Disclose Models, Hide the Data— How to Make Use of Confidential Corpora without Seeing Sensitive Raw Data

Erik Faessler, Johannes Hellrich, Udo Hahn

Jena University Language & Information Engineering (JULIE) Lab
Friedrich-Schiller-Universität Jena
Jena, Germany
`http://www.julielab.de`

Abstract

Confidential corpora from the medical, enterprise, security or intelligence domains often contain sensitive raw data which lead to severe restrictions as far as the public accessibility and distribution of such language resources are concerned. The enforcement of strict mechanisms of data protection consitutes a serious barrier for progress in language technology (products) in such domains, since these data are extremely rare or even unavailable for scientists and developers not directly involved in the creation and maintenance of such resources. In order to by-pass this problem, we here propose to distribute trained language models which were derived from such resources as a substitute for the original confidential raw data which remain hidden to the outside world. As an example, we exploit the access-protected German-language medical FRAMED corpus from which we generate and distribute models for sentence splitting, tokenization and POS tagging based on software taken from OPENNLP, NLTK and JCORE, our own UIMA-based text analytics pipeline.

Keywords: confidential corpora, language models, re-usability

1. Introduction

With the growing potential of language technologies to identify the underlying meaning of spoken or written natural language utterances, we currently witness a growing demand for computational content analytics in many fields. Whereas many application areas can make use of easily and publically accessible language raw data (such as newspaper articles or newswire feeds), others are characterized by varying levels of confidentiality which translate directly into corresponding protection layers to access such data. These concerns are most evident for clinical documents (such as those contained in electronic patient records or genomic databases, cf. e.g. Claerhout and De Moor (2005), Damschroder et al. (2007)), documents from enterprises (e.g. damage claims and case decision letters from insurance companies, pharmaceutical test reports, cf. e.g. Smallwood (2012)) or documents related to intelligence, security and police activities (blackmail letters, suicide notes, etc.; cf. e.g. Vetulani (2012)). We call these collections *confidential* because such sensitive data are usually collected under binding protection rules and non-disclosure commitments which preclude extramural use and further exploitation.

Indeed we ourselves faced such a confidentiality problem with our FRAMED corpus (Wermter and Hahn, 2004). FRAMED consists of a well-balanced mixture of medical document types such as discharge summaries, pathology reports but also medical textbook excerpts, all written in German language. Altogether, FRAMED is made of 7,000 sentences, with approximately 100,000 tokens. The original FRAMED corpus comes with manually supplied sentence boundary, token segmentation and part-of-speech tags. Over the years, we received several requests for making this corpus available to the broader language resources community. Unfortunately, in each case we had to deny such requests and apologize for explicit distribution barriers which were part of mutual agreements with the various clinical departments involved in the creation of this resource. One of the few cases where we were able to help involved a member of our team carrying out some statistical analysis for a requesting group. The results of this work are described in Chapman et al. (2013). This service-oriented perspective, acting as a kind of data oracle, let us think about alternatives to raw data distribution.

In order to mitigate this unsatisfactory situation where a highly coveted corpus remains inaccessible to the language resources community, we here propose the following solution. Rather than distributing the original raw data as a resource, we distribute language models which were derived from it and capture relevant language characteristics. Accordingly, the analytic potential is shifted from the exploitation of a corpus to the re-use of language models without unmasking the raw data which underlie their generation.[1]

To assure wide re-usability by the NLP community, we have generated sentence and token segmentation models, as well as POS tagging models within the framework of three NLP model repositories, namely OPENNLP,[2] NLTK[3] and JCORE.[4] At our lab web site,[5] we offer download options for these models. As a consequence, potential consumers of these models may process German-language clinical text by re-using pre-computed models, without requiring access-protected clinical raw data.

[1]We are aware of the possibility to read out portions of the textual data stored in models by employing programs accessing the API of the training frameworks being used. However, such an unfriendly use of our service will, at best, render a probabilistic model of the original textual resource, not the specific original one we started from. Of course, all sensitive data which allow to identify an individual patient have been deleted prior to corpus assembly. See also Section 5. where we further discuss this issue.

[2]`http://opennlp.apache.org/`

[3]`http://nltk.org/`

[4]`http://www.julielab.de/Resources.html`

[5]`http://www.julielab.de`

2. NLP Frameworks

In this section, we illustrate the model-for-data paradigm by selecting some popular NLP frameworks, namely OPENNLP, NLTK and JCORE, as an embedding for concrete NLP language models which account for sentence splitting, tokenization and POS tagging.

In the JAVA world, APACHE OPENNLP is very popular and widely used for a range of NLP tasks. It offers basic tools like sentence splitters, tokenizers, name finders, chunkers, parsers and a coreference resolver. OPENNLP comes with a set of pre-trained models (exploiting newspaper/newswire texts as raw data) for download from its web page.

The Natural Language Toolkit (NLTK) is a PYTHON framework for general NLP, not specialized to any particular domain and frequently used for educational purposes (Bird et al., 2009). It offers a comprehensive library of natural language analysis tools such as tokenizers, stemmers, POS taggers, syntactic parsers, and semantic reasoning tools and excels with its comparatively low entry barrier (relative to JAVA tooling). Yet some of its basic components, tokenizers in particular, are only available as rule-based systems. We used NLTK's PUNKT library for sentence splitting and trained the POS tagger with a MEGAM model.[6]

The JULIE Lab UIMA Component Repository (JCoRe) (Hahn et al., 2008) was built as a repository of interoperable UIMA components[7] already adapted to the special needs of the analysis of English life sciences-related literature. It contains self-developed components to account for special phenomena of the biomedical domain (e.g. especially hard tokenization problems for protein, gene or chemical names), as well as UIMA wrapper components for established NLP frameworks like OPENNLP.

There are further tools and frameworks that are very popular in the NLP community, such as GATE[8] or LINGPIPE.[9] However, most components we are focusing on this paper—sentence splitters, tokenizers and POS taggers—in these frameworks are rule-based and adaptation to certain corpora, if needed, relies on human intuition and thus in-depth manual inspection of the raw data. Since our goal is automatic model learning employing machine learning (ML) methodologies on the basis of language data, we refrain from considering these tools any further in this paper.

3. Training on FRAMED

Even some of the frameworks we focus on offer rule-based components besides trainable ones. We re-trained all trainable models for sentence splitting, tokenization and POS tagging for all repositories using FRAMED as the reference corpus. In order to allow for an assessment of the model quality, we performed a 10-fold cross-validation for all trained components. All tools except NLTK's sentence splitter come with evaluation methods of their own. However, the NLTK POS tagger does not implement a cross-validation algorithm. Thus, for the employed NLTK tools we conducted the cross-validation with custom algorithms.

Some of these evaluation tools use F_1 metrics, whereas others measure accuracy. For the JCORE tools, we added F-score numbers to the evaluation report.

An overview of the results is given in Table 1. Data for NLTK's tokenizer is missing because it is rule-based and requires manual tuning; JCORE's POS tagger is basically a wrapper for OPENNLP's tagging tool, so data are the same as for the OPENNLP POS tagger. We provide all evaluation values available, i.e. cells containing only a dash indicate that an evaluation for the corresponding measure was not offered by the tool's evaluation method.

framework/component	F-Score	accuracy
OPENNLP		
+ sentence splitter	0.968	—
+ tokenizer	0.995	—
+ POS tagger	—	0.969
NLTK		
+ sentence splitter	—	0.923
+ POS tagger	—	0.938
JCORE		
+ sentence splitter	0.994	0.987
+ tokenizer	0.996	0.992
+ POS tagger	—	0.969

Table 1: 10-fold cross-validation results for selected framework and various components.

Overall, the data indicate a nice fit of the trained models with the FRAMED gold standard within an F-score range of 0.97 to 0.99. The lowest performance on sentence splitting can be attributed to the NLTK sentence splitter, tokenization performance between OPENNLP and JCORE seems on par with each other. The POS tagging performance varies markedly between OPENNLP and NLTK.

While these raw evaluation data may provide a first impression on tool performance, the real benefit of training multiple language models for the same task is that users may choose between models depending on their application and specific task requirements. This choice is hard, if not impossible, to justify in the absence of data to test on. As a substitute, we now turn to a detailed error analysis of the different taggers on otherwise hidden data.

4. Error analysis

The following error analysis is structured by model type, i.e. models for sentence splitting, tokenization and part-of-speech tagging. Once again, 10-fold cross-validations were used. In this way, the whole corpus has been employed as test data and contributes to the error analysis.

4.1. Sentence splitters

This section deals with statistics and the types of errors observed on FRAMED using the sentence models trained on the corpus. Sentence detection is the task of disambiguating potential sentence-delimiting characters such as dots, colons etc. All errors of all sentence splitter models have been manually reviewed and classified.

The OPENNLP sentence splitter uses a maximum entropy model for the classification of end-of-sentence-delimiters.

[6]http://www.umiacs.umd.edu/~hal/megam/
[7]http://uima.apache.org/
[8]https://gate.ac.uk/
[9]http://alias-i.com/lingpipe/

It reaches an F_1 score of 0.968 on FRAMED with mostly default parameters. However, the set of possible end-of-sentence symbols has to be explicitly specified for the tool in order to archieve this performance figure. Otherwise, the sentence boundary detector will, e.g. not split at colons causing the performance of the tool to drop to 0.789 F_1.
For the OPENNLP sentence splitter, we encountered a total of 165 errors; Table 2 contain a breakdown into error classes. Error categories we distinguish here mean that the error happened after an abbreviation, a semicolon, etc.

error class	number	%
Total	165	**100.00**
FP	97	58.79
FN	68	41.21
Abbreviation	82	**49.70**
FP	71	43.03
FN	11	6.67
Semicolon (FN only)	28	**16.97**
Fullstop	20	**12.12**
FP	5	3.03
FN	15	9.09
Enumeration	17	**10.30**
FP	10	6.06
FN	7	4.24
Date (FP only)	7	**4.24**
Other	11	**6.67**
FP	4	2.42
FN	7	4.24

Table 2: Error statistics for OPENNLP sentence splitter ('FP' stands for false positive, 'FN' for false negative).

A closer look at the context of the error sources 'semicolon' and 'fullstop' reveals that a large portion of those mistakes can be attributed to an idiosyncratic property of the FRAMED corpus, namely pseudo-sentences without even a predicate. In extreme cases, such sentences consist of a single term, e.g. 'Aufnahme-EKG:'. In such cases, the error which occurs is that *before* such keyword-sentences no split is carried out. Thus, the pseudo-sentence is handled as an extension of the previous sentence.
The NLTK sentence boundary detector (Kiss and Strunk, 2006) is an unsupervised tool that heavily relies on the detection of collocation expressions with a special focus on the detection of abbreviations. It reaches 0.923 F_1 on FRAMED. The error statistics can be found in Table 3.
Most errors (more than 88%) are due to erroneous splits after enumerations like '3.', 'D.' etc. Only relatively few errors have their origin in a wrong split after abbreviations. This proliferation of enumeration errors may be a direct consequence of the principle of the sentence splitter to focus on the identification of collocations. Enumerations do not, in general, follow such a pattern and are thus difficult to handle. As an example for the disambiguation of a dot following a number consider an expression such as '12. Juni' ('12th of June'). The pattern described is '[number]. [month]'. General enumerations do not follow such a collocation pattern but rather '[number]. [arbitrary word]' which is clearly non-discriminative.

error class	number	%
Total	578	**100**
FP	575	99.48
FN	3	0.52
Enumeration	513	**88.75**
Abbreviation	45	**7.79**
Date	8	**1.38**
Closed Parenthesis	7	**1.21**
Fullstop	5	**0.87**
FP	2	0.35
FN	3	0.52

Table 3: Error statistics for NLTK sentence splitter. All errors are false positives (FP), unless indicated otherwise.

JCORE's sentence splitter (Tomanek et al., 2007) is based on conditional random fields (CRF). It scores 0.994 F_1 on FRAMED in a 10-fold cross-validation and thus shows the highest performance of sentence splitters in this comparison. Error class statistics are shown in Table 4.

error class	number	%
Total	122	**100.00**
FP	64	66.39
FN	58	33.60
Abbreviation	83	**68.03**
FP	54	44.26
FN	29	23.77
Number followed by dot or colon	22	**18.03**
FP	10	8.20
FN	12	9.84
Colon in open parentheses (FN only)	17	**13.93**

Table 4: Error statistics for JCORE's sentence splitter.

The main source of errors are abbreviations with 68%. The other major error class is built up by numbers followed by a dot or a colon, however with no particular tendency of the kind of numerical expression. Errors are a few enumerations as well as date expressions. The third group of errors stems from splits not made after colons in expressions like '(links:' or '(Lebergewicht:'. While there may be good reasons why the corpus specifies splits at these locations, in practice sentence boundaries within non-closing parenthesis expressions should be very rare.
Overall, the three tools have their individual strengths and weaknesses concerning their performance on FRAMED. The OPENNLP sentence splitter has some problems with abbreviations, particularly the abbreviation 'n.' which occurs frequently within the corpus. JCORE's splitter does not share this particular problem but errs similarly often on abbreviations despite having the best performance of all three tools. Here, the NLTK sentence detector makes the fewest errors but, on the other hand, does not handle enumerations so well.

4.2. Tokenizers

The sentence splitter error analysis could be made on the basis of all errors, either because there were only few of them or the main error class was rather well-defined. For tokenization, however, only a sample of the error types can be reviewed because of their diversity. For the two tokenizers employed, namely from OPENNLP and from JCORE, samples of 50 errors each were chosen randomly.

Most of the errors for the OPENNLP tokenizer depicted in Table 5 can be traced to abbreviations, with a remarkable percentage rate of 74%. Another source of errors is again a special feature of FRAMED where expressions like '4x7mg' appear regularly as dosage expressions (i.e. 'four times 7mg'). An error in the Dosage class may mean that either before or after the multiplication character 'x' a tokenization error occurred. This error happens on a frequent basis, however not always.

error class	number	%
Total	50	**100.00**
FP	37	74.00
FN	13	26.00
Abbreviation	37	**74.00**
FP	34	68.00
FN	3	6.00
Dosage (FN only)	6	**12.00**
Other	7	**14.00**
FP	2	4.00
FN	5	10.00

Table 5: Error statistics for the OPENNLP tokenizer based on a random error sample of size 50.

JCORE's error statistics for tokenization is shown in Table 6 and much more diverse, as indicated by a percentage rate of 30% for the 'other' error class. Those errors do not follow a particular pattern and consist of missing splits at full stops, commas and semicolons as well as additional splits at plus or percent characters. Most of the errors occur at the 'x' multiplication character, all of which are missing splits. Similarly to the OPENNLP tokenizer, this error occurs quite frequently, yet not always. As an extreme case, consider an expression like '18x7x4' which is tokenized as '18x 7 x 4' missing a single split in a longer sequence of necessary split decisions ('18' 'x' '7' 'x' '4').

Finally, the error class "x' as prefix or suffix' is unique to JCORE's tokenizer in our experiments. It means that in words like 'Appendix' the trailing 'x' is mistakenly judged as a multiplication character and split off. This error happens regularly, but not always. Furthermore, floating numbers, written with a comma in German texts (such as '16,7'), are mistakenly split in some cases.

error class	number	%
Total	50	**100.00**
Multiplication (FN only)	27	**54.00**
'x' as prefix or suffix (FP only)	5	**10.00**
Split of floating numbers (FP only)	3	**6.00**
Other	15	**30.00**
FP	9	18.00
FN	6	12.00

Table 6: Error statistics for the JCORE tokenizer based on a random error sample of size 50.

4.3. Part-of-Speech Taggers

Due to the large number of tags in a POS tagset (the standard German STTS tagset incorporates 54 elements (Schiller et al., 1999)), error analysis is not as easily carried out as for the sentence and token segmentation tasks. To carefully assess the behaviour of the two POS taggers—the OPENNLP POS tagger and the NLTK POS tagger—different perspectives are taken on the errors of each component ranging from very specific information to more and more generalized and aggregated analysis data.

First, the 20 most frequent errors are given where we focus on the decision task which exact tag was expected and which tag was actually tagged instead. This view is very fine grained because we distinguish as single errors whether ADJD was tagged instead of VVPP and whether VVPP was tagged instead of ADJD. We call this view *asymmetric*.

In an alternative *symmetric* error scenario, we abstract away from these directional issues. In order to avoid an overly specific error discussion—the STTS-MED tag set used for FRAMED consists of 57 tag elements—we, furthermore, map this original tag set to the coarser-grained 12-element universal tag set (Petrov et al., 2012).[10] This way, errors involving the same general part of speech type, e.g. verbs or adjectives, can easier be distinguished from ones transgressing different POS class boundaries. Depending on the application such major category errors (classifying, e.g. a verb as an adjective) might be more critical than in-between category misclassifications (classifying, e.g. a finite verb as an infinitive).

The OPENNLP POS tagger made overall 3,064 errors in a 10-fold cross-validation on FRAMED. The asymmetric, fine-grained error statistics are displayed in Table 7. Most errors deal with the wrong classification of proper nouns (NE *vs.* NN). However, these errors only make up close to 6% of all errors. Misclassifications between verbs and adjectives are the second most important error class on the level of 4%.

An even more telling view on error types is provided by the symmetric error overview in Table 8. When direction is ignored the NN-NE error class falls to second rank behind errors involving VVPP and ADJD tags.

Data aggregation is further elevated when we map the specific STTS-MED tags to the universal tag set, as shown in Table 9 for the asymmetric case and in Table 10 for the symmetric case. Here it becomes even more apparent that lots of errors (16.94%) involve the incorrect classification of verbs into the wrong verbal form, followed by misclassification of adjectives into verbs and vice versa. Both are crucial for any form of relation extraction.

With 6,085 errors, the NLTK POS tagger almost doubles the error rate of the OPENNLP POS tagger on the FRAMED corpus. The error statistics are organized in the same way

[10]The tag elements of this universal set are: . – punctuation, ADJ – adjective, ADP – pre-/postposition, ADV – adverb, CONJ – conjunction, DET – determiner, NOUN – noun, NUM – number, PRON – pronoun, PRT – particles, VERB – verb, X – non-word.

expected	actual	number	%
Total		3,064	100
NE	NN	183	5.97
ADJD	VVPP	142	4.63
VVPP	ADJD	116	3.79
VVINF	VVFIN	112	3.66
ADJA	NN	106	3.46
VVFIN	VVINF	97	3.17
ADJD	ADJA	96	3.13
PRELS	ART	85	2.77
VVFIN	VVPP	83	2.71
NNL	NN	73	2.38
NN	NE	65	2.12
ADJD	NN	55	1.80
NN	ADJA	54	1.76
VVFIN	ADJA	54	1.76
TRUNC	NN	47	1.53
REF	ENUM	46	1.50
VVPP	VVFIN	45	1.47
VVINF	VVPP	44	1.44
ADV	ADJD	43	1.40
ADJD	ADV	41	1.34

Table 7: Asymmetric POS tag error statistics of OpenNLP POS tagger on the basis of the STTS-MED tagset.

tag1	tag2	number	%
Total		3,064	100
ADJD	VVPP	258	8.42
NE	NN	248	8.09
VVFIN	VVINF	209	6.82
ADJA	NN	160	5.22
VVFIN	VVPP	128	4.18
ADJA	ADJD	107	3.49
ART	PRELS	103	3.36
ENUM	REF	87	2.84
ADJD	ADV	84	2.74
NN	NNL	84	2.74
ADJA	VVFIN	74	2.42
ADJD	NN	74	2.42
ADJD	VVFIN	73	2.38
VVINF	VVPP	61	1.99
ADJA	ADV	50	1.63
NN	TRUNC	47	1.53
VAFIN	VAINF	45	1.47
ADJA	VVINF	40	1.31
PDAT	PDS	39	1.27
APPR	PTKZU	36	1.17

Table 8: Symmetric POS tag error statistics of OpenNLP POS tagger on the basis of the STTS-MED tagset.

expected	actual	number	%
Total		3,064	100
VERB	VERB	519	16.94
NOUN	NOUN	333	10.87
VERB	ADJ	272	8.88
ADJ	VERB	211	6.89
ADJ	NOUN	163	5.32
PRON	DET	119	3.88
ADJ	ADJ	107	3.49
PRON	PRON	101	3.30
X	NOUN	101	3.30
X	X	90	2.94
NOUN	ADJ	83	2.71
ADV	ADJ	80	2.61
PRON	ADJ	58	1.89
ADJ	ADV	55	1.80
CONJ	ADP	55	1.80
PRT	ADP	55	1.80
VERB	NOUN	43	1.40
X	ADJ	36	1.17
PRON	ADV	30	0.98
ADJ	X	29	0.95

Table 9: Asymmetric POS tag error statistics of OpenNLP POS tagger on the basis of the universal tagset.

tag1	tag2	number	%
Total		3,064	100
VERB	VERB	519	16.94
ADJ	VERB	483	15.76
NOUN	NOUN	333	10.87
ADJ	NOUN	246	8.03
DET	PRON	138	4.50
ADJ	ADV	135	4.41
ADJ	ADJ	107	3.49
PRON	PRON	101	3.30
NOUN	X	97	3.17
X	X	90	2.94
ADP	PRT	84	2.74
ADJ	PRON	67	2.19
ADJ	X	65	2.12
ADP	CONJ	63	2.06
NOUN	VERB	53	1.73
ADV	CONJ	43	1.40
ADJ	ADP	39	1.27
ADV	PRON	39	1.27
ADV	NOUN	32	1.04
ADV	ADP	29	0.95

Table 10: Symmetric POS tag error statistics of OpenNLP POS tagger on the basis of the universal tagset.

as above. Corresponding to the previous figures, the fine-grained asymmetric and symmetric errors are displayed in Tables 11 and 12, respectively. Error statistics derived from aggregations to the universal tag set are shown in Tables 13 and 14. Again, the classification into the wrong verbal form is a dominant error, however, misclassifications between adjectives and nouns and errors between adjectives and verbs are even more frequent than for OpenNLP.

With this wealth of error data what can we conclude for making informed guesses for concrete NLP applications? One way to pave the way in this data jungle is to have a closer look at single tags, erroneous taggings they are in-

expected	actual	number	%
Total		6,085	100
NN	ADJA	391	6.42
ADJA	NN	356	5.85
NE	NN	252	4.14
ADJD	VVPP	229	3.76
VVFIN	VVPP	219	3.60
PTKVZ	APPR	167	2.74
ADJD	ADJA	145	2.38
VVPP	VVFIN	131	2.15
VVFIN	ADJA	130	2.14
ADV	ADJA	117	1.92
ART	PRELS	114	1.87
VVFIN	VVINF	111	1.82
VVINF	VVFIN	111	1.82
ADV	ADJD	93	1.53
VVINF	ADJA	88	1.45
NN	NE	86	1.41
ADJD	NN	85	1.40
VVPP	ADJD	78	1.28
PIDAT	ADJA	75	1.23
PIAT	ADJA	73	1.20

Table 11: Asymmetric POS tag error statistics of NLTK POS tagger on the basis of the STTS-MED tagset.

tag1	tag2	number	%
Total		6,085	100
ADJA	NN	747	12.27
VVFIN	VVPP	350	5.75
NE	NN	338	5.55
ADJD	VVPP	307	5.04
VVFIN	VVINF	222	3.65
ADJA	VVFIN	195	3.20
APPR	PTKVZ	190	3.12
ART	PRELS	184	3.02
ADJA	ADJD	155	2.55
ADJA	ADV	140	2.30
ADJD	ADV	136	2.23
ADJD	NN	135	2.22
ADJA	VVINF	118	1.94
ADJD	VVFIN	114	1.87
APPR	PTKZU	103	1.69
NN	NNL	100	1.64
VVINF	VVPP	87	1.43
ADV	NN	86	1.41
VAFIN	VAINF	81	1.33
ADJA	VVPP	80	1.31

Table 12: Symmetric POS tag error statistics of NLTK POS tagger on the basis of the STTS-MED tagset.

expected	actual	number	%
Total		6,085	100
VERB	VERB	831	13.65
VERB	ADJ	496	8.15
NOUN	ADJ	486	7.99
ADJ	NOUN	452	7.43
NOUN	NOUN	452	7.43
ADJ	VERB	424	6.97
PRON	ADJ	305	5.01
PRT	ADP	237	3.89
ADV	ADJ	211	3.47
ADJ	ADJ	155	2.55
VERB	NOUN	152	2.50
X	NOUN	121	1.99
DET	PRON	114	1.87
PRON	PRON	102	1.68
ADP	PRT	97	1.59
PRON	DET	94	1.54
PRON	NOUN	79	1.30
ADP	ADJ	77	1.27
X	X	68	1.12
ADJ	ADV	66	1.08

Table 13: Asymmetric POS tag error statistics of NLTK POS tagger on the basis of the universal tagset.

tag1	tag2	number	%
Total		6,085	100
ADJ	NOUN	938	15.41
ADJ	VERB	920	15.12
VERB	VERB	831	13.65
NOUN	NOUN	452	7.43
ADP	PRT	334	5.49
ADJ	PRON	306	5.03
ADJ	ADV	277	4.55
NOUN	VERB	217	3.57
DET	PRON	208	3.42
ADJ	ADJ	155	2.55
PRON	PRON	102	1.68
NOUN	X	97	1.59
ADV	NOUN	93	1.53
ADJ	ADP	90	1.48
NOUN	PRON	80	1.31
ADP	CONJ	79	1.30
X	X	68	1.12
ADJ	X	58	0.95
ADV	VERB	56	0.92
ADV	CONJ	52	0.85

Table 14: Symmetric POS tag error statistics of NLTK POS tagger on the basis of the universal tagset.

volved in and the number of occurrences of these single tags. This way, the effects of mis-tagging are weighed against their distributional importance. This intuition is captured by the notion of the *percentaged tag error ratio* defined as

$$\textit{[tag error ratio]}_i := \frac{\#\textit{tagging errors for tag}_i}{\#\textit{tagging decisions for tag}_i} \times 100$$

Table 15 compares the absolute number of errors of the OPENNLP POS tagger relative to the number of occurrences of that tag on the basis of the STTS-MED tagset and

tag	errors	number	error ratio
ADJD	395	4,039	9.78
VVFIN	307	2,659	11.55
ADJA	219	10,358	2.11
VVINF	215	801	26.84
VVPP	209	2,123	9.84
NE	199	1,184	16.81
NN	176	23,657	0.74
ADV	169	3,517	4.81
PRELS	88	524	16.79
NNL	79	457	17.29
APPR	68	7,965	0.85
PTKVZ	59	405	14.57
ENUM	57	866	6.58
XY	50	333	15.02
KON	46	3,119	1.47
VAFIN	39	2,565	1.52
PIDAT	37	296	12.50
PIAT	34	360	9.44
ART	29	10,418	0.28
CARD	26	2,189	1.19
KOUS	24	497	4.83
PPER	23	772	2.98
PROAV	18	324	5.56
VAINF	16	524	3.05
VMFIN	12	856	1.40
APPRART	6	1,582	0.38
$(	3	2,712	0.11
PRF	3	633	0.47
$,	1	3,848	0.03
PTKNEG	1	341	0.29
$.	0	7,560	0.00
Others	454	2,115	—

Table 15: Tag-wise error ratio of the OPENNLP POS tagger on the basis of the STTS-MED tagset.

tag	errors	number	error ratio
NN	688	23,657	2.91
ADJD	629	4,039	15.57
VVFIN	575	2,659	21.62
ADJA	540	10,358	5.21
ADV	339	3,517	9.64
VVPP	330	2,123	15.54
NE	300	1,184	25.34
VVINF	300	801	37.45
PTKVZ	240	405	59.26
APPR	217	7,965	2.72
ART	128	10,418	1.23
PIDAT	109	296	36.82
NNL	100	457	21.88
PROAV	99	324	30.56
PIAT	96	360	26.67
KON	84	3,119	2.69
XY	81	333	24.32
PRELS	80	524	15.27
VMFIN	72	856	8.41
CARD	68	2,189	3.11
KOUS	68	497	13.68
VAINF	65	524	12.40
VAFIN	63	2,565	2.46
ENUM	56	866	6.47
PPER	39	772	5.05
APPRART	13	1,582	0.82
$(	2	2,712	0.07
PTKNEG	2	341	0.59
$,	1	3,848	0.03
PRF	1	633	0.16
$.	0	7,560	0.00
Others	688	2,115	—

Table 16: Tag-wise error ratio of NLTK POS tagger on the basis of the STTS-MED tagset.

the error ratio for each tag. For reasons of compactness, we only depict tags with a total number of occurrences of 250 or higher. High error numbers, obviously, do not necessarily coincide with high error ratios. The third element of the list, 'ADJA', has more than 219 erroneous annotations, yet by the sheer number of decisions, this is hardly more than 2% of all tagging decisions for this tag item. So this error is almost negligible. Similar arguments can be brought forward for 'NN', 'APPR', etc. However, care must clearly be taken in the case of 'VVFIN', 'VVINF' or 'NE' whose absolute errors are in similar ranks to the previously discussed items but whose error ratios are exceeding 10%, which is a substantial proportion of errors.

Table 16 tells a similar story for the NLTK POS tagger on the basis of the STTS-MED tagset. The highest absolute error numbers do little harm when compared with the error ratios for 'NN', 'APPR', 'ART', etc. However, 'ADJD', 'VVFIN', 'NE', etc. suffer from rather high error rates.

Based on such considerations one might argue in favor or against one or the other language model depending on the requirements of the task to be solved. Still, such data help assess the potential value of the model under scrutiny.

5. De-Compiling Models to Trace Raw Data

We here claim that the model-for-data paradigm might replace the need for accessing sensitive raw data by the alternative disclosure of language models incorporating the relevant regularities contained in the raw data. When dealing with such sensitive data, one must assure that the published models do not incorporate large continuous portions of the corpus such as complete sentences. This requirement is absolutely crucial when we distribute language models rather than the original confidential data. Yet, a large variety of ML algorithms actually store different data configuration sets directly taken or derived from the training data. Accordingly, the process of de-identification of confidential data (for medical application, cf. e.g. Meystre et al. (2014)) is by no means not made obsolete by our solution. Indeed, all person names, dates and addresses in FRAMED have been altered to block recognizability of patients.

In the context of this work, we are mainly confronted with ML algorithms based on Maximum Entropy (ME) and Conditional Random Fields (CRFs). The ME approach as well as CRFs are parametric approaches that model the data by means of weighted feature functions where the specific feature instances are derived from training data and weights are chosen to fit the model to the data. CRFs additionally create a state-transition-graph where the states correspond to the possible labels in the data, e.g. POS tags. The crucial part for both types of models are the feature functions. Those functions capture information about the actual data to base the classification decision on. One of the most important features for classifier performance is the lexical identity of a classification unit, for example in POS tagging. Thus, ME or CRF models can be used to generate an enumeration of token strings observed in the training data, resulting in a set of unigrams. Fortunately, for the use case at hand, no sequence data on the observation level (i.e. words) are stored in these models. Only a label transition model for CRFs is available. Thus, the only direct information that can be derived about the original corpus is which words are included in it.

More critical given these concerns is the NLTK sentence detector which stores the set of identified abbreviations and collocations as a main part of the model. Thus, the most frequent bi-grams are stored as collocations within the sentence detector model. Numbers are masked by the expression `##number##` for generalization purposes. The disclosure of such collocation patterns should also not reveal any sensitive information.

While we judge the disclosed information about FRAMED to be generally uncritical, this discussion shows that caution must guide the decision which ML algorithms to use taking into account the de-compilation capabilities inherent to any of the models being used.

6. Conclusion

We propose a new way to by-pass the restricted access to corpora containing sensitive, usually confidential, and thus often protected textual data. The solution we propose is based on the idea to distribute models computationally derived from such data, rather than the original, i.e. textual raw data. The models we generate are based on (trainable) software provided by well-known NLP portals such as OPENNLP or NLTK. In addition, we offer a version for our text analytics pipeline, JCORE, which is UIMA based.

As an example of access-restricted copora, we here deal with FRAMED, a textual language resource composed mainly of clinical documents. We currently make available models for sentence and token segmentation, as well as POS tagging, yet plan to cover syntactic and semantic models in the future.

We claim that our approach bears a great potential for paving the way to or, at least, improving text analytics capabilities in fields where access to raw corpus data is highly regulated, if not excluded. The free distribution of trained models (from our website) and the lack of distribution restrictions, e.g. by appropriate GPL licenses, are additional organizational means to foster software reusability under heavily restricted conditions of corpus accessibility in sensitive, often strictly confidential domains.

Still an open legal issue our work might raise is how "derivative" models are relative to the raw data. The notion of derivability is key for licensing categories formulated, e.g. under the Creative Commons (CC) framework, relating, in particular to the CC-ND (no derivative use) category. Such concerns are crucial, e.g. for processing lots of social media data (Facebook, Twitter, etc.) the majority of which are, e.g. under the CC-ND constraint.

7. References

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, Sebastopol, CA.

Chapman, W. W., Hillert, D., Velupillai, S., and *et al.* (2013). Extending the NEGEX lexicon for multiple languages. In *MedInfo 2013 – 14th World Congress on Medical and Health Informatics*, pages 677–681.

Claerhout, B. and De Moor, G. J. E. (2005). Privacy protection for clinical and genomic data: The use of privacy-enhancing techniques in medicine. *Journal of Medical Informatics*, 74:257–265.

Damschroder, L. J., Pritts, J. L., Neblo, M. A., and *et al.* (2007). Patients, privacy and trust: Patients' willingness to allow researchers to access their medical records. *Social Science & Medicine*, 64(1):223–235.

Hahn, U., Buyko, E., Landefeld, R., and *et al.* (2008). An overview of JCORE, the JULIE Lab UIMA component repository. In *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP Workshop @ LREC'08*, pages 1–7.

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

Meystre, S. M., Ferrández, O., Friedlin, F. J., and *et al.* (2014). Text de-identification for privacy protection: A study of its impact on clinical text information content. *Journal of Biomedical Informatics*.

Petrov, S., Das, D., and McDonald, R. (2012). A universal part-of-speech tagset. In *LREC '12 – 8th International Conference on Language Resources and Evaluation*, pages 2089–2096.

Schiller, A., Teufel, S., Stöckert, C., and Thielen, C. (1999). Guidelines für das Tagging deutscher Textcorpora mit STTS (Kleines und großes Tagset). Technical report, Inst. für masch. Sprachverarbeitung, U. Stuttgart; Seminar für Sprachwissenschaft, U. Tübingen.

Smallwood, R. F. (2012). *Safeguarding Critical E-Documents. Implementing a Program for Securing Confidential Information Assets*. John Wiley, Hoboken/NJ.

Tomanek, K., Wermter, J., and Hahn, U. (2007). Sentence and token splitting based on conditional random fields. In *PACLING '07 – 10th Conference of the Pacific Association for Computational Linguistics*, pages 49–57.

Vetulani, Z. (2012). Language resources in a public security application with text understanding competence. A case study: POLINT-112-SMS. In *Language Resources for Public Security Applications Workshop @ LREC '12*, pages 54–63.

Wermter, J. and Hahn, U. (2004). An annotated German-language medical text corpus as language resource. In *LREC 2004 – 4th International Conference on Language Resources and Evaluation*, volume 2, pages 473–476.