Documentation for

JULIE Lab Part of Speech Tagger

Version 1.0

Johannes Hellrich
Jena University Language & Information Engineering (JULIE) Lab
Fürstengraben 30
D-07743 Jena, Germany
johannes.hellrich@uni-jena.de

1 Objective

The JULIE Lab Part of Speech Tagger (JPOS) is a generic and configurable POS tagger. JPOS was tested on the general-language news paper domain and in the biomedical domain; it performs very good for German texts, yet only mediocre for English [HMFH15].

As JPOS employs a machine learning (ML) approach (see Section 7), a model (for the specific domain and entity classes to be predicted) needs to be trained first. Thus, JPOS offers a training mode. Furthermore, JPOS also provides an evaluation mode to assess the current model performance in terms of accuracy.

2 About this documentation

This is a documentation on the functionality of JPOS, especially when used in a standalone manner. When using the UIMA-compliant version of JPOS, please refer to the UIMA-JPOS documentation for additional information.

3 Changelog

1.0 Initial release.

4 Installation

After extracting the JPOS-tgz package, nothing more has to be done. The program is written in Java. To run JPOS you need a Java 1.7 (or above) runtime environment installed on your system. In addition to the common Java libraries, JNET employs MALLET [McC02], a machine learning toolkit, and UEASTEMMER, a conservative word stemmer[1].

5 File Formats

In this section, the file formats relevant to JPOS are introduced. The first subsection explains the format for annotated texts used as training material or produced during annotation. The second subsection shows in detail how JPOS may be configured.

5.1 Annotation format

All *training files* need to have the following format: one sentence per line, tokens separated by whitespaces and followed by their respective POS tag, using the pipe symbol ("|") as a separator, e.g.:

```
These|DT mutations|NNS have|VBP been|VBN localized|VBN .|$.
```

POS tags can be chosen arbitrarily.

5.1.1 Feature Configuration File

A configuration file may be passed to JPOS where the features to be used can be parameterized. Both the training mode and the evaluation modes (because they include model training as well) can consume such a file.

The information within a configuration file serves to customise the behaviour of JPOS in creating its ML features. As the actual feature instances are generated depending on the respective training material, a configuration file together with the training material determines the features (and thus the model).

Next, details to the configurations are given. Generally, a configuration file consists of key-value pairs, one in a line. See Table 1 for an enumeration of these key-value pairs.

There are simple features, which can just be turned on or off (e.g. whether word stemming should be used or not), and more configurable features for which, when turned on, some

[1] `http://www.cmp.uea.ac.uk/Research/stemmer/`

parameter can be set (such as the context feature for which the size of the context can be set).

KEY	ALLOWED VALUES	DESCRIPTION
feat_lowercase_enabled	true/false	if enabled, tokens beginning with a capital letter are modified to lower case (this is done only if and only if the beginning letter is upper case
feat_wc_enabled	true / false	enables or disables the word class feature
feat_bwc_enabled	true / false	enables or disables the brief word class feature
feat_bioregexp_enabled	true / false	enables or disables some features primary used for bio or bio-medical texts
feat_plural_enabled	true / false	if enabled, this feature is activated in case the only difference between the stemmed and the unstemmed version of a token is a putative plural "s"
token_ngrams	integer list	defines the token-level ngrams to be generated as features. If uncommented from the feature configuration, no token ngrams are built (not subject to offset conjunction). Example: 2,3; ngrams of size 2 and 3 are built
char_ngram	integer list	ngrams on the character level (not subject to offset conjunction)
prefix_sizes	integer list	the prefixes to be build according to the specified length. Example: 2,3; prefixes of 2 and 3 characters are build
suffix_sizes	integer list	the suffixes to be build (compare to prefix_sizes)

offset_conjunctions	integer list	determines the feature generation environment of a token and combinations of token features; numbers correspond to token positions relatively to the actually viewed token. (0) stands for the actual token, (-1) for the preceding token etc. (-2) (-1) (0) (1) indicates that features for the tokens (-2), (-1), (0) and (1) are generated. something like (-1 -2) or (-1, -2) would combine the features of (-1) and (-2)
gap_character	character	character that serves for indicating that the annotation for a token is not available/not known in the training material

Table 1: Defined key-value pairs in feature configuration files.

Such a feature configuration file might look like this:

```
offset_conjunctions = (-1) (1)

feat_lowercase_enabled = false
feat_wc_enabled = true
feat_bwc_enabled = true
feat_bioregexp_enabled = true
feat_plural_enabled = true
token_ngrams = 2,3
char_ngrams = 3,4
prefix_sizes = 1,4
suffix_sizes = 1,4

# ------------------------------------------------------------
# the character for indicating that the annotation for a
# single token is not known/not available
gap_character = @
```

6 Using JPOS

JPOS is a command-line tool, supplied as an executable jar. To use JPOS you have to call:

```
java -jar <jpos-jar>
```

Depending on your system configuration you will have to increase the allotted RAM for some operations:

```
java -Xmx<required RAM in gigabyte>g -jar <jpos-jar>
```

Running JPOS without further parameters, you will be informed about the available modes:

```
usage: <mode> <mode-specific-parameters>

Available modes:
x: cross validation
c: compare goldstandard and prediction
t: train
p: predict
oc: output model configuration
ts: output tag set
```

6.1 Training

In order to train a model you need training material like described in 5.1 and a feature configuration file. The output of a training process is a model which may be used for prediction. If no `number of iterations` is set (or 0 is used) training will continue until convergence. You will probably need several GB of RAM for training on every non-trivial corpus. The arguments for training are:

```
t <trainData> <model-out-file> <featureConfigFile> [number of iterations]
```

6.2 Prediction

For tagging a given plain text you need a trained model. Annotated output is stored in the specified file. The arguments for prediction are:

```
p <unlabeled data> <modelFile> <outFile>
```

6.3 Evaluation

JPOS provides two evaluation modes. Each of them returns the performance in terms of accuracy.

6.3.1 Comparing Prediction and Gold Standard

For comparing the output of a prediction process with a given gold standard you need the prediction and the gold standard. Both are required to be text files in the format described in 5.1. They need to be of the same length. That is, the number of tokens and respectively the number of lines must match. The arguments for comparison are:

```
c <prediction> <gold>
```

6.3.2 Cross Validation

During cross validation, the prodived training material is randomly split into n subsets (*<x-rounds>* specifies the number of subsets). Then $n-1$ subsets are used for training, the remaining one for evaluation. This is repeated n times, the performance values are the mean average over the performance values of each round. Standard deviation is also shown.

The arguments for cross valiation are:

```
x <trainData> <x-rounds> <featureConfigFile> [number of iterations]
```

6.3.3 Model Configuration / Tag Set

Running JPOS with the argument *'oc'* outputs the feature configuration specified during training for the specified model, whereas *'ts'* outputs the tag set used by this model. Both modes need a model file as their second argument.

7 Background/Algorithms

JPOS is based on the JNET named entity recognizer from our lab and uses MaxEnt models.

8 Available Models

The directory `JPOS_data/models` contains two German models, trained on FraMed [WH04] and TiGER, achieving accuracy values of 97.6 and 97.7, respectively, during 10-fold cross-validation. These models can be used to annotate texts with an STTS derived tag set [WH04, STST99]

Using JPOS's training facilities your can easily train you own models – given training material is available.

References

[HMFH15] Johannes Hellrich, Franz Matthies, Erik Faessler, and Udo Hahn. Sharing models and tools for processing german clinical text

$$= proceedingsofmie2015$$

. In *Digital Healthcare Empowering Europeans*, Studies in Health Technology and Informatics, 210, pages 734–738, 2015.

[McC02] Andrew McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[STST99] Anne Schiller, Simone Teufel, Christine Stöckert, and Christine Thielen. 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, 1999.

[WH04] Joachim Wermter and Udo Hahn. An annotated German-language medical text corpus as language resource. In *LREC 2004—Proceedings of the 4th International Conference on Language Resources and Evaluation*, volume 2, pages 473–476, 2004.