Improving Access to Text
IMPACT

Evaluation of named entity work in IMPACT: NE Recognition and matching

Frank Landsbergen, INL

Part of D-EE2.6

Table of contents

KB IMPACT is supported by the European Community under the FP7 ICT Work Programme. The project is coordinated by the National Library of the Netherlands

Part I: Comparing two NER-tools on historical data

Abstract

We compare two well-known open source tools for Named Entity Recognition, Lingpipe[1] and the Stanford Named Enity Recognizer[2] for named entity recognition and investigate the possibilities to enhance performance of NE recognition for historical documents. Both tools have been tested with modern and historical data. We draw the following conclusions:

- Both tools do not necessarily perform poorly with historical data.
- Both tools do not necessarily perform poorly with OCR'ed data.
- Spelling variation does not necessarily lead to poor performance.
- A score of (almost) 100% is theoretically possible for both tools.
- A larger training file leads to better performance.
- Randomly generated (at sentence level) training files lead to better performance.
- Similarity between training- and test file (such as in time period or genre) seems to be the key factor affecting performance. A higher similarity leads to a better performance.
- Gazetteer lists lead to equal or better performance with Stanford, but can lead to worse performance with Lingpipe if there is little overlap between the list and the actual names in the test file.
- Using text heterogenity and overlap between training- and test files, it might be possible to predict the f1-score, but this has to be further tested because of the limited amount of data so far.

As regards the question of which tool is to be preferred in IMPACT, we observe:

- The performance of both tools is very similar.
- In general, Stanford performs slightly better than Lingpipe (with differences between 1-5%).
- Stanford is sensitive and crashed with a dataset that gave no problems for Lingpipe.
- The tools seem to have a different approach: Lingpipe is more 'greedy', its recall is high but its precision is low (many false negatives). Stanford is more precise, with a lower recall but a higher precision.
- Lingpipe's 'greedy' approach has the benefit of getting many of the correct names out of the data. While it also leaves you with a load of wrong names, these can be removed in postprocessing. Stanford's approach usually leads to an uncomplete list of correct names, but the list of incorrect names is shorter.
- If gazetteer lists are used, Stanford is to be preferred over Lingpipe, because its performance never decreases with such lists, even when they are useless.

Accordingly, we chose to use the Stanford tool as the main NE recognizer for IMPACT.

[1] http:/alias-i.com/lingpipe/

[2] http://nlp.stanford.edu/software/CRF-NER.shtml

Data used in the benchmarks

We used the following historical datasets:

Dataset	*OCR?*	*Genre*	*Time period*	*Number of words*
DBNL	no	prose, poetry, plays, non-fiction	18th and 19th c.	18th c: 581099 19th c: 272720
Staten Generaal	yes	parliamentary proceedings	19th and 20th c.	19th c: 273797 20th c: 280805
Newspapers	yes	various Dutch newspapers	19th c.	19th c: 254253

We also used the CONLL 2002[3] Dutch dataset, which consists of contemporary Belgian (Flemish) newspapers, for a first benchmark test.

Benchmark test I: historical data vs. contemporary data

One of our assumptions is that NE-tools perform less well with historical data when compared to contemporary data. This assumption turns out to be false. As figure 1 shows, the f1-score of tests with the contemporary CONLL-data are higher than those for the DBNL-data, but lower than those for both the *Staten Generaal* and 19th century newspaper data.

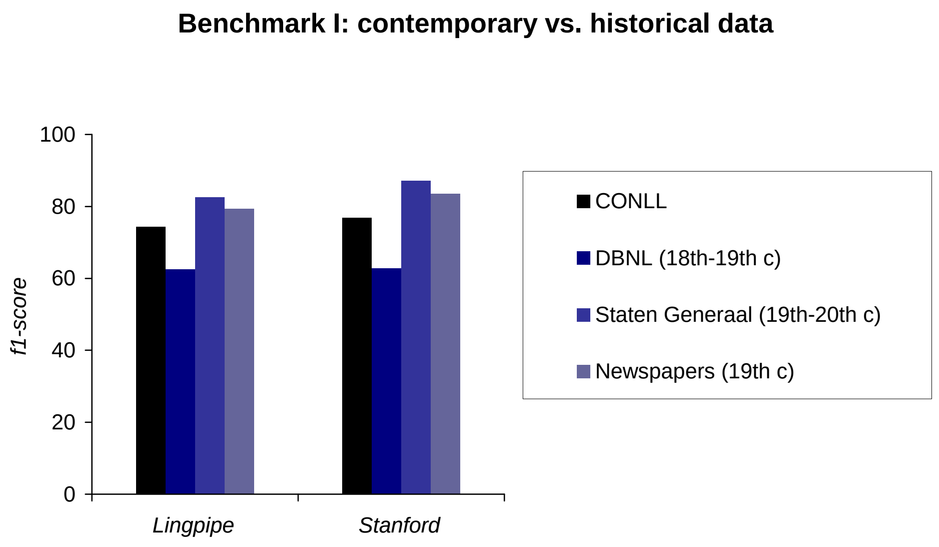


Figure 1: Performance of both tools with contemporary and historical data. Training file size is kept constant at ~250,000 words with the exception of the Newspaper dataset (~160,000 words). Test files are ~94,000 words.

[3] www.cnts.ua.ac.be/conll2002/ner/

Benchmark test II: maximally possible f1-scores

In order to understand what the maximal possible performance of both tools is, we performed two tests. In a first test, we used a test file that was identical to the training file. We repeated this setup in a second test, but added a gazetteer list of all NE's from the test file. The results show that it is possible to obtain an almost perfect f1-score if the test- and trainingdata overlap completely.

The results are shown in figure 2. Lingpipe scores reach around 99%, while Stanford performs even better, sometimes reaching 100%. Both tools perform slightly better with the gazetteer list (+0.5%).

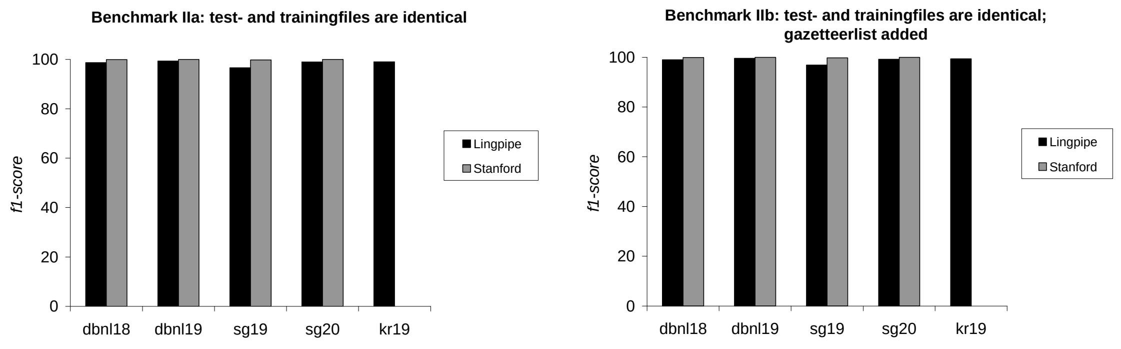
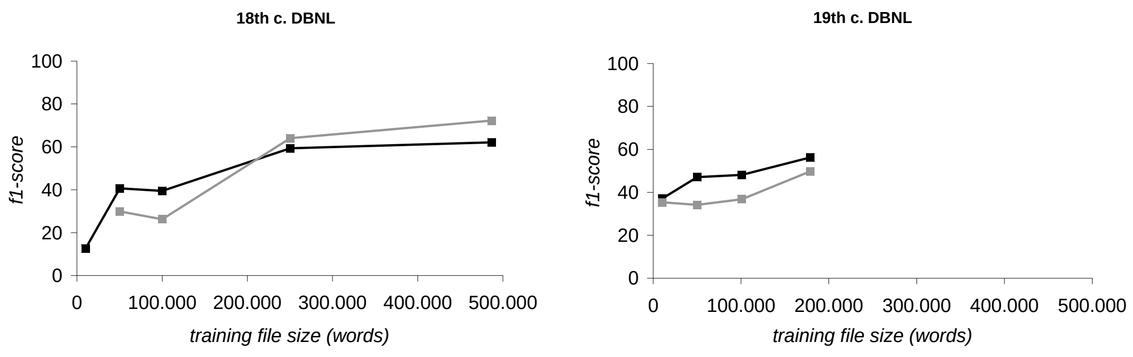


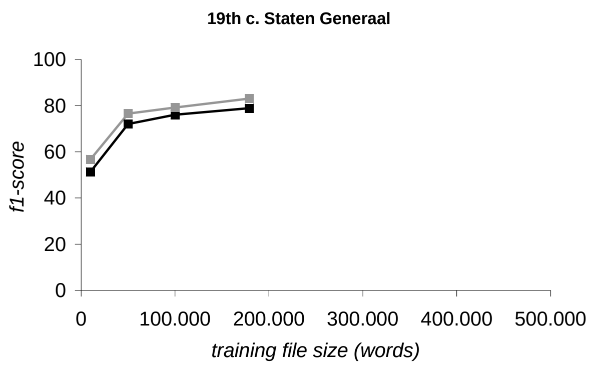
Figure 2: Performance of both tools with a test file identical to the training file. 2a) no gazetteerlist, 2b) gazetteerlist with all NE's from the test file.

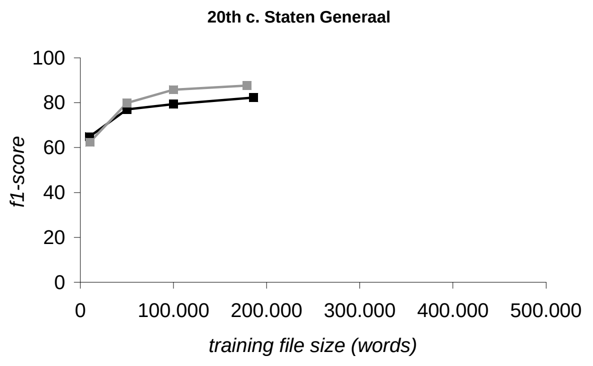
Test 1: effect of training file size

In general, a larger training file leads to a better tool performance, although the increase in performance becomes less strong with growing training file sizes. We tested with training files up to 500,000 words.

There are large differences between the various datasets, both in absolute f1-scores and in the effect of the training file size on the scores. There is not much difference between the tools, although Stanford performs slightly better in general. The DBNL is an exception: Lingpipe performs better with smaller training files, while Stanford seems to do better when training file sizes increase.







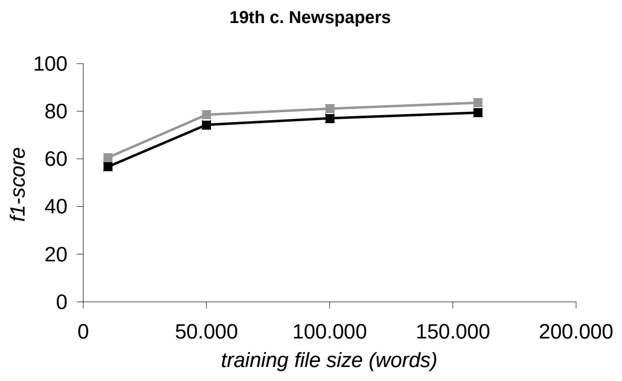
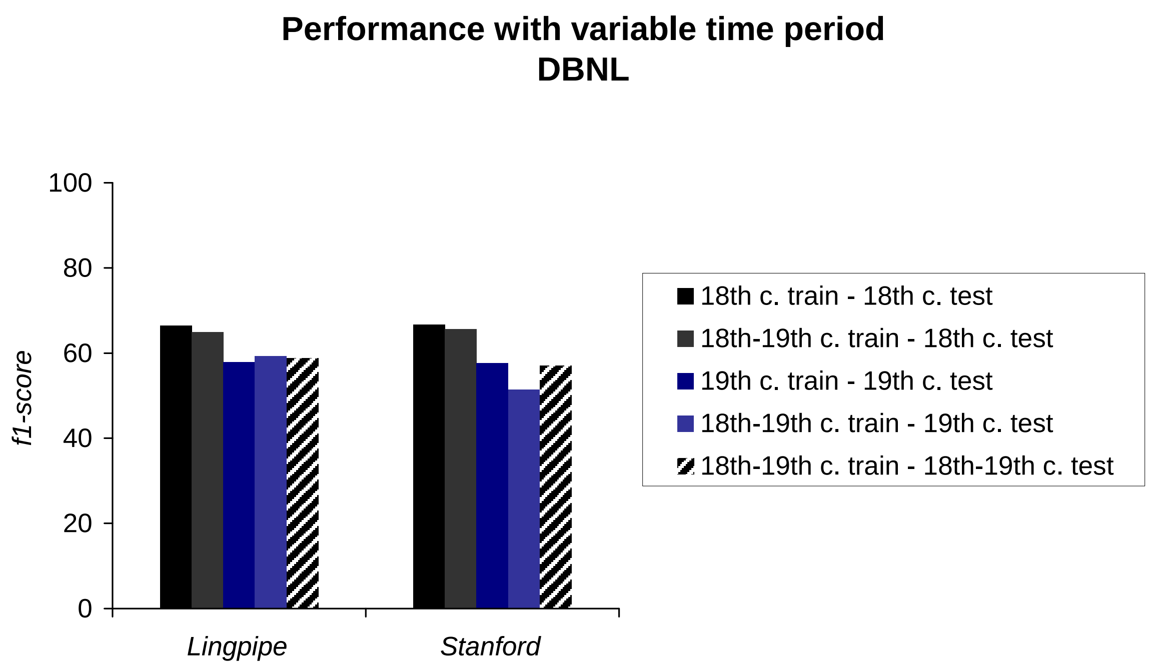


Figure 3: Performances of both tools with training files of different sizes. Black: Lingpipe; gray: Stanford. The size of the test file is kept constant (~94,000 words).

Test 2: effect of time period

In a second series of tests, we studied whether tool performance would increase if the tool is trained with a training file from the same time period as the test file. That is: given a dataset, is it useful to divide the data per century and then train and run the tools?

As figure 4 shows, such an approach does lead to a better performance, although the effect is small. In all cases, performance was better when the data was divided per time period. Also, performance increased when the tool was trained on data from the same century as the test file.



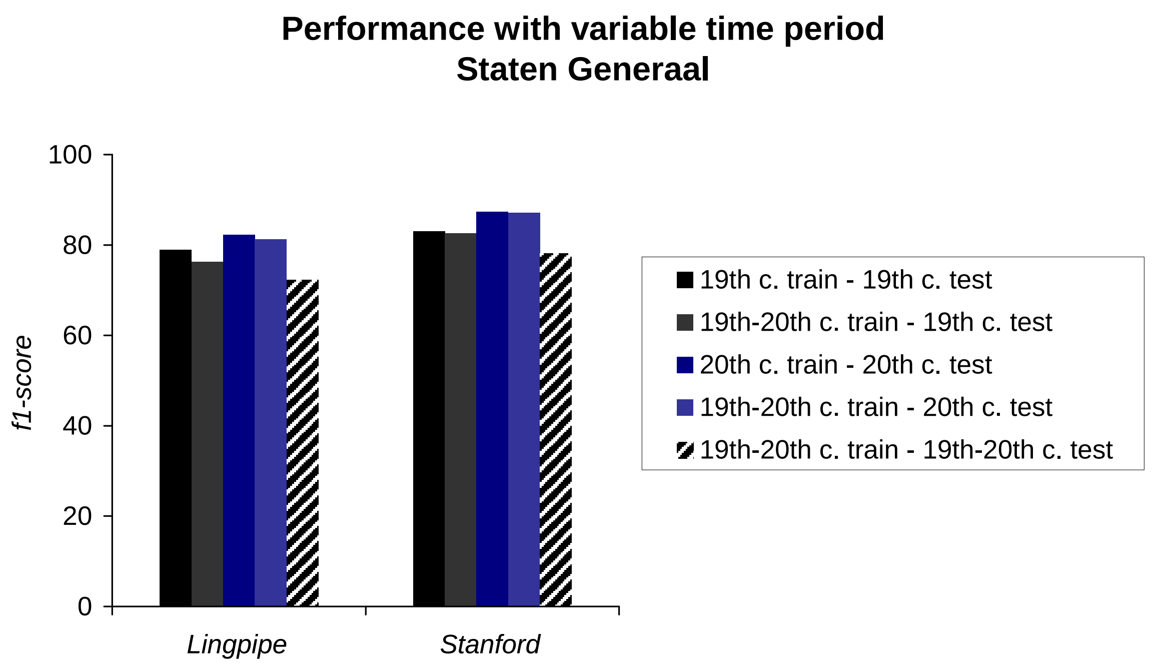


Figure 4: The effect of training and testing with variable data sets, divided per century.

Test 3: genre effects

How well do both tools perform when trained with one particular type of data (e.g. fiction) and tested on another? As figure 5 shows, their performance decreases, sometimes quite dramatically. It might be concluded from these results that tools are relatively sensitive to big discrepansies between the nature of the training and test material.

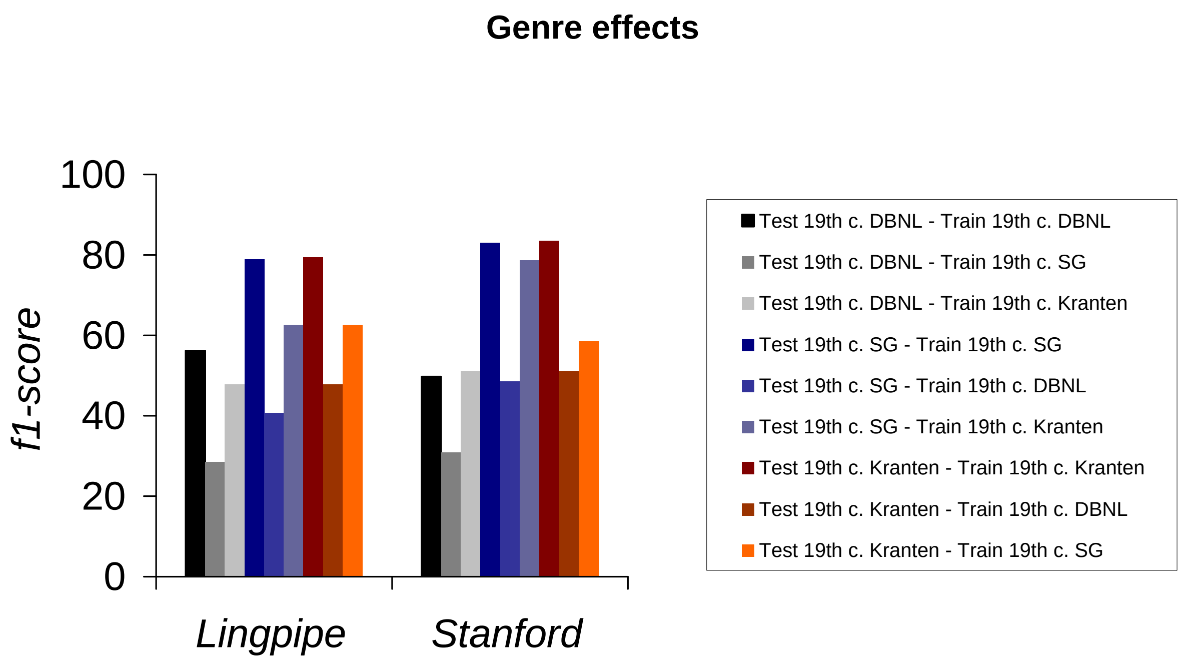
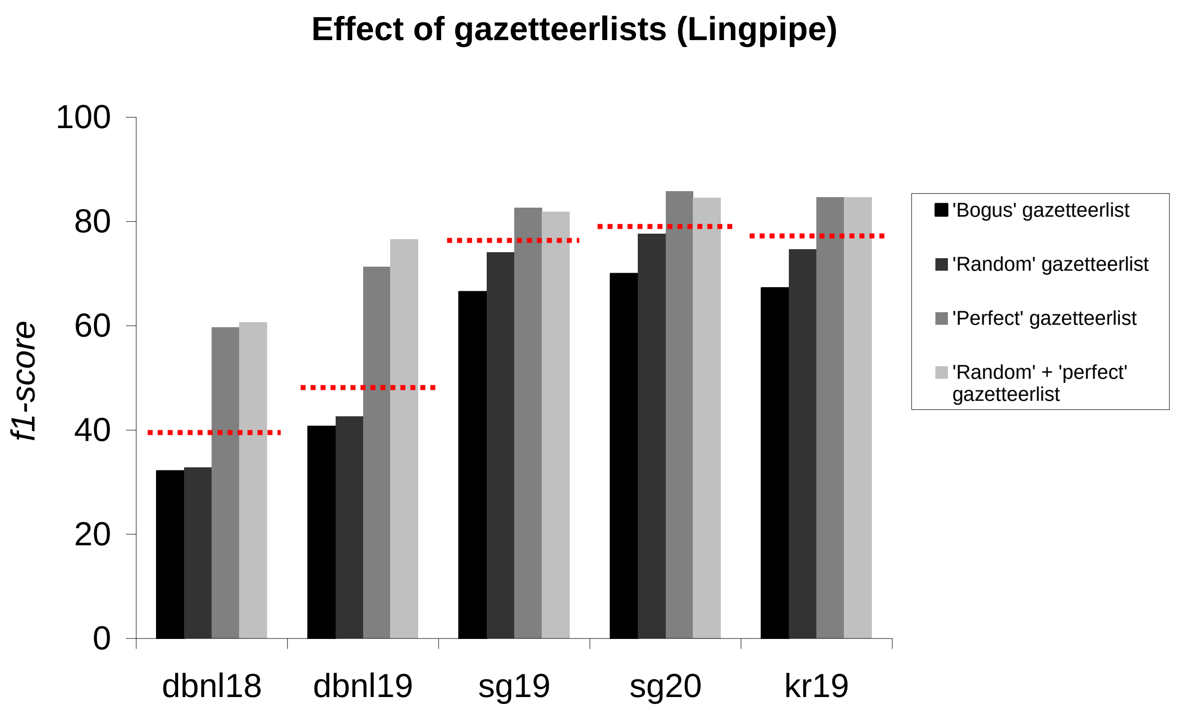


Figure 5: The effect of genre on tool performance. Training file sizes differ: DBNL>DBNL: 179,000; SG>SG: 179,000; Newspapers>Newspapers: 160,000. For the across genre tests: DBNL 273,000, SG 274,000, Newspapers 250,000.

Test 4: effect of gazetteer lists

We tested the effect on tool performance with four different gazetteer lists (figure 6). First, we added a 'bogus' list of words in the test file that are *not* named entities. This decreased performance of Lingpipe but did not harm Stanford. Second, we added a large list (~20,000 words) of random names, obtained from several namelists available on the internet. Again, this led to a worse performance of Lingpipe, but had no significant effect on Stanford's performance. Third, we added a list of all named entities from the test file. This improved the performance of both tools significantly - but not up to 100%, because it did not affect the precision of the tool. Finally, we combined the latter list and the random list, creating a situation in which all NE's from the test file are present in a (much larger) gazetteerlist. This led to results comparable to adding the perfect list.



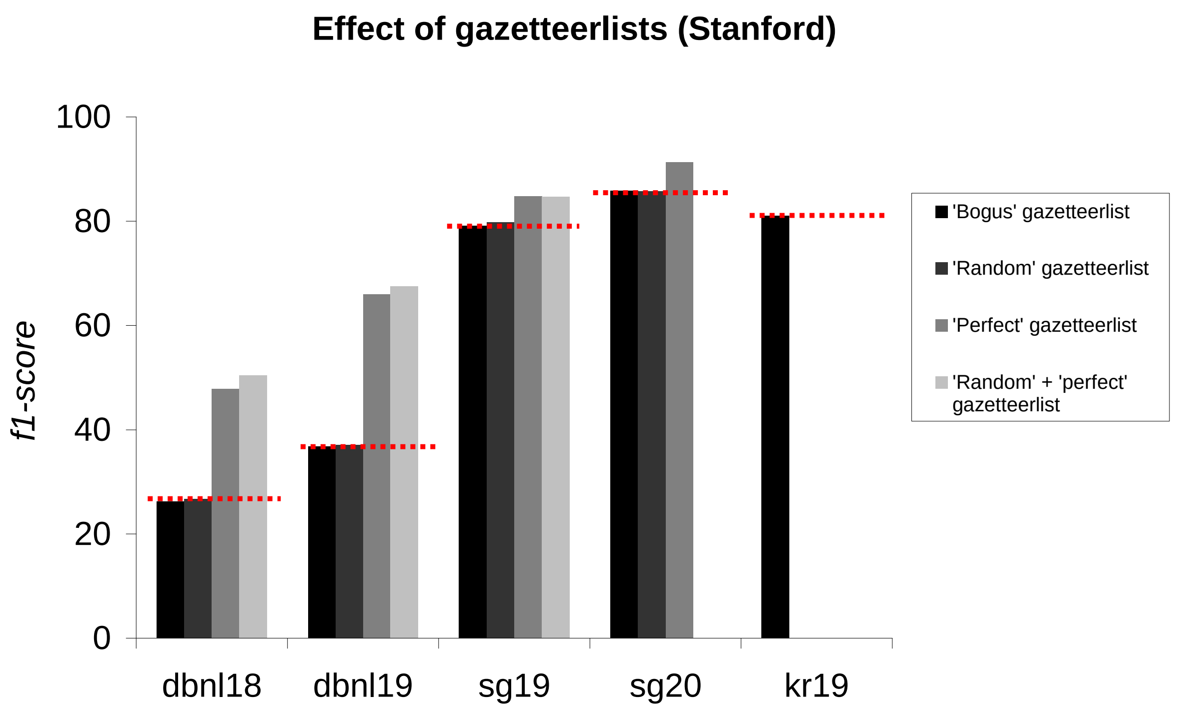


Figure 6: The effect of adding gazeteer lists to the training file on tool performance.
For each data set, the same training- and test file was used. All training files consisted of ~100,000 words. The dotted red line indicates performance without gazetteer list.

General trends

The results of the above tests show two main tendencies: (1) tool performance is different for each dataset, (2) the stronger the similarity between the training file and the test file, the better the performance of the tools. Below, we discuss these tendencies in more detail.

Tool performance is different for each dataset

The results from the performed test suggest a relationship between the nature of the data and the performance of the tool. We tested whether this 'nature' of a text can be defined with the particular measure of heterogenity.

The heterogenity of a text can be measured by listing all unique words (~types) in that text. A more heterogene text will consist of a longer list of unique words than a less heterogene one. Figure 7 shows the heterogenity of the tested data sets. In this figure, the number of unique words is plotted against the coverage of these words for the text as a whole. 0,8 coverage means that this particular number of unique words covers 80% of the total text. As figure 7 shows, there is a clear difference between the DBNL on the one hand, and the SG and Newspaper data on the other, but that there is also a difference between DBNL18 and DBNL19.

Figure 8 shows the relationship between the number of unique words and the performance of both tools on this dataset. Indeed there seems to be a link between text heterogenity and tool performance.

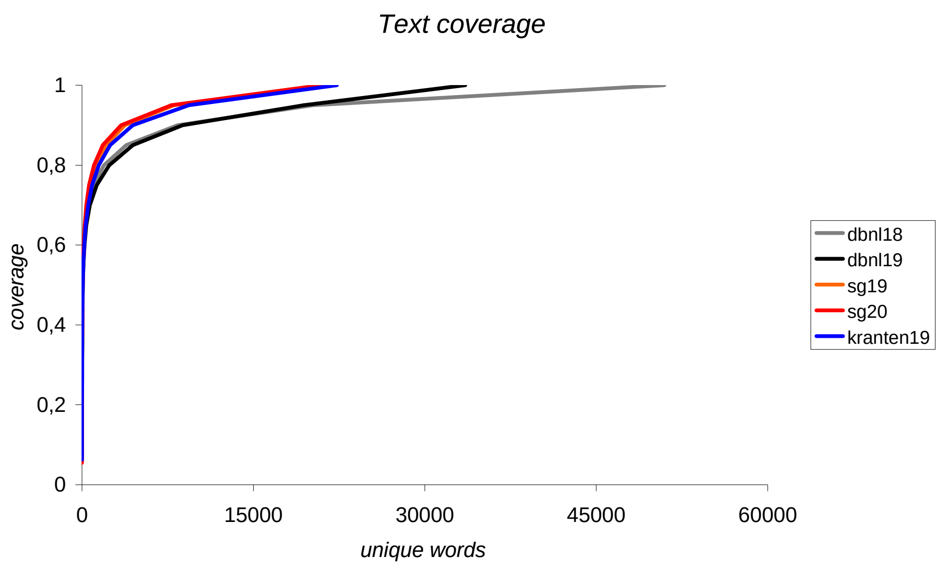


Figure 7: Coverage of the datasets.

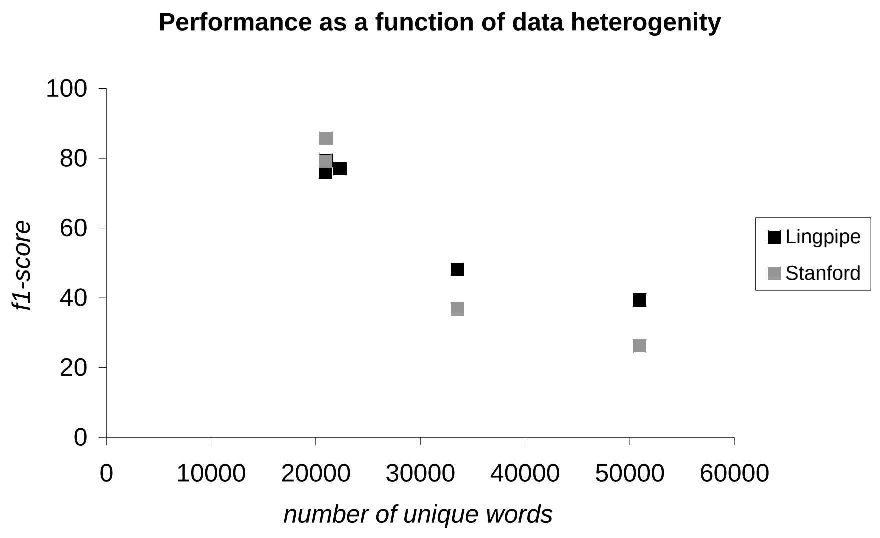
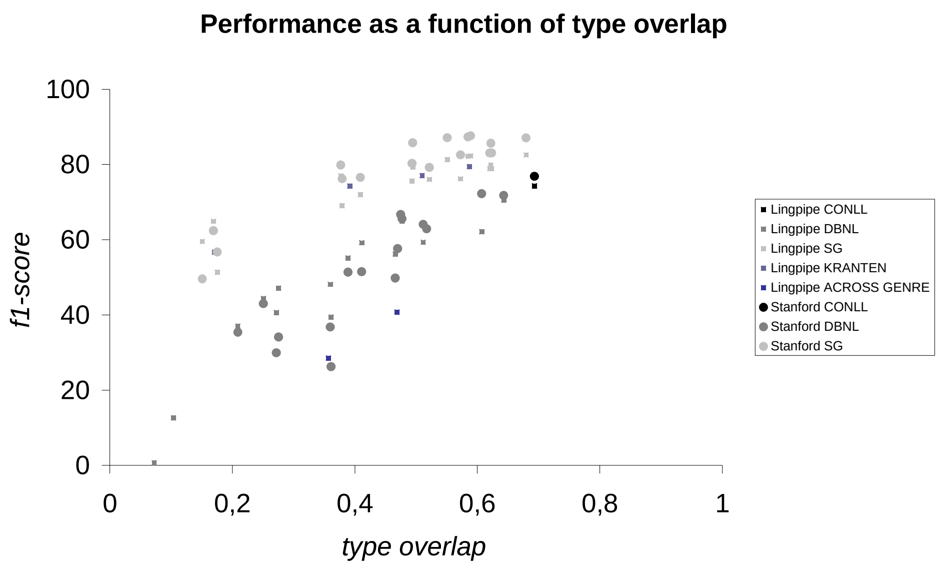


Figure 8: Performance of both tools as a function of the number of unique words.

Similarity between the training and test file

Apart from the nature of the text, the nature of the training file also has a strong effect on tool performance, particularly its similarity with the test file.

We tested the link between overlap in training- and test file and performance by listing the unique words of both files and counting the overlap between both lists. We also looked at overlap divided by the heterogenity of the text, to correct for these differences between texts. The results of both are shown in figure 9.



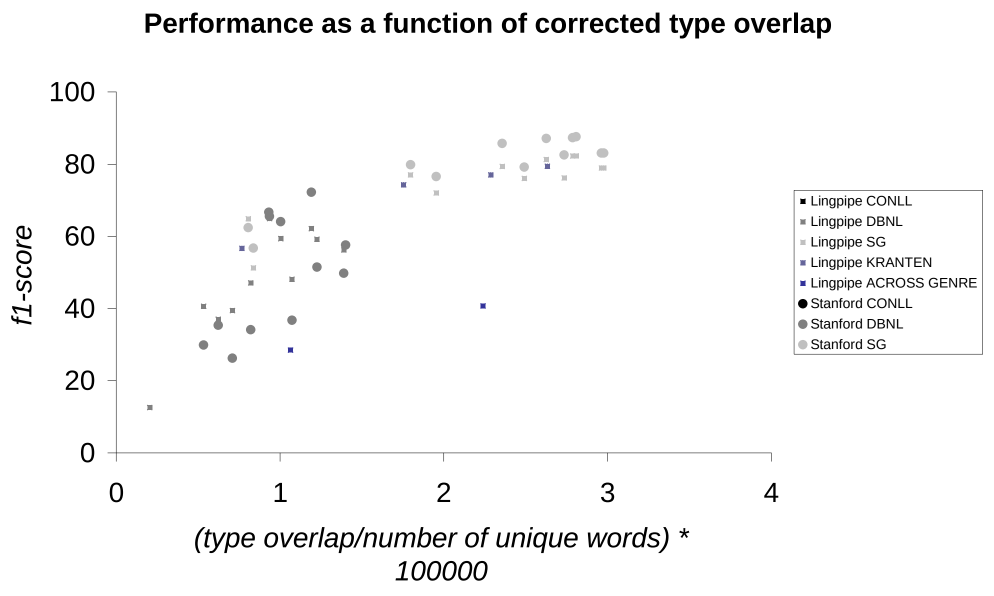


Figure 9: f1-score plotted as a function of overlap and corrected overlap between training- and test file. Corrected overlap is the type overlap divided by the total number of unique words of the text (and multiplied by 100,000).

Tool performance improves with a more randomly generated training file

In all previous tests, a training file was made by randomly selecting a series of dataset batches. This way, a text fragment either belongs to the training file or the test file.

When training files are generated at sentence level instead of text level, the average performance goes up: this method makes the training file a better representation of the test file. We tested this with ten randomly generated sets for each datasets. The results are shown in figure 10:

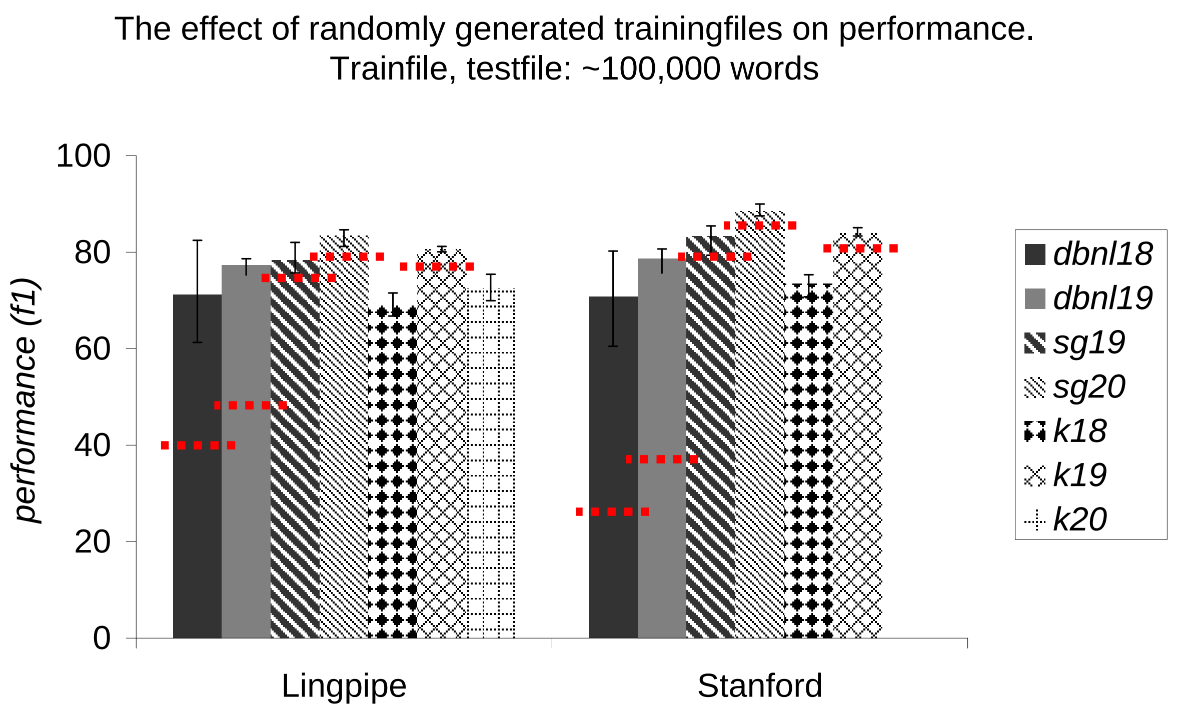
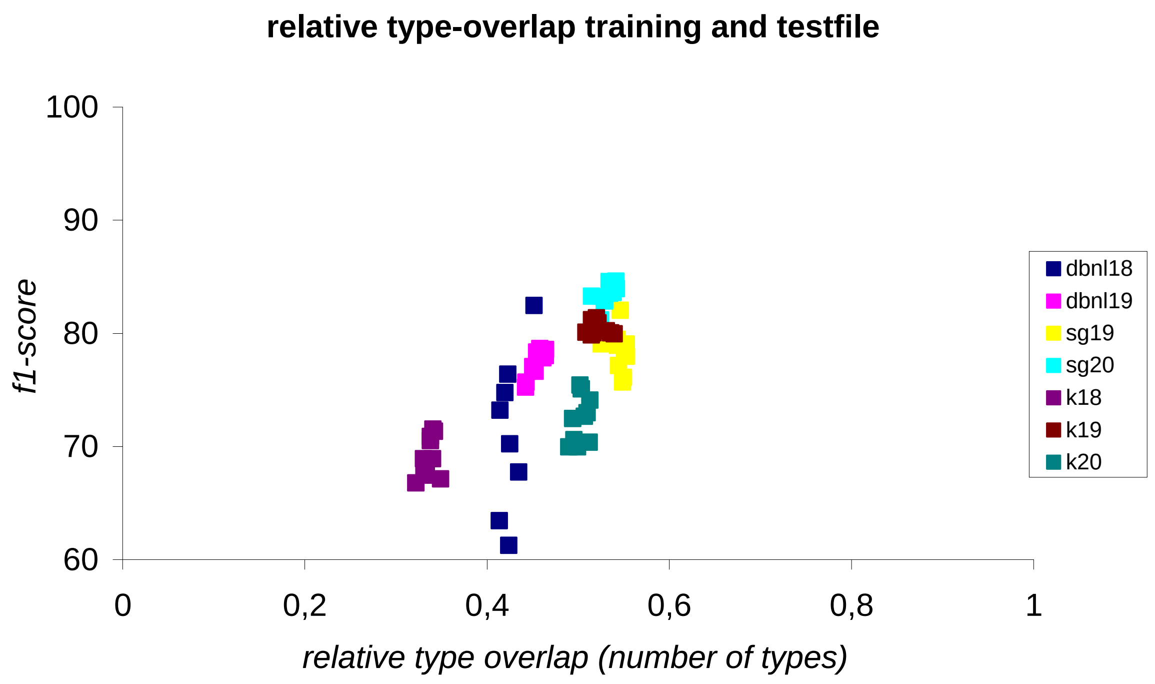


Figure 10: Averages results of ten randomly generated training- and test files. The errorbars indicate maximum and minimum scores. The red lines indicate the score from the previous series of tests, in which training files were generated at text level.

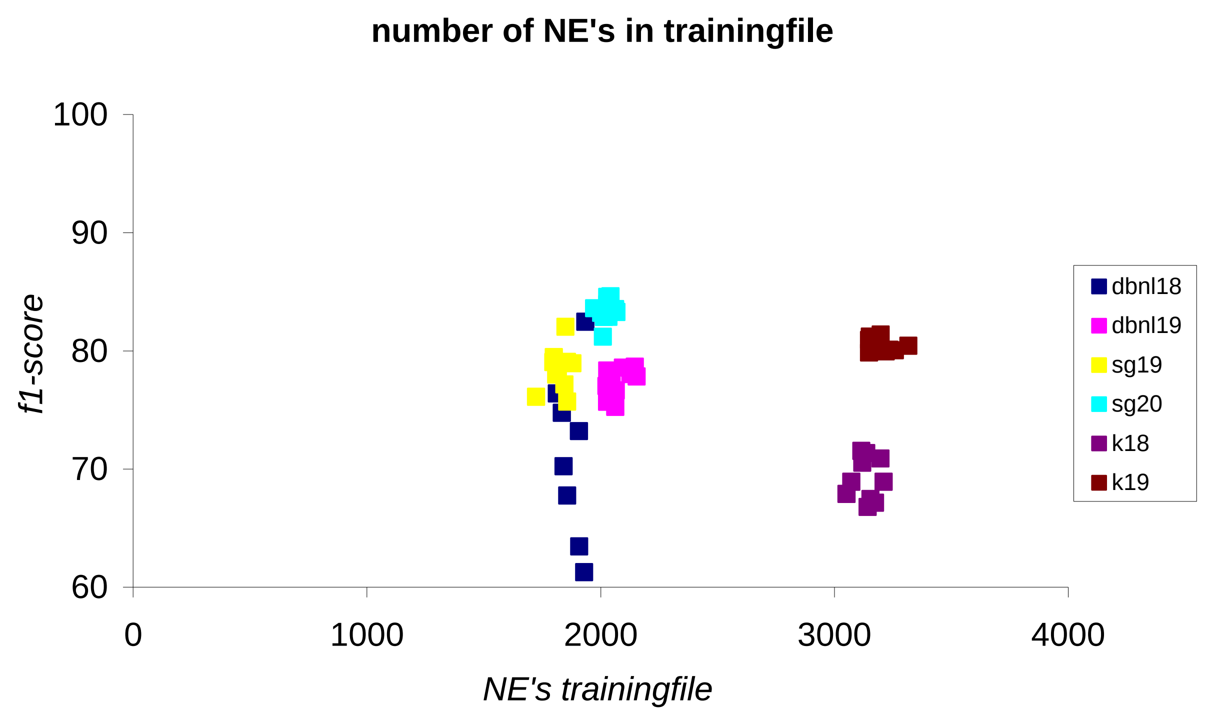
As figure 10 shows, there is some (for the DBNL18-set very much) variation in the results per dataset. We tested some possible predictors of the f1-score, but none of them seems to show a strong correlation:

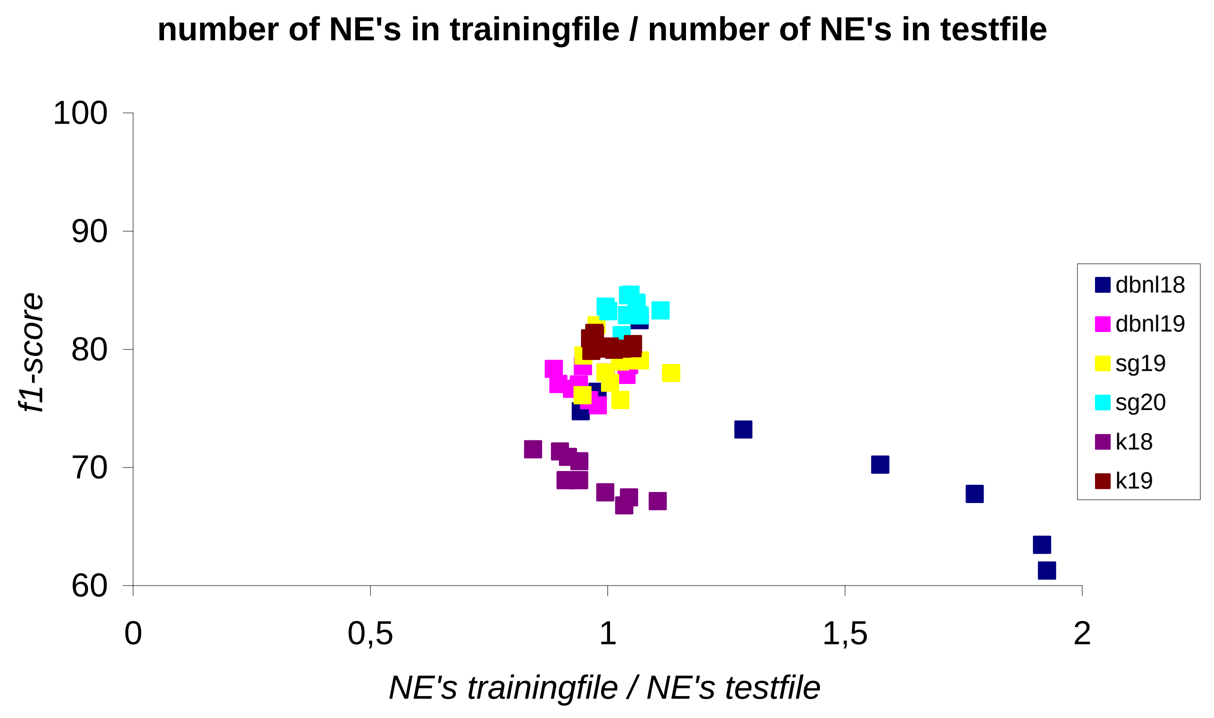
1. Relative type overlap

This is the number of types that are shared by both training file and test file, divided by the number of types in the test file.



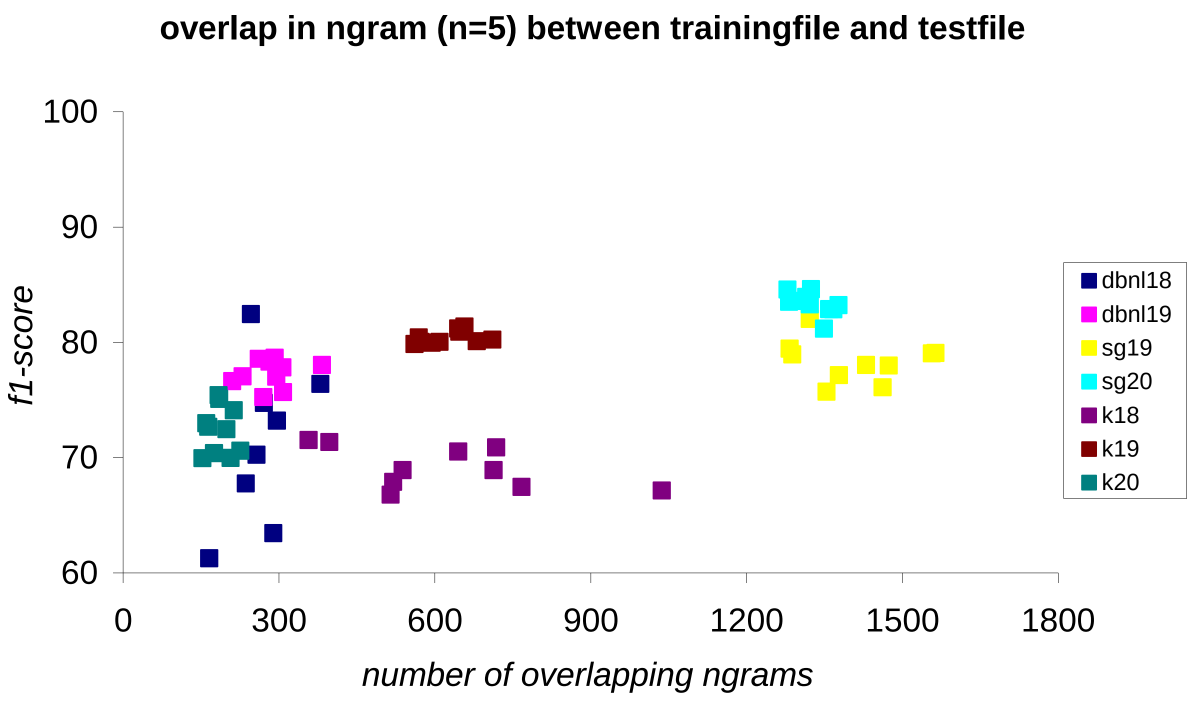
2. Absolute and relative number of NE’s in training file





3. Number of overlapping ngrams (n=5)

We listed all unique ngrams (n=5) in both training file and test file, and compared the two lists. The figure shows the absolute number of overlapping ngrams.



Part II: Evaluation of the NERT Matcher

1 Introduction

In this section we provide an evaluation of the performance of the NERT Matcher Tool on a dataset of Dutch named entities. The tool is part of the Named Entity Recognition Tool (NERT) that was created for IMPACT. It can be used in two ways: (1) to match a list of named entities against another list of named entities (e.g. as a lemmatization task) and (2) to match variants within a single list of named entities (e.g. as variant matching). This evaluation discusses its performance on both these tasks with a set of historical Dutch locations and a set of English person names:

- How well does the matcher perform on a lemmatization task of historical Dutch locations?
- How well does the matcher perform on a task of grouping variants in English person names?

2 About the matcher tool

The NERT Matcher Tool (which from here on will be referred to as 'matcher') is a command line java tool. A detailed account on its use can be found in the NERT-documentation in the IMPACT Lexicon Cookbook.

Purpose of the tool

The matcher tool is meant for matching named entity (NE) variants, both historic to historic (grouping historical spelling variants of e.g. *Amsterdam* such as *Amsteldam* and *Amstelredamme*) and historic to modern (linking a historical spelling to a present day variant, e.g. *Amstelredam* > *Amsterdam*).

How matching is done

The matcher tool compares phonetic transcriptions of NEs, and calculates the distance between them by breaking them up in chunks and by calculating the number of chunks two NEs have in common. This value is then corrected for string length and normalized on a scale from 0 – 100, with 100 being a perfect match.

The phonetic transcription takes place on the basis of a set of rules, which have to be given to the matcher. Examples of phonetic transcription are /mastrigt/ for the NE *Maastricht* and /franserepublik/ for *Fransche Republiek*. NERT comes with a set of default rules for Dutch and English.

3 Methods

Creating an evaluation dataset

We used a set of Dutch historical locations and a set of English historical person names. Both sets are Ground Truth quality. Table 1 gives an overview of the source material that was used in this dataset.

Language	Source	Type of text	Time period
Dutch	Digitale Bibliotheek der Nederlandse Letteren	prose, poetry, plays, non-fiction	1755 – 1900
	'Kort begrip der Waereld-historie voor de jeugd'	book (non-fiction)	1784
English	JISC Newspapers	newspaper articles	18th – 19th century
	'Dictionary of National Biography'	keyed namelist	1885
	'Alumni Oxonienses: The Members of the University of Oxford, 1500-1714', by J. Foster	keyed namelist	1891

Table 1. Overview of the datasets used for the evaluation.

From this material, we extracted two evaluation sets, one for locations and one for person names. This resulted in 2071 unique locations and around 15000 unique person names. The locations were manually assigned a modern lemma and the person names were manually assigned with their variants from the same set. This tagging process is described in the next two sections.

Principles of manually lemmatizing locations

Assigning a modern lemma to a location in a historical text is not as trivial as it may sound. Consider the following examples:

	historical name	modern lemma	translation
(1)	*Frankryk*	*Frankrijk*	*France*
(2)	*Fransche Ryk*	*Franse Rijk*	*French Empire*
(3)	*Venetia*	*Venetia*	*Venice*
(4)	*America*	*Amerika/America*	*America*

The location in example 1 is clear: a historical spelling of a country, and the modern lemma is straightforward. The location in example 2 is different, because this is not a the name of a country proper, but a mere description. In these cases, we have modernized its spelling. In example 3, we have a endonym location, that is, a foreign location spelled in the 'local' language, in this case Italian (the present-day Dutch word for Venice is *Venetië*), for which we used the present-day endonym as modern lemma. *America* in example 4 is ambiguous, because it can refer both to the continent and a village in the Netherlands. The modern lemma of the former is *Amerika*, while that of the latter is *America*. Our main principle was to keep lemmatizing 'formal', that is, to not use geographical considerations (e.g. to link historical *Karl-Marx-Stadt* to its present-day name *Chemnitz*), but this case shows that it was often crucial to check the location's context.

Another complication in the lemmatizing process was that no single extensive list with official spellings of locations for Dutch exists. A list of official spellings of a small subset of locations can be found online, but these only cover the main existing geographical places in the world and, in some cases, give more than one option (e.g. *Kairo* and *Caïro*). We therefore collected a set of lists and created a hierarchy of their importance:

1 *Taalunieversum* – official spelling list of Dutch locations
2 *Van Dale* – dictionary of Dutch
3 *U.S. GeoNames Search* – extensive list of locations in country's own language
4 *Wikipedia*

Principles of manually grouping person name variants

Grouping variant person names is a slippery exercise, because it depends both on the definition of 'variant' that is used and the purpose of the exercise. When using a strict definition, one would only consider two names to be variants if there is some known historical relationship between the spelling of the two variants, e.g. *William* vs. *Willyam*, and *Rodric* vs. Roderick. A looser definition would allow for any variation, regardless of its relationship being historical, e.g. *Romane* vs. *Romain* and *Frederich* vs. *Frederick*. When we look at the use of the variant lists for IMPACT, a looser approach is preferrable. In retrieval, variant lists are used for so-called 'query expansion', in which the user's single search term is expanded with marked variants. For this, it seems more helpful to show more variants of which some are false (=false positives), than to leave out possibly interesting variants and thus have them remain invisible to the user (=false negatives). We have therefore decided to take a 'loose' definition on person name variants in this evaluation. An example of variants with this approach is shown below:

Marnell	*Marinell*
Marowe	*Murrow*
Marratt	*Maret*
Marratt	*Marett*
Marratt	*Marret*
Marratt	*Marrett*
Marratt	*Marriatt*
Marratt	*Marryatt*
Philby	*Filbie*
Philcox	*Filcock*
Philip	*Philippe*
Philip	*Philippi*
Philip	*Phillyps*
Thornboro	*Thornborrow*
Thornboro	*Thornbrough*
Thornborrow	*Thornboro*
Thornborrow	*Thornbrough*
Thornbrough	*Thorneboroughe*

Matcher input

For the lemmatization of locations, the matcher had to be given lists of locations in present-day Dutch. Although, in principle, it is possible to feed the tool all possible locations in the universe, this is not the preferred way of usage, for two reasons: (1) it severely slows down the program, and (2) it will lead to an inadmissible number of false positives, since, e.g. for each Dutch village there will be some unwanted match with a Chinese or Botswana river or town. It therefore makes more sense to fine-tune the lists to the nature of the historical data. This was done by a quick trial-and-error procedure, running a test match with a small set of contemporary data, to see if there were gaps in specific types of locations, to add the proper list and to rerun the matcher. In our case, we found that one our sources held many biblical locations, and thus added a list of this type.

Matcher settings

The matcher tool uses a set of regular expressions for its conversion of NE's into a phonetic transcription. For the Dutch locations we used a set of around 30 patterns that were constructed in an trial-and-error process in an earlier stage. For English, such a set was not available. We therefore split the evaluation data of 15000 English person names into three parts, a training set (3000 names), a development set (6000 names) and a testset (3000 names). In a trial-and-error process, we constructed a set that worked reasonably well for the training and development set, and used the testset for the actual evaluation task. For both Dutch and English, the resulting patterns sets can be found in the appendix.

The matcher compares names by their phonetic transcription, counting the number of shared chunks (see the matcher documentation for a more detailed description). This means that it will return any two variants, provided they have at least two consecutive characters in common. We therefore limited the output to only those matches with a matching score of 70 or higher (with a maximum of 100) for Dutch locations. For the English person names, we tested different threshold scores.

4 Results

Locations

For each location, the matcher returned zero or more matches. If no match was returned, this meant that the matcher did not find any matching lemma with a score higher than or equal to 70. If multiple lemmata were returned, the matcher found different lemma, all with a score higher than or equal to 70. Table 2 gives an overview of the matcher's performance.

	Absolute	Percentage
Unique locations	783	-
Locations with a modern lemma present in the list	427	100
Correct matches	396	92.7
Incorrect matches	31	7,3
Single correct match	254	64.1

Table 2. Results of the matcher for Dutch locations.

Although 783 unique locations were used in the evaluation, only 427 of them turned out to have a correct modern lemma in one of the lists that were provided to the matcher. We only consider this group below.

Of these 427 locations, 396 (92.7%) were given a correct modern lemma (plus zero or more incorrect ones), while 31 (7.3%) were given either no lemma or one or more incorrect ones. Of the locations that were assigned a correct lemma, only a single, correct lemma was given in 64.1% of the cases. In the rest of the cases, the correct lemma was one of several assigned lemmata, the other ones being incorrect. Finally, if multiple lemmata were found and one of them was correct, the correct one was always the one with the highest score, with the exception of three locations, which will be discussed below.
Let us look beyond the numbers to get a better idea of the matcher's output. Table 3 gives some typical examples:

Historical location	Modern lemma	Matcher output (score)
Alkmaar	*Alkmaar*	*Alkmaar (100)*
Harderwyck	*Harderwijk*	*Harderwijk (100), Harderdijk (77)*
Babel	*Babel*	*Kabel (77), Bakel (73), Bavel (73)*
Egyptenland	*Egyptenland*	-
Kuilenburg	*Culemborg*	*Zuilenburg (76)*
Litthauwen	*Litouwen*	*Litouwen (100)*

Table 3. Examples of historical locations with according lemma and matcher output.

Many of the locations in the historical dataset had spellings similar to what is used today (e.g. *Alkmaar*). In case of a deviating spelling (e.g. *Harderwijck* and *Litthauwen*), the matcher was able to come up with a correct modern lemma in most of the cases. However, it failed when the historical spelling deviated too much from the spelling of the modern lemma (e.g. *Kuilenburg*). 'Too much' in this respect means differences in spelling that were unaccounted for in the phonetic transcription rules that were used, or a combination of both (however, note that the matcher does find the correct lemma *Culemborg*, but that it will not be shown because it's score is less than 70). When we look at more examples of incorrect or missing matches, we find the same pattern:

Historical location	modern lemma
Elve	*Elbe*
Asia	*Azië*
Amestelle	*Amstel*
Kattenwijk	*Katwijk*
Amstelredamme	*Amsterdam*
Den Haghe	*Den Haag*
Capelle op den Yssel	*Capelle aan den IJssel*
Lisbon	*Lissabon*
Neêrland	*Nederland*

The examples *Elve* and *Asia* show the troubles the Matcher has with short words. The examples *Den Haghe* and *Capelle op den Yssel* show that the Matcher could be improved by making it possible to add a list of words that can be ignored, such as determiners and adpositions such as *op, der* and *den*. The example *Neêrland* shows a missing pattern eê = > ede, a pattern that is actually quite regular for Dutch.

Person names

For the person name variants matching task, we used a slightly different evaluation task. Matches were divided in four groups: correct matches (=true positives), incorrect matches (=false positives), correct 'non-matches' (=true negatives) and incorrect 'non-matches' (=false negatives). From these numbers, a f1-score was calculated.

The F1-score is an indicator of how well the tool performs on precision and recall. Recall is about quantity: how many matches did the tool find? Precision is about quality: how many of its matches are correct? Or, in other words: recall is an indication of the number of false negatives (=missed matches), precision in an indication of the number of false positives (=incorrect matches). The F1-score is the harmonic mean of these measures.

We tested different threshold values and its effect on precision and recall. The threshold is the minimal score a match needs to have to be shown by the tool, and this parameter can be set manually.
5965 unique person names were used. This resulted in 1446 matches.

Score thres-hold	Correct matches (true positives)	Incorrect matches (false positives)	Correct 'non-matches' (true negatives)	Incorrect 'non-matches' (false negatives)	Precision	Recall	F1
65	1371	1284	3245	65	0.52	0.95	0.67
70	1338	748	3786	93	0.73	0.77	0.75
75	1113	412	4174	266	0.73	0.77	0.75
80	911	212	4419	423	0.81	0.63	0.71
85	726	135	4533	571	0.84	0.50	0.63

Table 4. Results of the matcher for person names.

The results show that the matcher finds a significant portion of the variants, but that there are differences in performance with different settings for the minimal score threshold. A low threshold results in a relatively high number of correct matches, but a rather high number of incorrect matches. A high threshold leads to a relatively low number of incorrect matches, but to a high number of missed matches. This result makes sense: a low threshold allows for more matches with lower scores. The upside is an increase in the number of correct matches, but a side effect is that the results will be more 'polluted' with incorrect matches. The reverse is true for a high threshold: less matches, but also less 'pollution' within them. Threshold values of 70-75 seem to give a good balance between both extremes.

Let's have a more detailed look at the results. Below, some examples of incorrect and missed matches are given in the results with a threshold score of 75:

Incorrect matches:	*Missed matches:*
Aggas / Agasse	Coillard / Collard
Baden / Beaden	Collamore / Colmore
Barram / Bramah	Conall / Kennall
Beddington / Berington	Corpe / Corpre
Means / Meany	Creighton / Crichton
Peeblee / Peblis	Wallen / Wallin
Tennington / Tottington	Whytynton / Withington
Tudman / Tedman	Woollacott / Wollascott
Whichcott / Whichcord	Wrayfod / Wrayford

Some of the incorrect matches are clearly incorrect such as *Beddington/Berington* and *Tennington/Tottington*. Also, we probably would like the matcher to find the pairs *Wallen/Wallin* and *Collamore/Colmore*. Still, the examples show the difficulty in defining which names should be considered variants, and which should not, when using a loose definition of 'variance'. For example, in the manually tagged data set, *Allport/Alporte* are marked as variants, but *Aggas/Agasse* are not. Similarly, *Chary/Keary* are tagged as variants, but *Baden/Beaden* are not.

5 Discussion

We tested the NERT matcher tool on a lemmatizing task for Dutch locations and a matching task of English person names. For locations, the matcher performed well, generating almost 93% correct matches. For person names, we were able to generate F1-scores up to 0.75, with a maximum precision of 0.84 and a maximum recall of 0.95.

One of the purposes of running the evaluation of the matcher was to see how well the tool performs on a task of lemmatizing historical locations. It can be said that, on the basis of these results, it performs this task well. The other purpose of the evaluation was to see how the matcher would perform on a task of grouping variants of historical person names. This task turned out to be more complex. First, whereas the lemmatization task of locations is relatively clear, defining whether two person names are variants is more difficult, and this has its impact on the construction of the phonetic transcription rules that are used within the tool. However, given this difficulty, we found that the tool works reasonably well for person name matching, too, especially because its output can be adjusted according to the user's wishes using the threshold score parameter. If more importance is put on having a clean, small set of correct matches and less on matches that have been missed, this parameter can be set high. If the user rather has a large collection of matches and does not mind that this set contains some errors, the parameter can be set low. A 'medium' setting (70-75) leads to a balance between the two extremes.

Appendix A: phonetic transcription rules for Dutch locations

Rewrite rules used for the conversion from Dutch NE’s into phonetic transcriptions in the evaluation. The rules are applied in the presented order. Note that a void right hand part of the rule means ‘remove’.

```
\s+jr\.?|\s+junior|\s+sr\.?|\s+senior|\s+zn|,\s+\w+z\b=>
\s-\s=>
ks=>x
n\b|'s\b|`s\b|’s\b=>
sch([^aeiouy])=>s$1
sche?\b=>s
^'s|^`s|^’s=>
tz=>z
st\s*\.|sint|santo|saint =>
\W=>
(\w)\1=>$1
eij|ij|ei|y|ey=>Y
(u|a|o)e=>$1
ch=>g
ou|au=>u
z=>s
d\b=>t
ck=>k
uw=>w
ce=>se
ci=>si
ca=>ka
co=>ko
cu=>ku
c=>k
ph=>f
th=>t
j=>i
(b|d|f|g|k|l|m|n|p|r|s|t|v|w|x|z)h=>$1
```

Appendix B: phonetic transcription rules for English person names

Rewrite rules used for the conversion from English NE’s into phonetic transcriptions in the evaluation. The rules are applied in the presented order. Note that a void right hand part of the rule means ‘remove’.

```
æ=>e
m'=>mac
^mc=>mac
ſ=>s
^o'=>
's\b|`s\b|’s=>
\W=>
(\w)\1=>$1
(eus|ius|us)$=>
([abdfghijklmnopqrstuvwxyz])e\b=>$1
(eigh|ei|ey|ie|y|i)=>i
s\b=>
(sz|z|s)=>s
([aeou]r)\b=>er
(ph|f|v)=>f
gh=>g
hn=>n
h$=>
([ieu]l)\b=>el
ges=>gs
c([eiy])=>s$1
c([aou])=>k$1
(ck|ch|c|k)$=>k
ck=>k
ch([ao])=>k$1
(th|t|d)$=>t
mps=>ms
([aeoui])mb([aeiou])=>$1mm$2
([wrtpsdfghjklzxcvbnm])=>$1$1
```