PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 98 OCTOBER 2012 75–85

TrTok: A Fast and Trainable Tokenizer for Natural Languages

Jiří Maršík[a], Ondřej Bojar[b]

[a] Shanghai Jiao Tong University
[b] Institute of Formal and Applied Linguistics, Charles University in Prague

Abstract

We present a universal data-driven tool for segmenting and tokenizing text. The presented tokenizer lets the user define where token and sentence boundaries should be considered. These instances are then judged by a classifier which is trained from provided tokenized data. The features passed to the classifier are also defined by the user making, e.g., the inclusion of abbreviation lists trivial. This level of customizability makes the tokenizer a versatile tool which we show is capable of sentence detection in English text as well as word segmentation in Chinese text. In the case of English sentence detection, the system outperforms previous methods. The software is available as an open-source project on GitHub[1].

1. Introduction

Researchers in statistical machine translation and other natural language processing fields make use of large corpora of text. However, not all of these corpora are immediately useful since not all of them are partitioned into words and sentences. This is in odds with the premise that words and sentences, not chunks of text, form the basic processing units of most NLP applications. This is where tokenization and segmentation have to step in.

Segmentation (a term we use for what is also referred to as sentence detection or sentence boundary disambiguation) has been tackled using a variety of techniques. The most common approaches include writing heuristics and constructing abbreviation lists (the Stanford Tokenizer, the RE system) or using machine learning algorithms to predict the role of a potential sentence terminator (Satz, MxTerminator,

[1] https://github.com/jirkamarsik/trainable-tokenizer

Corresponding author: bojar@ufal.mff.cuni.cz
Cite as: Jiří Maršík, Ondřej Bojar. TrTok: A Fast and Trainable Tokenizer for Natural Languages. The Prague Bulletin of Mathematical Linguistics No. 98, 2012, pp. 75–85. doi: 10.2478/v10108-012-0010-0.

Apache OpenNLP). There have also recently been some very successful systems using unsupervised methods (Punkt).

Tokenization is a problem which stops being trivial when we start considering whitespace-free languages such as Chinese or Japanese. In these languages, tokenization (also referred to as word segmentation) receives a lot of attention (Emerson, 2005).

TrTok aims to be a practical tool for tokenizing and segmenting text written in any language. To achieve such a goal, TrTok relies on the user determining the specifics of training and tokenization, and providing the necessary training data.

Continuing the approach outlined by Klyueva and Bojar (2008), TrTok's novelty comes in the openness and formalization of the tokenization process and in its resulting general applicability. The process is divided into several discrete stages, most of which are heavily customizable. For example, the user is able to say where in the text TrTok should consider breaking up or joining tokens or sentences, how TrTok should represent the context of these decision points to the underlying classifier, how the classifier should be trained, how existing whitespace should be treated and more.

TrTok was also built to be a practical tool, which means it can transparently process text interspersed with XML tags and HTML entities and it was designed to run fast.

The major inconveniences of TrTok are that, 1) due to its customizability it needs to be properly set up and, 2) due to its reliance on machine learning methods, it requires manually tokenized training data.

2. Previous Work

Established methods of sentence boundary disambiguation can be organized into three distinct groups: rule-based systems, supervised learning systems and unsupervised learning systems.

The **RE** system (Silla and Kaestner, 2004) is an example of a rule-based system. The program scans a document, looking for full stops. When one is found, the word preceding it is compared to a list of regular expression exceptions (mostly abbreviations) and unless the word is found to match one of them, it is assumed to end the sentence. Besides this core logic, the system also implements a small heuristic which checks for numbers preceding the full stop and the word following it.

MxTerminator (Reynar and Ratnaparkhi, 1997) is a supervised sentence boundary disambiguator using maximum entropy models to predict whether a potential sentence terminator does indeed signal the end of a sentence. The prefix and suffix of the word containing the potential sentence terminator and the words preceding and following it are analyzed and their features are passed to the classifier. The features consist of the tokens' type, their capitalization and their membership status on a list of abbreviations which are either hand-prepared or induced from data.

The biggest difference between TrTok and MxTerminator is that TrTok does not assume any particular selection of features and thus offers space for richer models

(e.g., by extending the width of the context or providing more complex features like part of speech tags).

An example of a system using more advanced features is the **Satz** system (Palmer and Hearst, 1997), which uses possible POS (part of speech) tags as features in the machine learning algorithm.

Unsupervised learning systems are the most distinct from TrTok amongst all the sentence boundary detection algorithms as they usually require no manual configuration nor any training data to function properly. A great example of an unsupervised sentence boundary disambiguator is the **Punkt** system (Kiss and Strunk, 2006).

Punkt relies mostly on collocation detection techniques but also makes use of an orthographic heuristic to analyze the test data in several passes and disambiguate abbreviations and sentence terminators. The system has shown remarkable performance without needing any manual tuning or training data.

3. System Description

TrTok is implemented by a parallel execution of several, configurable pipeline steps. This pipeline can be repurposed to train the embedded classifier using tokenized data, to tokenize new data using a trained classifier, and to evaluate the predictions of a trained classifier on manually tokenized data.

We will describe the important pipeline steps one by one, in the order in which they process data when tokenizing new text.

3.1. RoughTokenizer

The RoughTokenizer partitions the stream of input characters into small, discrete chunks of non-blank characters called *rough tokens*. The partitioning can be made more granular by user-defined rules which specify positions at which the desired tokenization might differ from the whitespace-induced one.

A location in the text may be marked as a `MAY_SPLIT` meaning that the characters in the text preceding and following it may be parts of different tokens even though they are not separated by whitespace (e.g. we might wish to put a `MAY_SPLIT` between *"was"* and *"n't"* in *"wasn't"*).

A location within a span of whitespace characters might be labeled as a `MAY_JOIN` signalling that the characters preceding and following the whitespace might be parts of the same token, as in the case of spaces entered in long numbers (e.g. *"12 345"*).

Finally, a location in the text may be marked as a `MAY_BREAK_SENTENCE` if the characters preceding and following it might belong to different sentences.

See Figure 1 for an example of how these potential tokenization operations can look like in a sentence. A rough token is defined as a maximal sequence of characters uninterrupted by whitespace nor by any symbol denoting a possible tokenization operation (the symbols underneath the sentence in Figure 1). For example, in the

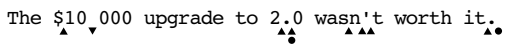

Figure 1. In this example, ▲ stands for MAY_SPLIT, ▼ for MAY_JOIN and • for MAY_BREAK_SENTENCE. This is how the rough tokenization might turn out given some reasonable settings for tokenizing English.

sentence from Figure 1, "was", "n", the apostrophe and "t" are all individual rough tokens. Note that the presence of a `MAY_*` event only signifies the possibility of a tokenization operation (splitting or joining of tokens or sentences). Whether a token split, token join or sentence break will occur is up to the Classifier.

The locations of these possible tokenization operations are determined by user-defined rules, each of which consists of a pair of regular expressions. The respective tokenization operation is signalled if the characters leading up to and following a position match the regular expressions in one of these rules.

If we look back at Figure 1, we might imagine more robust settings also placing a `MAY_BREAK_SENTENCE` after the apostrophe/single quote, while others might be more daring and not place a `MAY_BREAK_SENTENCE` after the point in "2.0", because it is followed by a non-blank character.

TrTok collects these rules and generates a Quex program implementing a fast FSM (Quex[2] is a fast and Unicode-friendly variation on the classic tools lex and flex for C++).

3.2. FeatureExtractor

The stream of rough tokens interleaved with potential tokenization operations output by the RoughTokenizer is processed using the FeatureExtractor. The FeatureExtractor annotates each rough token with a bit vector signifying which of the user-defined feature predicates hold for the rough token in question.

The features can be defined in two ways: either using a regular expression or a list of rough tokens. For a feature defined using a regular expression, a rough token is said to have that feature if and only if the regular expression matches the entire rough token. In the case of a feature defined using a list of rough tokens, a rough token is said to have that feature if and only if it is in the list.

This way it is easy to specify features which try to analyze the shape of rough tokens using regular expressions or to simply give a list of all interesting tokens (e.g. words of a certain part of speech or exceptions such as abbreviations).

[2] `http://quex.sourceforge.net`

3.3. Classifier

The Classifier is the other important element in the pipeline (besides the RoughTokenizer). Its job is to disambiguate the potential tokenization operations identified by the RoughTokenizer, i.e. it decides whether a `MAY_SPLIT` splits a word into two tokens, whether a `MAY_JOIN` truly joins two words into one token and whether a `MAY_BREAK_SENTENCE` ends a sentence. It does so by consulting a maximum entropy classifier for every location containing these potential tokenization operations.

The features passed to the classifier consist of the features of rough tokens in the context surrounding the potential tokenization operation and the presence of whitespace and potential tokenization operations in the context area. The user is free to select the size of the context area and which features from which rough tokens in the context area should be passed to the classifier.

Features can also be clustered together into conjunct features which provide a value for every combination of the constituent features' values (this lets the trainer estimate a different parameter for different combinations of the features' values, which is useful to model the joint influence of some features).

The classifier then marks each location with a potential tokenization operation as either a sentence boundary, token boundary or no boundary (meaning the location is inside a token). Using this classification, any potential tokenization operations are finally disambiguated.

The model used in the Classifier unit is a maximum entropy model trained using the Maximum Entropy Modeling Toolkit for Python and C++[3]. Training is performed via either the L-BFGS or GIS algorithm, depending on the user's choice. Other parameters of the learner, such as the number of iterations to spend on training, are controlled by the user as well.

3.4. OutputFormatter

The OutputFormatter is the point at which the stream of rough tokens is turned back into a character stream. This means that all the rough tokens are concatenated and whitespace is inserted between them depending on whether there originally was any whitespace between them and on the tokenization operations which are to be carried out in the space between them. Individual tokens end up being separated by a single space character and sentences are separated by line breaks.

4. Usage

TrTok is used as a command line application.

```
Example:
    trtok train en/satz-like/brown -l data/brown/train.fl -r "|raw|txt|"
```

[3] `http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html`

Its first argument is the mode of operation, which can be one of `train`, `tokenize`, `evaluate` or `prepare`. The `train` mode uses manually annotated files to train a model for the Classifier and save it, while the `evaluate` mode uses them to compare the tokenizer's predictions to the manual tokenization and outputs the comparisons. The `tokenize` mode takes the input files and segments them using the trained model. The `prepare` mode does the same but with a dummy model which performs every possible sentence and token break.

The second argument to TrTok is the tokenization scheme folder. The tokenization scheme folder contains a set of optional files which influence the behavior of the tokenizer. Files with the `.rep` and `.listp` extensions define new features in terms of regular expressions or lists of types, respectively. Files with the `.split`, `.join` and `.break` extensions contain pairs of regular expressions which define possible token splits, token joins and sentence breaks, respectively. The `features` file defines which features are to be used from which rough tokens relative to the possible tokenization operation. The `maxent.params` file contains values for tuning the performance of the training algorithm. The scheme folder also allows a few other configuration files for convenience. An important thing to note is that the scheme folders can be nested and that the inner schemes inherit all the files of the outer scheme, unless they provide their own files of the same name. This is useful in cases where e.g. some features or training data are applicable to all texts of a language but refinements exist for various domains or tokenization conventions.

The rest of the parameters are input files and various options for adjusting the behavior of the tokenizer.

TrTok requires CMake and Quex at runtime, while several multi-platform libraries are also required at compile time. Further details on the installation and use of TrTok can be found in the bundled documentation.

5. Evaluation

We evaluated our implementation of TrTok compared to a wide range of prominent implementations and approaches to sentence detection. The results are given in Table 1.

5.1. Dataset

The experiments were conducted on the Brown corpus (Francis and Kucera, 1982) as supplied through NLTK (Bird et al., 2009). A representative (covering each category of text proportionately) 20% of the corpus was used as the testing data. This number was chosen so that the testing data would be sure to contain at least 1,000 instances of a non-sentence-terminating full stop; the resulting test set ended up containing 1,481 such full stops. The rest of the data was made available for training to the supervised learning methods.

	Acc. ↓	Err. Rate	Prec.	Recall	F_1	Time
TrTok::Groomed	**98.86%**	**1.14%**	**99.12%**	99.57%	**99.34%**	5.10s
Stanford CoreNLP	98.83%	1.17%	98.78%	99.89%	99.33%	5.02s
TrTok::MxTerm-like	98.76%	1.24%	98.70%	99.89%	99.29%	1.10s
TrTok::Easy	98.70%	1.30%	98.61%	99.91%	99.26%	1.08s
Punkt	98.65%	1.35%	98.82%	99.63%	99.22%	3.13s
MxTerminator	98.27%	1.73%	98.30%	99.74%	99.01%	1.37s
Apache OpenNLP	98.20%	1.80%	98.20%	99.77%	98.97%	1.13s
Apache OpenNLP (ready)	97.71%	2.29%	98.62%	98.75%	98.68%	1.17s
RE	97.26%	2.74%	98.52%	98.32%	98.42%	16.93s
TrTok::Satz-like	96.50%	3.50%	97.91%	98.08%	97.99%	1.59s
TrTok::Baseline	91.84%	8.16%	91.67%	99.66%	95.50%	0.85s
Absolute Baseline	86.89%	13.11%	86.89%	**100.00%**	92.99%	**0.02s**

Table 1. The performance of the various sentence detectors on full stops from the Brown corpus testing data. The 1.15 MB of testing data consisted of 11,376 sentences and 232,893 tokens.

5.2. Performance Measurement

The performance of the evaluated systems was measured by their success (accuracy) in classifying instances of the full stop character. The text contains other sentence terminators such as the question mark and the exclamation mark, but they almost never serve as anything else but sentence terminators in the text. Other occasional sentence delimiters such as dashes, semicolons, colons and line breaks were ignored as well, since the other systems usually do not have a solution for them. This way, the comparison is fair. Furthermore, the full stop is the most common and ambiguous of the sentence delimiters, so it makes sense to focus on it.

Besides the systems' accuracy, we also measure the time spent for processing the whole testing data and we present the median of 11 runs to bring the implementation speed of the systems into consideration as well.

5.3. Sentence Detection Methods

Absolute Baseline simply marks every full stop as a sentence terminator.

Trtok::Baseline is the simplest tokenizer which can be written in TrTok. However, even the simplest TrTok configuration always uses the whitespace following the possible tokenization operation as a feature and thus it is able to perform better than the Absolute Baseline.

TrTok::Satz-like is a straightforward attempt at reconstructing the Satz system in TrTok. The POS-tagged training data was used to construct lexicons for each different part of speech tag (NLTK's method of simplifying tags was used to reduce the number

of different tags to overcome data sparsity). The POS tags for three tokens on either side of the full stop were used as the features.

TrTok::Satz-like's system of tags is not as refined as the original and it does not use its fallback regular expression heuristics and hence it does not perform as well as the original Satz system did (Palmer and Hearst, 1997).

The **RE** system, **MxTerminator** and **Punkt** were described in Section 2. For training, Punkt received the entire Brown corpus (training data and testing data) without any annotations while MxTerminator was trained using the training data.

Punkt achieves remarkable performance and stands as the strongest competitor to TrTok in the field of multilingual sentence detection. They are both accurate language-independent tools but TrTok's big shortcoming is its need for a corpus of manually tokenized data.

Apache OpenNLP contains a sentence detector based around a maximum entropy classifier. The implementation is nearly identical to the specification of MxTerminator with only minor deviations (such as signalling surrounding whitespace as features).

We performed experiments both with the ready-made model for English distributed via OpenNLP's website and with a model which was trained on our training data.

The **Stanford CoreNLP** sentence splitter works by applying its tokenizer to the input text which makes the distinction between a full stop as part of an abbreviation or an ordinal number as opposed to a full stop as a sentence terminator. Thus the task of sentence splitting is trivial after the tokenization has been performed. The tokenizer is a rule-based program implemented using a lexical analyzer generator, JFlex (similar to how TrTok uses Quex to implement the RoughTokenizer).

The Stanford Tokenizer's performance is excellent, especially considering it has not had the chance to train itself on the target corpus. However, the Stanford Tokenizer is written explicitly for English and it is likely that its performance would not carry over to other languages without significant work.

TrTok::MxTerm-like is a reconstruction of MxTerminator in TrTok. It is a nice demonstration of the ease with which new tokenization setups can be defined in TrTok. The entire setup consisted of creating a directory, collecting the abbreviations in a single file and writing five lines of configuration, two or three of which could be easily obsoleted by adopting saner defaults in TrTok and one of which is purely for convenience.

The reason why MxTerminator does not achieve the same performance could be that the maximum entropy trainer used in MxTerminator limits itself to 100 iterations of Generalized Iterative Scaling, which converges very slowly compared to L-BFGS (Malouf, 2002). Another reason might be the fact that both MxTerminator and OpenNLP cut off infrequent features.

The high accuracy of TrTok::MxTerm-like led us to try and see what happens when we simplify the tokenization setup even further, which led to **TrTok::Easy** which works the same way as TrTok::MxTerm-like, but which does not use any abbreviation lists, merely the token types surrounding the full stop. Therefore, TrTok::Easy

	True Words Recall	Test Words Precision	F-measure
Academia Sinica	0.933	0.919	0.926
City University	0.934	0.934	0.934
Peking University	0.923	0.933	0.928
Microsoft Research	0.951	0.952	0.951

Table 2. The scores assigned to our tokenizer by the official scoring script of the Second International Chinese Word Segmentation Bakeoff, sorted by dataset.

does not rely on any external linguistic knowledge and is fairly language universal, given that we have enough training data. The performance of TrTok::Easy also points out that the difference in performance between TrTok and MxTerminator/OpenNLP cannot be explained by the different abbreviation lists.

Finally, **TrTok::Groomed** is a large, hand-made tokenization setup ported from a previous version of the tokenizer. It considers 24 different potential sentence terminators, it includes seven distinct lists of abbreviations totalling 303 types (prefix and suffix titles, abbreviated names of months, etc.) and it implements features for detecting the case of tokens, for noticing numbers which happen to be in the range of years, or the days of the month, etc… These features are extracted from rough tokens within eight tokens distance from the full stop. The two closest tokens on either side of the full stop also contribute their token type as a feature.

Due to the large number of potential tokenization operations and user-defined features, TrTok::Groomed's speed lags significantly behind the other TrTok setups.

Interestingly, there is not much difference in the performance of TrTok::Groomed, TrTok::MxTerm-like and TrTok::Easy. This tells us that besides the token types in the close vicinity of the full stop, other features are not that important. This highlights another use for TrTok as a tool for the fast analysis of the importance of different contextual features for performing the task of sentence detection.

5.4. Chinese Word Segmentation

Since TrTok is a general program for splitting text into sequences (sentences) which are in turn composed of other sequences (tokens) based on user-defined features, TrTok can be used for more than just sentence detection. One other segmentation task we had hoped might be solvable using TrTok is Chinese word segmentation.

We ported the key features of one of the top contestants (which also happens to employ a maximum entropy classifier) (Low et al., 2005) in the 2005 Second International Chinese Word Segmentation Bakeoff into TrTok and evaluated its performance using the official evaluation scripts. The results achieved (see Table 2) are approximately a median of the scores reported for submissions to the Bakeoff.

6. Conclusion

We have presented and described a universal tool for segmenting and tokenizing textual data. We have applied the tool to detecting sentences in English text and identifying words in Chinese text. We have shown that in both cases, TrTok can offer performance which is competitive with previous approaches, more so in the case of English sentence detection. In our experiments, different setups of TrTok outperformed existing systems in either speed or accuracy, while some setups of TrTok outperformed nearly all competitors in both criteria at the same time.

Since TrTok lets us define a lot of its behavior using declarative rules and feature descriptions, it might be interesting to harness this ability to find out the effect of various contextual cues on the performance of a sentence detector.

On the software side of things, TrTok would also benefit from getting more user-friendly, which would entail providing a walkthrough of the setup process, distributing further example setups and trained models and offering an all-dependencies-included compiled package for easier deployment.

Bibliography

Bird, S., E. Klein, and E. Loper. *Natural language processing with Python*. O'Reilly Media, 2009.

Emerson, T. The second international chinese word segmentation bakeoff. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, volume 133. Jeju Island, Korea, 2005.

Francis, W. and H. Kucera. Frequency analysis of english usage. 1982.

Kiss, T. and J. Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

Klyueva, N. and O. Bojar. Umc 0.1: Czech-russian-english multilingual corpus. In *Proc. of International Conference Corpus Linguistics*, pages 188–195, 2008.

Low, J.K., H.T. Ng, and W. Guo. A maximum entropy approach to chinese word segmentation. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, volume 1612164. Jeju Island, Korea, 2005.

Malouf, R. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

Palmer, D.D. and M.A. Hearst. Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics*, 23(2):241–267, 1997.

Reynar, J.C. and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19. Association for Computational Linguistics, 1997.

Silla, C.N. and C.A.A. Kaestner. An analysis of sentence boundary detection systems for english and portuguese documents. *Lecture notes in computer science*, pages 135–141, 2004.

Address for correspondence:
Ondřej Bojar
bojar@ufal.mff.cuni.cz
Institute of Formal and Applied Linguistics
Charles University in Prague
Malostranské náměstí 25
118 00 Praha 1, Czech Republic