FlexTag: A Highly Flexible PoS Tagging Framework

Torsten Zesch, Tobias Horsmann
Language Technology Lab
Department of Computer Science and Applied Cognitive Science
University of Duisburg-Essen, Germany
{torsten.zesch,tobias.horsmann}@uni-due.de

Abstract

We present FlexTag, a highly flexible PoS tagging framework. In contrast to monolithic implementations that can only be retrained but not adapted otherwise, FlexTag enables users to modify the feature space and the classification algorithm. We categorize existing PoS tagger implementations into one of three categories with regards to model-training capabilities and the level of access those implementations give a researcher to intrinsic details. To this categorization we add a new fourth layer characterizing our FlexTag which fully exposes the feature space and the machine learning algorithm while sustaining a high usability. With FlexTag, rapid prototyping of tagger models using different feature spaces can be easily implemented, taking a huge technical burden from the NLP researcher who is experimenting with new ideas or resources. FlexTag makes it easy to quickly develop custom-made taggers exactly fitting a research problem. We demonstrate the capabilities of FlexTag by first training a PoS tagger model with state-of-the-art performance on the common Wall-Street-Journal split using a default feature set. Furthermore, we train a social media model fitted to the Twitter domain that extends the default feature set by adding domain-specific features and incorporating knowledge from unsupervised data resources.

Keywords: PoS tagging, domain adaptation, feature engineering

1. Motivation

Part-of-speech tagging is an important preprocessing step in natural language processing. Consequently, there are many implementations available such as the Stanford tagger (Toutanova et al., 2003), the TreeTagger (Schmid, 1994), or the ClearNLP tagger (Choi and Palmer, 2012). In order to adapt the behaviour of the tagger to the needs of the user, most taggers provide configurable *models* which are targeted towards different languages or domains. A user might e.g. switch between a standard English model and a German social media model depending on the task at hand. As there rarely is a perfect match between the available tagging models and the documents to be processed, most taggers allow users to train their own models. FlexTag goes beyond the usual re-training possibilities of established taggers by giving the user flexible control over the feature extraction and classification step. Users can choose from different machine learning algorithms or plug-in their own learning framework. FlexTag already comes with a wide range of implemented feature extraction modules that can be enabled when needed. However, users can also write new feature extraction modules. For example, unsupervised information from Brown clusters (Ritter et al., 2011) or LDA (Rehbein, 2013) is known to boost performance quite a bit, but the corresponding features are usually not supported in standard taggers.

FlexTag is embedded in an evaluation framework (Horsmann et al., 2015) that allows for quick comparison of the newly created taggers with previous versions and other implementations.

2. FlexTag Architecture

In order to better understand in which way FlexTag differs from existing PoS taggers, we distinguish four commonly used architectures – see Figure 1 for an overview.

Fixed Model Some taggers cannot be changed at all (without rewriting the tagger code itself) because model and features are hard-coded in the implementation. This is often the case for proof-of-concept implementations that might directly implement an optimized machine learning classifier or a domain-specific feature set. Figure 1a shows how taggers with a fixed model look from the user perspective. A fixed-model tagger is basically a big black box that accepts raw text as input and outputs tagged text.

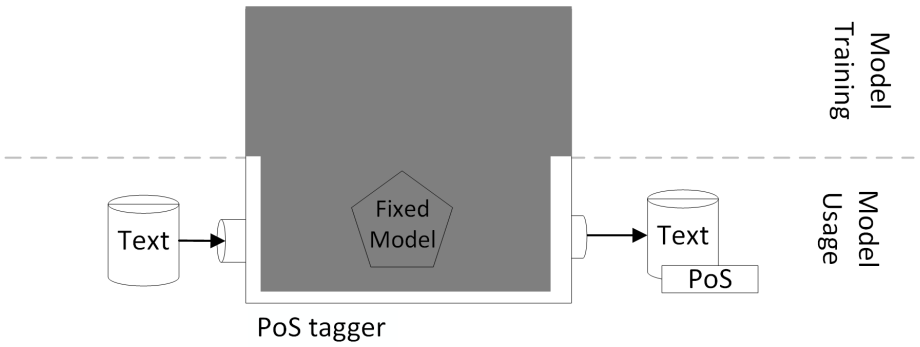
Replaceable Model The next step on the flexibility continuum are taggers with replaceable models (see Figure 1b). Here, the user can change the behavior of the tagger by choosing from a set of provided models, but the tagger itself provides no means for creating a model. An example is the rule-based Hepple tagger (Hepple, 2000), where a rule set for English is provided. Rulesets for other languages can be specified, but there is no method provided for creating new models from training data.

Trainable Model A major step towards really custom-made taggers is to let users train their own models as shown in Figure 1c. While the tagger is still a black box, it provides an additional interface to turn PoS annotated training data into a custom-made model.

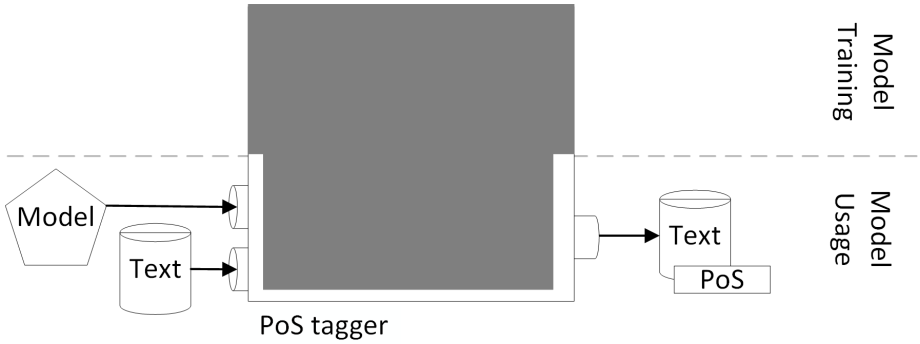
Once the model is trained, the tagger works exactly like in the *replaceable model* case. Examples for trainable taggers are Stanford (Toutanova et al., 2003) or TreeTagger (Schmid, 1994).

Flexibly Trainable Model The hard-coded feature set of trainable taggers complicates the process of adapting them to new domains. In our FlexTag architecture (see Figure 1d), we thus give the user flexible control over which feature extraction and machine learning should be used. A similar approach was taken by SVMTool (Giménez and Màrquez, 2004), but it does not allow to implement own features, only to parameterize existing features.

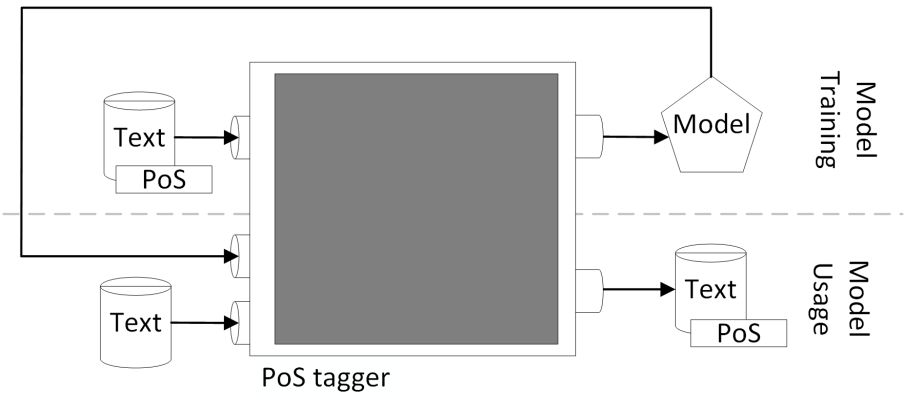
The main remaining challenge when exposing more parameters of a PoS tagger to the user is how to keep using and training the tagger usable also for non-experts.



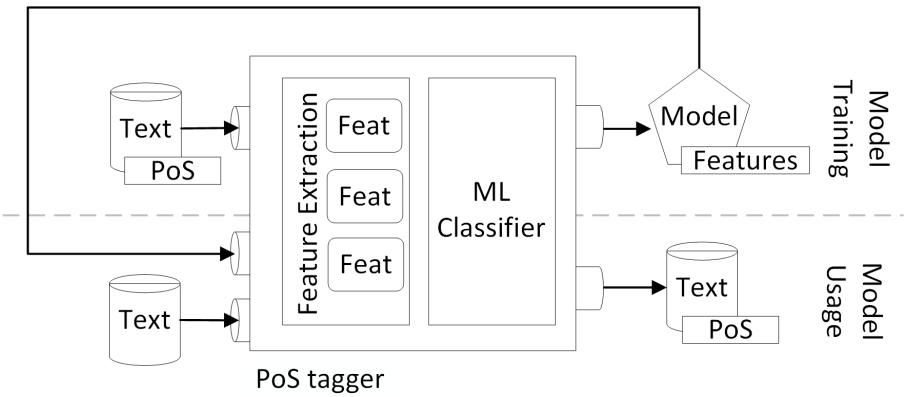
(a) Fixed model



(b) Replaceable model



(c) Trainable model



(d) Flexibly trainable model

Figure 1: The different levels of flexibility in adapting the behavior of a PoS tagger

3. Keeping it Usable

A key danger when exposing more functionality to the user is that using and adapting the tagger gets more complicated. We now explain how we keep both processes as simple as possible.

Using FlexTag As FlexTag is implemented in Java, it does not need to be installed but runs wherever a JVM is available. The user only needs to download the library, while existing FlexTag models are automatically downloaded upon first usage without any additional user intervention. For using a model, users need to specify a target language and the model type, e.g. trained on newswire vs. social media or slow but accurate vs. fast classifier.

In order to make this simplicity possible, FlexTag relies on DKPro Core[1] (Eckart de Castilho and Gurevych, 2014) for preprocessing and model loading, and DKPro TC (Daxenberger et al., 2014) for feature extraction and classification. FlexTag can be used standalone or as an Apache UIMA component (Ferrucci and Lally, 2004) within a more complex processing pipeline.

Training FlexTag Most other trainable taggers only support one input format and users are supposed to transform their data in the required format. In contrast, FlexTag makes no assumptions about the input format and relies on the UIMA reader concept supporting all readers that are compatible with the DKPro type system. For most common data formats, DKPro Core already provides the necessary readers for converting the corpus format in the correct internal representation. Supported formats include BNC, Brown, IMS-CWB, Negra, PTB, and TEI.[2]

Adding Features FlexTag already comes with a wide range of implemented feature extraction modules that can be enabled when needed. However, as it is impossible to foresee all future uses, we also let users add their own features. For example, when processing Twitter data it might be useful to detect user mentions (like `@TorstenZesch`) in order to reliably assign a specific tag (Ritter et al., 2011). Technically, users need to provide a self-contained Java class that implements the FlexTag interface for feature extractors. In case of PoS tagging, this is a unit or sequence classification interface (see Daxenberger et al. (2014) for the different classification modes) where features are extracted for each token separately. The extractor interface exposes the UIMA CAS (Götz and Suhre, 2004), an in-memory representation of the whole text from which all pre-processing results are easily accessible as stand-off annotations. Listing 1 shows the full source code of a feature extractor that detects user mentions in tweets. In this case, we simply request the text of the current token and check whether it starts with an @ sign. However, more complicated actions like accessing neighbouring tokens are easily possible.

Being able to modify the feature extraction step makes it necessary to store a representation of the utilised feature extractors with the model, while for taggers with a fixed feature-set the model is usually simply a persisted version of the machine learning classifier.

Context Features
current word (w_i)
previous word
next word
w_i length in character
Boolean Features ($\forall$ characters)
w_i contains only capital letters
w_i contains only special characters
w_i contains only numbers
Boolean Features ($\exists$ character(s))
w_i starts with a capital letter
w_i contains one or more numbers
w_i contains one or more hyphens
w_i contains one or more periods
Ngram Feature
1000 most frequent character ngrams (n=2-4) of w_i

Table 1: Default feature set

Switching Classifiers As FlexTag relies on DKPro TC, we can easily switch between all the provided classifiers by changing the configuration without having to change any code. Besides the well-known Weka framework (Hall et al., 2009), DKPro TC currently includes two sequence classification libraries that are of special interest for PoS tagging: CRFsuite (Okazaki, 2007) and SVMhmm (Joachims et al., 2009).

4. Use Case: Default Tagger

Even if FlexTag is all about defining your own features, there is a set of standard features that usually work well and are thus activated by default. Default features include the preceding and following word, the top 1000 most frequent 2-4 character ngrams, and boolean features testing if a word uses capitalized letters, hyphenation, periods, or is numeric. Table 1 gives an overview of the default set. Users are however free to *not* use these features, i.e. the full feature space is fully customizable and if other features definition for the task at hand seem more suited FlexTag does not prevent the user from building an own feature space.

When configuring the default tagger to use CRFSuite in standard configuration and using sections 0-18 of the Wall Street Journal (WSJ) corpus (Marcus et al., 1993) for training, we yield an accuracy of 96.3%. The state of the art[3] ranges from 96.5% (Brants, 2000) to 97.6% (Huang et al., 2015) accuracy. As we did not tune any parameters, we consider this result to be on par with the state of the art.

5. Use Case: Social Media Tagger

In order to create a social media tagger, we can simply train our default tagger on manually annotated Twitter data. However, tagging social media is harder than tagging news,

[1] `https://dkpro.github.io/dkpro-core`

[2] `https://dkpro.github.io/dkpro-core/releases/1.7.0/formats/`

[3] `http://aclweb.org/aclwiki/index.php?title=POS_Tagging_(State_of_the_art)`

```java
/**
 * Detects user mentions in tweets
 */
public class IsUserMentionFeatureExtractor
  extends FeatureExtractorResource_ImplBase
  implements ClassificationUnitFeatureExtractor
{
  public Set<Feature> extract(JCas jcas, TextClassificationUnit unit)
  {
    return new Feature(
      "isMention",
      unit.getCoveredText.startsWith("@") ? 1 : 0
    ).asSet();
  }
}
```

Listing 1: Example of a FlexTag feature extractor that detects user mentions in tweets

as there is usually only a small amount of PoS annotated data available and at the same time there is higher variability in spelling (Horsmann and Zesch, 2015). A typical dataset is provided by Ritter et al. (2011), containing 15k tokens compared to 1200k in the WSJ corpus. When evaluating the default tagger trained on this dataset using 10-fold cross validation, we reach an accuracy of 78.5%. A massive, but expected, drop compared to the 96.3% on the newswire WSJ corpus.

As FlexTag allows user-defined features, we can try to increase results by making the tagger fit the domain better. We add features for detecting user-mentions, hashtags, mail addresses, and URLs as those are easily recognizable based on regular expressions and frequently used in Twitter. This increases performance to 79.1%.

Due to the high variability in spelling, we cannot expect the training data to cover all surface forms and should better add some unsupervised knowledge. FlexTags already ships with a selection of feature extractors based on unsupervised methods (Brown clusters, LDA, and PoS dictionaries), but users are of course free (and invited) to experiment with other methods. Selecting Brown clusters, that are especially easy to use as Owoputi et al. (2013) provide precomputed clusters, we improve accuracy by 7.3 percent points to 86.4%.

To summarize: By adding few domain specific features and one resource, we implemented a domain fitted PoS tagger that improved tagging accuracy by 7.9 percent points over our default tagger. This shows that experimenting with (new) resources and ideas of feature engineering is no longer limited by the fixed feature sets that were dictated by previous PoS tagger implementations.

6. FlexTag Project

We present FlexTag, a highly flexible PoS tagging framework. The project is hosted on GitHub under the URL `https://github.com/Horsmann/FlexTag`. The website contains tutorials on how to train domain-specific models, how to define and configure special features, how to integrate FlexTag into existing experimental setups, and how to deploy FlexTag with an own model as standalone component.

While we see FlexTag mainly as a framework for researchers to build custom-made models exactly fitting their task, we nevertheless provide a rich set of pre-trained models for many languages and domains. FlexTag is publicly available under the Apache License 2.0.

Acknowledgments

We thank Johannes Daxenberger and Richard Eckart de Castilho for their constant support and advice.

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant No. GRK 2167, Research Training Group “User-Centred Social Media”.

7. Bibliographical References

Brants, T. (2000). Tnt: A statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, ANLC ’00, pages 224–231, Stroudsburg, PA, USA. Association for Computational Linguistics.

Choi, J. D. and Palmer, M. (2012). Fast and robust part-of-speech tagging using dynamic model selection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers – Volume 2*, ACL ’12, pages 363–367, Stroudsburg, PA, USA. Association for Computational Linguistics.

Daxenberger, J., Ferschke, O., Gurevych, I., and Zesch, T. (2014). DKPro TC: A java-based framework for supervised learning experiments on textual data. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 61–66. Association for Computational Linguistics.

Eckart de Castilho, R. and Gurevych, I. (2014). A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT (OIAF4HLT) at COLING 2014*, pages 1–11, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Ferrucci, D. and Lally, A. (2004). UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348.

Giménez, J. and Màrquez, L. (2004). SVMTool: A general pos tagger generator based on support vector machines. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC-2004)*, Lisbon, Portugal, May. European Language Resources Association (ELRA). ACL Anthology Identifier: L04-1373.

Götz, T. and Suhre, O. (2004). Design and implementation of the UIMA common analysis system. *IBM Syst. J.*, 43(3):476–489, July.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November.

Hepple, M. (2000). Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong.

Horsmann, T. and Zesch, T. (2015). Effectiveness of domain adaptation approaches for social media pos tagging. In *Proceeding of the Second Italian Conference on Computational Linguistics*, pages 166–170, Trento, Italy. Accademia University Press.

Horsmann, T., Erbs, N., and Zesch, T. (2015). Fast or Accurate ? – A Comparative Evaluation of PoS Tagging Models. In *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology (GSCL-2015)*, Essen, Germany.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Joachims, T., Finley, T., and Yu, C.-N. J. (2009). Cutting-plane training of structural svms. *Mach. Learn.*, 77(1):27–59, October.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330.

Okazaki, N. (2007). CRFsuite: a fast implementation of Conditional Random Fields (CRFs).

Owoputi, O., Dyer, C., Gimpel, K., Schneider, N., and Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Rehbein, I. (2013). Fine-Grained POS Tagging of German Tweets. In Iryna Gurevych, et al., editors, *Language Processing and Knowledge in the Web*, volume 8105 of *Lecture Notes in Computer Science*, pages 162–175.

Ritter, A., Clark, S., Mausam, and Etzioni, O. (2011). Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1524–1534, Stroudsburg, PA, USA.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology – Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

DKPro Organisation. (2014). *DKPro Core*. DKPro Organisation, NLP resources, 1.7.