Named Entity Recognition on Dutch Parliamentary Documents using Frog

Ragger Jonkers
10542604

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
dhr. dr. M.J. Marx

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 24th, 2016

Contents

Abstract

A Dutch natural language processing system, Frog, is used to recognize named entities in Dutch parliamentary documents, as well is assigning a type to the retrieved named entities. Its performance is evaluated by comparing results on the CoNLL-2002 test sets to the parliamentary documents test set, for which a newly annotated corpus has been constructed. The most difficult task seems to be type prediction of a named entity, for which, consequently, a type reclassification step is provided, in which majority voting and a parliamentary gazetteer that contains types for the most occurring named entities in the domain is used. These methods have shown an improvement in type precision, which, together with high recall, makes Frog perfectly suitable for a suggested query-based system that requires all occurring named entities per document.

1 Introduction

Named Entity Recognition(NER) is a technique used in Information Retrieval which can be described as the automatic retrieval of *named entities*(NE's) such as persons, locations and organizations in text documents, as well as determination of said types (Tjong Kim Sang & De Meulder, 2003). Different methods have been tried over the last two decades, most of them language dependent, due to the fact that language features are used as indicators, yet grammar and vocabulary differ greatly across languages. While near-human performing NER systems have already been developed for English (Ratinov & Roth, 2009) (Chieu & Ng, 2002) , this does not appear to be the case for Dutch. The reason for this being that most natural language applications are confined to the language in the field of study, for which English is predominant.

Although progress has been made, especially for English, several encountered problems are not paired with unanimously accepted solutions. One might for example reason that a large gazetteer, which is actively updated, could be used in the retrieval of entities. While good results have been found using this approach (Nadeau, Turney, & Matwin, 2006), it cannot be carried out into large scale applications by itself, because the completeness of the gazetteer and the application run time are inversely proportional to each other. While small gazetteers will affect recall negatively, comprehensive gazetteers require more time to be searched through, even with hashing techniques. Secondly, ambiguity is conceived as the principal difficulty for type assignment: An entity can be of different role, albeit completely identical in appearance. The following Dutch sentence contains the entity *Ford* associated with three distinct types: person, organization and product (or miscellaneous) respectively:

*"Henry **Ford**, oprichter van het bedrijf **Ford**, ging zelf ook altijd in een **Ford** naar z'n werk."*

Humans are able to quite easily disambiguate an entity, even though elucidation of choice can be difficult. Therefore, computer systems are set up with an arduous task that automates this decision process.

NER is generally applied to a specific domain for which the task is optimized, such as finding chemical NE's (Rocktäschel, Weidlich, & Leser, 2012). In this paper NER is applied in the domain of Dutch parliamentary items, which are for example resolutions proposed or letters written by De Eerste Kamer (the Senate) and De Tweede Kamer (the House of Representatives) to the government. A suggested goal system that utilizes the acquired NE's, notifies interested parties of any new, relevant parliamentary items. The answer to the question of how to effectively acquire these NE's with their corresponding type for the suggested system, will be sought after. A straightforward implementation of this system with an example query is revealed in section A. This is an improvement to a baseline string search, because the search space only consists of NE's, which vastly increases run time. On top of that, unknown strings are not retrieved, nor do the retrieved strings contain meaning, as they hold no type information.

A relatively new parser, Frog[1] (Bosch, Busser, Canisius, & Daelemans, 2007), can apply a multitude of natural language processing techniques on a text. Frog will be examined thoroughly in section 2. Hitherto no scientific evaluations of the latest iteration of the system have been published. For this reason an evaluation of the NER-task of Frog for the domain of parliamentary items has been performed and results have been compared to those achieved on the CoNLL-2002 dataset, which was used in the CoNLL-2002 shared task[2]. Subsequently, a method will be shown for the improvement of entity *type* assignment using majority voting and a parliamentary gazetteer.

I will try to answer the main question by answering the corresponding subquestions:

RQ1 How can the Named Entities in parliamentary items be acquired for the suggested system described in section A?

[1] `https://languagemachines.github.io/frog/`

[2] `http://www.cnts.ua.ac.be/conll2002/ner/`

RQ2 Is Frog suitable for the given NER task?

1. How does Frog perform on the retrieval of entities?
2. How does Frog perform on type assignment of retrieved entities?

RQ3 What domain independent improvements regarding type classification can be made for other NER tasks?

1.1 Overview of thesis

At first research that has been done on this topic, is looked into in section 2, which will provide a more in-depth look at Frog and its NER module. Secondly, the approach of evaluating Frog on parliamentary items is described in section 3, as well as the procedure of the error analysis and how improvement of the entity type assignment was attempted. This approach is followed by the results in section 4, which will be used to form a conclusion, apparent in section 5, about the suitability of Frog for the main task in addition to possible enhancements. A discussion of approach in section 6 will conclude this paper.

2 Related work

Named-entity recognition(NER) comprises two tasks. Retrieval of NE's is regarded as the primary task, and type classification of retrieved NE's as the secondary task (Buitinck & Marx, 2012). An averaged perceptron is used for both stages, each consisting of its own feature parameters. The benefit of two-staged NER, in which a different algorithm per stage is used, lies in the opportunity of optimizing each algorithm individually.

Recently, the most prevalent implementations of NER apply a supervised machine learning approach (Desmet & Hoste, 2014) (Buitinck & Marx, 2012) (Graus, Odijk, Tsagkias, Weerkamp, & De Rijke, 2014). An annotated corpus is required for apprehension of features (for instance syntax and context) that indicate the occurrence of an NE, as well as the type. This is state of the art as opposed to handcrafted rules (Zhou & Su, 2002). Rule-based systems, however, are not completely disregarded mainly "... due to the associated transparency of the internal system state, which leads to better explainability of errors." (Chiticariu, Krishnamurthy, Li, Reiss, & Vaithyanathan, 2010). The downside of learning off an annotated corpus is the large training corpus that is required, along with domain resemblance to the goal domain of the NER application. Currently, the only Dutch corpus that was made publicly and freely available, is the CoNLL-2002 data set of news items that appeared in the Belgian newspaper *De Morgen*. The data set contains 301,418 tokens with annotated Part-of-Speech(PoS) and IOB-tag[3] hyphenated with entity type. IOB, short for inside-outside-beginning, is an indicator for a token to be interpreted as the beginning of an entity consisting of multiple tokens; second or one thereafter in the tag sequence; or outside, meaning not an entity on its own or part of one. When spoken of an *entity tag* in this thesis, the complete tag (IOB+type) is referred to, which can be understood from the visualization in figure 1.

A variety of classifier types can be chosen from, such as a memory-based learning classifier, a support-vector machine or a conditional random field. No significant results are achieved by a combination of different classifiers, subsequently classifying based on weighted votes, as is shown by an in-depth evaluation of an ensemble of classifiers for Dutch (Desmet & Hoste, 2014). Higher precision and recall are found by the optimization of features in a single classifier.

[3] https://en.wikipedia.org/wiki/Inside_Outside_Beginning

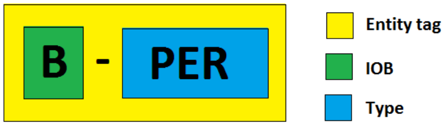
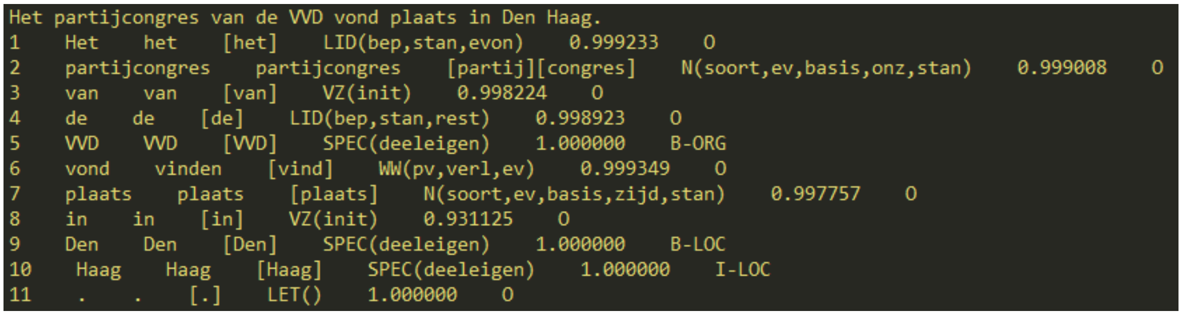


Figure 1: The components of the entity tag



```
Het partijcongres van de VVD vond plaats in Den Haag.
1    Het    het    [het]    LID(bep,stan,evon)    0.999233    O
2    partijcongres    partijcongres    [partij][congres]    N(soort,ev,basis,onz,stan)    0.999008    O
3    van    van    [van]    VZ(init)    0.998224    O
4    de    de    [de]    LID(bep,stan,rest)    0.998923    O
5    VVD    VVD    [VVD]    SPEC(deeleigen)    1.000000    B-ORG
6    vond    vinden    [vind]    WW(pv,verl,ev)    0.999349    O
7    plaats    plaats    [plaats]    N(soort,ev,basis,zijd,stan)    0.997757    O
8    in    in    [in]    VZ(init)    0.931125    O
9    Den    Den    [Den]    SPEC(deeleigen)    1.000000    B-LOC
10    Haag    Haag    [Haag]    SPEC(deeleigen)    1.000000    I-LOC
11    .    .    [.]    LET()    1.000000    O
```

Figure 2: The raw output of a processed Dutch sentence using Frog

Likewise an *unsupervised* machine learning approach has been suggested, for which no manual construction of gazetteers or annotated corpora is required (Kazama & Torisawa, 2008). While gazetteers are considered the most precise for NER, maintaining large dictionaries can be expensive. Clustering of verb-multi-noun dependencies allows for the automatic formation of gazetteers, avoiding said issue.

Frog is an expansion to TADPOLE (Bosch et al., 2007). The system is modular, composed of tokenization, lemmatization, chunking and morphological segmentation of word tokens. In addition NE's are detected in Dutch texts using a memory-based tagger (Daelemans, Zavrel, Van den Bosch, & Van der Sloot, 2010). Frog is designed for high accuracy, fast processing and low memory usage, qualifying itself to process numerous parliamentary items. Frog is trained on the SoNaR corpus (Oostdijk et al., 2008) consisting of one million annotated and manually verified NER labels. The NER module has not been trained on PoS, but on the contextual relation between words and entity tags directly. The assignment decision for each token is made with a lookup in memory, in which the instance base is stored. Unknown words are classified by a comparison to this instance base. The class of the instance that matches the contextual and internal features of the unknown instance the closest, is chosen. A processed Dutch sentence example is displayed in figure 2. Each token is associated with its own features, which are delimited by tabs.

3 Methodology

The following sections will describe the stages required to examine the suitability of Frog for doing NER on parliamentary items. Firstoff, a description of the data is given in section 3.1. The employment of Frog is apparent in section 3.2. Subsection 3.3 will then eloborate the preparation needed to evaluate Frog on the CoNLL-2002 data set and the set of parliamentary items. Lastly, from subsection 3.4 can be understood how reclassification of entity types was performed.

Type	Count
Parliamentary item	131906
News item	95261
Chamber inquiry	38804
Voting	20930
Chamber letter	19828
Agenda item	18950
Resolutions	4792

Figure 3: The distribution of the initial parliamentary documents

3.1 Description of the data

The dataset is a set of parliamentary documents published between 2015 and 2016 by the Dutch government that has been scraped from 1848[4]. The documents are most often used for *lobbying* which can be defined as the act of attempting to influence decisions made by officials in a government, most often legislators or members of regulatory agencies. Parliamentary documents are for example: resolutions, chamber inquiries, letters to the government and more. A distribution of different types can be seen in figure 3.1. The result of scraping these documents is that certain figures or itemization structures are textualized into words concatenated with itemization numbers, table entry titles or something alike. Nevertheless, Frog's tokenization module will most of the time seperate the tokens correctly. The documents have been exported as consistent JSON dicts (a sample can be seen in figure 4) with for each document its corresponding meta information such as ID, source, and type. The content of the item in the figure has been omitted, but it covers the dict entry in which the complete item can be found. The JSON format can effortlessly be read with Python. For domain reduction purposes, only parliamentary items are included in the data set. These items form the majority of the items and are a mix of all types, except for the news items, in which, undesirably, NE's vary greatly in domain compared to the parliamentary items.

3.2 Employment of Frog

Frog is open-source software that can be modified or redistributed under the GNU General Public License[5]. The results in this paper have been acquired by running Frog through a virtual machine on Windows. All necessary dependencies of Frog, including Frog itself, are provided by the LaMachine software distribution[6]. Frog has been launched through Python using the python-frog binding which is also included in the virtual machine.

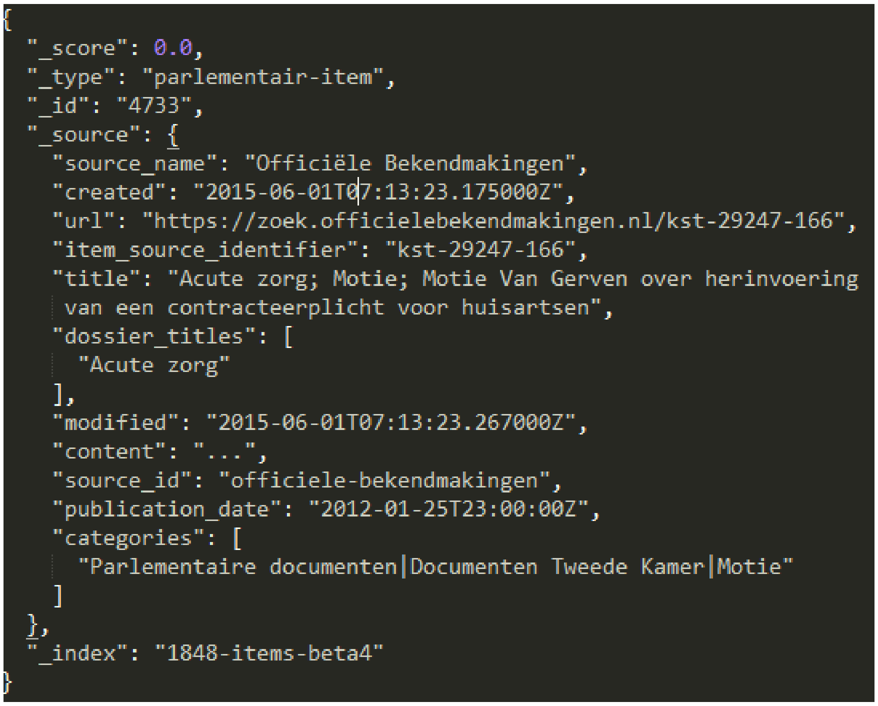
3.3 Evaluation preparation

Differences in format and annotation guidelines had to be overcome in order to make an evaluation. Additionally, a golden standard for parliamentary items has been created.

[4] https://1848.nl/

[5] http://www.gnu.org/copyleft/gpl.html

[6] https://proycon.github.io/LaMachine/



```
{
  "_score": 0.0,
  "_type": "parlementair-item",
  "_id": "4733",
  "_source": {
    "source_name": "Officiële Bekendmakingen",
    "created": "2015-06-01T07:13:23.175000Z",
    "url": "https://zoek.officielebekendmakingen.nl/kst-29247-166",
    "item_source_identifier": "kst-29247-166",
    "title": "Acute zorg; Motie; Motie Van Gerven over herinvoering
     van een contracteerplicht voor huisartsen",
    "dossier_titles": [
      "Acute zorg"
    ],
    "modified": "2015-06-01T07:13:23.267000Z",
    "content": "...",
    "source_id": "officiele-bekendmakingen",
    "publication_date": "2012-01-25T23:00:00Z",
    "categories": [
      "Parlementaire documenten|Documenten Tweede Kamer|Motie"
    ]
  },
  "_index": "1848-items-beta4"
}
```

Figure 4: A sample of a parliamentary item represented as a JSON dict

	CoNLL	Golden
LOC	543	101
ORG	1082	628
PER	1126	370
MISC	963	415
TOTAL	3714	1514

Figure 5: Type occurrences in the CoNLL and Golden Standard set

3.3.1 Format difference

The CoNLL-2002 data set uses the following format:

$$TOKEN\,[whitespace]\,POS\,[whitespace]\,ENTITY$$

Every line contains three elements, each seperated by a whitespace: the token, which is an unprocessed word from a sentence, its corresponding part-of-speech, and the entity tag that belongs to the token. The FoLiA format (Van Gompel & Reynaert, 2013), with tab-delimited columns instead of whitespaces looks similar, but comes without additional columns of token information. To release Frog on CoNLL, the test set first has to be reformatted to its original text. This is done by removing the PoS- and entity tags, concatenating the remaining words into their original sentences thereafter. Frog chunks together multi-word entities and other words that are closely related on the same line in the output. This will make it differ from the CoNLL output, as is why the output is resplitted with python using regular expressions. Superfluous information is removed from each line. The ultimate processed text has the same format as CoNLL, which implies identical line mapping per token.

3.3.2 Guideline difference

It needs to be considered that the annotation guidelines for the SoNaR corpus[7], on which Frog is trained, differ from CoNLL. SoNaR has a wider range of entity types, with addition of EVE for event and PRO for product. These types are annotated in CoNLL as MISC. Therefore these additional types have been mapped to MISC. There is also a guideline difference regarding locational adjectives, such as ’Dutch’ or ’European’. CoNLL guidelines[8] state to annotate such an adjective as MISC, while Frog is trained to assign LOC as type. These LOC outputs of Frog have been reassigned to MISC as well.

3.3.3 Parliamentary items: Golden Standard

A test set of 35,338 tokens has been constructed as a golden standard to compare Frog-processed parliamentary items to. The most common occurring entities are found in figure 6. As basis, Frog output of the tokens used for the Golden Standard set is formatted identical to CoNLL, keeping PoS as determined by Frog. All entity tags are changed to *O* beforehand to prevent bias in the annotation process. Subsequently, all entity tags have been manually assigned pursuing the CoNLL annotation guidelines. Even though the size of the

[7] `http://www.lt3.ugent.be/media/uploads/publications/2009/AnnotatierichtlijnenNE20091019.pdf`

[8] `http://www.cnts.ua.ac.be/conll2003/ner/annotation.txt`

NE	Count
Kamer	57
Comissie	42
Nederland	41
Minister	38
EU	29
Staten-Generaal	24
Europese	21
Nederlandse	20
Hoogers	19
Schiphol	16

Figure 6: Ten most occurring NE's in the Golden Standard set

Golden Standard is equal to that of CoNLL-2002 *ned.testa*[9], a substantually smaller amount of entities is present as is revealed in figure 5. This Golden Standard set is publicly available for download[10].

3.4 Reclassification of entity types

Reclassification of entity types has been done in two steps. Majority voting is only applicable when the gazetteer step has had no success.

3.4.1 Gazetteer

Firstly, all found entities are re-evaluated by search in a gazetteer in the domain of parliamentary items. This gazetteer has been constructed with a combination of automatic extraction of entities with Frog and manual annotation of the correct type. The automatic extraction is performed on the training set of thousand parliamentary items. These parliamentary items have been processed with Frog resulting in a dump that contains the occurrence counts per entity over all items. The dump has then been sorted descendingly, giving annotation priority to the most common entities. The effectiveness of this step is dependent on:

1. The amount of entities manually annoted in the gazetteer
2. The recall performance of Frog
3. The size of the training set
4. The relevance of the training set.

The amount of entries in the gazetteer has been kept modest by only including the three hundred most common entities. It needs to be acknowledged that entities that do not appear in the gazetteer cannot be reclassified in this step.

[9] `http://www.cnts.ua.ac.be/conll2002/ner/data/ned.testa`

[10] `https://github.com/Poezedoez/NER/blob/master/Code/data/lobby/golden_standard`

3.4.2 Majority voting

In the case of unknown words the contextual word relations may indicate a type correctly, however, with multiple occurrences of the same entity throughout a text, context differs from time to time. As a result an unambiguous entity can be tagged correctly as a PER 90% of the time, but have an incorrect type assigned for the remaining 10%. This is expected to be resolved using majority voting. In the case where a certain type is dominant over a minority, this minority is reclassified as the dominant type. To retrieve information regarding the dominant types per entity, a thousand parliamentary items, including items in the test set, have been processed by Frog. The total occurrences of the entity types have been counted over all documents. For each token in the test set that has been assigned an entity tag, majority voting has been applied. Whether an entity type is dominant for that entity is decided based on a threshold value. When the fraction of an entity type over the total amount of entity type counts is larger than the threshold value, the type is considered dominant. The formula for majority voting, as well a the complete reclassification procedure in pseudocode is apparent in algorithm 1.

4 Evaluation results

Subsection 4.1 contains the regular performance evaluation measures, such as recall, precision and F-measure. The example in subsection 4.2 will clarify how the performance measures have been obtained. The type prediction distribution in subsection 4.3 shows the varying difficulty in predicting specific types.

4.1 Performance measures

The performance evaluation for the different test sets can be seen in figure 7. The CoNLL performance is an average of both the *ned.testa* -and *ned.testb* set. The parliamentary item set is of the same size as ned.testa. The last column in the figure shows performance after reclassification (3.4) of entity types in the parliamentary items set. For all measures evaluation is done **per token**. Accuracy is obtained by binarily evaluating all tokens: if a token is predicted as an NE (regardless of type), but it turns out not to be, this token prediction is marked as incorrect. Tokens not seen as an entity while they are, idem. Low accuracy would be the result of Frog incorrectly identifying non-entities as entities, since the other way around would not hit accuracy hard due to the sparcity of entities in both test sets. Recall is measured as the proportion of retrieved tokens tagged as an entity that were annotated in the test set, again not looking at entity type. IOB precision is the precision of the first part of the entity tag (figure 1). This will measure how precisely the beginning of an entity consisting of multiple tokens has been found, or how succesfully different entities following each other have been seperated. Type precision is decided on the second part of the entity tag. Regular precision is the combination of type precision en IOB precision. F-measure is the harmonic mean of the regular precision and the recall.

4.2 Evaluation example

To examplify the performance measures, a single sentence has been evaluated in figure 8. The accuracy is $\frac{9}{10}$, because *S.A.M.* was not predicted as an entity. This miss is also noticeable in recall, to which a score of $\frac{4}{5}$ is assigned. Type -and IOB precision are only calculated for the recalled entities. Assigning the wrong type to *Kabeljauwherstelplan* makes for a type precision of $\frac{3}{4}$. Recognizing *Dijksma* as the first instead of the second token of the person entity determines the IOB precision at $\frac{3}{4}$. The last column shows the evaluation for the regular precision, which is $\frac{2}{4}$, as both type and IOB need to be correct.

```
Algorithm 1: Reclassification of entity types in Frog output using majority voting and a type gazetteer

reclassify ($frogged\_file, vote\_file, gazetteer\_file$);
Input  : Three files (path) each containing: the output of the Frogged text in FoLiA-XML format, a
         dict of the counts for each type per entity, a dict of entities and their correct type
Output: CoNLL format with reclassified types
treshold ← 0.7
type_counts ← load(vote_file)
gazetteer ← load(gazetteer_file)
foreach line in frogged_file do
    token, lemma, pos, chunk, iob+type etc. ← split(line)
    iob ← split(iob+type-tag)
    if is_entity(token) then
        new_type ← lookup(token, gazetteer)
        // if token is in gazetteer, no majority vote needed
        if new_type then
            line ← token, pos, iob+new_type
        else
            new_type ← vote_type(token, type_counts, treshold)
            if new_type then
                line ← token, pos, iob+new_type
            end
        end
    end
end
vote_type ($token, type\_counts, treshold$);
Input  : token of which to decide the type, the type_counts dictionary, and the treshold to enact on
         revoting
Output: new type for the token, none returned otherwise
new_type ← max(type_counts[token])
confidence_score ← max(type_counts[token]) / total(type_counts[token])
if confidence_score $>$ treshold then
    return new_type
end
```

	CoNLL	Parliamentary items	Parliamentary items + reclassification
Accuracy	0.950	0.973	0.974
Recall	0.845	0.906	0.908
IOB precision	0.915	0.890	0.890
Type precision	0.672	0.616	0.662
Precision	0.629	0.563	0.603
F-measure	0.720	0.694	0.725

Figure 7: Performance overview of both sets

S.A.M. Dijksma vergaderde over het Kabeljauwherstelplan in Ter Heijde.

Token	Predicted	Correct	Recalled	Type	IOB	Precise
S.A.M.	O	B-PER	×	N/A	N/A	N/A
Dijksma	B-PER	I-PER	✓	✓	×	×
vergaderde	O	O	N/A	N/A	N/A	N/A
over	O	O	N/A	N/A	N/A	N/A
het	O	O	N/A	N/A	N/A	N/A
Kabeljauwherstelplan	B-ORG	B-MISC	✓	×	✓	×
in	O	O	N/A	N/A	N/A	N/A
Ter	B-LOC	B-LOC	✓	✓	✓	✓
Heijde	I-LOC	I-LOC	✓	✓	✓	✓
.	O	O	N/A	N/A	N/A	N/A

Figure 8: Evaluation example

4.3 Type prediction

Since type precision has appeared to be relatively low, it would be interesting to analyze errors made per type. Figure 9 shows a prediction distribution for each type. It can be seen that location as a type is predicted very accurately, while prediction of the type person seems to be significantly more difficult. A plausible explanation for this could be the effectiveness of gazetteers per type. Location names such as *Den Haag*, *Amsterdam* and *Nederland* appear very frequent in parliamentary items and are more easily covered by a plain gazetteer, while person names are incredibly hard to cover in the same way. Ambiguity is, inter alia, caused by abbreviations. Aditionally, locations are almost never written shorthand. Organizations, however, especially political organs, often appear in shortened form.

5 Conclusions

Frog has turned out to be best available option in performing NER on Dutch texts. The high recall of 91% on the parliamentary items test set indicates that Frog is very capable in retrieving all relevant NE's in this domain. Type assignment, however, has appeared to be substantually more difficult, yielding an overall 62% precision. As a solution, reclassification of entity types has been proposed, and has proven to increase overall type precision up to 66%. This boost in type precision, using the parliamentary gazetteer as auxilliary, makes Frog even more suitable for the NER task of the suggested system, even though high recall is most benificient, considering incorrect type assignment can be circumvented by alternating the search query to include additional types of which the procedure is shown in section A.1.1.

Generally high recall of entities is the most important aspect of an any NER system, because the type assignment can be done in a seperate stage and individually optimized, of which the reclassification step in section 3.4 is an example. A semi-automatic gazetteer can not only be constructed for parliamentary items, but for any given domain. Additionally majority voting can be carried out, which is also domain-independent. Comparable techniques for type disambiguation, such as clustering based on Wikipedia pages (Cucerzan, 2007), allow Frog to be exported to other Dutch NER tasks.

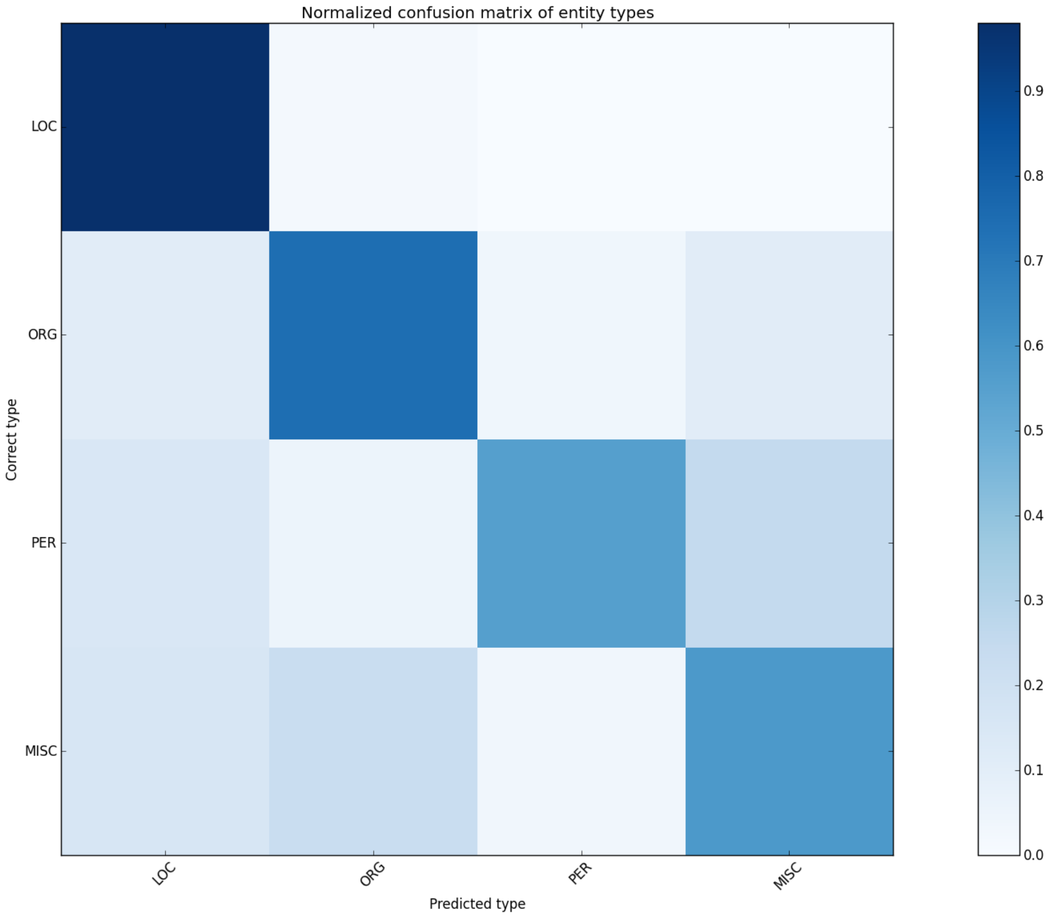


Figure 9: Prediction distribution per type

6 Discussion of approach

Although, as mentioned in section 1, gazetteers are generally not conceived as an adequate ground for a classifier purposed for large-scale applications, reclassifying previously detected entities as a secondary step only demands negligible computational resources. Hence, since the time required to construct a gazetteer with extracted entities from Frog in proportion to the performance increase it offers can be considered qualitative, this reclassification procedure appears to be very helpful.

While all of the tokens in the test set can be used in the determination of precision as performance measure in the PoS-task, only a fraction of the tokens (the NE's) can be used in the NER task. Therefore it seems reasonable that a larger test set is required to achieve higher performance measure assurance. Low density of NE's, of which many also occur multiple times, might not have challenged Frog to an ultimate extent.

Unfamiliarity with installing different external libraries and dependencies in order to employ existing NER systems has demanded sizeable time investment. Frog, however, has been an exception due to the possibility of using a virtual machine that is furnished with all necessary dependencies and libraries.
Using Python as programming language certainly has favored specific tasks such as reformatting output, reading and writing from a file, as well as string manipulation using regular expressions. Additionally, the use of JSON dictionaries to save counts and extracted entities per document –together with the provided JSON data–has facilitated data processing.

Frog itself may be improved by further expansion of the NER module. Possibly PoS tags can be added in as extra feature in the memory-based tagger. Additionally, the OpeNER module (Agerri Gascón, Cuadros Oller, Gaines, & Rigau Claramunt, 2013) could be incorperated into Frog. OpeNER is still in development, but is expected to classify using the same predefined categories as Frog with inclusion of the type *Tourist* which will refer to hotels, restaurants, monuments etc. On top of that, coreference resolution is proposed, which aims at grouping together all the mentions in the text to an NE.

7 Acknowledgements

Maarten Marx has supervised me over the course of this bachelor thesis and given intermediate feedback

Justin van Wees has provided the data

Maarten van Gompel has helped me with the set-up of python-frog

Ko van der Sloot has provided me with examples and tips of how to use Frog

Antal van den Bosch has answered some questions about the memory-based tagger of the NER module of Frog

References

Agerri Gascón, R., Cuadros Oller, M., Gaines, S., & Rigau Claramunt, G. (2013). Opener: Open polarity enhanced named entity recognition.

Bosch, A. v. d., Busser, B., Canisius, S., & Daelemans, W. (2007). An efficient memory-based morphosyntactic tagger and parser for dutch. *LOT Occasional Series*, *7*, 191–206.

Bosch, A. v. d., Daelemans, W., & Weijters, T. (1996). Morphological analysis as classification: an inductive-learning approach. *arXiv preprint cmp-lg/9607021*.

Buitinck, L., & Marx, M. (2012). Two-stage named-entity recognition using averaged perceptrons. In *Natural language processing and information systems* (pp. 171–176). Springer.

Chieu, H. L., & Ng, H. T. (2002). Named entity recognition: a maximum entropy approach using global information. In *Proceedings of the 19th international conference on computational linguistics-volume 1* (pp. 1–7).

Chiticariu, L., Krishnamurthy, R., Li, Y., Reiss, F., & Vaithyanathan, S. (2010). Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the 2010 conference on empirical methods in natural language processing* (pp. 1002–1012).

Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data. In *Emnlp-conll* (Vol. 7, pp. 708–716).

Daelemans, W., Van Den Bosch, A., & Zavrel, J. (1999). Forgetting exceptions is harmful in language learning. *Machine learning*, *34*(1-3), 11–41.

Daelemans, W., Zavrel, J., Van den Bosch, A., & Van der Sloot, K. (2010). Mbt: memory-based tagger. *Version*, *3*, 10–04.

Daelemans, W., Zavrel, J., van der Sloot, K., & Van den Bosch, A. (2004). Timbl: Tilburg memory-based learner. *Tilburg University*.

Desmet, B., & Hoste, V. (2014). Fine-grained dutch named entity recognition. *Language resources and evaluation*, *48*(2), 307–343.

Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., … Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, *165*(1), 91–134.

Graus, D., Odijk, D., Tsagkias, M., Weerkamp, W., & De Rijke, M. (2014). Semanticizing search engine queries: the university of amsterdam at the erd 2014 challenge. In *Proceedings of the first international workshop on entity recognition & disambiguation* (pp. 69–74).

Hachey, B., Radford, W., & Curran, J. R. (2011). Graph-based named entity linking with wikipedia. In *Web information system engineering–wise 2011* (pp. 213–226). Springer.

Kazama, J., & Torisawa, K. (2008). Inducing gazetteers for named entity recognition by large-scale clustering of dependency relations. *Proceedings of ACL-08: HLT*, 407–415.

Moro, A., Raganato, A., & Navigli, R. (2014). Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, *2*, 231–244.

Nadeau, D., Turney, P., & Matwin, S. (2006). Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity.

Niu, C., Li, W., Ding, J., & Srihari, R. K. (2003). A bootstrapping approach to named entity classification using successive learners. In *Proceedings of the 41st annual meeting on association for computational linguistics-volume 1* (pp. 335–342).

Oostdijk, N., Reynaert, M., Monachesi, P., Van Noord, G., Ordelman, R., Schuurman, I., & Vandeghinste, V. (2008). From d-coi to sonar: a reference corpus for dutch. In *Lrec*.

Ratinov, L., & Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the thirteenth conference on computational natural language learning* (pp. 147–155).

Rocktäschel, T., Weidlich, M., & Leser, U. (2012). Chemspot: a hybrid system for chemical named entity recognition. *Bioinformatics*, *28*(12), 1633–1640.

Siniakov, P. (2008). *Gropus an adaptive rule-based algorithm for information extraction* (Unpublished doctoral dissertation). Freie Universität Berlin.

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on natural language learning at hlt-naacl 2003-volume 4* (pp. 142–147).

Van Gompel, M., & Reynaert, M. (2013). Folia: A practical xml format for linguistic annotation-a descriptive and comparative study. *Computational Linguistics in the Netherlands Journal*, *3*, 63–81.

Zhou, G., & Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. In *proceedings of the 40th annual meeting on association for computational linguistics* (pp. 473–480).

A Suggested system

A.1 System overview

The system in mind requires three variables as input:

1. The dump file obtained after running Frog with all NE's per item
2. The original data file with the documents (because the source isn't included in the dump file)
3. The search query

A user of the system has two options: either acquire the NE's in a selected document (without necessary prior knowledge of its contents), or find a document that is most relevant to the search query of the user. The query has to be of the following format:

$$TYPE{:}NE(+TYPE{:}NE)^n$$

An NE can be negated by putting a '!' right before the term. The result of running the code below using the default parameters can be seen in figure 10.

A.1.1 Alternating the search query

If the company *Coca-Cola* is not classified as ORG, but as a MISC instead, the following search query will not retrieve results:

$$ORG{:}Coca\text{-}Cola$$

However, adding another possible type to the query fixes the issue:

$$ORG{:}Coca\text{-}Cola+MISC{:}Coca\text{-}Cola$$

In contrast, when *Coca-Cola* is not *recalled*, there is no alternate query that will retrieve the NE.

A.1.2 Unknown entities

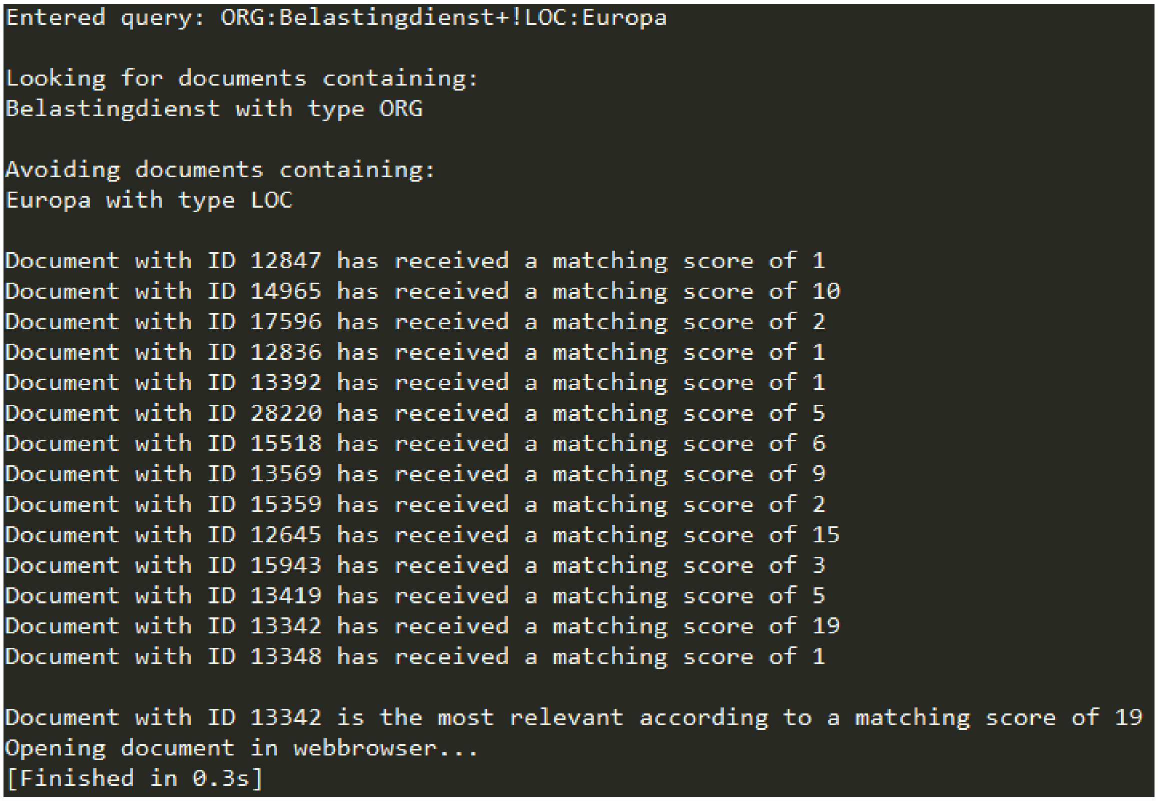
On the other hand, when a specific document is not sough after, the data acquired for the system will provide a summary based on the entities that were present per document. This will allow the user to examine a specific document, which may also contain unknown entities, instead of finding the most relevant based on a search query out of several.

A.2 Raw code

```
import json, re, webbrowser, sys, argparse
from operator import itemgetter

def main(dump_path, docs_path, query):
    with open(dump_path) as f:
        dump = json.load(f)

    winning_id = get_document(query, dump)
```



```
Entered query: ORG:Belastingdienst+!LOC:Europa

Looking for documents containing:
Belastingdienst with type ORG

Avoiding documents containing:
Europa with type LOC

Document with ID 12847 has received a matching score of 1
Document with ID 14965 has received a matching score of 10
Document with ID 17596 has received a matching score of 2
Document with ID 12836 has received a matching score of 1
Document with ID 13392 has received a matching score of 1
Document with ID 28220 has received a matching score of 5
Document with ID 15518 has received a matching score of 6
Document with ID 13569 has received a matching score of 9
Document with ID 15359 has received a matching score of 2
Document with ID 12645 has received a matching score of 15
Document with ID 15943 has received a matching score of 3
Document with ID 13419 has received a matching score of 5
Document with ID 13342 has received a matching score of 19
Document with ID 13348 has received a matching score of 1

Document with ID 13342 is the most relevant according to a matching score of 19
Opening document in webbrowser...
[Finished in 0.3s]
```

Figure 10: The command line output running the code of the suggested system

```
    with open(docs_path) as g:
        for doc in g:
            doc = json.loads(doc)
            if winning_id == doc['_id']:
                print 'Opening document in webbrowser...'
                webbrowser.open(doc['_source']['url'])

def get_document(query, dump):
    """
    Opens the document most relevant
    to the query in your webbrowser from its
    original source.
    """
    relevant_docs = []
    positive = []
    negative = []
    elements = query.split('+')

    print 'Entered query:', query

    ## Seperate the query into a list of wanted and unwanted entities in a doc
    for e in elements:
        entity_type, entity = e.split(':')
        if entity_type[0] == '!':
            negative.append((re.sub('!', '', entity_type), entity))
        else:
            positive.append((entity_type, entity))

    print '\n', 'Looking for documents containing:'
    for entry in positive:
        print entry[1], 'with type', entry[0]

    print '\n', 'Avoiding documents containing:'
    for entry in negative:
        print entry[1], 'with type', entry[0]

    for doc_id in dump:
        if matches_query(dump[doc_id], positive, negative):
            relevant_docs.append(doc_id)

    return most_relevant_doc_id(relevant_docs, query, dump)

def matches_query(doc, positive, negative):
    """
    Returns if the document matches the query
    by checking wanted and unwanted entities
    """
    match = False

    ## Look for a positive match
    for entity_type, entity in positive:
        if entity in doc['entities']:
            for tag in doc['entities'][entity]:
                t = tag.split('-')[1]
                if t == entity_type:
                    match = True

    ## If positive, look for a negative match
    if match:
        for entity_type, entity in negative:
            if entity in doc['entities']:
                for tag in doc['entities'][entity]:
                    t = tag.split('-')[1]
                    if t == entity_type:
                        match = False

    return match


def most_relevant_doc_id(relevant_docs, query, dump):
    """
    Returns the document ID with the highest
    matching score.
    """
    matching_scores = []
    for doc in relevant_docs:
        score = calculate_matching_score(dump[doc], query)
        matching_scores.append((doc, score))

    print ''
    for pair in matching_scores:
        print 'Document with ID %d has received a matching score of %d'%(int(pair[0]),


    ID, best_score = max(matching_scores, key=itemgetter(1))
    print '\n', 'Document with ID %d is the most relevant according to a matching score

    return ID


def calculate_matching_score(doc, query):
    ”””
    Returns the document's matching score
    to the query
    ”””
    elements = query.split('+')
    positive = []
    score = 0

    ## Get all wanted entities and their type out of the query
    for e in elements:
        entity_type, entity = e.split(':')
        if not entity_type[0] == '!':
            positive.append((entity_type, entity))

    ## Look for a positive match, and increase matching score accordingly
    for entity_type, entity in positive:
        if entity in doc['entities']:
            for tag in doc['entities'][entity]:
                t = tag.split('-')[1]
                if t == entity_type:
                    score += doc['entities'][entity][tag]

    return score

if __name__ == '__main__':

    p = argparse.ArgumentParser()
    p.add_argument('-dump', type=str, help='path to the dump with extracted entities pe
    p.add_argument('-docs', type=str, help='path to original unprocessed documents', de
    p.add_argument('-query', type=str, help='Query used to find the desired document', 
    args = p.parse_args(sys.argv[1:])

    main(args.dump, args.docs, args.query)
```