Survey of POS taggers

Approaches to making words tell who they are

DK-CLARIN WP 2.1 Technical Report
Jørg Asmussen, DSL
Final version of May 5, 2015[1]

Deliverables concerned

D10 Lemmatizer It is considered indispensable that corpus texts need to indicate the lemma form of each inflected word form in the corpus to let the user of the corpus perform more flexible queries. Therefore, it is necessary to either develop or configure a lemmatizer (that may be based on a full-form lexicon or a morphological analyzer). In the context of WP 2.1, a lemmatizer designed as an integral part of a POS tagger is the preferable solution. **Outcome:** Tool with documentation.

D11 POS tagger In order to tag tokens in corpus texts with part-of-speech information, it is necessary to either develop or configure a POS tagger (either based on a full-form lexicon or a morphological analyzer) and a suitable tag set. **Outcome:** Tool with documentation.

[1] A more recent version may be available at:
http://korpus.dsl.dk/clarin/corpus-doc/pos-survey.pdf

Outline of this document

This technical report describes WP 2.1's requirements to part-of-speech (= POS) tagging and provides a survey of existing POS approaches and their suitability. The survey is based on the requirements defined in Section 1. The report finishes with some conclusions on which approach to choose for tagging WP 2.1 corpus texts.

1 Requirements

Within a narrow DK-CLARIN context, the process of POS tagging could be reduced to just letting some kind of black box perform what is needed in order to get the text material of this corpus work-package marked-up with appropriate morphosyntactic info. Seen in this is light, the major requirement would be precision, therefore the only relevant answer allowed to ask on this topic may seem: How close to 100 % do we get? However, precision is not a quality in itself but the result of other properties.

As a black box is not configurable, and as requirements definitely will change over time and beyond the narrow scope of the DK-CLARIN project, "close to 100 %" may not be that close once some of the prerequisites of the tagging scenario have been modified. Therefore, an open configurable solution seems to be a more forward-looking approach. *Open* means that both the tagger software itself and its linguistic 'knowledge' must be open source and thus configurable, and available for free for everybody. Open source and free availability is considered a major requirement and crucial in order to achieve a permanent level of high precision.

A secondary requirement is that the tagger should not just apply a model of the language in question, i.e. Danish, and assign appropriate POS tags to words, but also perform lemmatization, i.e. assign the base forms of all (inflected) words. As

a consequence of that, the tagger should apply an open-source full form lexicon (that is available already, but definitely needs to be enhanced).[2]

Another requirement is that the tagger should be written in a widely-used, platform-independent programming language that also is used as major coding language in at least one of the DK-CLARIN corpus work-packages and thus provides a comprehensive API. As WP 2.1 uses Java and probably has the largest bulk of text to tag, Java is considered the programming language of choice for the tagger.

The tagger must be adaptive to various needs. For small occasional tagging purposes, a web-based solution seems optimal. In order to make a web-based tagger fit into specific text processing lines, it should come as a web service as well. Finally, the software should be executable on a stand-alone PC or workstation in order to process vast amounts of text quickly and without the need to access remote services.

Moreover, the tagger should be well-documented and continuously maintained (and enhanced) by a community rather than one single developer. It should be user-friendly to set up and get running.

To sum up, the following requirements given in prioritized order are considered crucial:

Availability: Free open-source tagger code and linguistic resources

Features: Capable of performing both POS tagging and lemmatization

Code: Tagger coded in Java or at least providing a Java API

Architecture: Flexible architecture adaptive to various usage scenarios

Usability: Well-documented and continuously maintained, user-friendly

2 Survey

In this section, various taggers are presented in arbitrary order and described according to the requirements given in the section above. The section is subdivided in a listing of universal taggers, i.e. taggers that in principle are language-independent, and taggers, specifically designed for or adopted to Danish.

2.1 Universal taggers

A comprehensive list of taggers can be found on Stanford University's NLP site[3]. Evert and Giesbrecht (2009) give an evaluation of the performance on German of some of these taggers, i.e. TreeTagger, TnT, SVMTagger, Stanford tagger, and the Apache UIMA Tagger.

[2] See http://korpus2000.dk/e-resurser/boejningsformer_download.php?lang=uk.

[3] http://nlp.stanford.edu/links/statnlp.html

1. IMS's TreeTagger – a language-independent POS tagger[4]

   Availability: Free for academic use, easy download, comes with free language models for approximately 10 languages, but not Danish. Not open source. Assessment: **poor**.

   Features: HMM[5] tagger using decision trees for smoothing. Performs lemmatization if a full form lexicon is supplied. Assessment: **good**.

   Code: Comes as precompiled binaries, a Java API is available, but no java source files. Assessment: **fair**.

   Architecture: The Java wrapper makes it possible to adopt the tagger to various usage-scenarios. Assessment: **good**.

   Usability: The tagger was developed as part of the project *Textcorpora und Erschließungswerkzeuge* (1993-1996) at the IMS (same project as Corpus Workbench, CWB/CQP) and has not changed substantially since then. Documentation is sparse (included as a *read me* in the download package) but probably enough to both use the software for training and tagging purposes. The tagger is maintained by one single person (Helmut Schmid). However, it seems to have a fairly large user community that partly overlaps with those who use CQP as well. Assessment: **fair**.

2. TnT[6]

   Availability: Free of charge for non-commercial research purposes. Download requires signed license agreement. Comes with language models for German and English. Not open source. Assessment: **poor**.

   Features: HMM tagger with standard smoothing. Does not perform lemmatization. Assessment: **fair**.

   Code: Comes as precompiled binaries, no wrappers available. Assessment: **poor**.

   Architecture: May work in an I/O pipeline setting. Assessment: **poor**.

   Usability: The tagger was developed by Thorsten Brants at Saarland University 1993-1999 and does not seem to have changed substantially since then. Documentation[7] is OK. The tagger does not seem to be actively maintained any longer. Assessment: **poor**.

3. SVMTool[8]

[4] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/
[5] Hidden Markov Models.
[6] http://www.coli.uni-saarland.de/~thorsten/tnt/
[7] http://www.coli.uni-saarland.de/~thorsten/publications/Brants-TR-TnT.pdf
[8] http://www.lsi.upc.es/~nlp/SVMTool/

Availability: Open source with models for Catalan, English, and Spanish. However, it must be trained by using the non open-source SVMlight software which can be used for free for academic purposes. Assessment: **fair**.

Features: Based on Support Vector Machines. Configurable in many ways. No lemmatization. Assessment: **fair**.

Code: C++ and Perl versions, Perl API. Assessment: **fair**.

Architecture: Can work in an I/O pipeline setting. Assessment: **poor**.

Usability: The SVMTool has been developed at the TALP Research Center NLP group at Universitat Politècnica de Catalunya. Latest version seems to be from 2006. Documentation[9] is OK. Assessment: **fair**.

4. Stanford Log-linear Part-Of-Speech Tagger[10]

Availability: Open source. Models for English, Arabic, Chinese, and German. Assessment: **good**.

Features: Based on the Maximum Entropy framework. It can be trained on any language on a POS-annotated training text for the language. No lemmatization. Assessment: **fair**.

Code: Java implementation. Assessment: **good**.

Architecture: Open source, thus easy integration in other environments. Assessment: **good**.

Usability: Comes with good documentation and seems well-maintained and up-to-date. Literature pointers on up-to-date website. Java NLP user lists are available for further information. However, applying the tagger to other languages than those with pre-compiled models, seems rather challenging. Assessment: **fair**.

5. Apache UIMA Tagger[11]

Availability: Open source. Comes with models for English and German. Assessment: **good**.

Features: HMM tagger as part of the *Apache Unstructured Information Management Architecture* (UIMA) framework. No lemmatization. Assessment: **fair**.

Code: Java. Assessment: **good**.

Architecture: Flexible. Web service integration as component of the framework. Assessment: **good**.

[9] http://www.lsi.upc.edu/~nlp/SVMTool/SVMTool.v1.3.pdf
[10] http://nlp.stanford.edu/software/tagger.shtml
[11] http://uima.apache.org/sandbox.html

Usability: Website and documentation seems OK even if latest website updates are from 2009. However, the UIMA framework has reached a degree of complexity (obscure code interdependencies) that makes the use of the tagger component rather cumbersome. Assessment: **poor**.

6. Chris Biemann's unsupos – unsupervised POS tagging[12]

Availability: Open source. Models for a number of languages available including Danish. It is not clear what type of material the Danish model is based on. Assessment: **good**.

Features: Unsupervised POS tagging. Does not require an annotated training corpus. Instead, word categories are determined by analyzing a large sample of monolingual, sentence-separated plain text. The tag set can probably not be determined by the user/linguist. No lemmatization. Assessment: **poor**.

Code: Java implementation. Assessment: **good**.

Architecture: Probably easy to integrate in various environments. Assessment: **good**.

Usability: Documentation is sparse, homepage and maintenance do not seem to be quite up-to-date. Assessment: **poor**.

[12] http://wortschatz.uni-leipzig.de/~cbiemann/software/unsupos.html

7. Eric Brill's simple rule-based part of speech tagger[13]

   Availability: Source code accessible at Plymouth Tech.[14] Assessment: **good**.

   Features: Based on rules derived from a training corpus. No lemmatization. However, absence of lemmatization may be resolved by modifying the code and implementing a lexicon (if the tagger is open source). Assessment: **fair**.

   Code: Originally implemented in C. Also implemented in Python as part of NLTK[15]. An interesting implementation of Brill's ideas combined with an HMM approach is the Erlangen-Tagger[16] though documentation of this approach seems poor and it is not open source. The same approach however is applied by Sujit Pal[17] whose Java code is available as open source (see also the next tagger reviewed here). Assessment: **good**.

   Architecture: Depending on the implementation, the tagger can be easily adopted to various conditions. Assessment: **good**.

   Usability: Especially the Python and the Java implementations come with good documentation. Assessment: **good**.

8. Sujit Pal's HMM-based tagger[18]

   Availability: Source code available from Sujit Pal's blog. Comes with a model for English derived from the Brown Corpus. Assessment: **good**.

   Features: HMM tagger based on Konchady (2006). No lemmatization. However, as the code is well-structured and not too complex, other features may be added. Sujit Pal is a software developer, not a linguist. His example makes some linguistic simplifications that may conceal the actual capabilities of his implementation. Assessment: **fair**.

   Code: Java. Assessment: **good**.

   Architecture: The code is well-documented and can easily be adopted to various needs. Assessment: **good**.

   Usability: Documentation is OK, code is clear and easy to modify. Assessment: **good**.

[13] http://en.wikipedia.org/wiki/Brill_tagger

[14] http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z

[15] http://www.nltk.org/

[16] http://www8.informatik.uni-erlangen.de/en/demosdownloads.html

[17] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-rule-based-pos-tagger.html

[18] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-hmm-based-pos.html

9. alias-i's LingPipe[19] toolkit

   Availability: Commercial. Assessment: **poor**.

   Features: HMM tagger. No lemmatization. Assessment: **fair**.

   Code: Java API. Assessment: **fair**.

   Architecture: Because of the Java API, integration in various settings seems feasible. Assessment: **fair**.

   Usability: Seems well-documented. POS tutorial[20] available on homepage. Assessment: **good**.

10. Jitar[21] is a simple trigram HMM POS tagger

    Availability: Open source. However, the code provided on the project homepage is incomplete.[22] Assessment: **poor** (as code is incomplete).

    Features: Simple trigram HMM tagger. No lemmatization. Assessment: **fair**.

    Code: Java. Assessment: **good**.

    Architecture: In principle, easy integration. Assessment: **fair**.

    Usability: Does not seem too complex which probably would make it fairly easy to use. However, it is maintained by just one person who already announced that Jitar development will be discontinued in favor of Jitar's C++ counterpart Citar.[23] Assessment: **poor**.

2.2 Taggers for Danish

Only two established taggers seem to be available although some others may be around as well. However, they may be narrowly tied to certain (closed) projects or companies.

1. CST's POS tagger

   Availability: Brill's allegedly modified code can be downloaded from the site of the *Centre for Language Technology* (CST) as "open source". CST has trained the tagger on DSL's publicly accessible PAROLE Corpus and thus derived a language model for Danish which they have decided not to give open public access to. Access is only given to an online version of the tagger after prior agreement. Neither conditions nor contents of the agreement are accessible on-site. Assessment: **poor**.

[19] `http://alias-i.com/lingpipe/index.html`
[20] `http://alias-i.com/lingpipe/demos/tutorial/posTags/read-me.html`
[21] `http://github.com/danieldk/jitar`
[22] Classes `LanguageModel.java` and `LinearInterpolationLM.java` have no contents.
[23] `http://langkit.org/`

Features: Based on Brill's rule-based framework. POS tagging only. However, CST provides a non-free lemmatizer with restricted access. Assessment: **fair**.

Code: C/C++. Assessment: **fair**.

Architecture: Restricted access to a web version only, not really suited for huge amounts of text. Assessment: **poor**.

Usability: A moderate amount of additional info and a demo is found on the tagger homepage. Assessment: **fair**.

2. VISL's Constraint Grammar parser

Availability: The VISL CG-3 software has been developed by GrammarSoft and is distributed as open source under the GNU General Public License. However, the Danish grammar DanGram is not publicly available and tagging/parsing of Danish can only be performed via text-by-text upload[24] or through a paid-for remote interface (accessed via the web). Conditions and prices have to be negotiated with GrammarSoft in advance. Assessment: **poor**.

Features: Based on the Constraint Grammar framework (Karlsson et al. (1995)), performs POS tagging, lemmatization and syntactic parsing. It is claimed to have a particularly high precision. Assessment: **good**.

Code: C++. Assessment: **fair**.

Architecture: Restricted access to web-based versions only, not really suited for larger amounts of text. Assessment: **poor**.

Usability: A comprehensive manual, a tutorial, examples, demos, and additional info is found on the homepage. Assessment: **good**.

2.3 Conclusions

As availability is considered a major requirement, the following taggers are of particular interest to the WP 2.1 project: Stanford, Apache UIMA, unsupos, Brill, and Sujit Pal's tagger implementations. Common to all these taggers is that they derive their language model from a training corpus and that they principally work as POS taggers only. The disadvantage of this is that lemmatization comes in as a separate process that requires specific tools or extensions to the existing implementations. In addition, unknown words, i.e. words not seen in prior training material, seem to be a problem for all taggers based on learning algorithms that produce language models.

However, some of the mentioned taggers may be modified to also take into account lexical knowledge and perform lemmatization as well, in particular Brill and

[24] Uploaded texts will be added to VISL's own corpora if their copyright status permits it.

Sujit Pal. Stanford and UIMA may be extendable as well, but their code is rather complex which probably makes the development of extensions difficult. As for unsupos, the unsupervised learning approach probably is not suitable for the needs of WP 2.1. Thus, it emerges that Sujit Pal's HMM and Brill implementations may be the most attractive solutions to start with. Stanford may be an alternative whereas UIMA seems far too complex for the needs of POS tagging only. It is a pity that both taggers specifically designed for handling Danish have severe usage restrictions, otherwise they might have been worth giving a try as well.

The conclusion is to conduct a case study with Sujit Pal's HMM implementation where it will be trained on the Danish Parole Corpus to evaluate the potential of his HMM approach. If it fails, Stanford can be considered as a fallback option.

3 Case study

The starting point of the case study is the Java code of Sujit Pal's HMM implementation[25] including the Java HMM library Jahmm by Jean-Marc François[26], University of Liège. Sujit Pal's demo is based on building an HMM from the Brown Corpus; in this study the tagged training corpus is the Danish PAROLE Corpus[27].

3.1 Building a token-based HMM

The common approach of modeling an HMM is to view the word forms of a text as visible observations and the set of possible POS tags as hidden states. In the following experiment, this approach, which is also demonstrated on Sujit Pal's blog, is applied to a setting for Danish.

The first step was to convert PAROLE from its TEI-like XML-structure to the necessary Brown input format, as illustrated here:

```
To/AC kendte/AN russiske/AN historikere/NC Andronik/NP
Mirganjan/NP og/CC Igor/NP Klamkin/NP tror/VA ikke/RG ,/XP
at/CS Rusland/NP kan/VA udvikles/VA uden/SP en/PI
"/XP jernnæve/NC "/XP ./XP

De/PP hævder/VA ,/XP at/CS Ruslands/NP vej/NC til/SP
demokrati/NC går/VA gennem/SP diktatur/NC ./XP

[...][28]
```

As can be seen, the original PAROLE tags have been cut off after two characters, the first character giving the POS, the second one giving a POS sub-classification.

[25] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-hmm-based-pos.html
[26] http://www.montefiore.ulg.ac.be/~francois/
[27] http://korpus.dsl.dk/e-resurser/parole-korpus.php?lang=uk

Inflectional information is not present in the tags used here – a simplification that should be avoided in a real setting.[29]

Building an HMM from PAROLE is quite straightforward and it actually does POS tagging afterwards, however some problems need to be addressed:

1. The PAROLE Corpus is quite small in size, approximately 250,000 tokens
2. The tag set has not been adjusted to the actual needs and is probably too large (25 different tags) to allow building an HMM from a corpus of this size
3. Only sentences with known word forms, i.e. such contained in PAROLE, can be tagged on the basis of this HMM

As corpus size cannot be augmented within the scope of the ongoing project, to cope with these restrictions, the tag set should be optimized as should the elements of the observable layer of the HMM. At the moment these are word forms but it might be worth trying to map them to more abstract representations by using a lexicon. The following experiment will address this approach.

3.2 Building a lexicon-based HMM

The idea behind this approach is to map word forms in the text to their possible POS tags prior to training and analyzing by applying a lexicon. During the training phase, possible tags for a given token constitute the observable layer and the actual tag the hidden state. Information not relevant to the disambiguation process should not be given in the tags at this state. Similarly, during tagging, the word forms of the text in question are mapped to this simple tag-set. The design of the jaPOS tagger described in Asmussen (2014) focuses on this approach.

[29] A comprehensive account of the the tag set used in PAROLE can be found in Keson (1998b) and an abridged version in Keson (1998a).

4 Document history

The most recent version of this report can be downloaded from:

- http://korpus.dsl.dk/clarin/corpus-doc/pos-survey.pdf

5 References

Asmussen, J. (2014). Design of the ePOS tagger. Technical report, DK-CLARIN, korpus.dsl.dk/clarin/corpus-doc/pos-design.pdf.

Evert, S. and Giesbrecht, E. (2009). Part-of-speech tagging – a solved task? An evaluation of POS taggers for the Web as corpus. In Alegria, I., Leturia, I., and Sharoff, S., editors, *Proceedings of the 5th Web as Corpus Workshop (WAC5)*, San Sebastian, Spain.

Karlsson, F. et al. (1995). *Constraint Grammar – A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter.

Keson, B. K. (1998a). Documentation of The Danish Morphosyntactically Tagged PAROLE Corpus. Technical report, DSL, korpus.dsl.dk/e-resurser/paroledoc_en.pdf.

Keson, B. K. (1998b). Vejledning til det danske morfosyntaktisk taggede PAROLE-korpus. Technical report, DSL, korpus.dsl.dk/e-resurser/paroledoc_dk.pdf.

Konchady, M. (2006). *Text Mining Application Programming*. Programming Series. Charles River Media, 1 edition.