Note, however, that this file is overwritten if subsequent modules are run one after another (e.g., first `ExtractBasisElements` and then `ExtractSpace`).

5.2 "Class def not found" when running one of the bin scripts

You need to run the makefile first (`make`).

5.3 Memory trouble: "Invalid maximum heap size" or "Out of memory"

Java requires that the heap size is specified when the VM is started. This means that I needed to fix the heap size to some constant; I picked 1 GB for all classes that are called from the command line. Depending on your circumstances, this may be too little or too much.

- If you have a machine with 1 GB memory or less, you will likely see a "could not allocate memory" error.
- On the other hand, if you try to compute a huge space, you will likely see an "out of memory" error when DEPENDENCYVECTORS attempts to allocate space for its data structures.

You can change the heap size with the compiler switch `-Xmx<size>`. The easiest way to do that is to change the java `-Xmx1G` statement in the bash scripts in `bin`, e.g. to `java -Xmx500m` (less memory) or to `java -Xmx5G` (more memory). A final complication is that programs of such sizes run into the limitations of 32-bit memory addressing. 32-bit Java can usually deal with heap spaces up to around 1.6GB, but not more. Thus, if you need to compute larger spaces, you will either need to split the target words into multiple batches, find a 64-bit machine with 64-bit Java to use, or implement more memory- efficient data structures (please let me know if you do that ;-)).

5.4 Further bugs

It is entirely possible that there are still bugs in the system. If you encounter one, please let me know, and I will try to fix it. (Note that fixing bugs is easier and faster the more detailed the error report; it would be optimal if you could send me a "minimal example").

Note that Vector Spaces computed with one parameter setting are not guaranteed to be interpretable with different settings, so if weird errors occur, first make sure that you haven't modified the parameters between two runs.

6 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`.
- Version 2.1 (May 2007)
  - Doc: revised documentation
  - Implementation: Administration of command line arguments centralised
  - `FrequencyList` and `SemanticSpaceArray` debugged.
  - `LLT` debugged.
- Version 2.1a (July 2007)
  - Minor debugging.
  - Added usage sample.
- Version 2.2 (August 2007)
  - More general recognition of the triples vs. full MINIPAR format
  - More general handling of lemmas
- Version 2.3 (January 2008)
  - Bugfixes (handling of antecedents; handling of default context specification)
- Version 2.3a (March 2008)
  - More bugfixes (handling of brittle log transformation)
- Version 2.4 (July 2008)
  - Two bugfixes in triple format reader (thanks to Bram Vandekerckhove)
  - Principled handling of multiple edges in triple format
  - Move to Java 5
  - Modular specification of basis mapping function
- Version 2.4a (December 2009)
  - Refactorization of the Corpus code to improve extensibility
  - Refactorization of the package structure