Documentation for

JULIE Lab Token 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`

1 UIMA Wrapper

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

UIMA-JTBD is an UIMA wrapper for JTBD, the respective command-line version. For more detailed information on the functioning of JTBD check the JTBD documentation or refer to [?].

1.1 Installation

UIMA-JTBD 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 `TokenAnnotator.xml` for UIMA-JTBD. You may now e.g. run UIMA's Collection Proeccessing Engine Configurator (`cpeGUI.sh`) and add UIMA-JTBD as a component into your NLP pipeline.

This pear package also contains a model for tokenization splitting. The model was trained on a special bio-medical corpus which consists of data from (manually annotated) material which we took from MedLine abstracts and a modified version of PennBioIE's[2] underlying tokenization. In the PennBioIE corpus, some purely alphanumeric strings

[1] `http://www.julielab.de/`
[2] `http://bioie.ldc.upenn.edu/`

are divided into smaller tokens to support PennBioIE's entity annotation, especially in a common annotation for variation events (e.g. "S45F" with "S"=state_original, "45"=location, "F"=state_altered). Those splits were (manually) undone to fit our tokenization guidelines. Currently, our tokenization corpus comprises about 36000 sentences. An accuracy of ACC=96.7% is reached 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-JTBD 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]

UIMA-JTBD employs the machine learning toolkit MALLET [**?**].

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/TokenAnnotator.xml`).

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

[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.

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

Subsection	Key	Value
Implementation Details	Implementation Language	Java
	Engine Type	primitive
Runtime Information	updates the CAS	yes
	multiple deployment allowed	yes
	outputs new CASes	no
	Name of the Java class file	`de.julielab.jules.ae.TokenAnnotator`
Overall Identification Information	Name	Token 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.

Parameters See Table **??** 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 to model trained for JTBD

Table 2: Parameters of this AE.

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

Parameter Name	Parameter Syntax	Example
ModelFilename	full path	`resources/JULIE_life-science-1.6.mod.gz`

Table 3: Parameter settings of this AE.

Type System On this page, go to *Imported Type* and add the *JULIE UIMA Type System*. (Use “Import by Location”).

Capabilities The tokenizer takes as input annotations from type `de.julielab.jules.types.Sentence` and returns annotations from type `de.julielab.jules.types.Token`. See Table **??**.

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

Table 4: Capabilities of this AE.

Index Nothing needs to be done here.

Resources Nothing needs to be done here.

2 JTBD: Core Functionality and Stand Alone Tool

JULIE Token Boundary Detector (JTBD) is a tokenizer developed and optimized for the bio-medical domain. In contrast to many other tokenizers which consists of simple patterns, JTBD is based on machine learning (see Section **??**) which enables it to handle also tricky cases occuring frequently in life science documents. See [**?**] for a more in depth description of JTDB and a performance study.

JTBD 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 JTBD was only tested with Java 1.5; you need at least the Java 1.5 runtime environment installed on your system to run JTBD. In addition to the common Java libraries, JTBD employs MALLET [**?**], 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, two files are needed, called <sent-file> and <tok-file> later on. <sent-file> contains sentence information (one sentence per line) and <tok-file> contains the tokenization information (also here one sentence per line and tokenization made by adding white spaces). See the files in `testdata` for examples (`train.*`).
- For tokenization, the input files have to be sentence splitted, i.e. there has to be exactly one sentence per line. See the files in `testdata` for examples (`split.txt`).

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

2.3 Using JTBD

To execute JTBD just type

```
./runJTBDpackaged.sh
```

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

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

Available modes:
c: check data
s: 90-10 split evaluation
x: cross validation
t: train a tokenizer
p: predict with tokenizer
e: evaluation on previously trained model
```

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

2.4 Data Check

The provided data is checked for the correct format.

```
./runJTBDpackaged.sh c

-> usage: JTBD c <sent-file> <tok-file>
```

<sent-file> is a sentence splitted document (one sentence in each line); *<tok-file>* is a sentence splitted and tokenized document (one sentence in each line and a white space after each single token).

2.5 Evaluation

There are two evaluation modes to evaluate the tokenizer 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)

```
./runJTBDpackaged.sh s

-> usage: usage: JTBD s <sent-file> <tok-file> <predout-file> <errout-file>
```

<sent-file> is a sentence splitted document (one sentence in each line); *<tok-file>* is a sentence splitted and tokenized document (one sentence in each line and a white space after each single token); *predout-file* is a file where all the predictions made during this evaluation are written to. *<errorFile>* is a file where all predictions errors are written to. The Format of this file is as follows: @P->N indicates a false negative error, @N->P indicates a false positive error, then the prediction and the original tokenization are shown (the problematic token itself and some neighbouring tokens) and finally the complete sentence where the error occured (predicted and original version).

X-fold cross-validation the given data is split into X data set. 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)

```
./runJTBDpackaged.sh x

-> usage: usage: JTBD x <sent-file> <tok-file>
                        <cross-val-rounds> <predout-file> <errout-file>
```

<sentFile>, *<tokFile>*, *<predout-file>* and *<errorout-file>* 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.

2.6 Training

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

```
./runJTBDpackaged.sh t

-> usage: JTBD t <sent-file> <tok-file> <model-file>
```

<sent-file> and *<tok-file>* is the sentence splitted and tokenized training data. *<model-file>* is the file where the resulting model is saved to.

2.7 Prediction

To employ the tokenizer, you need a trained model. Directory `resources` contains a pre-trained model for life-science documents.

```
./runJTBDpackaged.sh p

-> JTBD p JTBD p <inDir> <outDir> <model-file>
```

<inDir> is a directory of text documents which should be tokenized. *<model-file>* is the file where a previously trained model was saved to. The processed texts with one sentence per line and white spaces between the single tokens are written to *<outDir>*

3 Background/Algorithms

JTBD is based on Conditional Random Fields (CRFs) [**?**], a sequential learning algorithm.

4 Evaluation Studies and Available Models

JTBD 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 JTBD on our tokenization corpus which we compiled for the bio-medical domain. It consists of data from (manually annotated) material which we took from MedLine abstracts and a modified version of PennBioIE's[7] underlying tokenization. In the PennBioIE corpus, some purely alphanumeric strings are divided into smaller tokens to support PennBioIE's entity annotation, especially in a common annotation for variation events (e.g. "S45F" with "S"=state_original, "45"=location, "F"=state_altered). Those splits were (manually) undone to fit our tokenization guidelines.

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

Currently, our tokenization corpus comprises about 36000 sentences. An accuracy of ACC=96.7% is reached 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 with JTBD and evaluation results.

5 Copyright and License