texrex

User's Manual

Roland Schäfer

(Freie Universität Berlin)

Release: texrex-neuedimensionen

Manual date: 2014/23/06

texrex is designed and used for the COW project
http://www.corporafromtheweb.org/

Contents

Author

The author of the software and the manual is Roland Schäfer of Freie Universität Berlin. Contact him by email: mail@rolandschaefer.net.

Acknowledgments

Development of texrex and the texrex data would have been impossible with the help/support of: Felix Bildhauer, Sarah Dietzfelbinger, Lea Helmers, Stefan Müller at Freie Universität Berlin. The GNU/Linux support group at the Zedat data center (Robert Schüttler, Holger Weiss, and others) greatly helped by maintaining the infrastructure for the project. Finally, the FreePascal developers (especially Florian Klämpfl and Jonas Maebe on compiler development) supply the world with free development tools without which the work would have taken much longer.

The name

Why is it called texrex? The answer is that it grew out of a very simple HTML stripper which I wrote called tex for text extractor. First of all, I noticed after a day that that name was taken. Furthermore, I added boilerplate and text quality detection capabilities, etc. Thus, it is now rather a **tex**t **r**ecognizer and **ex**tractor, or texrex. All other names of tools (like tender) were derived by shortening texrex, then adding more acronym letters.

The current release name neuedimensionen is a reference to Techno Bert's classic *Neue Dimensionen*, released on vinyl in 1990, when I was still a child.[1] At the time, I was persistently failing in my attempts to write an open-source database application for the Amiga OS while teaching myself OOP.

Background

The research behind texrex was described in **?**Schäfer et al. (2013); **?**. The general methodology is also described in Schäfer and Bildhauer (2013).

[1] http://www.discogs.com/release/57994

1 Purpose

The primary purpose of the `texrex` tools is to create ad-hoc text-only corpora from large crawl archives, such as Heritrix ARC files. It fills the gap between the crawler and the tools for the linguistic processing of the corpus (tokenizer, tagger, lemmatizer, etc.).

First (with `texrex`):

1. strip HTML,
2. convert all documents to UTF-8 (using ICU),
3. convert all kinds of HTML entities to UTF-8 encoded unicode codepoints,
4. insert "paragraph" boundaries (actually called "divisions" or `<div>`) into the output,
5. detect paragraphs which do not contain usable text but just boilerplate material (like menus, tag clouds, copyright notices, etc.),
6. filter documents which are not written in the target language by examining the distribution of the most frequent words in the document (some pregenerated profiles included),
7. detect perfect duplicates by a simple fingerprinting algorithm
8. check UTF-8 well-formedness,
9. apply normalization by customizable replacement tables,
10. generate w-shingle-based fingerprints of all documents,
11. extract out links from web pages for graph analysis,
12. lookup IP geolocation information.

Then (with `tender` and `tecl`):

1. calculate document similarities based on the w-shingle fingerprints,
2. remove near-duplicate documents from the corpus.

As opposed to older versions, `texrex` ships with an ARC file reader. In order to read other formats, other reader classes have to be implemented. This should take no more than one working day per reader.

Additional tools included are:

1. **`tenet`**, a FANN neural network trainer,
2. **`HyDRA`** (**Hy**phenation **D**etection and **R**emoval **A**pplication), a tool to detect hard-hyphenation and fix it, based on unigram frequencies. Specially adapted for German,
3. **`rofl`** (**r**un-**o**n **f**ixer **l**ab), a tool to fix run-together sentences based on word lists.

2 Compilation and installation

You can do a fresh compile with the FreePascal compiler (FPC), version 2.6.0 or above, which itself requires no setup beyond the automated installation procedures.[2] However, binary packages might be provided for your platform.

2.1 Prerequisites

The following libraries/tools have to be installed.

1. FreePascal 2.6.0 or above (and its dependencies) – This is required **only for compilation**.
2. libfann 2.1 or above (binaries)
3. libicu 4.x or above (binaries) – If you use binary releases, the version number must match **exactly**, and creating "versioned" symbolic links will not suffice.

Notice that `texrex` comes with its own headers/bindings for those libraries, such that *dev* (= header) packages are not required for compilation.

[2] http://www.freepascal.org/

2.2 Compilation

Current distribution files can be obtained from the `texrex` web site.[3] First, unpack the source package. Makefiles are generated with `fpcmake`. Normally, just go to the root folder of the distribution package (where `Makefile` and `Makefile.fpc` are) and type:

```
make clean all
```

If that does not work, the `Makefile` probably needs to be refreshed, which works only with `fpcmake` installed. `fpcmake` requires the `fpc` sources and an environment variable `FPCDIR` pointing to the sources. In this case, type:

```
fpcmake && make clean all
```

2.3 Installation

2.3.1 After Compilation

Installation after compilation should work on all platforms by typing, given you have the appropriate rights:

```
make install
```

2.3.2 From binary release

You only have to make sure the binaries are in your `PATH`, or that you specify the path to them fully. There is no installation procedure as such.

2.3.3 Additional settings and the data files

In addition, you can also create an environment variable `TEXREXDATA`, which must point to a directory containing FANN network files (Section 3.3.10), language profiles (Section 3.3.12), normalizer rule files (Section 3.3.14), and the IP geolocation database (Section 3.3.17). If such a folder is specified by the `TEXREXDATA` variable (and if it exists), the `texrex` tools look for such files relative to this directory first. If it is not specified or the files are not there, the programs will look for them relative to the execution directory, unless an absolute path is given. Note that some distribution packages contains language profiles and network files in the sub-folder `data`. There are also binary-only packages, for which all data files need to be downloaded separately from the SourceForge site. **The large IP geolocation database must be downloaded separately from the Maxmind homepage.**

[3] http://sourceforge.net/projects/texrex/

2.3.4 Compiling FANN network files

The `texrex` distribution contains `.net` FANN multi-layer perceptron networks to be used with the deboilerplater (3.3.10). If they do not work (for example because the installed FANN version does not match exactly the one used to compile the distribution), then the `.dat` files have to be used to train FANN networks. The aforementioned `data` folder contains some pre-packaged `.dat` and `.net` files. To train networks, execute the following command (or similar, depending on the choice of `.dat` file) there:

```
tenet boilerplate.iso.dat boilerplate.iso.net
```

This will compile the FANN network file in format compatible to your version of the FANN library.

3 Using `texrex`

3.1 Basic architecture

`texrex` is designed for powerful single machines. More cores mean better performance. Hard disk performance becomes critical only if you have a lot of cores and want to use them. RAM requirements are modest, especially with deduplication turned off. Deduplication uses a scaling Bloom filter, which will roughly use 100 MB of RAM for 20 million adds at very low desired error rates such as 10^{-6}.[4]

`texrex` uses three thread "pools", which are technically speaking not true pools but rather simple collections of minimally managed threads. The threads interact with two queues. The overall data structure which is exchanged between threads via queues are of class type `TTrDocument` defined in `trdata.pas`.

Queues

1. The `InQueue` receives virtually unprocessed documents with a buffer containing the raw HTML data of a web page from the reader threads. They are then popped by worker threads.

2. The `OutQueue` receives fully processed documents from the workers. They are then popped by writer threads.

[4] Cf. Bloom (1970); Broder and Mitzenmacher (2004); Almeida et al. (2007). Scaling Bloom filters are slightly slower than static Bloom filters, but they do not require a priori memory allocation according to an expected number of members and a desired error rate.

Thread collections

1. The reader threads read ARC files and split them into documents, buffering the raw HTML in a `TTrDocument` object. They also extract some meta information (Last-Modified, crawl time, position in the ARC file, etc.)

2. The worker threads strip HTML and perform all the other processing to produce a clean corpus document.

3. The writer threads format the processed documents into XML. They also write shingles and links to separate files.

A single logger thread is also instantiated, which periodically reads, analyzes and logs (to a separate file) the activity of `texrex`.

3.2 Running `texrex`

Running a `texrex` job is very simple. An INI file (say, `job.ini`) specifying the job options is required (cf. 3.3), and `texrex` can then be run using any of these two:

```
texrex -j job.ini
texrex --job=job.ini
```

If the configuration is OK, `texrex` will run until the input is exhausted or the user shuts it down. Minimal startup information is printed on `stdout`. Minimal information about threads and queues is printed continuously, unless silent mode is activated (cf. 3.3).

Shutdown and other interactions with the running process can and should be achieved by using `texcomm` (cf. 3.4). Specifically, shutting down by pressing Ctrl+C is neither possible nor recommended. Also, using `kill` or similar commands to shutdown `texrex` is usually not necessary, because all processing is done in separate threads, and `texcomm` runs in the main thread, almost always allowing for graceful termination (unless you encounter a bug, of course).

If the `[TTrApplication] Debug` option is set to `1`, then `texrex` emits warnings about exceptions on `stderr`. If you want to catch this information, it might be best to start the program like this (routing `stderr` to a file, here `debug.log`), so `stderr` messages do not interfere with console operation:

```
texrex -j job.ini 2> debug.log
```

3.3 Job configuration

`texrex` jobs are configured via an INI file with sections and options. Each section corresponds to a Pascal class instantiated by `texrex`, and each option specifies a value for a published property of this class. Values are given after an equals sign (=). Booleans are specified as `0` for *false* or `1` for *true*, and strings should be quoted, like so:

```
JobName="mywebcorpus"
```

The options are as follows, organized by sections.

3.3.1 `[TTrApplication]`

This section configures the core parameters of the application.

• `JobName=<STRING>`

Name for the processing job, which can be chosen freely. The log file will use this as a prefix.

• `Comment=<STRING>`

A comment to be chosen freely. Use it to remind yourself what this job was for, for example.

• `Silent=<BOOLEAN>`

If `0`, `texrex` displays startup messages and later the queue fill status and the thread numbers continuously while the job is running. If `1`, it does a silent run with no output on `stdout` except error messages if command line options are incorrect.

• `InQSize=<INTEGER>`

Maximal number of documents (capacity) which the queue between the reader threads and the worker threads can hold. Since memory requirements are modest, and since reader threads read faster than worker threads can process them, this can usually be quite a high value, say 100,000 (but specify it without the thousand separator).

• `OutQSize=<INTEGER>`

Maximal number of documents (capacity) which the queue between the worker threads and the writer threads can hold. Since writers are much faster than workers, this queue will ususally not fill above 100.

• `WorkerManagement=<BOOLEAN>`

If `1`, the number of worker threads is controlled automatically, depending on the fill status of the InQueue. This is a very simple mechanism which increases/ reduces the worker thread count to keep the InQueue within a certain quartile of its capacity.

• `ManagementInterval=<INTEGER>`

The queue management (if active) checks the fill status of the InQueue at certain intervals. This option specifies the interval in seconds. Ineffective if worker management is off.

• `PreferedInQueueQuartile=<1|2|3|4>`

The quartile of the capacity of the InQueue within which the worker management should try to keep the fill status. Ineffective if worker management is off.

• `StatsInterval=<INTEGER>`

The interval in seconds at which the status of the system should be written to the log file. For huge production runs, anything below 60 should not be very informative.

• `Debug=<BOOLEAN>`

If `1`, then exceptions and other serious errors/warnings are reported on `stderr`.

3.3.2 `[TTrArcReaderPool]`

This section configures the thread collection which reads the raw ARC data from ARC files. Reading gzipped files happens automatically. With the original Heritrix gzip format, you need to specify an external gzip path, cf. below.

• `ExternalGzipPath=<STRING>`

The exact path to your `gzip` executable. If set, the specified external `gzip` is used instead of the internal gzip decompression. It must accept the `-c -d` option setting for piping uncompressed output to texrex. When Heritrix ARC files are processed, this is recommended, because the internal code cannot deal with multi-record gzip files as produced by Heritrix.

• `FileName=<STRING>`

The file name mask of the input ARC files. Use wildcards to specify as many ARC files as you want. If you specify a directory, all files in that directory will be used.

• `MinDocSize=<INTEGER>`

The minimal raw HTML document size in KB below which documents are discarded right away. Many crawled documents, like soft 404s, are too small to be of any value.

• `MaxDocSize=<INTEGER>`

The maximal raw HTML document size above which documents are discarded right away. You should configure your crawler to filter documents above, say, 256 or 512 KB. If you have **not** done that, use this option. Even 100 MB gzipped ARC files can contain documents several unzipped GB large, and such documents really do exist.

• `ReaderNumber=<INTEGER>`

The number of reader threads to be instantiated. Usually, a single reader can feed more than ten workers without queue starvation.

• `DocumentBufferSize=<INTEGER>`

To minimize push/pop collisions on the queue, reader threads buffer a number of documents before they push all of them onto the queue. Use this to set this number.

• `RetryWait=<INTEGER>`

How long (in milliseconds) a reader waits after a collision before trying to push again.

- `CrawlHeaderExtract=<STRING>`

In the ARC file format, the HTML dump of the crawled page is preceded by HTTP headers. This allows you to extract values of such headers. Each desired header name (case-insensitive) is given, separated by a pipe |. To extract *Last-Modified* and *Date*, for example, specify this:

```
CrawlHeaderExtract="Last-Modified|Date"
```

The headers are stored as meta values and can be written by properly specifying `WriteDocAttr` and `WriteDocMeta` later in the `TTrWriterPool` section.

3.3.3 `[TTrWorkerPool]`

This section configures the processing chain and the worker threads which perform the processing. The first block switches certain processors on or off. If this leads to an impossible configuration (because some processors require others to operate first in order to work properly), this will currently only manifest itself when single documents hit the respective processor, emitting numerous "precondition not met" messages on `stderr`. Always test your configuration on small sample files.

- `UseDuplicateDetector=<BOOLEAN>`
- `UseSimpleFilter=<BOOLEAN>`
- `UseUtf8validator=<BOOLEAN>`
- `UseDeboilerplater=<BOOLEAN>`
- `UseTextAssessment=<BOOLEAN>`
- `UseShingler=<BOOLEAN>`
- `UseNormalizer=<BOOLEAN>`
- `UseDivDeduplicator=<BOOLEAN>`
- `UseMetaExtractor=<BOOLEAN>`
- `UseGeolocator=<BOOLEAN>`

All the above activate the respective processor if set to `1`.

- `GeoBlocksFile=<STRING>`

Name of the IP geolocation blocks database. The file can be located relative to the `TEXREXDATA` directory, relative to the execution directory, or it has to be specified in the form of an absolute path name. Loading it will take a few seconds on startup. Ignored if `UseGeolocator=0`.
It is no longer included, because we can now read the original file format. Get the files directly from

http://dev.maxmind.com/geoip/legacy/geolite/

You need the **Legacy GeoLite City** database, and the file to be specified here is usually called *GeoLiteCity-Blocks.csv*. Unpack it and **convert it from ISO-8859-1 to UTF-8**, for example with `iconv`.

• `GeoLocationsFile=<STRING>`

Name of the IP locations database. The file can be located relative to the `TEXREXDATA` directory, relative to the execution directory, or it has to be specified in the form of an absolute path name. Loading it will take a few seconds on startup. Ignored if `UseGeolocator=0`.
It is no longer included, because we can now read the original file format. Get the files directly from

http://dev.maxmind.com/geoip/legacy/geolite/

You need the **Legacy GeoLite City** database, and the file to be specified here is usually called *GeoLiteCity-Location.csv*. Unpack it and **convert it from ISO-8859-1 to UTF-8**, for example with `iconv`.

• `WorkerNumber=<INTEGER>`

The number of worker threads. Memory is usually not an issue, so at least one thread per physical core is reasonable. The whole system performances hinges mainly on this setting. The more, the merrier.

• `MaxWorkerNumber=<INTEGER>`

If thread management is on, this specifies the maximal number of threads `texrex` will ever run. It cannot be overridden at runtime, neither by worker management, nor from the `texcomm` console (even in god mode).

• `MinWorkerNumber=<INTEGER>`

If thread management is on, this specifies the minimal number of threads `texrex` will ever run after the job was started. It cannot be overridden at runtime, neither by worker management, nor from the `texcomm` console.

• `BufferSize=<INTEGER>`

Same as `BufferSize` for the `TTrArcReaderPool`.

• `PopSleep=<INTEGER>`

The number of milliseconds a worker thread waits if popping from the InQueue resulted in a collision.

• `PushSleep=<INTEGER>`

The number of milliseconds a worker thread waits if pushing to the OutQueue resulted in a collision.

• `PushLimit=<INTEGER>`

The number of push retries before the thread gives up. This should never happen, and this is a debugging option which you should leave alone or set to extremely high values, like 999999999.

• `BloomErrorRate=<REAL>`

The worker pool initializes the Scaling Bloom filter for the deduplicator. This specifies the desired error (false positive) rate, usually something like 0.000001.

3.3.4 `[TTrWriterPool]`

This section configures the writer threads and defines the form of the XML output of the final corpus documents.

• `WriterNumber=<INTEGER>`

The number of writer threads. Usually, one writer can handle the output of at least 10 worker threads.

• PopSleep=<INTEGER>

The number of milliseconds a writer thread waits if popping from the OutQueue resulted in a collision.

• BufferSize=<INTEGER>

Same as BufferSize for TTrArcReaderPool.

• StrictXml=<BOOLEAN>

If this is 1, then the writer will replace protected XML characters with entities (i. e., " becomes ", etc.). This is highly recommended if well-formed XML is required.

• XmlHeader=<BOOLEAN>

If 1, the writer writes a standard compliant XML header for each output file (including an <xml></xml> container). This causes problems with certain uses of tender and tecl and should not be used, if possible.

• Prefix=<STRING>

Output file name prefix. **No** wildcards.

• WriteDocMeta=<STRING>

A pipe-separated list of meta information that should be written after the <doc> tag in the form of <meta name="" value=""> tags. Currently, texrex gathers the following meta information:

```
arcfile|arcoffset|arclength|mime|size
```

If TARC files are written, then

```
tarcfile|tarcheaderoffset|tarcbodyoffset|
    tarcheaderclength|tarcbodylength
```

are available for indexing the TARC files (cf. TTrWriterPool). TARC files are like Heritrix ARC files without the header, i. e., they contain only single ARC file records. Also, HTML is dumped in one line, and all line feeds from the original HTML document are lost.

If the MetaExtractor is switched on, then

```
title|keywords
```

are available if configured (cf. `TTrMetaExtractor`).
If the Geolocator is switched on, then

```
country|region|city
```

are available if configured (cf. `TTrGeolocator`). The meta tag is only written if the respective information is stored for the document.

• `WriteDocAttr=<STRING>`

Like `WriteDivMeta`, but the information is written as attributes to the `<doc>` tag in the form `name="value"`. It is always written, and if the respective meta value is not set for a document, `_unk_` is inserted.

• `WriteDivMeta=<STRING>`

Same as `WriteDocMeta` for divisions. Currently unused.

• `WriteDivAttr=<STRING>`

Same as `WriteDocAttr` for divisions. Currently unused.

• `WriteText=<BOOLEAN>`

If `1`, `texrex` writes the actual stripped text of the divisions. Setting this to `0` only makes sense for debugging purposes.

• `WriteBpc=<BOOLEAN>`

If `1`, the boilerplate class is written to the `<div>` tag as an attribute `bpc="x"`, where `x` is a letter from `a`, which means that the boilerplate level is in $(-1, 0)$ to `k`, which means that the boilerplate level is in $(0.9, 1)$.

• `WriteBpv=<BOOLEAN>`

If `1`, `texrex` writes the boilerplate value calculated by the MLP to the `<div>` tag as an attribute `bpv="r"`, where `r` is a real.

• `WriteBdc=<BOOLEAN>`

If 1, the badness class is written to the <doc> tag as bdc="x", where x is a letter from a to the n-th letter of the alphabet, and the corresponding badness score is in $(n \times 2 - 2, n \times 2]$.

• WriteBdv=<BOOLEAN>

If 1, the badness score is written to the <doc> tag as bpv="r", where r is a real.

• WriteNonBoilerplateDivCount=<BOOLEAN>

If 1, the count of non-boilerplate in the document is written to the <doc> tag as nbd="i", where i is an integer.

• WriteNonBoilerplateCharCount=<BOOLEAN>

If 1, the count of non-boilerplate characters is written to the <doc> tag as npc="i", where i is an integer.

• WriteNonBoilerplateDivProportion=<BOOLEAN>

If 1, the proportion of non-boilerplate in the document is written to the <doc> tag as nbdprop="r", where r is a real.

• WriteNonBoilerplateCharProportion=<BOOLEAN>

If 1, the proportion of non-boilerplate characters is written to the <doc> tag as nbcprop="r", where r is a real.

• WriteAverageBoilerplateCharacter=<BOOLEAN>

If 1, the average boilerplate value of a character is written to the <doc> tag as avbpc="r", where r is a real.

• WriteAverageBoilerplateDiv=<BOOLEAN>

If 1, the average boilerplate value of a is written to the <doc> tag as avbpd="r", where r is a real.

• WriteDups=<BOOLEAN>

If 1, then duplicate paragraphs are actually written as <dup>, otherwise they are silently deleted. This is ineffective if TTrDuplicateDetector is deactivated.

• DupBlank=<STRING>

If your indexing software needs a token in order to create a region (in order to represent duplicate paragraphs), this is what will be inserted as a dummy token instead of the text of duplicate divisions. If you set it to dupblank, for example, then paragraph (= division) 23 will look like this if it is a duplicate of division 11:

```
<div id="23" dup_of="11">dupblank</div>
```

For example, if you switch on the duplicate paragraph detector, set WriteDups=1, use the IMS OCWB to index your corpora, **and** you want to see the duplicate paragraph positions in your indexed corpus, you have to specify a DupBlank.

• WriteDivMetrics=<BOOLEAN>

If 1, texrex writes the boilerplate metrics for each as a <metrics> tag. Useful if you want to generate training data for tenet (cf. 3.5).

• WriteShingles=<BOOLEAN>

If set to 1, then shingles will be written to shingle files.

• WriteLinks=<BOOLEAN>

If set to 1, then files will be created logging each http out link from processed pages. The files have the form of a four-column tab-separated file. The columns are:

1. linking page
2. linked page
3. badness of linking page
4. boilerplate value of where the link was found

• WriteTokens=<BOOLEAN>

If a tokenizer is activated, and this is `1`, then separate files are written containing the token count profile for each document. You need this to create language profiles for the `TTrTextAssessment` processor.

• `WriteMaxTokens=<INTEGER>`

This defines how many type-token counts are written. Only the top *n* types are written, where *n* is the integer set here.

• `WriteTarc=<BOOLEAN>`

If `1`, then TARC files will be written containing the HTML dumps of those documents which are also written (in stripped form) to the XML corpus files.

• `SplitSizeXml=<INTEGER>`

Set this to the number of uncompressed MB after which a corpus file should be split.

• `SplitSizeShingles=<INTEGER>`

Set this to the number of uncompressed MB after which a shingle file should be split.

• `SplitSizeLinks=<INTEGER>`

Set this to the number of uncompressed MB after which a link file should be split.

• `SplitSizeTokens=<INTEGER>`

Set this to the number of uncompressed MB after which a token file should be split.

• `SplitSizeTarc=<INTEGER>`

Set this to the number of uncompressed MB after which a TARC file should be split.

• `GzipXml=<BOOLEAN>`

If 1, compress corpus files with gzip.

- GzipShingles=<BOOLEAN>

  If 1, compress shingle files with gzip.

- GzipLinks=<BOOLEAN>

  If 1, compress link files with gzip.

- GzipTokens=<BOOLEAN>

  If 1, compress token files with gzip.

- GzipTarc=<BOOLEAN>

  If 1, compress TARC files with gzip.

3.3.5 [TTrDuplicateDetector]

The duplicate detector detects perfect duplicates by looking at the raw HTML.

- FingerprintSize=<INTEGER>

  Not the full document is hashed, but a string of FingerprintSize bytes, taken with even distances from the raw HTML.

3.3.6 [TTrHtmlStripper]

The stripper removes markup from the raw HTML, performs splitting into divisions, and records a number of metrics for each division.

- DebugParse=<BOOLEAN>

  Emit parsing debug information (huge amounts!) on stdout. Use with Silent=1 and pipe the output to a file. I cannot imagine anyone but the texrex programmers needing this kind of information. The DEBUGPARSE symbol must be defined at compile time, or this won't be available.

• `ExtractAnchors=<BOOLEAN>`

If `1`, links are extracted from HTML documents.

• `KeepSameHostLinks=<BOOLEAN>`

If `1`, links to the same host are kept, i. e., links where the `vhost.host.tld` parts are identical between the addresses of the linking and the linked document.

• `KeepSameVirtualHostLinks=<BOOLEAN>`

If `1`, links to the same virtual host are kept, i. e., links where `host.tld` parts are identical between the addresses of the linking and the linked document.
This does not include `KeepSameHostLinks`.

• `KeepExternalLinks=<BOOLEAN>`

If `1`, links to other hosts are kept.

• `MinimalLinkLength=<BOOLEAN>`

How long (in characters) a link must be in order to be kept.

• `MaximalLinkLength=<BOOLEAN>`

How long (in characters) a may may be in order to be kept.

3.3.7 `[TTrCharsetConverter]`

This component uses ICU to detect encodings and convert them all to UTF-8.

• `Iso88591IsWin1252=<BOOLEAN>`

Set this to 1 in order to treat all documents declared as ISO-8859-1 as Window-1252 (also known as Latin1 or ANSI). This is basically a very good idea, and the HTML5 standard even requires this behavior.

3.3.8 [TTrSimpleDocumentFilter]

This filter invalidates documents which are too short after HTML stripping and conversion.

- `DivThreshold=<INTEGER>`

  Discard documents with less than this number of divisions.

- `SizeThreshold=<INTEGER>`

  Discard documents with less than this number of UTF-8 characters.

3.3.9 [TTrSecondPass]

This processor cleanses problematic material from the division text. It is required for the deboilerplater!

- `CleanseTags=<BOOLEAN>`

  Set to `1` to remove all literal tags (highly recommended for clean XML output).

- `CleanseEmail=<BOOLEAN>`

  Set to `1` to replace all email addresses by `EmailReplacer`.

- `CleanseUri=<BOOLEAN>`

  Set to `1` to replace many common URIs by `UriReplacer`.

- `CleanseHashtag=<BOOLEAN>`

  Set to `1` to replace Twitter hashtags by `HashtagReplacer`.

- `EmailReplacer=<STRING>`

  Cf. `CleanseEmail`.

- `UriReplacer=<STRING>`

Cf. `CleanseUri`.

- `HashtagReplacer=<STRING>`

  Cf. `CleanseHashtag`.

3.3.10 `[TTrDeboilerplater]`

This configures the boilerplate detector based on a Multilayer Perceptron.

- `TrainingMode=<BOOLEAN>`

  If `1`, then the MLP is not actually called, and all divisions receive a score of `-1`. This is useful when training data are generated for compiling a new MLP. In this case, also setting `WriteDivMetrics` in `TTrWriterPool` is necessary to output the feature values for each paragraph.

- `FannFile=<STRING>`

  Specify the FANN compiled network file (absolute, relative to execution path, or relative to `TEXREXDATA`). If you get errors like *Error reading "connection_rate" from configuration file* or similar, you need to set `LC_ALL` to `C`. Under Bash, this is achieved by:

```
export LC_ALL=C
```

- `CustomRegex`

  If a division macthes this regex, it will be considered as boilerplate with a 1 boilerplate value. This is intended to mark "read more" divisions with cut off sentences at the end unequivocally as boilerplate.

- `Threshold=<REAL>`

  Set this threshold above which a division is marked as boilerplate.

- `MinDivsAboveThreshold=<INTEGER>`

  If the number of `<div>` in the document for which the boilerplate value is lower than the threshold is below this number, the document will be discarded.

• `MinDivProportionAboveThreshold=<REAL>`

If the proportion of `<div>` in the document for which the boilerplate value is lower than the threshold is below this number, the document will be discarded.

• `MinCharsAboveThreshold=<INTEGER>`

If the number of characters in the document for which the boilerplate value is lower than the threshold is below this number, the document will be discarded.

• `MinCharProportionAboveThreshold=<REAL>`

If the proportion of characters in the document for which the boilerplate value is lower than the threshold is below this number, the document will be discarded.

3.3.11 `[TTrUnicodeLetterRangeTokenizer]`

This processor tokenizes documents. It can only handle Latin alphabets which use blanks for word separation. Alternative tokenizers are being developed. The tokenization is used by `TTrTextAssessment` and `TTrShingler`.

• `MaxBoilerplate=<REAL>`

Only tokenize division which have a boilerplate value below this value. Keeps the tokenizer from tokenizing boilerplate.

• `MinLength=<INTEGER>`

Minimum length (in UTF-8 characters) which a division must have to be tokenized.

3.3.12 `[TTrTextAssessment]`

This processor evaluates the linguistic quality of the document and checks the target language as a by-product. Cf. Schäfer et al. (2013). Presupposes the activation of a tokenizer.

• `ProfileFile=<STRING>`

Set the name of the profile file. Profiles for some European languages are included in the data folder.

• `Threshold=<INTEGER>`

Badness threshold above which a document is invalidated. The paper recommends 35.

3.3.13 `[TTrShingler]`

This processor generates the shingles for near-duplicate detection. Presupposes the activation of a tokenizer.

• `NGramSize=<INTEGER>`

Shingles are essentially hashed token-n-grams. Set n here. Typically 5.

• `HashesNumber=<INTEGER>`

How many hash functions (= random permutations) to use. Typically 100 or 200, the more, the better the accuracy of the process.

3.3.14 `[TTrNormalizer]`

This processor allows users to define simple replacement rules, for example to map all UTF-8 quotes to the simple ". A demo replacement file is included. It contains comment lines beginning with # or replacement rule lines with `INPUT <TAB> OUTPUT` lines. There are no regex capabilities, rules are simple replacement rules. For example, a rule line like this:

`ä  ae`

(where the space between the letter groups is a literal tab) represents a context-free replacement rule $\ddot{a} \longrightarrow ae$ and simply replaces all `ä` characters by the sequence `ae`.

• `ReplacementFile=<STRING>`

The name of the file specifying the replacements (absolute, relative, or relative to `TEXREXDATA`).

3.3.15 [TTrDivDeduplicator]

This processor effectively detects and marks duplicate divisions in stripped documents.

- `CharacterThreshold=<INTEGER>`

  Look only at divisions which contain more UTF-8 characters than this threshold. It is usually useless to look at divisions with 3 or 5 characters.

3.3.16 [TTrMetaExatrctor]

This processor extracts the HTML document title and declared meta information. All meta-information is converted to UTF-8, second-pass cleansed, checked for UTF-8 validity and normalized just like paragraphs if the respective processors are switched on.

- `ExtractKeywords=<BOOLEAN>`

  If `1`, the HTML `<title>` element is extracted.

- `ExtractTitle=<BOOLEAN>`

  If `1`, the HTML `<meta name="keywords">` element is extracted.

3.3.17 [TTrGeolocator]

This processor looks up the IP address of the server serving a page in a geolocation database and adds meta information. Albeit unusual from a user's perspective, the file location of the geolocation database has to be specified in the `TTrWorkerPool` section (cf. 3.3.3) for technical reasons.

- `AddCountry=<BOOLEAN>`

  Set to `1` to lookup country information.

- `AddRegion=<BOOLEAN>`

  Set to `1` to lookup region information.

- `AddCity=<BOOLEAN>`

  Set to `1` to lookup city information.

3.4 texcomm

`texcomm` is a (stateless) protocol and a console implementation for controlling a running `texrex` system. If you have access to the running process, you can drop to `texcomm (core)` directly, but using the `texcomm` IPC client is recommended.

3.4.1 texcomm (core)

You can invoke `texcomm` on a running `texrex` process by pressing the `Z` key. Immediately, you drop to the console saying `texcomm $`. There is no pager functionality. You can control the running process using the commands explained by entering `h` or `help`. Currently, this is the list of commands (long and short) and their usage:

```
bye            b       Exit texcomm (does not shutdown texrex).
shutdown [M]   s [M]   Shutdown texrex. Pass magic number as M.
                       Without M, the magic number will be shown.
dash [force]   d [f]   Show dashboard. 'force' for fresh calculations.
peek           p       Show a processed recent document (plain text
                       prior to final normalization in XMLWriter).
reader +|-     r  +|-  Add (+) or remove (-) a reader thread.
worker +|-     wo +|-  Add (+) or remove (-) a worker thread.
writer +|-     wr +|-  Add (+) or remove (-) a writer thread.
inqueue N      iq N    Set 'in' queue size to N.
outqueue N     oq N    Set 'out' queue size to N.
manage         ma      Toggle dynamic worker management.
conf S         c       Print (original!) configuration section S.
                       Pass no parameter to see the list of sections.
ident          i       Identify this server.
silence        si      Toggle silent mode.
help           h       Show help.
```

To shutdown `texrex`, you need to enter `shutdown` once without an argument. This will give you a magic number (randomly selected at each `texrex` startup) which must be entered to actually shutdown the process. Since the `texcomm` protocol is stateless, there will be no further confirmation. Entering `shutdown` with the correct magic number shuts down `texrex` for good.

3.4.2 texcomm (IPC)

The IPC client supports exactly the same commands, and it connects to the server process via a platform-independent FreePascal implementation of inter-process communication. Once you start it by typing `texcomm`, a console comes up. You are not connected to a server (= running `texrex` instance), which you can do by entering `connect texrex N`, where `N` is the process ID of the server. On startup, `texrex` emits this number, or you can get it from `texcomm (core)` via the `ident` command.

Using the `texcomm (IPC)` instead of `texcomm (core)` is highly recommended. First of all, as long as `texcomm (core)` is open, no IPC client can connect. Secondly and more importantly, thread management is disabled while `texcomm (core)` is running, because they are both executed in the main thread. This is intended behavior, but for most uses, it is not desirable.

3.5 tenet

`tenet` is takes FANN training data and compiles a FANN network file. `tenet` is controlled by command line switches and can also be used as a simple general FANN network creator, although there are much more versatile general-purpose options available. `tenet` was rewritten from the earlier `texnet` tool, and it now behaves as all other command-line tools from the suite.

Reasonable defaults are used if no options are given, like so:

```
tenet -i INPUT -o OUTPUT
```

This will train and save a multi-layer perceptron network with the default options which worked best in our test runs.[5] Either the output file must not exist, or the `-e` option must be given. The defaults can be changed with the following command line options.

-i|--input

`-i FILENAME | --input=FILENAME`

Input file name. The file must exist.

-o|--output

`-o FILENAME | --output=FILENAME`

Output file name. The file must not exist, or use `-e`.

-e|--erase

`-e | --erase`

If this option is given, and the output file exists, it is erased before a new network is trained.

-d|--desire

`-d REAL | --desire=REAL`

Attempt to reach an MSE of the specified value. If this MSE is reached before the maximum number of epochs was trained, training is stopped anyway.

-r|--report

`-r INTEGER | --report=INTEGER`

Sets the interval (in epochs) for reports on training progress on standard out.

-m|--maximum

`-m INTEGER | --maximum=INTEGER`

Train for maximally this number of epochs. If the desired MSE is reached before the maximum number of epochs was trained, training is stopped anyway.

-I|--innum

`-I INTEGER | --innum=INTEGER`

Specify the number of input values (as a convenience option if tenet is used as a general-purpose network trainer). Notice that this number must match exactly the actual number of input values encoded in the training data. This option should usually not be specified for `texrex` usage, since `texrex` generates a fixed number of values by hard-coded algorithms.

-O|--outnum

`-O INTEGER | --outnum=INTEGER`

Specify the number of output values (as a convenience option if tenet is used as a general-purpose network trainer). Notice that this number should be the actual number of output values encoded in the training data. This option should usually not be specified for `texrex` usage, since `texrex` generates a fixed number of values by hard-coded algorithms.

-1|--one

`-1 INTEGER | --one=INTEGER`

Specify the number of neurons on hidden layer 1. Set to `0` to deactivate hidden layer 1 and all subsequent hidden layers.

-2|--two

`-2 INTEGER | --two=INTEGER`

Specify the number of neurons on hidden layer 2. Set to `0` to deactivate hidden layer 2 and all subsequent hidden layers.

-3|--three

`-3 INTEGER | --three=INTEGER`

Specify the number of neurons on hidden layer 3. Set to `0` to deactivate hidden layer 3 and all subsequent hidden layers.

-4|--four

```
-4 INTEGER | --four=INTEGER
```

Specify the number of neurons on hidden layer 4. Set to `0` to deactivate hidden layer 4 and all subsequent hidden layers.

-5|--five

```
-5 INTEGER | --five=INTEGER
```

Specify the number of neurons on hidden layer 5. Set to `0` to deactivate hidden layer 5. You cannot have more than 5 hidden layers with tenet.

-t|--train

```
-t STRING | --train=STRING
```

Specify the C name of the training algorithm to be used.[6]

-H|--hidden

```
-H STRING | --hidden=STRING
```

Specify the C name of the hidden activation function.[6]

-a|--activ

```
-a STRING | --activ=STRING
```

Specify the C name of the output activation function.[6]

-w|--widrow

```
-w | -{-}widrow
```

If this option is given, the network weights are **not** pre-initialized with the Widrow & Nguyen algorithm.

-h|--help

```
-h | --help
```

Print help, do thing else.

[5]See http:http://leenissen.dk/fann/ for details of the FANN library and some artifical neural network theory.

[6]http://leenissen.dk/fann/html/files/fann_data-h.html

4 The shingling helper tools

Shingling is a method for removing near-duplicate documents from a collection of documents (**?**). In its original form, it is actually a way of clustering documents based on similarity, but the `texrex` tools use a simpler form (without clustering) for a simpler purpose. In essence, each document is tokenized, and then the set of unique token-`n`-grams of the document is formed. These `n`-grams are hashed with a 64-bit Rabin hash function (cf. Rabin (1981)), and from each of `m` random permutations of these hashes, the minimal hash is stored in the **fingerprint** of the document. Notice that `tender` does not use (pseudo-)random permutations of one hash, but different Rabin hash functions based on randomly selected irreducible polynomials over `GF(2)` of degree 64. If `100` random permutations (different Rabin hash functions) are used, then each fingerprint consists of `100` (minimal) hash values. Then, without actually having to compare each document with each other document pairwise (which is not feasible with even moderately large numbers of documents), the overlap between the fingerprints is calculated. Finally, documents with a higher than desired overlap are removed from the corpus. The process is explained in (Schäfer and Bildhauer, 2013, p. 58ff).

If configured properly, `texrex` generates the shingles in the normal cleaning process. Then, `tender` calculates shingle overlaps and `tecl` removes the documents from the corpus. Both `tender` and `tecl` read gzipped input transparently without special configuration.

4.1 `tender`

`tender` takes the shingles created by `texrex`, sorts them, creates doc-doc-pairs, sorts and counts those, and finally calculates similarities between documents, writing document IDs of duplicates to a blacklist file. It is a simple command line tool controlled by a number of flags/options.

Use `tender -h` or `tender --help` to see a short help and the default settings.

-i|--input

`-i FILE | --input=FILE`

Specify the input file name pattern (including wildcards), i. e., the shingle files created by `texrex`. If wildcards are used, the filename **must** be put in quotes.

-o|--output

`-o FILE | --output=FILE`

Specify the output file prefix (**no** wildcards). The names of the intermediate files and the blacklist files will begin with this prefix.

-b|--black

`-b FILE | --black=FILE`

Specify the blacklist file name pattern (including wildcards). Shingles from documents listed in any of the blacklist files will be completely ignored in this run (cf. 4.3). If wildcards are used, the filename **must** be put in quotes.

-g|--gzip

`-g | --gzip`

Use gzip compression for output files. This is not required for transparent gzip input processing, which uses automatic gzip file detection.

-t|--threads

`-t INTEGER | --threads=INTEGER`

The number of sorter threads to use. Sorting is done in memory, so each thread needs roughly twice as much memory as required to store `-s` shingle lines (cf. below), which take up 64 bytes each. Cf. below for the calculations.

-s|--size

`-s INTEGER | --size=INTEGER`

How many lines to sort in one thread/file in the divide-sort-merge sorting. Sorting is done in memory, so if you specify a size of 1,000,000 lines to sort, and each line is 64 bytes long, you need $2 * 10^6 * 64\ B \approx 122\ MB$ of RAM. This number "times" the number of threads roughly estimates your worst-case total memory needs.

-d|--ddsize

`-d INTEGER | --ddsize=INTEGER`

How many lines to send to a doc-doc pair creator thread. This is like `--size`, but the exact amount of memory needed is much harder to estimate, because it depends on the amount of duplication in the data. As a rule of thumb, set this to one third of `--size` for similar memory requirements if your corpus contains around 50% (near-)duplicates according to your similarity threshold.

-p|--presort

`-p | --presort`

If you use this, `tender` assumes (without checking!) that the input files are presorted, for example using `GNU sort`. You can even pre-merge input files to save time on the merge process.

-l|--limit

`-l INTEGER | --limit=INTEGER`

If two documents have more than this number of shingles in common, the shorter one will be blacklisted.

-m|--max

`-m INTEGER | --max=INTEGER`

A shingle occurring more than the number of times specified here will not be used at all for shingling. Because such shingles lead to a high number of doc-doc pairs being created, this must be considered a very effective performance hack which lowers the accuracy of the shingling method, however.

-f|--full

`-f | --full`

Write separate files which contain the doc-doc pairs and their calculated shingle overlap (= similarity). The files will also be gzipped if `-g` is set.

4.2 `tecl`

`tecl` erases or extracts documents with IDs specifiable in a blacklist or whitelist file from `texrex` corpus files, creating a "clean" output.

-b|--black

`-b FILE | --black=FILE`

Specify the blacklist files created by `tender`, possibly using wildcards. If wildcards are used, the filename **must** be put in quotes.

-i|--input

`-i FILE | --input=FILE`

Specify the corpus input files created by `texrex`, possibly using wildcards. Documents matching a blacklisted item will not be in the output. If wildcards are used, the filename **must** be put in quotes.

-o|--output

`-o FILE | --output=FILE`

Specify the output file name prefix (**no** wildcards).

-u|--uniqids

`-u | --uniqids`

Set this to make IDs unique by removing documents if their ID was already encountered. Technically, this will cause document IDs of good documents to be added to the blacklist once they have been written, such that they will be considered bad documents later. Notice that in releases from 2014 or later, document IDs are MD5 hashes of the URL (32 8-bit characters, 256 bits) plus a random suffix (a 16-bit integer in hexadecimal encoding), resulting in a 36 character hexadecimal string. Thus, even if you re-crawl URLs, documents virtually never have the same ID. But it's safer to use this option if you want to make absolutely sure that you never have identical IDs in your corpus. **Incompatible with `-w|-white` option.**

-s|--split

`-s INTEGER | --split=INTEGER`

Split the output files after INTEGER documents.

-g|--gzip

`-g | --gzip`

Use gzip compression for output files.

-w|--white

`-w | --white`

Interpret the blacklist as a whitelist. **Incompatible with `-u|-uniqids` option.**

4.3 Cyclic fingerprint overlap calculation

If you have a lot of shingles, it is more efficient to split the process into cycles and keep merging the results, using `tender`'s `-b|--black` option. The optimal strategy depends heavily on your system and has to be determined experimentally. Since `tender` uses a lot of memory in certain configurations, you might actually be **forced** to experiment with cyclic overlap calculation.

The method goes like this: First, you use `tender` several times to create a blacklist file from only a selection of all available shingle files at a time. Later, you run it again, specifying the previously created blacklists with the `-b` or `--black` option as well as **all** corresponding shingle files as input. All files which are already blacklisted will be ignored (= considered to be removed already) in the subsequent runs. In the end, use all blacklist files when you clean the corpus with `tecl` (Section 4.2). A simple session could look like this (using defaults where possible):

1. `tender -i shingles1.gz -o blacklist1 -g`

   (This creates `blacklist1.gz`.)

2. `tender -i shingles2.gz -o blacklist2 -g`

   (This creates `blacklist2.gz`.)

3. `tender -i "shingles*.gz" -o blacklist3 -b "blacklist*.gz" -g`

   (This creates `blacklist3.gz`.)

4. `tecl -b "blacklist*.gz" -i "corpus*.xml.gz" -o cleancorpus`

   (Note: For the final cleaning, you need to specify **all** blacklist files! `blacklist3.gz` is just an addendum to the previously created blacklists.)

5 Additional normalization tools

5.1 `HyDRA`

`HyDRA` removes hard-hyphenation (like *hy- phenation*) with high accuracy based on unigram frequencies. It does not rely on information about line endings, since they are usually not available in stripped web documents. Please use `hydra -h` to get help. You need language-specific unigram lists, which might be available from the web site.

5.2 `rofl`

`rofl` fixes run-together sentences (like *I am a sentence.This sequence is glued together.*) with high accuracy based on word lists. Please use `rofl -h` to get help. You need language-specific word lists, which might be available from the web site.

References

P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison. Scalable bloom filters. *Information Processing Letters*, 101:255–261, 2007.

B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, 13(7):422–426, 1970.

A. Z. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-CSE-03-01, Center for Research in Computing Technology, Harvard University, Harvard, 1981.

R. Schäfer and F. Bildhauer. *Web Corpus Construction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, San Francisco, 2013.

R. Schäfer, A. Barbaresi, and F. Bildhauer. The good, the bad, and the hazy: Design decisions in web corpus construction. In S. Evert, E. Stemle, and P. Rayson, editors, *Proceedings of the 8th Web as Corpus Workshop (WAC-8)*, pages 7–15, Lancaster, 2013. SIGWAC.