Documentation for

JULIE Lab Sentence Boundary Detector

Version 2.4

Katrin Tomanek
Jena University Language & Information Engineering (JULIE) Lab
Fürstengraben 30
D-07743 Jena, Germany
`katrin.tomanek@uni-jena.de`

Contents

1 UIMA-Wrapper

The JULIE Lab Sentence Boundary Detector (UIMA-JSBD) is a sentence boundary detector for UIMA. It is part of the JULIE Lab NLP tool suite[1] which contains several UIMA-compliant NLP components from sentence splitting to named entity recognition and normalization as well as a comprehensive UIMA type system.

1.1 Installation

UIMA-JSBD comes as a UIMA pear file. Run the Pear-Installer (e.g., `./runPearInstaller.sh` for Linux) from your UIMA-bin directory. After installation, you will find a subfolder `desc` in you installation folder. This directory contains a descriptor `SentenceAnnotator.xml` for UIMA-JSBD. You may now e.g. run UIMA's Collection Proeccessing Engine Configurator (`cpeGUI.sh`) and add UIMA-JSBD as a component into your NLP pipeline.

This pear package also contains a model for sentence splitting. The model was trained on a special bio-medical corpus which consists of data from both the GENIA [**?**] and the PennBioIE[2] corpus and additional material which we took from MedLine abstracts. Currently, it comprises about 62000 sentences. An accuracy of 99.8% is yielded on this data using 10-fold cross-validation. You will find the model trained on this data in the directory `resources`.

1.2 Requirements and Dependencies

UIMA-JSBD is written in Java (version 1.5 or above required) using Apache UIMA version 2.2.x-incubation[3].

The input and output of an AE takes place by annotation objects. The classes corresponding to these objects are part of the *JULIE Lab UIMA Type System* in its current version (2.1).[4]

This version of UIMA-JSBD is based on JSBD-2.4 which employs the machine learning toolkit MALLET [McC02].

[1]`http://www.julielab.de/`
[2]`http://bioie.ldc.upenn.edu/`
[3]`http://incubator.apache.org/uima/`
[4]The *JULIE Lab UIMA type system* can be separately obtained from `http://www.julielab.de/`, however, this package already includes the necessary parts of the type system.

1.3 Using the AE – Descriptor Configuration

In UIMA, each component is configured by a descriptor in XML. In the following we describe how the descriptor required by this AE can be created with the *Component Descriptor Editor*, an Eclipse plugin which is part of the UIMA SDK.

A descriptor contains information on different aspects. The following subsection refers to each sub aspect of the descriptor which is, in the Component Descriptor Editor, a separate *tabbed page*. For an indepth description of the respective configuration aspects or tabs, please refer to the *UIMA SKD User's Guide*[5], especially the chapter on "Component Descriptor Editor User's Guide".

To define your own descriptor go through each tabbed pages mentioned here, make your respective entries (especially in page *Parameter Settings* you will be able to configure JNET to your needs) and save the descriptor as `SomeName.xml`.

Otherwise, you can of course employ the descriptor that is contained in the pear package you downloaded (in your installation directory, see `desc/SentenceAnnotator.xml`).

Overview This tab provides general informtion about the component. For UIMA-JSBD you need to provide the information as specified in Table 1.

Subsection	Key	Value
Implementation Details	Implementation Language	Java
	Engine Type	primitive
Runtime Information	updates the CAS	check
	multiple deployment allowed	check
	outputs new CASes	don't check
	Name of the Java class file	`de.julielab.jules.ae.SentenceAnnotator`
Overall Identification Information	Name	Sentence Annotator
	Version	2.4
	Vendor	JULIE Lab
	Description	not needed

Table 1: Overview/General Settings for AE.

Aggregate Not needed here, as this AE is a primitive.

[5] `http://incubator.apache.org/uima/`

Parameters See Table 2 for a specification of the configuration parameters of this AE. Do not check “Use Parameter Groups” in this tab.

Parameter Name	Parameter Type	Mandatory	Multivalued	Description
ModelFilename	String	yes	no	filename of trained model for JSBD
Postprocessing	Boolean	no	no	Indicates whether post-processing should be run. Default: no post-processing
ProcessingScope	String	no	no	The UIMA annotation type over which to iterate for doing the sentence segmentation. If nothing is given, the document text from the CAS is taken as scope! This is recommended as default!

Table 2: Parameters of this AE.

Parameter Settings The specific parameter settings are filled in here. For each of the parameters defined in 1.3, add the respective values here (has to be done at least for each parameter that is defined as mandatory). See Table 3 for the respective parameter settings of this AE.

Parameter Name	Parameter Syntax	Example
ModelFilename	full path	`resources/JSBD-2.0-biomed.mod.gz`
Postprocessing	true/false	true
ProcessingScope	full class name to annotation type	`de.julielab.jules.paragraph` (assuming you downloaded the document structure part of the JULIE Lab Type System). If you don’t know what to do here, leave it blank!

Table 3: Parameter settings of this AE.

Type System On this page, go to *Imported Type* and add the following layers of the *JULIE UIMA Type System* (Use “Import by Location”): `julie-basic-types.xml` and `julie-morpho-syntax-types.xml`. If you use the *ProcessingScope* parameter make sure that the respective type/type system is also included.

Capabilities The sentence splitter only returns annotations from type `de.julielab.jules.types.Sentence`. See Table 4.

Type	Input	Output
de.julielab.jules.types.Sentence		√

Table 4: Capabilities of this AE.

Index Nothing needs to be done here.

Resources Nothing needs to be done here.

2 JSBD: Core Functionality and Stand Alone Tool

JULIE Sentence Boundary Detector (JSBD) is a sentence splitter developed and optimized for the bio-medical domain. In contrast to most other sentence splitters which consists of simple patterns, JSBD is based on machine learning (see Section 2.4) which enables it to handle also tricky cases occuring frequently in life science documents. See [TWH07] for a more in depth description of JSDB and a performance study.

JSBD offers the following functionalities:

- training a model
- prediction using a previously trained model
- evaluation

2.1 Installation

Just unpack the tar-ball. The program is written in Java[6]. Note that JSBD was only tested with Java 1.5; you need at least the Java 1.5 runtime environment installed on your system to run JSBD. In addition to the common Java libraries, JSBD employs MALLET [McC02], a machine learning toolkit (no further installation steps are required here).

2.2 File Formats

There are two input formats for text documents. There are some example documents in directory `testdata`. All data need to be simple ASCII.

- For training and evaluation, JSBD needs to know the sentence boundaries. Therefore the documents must have exactly one sentence per line. (see directory `testdata/train/`).
- For sentence splitting, the only requirement for the documents is that they are plain text, without any XML tags etc. (see `testdata/split/`)

2.3 Using JSBD

To execute JSBD just type

```
./runJSBDpackaged.sh
```

without any arguments. This will print the following list of available modes:

[6]Java is a registered trademark of Sun Microsystems, Inc.

```
usage: JSBD <mode> {mode_specific_parameters}
different modes:
c: check texts
t: train a sentence splitting model
p: do the sentence splitting
s: evaluation with 90-10 split
x: evaluation with cross-validation
e: evaluation on previously trained model
```

When running JSBD only with the mode as its only parameter it will return the specific parameters needed for this mode.

2.3.1 Data Check

The provided data is checked for the correct format. In training mode, all training files need to be in the following format: one sentence per line. At the end of the line there should be a EOS (end-of-sentence) symbol (is defined in the EOSSymbols class). If this is not the case, an error message is thrown.

```
./runJSBDpackaged.sh c

-> usage: JSBD c <textDir>
```

¡textDir¿ is the directory with the text documents to be checked, all files in this directory are considered

2.3.2 Evaluation

There are two evaluation modes to evaluate the sentence splitter on given training material. Performance is measured in terms of accuracy, i.e. the number of correct decisions divided by the total number of decisions being made.

90-10 split evaluation the given data is split into two data sets, 90% of the files are used for training the model, the other 10% are used to evaluate the trained model (splits are made on file level, i.e. you should make sure, that the files are more or less of the same size)

```
./runJSBDpackaged.sh s

-> usage: JSBD s <textDir> <errorFile>
```

¡textDir¿ is the directory with the text documents used for evaluation (i.e. the same format as the training data). All files in this directory are considered. *¡errorFile¿* is a file where all predictions errors are written to.

X-fold cross-validation the given data is split into X data sets. X rounds of evaluation are run, in each round X-1 of these data sets are used for training and the remaining one is used for evaluation. Finally, the results of each round are averaged. (splits are made the same way as in 90-10 mode)

x-validation:

```
./runJSBDpackaged.sh x

-> usage: JSBD x <textDir> <cross-val-rounds> <errorFile>
```

¡textDir¿ and *¡errorFile¿* are the same as in 90-10 split mode. *¡cross-val-round¿* is the number of rounds for cross-validation. Typically, this might be set to 10.

2.3.3 Training

To train a sentence splitter model, you need to provide some training material (format: see above). The trained model can then be saved to disk and used for sentence splitting.

```
./runJSBDpackaged.sh t

-> usage: JSBD t <trainDir> <modelFilename>
```

¡trainDir¿ is the directory with training data. All files are considered and should be in the according format (see above). *¡modelFilename¿* is the file where the resulting model is saved.

2.3.4 Prediction

To employ the sentence splitter, you need a trained model. You can get a sentence splitting (trained on our manually compiled training material for the bio-medical domain (language: english)) from our website.

```
./runJSBDpackaged.sh p

-> JSBD.sh p <inDir> <outDir> <modelFilename>
```

¡inDir¿ is a directory of text documents which should be sentence splitted. *¡modelFilename¿* is the file where a previously model was saved to. The processed texts with one sentence per line are written to *¡outDir¿*

2.4 Background/Algorithms

JSBD is based on Conditional Random Fields (CRFs) [LMP01], a sequential learning algorithm.

2.5 Evaluation Studies and Available Models

JSBD was developed and optimized for the bio-medical domain. However, when training it on respective corpora, it may also be used for other domains.

We have evaluated JSBD on our data which we compiled for the bio-medical domain. It consists of data from both the GENIA [?] and the PennBioIE[7] corpus and additional material which we took from MedLine abstracts.

Currently, it comprises about 62000 sentences. An accuracy of 99.8% is yielded on this data using 10-fold cross-validation. You will find the model trained on this data in the directory `resources`.

If you run any evaluations on other data, we would be happy to learn about your experiences and evaluation results with JSBD.

3 Copyright and License

References

[LMP01] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers.

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

[7] `http://bioie.ldc.upenn.edu/`

[OTK02] Tomoko Ohta, Yuka Tateisi, and Jin-Dong Kim. The Genia corpus: An annotated research abstract corpu s in molecular biology domain. In M. Marcus, editor, *HLT 2002 – Human Language Technology Conference. Proceedings of the 2nd International Conference on Human Language Technology Research*, pages 82–86. San Diego, Cal., USA, March 24-27, 2002. San Francisco, CA: Mo rgan Kaufmann, 2002.

[TWH07] Katrin Tomanek, Joachim Wermter, and Udo Hahn. A reappraisal of sentence and token splitting for life science documents. In *MEDINFO 2007 – Proceedings of the 12th World Congress on Me dical Informatics.*, 2007.