Automatic Open Domain Information Extraction from Indonesian Text

Yohanes Gultom
Faculty of Computer Science
University of Indonesia
Email: yohanes.gultom@ui.ac.id

Wahyu Catur Wibowo
Faculty of Computer Science
University of Indonesia
Email: wibowo@cs.ui.ac.id

***Abstract*—Availability of big amount digital documents calls for an automatic method to extract information from any text document regardless of domain. Unfortunately, existing open domain information extraction (open IE) systems are not suitable for low-resource language such as Indonesian. This paper introduces a system to extract relation triples from Indonesian text using rule-based triple candidates generator, rule-based token expander and machine-learning-based triple selector. Trained using our 2,344 triples dataset (166 positives & 2,183 negatives), a Random Forest triple selector model achieves cross-validation score of 0.58 F1 (0.62 precision and 0.58 recall).**

1. Introduction

Open domain information extraction (open IE) is a paradigm that facilitates domain-independent discovery of triple relations from text document [1]. It extracts relations from sentence in three-values tuples or triples format (x, r, y) where x and y called arguments and r is the relation [2]. In more linguistic term, the arguments are also referred as subject and object while relation are referred as predicate [3]. The example of this extraction is described in Figure 1.

As described in Table 1, unlike traditional information extraction (IE), open IE extracts domain-independent relations from sentence. While it retrieves relations in format of triples similar to knowledge extraction (KE), open IE doesn't follow whole Resource Data Format (RDF) specification[1] like KE [4] [5]. Although mapping to existing relation schema is required in real word task such as slot filling [3], ontology is not in the scope of open IE research. Open IE has also been reported to be useful for tasks such as question answering [6] and information retrieval [7].

Due to the nature of NLP tasks and heuristics used in open IE system, it is only applicable for a specific language [1]. So in order to extract open domain information from Indonesian text, a specific system has to be defined for this language. Furthermore, considering the scarcity of Indonesian NLP resources, the system need to effectively utilize them to achieve the objective. Through this paper, we propose a open IE system that addresses these issues.

1. https://www.w3.org/RDF/

TABLE 1. General comparison between traditional information, open domain information and knowledge extraction

	IE	Open IE	KE
Domain	Closed	Open	Open
Format	Depends on domain	Triples	RDF Triples
Ontology	Not available	Optional	Mandatory

Input

"Sembungan adalah sebuah desa yang terletak di kecamatan Kejajar, kabupaten Wonosobo, Jawa Tengah, Indonesia."

Output

1. (Sembungan, adalah, desa)
2. (Sembungan, terletak di, kecamatan Kejajar)

Figure 1. Example of expected input and output of open domain information extraction

We propose an open IE system that combine heuristics (rule-based) models and a supervised learning model to extract relation triples from Indonesian text. This approach only requires single manually annotated dataset which is required to train triple selector/classifier. Our objective is to define a baseline system for Indonesian open IE that may be encourage more research in the future.

In general, the contributions from this research are:

- Open domain information extraction system for Indonesian text
- Open-source implementation of the system in public repository[2]
- Dataset of manually tagged triple candidates
- Reusable Indonesian NLP pipelines (lemmatizer, part of speech tagger, named-entity recognizer and dependency parser) built by extending Stanford

2. https://github.com/yohanesgultom/id-openie

CoreNLP[3] API

Further in this paper we will described some of the pre-eminent related works in open IE, the details about proposed system, experiments using some supervised-learning models as triples selector, analysis of the experiments results, and finally, conclusions and future works of this research.

2. Related Work

There has been plenty of works done in the open IE research. Starting from the introduction of open IE along with its first fully-implemented system, TextRunner, which further succeeded by systems built on top of it: ReVerb, R2A2 and Ollie (all from the same research group). The most recent research introduces Stanford OpenIE[4] which is an implementation of novel open IE system that outperforms Ollie in TAC-KBP 2013 Slot Filling task [3].

TextRunner was designed for massive size of open-domain web documents by avoiding heavy linguistic tasks and used inverted index to store extraction result [1]. It generates its own dataset (self-supervised) by using part of speech and dependency features and train a naive bayes classifier to select the triples. It argues that heavy linguistic tasks such as dependency parsing are not scalable to handle million of web documents. Additionally, it also uses redundancy assessor to remove redundant words (stop words, adverbs .etc).

ReVerb is an immediate successor of TextRunner which solves two significant problems in its predecessor: incoherent extractions and uninformative extractions [6]. It is composed of two algorithm: (1) Relation Extraction that extracts relations using syntactical and lexical constraint to solve the problems, and (2) Argument Extraction which retrieve the noun phrases as arguments of the relation. ReVerb takes as input a POS-tagged and NP-chunked and returns a set of relation triples.

R2A2 is a system built to fix argument extraction problem in ReVerb [2]. Instead of using heuristics to extract the arguments, it uses a learning-based system, ArgLearner, that accepts relation and sentence as inputs and returns the first (Arg1) and second arguments (Arg2). ArgLearner extracts the arguments using three classifiers based on REPTree and sequence labeling CRF as described in Figure 2.

Furthermore, Ollie (Open Language Learning for Information Extraction) [8] utilizes ReVerb to learn open pattern templates to guide triples extraction from sentence. Additionally, Ollie does a context analysis to extend the tuples with contextual information in order to improve precision. Its training and extraction architecture is describe in Figure 3.

One of the most research proposes new open IE system that replaces the usage of large open patterns in Ollie with a set of fewer patterns for canonically structured sentences and a classifier that learns to extract self-contained clauses from a sentence [3]. This system is implemented in Stanford OpenIE which is also integrated in the populer open source suites, Stanford Core NLP.

3. https://stanfordnlp.github.io/CoreNLP
4. https://nlp.stanford.edu/software/openie.html

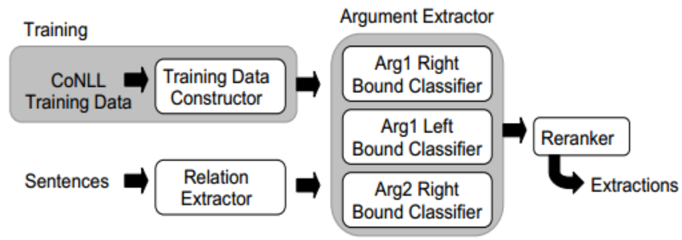

Figure 2. ArgLearner architecture training and extraction architecture

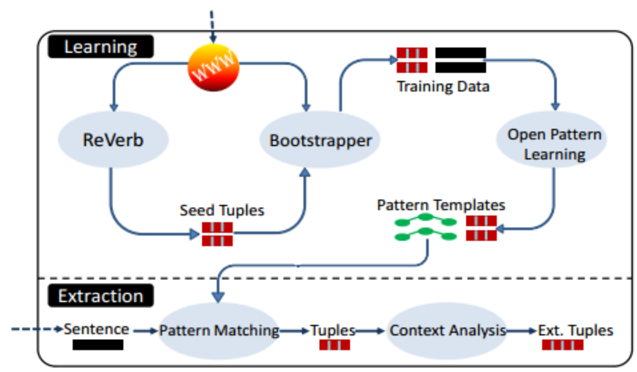

Figure 3. Ollie labeling and extraction architecture

3. Proposed System

Our proposed system also follows the pattern of three-steps [2] method used by open IE system:

1) **Label**: sentence are labeled to create a training dataset for the classifier. Although most of the related systems choose to do it automatically (using heuristics or distant supervision) [1] [2] [**?**]schmitz2012open, we choose to follow the method in recent research [3] to manually label our training data to ensure the quality.
2) **Learn**: train a classifier using the dataset to extract dataset. We use an ensemble model, Random Forest [9], as a classifier since it achieves the best score in our experiment.
3) **Extract**: use the classifier to extract relations (predicate) and arguments (subjects & objects) as triples. In our case, we also do token expansion to expand the token into meaningful clause.

As shown in the flowchart Figure 4, our system has three main components:

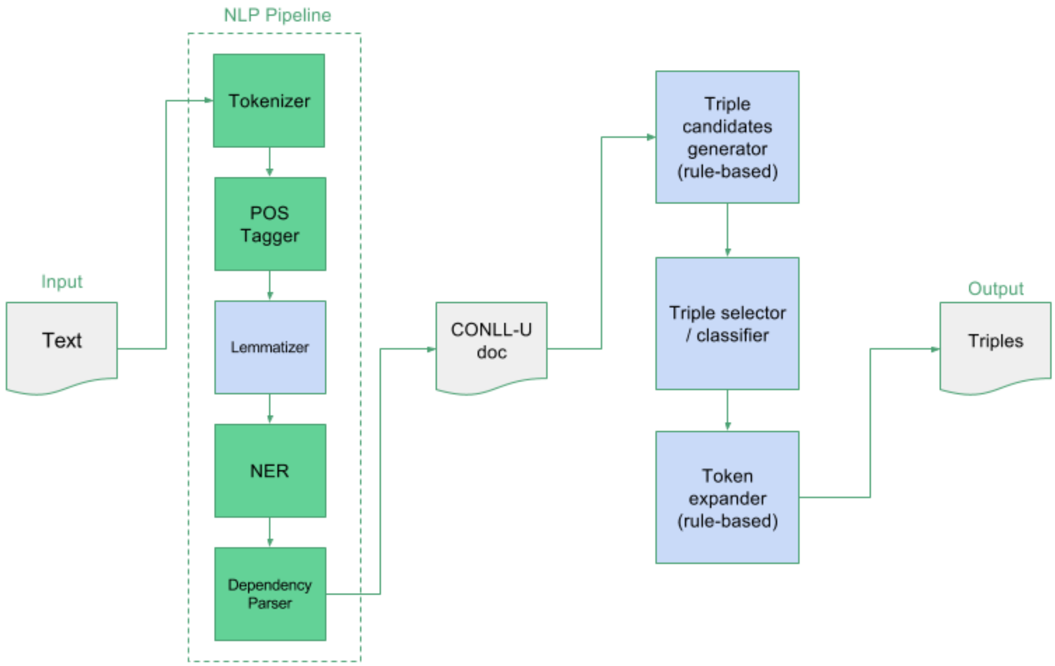

Figure 4. Indonesian open domain information extraction flowchart

3.1. NLP Pipeline

The NLP pipeline is a series of NLP tasks that annotates one or more sentences and saves them in CONLL-U[5] format, a token-based sentence annotation format containing lemma, POS tag, dependency relation and a slot for additional annotation. The model used in each of the NLP tasks in the pipeline are explained below:

1) **Tokenizer**
   We use default tokenizer provided by Stanford Core NLP, `PTBTokenizer` [10], which mimics Penn Treebank 3 tokenizer[6]. While this tokenizer provides many options to modify its behavior, we stick to default configuration that split sentence by whitelines to get the tokens.

2) **Part of Speech Tagger**
   We trained default Stanford Core NLP `MaxentTagger` [11] with Indonesian universal POS tag dataset which we convert from dependency parsing dataset[7]. This POS tagger uses Max Entropy (multi-class logistic regression) classifier which yields **93.68%** token accuracy and **63.91%** sentence accuracy when trained using 5,036 sentences and tested with 559 sentences from the dataset.

3) **Lemmatizer**
   The lemmatizer used in this pipeline, `IndonesianLemmaAnnotator`, is implemented based on an existing Indonesian rule-based Lemmatizer [12] with some improvements:

   - Reimplementation in Java language
   - Usage of in-memory database to speed up dictionary lookup
   - Integration with Stanford Core NLP annotator API for reusability

   This lemmatizer yields **99%** accuracy when tested using dataset of 5,638 token-lemma pairs[8].

4) **Named-Entity Recognizer**

   Stanford NLP `CRFClassifier` [13], a linear chain Conditional Random Field (CRF) sequence models, is trained using a dataset containing 3,535 Indonesian sentences with 5 entity class: Person, Organization, Location, Quantity and Time. When tested using 426 sentences, this models achieves 0.86 precision, 0.85 recall and **0.86** F1-score. The dataset itself is a combination between dataset from Faculty of Computer Science, University of

5. http://universaldependencies.org/format.html
6. https://catalog.ldc.upenn.edu/LDC99T42
7. https://github.com/UniversalDependencies/UD_Indonesian
8. https://github.com/davidchristiandy/lemmatizer

1	Sembungan	sembung	PROPN	_	_	4	nsubj	_	_
2	adalah	adalah	VERB	_	_	4	cop	_	_
3	sebuah	buah	DET	_	_	4	det	_	_
4	desa	desa	NOUN	_	_	0	root	_	_
5	yang	yang	PRON	_	_	6	nsubj:pass	_	_
6	terletak	letak	VERB	_	_	4	acl	_	_
7	di	di	ADP	_	_	8	case	_	_
8	kecamatan	camat	PROPN	_	_	6	obl	_	LOCATION
9	Kejajar	jajar	PROPN	_	_	8	flat	_	LOCATION
10	,	,	PUNCT	_	_	4	punct	_	_
11	kabupaten	kabupaten	NOUN	_	_	4	appos	_	_
12	Wonosobo	Wonosobo	PROPN	_	_	11	flat	_	LOCATION
13	,	,	PUNCT	_	_	11	punct	_	_
14	Jawa	Jawa	PROPN	_	_	11	appos	_	LOCATION
15	Tengah	tengah	PROPN	_	_	14	amod	_	LOCATION
16	,	,	PUNCT	_	_	11	punct	_	_
17	Indonesia	Indonesia	PROPN	_	_	11	appos	_	_
18	0	0	PUNCT	_	_	4	punct	_	_

Figure 5. CONLL-U Format Example

Indonesia and a public dataset[9].

5) **Dependency Parser**

We relied on Stanford NLP `nndep.DependencyParser` [14], to annotate dependency relation of each token in the the sentence. We train this transition-based neural network model using a Indonesian universal dependencies dataset of 5,036 sentences and 3,093 Indonesian word embeddings[10] (vector representation of words). Tested with 559 sentences, this model scores **70%** UAS (Unlabeled Attachment Score) and **46%** LAS (Labeled Attachment Score).

This pipeline is built by extending Stanford Core NLP classes and packaged as single Java program (JAR). Therefore it can be reused in any other system that require same kind of annotations.

3.2. Triple Candidates Generator

Triple candidates generator is used to extract relation triples candidates from CONLL-U document produced by NLP pipline. It uses a set of rules listed in Table 2 to extract relations (predicates) and arguments (subjects and predicates) from the sentence. The results of triples extraction are not always the positive or valid relation triples so, unlike TextRunner [1], we cannot use them directly as training data for triple selector/classifier.

For example, applying the rules to a annotated sentence in Figure 5 will generate these 17 triples candidates–where only five of them are valid triples:

- (Sembungan, adalah, desa) ✓
- (Sembungan, adalah, terletak)
- (Sembungan, adalah, kecamatan)
- (Sembungan, adalah, kabupaten)
- (Sembungan, adalah, Jawa)
- (Sembungan, adalah, Tengah)
- (Sembungan, adalah, Indonesia)
- (Sembungan, terletak, kecamatan) ✓
- (Sembungan, terletak, kabupaten) ✓
- (Sembungan, terletak, Jawa) ✓
- (Sembungan, terletak, Tengah)
- (Sembungan, terletak, Indonesia) ✓
- (desa, terletak, kecamatan)
- (desa, terletak, kabupaten)
- (desa, terletak, Jawa)
- (desa, terletak, Tengah)
- (desa, terletak, Indonesia)

TABLE 2. TRIPLE CANDIDATE GENERATION RULES

Type	**Condition**
Subject	Token's POS tag is either PROPN, NOUN, PRON or VERB
	Token is not "yang" nor "adalah"
	Token's dependency is neither "compound" nor "name"
	Token's dependency is either "compound" or "name" but separated by more than 2 tokens from its head
Predicate	Token's position is after Subject
	Token's POS tag is either VERB or AUX
Object	Token's position is after Subject and Predicate
	Token's POS tag is either PROPN, NOUN, PRON or VERB
	Token is not "yang" nor "adalah"
	Token's dependency is neither "compound" nor "name"
	Token's dependency is either "compound" or "name" but separated by more than 2 tokens from its head

In order to build a training data for the triple selector, we used triple candidates generator to generate 1,611 triple candidates from 42 sentences. From the candidates, we manually labeled 132 positive and 1,479 negative triples which we use to train binary classifier as triple selector in the training phase.

During the extraction phase, triple candidates generator is used in the system to extract unlabeled candidates from CONLL-U document. These unlabeled triples will be labeled by trained triple selector as described in (referring to flowchart in Figure 4.

3.3. Triple Selector

Triple selector is a machine learning classifier trained using manually labeled dataset of valid and invalid relation triples. For example, given the input of 17 candidates in Section 3.2, the selector will label the five check-marked triples as true and label the rest as false.

We use Random Forest [9], an ensemble methods that aggregate classification results from multiple decision trees,

9. https://github.com/yusufsyaifudin/indonesia-ner
10. https://github.com/yohanesgultom/id-openie/blob/master/data/parser-id.embed

TABLE 3. TRIPLE SELECTOR FEATURES

#	Features
1	Subject token's POS tag
2	Subject token's dependency relation
3	Subject token's head POS tag
4	Subject token's named entity
5	Subject token's distance from predicate
7	Subject token's dependency with predicate
8	Predicate token's POS tag
9	Predicate token's dependency relation
10	Predicate token's head POS tag
11	Predicate token's dependents count
12	Object token's POS tag
13	Object token's dependency relation
14	Object token's head POS tag
15	Object token's named entity
16	Object token's dependents count
17	Object token's distance from predicate
18	Object token's dependency with predicate

as the model for the classifier. We use the Scikit-Learn[11] implementation of Random Forest with following configuration:

- Decision tree criterion: Gini Impurity
- Minimum number of samples to split internal node: 5
- Maximum trees depth: 8
- Number of trees: 20
- Maximum features used in each tree: 4 (square root of the number of features)
- Class weight: balanced (multiplied by the ratio of training samples)

As for the features, we use 18 features described in Table 3 which are based on POS tag, named-entity and dependency relation, instead of shallow syntactic features used by TextRunner or ReVerb [1] [2]. We encode every nominal features and normalize the whole dataset by removing the mean and scaling to unit variance in order to improve the precision and recall of the classifier.

3.4. Token Expander

Instead of using lightweight noun phrase chunker [1], our system uses rule-based token expander to extract relation or argument clauses. It uses heuristics based on syntactical features (POS tag, dependency relation and named-entity) described in Table 4 and Table 5 to determine whether to expand a token to its dependent, ignore the dependent or even remove the token itself. For example, token expander will expand check-marked triples in Section 3.2 into:

11. http://scikit-learn.org

TABLE 4. TOKEN EXPANSION RULES FOR SUBJECT OR OBJECT TOKEN

#	Condition for Subject or Object Token	Action
1	If dependent's relation to the token is either compound, name or amod	Expand
2	If dependent has same named entity as the token	Expand
3	If dependent and the token are wrapped by quotes or double quotes	Expand
4	If the head is a sentence root	Ignore
5	If dependent's POS tag is CONJ or its form is either , (comma) or / (slash)	Ignore
6	If dependent's POS tag is either VERB or ADP	Ignore
7	If dependent has at least one dependent with ADP POS tag	Ignore
8	If the first or last token in expansion result has CONJ or ADP POS tag	Remove
9	If the first or last index of expansion result is an incomplete parentheses symbol	Remove
10	If the last index of expansion result is yang	Remove
11	Else	Ignore

TABLE 5. TOKEN EXPANSION RULES FOR PREDICATE TOKEN

#	Condition for Predicate Token	Action
1	If dependent is tidak	Expand
2	Else	Ignore

- (Sembungan, adalah, desa)
- (Sembungan, terletak di, kecamatan Kejajar)
- (Sembungan, terletak di, kabupaten Wonosobo)
- (Sembungan, terletak di, Jawa Tengah)
- (Sembungan, terletak di, Indonesia)

4. Experiments

In this research, we did two experiments. First, we compared the performance (precision & recall) of four classifiers in selecting valid triples from given candidates. Second, we used the best classifier and did end-to-end extraction triples from unlabeled document and observed the required time.

In the first experiment, we chose four classifiers each representing unique characteristics:

1) Logistic Regression [15] (linear-model)
2) Support Vector Machine [16] (non-linear model)
3) Multi Layer Perceptron [17] (neural network model)
4) Random Forest [18] (ensemble model)

5. Conclusion

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu,

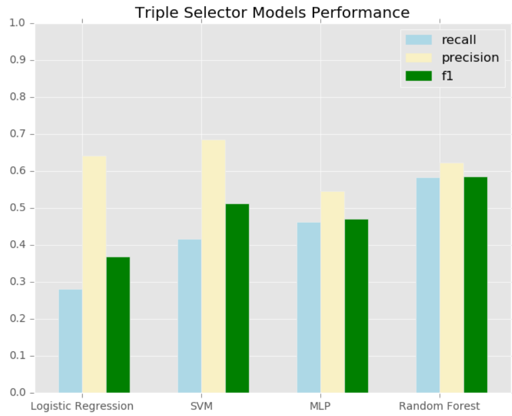

Figure 6. Triple selector models performance comparison chart

TABLE 6. TRIPLE SELECTOR MODELS PERFORMANCE

Models	**P**	**R**	$\mathbf{F_1}$
Logistic Regression	0.64	0.28	0.36
SVM	**0.68**	0.41	0.51
MLP	0.54	0.46	0.47
Random Forest	0.62	**0.58**	**0.58**

TABLE 7. SYSTEM END-TO-END EXTRACTION TIME

Sentences	**Triples Extracted**	**Total Time (s)**	**Time per Sentence (s)**
2	7	6.1	0.800
138	429	11.3	0.082
5,593	19,403	78.6	0.014

sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

References

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web." in *IJCAI*, vol. 7, 2007, pp. 2670–2676.

[2] O. Etzioni, A. Fader, J. Christensen, S. Soderland *et al.*, "Open information extraction: The second generation," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[3] G. Angeli, M. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL 2015)*, 2015.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," *The semantic web*, pp. 722–735, 2007.

[5] P. Exner and P. Nugues, "Refractive: An open source tool to extract knowledge from syntactic and semantic relations." in *LREC*, 2014, pp. 2584–2589.

[6] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1535–1545.

[7] O. Etzioni, "Search needs a shake-up," *Nature*, vol. 476, no. 7358, pp. 25–26, 2011.

[8] M. Schmitz, R. Bart, S. Soderland, O. Etzioni *et al.*, "Open language learning for information extraction," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 523–534.

[9] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] C. Manning, T. Grow, T. Grenager, J. Finkel, and J. Bauer, "Ptbtokenizer."

[11] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003, pp. 173–180.

[12] D. Suhartono, "Lemmatization technique in bahasa: Indonesian," *Journal of Software*, vol. 9, no. 5, p. 1203, 2014.

[13] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[14] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks." in *EMNLP*, 2014, pp. 740–750.

[15] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[16] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[17] G. E. Hinton, "Connectionist learning procedures," *Artificial intelligence*, vol. 40, no. 1-3, pp. 185–234, 1989.

[18] D. Wasserman, "Grid search optimization," 2015.