TGrep2 Database Tools (TDT) User Manual

Judith Degen
Brain and Cognitive Sciences
University of Rochester

March 29, 2011

1 Introduction

The TGrep2 Database Tools are a collection of command line scripts written by Florian Jaeger, Austin Frank, Judith Degen, and Neal Snider that allow you to extract data from large corpora and combine this data into a comprehensive database in a format suitable for importing into your favorite statistical analysis program.

The following steps are involved in doing a corpus analysis of linguistic data with the TDT Tools:

1. . . .come up with an interesting question. . .
2. Create TGrep2 patterns to run your corpus queries with.
3. Extract data from a corpus and create a database.
4. Do statistical analysis on your data.

We leave step 1 as an exercise to the reader. For information on TGrep2 pattern syntax consult the TGrep2 manual (Rohde 2005).

This manual will focus on how to execute step 3. Note that the TGrep2 Database Tools are a collection of scripts initially written for individual use and to solve very specific problems. In consequence, some scripts may not behave the way you intend them to, and some features you think the TDT Tools should have will not be implemented. Please report any bugs or feature requests to `tdt@bcs.rochester.edu`.

2 Getting started

The TDT Tools require perl and at least python 2.5. This is the case on most Unix machines. In addition, you will need to install TGrep2.

2.1 Setting environment variables

Set the following environment variables in your profile:

- `TGREP2_CORPUS` - Set this to the TGrep2 default corpus. If you run TGrep2 without a corpus argument, it will run on this corpus.
- `TGREP2ABLE` - Set this to the directory that contains the TGrep2 corpora.
- `TDTlite` - Set this to the directory that contains the TDT scripts.
- `TDT_DATABASES` - Set this to the directory that contains the TDT databases.

In addition, add the TDTlite script directory to your `PATH` variable. For example, this is an example of what to add to your profile (file .bash_login, .bash_profile, or .profile in your home directory) if you're operating in a bash.

```
TGREP2ABLE="/corpora/TGrep2able"
export TGREP2ABLE
TGREP2_CORPUS="$TGREP2ABLE/swbd.t2c.gz"
export TGREP2_CORPUS
TDTlite="/corpora/TDTlite/"
export TDTlite
TDT_DATABASES="/corpora/TDT/databases/"
export TDT_DATABASES
PATH="$HOME/bin:$PATH:/corpora/TDTlite"
```

In a C shell, add the following to your profile (file .login in your home directory) instead:

```
setenv TGREP2ABLE /corpora/TGrep2able
setenv TGREP2_CORPUS $TGREP2ABLE/swbd.t2c.gz
setenv TDTlite /corpora/TDTlite/
setenv TDT_DATABASES /corpora/TDT/databases/
setenv PATH $PATH:corpora/TDTlite
```

2.2 Creating a project

Start by creating your project directory. If you are planning on using the **run** script included in the TDTlite Tools to create your database, you will need to create the following directories in your project directory:

- **data** is the output directory for **run**'s calls to TGrep2, i.e. this is where the TGrep2 output files (with the extension .t2o) will be stored in 'data/*corpus_name*'.

- **results** is where the final database file ('*corpus_name*.tab') will be stored by **collectData**.
- **ptn** will contain your TGrep2 pattern files (see the TGrep2 manual for more information on creating patterns). ptn itself contains further subdirectories for the different variable types:
  - **CatVar** contains .ptn files that are assumed to be categorical variables. The .t2o files generated from .ptn files in CatVar will contain only the IDs of the matched cases, nothing else.
  - **CtxtVar** contains .ptn files that will output context sentences that occur either directly precede or follow the match. The filename should be *number_of_sentences-b/a*.ptn. That is, to get the 3 sentences preceding the match, create a file **3-b.ptn** that contains the pattern to match. To get the 2 sentences following the match, create a file **2-a.ptn**.
  - **ParseVar** contains .ptn files that will output the match's parse tree.
  - **NodeVar** contains .ptn files that are assumed to extract node labels (for terminals the node labels provide part-of-speech information). In the pattern, label the node containing the part-of-speech (or the node label you are interested in) with '=pos' . For example, to get the part-of-speech of the head verb in a VP, create a .ptn file in NodeVar with the content: `/^VP/ < (/^V/=pos < *)`
  - **StringVar** contains .ptn files that are assumed to extract words. In the pattern, label the node to print with '=print'. For example, to print all VPs: `/^VP/=print`

  Create one .ptn file per subpattern you wish to extract, in the appropriate subdirectory. For each .ptn file, one TGrep2 output file with the same name (but without the extension .t2o instead of .ptn) will be created in 'data/*corpus_name*'.

In addition to these subdirectories, you will need to create an **options** file (see section 3.1.2 for details) and a macro file (see the TGrep2 user manual for information on how to create macro files) in the top level of your project directory, parallel to the three subdirectories **data, ptn** and **results**. There are three different options for naming your macro file. The easiest option is to name it **MACROS.ptn**. Alternatively, if you know which corpus you will use it with, name it *corpus*MACROS.ptn (see section 3.1.2 for *corpus* tags). For example, if you want to use your macro file for a TGrep2 search on the written BNC, you can name it **bncwMACROS.ptn**. Finally, if you want to use more than one macro file simultaneously, name them in the following way: MACROS-$name_1$.ptn, MACROS-$name_2$.ptn, ..., MACROS-$name_n$.ptn.

3 Creating a database

Once your pattern and macro files are ready, you are ready to create your database. There are several ways to do this:

- Use the **run** script. You have two options:
  - The easier alternative (recommended for beginners, but allows slightly less flexibility) is to specify in an **options** file the options for **run** to create a script called **collectData** which combines TGrep2 output files to create your database all in one step.
  - The other alternative is to create the **collectData** shell script manually. This involves knowledge of the perl scripts described in section 4, but allows for greater flexibility specifying the manner in which to add variables to your database.
- Alternatively, you can do everything manually: First generate TGrep2 output files by running TGrep2 on your patterns individually, and then combine the output files to a database via use of the perl scripts described in section 4.

These methods are described in sections 3.1 and 3.2.

3.1 Creating a database with `run`

3.1.1 The run script

Usage

run [**-h**] [**-c** *corpus*] [**-e**] [**-j**] [**-collect**] [**-o**] [*macrofiles*]

Options

-h[**elp**]	Prints help.
-c[**orpus**] *corpus*	Specifies the corpus to extract or collect data from. Default is swbd (Switchboard). See secton 3.1.2 for corpus tags.
-e[**xtract**]	Extracts all matches for the patterns specified in one or more macro files from the corpus specified by '-c *corpus*' (default: 'no'). If no *macrofiles* argument is provided, **run** searches for a macro file named '*corpus*MACROS.ptn' or 'MACROS.ptn' in the top level of the project directory. The output will be saved in 'data/*corpus*' in the project directory. If *macrofiles* is provided, one subdirectory for the output of each macro file is created in the data directory. Use the **-j** option to concatenate all output files and save them in the directory 'data/*corpus*'. Naming convention for the *macrofiles* argument: 'MACROS-*name*.ptn'. When passing the *macrofiles* arguments, specify only *name*. Default is not to extract.

-collect	Collects the information from the TGrep2 data files in the '../data' directory and combines them to a database in '../results'. Requires the 'collectData' script to be in the same directory. See section 3.1.3 for information on how to create 'collectData' manually. Default is not to collect.
-i[mport]	Imports the collected information into an R file (not implemented yet).
-j[oin]	Joins the output of each macro file for each TGrep2 pattern into one file in the 'data/*corpus*' directory. Default is not to join.
-k[eep]	Prevents deletion of the 'collectData' script from the project directory top level after it has been used to collect data from the .t2o files.
-o	Like **-collect**, but creates the 'collectData' script on the fly from options specified in a file named 'options' in the project directory top level. See section 3.1.2 for information on how to create an 'options' file. Default is not to collect.

Examples

- The following will call TGrep2 on the pattern files specified in the 'ptn' directory and the macro file 'bncwMACROS.ptn' or 'MACROS.ptn' in the project directory top level. It will then create 'collectData' from the options specified in the 'options' file and 'collectData' will combine the data from the specified files in 'data/bncw' to a database in 'results' called 'bncw.tab'.

  ```
  $ run -c bncw -e -o
  ```

  See section 3.1.2 for details on creating an 'options' file.

- This command does essentially the same, with the difference that instead of expecting an 'options' file, it expects 'collectData' in the same directory.

  ```
  $ run -c bncw -e -collect
  ```

  See section 3.1.3 for details on creating a 'collectData' script.

3.1.2 Method 1: specifying an `options` file

This is the easiest way to create your database, but it also offers the least flexibility. The **run** script creates a database in two steps: it first calls TGrep2 with the patterns specified in the .ptn files in the pattern directory and the macro file. It then builds a database of the output according to the options specified in the **options** file.

The purpose of the **options** file is to specify a number of parameters for the extracted data to be properly combined into a database.

Obligatory parameters
The obligatory parameters are the location of the **data** directory that contains the directory (named after the corpus you run the queries on) that in turn contains the TGrep2 .t2o output files and the **results** directory that the database should be written to.

Usage
data=*/path/to/data/directory*
results=*/path/to/results/directory*
For example:

```
**************************************************************************
data=/home/projects/perspective/data
results=/home/projects/perspective/results
**************************************************************************
```

Initializing the database
To create a database, first initialize it with a column of match IDs which are taken from the file you specify (without file extension) and is assumed to be in the data directory that you specified above.[1].

Usage
init *IDfile*
For example, if *IDfile* is the file ID.t2o in the data directory:

```
**************************************************************************
init ID
**************************************************************************
```

Adding variables to your database
To add different variable columns to your database, use the **add** command.

Usage
add *variabletype arguments*
Depending on what kind of variable you are adding, *variabletype* and *arguments* will differ. The following options are available for *variabletype*:

- CategoricalVar - adds a categorical factor, i.e. a value (specified by the user) if a given ID finds a match for the specified variable.[2]
- CondProb - adds one column with the joint frequency of the value of the specified variable and the predicted event (i.e. your database). That is, a column with the frequency of the value of the specified variable in your database. It adds a second column with the conditional probability of the target event (i.e. your database) given the value of the specified variable in the corpus.

[1] Specifying this option is like running the initDatabase.pl script
[2] see section 4.2.2 for information on how to specify different values by using the addCategoricalVar.pl script.

- CountVar - adds a count variable, i.e. the number of matches for a given ID.
- Frequency - adds a column with the frequency of the words in the specified variable.
- InfoDensity - adds one column with the information of the specified variable (already in the database) given a 3gram model.[3] It adds a second column providing the length of the specified variable in words.
- LemmaVar - adds a lemma variable, i.e. the specified variable/word's lemma.
- LengthVar - adds a length variable, i.e. the total length of the specified variable's value (number of words) for a given ID.
- Phonology - adds segmental information about the specified variable (already in the database): one column for the phonemic transcription of the word (see the Carnegie Mellon Pronunciation Dictionary for transcription information), one column each for place and manner of articulation of the first and last phoneme (unless they are vowels, in which case the label "vowel" is inserted), and a column specifying syllable structure (one digit per syllable, 0 for no stress, 1 for primary stress, 2 for secondary stress).
- NodeVar - adds a node label variable, e.g the specified variable/word's part of speech.
- StringVar - adds a string variable, i.e. terminals/words to the database.

Variable type specific usages:
add CategoricalVar *variablename=label*$_1$*:file*$_1$*,label*$_2$*:file*$_2$*,...,label*$_n$*:file*$_n$
add CondProb [*variablename*]
add CountVar [*variablename=*]*filename*
add Frequency *variablename*
add InfoDensity *variablename*
add LemmaVar [*variablename=*]*filename*
add LengthVar [*variablename=*]*filename*
add Phonology *variablename1* [*variablename2*] [...]
add NodeVar [*variablename=*]*filename*
add StringVar [*variablename=*]*filename*

For example, the following commands add

- a string variable column for the target event's form by adding the data from the Form.t2o file. The name of the variable column in the database and the file name are the same.
- a node label variable column for the target event's part-of-speech by adding the data from the POS.t2o file. The name of the variable column and file name are the same.

[3] see section 4.2.5 for information on how to specify different ngram models by using the addInformationDensity.pl script

- a lemma variable column called 'Lemma_Form' containing the lemma of the word form in Form.t2o.
- a string variable column that adds the entire sentence containing the match from the TOPstring.t2o file. The column name ('Sentence') is different from file name.
- a categorical variable column 'PPfrom' that contains a label "pp" if that match contains a PP after a following NP and an empty cell if it doesn't (according to the data in the PPafterNP.t2o file).
- the information density of the NP preceding the match.
- columns containing phonological information about the entries in the Form column.
- one column with the joint frequency of each value of Form and the target event, and one column with the conditional probability of the target event given the value of Form in the entire corpus.
- a count variable that for each row ID contains the number of matches for that ID in the PPFROMafterNP file.
- a length variable that for each row ID contains the total length (ie number of words) of all matches for that ID in the PPFROMafterNP file.
- a frequency variable that contains the frequency of all the verbs in the Form column, estimated from the corpus.

```
**************************************************************************
add StringVar Form=Form
add NodeVar POS
add LemmaVar Form
add StringVar Sentence=TOPstring
add CategoricalVar PPfrom=pp:PPafterNP
add InfoDensity NPpreceding
add Phonology Form
add CondProb Form
add CountVar CntPPfrom=PPFROMafterNP
add LengthVar LenPPfrom=PPFROMafterNP
add Frequency Form

**************************************************************************
```

3.1.3 Method 2: creating `collectData` manually

If you want to retain the previous method's advantage of extracting your data, creating your database, and adding all the desired variables to it in one step via batch mode, but also want additional flexibility in specifying certain options, you can create the **collectData** shell script, which is essentially a collection of calls to the perl scripts described in section 4, manually.

The first line of your script depends on your shell, for example

```
#!/bin/bash
```

if you're in a bash, or

```
#!/bin/csh -f
```

if you're in a C shell. This will be followed by a directory change to your results directory, and setting of project variables `Pdata, Presults` to the **data** and **results** directory, respectively. For example, in a C shell:

```
cd /home/projects/perspective/results
setenv Pdata /home/projects/perspective/data/bncw
setenv Presults /home/projects/perspective/results
```

Instead, if you're in a bash:

```
cd /home/project/perspective/results
export Pdata=/home/projects/perspective/data/bncw
export Presults=/home/projects/perspective/results
```

Next, you need to initialize the database with a column of match IDs by calling the **initDatabase.pl** script on the match ID file in your data directory. See section 4.2.1 for details on how to use **initDatabase.pl**. For example, the following will print an initialization message and call **initDatabase.pl**:

```
echo Creating new corpus file bncw.tab
initDatabase.pl -roc bncw --files $Pdata/ID
```

You can now add further variable columns to the database via calls to the **addX.pl** scripts. See section 4 for details on how to use the scripts for adding different variable types to the database. For example, the following will add a string variable, a node label variable, a lemma variable, a categorical variable, a variable coding information density, phonological information, conditional probabilities, a count variable and a length variable. Finally, change back into the TDTlite directory (which is the location of 'run').

```
echo Beginning data extraction...

addStringVar.pl -roc bncw -f Form=$Pdata/Form
addNodeVar.pl -roc bncw -f POS=$Pdata/POS
addLemma.pl -roc bncw -f Form=$Pdata/Form
addStringVar.pl -roc bncw -f Sentence=$Pdata/TOPstring
addCategoricalVar.pl -roc bncw -f PPfrom 1 $Pdata/PPFROMafterNP
addInformationDensity.pl -roc bncw -f NPpreceding 3
addPhonology.pl -roc bncw -f Form
addConditionalProbability.pl -c bncw -f Form
addCountVar.pl -roc bncw -f CntPPfrom=$Pdata/PPFROMafterNP
addLengthVar.pl -roc bncw -f LenPPfrom=$Pdata/PPFROMafterNP

cd $TDTlite
```

See Appendix B for a full example of a **collectData** script.

3.2 Method 3: Adding TGrep2 output to database manually

Use this method to add individual variables to your database if you already have TGrep2 .t2o output files that you want to combine to a database, or information about which you want to add to a database. Each of the scripts in 4.2 can be used to add a different variable type to your database. See section 4.1 for general usage information.

4 Perl scripts

These scripts combine the data files from the project data directory to create a database in the results directory. You can use these scripts individually or modify **collectData** to call the scripts you want. Each script will add one or more new variables (i.e. columns) to the database - the variable type should determine which script you use.

All of the scripts assume that there is a database file named *corpus*.tab (see the **-c** option in section 4.1 for *corpus* options) in the directory the script is being called from. Use the **-d** option to pass the script a different location. The exception is **initDatabase.pl**, which initially creates a database *corpus*.tab from IDs specified in an .t2o TGrep2 output file.

The next section describes the options that may be used with all the perl scripts unless noted otherwise. Sections 4.2.1 to 4.3.3 describe the individual scripts.

4.1 General options

Usage

perl *scriptname* [**-horw**] [**--about**] [**-c** *corpus*] [**-d** *database*] [**--default** *value*] [**-f** *variable(s)*] [**--files***file(s)*] [**--noversion**]

Options

--about Provide information about the program. The default is not to.

-c[**orpus**] *corpus* *corpus* is an argument describing the corpus to be used. This determines a variety of things, e.g. which ngram files will be used and how corpus-specific information will be stripped from the terminal output of TGrep2 (e.g. when extracting strings from the corpus). Currently the following arguments are recognized:

- arab - the Arabic Treebank (arabic-collapsed.t2c.gz)
- bnc - the entire BNC (BNC.parsed.t2c.gz)
- bncs - the spoken parts of the BNC (BNC_spoken.parsed.t2c.gz)
- bncw - the written parts of the BNC (BNC_written.parsed.t2c.gz)
- brown - Brown corpus (brown.t2c.gz)
- chin - the Chinese Treebank (chtb6.t2c.gz)
- ice - International Corpus of English (icegb.t2c.gz)
- negra - NEGRA (negra.t2c.gz)
- swbd - Switchboard Corpus (swbd.t2c.gz)
- tiger - TIGER corpus (tiger.t2c.gz)
- wsj - Wall Street Journal (wsj_mrg.t2c.gz)
- ycoe - York-Toronto-Helsinki Parsed Corpus of Old English Prose (ycoe.t2c.gz)

-d[**atabase**] *database* *database* is the filename of the database you want to manipulate (create, add information to, etc.). The default is *corpus-name*.tab. Note that this implies that the database file must be in the scripts folder unless a path to it is specified.

--default *value* *value* will be the default value for any empty cell of the variable(s) modified by the scripts. The variable must be given with the **-f** option.

-f[actors] *variable(s)*	*variable(s)* is one or more names of variables (i.e. columns) in the database that you want to create, import, or manipulate. If there is more than one variable, separate variable names by commas with no intervening spaces (i.e. $variable_1$,$variable_2$,...$variable_n$). Most scripts allow several variable names as input and will loop over all variables. In case the script expects an input file (e.g. a TGrep2 .t2o output file) for each variable, these can either be provided separately (see **--files** option) or in conjunction with the variable specification by using $variable_1$=$file_1$,$variable_2$=$file_2$,...,$variable_n$=$file_n$
--files *file(s)*	Use this option to specify one or more *files* to be read from. If there is more than one file, separate file names by commas with no intervening spaces (i.e. $file_1$,$file_2$,...,$file_n$). When used in conjunction with **-f**, the order of *variables* and *files* must be the same (i.e. **-f** $variable_1$,$variable_2$,...$variable_n$ **--files** $file_1$,$file_2$,...,$file_n$)
--h[elp]	This option is not yet implemented. See this manual for information about options.
--noversion	Don't print the version header. The default is to print the header.
-o[verwrite]	Overwrite cells that already have a value if a new value is obtained by the operations executed by the script (e.g. by importing information from a TGrep2 .two output file). The default is not to overwrite.
-r[eset]	Reset all cell values of the variables (i.e. columns) specified with the **-f** option to the default value (usually an empty cell). The default is not to reset.
-w[arnings]	Print detailed warnings. The default is not to print.

4.2 Adding variables - `initDatabase.pl` and `addX.pl` scripts

4.2.1 initDatabase.pl

This script creates your database from a set of match IDs provided in a TGrep2 .t2o output file. If you want to combine different variables (corresponding to different TGrep2 output files) to form a database, initDatabase.pl must be run before any variable scripts can be run. It creates a *corpusname*.tab file in the scripts directory unless specified otherwise (see **-d** option above). For example, if you want to create a database from data that was extracted from the written BNC and that contains an ID file ID.t2o in /home/perspective/data/bncw:

```
$ initDatabase.pl -roc bncw --files /home/perspective/data/bncw/ID
```

The **-r, -o, -c**, and **--files** options apply as specified in section 4.1. Note that the file extension should not be included in the path.
The following options are not supported: **--default**, **-f**

4.2.2 addCategoricalVar.pl

This script adds a categorical variable to the database. The added column will contain a value (specified in the options) for the matched rows and empty cells for the non-matched rows. After the column name, this script expects the value to enter in the column for matched rows. For example, for a file PPafterNP.t2o that contains an ID for each of the matches in the entire database in case that match contains a PP following an NP, if you want to add a column CntPP that contains a 1 for every ID in the file PPTOafterNP.t2o:

```
$ addCategoricalVar.pl -roc bncw -f CntPP 1 /project/data/bncw/PPTOafterNP
```

The **-r, -o, -c**, and **-f** options apply as specified in section 4.1. 1 is the value to enter for matched rows, the path to PPafterNP specifies the TGrep2 output file containing the PPs (and match IDs). Note that the file extension should not be included in the path. To add an empty value, specify " " as the level value.

More than one factor level may be added to the specified variable in two ways:

1. You can run addCategoricalVar.pl a second time, using the **-o** option to overwrite existing values. The following will add two factor levels with values 1 and 2 to the variable **CntPP**.
   ```
   $ addCategoricalVar.pl -roc bncw -f CntPP 1 /project/data/bncw/PPTOafterNP
   $ addCategoricalVar.pl -oc bncw -f CntPP 2 /project/data/bncw/PPFROMafterNP
   ```

2. Alternatively, you can add both levels in one step. After the variable name you can specify arbitrarily many factor levels with the syntax:
   $level\_value_1$ $file_1$ $level\_value_2$ $file_2$ ... $level\_value_n$ $file_n$
   The following command will have the same effect as alternative 1.
   ```
   $ addCategoricalVar.pl -oc bncw -f CntPP 1 /project/data/bncw/PPTOafterNP
     2 /project/data/bncw/PPFROMafterNP
   ```

The following options are not supported: **--files**

4.2.3 addConditionalProbability.pl

This script adds two columns to your database: one column called JFQ_*variablename* with the joint frequency of the value of the specified variable and the predicted event (i.e. your database). That is, a column with the frequency of the value of the specified variable in your database. The second column, called CndP_*variablename*, contains the conditional probability of the target event (i.e. your database) given the value of the specified variable in the corpus. This script does not require .t2o output files to extract data from, rather it expects the column name of an already existing variable in the database to calculate the conditional probability of.

For example, consider a database that contains all complement clauses in the Switchboard. It further contains a column **Verb** with all the different verbs that occur immediately before

the complement clause. For each row (i.e. each value of Verb), addConditionalProbability.pl will add p(complement clause|Verb). That is

$$p(\text{complement clause}|\text{Verb=“think”})$$
$$p(\text{complement clause}|\text{Verb=“believe”})$$
$$p(\text{complement clause}|\text{Verb=“guess”})$$
$$\vdots$$

To add conditional probabilities for **Verb** in the database swbd.tab just described:

```
$ addConditionalProbability.pl -c swbd -f Verb
```

The following options are not supported: **--default**, **--files**

4.2.4 addCountVar.pl

This script adds a column to your database with the number of matches in the TGrep2 .t2o output file for that row ID. For example, if you have a file Disfluencies.t2o that contains all the disfluencies for each matched sentence in the BNC, and you want a disfluency count for each sentence:

```
$ addCountVar.pl -roc bnc -f CntDis --files /project/data/bnc/Disfluencies
```

Here, CntDis is the name of the column that will contain the disfluency count. Specify the path to the .t2o file to count from with the **--files** option.
There are two ways to add more than one count variable at once:

1. Specify the variable names (the column names to be created) with the **-f** option and the file names with the **--files** option. Variable and file names should be comma-delimited, with no intervening spaces (i.e. **-f** *variable*$_1$,*variable*$_2$,... **--files** *file*$_1$,*file*$_2$,...). For example:
   ```
   $ addCountVar.pl -roc bnc -f CntDisPreceding,CntDisFollowing
     --files /project/data/bnc/Dpreceding,/project/data/bnc/Dfollowing
   ```

2. Alternatively, you can specify both the variable names and the file names with the **-f** option: **-f** *variable*$_1$=*file*$_1$,*variable*$_2$=*file*$_2$,.... For example:
   ```
   $ addCountVar.pl -roc bnc -f CntDisPreceding=/project/data/bnc/Dpreceding,
     CntDisFollowing=/project/data/bnc/Dfollowing
   ```

This script supports all options. See section 4.2.7 for the difference between addCountVar.pl and addLengthVar.pl.

4.2.5 addInformationDensity.pl

This script adds two columns to the database: one column **Information_***variable_n***gram** containing the information of the specified variable, given an n-gram model.[4] The second column, **Length_***variable_n***gram**, contains the length of the specified variable (in number of words). Specify the variable with the **-f** option, and n as the last argument. For example, the following will add columns **Information_Form_2gram** and **Length_Form_2gram** to your database, which will contain information and length of the entries in the column **Form**, given a bigram model.

```
$ addInformationDensity.pl -roc bncw -f Form 2
```

The following options are not supported: **--files**. **-f** takes exactly one argument.

4.2.6 addLemma.pl

This script adds a column *variablename*_Lemma that contains the lemma of the specified variable (i.e. this variable should only contain one word per row). Specify the variable with the **-f** option and the database with the **-c** or **-d** option.

For example, to create a column **Form_Lemma**, containing lemma information about the **Form** column, to the database bncw.tab:

```
$ addLemma.pl -roc bncw -f Form
```

To add lemma information for more than one column, specify all column names in comma-separated format (without intervening spaces) with the **-f** option. For example, the following will create two columns, **Form_Lemma** and **NPpreceding_Lemma**:

```
$ addLemma.pl -roc bncw -f Form,NPpreceding
```

The following options are not supported: **--files**

4.2.7 addLengthVar.pl

This script adds a column to your database that contains the total variable length (in number of words) for that row ID, given a TGrep2 .t2o file containing IDs and strings. Specify the name of the column to be created with the **-f** option, and the files from which to compute the length data with the **--files** option. For example, given a file Disfluencies.t2o that contains all disfluencies that occur in any of the matched sentences, the following will add a column **LenDis** with the total length of all disfluencies that occur in each matched sentence to the database bncw.tab.

```
$ addLengthVar.pl -roc bncw -f LenDis --files /project/data/bncw/NP
```

There are two ways to add more than one length variable at once:

[4] This information will be read from corpus-specific n-gram database files.

1. Specify the variable names (the column names to be created) with the **-f** option and the file names with the **--files** option. Variable and file names should be comma-delimited, with no intervening spaces (i.e. **-f** *variable*$_1$,*variable*$_2$,... **--files** *file*$_1$,*file*$_2$,...). For example:
   ```
   $ addLengthVar.pl -roc bnc -f LenDisPreceding,LenDisFollowing
     --files /project/data/bnc/Dpreceding,/project/data/bnc/Dfollowing
   ```

2. Alternatively, you can specify both the variable names and the file names with the **-f** option: **-f** *variable*$_1$=*file*$_1$,*variable*$_2$=*file*$_2$,.... For example:
   ```
   $ addLengthVar.pl -roc bnc -f LenDisPreceding=/project/data/bnc/Dpreceding,
     LenDisFollowing=/project/data/bnc/Dfollowing
   ```

This script supports all options.

On the difference between addCountVar.pl and addLengthVar.pl
The difference between addCountVar.pl and addLengthVar.pl is that the former counts the number of matches for a given ID, while the latter computes the total length of a given ID in number of words. Consider the following example, an excerpt from our fictitious file Disfluencies.t2o.

```
5:34 um
5:34 you know
5:34 I mean
5:34 Jo-
```

The ID 5:34 has four disfluencies associated with it. That is, addCountVar.pl (see section 4.2.4) will add '4' in the **CntDis** column at the row with the row ID 5:34. However, the total length of all disfluencies is 6, so addLengthVar.pl will add '6' in the **LenDis** column at the row with the row ID 5:34. In consequence, if each ID has only one word associated with it, addCountVar.pl and addLengthVar.pl will yield the same result.

4.2.8 addPhonology.pl

This script adds 6 columns with phonological information about the specified variable: **PHON**_*variable*, **PHONstartPLC**_*variable*, **PHONstartMNR**_*variable*, **PHONendPLC**_*variable*, **PHONendMNR**_*variable*, **SYLS**_*variable*. The first column contains the phonemic transcription of the word (see the Carnegie Mellon Pronunciation Dictionary for transcription information). The second and third column contain information about place and manner of articulation of the first phoneme in the word, while the fourth and fifth column contain the same information for the last phoneme in the word. If the phonemes are vowels, these columns will contain the label 'vowel'. The las column specifies the word's syllable structure (one digit per syllable, '0' if no stress, '1' if primary stress, '2' if secondary stress).
For example, the following adds phonological information about the variable **Form** (which already exists in the database) to the database bncw.tab.

```
$ addPhonology.pl -roc bncw -f Form
```

To add phonological information about more than one variable, specify all column names in comma-separated format (without intervening spaces) with the **-f** option.
The following options are not supported: **--files**

4.2.9 addNodeVar.pl

This script adds a column that contains part-of-speech information contained in the specified file. Specify the column name with the **-f** option and the database with the **-c** or **-d** option. For example, to create a column **POS**, containing part-of-speech information taken from the **POS.t2o** file, to the database bncw.tab:

```
$ addNodeVar.pl -roc bncw -f POS --files /home/project/data/bncw/POS
```

There are two ways to add more than one POS variable at once:

1. Specify the variable names (the column names to be created) with the **-f** option and the file names with the **--files** option. Variable and file names should be comma-delimited, with no intervening spaces (i.e. **-f** *variable*$_1$,*variable*$_2$,... **--files** *file*$_1$,*file*$_2$,...). For example:
   ```
   $ addNodeVar.pl -roc bnc -f POS1,POS2 --files /project/data/bnc/Pos1,
     /project/data/bnc/Pos2
   ```
2. Alternatively, you can specify both the variable names and the file names with the **-f** option: **-f** *variable*$_1$=*file*$_1$,*variable*$_2$=*file*$_2$,.... For example:
   ```
   $ addNodeVar.pl -roc bnc -f POS1=/project/data/bnc/Pos1,POS2=
     /project/data/bnc/Pos2
   ```

This script supports all options.

4.2.10 addStringVar.pl

This script adds a column that contains the words corresponding to a given pattern, to be taken from a specified file. For example, the following adds a column **Verb** with information from the file Verb.t2o to the database bncw.tab:

```
$ addStringVar.pl -roc bncw -f Verb=/project/data/bncw/Verb
```

There are two ways to add more than one string variable at once:

1. Specify the variable names (the column names to be created) with the **-f** option and the file names with the **--files** option. Variable and file names should be comma-delimited, with no intervening spaces (i.e. **-f** *variable*$_1$,*variable*$_2$,... **--files** *file*$_1$,*file*$_2$,...). For example:
   ```
   $ addStringVar.pl -roc bnc -f NPpreceding,NPfollowing --files
     /project/data/bnc/NPprec,/project/data/bnc/NPfoll
   ```

2. Alternatively, you can specify both the variable names and the file names with the **-f** option: **-f** *variable*$_1$=*file*$_1$,*variable*$_2$=*file*$_2$,.... For example:
   ```
   $ addStringVar.pl -roc bnc -f NPpreceding=/project/data/bnc/NPprec,
     NPfollowing=/project/data/bnc/NPfoll
   ```

This script supports all options.

4.3 Further handy scripts

4.3.1 stripTGrep2Terminals.pl

This script strips "junk" (e.g. speaker information, disfluencies, other markers that may impede ease of reading) from TGrep2 output, reformats punctuation, and prints it to standard output.
For example, the following will strip extra markers from the file adjp.t2o.

```
$ stripTGrep2Terminals.pl --files adjp.t2o
```

The first line of adjp.t2o before stripping:

```
She had a rather massive stroke \[ about , \+ uh , about \] uh , eight
months ago I guess . E_S
```

After stripping:

```
She had a rather massive stroke about, uh, about uh, eight months ago I guess.
```

Instead of using the **--files** option, stripTGrep2Terminals.pl also accepts input from standard in, e.g. the piped output of a TGrep2 query. The following command does the same as above (assuming that `"TOP << ADJP"` is the pattern that generated the contents of adjp.t2o):

```
$ tgrep2 -t "TOP << ADJP" | stripTGrep2Terminals.pl
```

The following options are not supported: **-d**, **--default**, **-f**,

4.3.2 importVariable.pl

This script lets you import variables from other files by matching the row IDs of the target database and the input file. There are several ways to do this, but in each case the target database must be specified by using the **-c** or **-d** option. In addition, the first column of each input file is assumed to contain IDs (to match with the target database).

- Use the **--files** options to pass files containing variables to add to the database. If you specify only the file names, the second column (right after the ID column) will be imported. The name of the imported variable in the database will be the name of the input file. For example, the following will create two new variables **adjp.t2o** and **np.t2o** in the database bncw.tab.
  `$ importVariable.pl -d bncw.tab --files /home/adjp.t2o,/home/np.t2o`

- To specify the column to be imported from each file, use the **--cols** option and specify the number of the column to be imported. The column name in the database will be the same as the column name in the input file. For example, the following will create two new variables in the database bncw.tab that are imported from column **1** in **adjp.t2o** and column **3** in **np.t2o**.
  `$ importVariable.pl -c bncw --files adjp.t2o,np.t2o --cols 1,3`

- If you want to import more than one column from a given file, specify only one file with the **--files** option and all the columns you want to import from that file with the **--cols** option. For example, the following will import columns **2, 3**, and **5** from the file **adjp.t2o**.
  `$ importVariable.pl -d bncw.tab --files adjp.t2o --cols 2,3,5`

In addition, if you want to specify the names of the columns that are added to the database, you can do this with the **--colnames** options. For example, the following creates a column **ADJP** by importing column **Adjp1** from file **adjp.t2o**.
`$ importVariable.pl -d bncw.tab --files adjp.t2o --cols Adjp1 --colnames ADJP`
The following options are not supported: **--default**, **-f**

4.3.3 sampleDatabase.pl

This script draws a pseudo-random sample from your *corpus*.tab database and calls it *corpus*_sample.tab. If *corpus*_sample.tab already exists, the file name will be unknown_sample.tab. Use the **-c** or **-d** option to specify the database to draw the sample from. A second argument determines the sample size. For example, the following will draw a sample of 200 cases from the file bncw.tab and write it to bncw_sample.tab.

```
$ sampleDatabase.pl -c bncw 200
```

The following options are not supported: **--d**, **-f**, **--files**

A Sample options file

```
*************************************************************************
data=/home/projects/perspective/data
results=/home/projects/perspective/results

init ID

add StringVar Form=Form
add NodeVar POS
add LemmaVar Form
add StringVar Sentence=TOPstring
add CategoricalVar PPfrom=pp:PPafterNP
add InfoDensity NPpreceding
add Phonology Form
add CondProb Form
add CountVar CntPPfrom=PPFROMafterNP
add LengthVar LenPPfrom=PPFROMafterNP
*************************************************************************
```

B Sample `collectData` script

```
*************************************************************************
#!/bin/csh -f

cd /home/projects/perspective/results
setenv Pdata /home/projects/perspective/data/bncw
setenv Presults /home/projects/perspective/results
echo Creating new corpus file bncw.tab
        initDatabase.pl -roc bncw --files $Pdata/ID

echo Beginning data extraction...
echo
        addStringVar.pl -roc bncw -f Form=$Pdata/Form
        addNodeVar.pl -roc bncw -f POS=$Pdata/POS
        addLemma.pl -roc bncw -f Form=$Pdata/Form
        addStringVar.pl -roc bncw -f Sentence=$Pdata/TOPstring
        addCategoricalVar.pl -roc bncw -f PPfrom 1 $Pdata/PPFROMafterNP
        addInformationDensity.pl -roc bncw -f NPpreceding 3
        addPhonology.pl -roc bncw -f Form
        addConditionalProbability.pl -c bncw -f Form
        addCountVar.pl -roc bncw -f CntPPfrom=$Pdata/PPFROMafterNP
        addLengthVar.pl -roc bncw -f LenPPfrom=$Pdata/PPFROMafterNP

cd $TDTlite
*************************************************************************
```

References

Rohde, D. (2005). *TGrep2 Manual*. http://tedlab.mit.edu/∼dr/Tgrep2/tgrep2.pdf.