# Manual for JVnTextPro

Version: 2.0

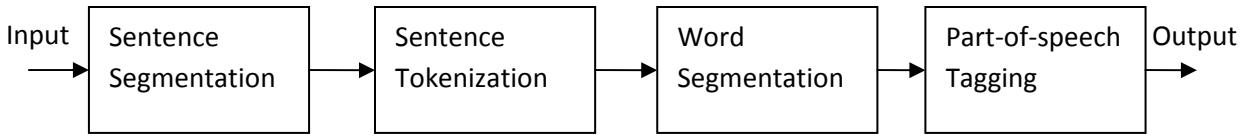
Developers:


| | |
|---|---|
| Cam-Tu Nguyen | ncamtu@gmail.com |
| Xuan-Hieu Phan | hieupx@gmail.com |
| Thu-Trang Nguyen | trangnt84@gmail.com |

# Contents

# 1 Introduction

JVnTextPro is a Java open source tool, which is based on Conditional Random Fields (CRFs) [1, 2] and Maximum Entropy (Maxent) [3], for Natural Language Processing (NLP) in Vietnamese. This tool consists of several steps (or sub-problem tools) for Vietnamese preprocessing and processing designed in a pipeline manner in which output of one step is used for the next step.




It depends on users to process their data in the whole pipeline or just call some sub-problem steps among these steps. There are two methods to call only some of those steps:

- Create a JVnTextPro object and specify the processing option (initialize sub-problem tools that are necessary only).
- Create and call directly the sub-problem tools (JVnSegmenter, …).

Note that the input for each sub-problem tool is required in correct format to make sure it works properly. JVnTextPro was built on Java (version 6.0). We need Java Runtime Environment to work with the tool.

# 2 JVnTextPro

## 2.1 How to call JVnTexPro from command line

**Command Line**

```
Java –mx1024M –cp [classpath] jvntextpro.JVnTextProTest –modeldir [modelDir]
        [options…] –input [infile/indirectory] (-filetype (filetype))
```

Suppose that we are in the outermost folder of JVnTextPro, we can set the above parameters as follows:

[classpath]=bin:lib/lbfgs.jar:lib/args4j.jar (or bin;lib\lbfgs.jar;lib\args4j.jar in Windows)

[modelDir] = path to the model containing models of sub-problem tools of JVnTextPro

[options…] = initialize one or more tools (-senseg, -wordseg, -sentoken, -postag) to process text in Vietnamese. Users should ensure the input data is in the right format for each tool to work properly. For example, if we want to perform part-of-speech tagging, the input should have words segmented. If option –input is set to a directory, all files ended with .txt in the specified

directory will be considered as input files unless we explicitly set to other file types by using option –filetype (optional).

The output of JVnTextProTest is ".pro" file if –input is a file, a set of ".pro" files if –input is a directory.

Example:

```
java -mx1024M -cp bin;libs\lbfgs.jar;libs\args4j.jar jvntextpro.JVnTextProTest
-modeldir models -senseg -sentoken -wordseg -postag -input samples\input1.txt
```

If the input were performed sentence segmentation and tokenization, you may want to perform word segmentation and POS tagging only. The command to do it as follows (we skip the options –senseg and –sentoken):

```
java -mx1024M -cp bin;libs\lbfgs.jar;libs\args4j.jar jvntextpro.JVnTextProTest
-modeldir models -wordseg -postag -input samples\input1.txt.sent.tkn
```

## 2.2 JVnTextPro API

You are able to integrate `JVnTextPro` for your own purpose via our API. Basically, JVnTextPro accepts Vietnamese texts in Unicode or Composite Unicode.

**Step 1**: Create a `JVnTextPro` and initialize processing sub-tools

```
JVnTextPro textPro = new JVnTextPro();
textPro.initSenSegmenter([modelDir of Sentence segmentation]);
textPro.initSenTokenizer();
textPro.initSegmenter([modeldir to Word Segmentation model folder]);
textPro.initPosTagger([modeldir to part-of-speech tagging model folder]);
```

Note that the initializations are the most time-consuming step, hence we should initialize one JVnTextPro once and use it for later calls.

**Step 2**: Call `process()` method to process input text in pipeline manner of initialized tools or call individual methods to perform text processing with different sub-problem tools.

```
textPro.process([input String]);
```

**or**

```
textPro.senSegment([input String]);
textPro.sentToken([input String]);
textPro.wordSegment([input String]);
textPro.posTagging([input String]);
```

## 2.3 JVnTextPro Service

We provided a service interface for systems in different programming languages to connect to and benefits from our Vietnamese text processing.

**Server side**: run the service using

```
java –mx1024M –cp [classpath] jvntextpro.service.TaggingService –moderdir
[modeldir] [options…]
```

where `classpath`, `modeldir`, and `options` can be set the same as in Section 3. You may enable one or more processing tools of JVnTextPro as described in the previous section. It is also likely to have several versions of TaggingService in different servers and provide different services (one server for word segmentation, one server for POS tagging, etc ..)
After running this command line, the service will listen at the port 2929. Data stream sent to this service need to be in UTF-8 format and ended with '0'.

**Client side:**

- Create a `TaggingClient` object, open the connection

```
TaggingClient client  = new TaggingClient([server], 2929); //server address
and port
client.connect();
```

- Perform text processing

```
client.process(input);
```

- Close connection when necessary

```
client.close();
```

# 3 Sentence Segmentation

It is the basic preprocessing functionality of `JVnTextPro`. It segments a paragraph of text into sentences. Note that some time, the dot notation '.' Is not necessary indicate the end of the sentence. The problem of sentence segmentation is mostly dealing with the ambiguity of end of sentence notations.
By default, before a paragraph of text is segmented into sentences, we will convert the text from Composite Unicode to Unicode (if necessary). By doing so, our system is able to work with Composite Unicode.

**Command Line**: Suppose that you are at the outermost folder of JVnTextPro, you can perform sentence segmentation for file in the sample directory using the following command line.

```
java -cp libs\args4j.jar;libs\lbfgs.jar;bin jvnsensegmenter.JVnSenSegmenter
-modeldir models\jvnsensegmenter -inputfile samples\input1.txt
```

We also can replace option "-inputfile samples\input1.txt" by "-inputdir samples". In this case, the tool will take all the ".txt" files as input files. The output of sentence segmentation is one (if –inputfile is used) or a set (if –inputdir is used) of ".sent" files.

# 4 Sentence Tokenization

In normal writing style, most of the marks in sentences are attached to previous words. The Sentence Tokenization is to make them apart, and make the input cleaner for later processing.

**Input & Output**: the sentence "Even that it rains, we still go to school" → "Even that it rains , we still go to school". Note the white space before the comma in the sentence.

**Command Line**: Suppose that you are at the outermost folder of JVnTextPro, you can perform sentence tokenization for file in the sample directory using the following command line.

```
java -cp bin;libs\args4j.jar;libs\lbfgs.jar jvntokenizer.JVnTokenizer
-inputfile samples\input1.txt.sent
```

We also can replace option "-inputfile samples\input1.txt.sent" by "-inputdir samples". In this case, the tool will take all the ".sent" files as input files. The output of sentence segmentation is one (if –inputfile is used) or a set (if –inputdir is used) of ".tkn" files.

# 5 Word Segmentation

JVnSegmenter is a Java-based and open-source Vietnamese word segmentation tool. The segmentation model in this tool was trained on about 8,000 labeled Vietnamese text sentences using conditional random fields (FlexCRFs). Refer to our paper at PACLIC 2006 for more information. This tool would be useful for Vietnamese NLP community. We highly appreciate any bug report, comment, and suggestion that help to fix errors and improve the segmentation accuracy.

The previous version of word segmentation is available at http://jvnsegmenter.sourceforge.net/.

**Input & Output format**:

Input: Học sinh học sinh học →Output: Học_sinh học sinh_học

**Command Line:**

```
java -mx512M -cp bin;libs\args4j.jar;libs\lbfgs.jar jvnsegmenter.WordSegmenting
-modeldir models\jvnsegmenter -inputfile samples\input1.txt.sent.tkn
```

We also can replace option "-inputfile samples\input1.txt" by "-inputdir samples". In this case, the tool will take all the ".tkn" files as input files. The output of sentence segmentation is one (if –inputfile is used) or a set (if –inputdir is used) of ".wseg" files.

# 6 Part of Speech Tagging (POS Tagging)

JVnTagger is a tool for Vietnamese Part-of-Speech tagging based on Conditional Random Fields [1, 3] (CRFs) and Maximum Entropy [2]. JVnTagger was built as a part of the national project "Building Basic Resources and Tools for Vietnamese Language and Speech Processing" (VLSP) during 2 years (from 2007 to 2009). The training datasets consist of 10.000 and 20.000 sentences from Vietnamese TreeBank, which are subsequently referred to as VTB-10,000 and VTB-20,000. 5-fold-cross validation with CRFs on VTB-10,000 shows that we are able to achieve the result of 93.45% of F1-measure. Experiments with Maxent on VTB-20,000 using 10-fold-cross validation show the best result of 93.32% of F1-measure.

**Note:** The Tagger integrated into JVnTextPro pipeline is MaxentTagger.

**Input and Output Formats:**

**Input:** Text with words segmented. For example: Học_sinh học sinh_học

**Output:** Text with words annotated with POS tags. For example: Học_sinh/N học/V sinh_học/N ./.

**Command line**

```
Java –mx512M –cp [classpath] jtextpro.POSTagging –tagger [tagger]
      –modeldir [modeldir] –inputfile/-inputdir [inputfile/inputdir]
```

Suppose that we are in the outer folder of JVnTagger, we can set the above parameters as follows:

[classpath] = bin:lib/lbfgs.jar (or bin;lib\lbfgs.jar in Windows)

[tagger] = crfs or maxent

[modeldir] = directory containing the model of crfs (maxent) if tagger is set to crfs (or maxent)

[inputfile/inputdir]=path to file (directory) containing data to be processed.

**For example:**

```
java -mx512M -cp bin;libs\args4j.jar;libs\lbfgs.jar jvnpostag.POSTagging
-tagger maxent -modeldir models\jvnpostag\maxent -inputfile
samples\input1.txt.sent.tkn.wseg
```

We also can replace option "-inputfile samples\input1.txt" by "-inputdir samples". In this case, the tool will take all the ".wseg" files as input files. The output of sentence segmentation is one (if –inputfile is used) or a set (if –inputdir is used) of ".pos" files.

**Tags and Explaination**

| | |
|---|---|
| 1. N: Noun (danh từ)<br>2. Np: Personal Noun (danh từ riêng) | 12. C: conjunction (liên từ)<br>13. I: Interjection (thán từ) |

| | |
|---|---|
| 3. Nc: Classification Noun (danh từ chỉ loại)<br>4. Nu: Unit Noun (danh từ đơn vị)<br>5. V: verb (động từ)<br>6. A: Adjective (tính từ)<br>7. P: Pronoun (đại từ)<br>8: L: attribute (định từ)<br>9. M: Numeral (số từ)<br>10. R: Adjunct (phụ từ)<br>11. E: Preposition (giới từ) | 14. T: Particle, modal particle (trợ từ, tiểu từ)<br>15. B: Words from foreign countries (Từ mượn tiếng nước ngoài ví dụ Internet, ...)<br>16. Y: abbreviation (từ viết tắt)<br>17. X: un-known (các từ không phân loại được)<br>18. Mrk: punctuations (các dấu câu) |

# 7 How to Develop New Tagging Tools into JVnTextPro

## 7.1 Package jvntextpro.data

| Classes | Explanations |
|---|---|
| `TWord` | `TWord` is short for tagged words in which the main variable members are the main token to be tagged and tag.<br>For example, in `jvnsegmenter`, main token is syllable but in `jvntagger`, main token is word. Tag in `jvnsegmenter` is B_W, I_W, …<br>We also provide an auxiliary data structure that is `secondarytags` to store secondary tags, in which important information to predict main tags is stored. For example, in noun phase chunking, POS tag (secondary tags) is important beside words in sentences (which are token) |
| Sentence | Sentence is a list of TWord objects. |
| `DataReader` | Extend this class to read in data in different formats into an array of Sentence. Implementations of this class are `WordDataReader` and `POSDataReader` of jvnsegmenter and jvntagger.<br>If tags are available for input text (in the case of reading training data), they are assigned to `TWord` objects of Sentence. Otherwise, tags of `TWord` objects are set to **null**. |
| `DataWriter` | Extend this class to write an array of Sentence of `TWord` objects with annotated tags (generated by machines) into a specific output format. The two implementations of `DataWriter` are `WordDataWriter` and `POSDataWriter`, in which the output of `WordDataWriter` is text with words specified such as "Mãi_mãi tuổi 20" and output of `POSTagger` is "Mãi_mãi/R tuổi/N 20/M" |
| `ContextGenerator` | Take one position of some Sentence object as an input. The major function of `ContextGenerator` is to gather context surrounding that text. |
| `TaggingData` | Contains an array of `ContextGenerator` objects, performs gathering information according to added `ContextGenerator` objects for the whole Sentence. |

## 7.2 Develop a New Tagging Tool into JVnTextPro

Several steps to develop a new tagging tool, which is similar to `JVnSegmenter` and `JVnTagger`, are described in the following:

1. Define Tags, input format, output format and implement `DataReader` and `DataWriter` correspondingly.

   **For example:**

- In word segmentation, we defined 3 tags B_W (begin of word), I_W (inside of word), and O for syllables.
- Input format of word segmentation is normal text in which syllables are separated by white spaces, and sentences are optionally segmented. `WordReader` extended `DataReader` to read in an array of input `Sentence` of `TWord` (tags are B_W, I_W, O).
- Output format of word segmentation is text with syllables joined by "_" to form words. `WordWriter` extended `DataWriter` to write out array of `Sentence` of `TWord` in that format.

2. Prepare training data that is text in the defined input format and annotated with defined tags.

3. Implements one (or more) `ContextGenerator` classes.

- **For example:** in word segmentation, there are 4 classes extended `ContextGenerator`. All of them are added to a `TaggingData` to generate contexts for training, and input string for later labeling.

4. Extend `jvntextpro.data.TrainDataGenerating` to generate training data based on `ContextGenerator` objects and in the format of input files for `FlexCRFs`++ as described in http://flexcrfs.sourceforge.net/ (the format is the same for `FlexCRFs`++ and jmaxent). See `WordTrainGenerating` and `POSTrainGenerating` for references.

   For re-training word segmentation and pos tagging model, you can modify `featuretemplate` files and use `WordTrainGenerating`, and `POSTrainGenerating` to regenerate training data for retraining word segmentation and POS tagging models.

5. Train a new model with `FlexCRFs`++ and `jmaxent`

   For `FlexCRFs++`, see http://flexcrfs.sourceforge.net/ for more details about training with the training data generated from step 4 above.

   For `JMaxent`, use `jmaxent.Trainer` to train a model with Maximum Entropy for file generated from step 4 above.

6. Implement a new tagging class to label new input data.

- Implement a class similar to `jvntagger.CRFTagger` (if a model of CRFs was trained) or `jvntagger.MaxentTagger` (if a model of Maxent was trained)

7. Update `jvntextpro.JVnTextPro` and `jvntextpro.JVnTextProTest` to include the new tool.

- Add the class defined in 6 to `JVnTextPro.`
- Create an initialization methods like `initSenSegmenter` or `initSegmenter` in `JVnTextPro`
- Create a wrapping function to perform new tagging functionality with checking whether this functionality is enabled (initialized or not). For example, `wordsegment` method of `JVnTextPro` is one wrapping function for word segmentation.
- Add a call to the wrapping function in the method `process` of `JVnTextPro` at a right position of the pipeline chain. It is similar to putting word segmentation before Pos tagging to meet the input requirement of POS tagging.

# 8 Acknowledgements

We would like to thank Trung-Kien Nguyen for spending a lot of time to annotate the training data for Word segmentation. We would like to thank the research project VLSP and VietTreeBank group for providing us the training data for Pos tagging.

We would also like to thank Sourceforge.net for hosting this project.

This tool is under the terms of GNU. Any reference to this tool should be cited as:

*Cam-Tu Nguyen, Xuan-Hieu Phan, Thu-Trang Nguyen, JVnTextPro: a tool to process Vietnamese texts, version 2.0, 2010.*

# 9 References

[1] Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *Proc. 18th International Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA (2001) 282–289

[2] Kamal Nigam, John Lafferty, Andrew Mccallum. In IJCAI-99 Workshop on Machine Learning for Information Filtering (1999), pp. 61-67

[3] FlexCRFs: http://flexcrfs.sourceforge.net/