TRACER

TEXT REUSE DETECTION MACHINE

A User Manual

Greta Franzini, Emily Franzini, Kirill Bulert, Marco Büchler

2016

Contents

1 Introduction **5**
1.1 Text Reuse 5
1.2 Medusa 5
1.3 TRACER 5
1.4 TRACER's limitations 6
1.5 TRACER's status quo 6
1.6 Under the hood 6
1.7 TRACER publications 6

2 System prerequisites - before you start with TRACER **7**
2.1 Java 7
2.2 Text Editor 7

3 TRACER installation and configuration file **8**
3.1 Download TRACER 8
3.2 Download TRACER with Git (ALTERNATIVE) 8
3.3 Working with Git 8
3.3.1 Contributing 9
3.3.2 Reporting 9
3.4 TRACER's configuration file 9

4 Formatting a text to meet TRACER requirements **11**
4.1 Where to store your texts (Recommendation) 11
4.2 How to prepare your texts 11
4.2.1 Segmentation 11
4.2.2 Columns 11
4.2.3 Other relevant files 12

5 Preprocessing **14**
5.1 First run of TRACER 14
5.2 Results 14
5.2.1 How the results are organised 14
5.3 Types of preprocessing 15
5.3.1 Letter-level preprocessing 16
5.3.2 Word-level preprocessing 16
5.3.3 Results 16
5.3.4 Customising parameters or *properties* 17
5.4 Understanding preprocessing 18

6 Training/Featuring **20**
6.1 Types of featuring 20
6.1.1 Overlapping 20
6.1.2 Non-overlapping 21
6.1.3 Distance-based 21
6.1.4 Customising parameters or properties 21

7 Selection **25**
7.1 Selection strategies 25

8 Linking **27**
8.1 Types of Linking 27

9 Scoring **29**

10 Postprocessing: visualising the reuse results **30**
10.1 Download the visualisation package 30
10.2 Produce the visualisation 30
10.3 Updating the visualisations to modified parameters 33

About this manual

First edition: 2016
Written by: Greta Franzini, Emily Franzini, Kirill Bulert, Marco Büchler

This user manual describes the implementation of the TRACER machine, a powerful and flexible suite of some 700 algorithms for the automatic detection of (historical) Text Reuse. TRACER was developed by Marco Büchler and is written in Java. It is the most comprehensive tool yet and it is continuously improved thanks to the feedback gathered by the numerous tutorials and workshops given by the eTRAP (Electronic Text Reuse Acquisition Project) team at international conferences and events. For more information about the eTRAP Research Group, please visit: `http://www.etrap.eu`

About TRACER

TRACER Developer: Marco Büchler
TRACER homepage: `http://www.etrap.eu/research/tracer/`
TRACER repository: `http://vcs.etrap.eu/tracer-framework/tracer.git`
TRACER javadoc: `http://www.etrap.eu/hackathon/tracer/docu/javadoc/`
Medusa repository: `http://vcs.etrap.eu/tracer-framework/medusa.git`
Medusa javadoc: `http://www.etrap.eu/medusa/doc/javadoc/Medusa-2.0/`
TRACER bug reports: `http://www.etrap.eu/redmine/projects/tracer/`
TRACER presentation slides: `http://www.etrap.eu/tutorial/tracer/slides/`

Access to the `.git` repositories can be requested for free by writing to contact@etrap.eu

Copyright

Version history

Version 1.0 (2016-09-01): Initial version of the manual.

1 Introduction

1.1 Text Reuse

At its most basic level, Text Reuse is a form of text repetition or borrowing. Text Reuse can take the form of an allusion, a paraphrase or even a verbatim quotation, and occurs when one author borrows or reuses text from an earlier or contemporary author. The borrower, or *quoting* author, may wish to reproduce the text of the *quoted* author word-for-word or reformulate it completely. We call this form of borrowing "intentional" Text Reuse. "Unintentional" Text Reuse can be understood as an idiom or a winged word, whose origin is unknown and that has become part of common usage. Text Reuse detection on historical data is particularly challenging due to the fragmentary nature of the texts, the language evolution, as well as linguistic variants and copying errors.

1.2 Medusa

Medusa is an NLP framework for high performance computing of co-occurrences and n-grams of lengths between 2 and 10 [Büc08]. Furthermore, Medusa tokenises texts and created inverted lists both for words and co-occurrences. Medusa's speed is the result of a custom-built hash-system that brings together perfect and open hashing. *Perfect hashing* in terms of an Array hash is used for frequent co-occurrences or bigrams, while the open hashing is used for less frequent data by way of a bucket hashing. The RAM hash is split into separate buckets each containing individual data-stores. Once a buckets is completely filled the entire RAM hash is stored on disc and emptied in memory. After a complete run through the text the Array hash and all temporary hashes, written on disc, are merged into one persistent hash. With 1GB of memory Medusa reaches an average of about 2 accesses to the data structure to find the right location for the data-set. When increasing the memory this value can be further reduced. Medusa is used in TRACER for the tokenisation process (*Preprocessing*) and for all kinds of featuring techniques (*Featuring*). All files with `.wnc`, `.tok` and `.expo` suffixes are created by Medusa.

1.3 TRACER

TRACER is a suite of 700 algorithms, whose features can be combined to create the optimal model for detecting those words, sentences and ideas that have been reused across texts. Created by Marco Büchler, TRACER is designed to facilitate research in Text Reuse detection and many have made use of it to identify plagiarism in a text, as well as verbatim and near verbatim quotations, paraphrasing and even allusions. The thousands of feature combinations that TRACER supports allow to investigate not only contemporary texts, but also complex historical texts where reuse is harder to spot. TRACER is a command line engine. The reason it does not come with a user-interface is to boost the computing speed and performance. TRACER uses large and remotely accessible servers, which also facilitate the computation of large datasets. The reuse results, however, can be visualised in a more readable format via TRAViz (see section 10).

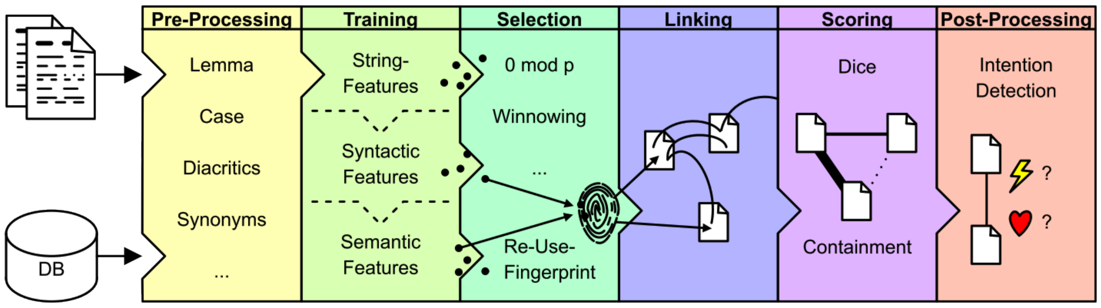


Figure 1: The six-step pipeline of TRACER.

1.4 TRACER's limitations

1.5 TRACER's status quo

TRACER has been successfully tested on English, German, Arabic, Hebrew, Tibetan, Ancient Greek, Latin, Coptic. We continuously seek new languages to work with. TRACER has been used on both historical and modern texts. A list of bug reports and current developments is available here.

1.6 Under the hood

This manual provides a user-friendly guide to Text Reuse detection for both novice and expert users. For this reason, it does not give an accurate description of the 700 algorithms constituting TRACER but only the necessary knowledge in order to perform simple Text Reuse detection tasks. Those who wish to study TRACER's algorithms in detail should navigate to this webpage.

1.7 TRACER publications

More about TRACER and its applications:

- Büchler, M., Franzini, G., Franzini, E. Bulert, K. (2016 forthcoming) 'TRACER - a multilevel framework for historical Text Reuse detection', *Journal of Data Mining and Digital Humanities - Special Issue on Computer-Aided Processing of Intertextuality in Ancient Languages*.
- Büchler, M., Burns, P. R., Müller, M., Franzini, E., Franzini, G. (2014) 'Towards a Historical Text Re-use Detection', In: Biemann, C. and Mehler, A. (eds.) *Text Mining, Theory and Applications of Natural Language Processing*. Springer International Publishing Switzerland.
- Büchler, M., Geßner, A., Berti, M. and Eckart, T. (2013) 'Measuring the Influence of a Work by Text Re-Use', In: Dunn, S. and Mahony, S. (eds.) *Digital Classicist Supplement: Bulletin of the Institute of Classical Studies*. Wiley-Blackwell.
- Büchler, M., Franzini, G., Franzini, E., Moritz, M. (2014) 'Scaling Historical Text Re-Use', In: (Proceedings) *The IEEE International Conference on Big Data 2014 (IEEE BigData 2014)*. Washington DC, October 2014, 27-30.
- Büchler, M. (2013) *Informationstechnische Aspekte des Historical Text Re-use* (English: Computational Aspects of Historical Text Re-use). PhD Thesis. Leipzig.

2 System prerequisites - before you start with TRACER

To use TRACER you must first ensure that you have version 8 of Java or a later version, and a text editor running on your computer.

2.1 Java

The precompiled version of TRACER requires version 8 or higher of Java. To check your Java version, open the terminal or command line[1] and type:

```
java -version
```

If the resulting message displays a version number below 8, a new Java package needs to be installed.[2] Download the package for your operating system, install, restart the computer, open the terminal and re-type the command above. Your version should have changed to 8.

If TRACER is downloaded from its GitLab repository (see section 3.2) and compiled *locally* the earlier Java 6 version is sufficient. Nevertheless, the use of the latest compiler, such as JDK 8 or later, is recommended.

2.2 Text Editor

`VIM` is the command line's inbuilt text editor and can be invoked with the `vim` command (under Linux and Mac). However, Windows machines might not recognise the `vim` command. If this happens, you can switch to a standalone text editor such as *Notepad++* or any other text editor of your choice. The best editor in terms of readability and performance is *Sublime Text*.

[1] A command line cheat-sheet describing basic commands is available here.

[2] These can be accessed here.

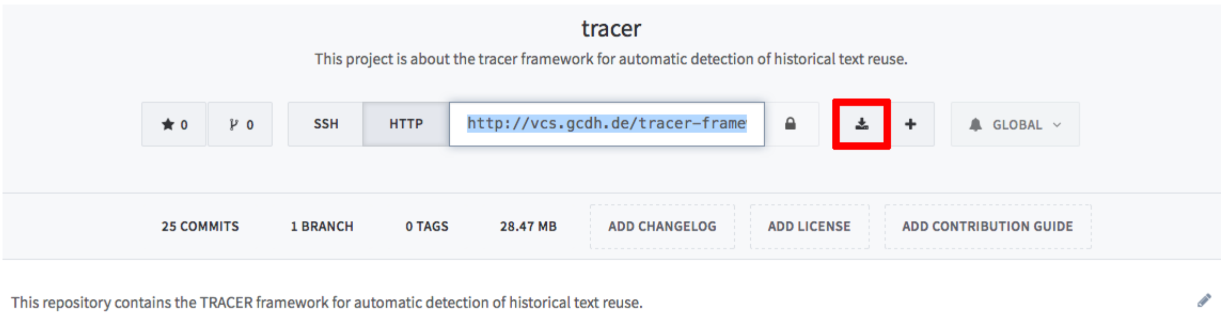
3 TRACER installation and configuration file

3.1 Download TRACER

Download and unzip TRACER from `http://www.etrap.eu/tracer/` to a folder on your computer. TRACER zipped is some 107MB in size, unzipped it's roughly 500MB. Additionally, in order to run TRACER needs between 5GB and 10GB of space. It is therefore important that you locate a folder on your computer with 10GB of space in which TRACER can be installed and computed.

3.2 Download TRACER with Git (ALTERNATIVE)

You can also clone most recent releases from our `git repository` [3]. Just be aware that even if we try our best, the most recent version might be unstable. Use only the release version for long-running tasks. Once you've obtained an account you can download the latest version here or by using git.


tracer

This project is about the tracer framework for automatic detection of historical text reuse.

This repository contains the TRACER framework for automatic detection of historical text reuse.

Figure 2: TRACER's GitLab repository.

3.3 Working with Git

Git is a so-called *version control system* that tracks the changes of specified files. It is heavily used by software developers and teams, who work simultaneously on the same project.

If you don't have Git already you can get a copy for your operating system from the official homepage. After installation you use multiple commands in you Terminal beginning with `git`, for example `git status`, which shows pending changes and suggests possible steps.

To get a copy of TRACER's repository you first need to clone it. This is done by typing:

```
git clone http://vcs.etrap.eu/tracer-framework/tracer.git
```

into your terminal.

You will be prompted for your username and password and a new folder will appear if successfully cloned. Before you can start work with TRACER you first have to build it from source. Change into the `tracer` directory and execute the command `ant .` If you get a `BUILD FAILED` some of the requirements are not met.

The recommended Git work flow is to first create a new branch by:

```
git checkout -b <your branchname>
```

You are now on a new local branch and all changes done to the tracked files will only remain in this local branch. You can get an overview of all branches by typing:

```
git branch -a
```

And you can switch branches the same way you already created one. The original branch is always called `master`. If you switch branches the tracked files will change according to your branch, while untracked files will remain in all branches. If you already build TRACER you can type `git status` and see that some untracked files where created by the build process. If you want to obtain the newest version of TRACER just switch to the master branch and type: `git pull` This will download the latest changes into your master branch. Remember to not work on the master branch unless you are familiar with Git.

[3] You can request an account at contact@etrap.eu

3.3.1 Contributing

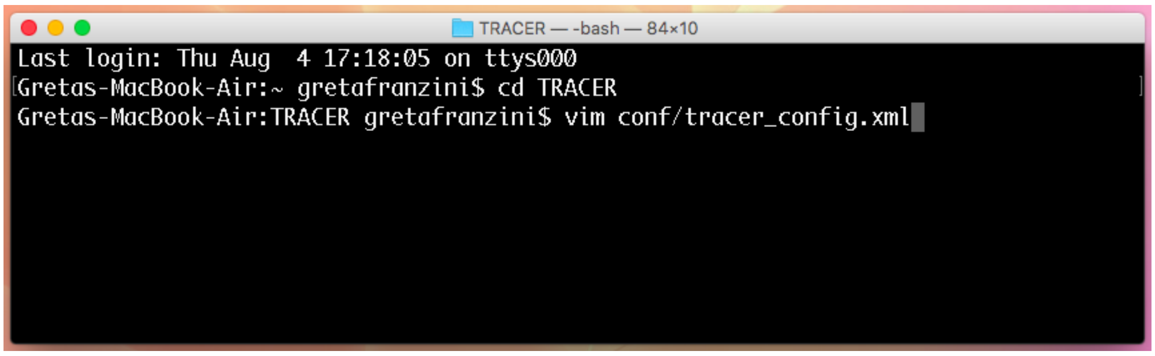
Before contributing please read the git commit conventions. With your account you also received the right to create your own branches and create merge requests on Gitlab. Please create a merge request if you want to contribute to TRACER.

3.3.2 Reporting

With your account you also have the ability to create new issues. If you encounter a bug or are in need of a feature please create a new issues. In case of a bug always provide your build number.

3.4 TRACER's configuration file

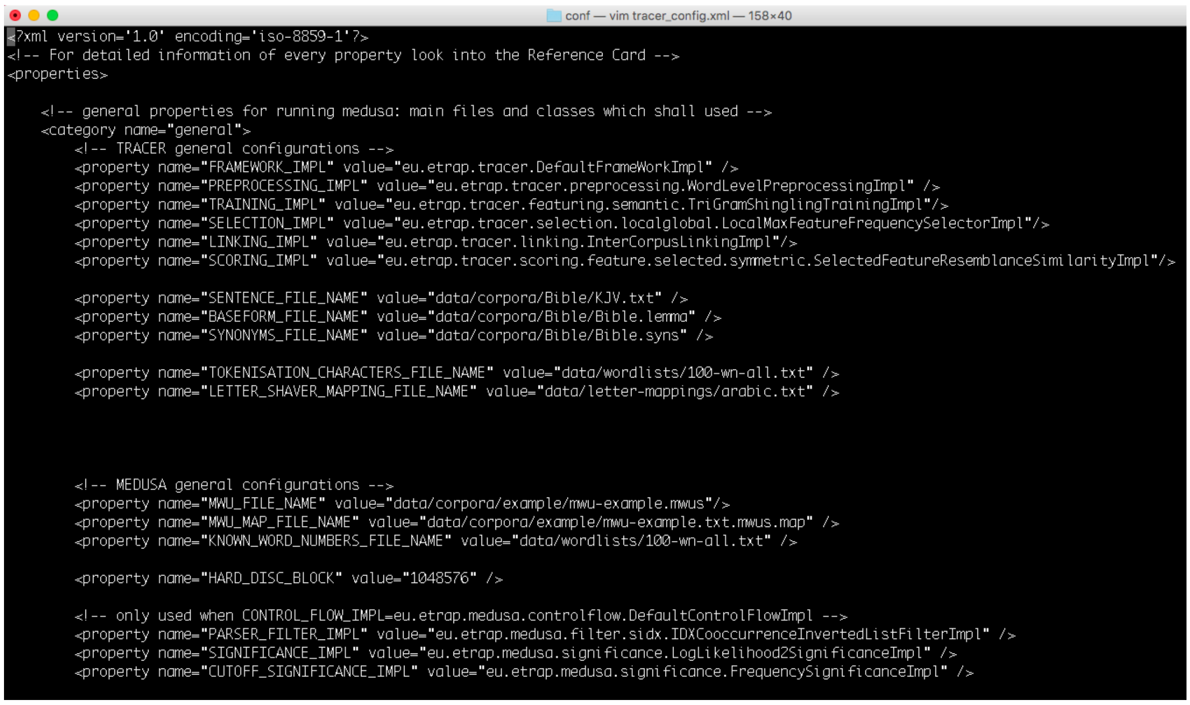
In the Terminal, open TRACER's configuration file `tracer_config.xml` by navigating to the TRACER folder (with `cd`) and typing `vim conf/tracer_config.xml`.



```
Last login: Thu Aug  4 17:18:05 on ttys000
Gretas-MacBook-Air:~ gretafranzini$ cd TRACER
Gretas-MacBook-Air:TRACER gretafranzini$ vim conf/tracer_config.xml
```

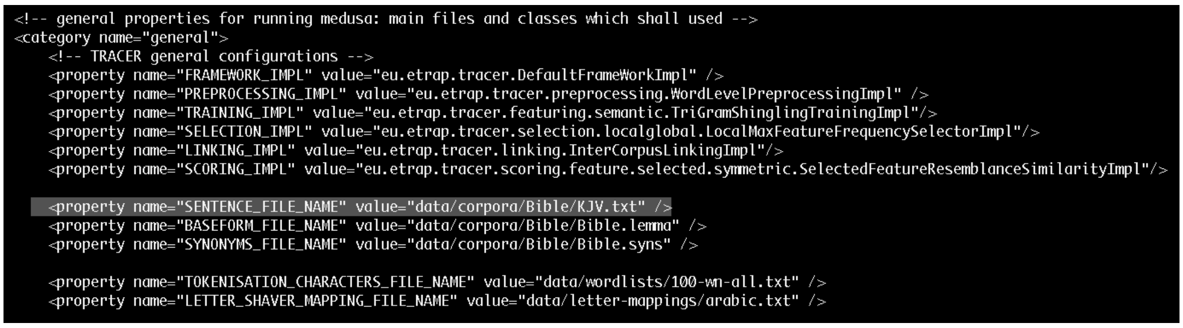
Figure 3: Open the terminal and navigate to the TRACER folder on your machine using the `cd` command. Then open TRACER's configuration file to define properties. The concept of 'defining properties' will become clear later in the manual (see section 5 onwards).

Press `ENTER` and you should now see the following screen:
To define the corpus you want TRACER to work with, locate the eighth property,
`<property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />`, in the same window:



```
<?xml version='1.0' encoding='iso-8859-1'?>
<!-- For detailed information of every property look into the Reference Card -->
<properties>

    <!-- general properties for running medusa: main files and classes which shall used -->
    <category name="general">
        <!-- TRACER general configurations -->
        <property name="FRAMEWORK_IMPL" value="eu.etrap.tracer.DefaultFrameWorkImpl" />
        <property name="PREPROCESSING_IMPL" value="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl" />
        <property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.semantic.TriGramShinglingTrainingImpl"/>
        <property name="SELECTION_IMPL" value="eu.etrap.tracer.selection.localglobal.LocalMaxFeatureFrequencySelectorImpl"/>
        <property name="LINKING_IMPL" value="eu.etrap.tracer.linking.InterCorpusLinkingImpl"/>
        <property name="SCORING_IMPL" value="eu.etrap.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl"/>

        <property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />
        <property name="BASEFORM_FILE_NAME" value="data/corpora/Bible/Bible.lemma" />
        <property name="SYNONYMS_FILE_NAME" value="data/corpora/Bible/Bible.syns" />

        <property name="TOKENISATION_CHARACTERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />
        <property name="LETTER_SHAVER_MAPPING_FILE_NAME" value="data/letter-mappings/arabic.txt" />



        <!-- MEDUSA general configurations -->
        <property name="MWU_FILE_NAME" value="data/corpora/example/mwu-example.mwus"/>
        <property name="MWU_MAP_FILE_NAME" value="data/corpora/example/mwu-example.txt.mwus.map" />
        <property name="KNOWN_WORD_NUMBERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />

        <property name="HARD_DISC_BLOCK" value="1048576" />

        <!-- only used when CONTROL_FLOW_IMPL=eu.etrap.medusa.controlflow.DefaultControlFlowImpl -->
        <property name="PARSER_FILTER_IMPL" value="eu.etrap.medusa.filter.sidx.IDXCooccurrenceInvertedListFilterImpl" />
        <property name="SIGNIFICANCE_IMPL" value="eu.etrap.medusa.significance.LogLikelihood2SignificanceImpl" />
        <property name="CUTOFF_SIGNIFICANCE_IMPL" value="eu.etrap.medusa.significance.FrequencySignificanceImpl" />
```

*Figure 4: After pressing **ENTER**, you should see this screen. This is the configuration file of TRACER, listing all of the properties with which you'll be working.*



```
<!-- general properties for running medusa: main files and classes which shall used -->
<category name="general">
    <!-- TRACER general configurations -->
    <property name="FRAMEWORK_IMPL" value="eu.etrap.tracer.DefaultFrameWorkImpl" />
    <property name="PREPROCESSING_IMPL" value="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl" />
    <property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.semantic.TriGramShinglingTrainingImpl"/>
    <property name="SELECTION_IMPL" value="eu.etrap.tracer.selection.localglobal.LocalMaxFeatureFrequencySelectorImpl"/>
    <property name="LINKING_IMPL" value="eu.etrap.tracer.linking.InterCorpusLinkingImpl"/>
    <property name="SCORING_IMPL" value="eu.etrap.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl"/>

    <property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />
    <property name="BASEFORM_FILE_NAME" value="data/corpora/Bible/Bible.lemma" />
    <property name="SYNONYMS_FILE_NAME" value="data/corpora/Bible/Bible.syns" />

    <property name="TOKENISATION_CHARACTERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />
    <property name="LETTER_SHAVER_MAPPING_FILE_NAME" value="data/letter-mappings/arabic.txt" />
```

Figure 5: The highlighted property provides the TRACER directory containing the texts to be investigated.

As you can see, the default corpus is the `KJV.txt` Bible file (*King James Version*). If you want to change to another corpus or to your corpus, you first need to format the text. It is recommended you upload the new corpus to the `corpora` subfolder of TRACER (see section 4.1).

4 Formatting a text to meet TRACER requirements

TRACER works with plain text files (`.txt`). This means that if you have marked-up texts in XML, you must remove all of the XML tags before you can use TRACER.

4.1 Where to store your texts (Recommendation)

To optimise the performance of TRACER, place your texts (two or more, depending on how many you want to query) in the same `.txt` file, one under the other. Placing all texts in a single file is also beneficial as TRACER will consider every line as a reuse unit to compare. To distinguish the texts, you will use different IDs (keep reading to find out how to do this). You'll place your texts in the `corpora` folder of TRACER's `data` folder (see Figure 6). It is recommended to use `corpora` as the data directory in order to avoid any rights issues.

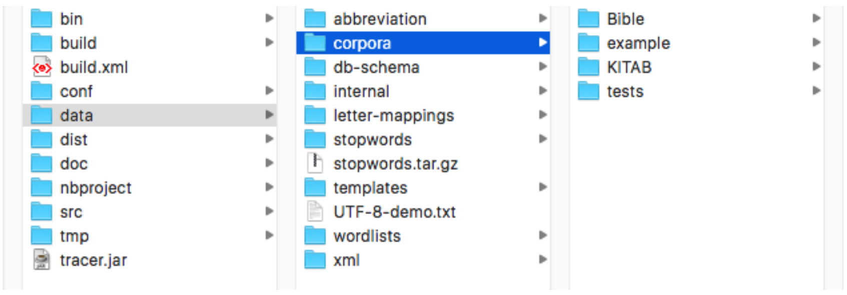


Figure 6: Structure of the TRACER folder (Mac 'Finder' view). Deposit your `.txt` file in a new folder under `data` > `corpora`.

4.2 How to prepare your texts

The plain text file has to be further formatted in order to meet TRACER's requirements.

4.2.1 Segmentation

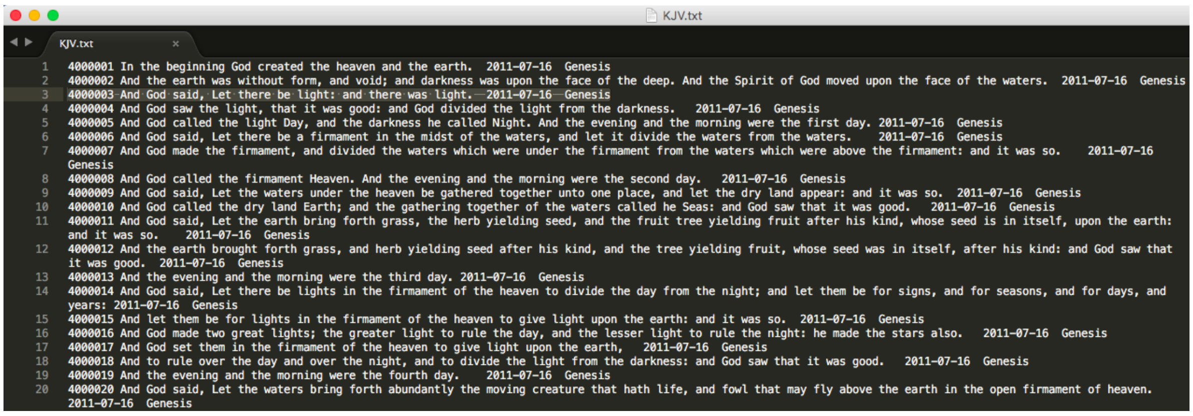
The first thing you need to do is to segmentise your texts by verse, paragraph, sentence, or whatever unit you believe best suits your reuse analysis. Every unit must appear on a separate line in the `.txt` file. You can use the free NLTK to quickly segmentise your text, but this tool will use full-stops as the only cue to identify the end of a sentence. Note that this might not be the best tool if you are trying to segmentise a very long text, as the online application only takes a certain number of lines. Always double-check the output because sometimes formatting errors can sneak in unnoticed.

4.2.2 Columns

Next, create four columns separated by `TAB`s. A good way of achieving this is by using Microsoft Excel. Let us assume you have segmentised your text by *sentence*:

1. the **first column** contains unique sentence IDs (running numbers are recommended, as shown in Table 1);
2. press `TAB` and in the **second column** put the sentence itself;
3. press `TAB` and in the **third column** put either the date of the file creation in the `YYYY-MM-DD` format or `NULL`. If `NULL`, make sure it is written in upper-case.
4. press `TAB` and in the **fourth column** put the book or section the sentence is taken from. This information is crucial for the visualisation shown in Figure 32. The top drop-down menu you see there will list the information you provide in this fourth column.

Figure 7 below provides an example of the King James Bible Version text formatted for TRACER.



```
4000001 In the beginning God created the heaven and the earth.  2011-07-16  Genesis
4000002 And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters.  2011-07-16  Genesis
4000003 And God said, Let there be light: and there was light.  2011-07-16  Genesis
4000004 And God saw the light, that it was good: and God divided the light from the darkness.   2011-07-16  Genesis
4000005 And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day. 2011-07-16  Genesis
4000006 And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters.    2011-07-16  Genesis
4000007 And God made the firmament, and divided the waters which were under the firmament from the waters which were above the firmament: and it was so.    2011-07-16
Genesis
4000008 And God called the firmament Heaven. And the evening and the morning were the second day.   2011-07-16  Genesis
4000009 And God said, Let the waters under the heaven be gathered together unto one place, and let the dry land appear: and it was so.  2011-07-16  Genesis
4000010 And God called the dry land Earth; and the gathering together of the waters called he Seas: and God saw that it was good.   2011-07-16  Genesis
4000011 And God said, Let the earth bring forth grass, the herb yielding seed, and the fruit tree yielding fruit after his kind, whose seed is in itself, upon the earth:
and it was so.    2011-07-16  Genesis
4000012 And the earth brought forth grass, and herb yielding seed after his kind, and the tree yielding fruit, whose seed was in itself, after his kind: and God saw that
it was good.  2011-07-16  Genesis
4000013 And the evening and the morning were the third day. 2011-07-16  Genesis
4000014 And God said, Let there be lights in the firmament of the heaven to divide the day from the night; and let them be for signs, and for seasons, and for days, and
years: 2011-07-16  Genesis
4000015 And let them be for lights in the firmament of the heaven to give light upon the earth: and it was so.  2011-07-16  Genesis
4000016 And God made two great lights; the greater light to rule the day, and the lesser light to rule the night: he made the stars also.   2011-07-16  Genesis
4000017 And God set them in the firmament of the heaven to give light upon the earth,   2011-07-16  Genesis
4000018 And to rule over the day and over the night, and to divide the light from the darkness: and God saw that it was good.   2011-07-16  Genesis
4000019 And the evening and the morning were the fourth day.    2011-07-16  Genesis
4000020 And God said, Let the waters bring forth abundantly the moving creature that hath life, and fowl that may fly above the earth in the open firmament of heaven.
2011-07-16  Genesis
```

Figure 7: The King James Bible (KJV) text formatted into four columns, as per TRACER's requirements. The columns, from left to right, are: Unique ID, Bible verse, Creation date, Book (source). This file was opened with Sublime text editor.

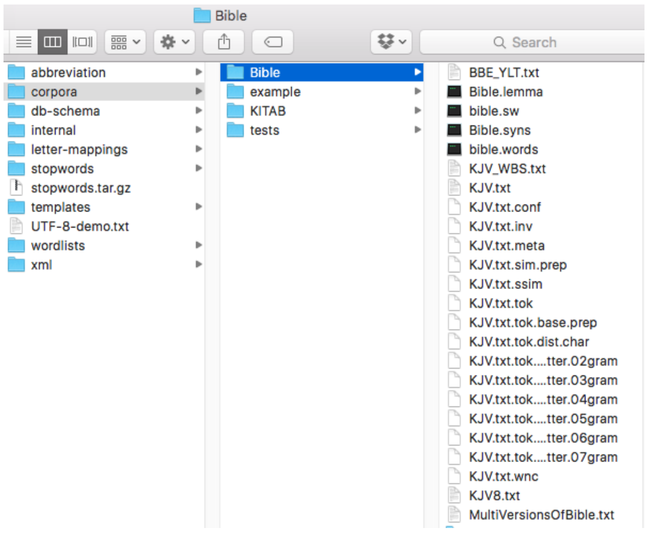
The sentence IDs should be sequential and unique. If you're analysing two texts, make sure to restart the ID sequence for your second text. So, for example:

1200001	text A
1200002	text A
1200003	text A
...	text A
1200349	text A
1300001	text B
1300002	text B
1300003	text B
...	text B
1300563	text B

Table 1: Required segment (sentence, verse, etc.) ID formatting.

4.2.3 Other relevant files

Besides `KJV.txt`, in the `Bible` subfolder of the `corpora` folder you will find a list of other files:



```
abbreviation        Bible       BBE_YLT.txt
corpora             example     Bible.lemma
db-schema           KITAB       bible.sw
internal            tests       Bible.syns
letter-mappings                 bible.words
stopwords                       KJV_WBS.txt
stopwords.tar.gz                KJV.txt
templates                       KJV.txt.conf
UTF-8-demo.txt                  KJV.txt.inv
wordlists                       KJV.txt.meta
xml                             KJV.txt.sim.prep
                                KJV.txt.ssim
                                KJV.txt.tok
                                KJV.txt.tok.base.prep
                                KJV.txt.tok.dist.char
                                KJV.txt.tok....tter.02gram
                                KJV.txt.tok....tter.03gram
                                KJV.txt.tok....tter.04gram
                                KJV.txt.tok....tter.05gram
                                KJV.txt.tok....tter.06gram
                                KJV.txt.tok....tter.07gram
                                KJV.txt.wnc
                                KJV8.txt
                                MultiVersionsOfBible.txt
```

Figure 8: Files contained in the `Bible` data folder of TRACER.

`KJV.txt.inv`
`inv` stands for *inverted list*. This file works like a word index and is the heart of any retrieval system. It shows you that a specific word appears in a specific verse in a specific position.

Word number	Verse ID	Position of word in verse
114	4003870	2

`KJV.txt.wnc`
`.wnc` stands for *words number complete*. This file gives more information about the word types in the corpus including frequency, rank and word length. IDs are frequency-sorted.

`KJV.txt.meta`
This file provides an overview or statistics, the *metadata* as it were, of the corpus segmentation:

- `SENTENCES`: lines.
- `WORD_TYPES` : number of unique words as dictionary entries.
- `WORD_TOKENS`[4]: occurrences of a word; every word in a text, no matter how many times it occurs.
- `SOURCES`: for example, a book.
- `SSIM_THRESHOLD`: degree of similarity required to consider two words as similarly written.
- `SSIM_EDGES`: number of links or word pairs satisfying the similarity requirement stated in `SSIM_THRESHOLD`.
- `BOW_WORD_TOKENS`: tokens that appear in a line, but counting them as one even if they appear multiple times in the line e.g. 'the' appears 3 times in a line, but is counted as 1.
- `SSIM_`: information is needed for the preprocessing step of TRACER. It's basically the preprocessing of the preprocessing.

`KJV.txt.tok`
This file is a *tokenized* version of the source text. This means that all punctuation has been removed or separated from the word (depending on the settings). The default setting in TRACER is `delete`.

`KJV.txt.tok.dist.char`
This file displays the *distribution* of the *characters* across the corpus.

`KJV.txt.tok.dist.letter.02gram`
This file displays the *distribution* of *letter bigrams* across the corpus.

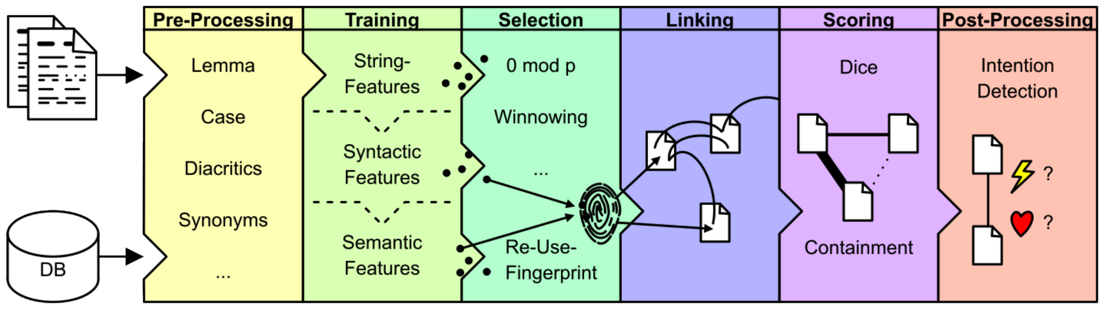
`KJV.txt.ssim`
Everything comes together in this file. The first and second columns represent the two words; the third column is the overlap of letter bigrams; the fourth column is the weighted overlap §w§between

$$\S w \in]0,1]\S \tag{1}$$

by Broder's Resemblance measure.

[4]Make sure you memorise the distinction between **word type** and **word token** as this terminology will come-up again later in the manual. An example might help: the sentence "The cat is eating the mouse" has 6 word tokens but 5 word types.

5 Preprocessing



5.1 First run of TRACER

Open the terminal and navigate to the TRACER folder as instructed at the beginning (using the command `cd`). To start TRACER, type:

```
java -Xmx600m -Dde.gcdh.medusa.config.ClassConfig=conf/tracer_config.xml -jar tracer.jar
```

Where:

- `java` opens a Java programme.
- `-Xmx600m` (up to 600 MB memory). If you work on your own data, you might have to raise this parameter to 1GB or 2GB.
- OPTIONAL `-Dfile.encoding` sets the encoding of your input file. This is important because operating systems use different default encoding.
- `-Dde.gcdh.medusa.config.ClassConfig` (configuration file).
- `D` = system property.

Press `ENTER` and TRACER will perform a first Text Reuse run of the texts. The speed of the first run will depend on the memory of your computer but it generally takes a few minutes. When TRACER is done, you'll see the screen below:

5.2 Results

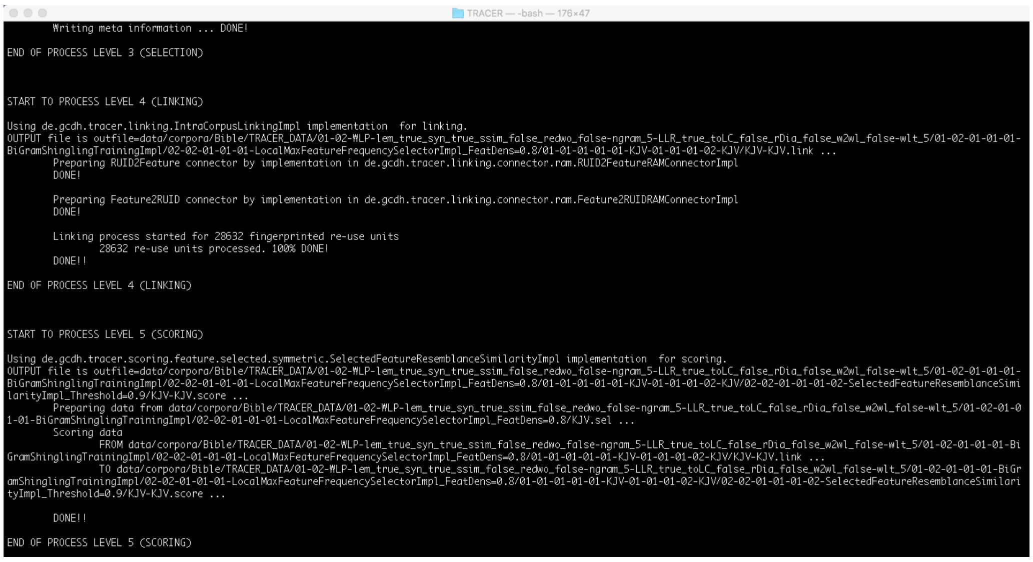
The results of the first run and, indeed, the results of every run of TRACER are automatically stored in a newly generated folder called `TRACER_DATA` under `data > corpora > Bible`:
If you're working with your own texts –not TRACER's default Bible data– the `TRACER_DATA` folder would be created in the relevant directory (specified in section 3.4).

5.2.1 How the results are organised

TRACER organises the results to reflect any changes the user makes to the configuration parameters (more about configuration customisations in section 5.3). It creates a deep folder structure with long but self-explanatory folder names. Figure 11 below is the extension of Figure 10.
As you can see, the first run of TRACER produced a folder with a very long name:

```
01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5
```

Here's how you read it (see also section 5.3.1 and section 5.3.2):



```
        Writing meta information ... DONE!

END OF PROCESS LEVEL 3 (SELECTION)



START TO PROCESS LEVEL 4 (LINKING)

Using de.gcdh.tracer.linking.IntraCorpusLinkingImpl implementation  for linking.
OUTPUT file is outfile=data/corpora/Bible/TRACER_DATA/01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5/01-02-01-01-01-
BiGramShinglingTrainingImpl/02-02-01-01-01-LocalMaxFeatureFrequencySelectorImpl_FeatDens=0.8/01-01-01-01-01-KJV-01-01-01-02-KJV/KJV-KJV.link ...
        Preparing RUID2Feature connector by implementation in de.gcdh.tracer.linking.connector.ram.RUID2FeatureRAMConnectorImpl
        DONE!

        Preparing Feature2RUID connector by implementation in de.gcdh.tracer.linking.connector.ram.Feature2RUIDRAMConnectorImpl
        DONE!

        Linking process started for 28632 fingerprinted re-use units
                28632 re-use units processed. 100% DONE!
        DONE!!

END OF PROCESS LEVEL 4 (LINKING)



START TO PROCESS LEVEL 5 (SCORING)

Using de.gcdh.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl implementation  for scoring.
OUTPUT file is outfile=data/corpora/Bible/TRACER_DATA/01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5/01-02-01-01-01-
BiGramShinglingTrainingImpl/02-02-01-01-01-LocalMaxFeatureFrequencySelectorImpl_FeatDens=0.8/01-01-01-01-01-KJV-01-01-01-02-KJV/02-02-01-01-01-02-SelectedFeatureResemblanceSimi
larityImpl_Threshold=0.9/KJV-KJV.score ...
        Preparing data from data/corpora/Bible/TRACER_DATA/01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5/01-02-01-0
1-01-BiGramShinglingTrainingImpl/02-02-01-01-01-LocalMaxFeatureFrequencySelectorImpl_FeatDens=0.8/KJV.sel ...
        Scoring data
                FROM data/corpora/Bible/TRACER_DATA/01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5/01-02-01-01-01-Bi
GramShinglingTrainingImpl/02-02-01-01-01-LocalMaxFeatureFrequencySelectorImpl_FeatDens=0.8/01-01-01-01-01-KJV-01-01-01-02-KJV/KJV-KJV.link ...
                TO data/corpora/Bible/TRACER_DATA/01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5/01-02-01-01-01-BiGr
amShinglingTrainingImpl/02-02-01-01-01-LocalMaxFeatureFrequencySelectorImpl_FeatDens=0.8/01-01-01-01-01-KJV-01-01-01-02-KJV/02-02-01-01-01-02-SelectedFeatureResemblanceSimilari
tyImpl_Threshold=0.9/KJV-KJV.score ...

        DONE!!

END OF PROCESS LEVEL 5 (SCORING)
```

Figure 9: The first run of TRACER is complete. The words ***`DONE!!`*** *and* ***`END OF PROCESS LEVEL 5 (SCORING)`*** *indicate that TRACER has successfully gone through the Preprocessing, Training, Selection, Linking and Scoring steps.*

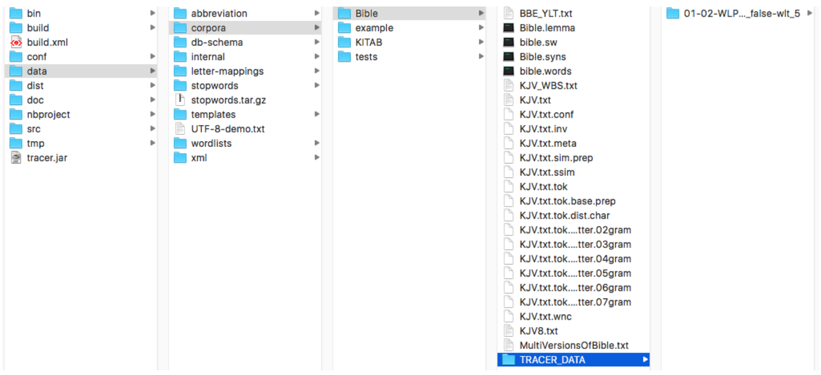
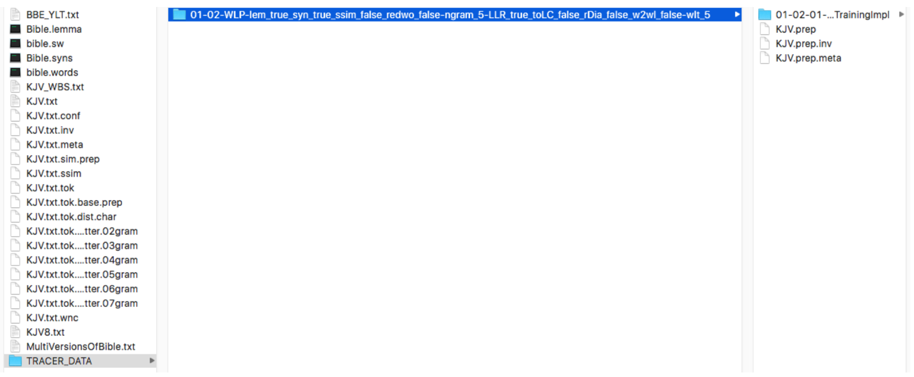


Figure 10: The folder path and location of the Text Reuse results produced by TRACER.

01-02-WLP-	Preprocessing step (01)-word-level (02)-Word Level Processing
lem_true_	Lemmatisation_ENABLED_
syn_true_	Synonym replacement_ENABLED_
ssim_false_	Replace String similar words_DISABLED_
redwo_false-ngram_5-LLR_true_	Reduced words option_DISABLED-to the most significant 5 letter n-gram- significance is measured by the Log-Likelihood Ratio_ENABLED_
toLC_false_	Everything to Lower Case_DISABLED_
rDia_false_	remove Diachritics_DISABLED_
w2wl_false-wlt_5	Replace word by word length_DISABLED-For words of 5 letters and more

5.3 Types of preprocessing

After the first run, you can decide whether to keep TRACER's default preprocessing properties or if you want to change them. Any changes you make entirely depend on your texts but **TRACER can perform two levels of preprocessing: letter-level** and **word-level processing**.

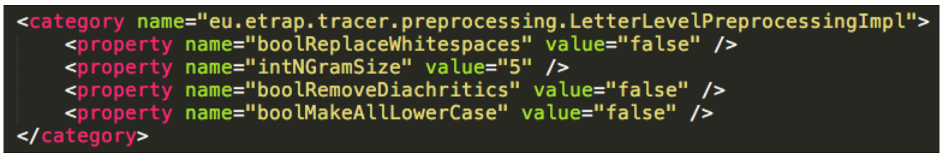


```
BBE_YLT.txt
Bible.lemma
bible.sw
Bible.syns
bible.words
KJV_WBS.txt
KJV.txt
KJV.txt.conf
KJV.txt.inv
KJV.txt.meta
KJV.txt.sim.prep
KJV.txt.ssim
KJV.txt.tok
KJV.txt.tok.base.prep
KJV.txt.tok.dist.char
KJV.txt.tok....tter.02gram
KJV.txt.tok....tter.03gram
KJV.txt.tok....tter.04gram
KJV.txt.tok....tter.05gram
KJV.txt.tok....tter.06gram
KJV.txt.tok....tter.07gram
KJV.txt.wnc
KJV8.txt
MultiVersionsOfBible.txt
TRACER_DATA

01-02-WLP-lem_true_syn_true_ssim_false_redwo_false-ngram_5-LLR_true_toLC_false_rDia_false_w2wl_false-wlt_5

01-02-01-...TrainingImpl
KJV.prep
KJV.prep.inv
KJV.prep.meta
```

Figure 11: The folder structure within `TRACER_DATA`. Long folder names are used to reflect the property settings in the TRACER `tracer_config.xml` file. This system allows users to better locate their results, especially when running TRACER multiple times with modified parameters.

5.3.1 Letter-level preprocessing

This makes sense for historical texts to, for example, address *scriptura continua* where the words are not separated by punctuation.

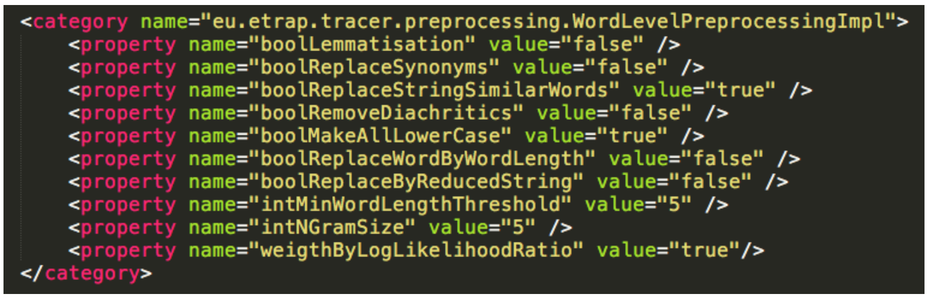


```xml
<category name="eu.etrap.tracer.preprocessing.LetterLevelPreprocessingImpl">
    <property name="boolReplaceWhitespaces" value="false" />
    <property name="intNGramSize" value="5" />
    <property name="boolRemoveDiachritics" value="false" />
    <property name="boolMakeAllLowerCase" value="false" />
</category>
```

Figure 12: The letter-level properties of TRACER as listed in the `tracer_config.xml` file.

5.3.2 Word-level preprocessing

This processing can be used to, for example, reduce inflected words to their base form (linguistic approach). *Synonym processing* is the default/most popular word-level processing but *cohyponym processing* yields better results. Other types of this processing are *word-length replacement*, which helps with calculation speed but also to better visualise the total number of features; or *string similarity*, that is, detecting which words or strings are similar to reveal OCR errors, morphological variation, typing or spelling errors, etc.



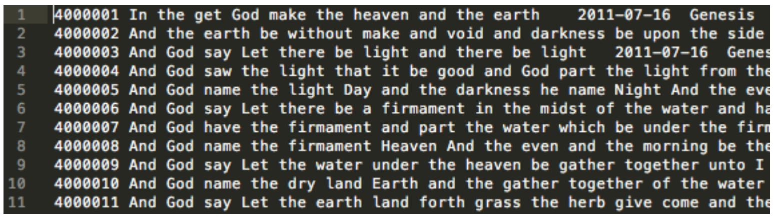
```xml
<category name="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl">
    <property name="boolLemmatisation" value="false" />
    <property name="boolReplaceSynonyms" value="false" />
    <property name="boolReplaceStringSimilarWords" value="true" />
    <property name="boolRemoveDiachritics" value="false" />
    <property name="boolMakeAllLowerCase" value="true" />
    <property name="boolReplaceWordByWordLength" value="false" />
    <property name="boolReplaceByReducedString" value="false" />
    <property name="intMinWordLengthThreshold" value="5" />
    <property name="intNGramSize" value="5" />
    <property name="weigthByLogLikelihoodRatio" value="true"/>
</category>
```

Figure 13: The word-level properties of TRACER as listed in the `tracer_config.xml` file.

5.3.3 Results

Glancing back at Figure 11, you'll notice that the `WLP` folder (the one with the very long name!) contains a number of files with the `.prep` (standing for `preprocessing`) suffix. Open each of these in your text editor and take a look.

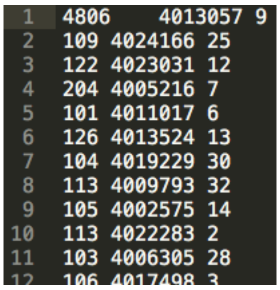
`KJV.prep`
This is the result file of the preprocessing step. If you look carefully, you'll notice that sentence `4000001` contains the words `get` and `make`, which are the preprocessed versions of the original `beginning` and `created` (see Figure 7). The lemmatisation settings have erroneously replaced `beginning` with `get` and the synonym replacement has changed `created` to `make`. These settings can be changed in order to correct any mistakes.



```
4000001 In the get God make the heaven and the earth    2011-07-16  Genesis
4000002 And the earth be without make and void and darkness be upon the side
4000003 And God say Let there be light and there be light   2011-07-16  Genes
4000004 And God saw the light that it be good and God part the light from the
4000005 And God name the light Day and the darkness he name Night And the eve
4000006 And God say Let there be a firmament in the midst of the water and ha
4000007 And God have the firmament and part the water which be under the firn
4000008 And God name the firmament Heaven And the even and the morning be the
4000009 And God say Let the water under the heaven be gather together unto I
4000010 And God name the dry land Earth and the gather together of the water
4000011 And God say Let the earth land forth grass the herb give come and the
```

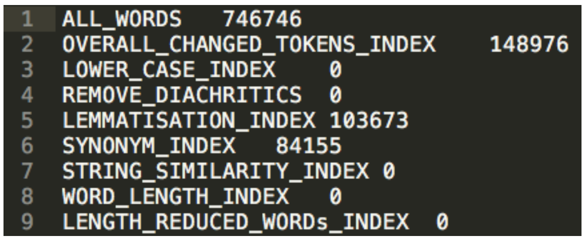
Similarly, in sentence `4000006`, the archaic term `midst` has not been replaced with the modern equivalent `middle` but it could if we wanted! For this reason, it's important that you thoroughly check the `KJV.prep` file before moving onto the next step.

`KJV.prep.inv`
`inv` stands for *inverted list*. It shows you that a specific word (first number) appears in a specific verse (second number) in a specific position (third number).



```
4806    4013057 9
109 4024166 25
122 4023031 12
204 4005216 7
101 4011017 6
126 4013524 13
104 4019229 30
113 4009793 32
105 4002575 14
113 4022283 2
103 4006305 28
106 4017498 3
```

`KJV.prep.meta`
This file provides some overview information about the preprocessing tasks, settings and results. For example, it tells us that 103673 words out of the entire corpus were lemmatised.



```
ALL_WORDS   746746
OVERALL_CHANGED_TOKENS_INDEX    148976
LOWER_CASE_INDEX    0
REMOVE_DIACHRITICS  0
LEMMATISATION_INDEX 103673
SYNONYM_INDEX   84155
STRING_SIMILARITY_INDEX 0
WORD_LENGTH_INDEX   0
LENGTH_REDUCED_WORDs_INDEX  0
```

The `WLP` folder also contains further nested folders, described in this manual as we move along.

5.3.4 Customising parameters or *properties*

To change preprocessing properties, enable/disable them in the `tracer_config.xml` as you see fit. For example, to switch off `lemmatisation`, replace its corresponding value `true` with the value `false` (as illustrated in Figure 13) and save the changes. When all the changes have been made, rerun TRACER and a new folder within `TRACER_DATA` will be produced with the updated results.[5] Lemmatisation helps to, for example, discriminate nouns from verbs (e.g. the word 'power', which can be both a verb and a noun). Lemmatisation is especially important when we wish to analyse paraphrases and allusions. If we wanted

[5]Results are never overwritten.

to find direct quotations (verbatim or near verbatim), lemmatisation is not going to be useful so we would switch its value in the `tracer_config.xml` file back to `false`.

5.4 Understanding preprocessing

The processing techniques described so far are common practice in Natural Language Processing (NLP). One of the laws of NLP is known as *Zipf's law*. According to Zipf, 90% of words in a text are rare, occurring 10 times or less; 50% of all words occur only once; 16% of all words occur only twice and 8% only three times, and so on. This means that the frequency of any word is inversely proportional to its statistical rank. For example, according to wordcount.org, the most popular word in the English language is 'the'. As such, its rank is 1 (see Figure 14).

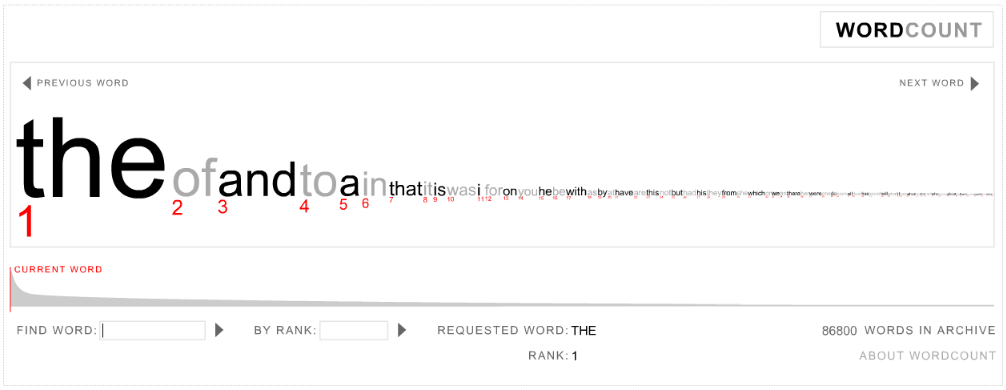


Figure 14: 'The' is the most popular word in the English language.

The word 'cat' ranks 2532 and does not appear as often as the word 'the' (Figure 15).

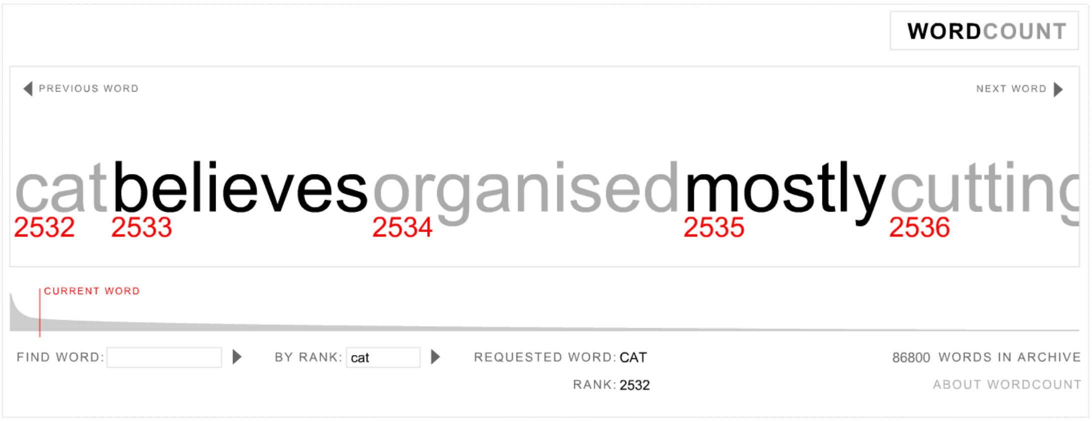


Figure 15: Rank of the English word 'cat'.

So, again, the higher the rank, the less frequent the word and vice versa. We can visualise this proportion as a log-log graph, which reveals a 'straight line' relation between word frequency and ranking (Figure 16).

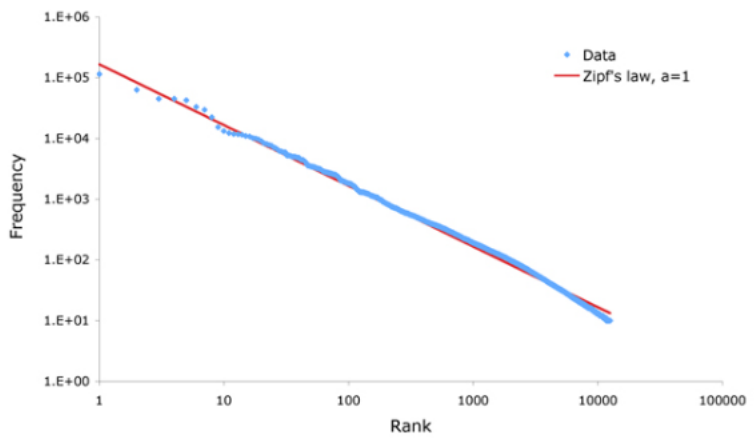
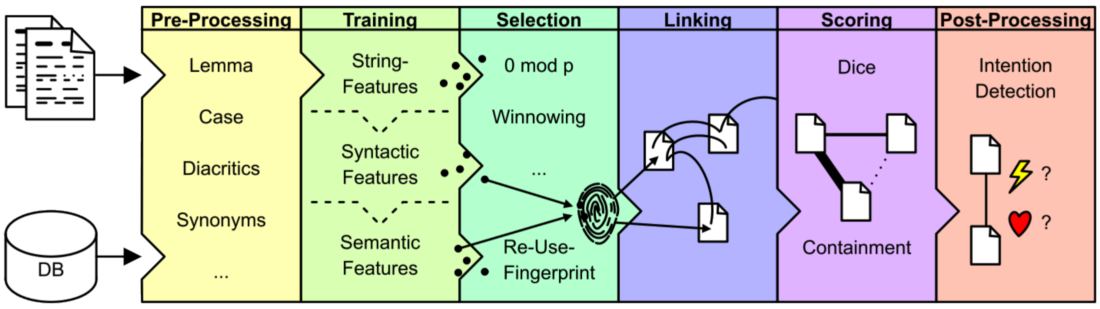


Figure 16: Log-log graph of word frequency and ranking. The higher the rank of a word, the lower its frequency and vice versa.

Amazingly, Zipf's law applies to all languages.

6 Training/Featuring



Now that we've preprocessed the text, we can proceed to breaking it down into units that can be compared (e.g. words, bigrams, trigrams). This second step is called *Training* or *Featuring* and with it we get a cleaned reuse unit and thus a digital fingerprint. One fingerprint for each measurable unit.

6.1 Types of featuring

6.1.1 Overlapping

An overlapping type of featuring is ***shingling***, a process typically used to detect near-verbatim reuse. In NLP or text mining, a shingle is an *n*-gram and the process of shingling creates a subsequence of overlapping tokens in a document. Here are some examples of shingling:

Example sentence: *I have a very big house*

Bigram shingling (*bigram* = an *n*-gram of size 2)
(I have), (have a), (a very), (very big), (big house)

Feature 1 = I have
Feature 2 = have a
Feature 3 = a very
Feature 4 = very big
Feature 5 = big house

Trigram shingling (*trigram* = an *n*-gram of size 3)
(I have a), (have a very), (a very big), (very big house)

Feature 1 = I have a
Feature 2 = have a very
Feature 3 = a very big
Feature 4 = very big house

Four-gram shingling (*four-gram* = an *n*-gram of size 4)
(I have a very), (have a very big), (a very big house)

Feature 1 = I have a very
Feature 2 = have a very big
Feature 3 = a very big house

Shingling creates more features, so your detector will need more time to compute similarity.

6.1.2 Non-overlapping

A non-overlapping type of featuring is ***hash-breaking***, a process typically used to detect duplicates, exact copies. Hash-breaking creates features with no overlap. Here are some examples of hash-breaking:

Example sentence: *I have a very big house*

Bigram hash-breaking (*bigram* = an *n*-gram of size 2)
(I have), (a very), (big house)

Feature 1 = I have
Feature 2 = a very
Feature 3 = big house

Trigram hash-breaking (*trigram* = an *n*-gram of size 3)
(I have a), (very big house)

Feature 1 = I have a
Feature 2 = very big house

6.1.3 Distance-based

A special type of bigram, called Distance-based Bigram, can be used to improve retrieval performance. Distance-based Bigrams are good for detecting paraphrases, or reuse that is not so literal. The Distance-based Bigram is a word pair whose distance between the two components is greater than or equal to 1.

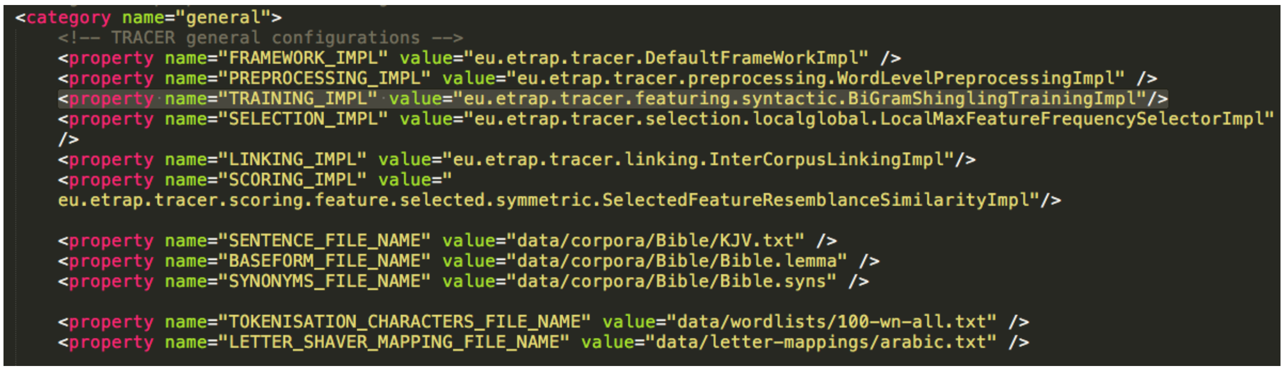
Example sentence: *I have a big house*

Bigram

I have 1	where 1 describes the distance between 'I' and 'have' (one word)
have a 2	where 2 describes the distance between 'I' and 'a'
a big 3	where 3 describes the distance between 'I' and 'big'
big house 4	where 4 describes the distance between 'I' and 'house'

6.1.4 Customising parameters or properties

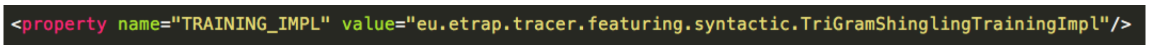
In TRACER, the featuring settings can be changed in the `tracer_config.xml` (see Figure 17).



```
<category name="general">
    <!-- TRACER general configurations -->
    <property name="FRAMEWORK_IMPL" value="eu.etrap.tracer.DefaultFrameWorkImpl" />
    <property name="PREPROCESSING_IMPL" value="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl" />
    <property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.syntactic.BiGramShinglingTrainingImpl"/>
    <property name="SELECTION_IMPL" value="eu.etrap.tracer.selection.localglobal.LocalMaxFeatureFrequencySelectorImpl"
    />
    <property name="LINKING_IMPL" value="eu.etrap.tracer.linking.InterCorpusLinkingImpl"/>
    <property name="SCORING_IMPL" value="
    eu.etrap.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl"/>

    <property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />
    <property name="BASEFORM_FILE_NAME" value="data/corpora/Bible/Bible.lemma" />
    <property name="SYNONYMS_FILE_NAME" value="data/corpora/Bible/Bible.syns" />

    <property name="TOKENISATION_CHARACTERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />
    <property name="LETTER_SHAVER_MAPPING_FILE_NAME" value="data/letter-mappings/arabic.txt" />
```

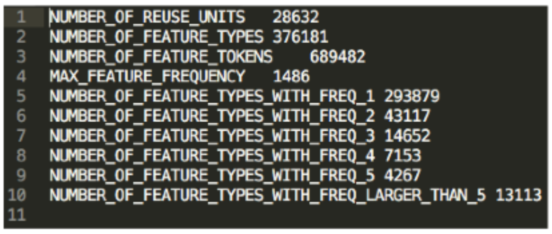
Figure 17: The value of the highlighted property can be changed to perform shingling or hash-breaking tasks.

Let's assume we want to change the above property in order to run trigram shingling on the text. The changed property will look like this:
We save the document and rerun TRACER. This reuse detection task produces a number of files, including:



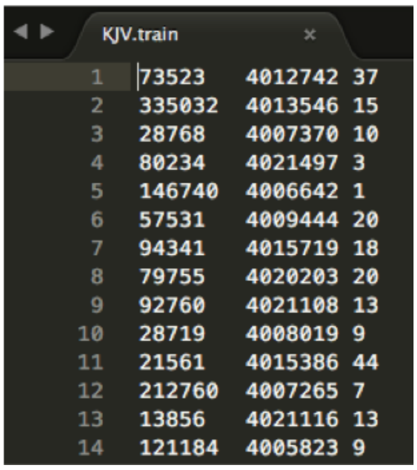
```
<property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.syntactic.TriGramShinglingTrainingImpl"/>
```

Figure 18: The value of the highlighted property has been changed from `BiGramShinglingTrainingImpl` to `TriGramShinglingTrainingImpl`.



```
NUMBER_OF_REUSE_UNITS	28632
NUMBER_OF_FEATURE_TYPES 376181
NUMBER_OF_FEATURE_TOKENS	689482
MAX_FEATURE_FREQUENCY	1486
NUMBER_OF_FEATURE_TYPES_WITH_FREQ_1 293879
NUMBER_OF_FEATURE_TYPES_WITH_FREQ_2 43117
NUMBER_OF_FEATURE_TYPES_WITH_FREQ_3 14652
NUMBER_OF_FEATURE_TYPES_WITH_FREQ_4 7153
NUMBER_OF_FEATURE_TYPES_WITH_FREQ_5 4267
NUMBER_OF_FEATURE_TYPES_WITH_FREQ_LARGER_THAN_5 13113
```

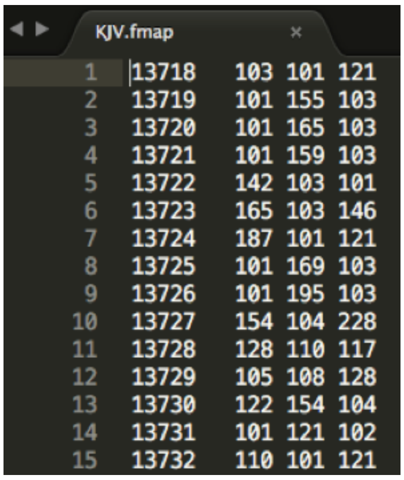
- `KJV.meta`
  This file provides a human-readable overview of the trigram shingling training.
- `KJV.train`
  This file is organised as follows:
  `FEATURE ID - REUSE UNIT ID - POSITION IN REUSE UNIT`



KJV.train

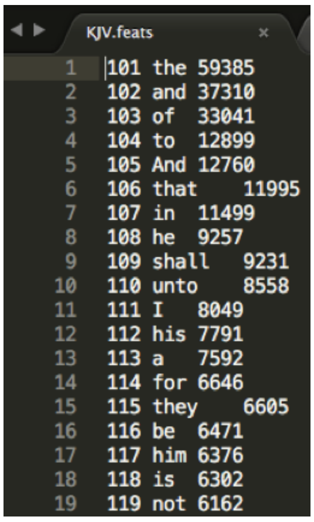
```
73523   4012742 37
335032  4013546 15
28768   4007370 10
80234   4021497 3
146740  4006642 1
57531   4009444 20
94341   4015719 18
79755   4020203 20
92760   4021108 13
28719   4008019 9
21561   4015386 44
212760  4007265 7
13856   4021116 13
121184  4005823 9
```

- `KJV.fmap`
  This file is arranged as follows:
  `FEATURE ID - WORD ID - WORD ID - WORD ID`



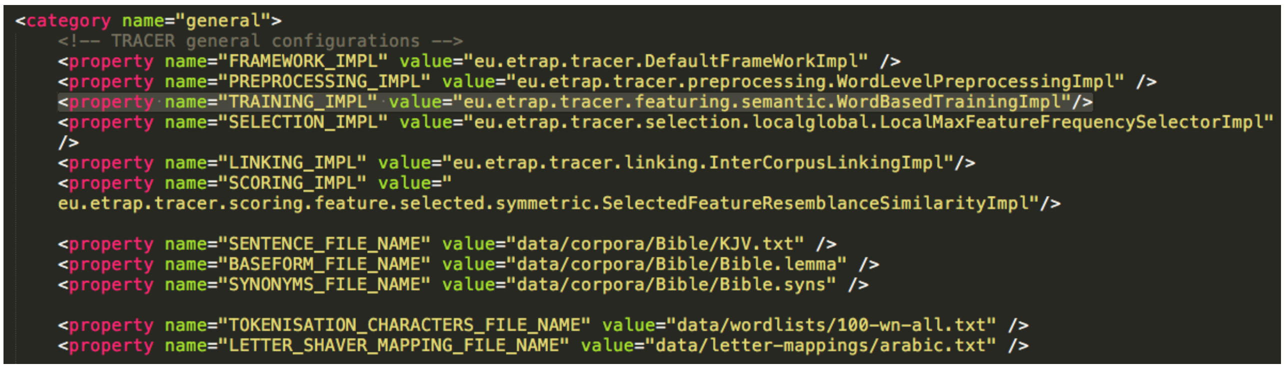
```
13718	103 101 121
13719	101 155 103
13720	101 165 103
13721	101 159 103
13722	142 103 101
13723	165 103 146
13724	187 101 121
13725	101 169 103
13726	101 195 103
13727	154 104 228
13728	128 110 117
13729	105 108 128
13730	122 154 104
13731	101 121 102
13732	110 101 121
```

- `KJV.feats`
  This document is the word index, where words are sorted by frequency:
  `WORD ID - WORD - FREQUENCY`



```
101 the 59385
102 and 37310
103 of  33041
104 to  12899
105 And 12760
106 that    11995
107 in  11499
108 he  9257
109 shall   9231
110 unto    8558
111 I   8049
112 his 7791
113 a   7592
114 for 6646
115 they    6605
116 be  6471
117 him 6376
118 is  6302
119 not 6162
```

If you look again at the *Training* property in TRACER's `tracer_config.xml` (Figure 18), you'll notice that the trigram shingling training we've done is *syntactic*. Syntactic dependency parsing is not always accurate because parsers (and the TRACER parser) are not yet able to infer dependencies. But if the computer takes manually annotated data from treebanks, for example, then the parser would be able to produce more accurate results. TRACER also contains a *semantic* featuring property, immediately preceding the syntactic property (Figure 19). This property is commented out by default but can be enabled by removing the comment syntax (`<!-.....->`).



```
<category name="general">
    <!-- TRACER general configurations -->
    <property name="FRAMEWORK_IMPL" value="eu.etrap.tracer.DefaultFrameWorkImpl" />
    <property name="PREPROCESSING_IMPL" value="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl" />
    <property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.semantic.WordBasedTrainingImpl"/>
    <property name="SELECTION_IMPL" value="eu.etrap.tracer.selection.localglobal.LocalMaxFeatureFrequencySelectorImpl"
    />
    <property name="LINKING_IMPL" value="eu.etrap.tracer.linking.InterCorpusLinkingImpl"/>
    <property name="SCORING_IMPL" value="
    eu.etrap.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl"/>

    <property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />
    <property name="BASEFORM_FILE_NAME" value="data/corpora/Bible/Bible.lemma" />
    <property name="SYNONYMS_FILE_NAME" value="data/corpora/Bible/Bible.syns" />

    <property name="TOKENISATION_CHARACTERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />
    <property name="LETTER_SHAVER_MAPPING_FILE_NAME" value="data/letter-mappings/arabic.txt" />
```

Figure 19: TRACER's training or featuring can also be semantic.

For *semantic* training, TRACER looks at *words* and *co-occurrences*. Figure 20 below provides an overview of *Training/Featuring*.

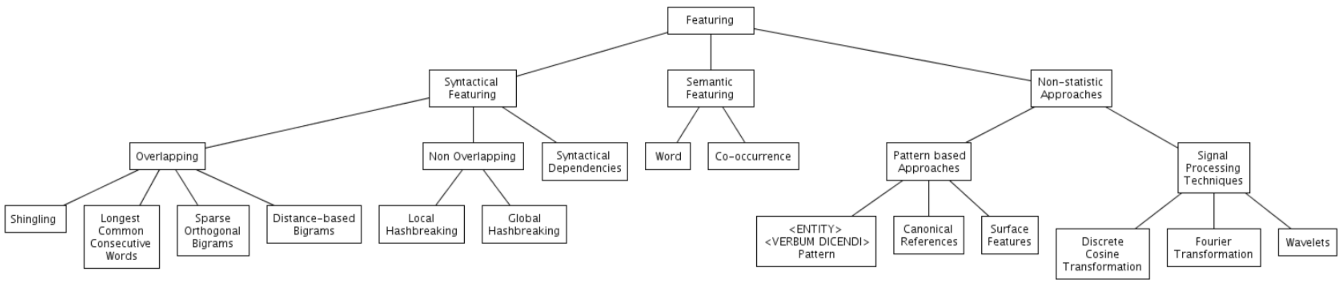
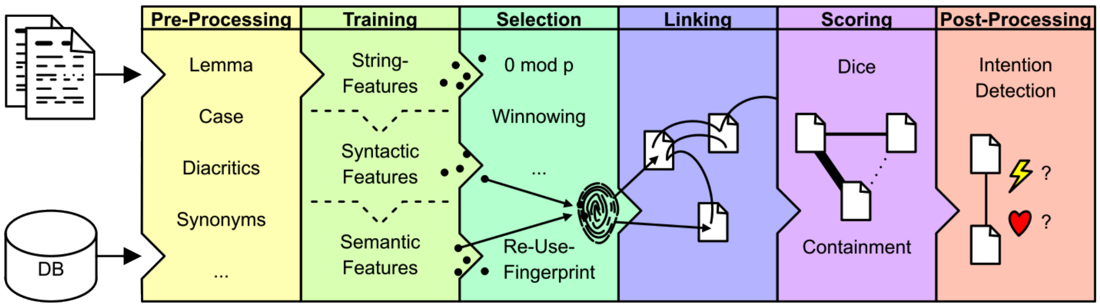


Figure 20: Overview of Featuring.

As you can see, there exists a third type of Featuring implementation, the *Non-statistic Approaches*. These aren't part of TRACER but are worth mentioning for the sake of contextualisation. *Verba dicendi* featuring could be used for fragmentary texts but is not a stable feature; *Surface Features*, for example quotation marks, are also an unstable feature as they don't always occur in historical texts; for *Canonical References*, please consult the research of Dr Matteo Romanello; for *Signal Processing* see [SC08].

7 Selection



There are roughly **sixty different selection strategies** in TRACER, which can also be combined. Through *Selection* we look at the idiosyncrasies or minutiae of the features we've computed, thus focusing on fewer features while being faster and more precise. This is the point where we create our 'fingerprint' with the specific primitives we're interested in. Algorithmically speaking, through *Selection* some entries in the `KJV.train` file will be kicked out. With *Selection* weire narrowing down the search by filtering out all irrelevant results. The challenge in this step is to find out what our *minutiae* are.

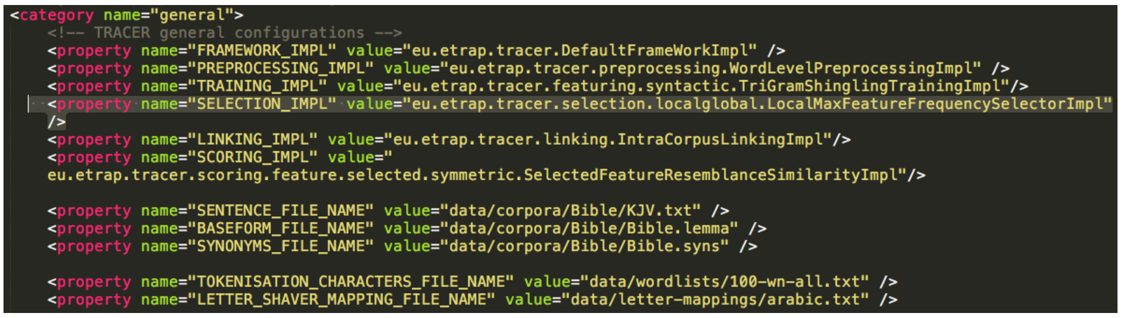
7.1 Selection strategies

There are many different selection strategies, including:

- *Pruning*: maximum or minimum.
  - *Maximum* pruning cuts out high frequency words (e.g. function words); why is this good? SPEED. For example, if you're looking at a feature that occurs only 10 times you are comparing one feature with 9 others - this means that you are making 90 comparisons. This is reasonably fast. If you, however, do not prune and have a frequency of 100, you would have to make 900 comparisons. That's slow.
  - *Minimum* pruning cuts out low frequency words (e.g. content words); why is this good? SPACE. The index is reduced so there is more space on the disc.
- *Term weighting*: we give features a weight. Rarer words have a higher weight.
- *Frequency classes*: you can take the frequency of the most frequent word and divided it by the frequency of the word you want to compare it to.
- *Feature dependencies*: it makes a comparison between paired features and sees if there are features that tend to co-occur.
- *Random Selection*: random selection of features. Very good if you don't have a clue of what to expect from your data.
- *Winnowing*: with this strategy we can pick a window. For example window size $w = 2$ means that the selection will work for every two features. If one word is a feature:

word 1	word 2	word 3	word 4
The	house	is	blue

  The selection is made based on the rarest parts of the winnowing window - i.e. the most infrequent features. But in this way function words are not entirely removed, since sometimes they can still be useful. With the winnowing algorithm it's possible to select features all over the reuse unit and avoid clusters. For example, "To be or not to be that is the question". It would appear that the only interesting word here is 'question' but we don't want to be eliminating all of the rest. Winnowing, with its windows, allows us to pick the lowest ranked feature (in frequency) for every window - so there is a selected feature for every window and features are distributed evenly.



```
<category name="general">
    <!-- TRACER general configurations -->
    <property name="FRAMEWORK_IMPL" value="eu.etrap.tracer.DefaultFrameWorkImpl" />
    <property name="PREPROCESSING_IMPL" value="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl" />
    <property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.syntactic.TriGramShinglingTrainingImpl"/>
    <property name="SELECTION_IMPL" value="eu.etrap.tracer.selection.localglobal.LocalMaxFeatureFrequencySelectorImpl"
    />
    <property name="LINKING_IMPL" value="eu.etrap.tracer.linking.IntraCorpusLinkingImpl"/>
    <property name="SCORING_IMPL" value="
    eu.etrap.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl"/>

    <property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />
    <property name="BASEFORM_FILE_NAME" value="data/corpora/Bible/Bible.lemma" />
    <property name="SYNONYMS_FILE_NAME" value="data/corpora/Bible/Bible.syns" />

    <property name="TOKENISATION_CHARACTERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />
    <property name="LETTER_SHAVER_MAPPING_FILE_NAME" value="data/letter-mappings/arabic.txt" />
```

Figure 21: The value of the highlighted Selection property in the `tracer_config.xml` file can be changed according to the preferred strategy.

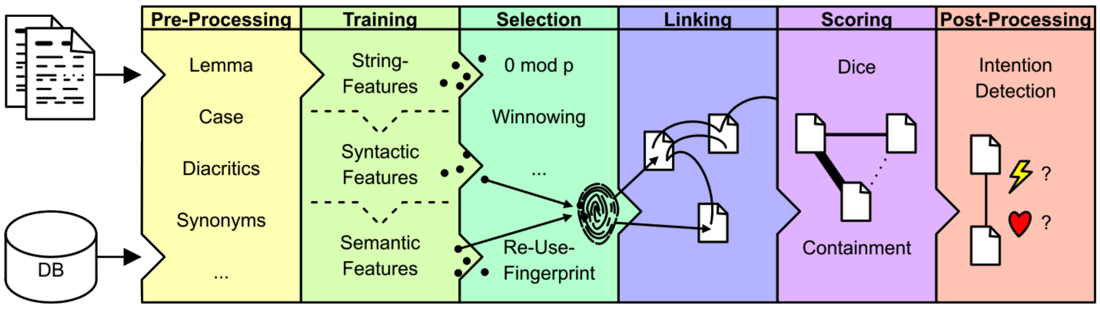
To change the selection strategy in TRACER, locate the *Selection* property in the `tracer_config.xml` file, as shown in Figure 21 below.
Different selection strategies require different parameters. This makes it difficult to compare selection strategies. For this reason, we introduce the ***Feature Density*** or, in other words, the comparison between the overall number of features and the number of features selected. Here's how TRACER computes feature density:

$$F = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m'} s_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij}}$$

		Selection Knowledge	
		local	global
Selection Usage	local	**Pro**: Streaming ability, simple parallelisation **Con**: a *Meme* may not have the same features in two s_i and s_j **Example**: Word length	**Pro**: Analysing relations between *Features* through the *Digital Signature* $\vec{s_i}$ with $\|\vec{s_i}\| \neq 0$ **Example**: *Contrastive Semantics*
	global	**Pro**: Linguistic features can be considered **Con**: Distribution of *Local Knowledge* must be reduced to a score **Example**: *PoS*-Tags	**Pro**: Analysing relations between *Features* **Con**: The *Digital Signature* $\vec{s_i}$ with $\|\vec{s_i}\| = 0$ can be generated **Example**: *max pruning*

Table 2: Selection Knowledge vs. Selection Usage. The matrix compares Pros and Cons between the respective categories of Selection processes. Global Selection Knowledge with Local Selection Usage offers the best compromise between the mentioned advantages and disadvantages.

8 Linking



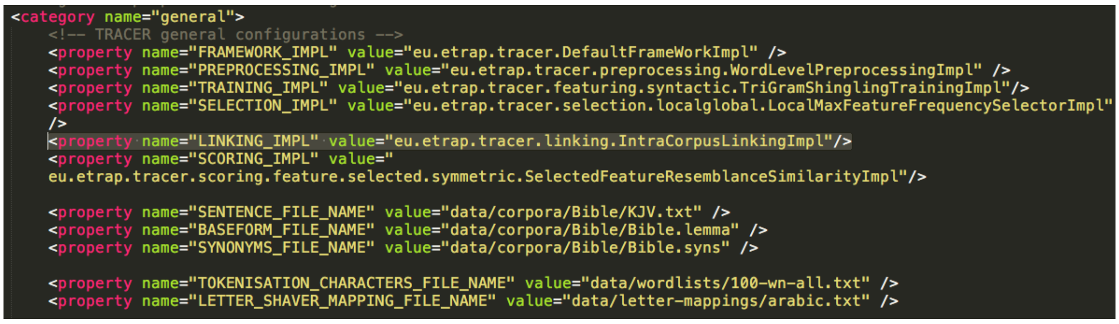
During the *Linking* stage, TRACER searches for the occurrences of the features defined in previous steps. Unlike other TRACER steps, which are all linear, *Linking* is of squared complexity depending on the feature frequency. Through *Linking* TRACER will deliver results based purely on the parameters set in previous steps. It is up to us to interpret those results and filter out what is good and what is bad.

8.1 Types of Linking

TRACER can perform two types of linking:

- ***Intra**linking*: looking for matches within the entire digital library (= multiple, different texts in one file), both within the same text and across texts;
- ***Inter**linking*: looking for matches across texts within the same file;

This parameter can be changed in the TRACER `tracer_config.xml`, as illustrated in Figure 22:



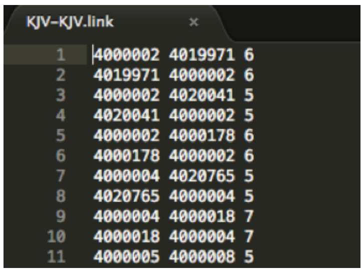
```
<category name="general">
    <!-- TRACER general configurations -->
    <property name="FRAMEWORK_IMPL" value="eu.etrap.tracer.DefaultFrameWorkImpl" />
    <property name="PREPROCESSING_IMPL" value="eu.etrap.tracer.preprocessing.WordLevelPreprocessingImpl" />
    <property name="TRAINING_IMPL" value="eu.etrap.tracer.featuring.syntactic.TriGramShinglingTrainingImpl"/>
    <property name="SELECTION_IMPL" value="eu.etrap.tracer.selection.localglobal.LocalMaxFeatureFrequencySelectorImpl"
    />
    <property name="LINKING_IMPL" value="eu.etrap.tracer.linking.IntraCorpusLinkingImpl"/>
    <property name="SCORING_IMPL" value="
    eu.etrap.tracer.scoring.feature.selected.symmetric.SelectedFeatureResemblanceSimilarityImpl"/>

    <property name="SENTENCE_FILE_NAME" value="data/corpora/Bible/KJV.txt" />
    <property name="BASEFORM_FILE_NAME" value="data/corpora/Bible/Bible.lemma" />
    <property name="SYNONYMS_FILE_NAME" value="data/corpora/Bible/Bible.syns" />

    <property name="TOKENISATION_CHARACTERS_FILE_NAME" value="data/wordlists/100-wn-all.txt" />
    <property name="LETTER_SHAVER_MAPPING_FILE_NAME" value="data/letter-mappings/arabic.txt" />
```

Figure 22: The value of the highlighted property in the TRACER `tracer_config.xml` can be changed to `InterCorpusLinkingImpl`, if needed.

TRACER outputs *Linking* results in a `.link` file in a three column structure:

```
REUSE ID 1 - REUSE ID 2 - OVERLAP
```

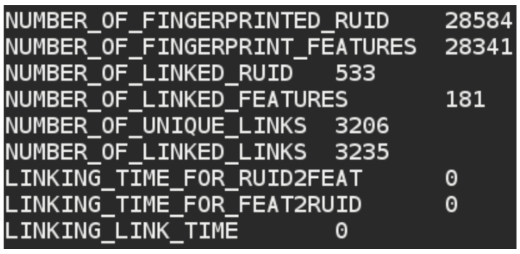
Here's how this structure looks like in the corresponding file:



```
4000002 4019971 6
4019971 4000002 6
4000002 4020041 5
4020041 4000002 5
4000002 4000178 6
4000178 4000002 6
4000004 4020765 5
4020765 4000004 5
4000004 4000018 7
4000018 4000004 7
4000005 4000008 5
```

Figure 23: The three-column structure in the Linking output file of the King James Version Bible text: ***`REUSE ID 1 - REUSE ID 2 - OVERLAP`***

What does TRACER mean by *overlap*? The (minimal) overlap is the number of common features shared by the first two columns (`REUSE ID 1` and `REUSE ID 2`), which TRACER sets as default to 5. This overlap number can be changed and is used to cut the long tail of reuses which would likely not be relevant matches or reuses at all. The `.meta` *Linking* file will provide you with an overview of the features linked:



```
NUMBER_OF_FINGERPRINTED_RUID    28584
NUMBER_OF_FINGERPRINT_FEATURES  28341
NUMBER_OF_LINKED_RUID   533
NUMBER_OF_LINKED_FEATURES       181
NUMBER_OF_UNIQUE_LINKS  3206
NUMBER_OF_LINKED_LINKS  3235
LINKING_TIME_FOR_RUID2FEAT      0
LINKING_TIME_FOR_FEAT2RUID      0
LINKING_LINK_TIME       0
```

Figure 24: Overview of Linking results provided by the Linking `.meta` *file.*

Linking is the most time-consuming step and the step that can be best parallelised, as shown in Figure 25 below.

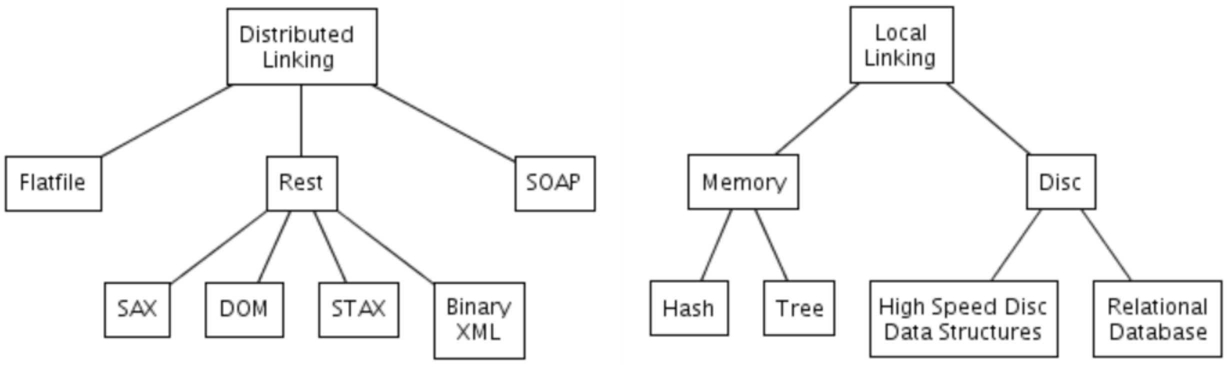
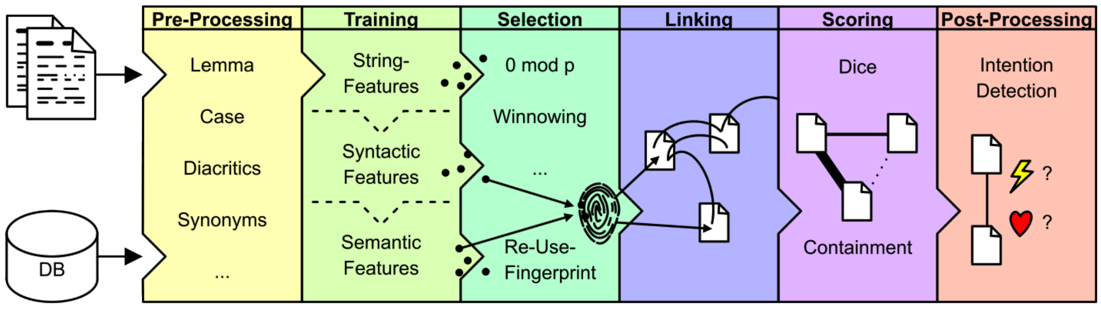


Figure 25: Overview of the Linking step. TRACER mostly deals with Local Linking but, if necessary,can also support Distributed Linking.

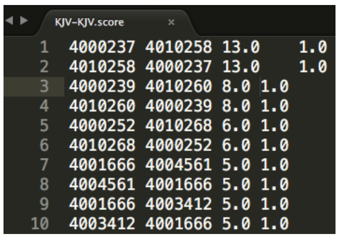
9 Scoring



The *Scoring* process of TRACER simply adds a fourth column to the previous `KJV-KJV.link` file (see Figure 23) with the *weighted overlap*.[6] To calculate the *Resemblance* score, TRACER uses the formula:

$$\theta_{\Theta}^{R'}(s_i, s_j) = \theta_{\Theta}^{O}(s_i, s_j) \cdot \theta_{\Theta}^{R}(s_i, s_j) = \frac{|\vec{s_i} \cap \vec{s_j}|^2}{|\vec{s_i} \cup \vec{s_j}|} \tag{2}$$

And the resulting file will look something like:



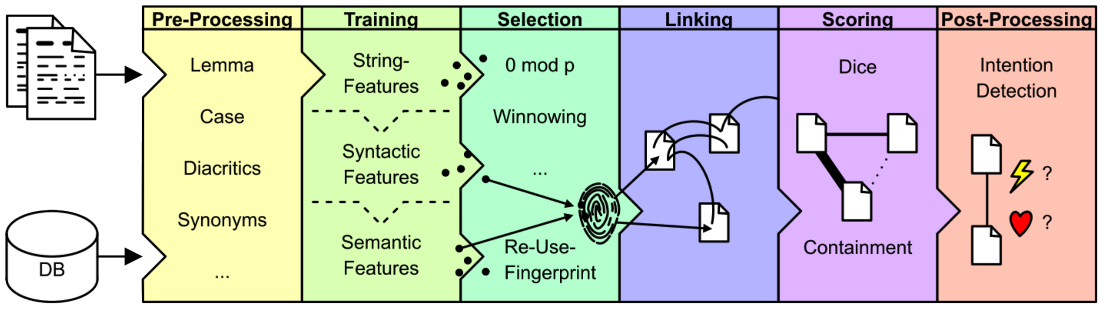
```
KJV-KJV.score
4000237 4010258 13.0     1.0
4010258 4000237 13.0     1.0
4000239 4010260 8.0 1.0
4010260 4000239 8.0 1.0
4000252 4010268 6.0 1.0
4010268 4000252 6.0 1.0
4001666 4004561 5.0 1.0
4004561 4001666 5.0 1.0
4001666 4003412 5.0 1.0
4003412 4001666 5.0 1.0
```

Figure 26: The fourth column in the `KJV-KJV.scoring` file provides the weighted overlap.

[6]The third column in Figure 26 represents *absolute overlap*.

10 Postprocessing: visualising the reuse results

Slides not available.



TRACER integrates a variant graph visualisation tool called TRAViz[7] in order to visualise the reuse results in a more legible format. TRAViz provides different types of visualisations for both close and distant reading, including dot-plots and text-alignments.

10.1 Download the visualisation package

To view your results with TRAViz, download the visualisation package `TextReuseVisualization.tar.gz` from here. Unzip the package and store it in the same directory as TRACER (i.e. *not* in the `TRACER` folder).

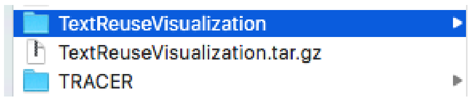

TextReuseVisualization
TextReuseVisualization.tar.gz
TRACER

Figure 27: Unzip the `TextReuseVisualization.tar.gz` file and store the `TextReuseVisualization` folder next to the `TRACER` folder.

10.2 Produce the visualisation

Copy both the `KJV-KJV.score` file produced during the previous *Scoring* step (from the relevant folder in `TRACER > data > corpora > Bible > TRACER_DATA`) and the `KJV.txt` file and paste them in the `TextReuseVisualization > html` folder, as illustrated in Figure 28:

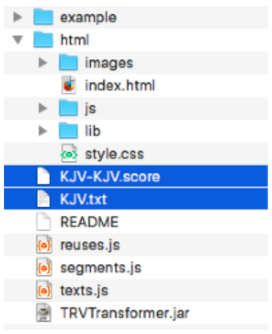

example
html
images
index.html
js
lib
style.css
KJV-KJV.score
KJV.txt
README
reuses.js
segments.js
texts.js
TRVTransformer.jar

Figure 28: Contents of the `TextReuseVisualization` folder as seen on a Mac computer. The two highlighted files should be copied from the `TRACER` folders and pasted in this folder.

Open the terminal, navigate to the `TextReuseVisualization` folder with the `cd` command and run the following:

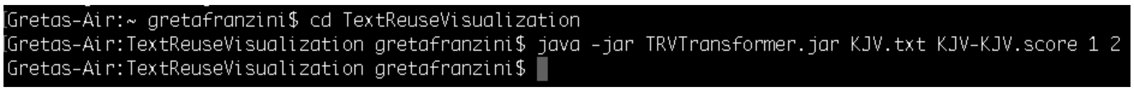
```
java -jar TRVTransformer.jar KJV.txt KJV-KJV.score 1 2
```

[7]TRAViz was developed and is maintained by Stefan Jänicke at the University of Leipzig.

Where:

- `1` is an internal parameter (you can choose between `1` or `2`); the `README` file in the `TextReuseVisualization` folder explains the use of these parameters in more detail.
- `2` is the minimum number of features; this can be changed depending on the number of common features you wish to visualise. The higher the number the better the performance but the lower the results.

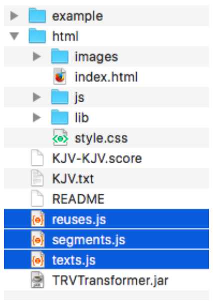
The terminal will return an empty line, as shown in Figure 29:



```
Gretas-Air:~ gretafranzini$ cd TextReuseVisualization
Gretas-Air:TextReuseVisualization gretafranzini$ java -jar TRVTransformer.jar KJV.txt KJV-KJV.score 1 2
Gretas-Air:TextReuseVisualization gretafranzini$ 
```

Figure 29: The `java` command above seemingly produces nothing but has computed results in the background.

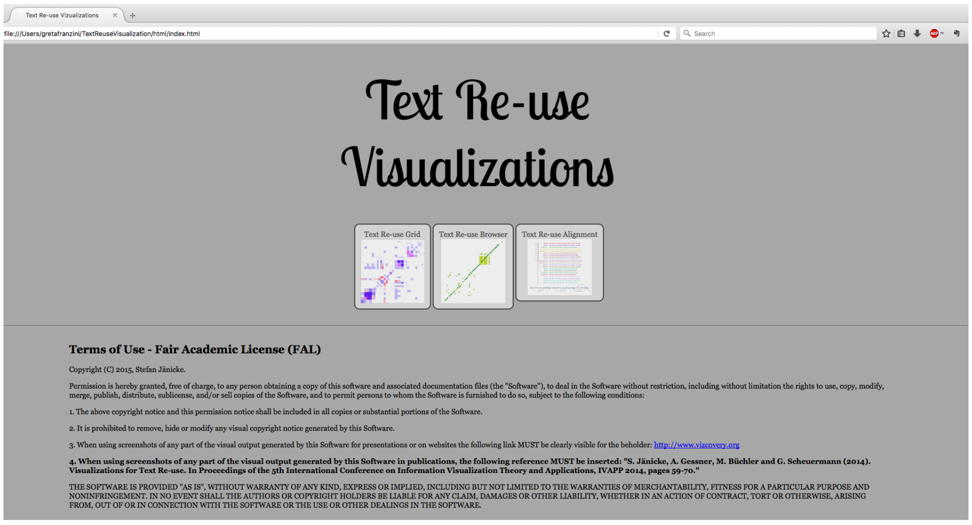
But in reality results have been computed in the background and the JavaScript results are now written over the three `.js` files you see listed in Figure 30 below:



```
▶ example
▼ html
    ▶ images
      index.html
    ▶ js
    ▶ lib
      style.css
  KJV-KJV.score
  KJV.txt
  README
  reuses.js
  segments.js
  texts.js
  TRVTransformer.jar
```

Figure 30: The `.js` results produced by the `java` command in Figure 29 are written over the existing `.js` files in the `TextReuseVisualization` folder.

To view the computed results, double-click on the `index.html` file you see in the `TextReuseVisualization` folder and your browser should open the page displayed in Figure 31:


Text Re-use Visualizations
file:///Users/gretafranzini/TextReuseVisualization/html/index.html
Search
Text Re-use Visualizations
Text Re-use Grid
Text Re-use Browser
Text Re-use Alignment

Terms of Use - Fair Academic License (FAL)

Copyright (C) 2015, Stefan Jänicke.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

1. The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

2. It is prohibited to remove, hide or modify any visual copyright notice generated by this Software.

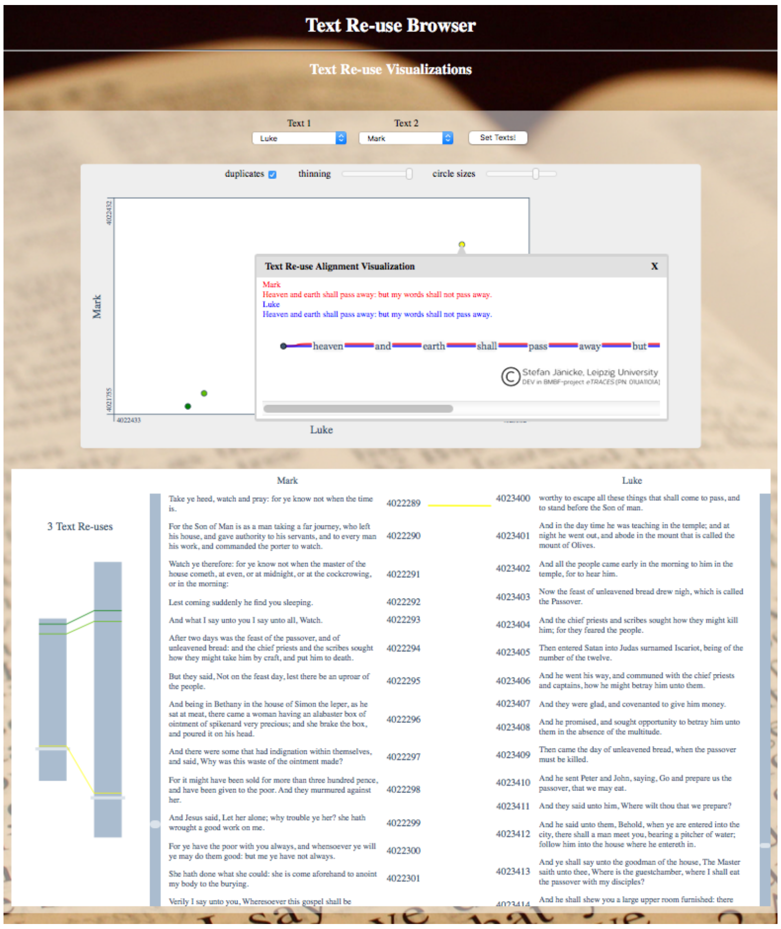
3. When using screenshots of any part of the visual output generated by this Software for presentations or on websites the following link MUST be clearly visible for the beholder: http://www.vizcovery.org

4. When using screenshots of any part of the visual output generated by this Software in publications, the following reference MUST be inserted: "S. Jänicke, A. Gessner, M. Büchler and G. Scheuermann (2014). Visualizations for Text Re-use. In Proceedings of the 5th International Conference on Information Visualization Theory and Applications, IVAPP 2014, pages 59-70."

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Figure 31: The `index.html` file in the `TextReuseVisualization` folder is the access point to the visualisations computed by TRACER/TRAViz.

Here, you can explore three different visualisations of your results. Try opening the middle one, the *Text Re-use Browser*.



Mark
Heaven and earth shall pass away: but my words shall not pass away.
Luke
Heaven and earth shall pass away: but my words shall not pass away.

Mark

Take ye heed, watch and pray: for ye know not when the time is. 4022289

For the Son of Man is as a man taking a far journey, who left his house, and gave authority to his servants, and to every man his work, and commanded the porter to watch. 4022290

Watch ye therefore: for ye know not when the master of the house cometh, at even, or at midnight, or at the cockcrowing, or in the morning: 4022291

Lest coming suddenly he find you sleeping. 4022292

And what I say unto you I say unto all, Watch. 4022293

After two days was the feast of the passover, and of unleavened bread: and the chief priests and the scribes sought how they might take him by craft, and put him to death. 4022294

But they said, Not on the feast day, lest there be an uproar of the people. 4022295

And being in Bethany in the house of Simon the leper, as he sat at meat, there came a woman having an alabaster box of ointment of spikenard very precious; and she brake the box, and poured it on his head. 4022296

And there were some that had indignation within themselves, and said, Why was this waste of the ointment made? 4022297

For it might have been sold for more than three hundred pence, and have been given to the poor. And they murmured against her. 4022298

And Jesus said, Let her alone; why trouble ye her? she hath wrought a good work on me. 4022299

For ye have the poor with you always, and whensoever ye will ye may do them good: but me ye have not always. 4022300

She hath done what she could: she is come aforehand to anoint my body to the burying. 4022301

Verily I say unto you, Wheresoever this gospel shall be

Luke

4023400 worthy to escape all these things that shall come to pass, and to stand before the Son of man.

4023401 And in the day time he was teaching in the temple; and at night he went out, and abode in the mount that is called the mount of Olives.

4023402 And all the people came early in the morning to him in the temple, for to hear him.

4023403 Now the feast of unleavened bread drew nigh, which is called the Passover.

4023404 And the chief priests and scribes sought how they might kill him; for they feared the people.

4023405 Then entered Satan into Judas surnamed Iscariot, being of the number of the twelve.

4023406 And he went his way, and communed with the chief priests and captains, how he might betray him unto them.

4023407 And they were glad, and covenanted to give him money.

4023408 And he promised, and sought opportunity to betray him unto them in the absence of the multitude.

4023409 Then came the day of unleavened bread, when the passover must be killed.

4023410 And he sent Peter and John, saying, Go and prepare us the passover, that we may eat.

4023411 And they said unto him, Where wilt thou that we prepare?

4023412 And he said unto them, Behold, when ye are entered into the city, there shall a man meet you, bearing a pitcher of water; follow him into the house where he entereth in.

4023413 And ye shall say unto the goodman of the house, The Master saith unto thee, Where is the guestchamber, where I shall eat the passover with my disciples?

4023414 And he shall shew you a large upper room furnished: there

Figure 32: Text Re-use Browser visualisation of TRACER results, comparing Mark with Luke.

The greener the colour the more similar the matches. The top plot (side note: this plot is typically used in plagiarism detection) displays Mark on the Y axis and Luke on the X axis. Mark develops from bottom to top, Luke from left to right. The dots pinpoint similarities and the colour the degree of similarity. In the bottom plot you can explore the reuse alignments and at which point of the works they occur. Play with the visualisation and try to interpret the results!

10.3 Updating the visualisations to modified parameters

Every time you change a parameter in the `tracer_config.xml` file you have to repeat this process in order to obtain an updated visualisation. This means copying the new `KJV-KJV.score` file to the `TextReuseVisualization` folder, running the visualisation command in the terminal and refreshing the `index.html` page. In short, the visualisation does not update itself. A future implementation of TRACER/TRAViz is planned in order to automate this process as much as possible.

Feedback

We greatly appreciate your feedback on this handbook. Your suggestions and comments will help us improve our documentation and adapt TRACER to your needs. We welcome both general feedback about TRACER and manual-specific suggestions. For the latter, please remember include information about the step(s) requiring our attention. To get in touch, please email us at contact@etrap.eu

Links

- Historical Text Reuse Detection Google Group;
- NLTK Sentence Segmentation;
- Text Reuse Zotero Group;
- Wordcount.org

Frequently Asked Questions (FAQ)

Q. How do I cite TRACER?
The first TRACER article in English is still under publication. Depending on your referencing style, you can adapt the following:

Büchler, M., Franzini, G., Franzini, E. Bulert, K. (2016 forthcoming) 'TRACER - a multilevel framework for historical Text Reuse detection', *Journal of Data Mining and Digital Humanities - Special Issue on Computer-Aided Processing of Intertextuality in Ancient Languages*.

Q. Can I reuse TRACER's code in my work?
Yes. As per TRACER's Academic Free License (see Copyright), you're allowed to repurpose the source code as long as you credit the source, share your code in a similar fashion and don't financially profit from it.

Q. Where can I find information about future TRACER tutorials?
The `www.etrap.eu` website always lists upcoming tutorials a few weeks in advance. These are typically run during conferences or summer schools.

References

[Büc08] Marco Büchler. *Medusa: Performante Textstatistiken auf großen Textmengen: Kookkurrenzanalyse in Theorie und Anwendung*. VDM Verlag Dr. Müller, 2008.

[SC08] Jangwon Seo and W. Bruce Croft. Local text reuse detection. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 571–578, New York, NY, USA, 2008. ACM.