EmpiriST: AIPHES
Robust Tokenization and POS-Tagging for Different Genres

Steffen Remus[†§] and **Gerold Hintz**[†§] and **Darina Benikova**[‡§] and **Thomas Arnold**[†§] and **Judith Eckle-Kohler**[†§] and **Christian M. Meyer**[†§] and **Margot Mieskes**[*§] and **Chris Biemann**[†§]

[†]**Computer Science Dept.**
Technische Universität Darmstadt

[‡]**Computer Science and Applied Cognitive Science Dept.**
Universität Duisburg-Essen

[*]**Information Science**
University of Applied Sciences, Darmstadt

[§]**Research Training Group AIPHES**
Heidelberg University and
Technische Universität Darmstadt

[§]www.aiphes.tu-darmstadt.de

Abstract

We present our system used for the AIPHES team submission in the context of the EmpiriST shared task on "Automatic Linguistic Annotation of Computer-Mediated Communication / Social Media". Our system is based on a rule-based tokenizer and a machine learning sequence labelling POS tagger using a variety of features. We show that the system is robust across the two tested genres: German computer mediated communication (CMC) and general German web data (WEB). We achieve the second rank in three of four scenarios. Also, the presented systems are freely available as open source components.

1 Introduction

Tokenization and part-of-speech (POS) tagging are considered core tasks in a standard Natural Language Processing (NLP) pipeline. NLP tasks, such as summarization, information extraction, event detection, machine translation, and many others, are typically based on machine learning algorithms which use the outcome of lower level NLP tasks, such as tokens or intermediate linguistic phenomena including parts-of-speech or grammatical relations, as features. Though tokenization and part-of-speech tagging are considered simple tasks, it is highly important to achieve high-quality results, as errors propagate to downstream applications, where they are hard to repair and may cause notable consequential errors. Thus, a major goal is the minimization of the propagation of errors by using methods that perform as accurate as possible in lower level tasks on a diversity of texts and genres.

In this paper we present a simple, yet flexible and universally applicable system for tokenization and POS tagging German text. Our system participated in the EmpiriST shared task on "Automatic Linguistic Annotation of Computer-Mediated Communication / Social Media" (Beißwenger et al., 2016). For this task, we applied our solution to texts from two different genres: *a*) general, html-stripped *web data* and *b*) colloquial language from *social media texts*.

The paper is organized as follows: We first describe the shared task and related work Section 2. Our systems for tokenization and POS tagging are laid out in Section 3 and evaluated in Section 4, which includes a detailed error analysis. Section 5 concludes.

2 Task Description & Related Work

The main goal of the GSCL Shared Task "Automatic Linguistic Annotation of Computer-Mediated Communication / Social Media" was to encourage adaptation and development of language processing tools for German texts of computer-mediated communication genres. The shared task was divided into two subtasks, tokenization and POS tagging, which made use of an extended STTS-EmpiriST tag set. For both tasks, two data sets were provided for trial and training purposes.

- A computer-mediated communication data

Proceedings of the 10th Web as Corpus Workshop (WAC-X) and the EmpiriST Shared Task, pages 106–114,
Berlin, Germany, August 7-12, 2016.

set (CMC) that included chat texts, tweets, blogs and Wikipedia talk pages.

- A Web data set (WEB) with various web text genres.

The training data set includes 5,109 (WEB) and 6,034 (CMC) manually annotated and expert-checked tokens. System submissions for the tasks were evaluated by the organizers on 7,800 (WEB) and 6,142 (CMC) tokens of blind test data.

2.1 Tokenization

Tokenization is usually the first step in a NLP system. Even systems that do not follow the classical NLP pipeline architecture still mostly operate on the basis of tokens, including unified architectures starting from scratch (Collobert et al., 2011). This is common, since tokens – either directly or indirectly – are usually considered to bear the information in a text eventually. However, the importance of tokenization is often neglected, as simple methods like whitespace segmentation *can* yield acceptable accuracies for many languages at first sight (Webster and Kit, 1992). But errors in an early phase of an NLP pipeline can have severe effects to higher level tasks and influence their performance by a large margin.

Existing tokenizers can be organized into three categories: *a*) rule-based methods, *b*) supervised methods, *c*) unsupervised methods. Manning et al. (2014)[1], for example, internally use JFlex[2], which is a meta language for rules based on regular expressions and procedures to execute when a rule matches. In contrast, Jurish and Würzner (2013) present a supervised system for joint tokenization and sentence splitting, which employs a Hidden Markov Model on character features for boundary detection. Kiss and Strunk (2006) introduce *Punkt*, providing an unsupervised model for sentence splitting and tokenization. Kiss and Strunk (2006) use the fact that most ambiguous token or sentence boundaries happen around punctuation characters, such as periods/full stops. Punkt finds collocations of characters before and after punctuations, assuming that these collocations are typical abbreviations, initials, or ordinal numbers which can be maintained as a simple list of non-splittable tokens.

[1] As of the current version v3.6 of the Stanford Core Utils, the default PTBTokenizer uses JFlex.

[2] `http://jflex.de/`

Automatically learned models, both supervised and unsupervised, are typically hard to debug and the results might need post cleaning, e.g. post-merging or splitting of common mistakes, because modifying learned models is usually not trivial but need to be re-learned with different parameter settings or training data. However, it is important to offer the possibility to easily debug and change the outcome of the tokenization, hence, our goal is to implement a small and reasonable ruleset.

2.2 POS Tagging

Existing POS taggers for German primarily rely on the Stuttgart-Tübingen Tagset (STTS, Schiller et al. (1999)), which consists of 54 POS tags and distinguishes between eleven main parts of speech, which are further divided into various subcategories. The STTS tagset has become a de facto standard for German, as it is also used in major German treebanks, such as the Tiger treebank (Brants et al., 2004), called Tiger henceforth. Tiger consists of approx. 900,000 tokens of German newspaper text (taken from the Frankfurter Rundschau), and the POS annotations have been added semi-automatically. For this, the TnT tagger (Brants, 2000) was used, because it also outputs probabilities that can be used as confidence scores. Only POS tags with a low confidence score were checked for correctness by human annotators.

As the basis for the development of the STTS-tagset were newspaper corpora, STTS only contains six POS tags that describe categories other than the standard grammatical word categories (e.g., non-words or punctuation marks). In contrast, the extended version of STTS used in the EmpiriST shared task contains 18 additional tags for elements that are specific for computer-mediated communication, for example, tags for emoticons, hashtags and URLs, or tags for phenomena which are typical for spoken language.

State-of-the-art POS taggers use supervised machine learning to train a model from corpora annotated with POS tags. While there are several ways to model POS tagging as a machine learning problem, casting it as a sequence labeling problem is a frequent approach, used already for the early TnT tagger by Brants (2000). In sequence tagging, the learning algorithm – e.g. Hidden Markov Models or Conditional Random Fields (CRFs) – optimizes the most likely tags over the sequence, while taking interdependencies of tags into account – as op-

posed to a mere token-based classification.

Another annotation task that is a typical example of sequence labeling, is named entity recognition. For example, the GermaNER toolkit (Benikova et al., 2015) uses CRFs for learning to tag named entities. GermaNER has been built in a modular fashion and is highly configurable, which allows users to easily train it with new data and features sets, and hence we chose to build upon the GermaNER system for POS tagging in this shared task.

3 System Description

The systems we describe in the following subsections are available as open source components under the Apache v2 license.[3] For tokenization, we have not attempted to create different variants for the two text genres of the shared task, but rather provide a robust generic solution, since we would not want to adopt subsequent processing steps when applying them to a different genre.

3.1 Tokenization

We present a rule-based tokenizer where the rules describe merging routines of two or more conservatively segmented tokens. Rules are defined in terms of a list of common non-splittable terms and simple regular expressions. The tokenizer is configured with a set of configuration files, which we call a *ruleset*. A ruleset can be easily adapted or changed depending on a particular language. In the following we present the tokenizer's configuration options and show selected toy examples.

The main building blocks of the tokenizer are the following:

Conservative splits: A *base tokenizer* provides the initial tokens that are refined in the next steps. We chose a robust tokenizer that operates on general unicode character categories, i.e. a stream of characters is processed and for each character its general unicode category is retrieved. Based on the transition from the current character's unicode category to the next character's unicode category new token segments are created by some specified rules. More specifically, new token segments are created for empty space[4] to non-empty space transitions, letter[5] to non-letter and number[6] to non-number transitions or vice versa.

Merge list: We maintain a list of common abbreviations, which contains words or expressions with non-letter characters such as dots or hyphens. Additionally, this list contains a collection of common text-based emojis. Some selected examples are listed in Listing 1. The file was manually compiled from various sources in the web, including Wikipedia.

Merge rules: Since merge lists contain only fixed tokens that must match entirely and hence do not allow for modifications within tokens, we additionally maintain a list of merge rules which are specified as regular expressions. This is particularly important for expressions involving digits, such as date expressions, usernames, etc. Rules are processed in the order of their definition. Unfortunately, as with potentially every rule-based system, too many handwritten rules start to interfere and introduce unwanted behavior. This is especially true if rules are too general, i.e. they match more examples than they should. We balance this trade-off between rule complexity and rule interaction by introducing *global* and *local reject rules*, i.e. merge rules are rejected iff a reject rule also matches. The scope of these reject rules can be defined globally, matching tokens that should never be considered for merging, or locally, matching tokens that should not be considered for merging only if a particular merge rule matched. Multiple consecutive reject rules are possible. Listing 2 shows a snippet of the respective configuration file.

The tokenizer is implemented in Java using the Java default regular expression engine. It was developed as part of the *lt-segmenter*[7] and is provided as a branch[8].

3.2 POS Tagging

For POS tagging, we have adapted the GermaNER system, an open-source named entity recognition

[3] `https://github.com/AIPHES`
[4] general unicode categories `Zl`, `Zs`, `Zp`
[5] general unicode categories `Lu`, `Ll`, `Lt`, `Lm`, `Lo`
[6] general unicode categories `Nd`, `Nl`, `No`
[7] `https://tudarmstadt-lt.github.io/seg/`
[8] `https://github.com/AIPHES/tokenizer`

```
## lookahead-list.txt
C-Jugend
Ü-Ei
altgriech.
24/7
2B~not2B
a-z
a.k.a.
>_<
:-}
X8-{}
...
```

Listing 1: Examples of fixed entries, i.e. non-splitable tokens in the tokenizer's look-ahead list. Comments begin with a # character.

```
## lookahead-rules.txt
# reject ) followed by ; globally
- \);

# email a@b.com
+ [\.+\w\-]+@([\w\-]+\.)+[\w]{2,6}

# reverse emoticons (-:
+ (\[\]\)\(DP*)\1{0,}-?'?[:;8B=]
# reject ):
- \) ?:

...
```

Listing 2: Examples for merge rules defined as regular expressions. Merge rules are defined with an initial '+' in the beginning of the line, whereas reject rules are defined with an initial '−'. Global reject rules are defined before any positive rule and comments begin with a # character. A description of the rules can be found as comment before the actual rule.

tool written in Java. *GermaPOS*[9] is a fork of the software, adapting the framework for this purpose. As a machine learning algorithm, a CRF sequence tagger (Lafferty et al., 2001) is used. Specifically the implementation provided by CRFsuite (Okazaki, 2007), as is in the clearTK framework is employed.

The architecture of *GermaPOS* is a highly extensible UIMA[10] pipeline (Ferrucci and Lally, 2004), providing a simple interface to both training a new tagger based on user-provided training data, as well as running a pretrained model on simple text files. The pipeline first reads a tab-separated input file. In a subsequent step, feature extraction is performed per token, using additional information from external sources, e.g. word lists. Feature extraction can further take into account any surrounding context of the current token, e.g. time-shifted features of relative position -2, -1, 0, +1, +2. In training mode, a CRF model is then built on the basis of feature annotations; at runtime the model provides POS tags as UIMA annotations. An optional output step in the pipeline produces a POS-annotated file. Alternatively, the pipeline can be used within UIMA projects out of the box. We perform a post-hoc assignment of POS tags based on a subset of our mapping rules that cover EmpiriST-specific conventions. For example a token *emojiQsmilingFace* will be assigned the tag *EMOIMG*, regardless of the output of the sequence tagger.

Features We adapt nearly the full feature set of *GermaNER*, with the exception of POS features. In the following list, we give a brief overview – a more detailed description can be found in (Benikova et al., 2015).

1. **Character n-grams** First and last character n-grams for $n \in \{1, 2, 3\}$ of the current token, as well as time-shifted versions of this feature with offset from -2 to 2 are extracted.

2. **Gazetteers and word lists** We adapt most gazetteers from GermaNER, containing mostly named entities (NE). As we gained no performance increase from a higher coverage of NEs in our datasets through Freebase (Bollacker et al., 2008), we omit this resource in favor of a more lightweight system. In addition, we incorporate word lists. We employ a small list of English words[11], as well as hand-crafted lists[12] of onomatopoeia, discourse markers, Internet abbreviations, intensity markers, as well as various types of particles.

3. **Similar words** JoBimText (Biemann and Riedl, 2013) to obtain a distributional thesaurus (DT) from which the four most similar words for the current token are used. The underlying motivation is to be able to correctly

[9] *GermaPOS* is available at `https://github.com/AIPHES/GermaPOS`

[10] Unified Information Management Architecture, `https://uima.apache.org/`

[11] We use a list of English words as these cover most occurrences of foreign language tags

[12] Partially compiled from Wikipedia and enriched by data from various internet sites e.g. `internetslang.com`.

tag infrequent or unseen targets, by expanding them with a frequent similar term, most likely sharing the same part of speech.

4. **Topic clusters** LDA topic modeling was applied on the DT defined above, resulting in a fixed number of topic clusters. For each token, and time-shifted context tokens, its topic index is extracted as a feature. We again build on existing work of GermaNER and use a precomputed set of 200 clusters.

5. **Syntax** We use simple syntactic features, such as the word position and casing of tokens. We generalize the original GermaPOS setup to use arbitrary regular expressions as binary features. We then use all regular expressions designed for tokenization as features. This way, we also cover most casing information.

Furthermore, we extract the character range of each token as a feature, in case all characters fall into the same class. Hence, if all characters are from the same Unicode code block, this block is extracted as a feature. This feature allows, for example, to capture Unicode emoticons, not specifically preprocessed as in the EmpiriST data.

Training In the context of the EmpiriST shared task, we train a separate model for both the CMC and WEB datasets. As the training data is comparatively small for the purpose of POS tagging, we add the Tiger dataset to the respective training sets. The Tiger corpus is annotated using the standard STTS tagset, whereas the task at hand provided an extended tagset. In order to make learning from Tiger feasible, we have manually converted the Tiger data to the extended tagset using a set of simple rules, which aim at covering most of the easy cases.

As with GermaNER, the selection of resources and software components was done in favor of choosing a permissive license rather than focusing on system performance. Although it is plausible to improve POS tagging performance by integrating high-quality resources, we have opted to release GermaPOS with only free components, i.e. those already employed in GermaNER as well as manual additions not encumbered with restrictive usage rights. Where applicable, the system can be customized to utilize additional resources. A possible extension is the integration of another third-party POS tagger to be utilized as a feature.

Usage *GermaPOS* is provided as a runnable jar file with a pre-bundled model trained on the data described above. The training format is – equivalent to the EmpiriST training data – a tab-separated file of one token-tag pair per line and sentences being separated by an empty line.

4 Evaluation

Following the EmpiriST task setup, we evaluate our tokenizer by measuring precision P, recall R, and the F_1 score as in Jurish and Würzner (2013). Precision denotes the proportion of correctly identified token boundaries over the total number of token boundaries proposed by our tokenizer and recall denotes the proportion of correctly identified token boundaries over the total number of token boundaries in the gold standard. The F_1 score is the harmonic mean of precision and recall.

For our POS tagger, we report the tagging accuracy. That is, we measure the fraction of correct tag guesses over the total number of tokens to tag. To enable a comparison of our tagger's results with previous work on German, we additionally use the STTS mapping provided by the shared task organizers and measure the tagging accuracy using the mapped tags.

Below, we first discuss our results according to these standardized metrics and then conduct a careful analysis of the most prominent errors of our tools.

4.1 Results

We present results according to the tasks evaluation. Table 1 shows the results for the tokenization task for the two datasets CMC and WEB. Without adapting the rules for the particular sub-tasks, we achieved good performance on both sets such that we positioned on rank two in both categories.

The results for the POS tagging task are shown in Table 2. We achieve clearly better results on the WEB dataset (second best results) than on the CMS dataset. One possible reason for that is the distribution of the new POS tag labels in the test set. As can be seen in Table 3, the CMC data make more use of the new labels. Another reason might be the adaption of our system to the text style, which is dominated by the much larger Tiger training set.

Genre	*Rec*	*Prec*	F_1	*Rank*
CMC	99.30	98.62	98.96	2
WEB	99.63	99.89	99.76	2

Table 1: Tokenization results. We achieved rank two of six submissions in both categories. Two submissions were non-competitive but do not change our rank.

Genre	*Acc*	*Rank*
CMC	84.22	5
CMC (STTS Map)	87.10	2
Web	93.27	2
Web (STTS Map)	94.30	2

Table 2: POS tagging results. Among 17 submissions from eight teams, of which two were out of competition, we ranked second on the web data and fifth on the CMC data.

Tag	CMC	Web
ONO	2	
DM	6	
PTKIFG	72	61
PTKMA	74	11
PTKMWL	10	14
VVPPER	6	
VAPPER	4	
KOUSPPER	1	1
PPERPPER	1	1
ADVART	3	
EMOASC	71	
EMOIMG	63	
AKW	60	
HST	42	
ADR	48	
URL	16	
EML		1

Table 3: Distribution of new POS tag labels in the test sets.

4.2 Common Errors

We identified three main sources of **tokenization errors**. Examples in the following show gold tokenization on the left and system tokens on the right, errors are marked with an asterisk.

1. **Rules are underspecified**, which means that certain rules were not specified or the look-ahead list did not contain the particular abbreviation. Also, note that we deviated from the annotation guidelines and did not perform token splitting at camel case boundaries.

   Examples:

   ```
   * Eingetr.          | * Eingetr
     Lebenspartnersch. | * .
                       |
   * die               | * dieFeststellung
   * Feststellung      |   ,
                       |
     der               |   war
   * 1.                | * der1
     Teil              | * .
     meiner            |   Teil
   ```

2. **Rules are overspecified**, which means that rules are specified in our ruleset although they were not specified in the annotation guidelines.

   Example:

   ```
     Backlinks  |   Backlinks
     :          |   :
   * [[         | * [[sec:verschl]]
   * sec        |   Navigation
     :          |   Passwort-
     verschl    |   generator
   ```

3. **Current scheme cannot capture certain phenomena**, which happens on phenomena that are syntactically hard to distinguish. For instance, section listings that get identified as a date, e.g.

   ```
   * 1.3.  | * 1.
     Kekse | * 3.
   ```

POS tagging error analysis We have performed a post-hoc error analysis on the EmpiriST data. Table 4 shows a confusion matrix regarding classes of POS tags by their prefix (first character). Note that this matrix only lists tagging *errors*, so that the diagonal of the matrix denotes incorrect tagging within the same prefix class. It can be seen that the majority of errors happen within these classes, such as *N**. The most common tagging error is in fact mistagging *NE* and *NN*, which

		$*	AD*	AKW*	AP*	AR*	EML*	EMO*	N*	P*	PPER*	PT*	V*
$*	68	0	0	1	1	0	0	0	1	0	1	0	1
AD*	2	49	1	9	0	1	0	81	21	3	159	21	34
AKW*	0	1	0	0	0	0	0	1	0	0	0	0	3
AP*	2	0	0	4	0	0	0	0	1	0	6	2	10
AR*	0	1	0	0	0	0	0	0	22	0	2	0	0
EML*	0	0	0	0	0	0	0	0	0	0	0	0	0
EMO*	1	0	0	0	0	0	0	0	0	0	0	0	0
N*	1	25	2	3	3	0	2	163	3	2	6	16	43
P*	0	3	0	1	10	0	0	4	37	4	1	1	10
PPER*	0	0	0	0	0	0	0	0	7	0	0	0	0
PT*	0	34	0	5	0	0	0	1	7	0	16	0	0
V*	0	16	3	4	1	0	0	30	1	0	0	137	7
other	11	10	5	9	1	0	1	28	9	1	5	1	25

Table 4: Confusion matrix for POS tag prefixes (errors only)

Error class	count	(%)
1. missed extended tagset	28	17.6
2. incorrectly assigned new tag	4	2.5
3. confusion of function word tags	22	13.6
4. mistagged *NN* due to lower case	23	14.4
5. mistagged *NE* as *NN* & vice-versa	20	12.5
6. mistagged *NE* as other	12	10.0
7. unknown emoticon	1	0.6
8. unknown foreign language word	2	1.3
9. error due to abbreviation	2	1.3
10. incorrect punctuation tag	20	12.5
11. other	26	16.3

Table 5: POS tagging error classes

is sometimes also difficult to discriminate for human annotators.

We define a number of error classes to better quantify the types of errors introduced by our tagger. For this, we construct an ordered list from which we select the first item that applies as the error class:

1. **missed extended tagset**

   A tag from the extended set was required, but a standard STTS tag was assigned. Example:

   ```
   wohl  PTKMA      | wohl  ADV
   ```

2. **incorrectly assigned new tag**

   A tag from the extended set was assigned incorrectly. Example:

   ```
   mal ADV          | mal PTKMA
   gucken  VVINF    | gucken  VVINF
   ```

3. **confusion of function word tags**

   Incorrect tag within the class of function words. Example:

   ```
   den ART          | den ART
   Irrsinn NN       | Irrsinn NN
   nicht PTKNEG     | nicht PTKNEG
   endlich ADJD     | endlich ADV
   beenden VVINF    | beenden VVINF
   ```

4. **mistagged *NN* due to lower case**

   A lower-case noun was not captured. Example:

   ```
   ihre  PPOSAT     | ihre  PPOSAT
   entscheidung     | entscheidung VVFIN
   NN               |
   ```

5. **mistagged *NE* as *NN* and vice versa**

   Incorrect tagging of named entities and nouns. Example:

   ```
   HErr  NN         | HErr  NE
   Ozdemir NE       | Ozdemir NE
   ```

6. **mistagged *NE* as other**

   A named entity was not recognized and tagged with a tag other than *NN*. Example:

   ```
   Frage NN         | Frage NN
   von APPR         | von APPR
   @DieMaJa22  NE   | @DieMaJa22  ADR
   ```

7. **unknown emoticon**

   An emoticon was not identified as such (due to not being covered by regular expressions). Example:

   ```
   *<:-) EMOASC     | *<:-) NE
   ```

8. **unknown foreign language word**

   A foreign word was not tagged as *FM*. Example:

meinst VVFIN	meinst VVFIN
du PPER	du PPER
bazdmeg FM	bazdmeg VVFIN

9. **error due to abbreviation**

   Word abbreviations leading to incorrect tagging. Example:

Anerkennung NN	Anerkennung NN
der ART	der ART
Eingetr. ADJA	Eingetr. NN
Partnerschaft NN	Partnerschaft NN

10. **incorrect punctuation tag**

    Errors within the class of punctuation tags. Example:

Thema NN	Thema NN
: $(	: $.
Drogenpolitik NN	Drogenpolitik NN
... $.	... $(

11. **other**

    if none of the other criteria apply

We then annotate the first 160 errors from the CMC test set with their respective error classes. The results are shown in Table 5. It can be observed that most errors are related to nouns or named entities. The tagger commonly confuses these two. For CMC data, a very common error which throws off the tagger are nouns written in lower case, which generally get assigned a completely different POS. As we have trained our tagger on a standard STTS-annotated corpus (with minimal postprocessing), some errors also stem from not capturing the new rules introduced by the extended EmpiriST tagset. There are also a few errors resulting from unknown foreign language words or emoticons not captured by our regular expressions, but regarding their quantity this is much less of a problem and they only account for a tiny percentage of errors.

5 Conclusion

We have presented our submission to the EmpiriST shared task on "Automatic Linguistic Annotation of Computer Mediated Communication / Social Media", comprising a rule-based tokenizer and a machine-learning-based POS tagger. Overall, we achieved a very good, but not the best performance amongst the participating systems, ranking second throughout except for CMC POS tagging with the extended tagset. Our submission was aimed at robustness; we have not tuned our tokenizer per genre, and show good POS tagging performance throughout. Both systems are freely available as open source under a permissive license.

Acknowledgments

This work has been supported by the German Research Foundation as part of the Research Training Group "Adaptive Preparation of Information from Heterogeneous Sources" (AIPHES) under grant No. GRK 1994/1 and by the German Institute for Educational Research (DIPF) under the KDSL program.

References

Michael Beißwenger, Sabine Bartsch, Stefan Evert, and Kay-Michael Würzner. 2016. Empirist 2015: A shared task on the automatic linguistic annotation of computer-mediated communication, social media and web corpora. In *Proceedings of the 10th Web as Corpus Workshop (WAC-X)*, Berlin, Germany.

Darina Benikova, Seid Muhie Yimam, and Chris Biemann. 2015. GermaNER: Free open German named entity recognition tool. In *International Conference of the German Society for Computational Linguistics and Language Technology (GSCL-2015)*, Essen, Germany.

Chris Biemann and Martin Riedl. 2013. Text: Now in 2D! a framework for lexical expansion with contextual similarity. *Journal of Language Modelling*, 1(1):55–95.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA. ACM.

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: linguistic interpretation of a German corpus. *Research on Language and Computation*, 2(4):597–620.

Thorsten Brants. 2000. TnT: A Statistical Part-of-speech Tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231, Seattle, Washington.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

David Ferrucci and Adam Lally. 2004. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Bryan Jurish and Kay-Michael Würzner. 2013. Word and sentence tokenization with Hidden Markov Models. *Journal for Language Technology and Computational Linguistics (JLCL)*, 28(2):61–83.

T. Kiss and J. Strunk. 2006. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, MD.

Naoaki Okazaki. 2007. CRFsuite: a fast implementation of conditional random fields (CRFs). *http://www.chokkan.org/software/crfsuite/*.

Anne Schiller, Simone Teufel, Christine Stöckert, and Christine Thielen. 1999. Guidelines für das Tagging deutscher Textcorpora mit STTS (Kleines und großes Tagset). Technical Report, Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung.

Jonathan J. Webster and Chunyu Kit. 1992. Tokenization as the initial phase in NLP. In *The 15th International Conference on Computational Linguistics (COLING)*, pages 1106–1110, Nantes, France.