Automatic Open Domain Information Extraction  
from Indonesian Text

The vast amount of digital documents, that have surpassed human processing capability, calls for an automatic information extraction method from any text document regardless of their domain. Unfortunately, open domain information extraction (open IE) systems are language-specific and there is no published system for Indonesian language. This paper introduces a system to extract entity relations from Indonesian text in triple format using rule-based candidates generator, token expander and machine-learning-based triple selector. We cross-validate four candidates: logistic regression, SVM, MLP, Random Forest using our dataset to discover that Random Forest is the best classifier for the triple selector achieving 0.58 F1 score (0.62 precision and 0.58 recall). The low score is largely due to the simplistic candidate generation rules and the coverage of dataset.

# Introduction

Open domain information extraction (open IE) is a paradigm that facilitates domain-independent discovery of triple relations from text document(Banko et al. 2007). It extracts relations from sentence in three-values tuples or triples format where and called arguments and is the relation(Etzioni et al. 2011). In more linguistic term, the arguments are also referred as subject and object while relation are referred as predicate(Angeli, Premkumar, and Manning 2015). The example of this extraction is described in Figure [fig\_example\_io\_openie].

**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)

As described in Table [table\_paradigm\_comparison], unlike traditional information extraction (IE), open IE extracts domain-independent relations from sentence. While it retrieves relations in triples format similar to knowledge extraction (KE), open IE doesn’t follow whole Resource Data Format (RDF) specification[[1]](#footnote-22) like KE(Auer et al. 2007) (Exner and Nugues 2014). Although mapping to existing relation schema is required in real word task such as slot filling(Angeli, Premkumar, and Manning 2015), 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(Fader, Soderland, and Etzioni 2011) and information retrieval(Etzioni 2011).

l >p1.9cm >p1.9cm >p1.9cm **Aspect** & **IE** & **Open IE** & **KE**  
**Domain** & Closed & Open & Open  
**Format** & Depends on domain & Triples & RDF Triples  
**Ontology** & Not available & Optional & Mandatory

Due to the nature of NLP tasks and heuristics used in open IE system, it is only applicable for a specific language(Banko et al. 2007). 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.

We propose an open IE system that combine heuristics (rule-based) models and a supervised learning model to extract entity relations from Indonesian text in triple format. 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 encourage further 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
* 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 CoreNLP[[2]](#footnote-24) API

Further in this paper we will described some of the preeminent 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.

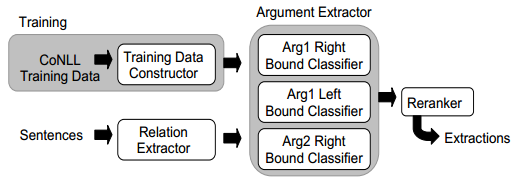
# Related Work

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(Banko et al. 2007), which further succeeded by systems built on top of it: ReVerb, R2A2(Etzioni et al. 2011) and Ollie(Schmitz et al. 2012) (all from the same research group). The most recent research introduces Stanford OpenIE[[3]](#footnote-27) which is an implementation of novel open IE system that outperforms Ollie in TAC-KBP 2013 Slot Filling task(Angeli, Premkumar, and Manning 2015).

**TextRunner** is designed for massive size of open-domain web documents by avoiding heavy linguistic tasks and used inverted index to store extraction result(Banko et al. 2007). It generates its own dataset (self-supervised) by using part of speech and dependency features to train a naive bayes classifier used 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).

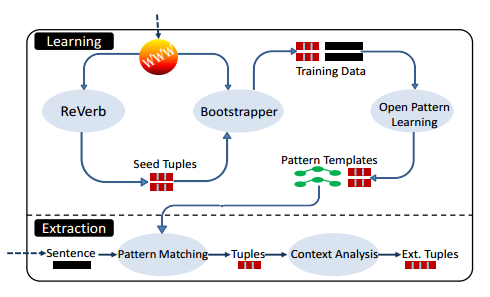
**ReVerb** is an immediate successor of TextRunner which solves two significant problems in its predecessor: incoherent extractions and uninformative extractions (Fader, Soderland, and Etzioni 2011). It is composed of two algorithms: (1) Relation Extraction that extracts relations using syntactical and lexical constraint to solve the problems, and (2) Argument Extraction which retrieves the noun phrases as arguments of the relation. ReVerb takes as input a POS-tagged and NP-chunked and returns a set of triples.

**R2A2** is a system built to fix argument extraction problem in ReVerb (Etzioni et al. 2011). 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 [fig\_arglearner\_architecture].



ArgLearner architecture training and extraction architecture

Furthermore, **Ollie** (Open Language Learning for Information Extraction)(Schmitz et al. 2012) utilizes ReVerb(Fader, Soderland, and Etzioni 2011) 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(Schmitz et al. 2012). Its training and extraction architecture is describe in Figure [fig\_ollie\_architecture].



Ollie labeling and extraction architecture

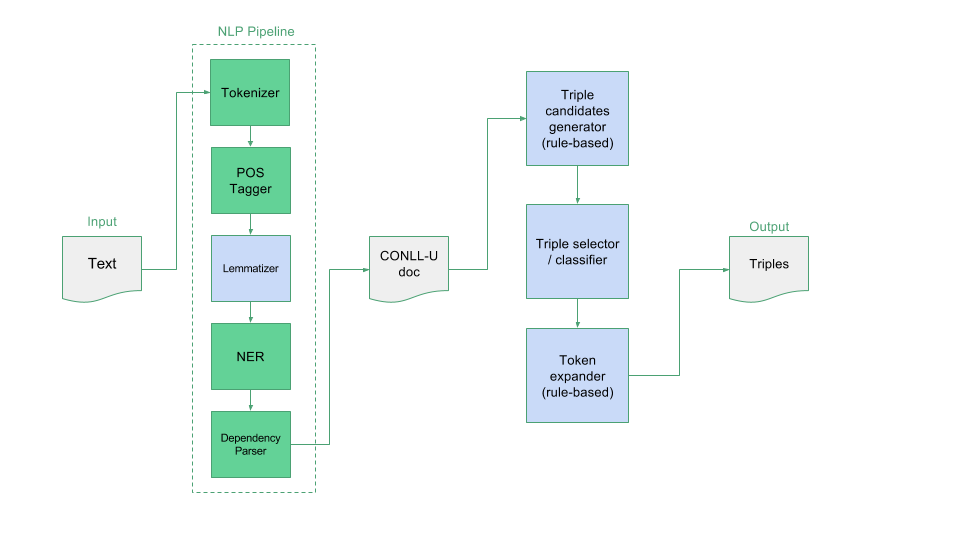
One of the most research proposes new open IE system that replaces the usage of large open patterns in Ollie(Schmitz et al. 2012) with a set of fewer patterns for canonically structured sentences and a classifier that learns to extract self-contained clauses from a sentence(Angeli, Premkumar, and Manning 2015). This system is implemented in **Stanford OpenIE** which is also integrated in the populer open source suites, Stanford Core NLP.

# Proposed System

Our proposed system also follows the pattern of three-steps(Etzioni et al. 2011) 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)(Banko et al. 2007)(Etzioni et al. 2011)(Schmitz et al. 2012), we choose to follow the method in recent research(Angeli, Premkumar, and Manning 2015) 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(Breiman 2001), as a classifier since it achieves the best score in our experiment.
3. **Extract**: use the classifier to extract relations (predicates) 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 [fig\_program\_flowchart], our system is composed of four main components: **NLP pipeline**, **triple candidate generator**, **triple selector** and **token expander**. Each of them are explained further in following subsections.



## NLP Pipeline

The NLP pipeline is a series of NLP tasks that annotates one or more sentences and saves them in CONLL-U[[4]](#footnote-34) format, a token-based sentence annotation format containing lemma, POS tag, dependency relation and a slot for additional annotation. The pipeline assumes that each sentence in the input document is separated by new line so preprocessing may be required. The detail of each model the pipeline are described below:

1. Tokenizer  
   We use default tokenizer provided by Stanford Core NLP, PTBTokenizer(C. Manning et al., n.d.), which mimics Penn Treebank 3 tokenizer[[5]](#footnote-36). 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(Toutanova et al. 2003) with Indonesian universal POS tag dataset which we convert from dependency parsing dataset[[6]](#footnote-38). 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(Suhartono 2014) 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[[7]](#footnote-40). We use lemma as one of the features for NER classifier.

1. Named-Entity Recognizer (NER)

* Stanford NLP CRFClassifier(J. R. Finkel, Grenager, and Manning 2005), 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 Indonesia and a public dataset[[8]](#footnote-42).

1. Dependency Parser

* We relied on Stanford NLP nndep.DependencyParser(Chen and Manning 2014), 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[[9]](#footnote-44) (vector representation of words). Tested with 559 sentences, this model scores **70%** UAS (Unlabeled Attachment Score) and **46%** LAS (Labeled Attachment Score).

The output of the pipeline is a CONLL-U document containing annotated sentence such as Figure [fig\_conllu\_example]. The document becomes an input for next model, the triple candidate generator which is described in Section [Triple Candidates Generator]. Since the annotations that are directly used by following process are POS tag, named entity and dependency relation, we estimate that the accuracy of this NLP pipeline is **65.30%** which comes from the average of POS tagger sentence accuracy, NER F1-score (in percent) and dependency parser LAS. Additionally, this pipeline is built by extending Stanford Core NLP classes and packaged as single Java program (JAR) to improve reusability.



CONLL-U Format Example

## Triple Candidates Generator

Triple candidates generator is used to extract relation triples candidates from CONLL-U document produced by NLP pipeline. It uses a set of rules listed in Table [table\_triple\_candidate\_generation\_rules] 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(Banko et al. 2007), we cannot use them directly as training data for triple selector/classifier.

For example, applying the rules to a annotated sentence in Figure [fig\_conllu\_example] 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)

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. As part of the label step, we manually label **132 positive** and **1,479 negative** triples which we use to train binary classifier as triple selector in the learn step.

During the extraction step, 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 [fig\_program\_flowchart].

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 |

## 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 [Triple Candidates Generator], the selector will label the five check-marked triples as true and label the rest as false.

We use Random Forest(Breiman 2001), an ensemble methods that aggregate classification results from multiple decision trees, as the model for the classifier. We use the Scikit-Learn[[10]](#footnote-49) implementation of Random Forest with following configuration:

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

We discover the configuration by using Grid Search(Wasserman 2015), an exhaustive search algorithm to find optimal hyper-parameters, to find the best F1 score for Random Forest classifier using dataset described in Section [Triple Candidates Generator].

We extract 17 features described in Table [table\_models\_features] from each triple candidates. These features are based on POS tag, named-entity and dependency relation, instead of shallow syntactic features used by TextRunner or ReVerb(Banko et al. 2007)(Etzioni et al. 2011). Every nominal features are also encoded and normalized along with the whole dataset by removing the mean and scaling to unit variance in order to improve the precision and recall of the classifier.

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 |
| 6 | Subject token’s dependency with predicate |
| 7 | Predicate token’s POS tag |
| 8 | Predicate token’s dependency relation |
| 9 | Predicate token’s head POS tag |
| 10 | Predicate token’s dependents count |
| 11 | Object token’s POS tag |
| 12 | Object token’s dependency relation |
| 13 | Object token’s head POS tag |
| 14 | Object token’s named entity |
| 15 | Object token’s dependents count |
| 16 | Object token’s distance from predicate |
| 17 | Object token’s dependency with predicate |

During the train step, we use the dataset prepared in the label step (described in Section [Triple Candidates Generator]) to train triple selector and save the best model as binary file. This model is included in the system to be use during the extraction step.

## Token Expander

Instead of using lightweight noun phrase chunker(Banko et al. 2007), our system uses rule-based token expander to extract relation or argument clauses. While having different objective and approach, this token expander works similarly to Clause Selector in Stanford Open IE(Angeli, Premkumar, and Manning 2015) where the algorithm starts from a token then decides whether to expand to its dependents. Instead of using machine learning model like Clause Selector, it uses simple heuristics based on syntactical features (POS tag, dependency relation and named-entity) described in Table [table\_token\_expansion\_rules\_s\_o] and Table [table\_token\_expansion\_rules\_p] to determine whether to: (1) expand a token to its dependent, (2) ignore the dependent or (3) remove the token itself. For example, token expander will expand check-marked triples in Section [Triple Candidates Generator] into:

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

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 |

Token expansion rules for Predicate token

|  |  |  |
| --- | --- | --- |
| **#** | **Condition for Predicate Token** | **Action** |
| 1 | If dependent is “tidak” | Expand |
| 2 | Else | Ignore |

During the label step, token expander is used to make manual annotation process easier. We label a triple candidate as valid only if it makes sense after being expanded to clause. For example, *(Sembungan, terletak, kecamatan)* doesn’t really make sense before expanded to *(Sembungan, terletak di, kecamatan Kejajar)*.

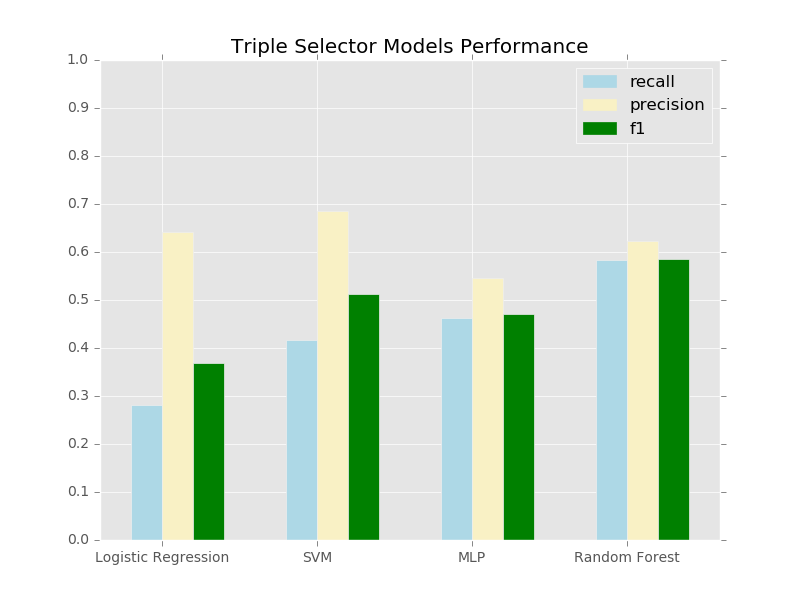
# Experiments

In this research, we report two experiments. The first one shows the performance comparison of four classifiers in selecting valid triples from given candidates. While the second one shows the scalability of our system (using the best classifier) extracting triples from documents (unannotated). Both of the experiments are run on an Ubuntu 15.04 64-bit, Intel Core i7 5500U (dual cores), DDR3 8 GB RAM, SSD 250 GB machine.

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

1. Linear Logistic Regression(Fan et al. 2008) (linear model)
2. Polynomial Support Vector Machine (SVM)(C.-C. Chang and Lin 2011) (nonlinear model)
3. Multi Layer Perceptron (MLP)(Hinton 1989) with 2 hidden layers (20 and 10 ReLU(Nair and Hinton 2010) neurons)
4. Random Forest(Wasserman 2015) (ensemble decision trees)

We use the manually annotated triple selector dataset described in Section [Triple Candidates Generator] to cross-validate(Kohavi and others 1995) (k-Fold with ) the four classifiers. Since open IE systems requires both precision and recall(Angeli, Premkumar, and Manning 2015), we choose F1 score to determine the best classifier for triple selector. The result of this experiment is shown by Figure [fig\_models\_performance] and Table [table\_models\_performance] where Random Forest achieves the highest F1 score 0.58.



Triple selector models performance comparison chart

Triple selector models performance

|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | **P** | **R** |  |
| 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** |

In the second experiment, we evaluate the performance of our system by extracting triples from three documents with different number of sentences, measuring the total execution time and calculating the average execution time per sentence. The result in Table [table\_system\_extraction\_time] shows that the lowest execution time (or fastest execution time) is 0.014 seconds when processing document of 5,593 sentences.

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 |

# Analysis

The first experiment shows that all classifiers are still having problem learning the pattern of triples when cross-validated using which means two thirds of our dataset is insufficient to cover the patterns in other one third part. The dataset also suffers unbalance 1:11 ratio of positive and negative samples which is caused by lack of efficiency in triple candidates generator. To solve this issue, we plan to annotate more sentences to increase the coverage and improve the efficiency of triple candidates generator. The low performance of linear logistic regression indicates that this problem is not linearly separable. The random forest performs better than other nonlinear models (SVM and MLP) because it is easily tuned to balance the precision and recall by changing the number and the depth of decision trees.

We are also aware that the heuristics used in triple candidates generator and token expander are still limited to explicit pattern. For instance, triple candidate generator can not extract relations *(kecamatan Kejajar, terletak di, Jawa Tengah)* and *(Jawa Tengah, terletak di, Indonesia)* from the sentence in Figure [fig\_example\_io\_openie] yet. In the future research, we plan to improve the model to extract implicit patterns while keeping the number of negative candidates. The token expander is having problem in expanding token to implicitly expected clauses such as *“seorang pelatih sepak bola”* from *“seorang pelatih dan pemain sepak bola”* or *“satu buah torpedo”* from *“satu atau dua buah torpedo”*. We expect there will be more patterns that need to be considered in order to properly expand the token so further research on effective model to achieve this is required. Also, in order to properly evaluate the performance of these components, we need to create test datasets for both triple candidates generator and token expander.

Additionally, through the second experiment, we also find that our system average extraction performance is 0.014 seconds/sentence (for 5,593 sentences document) which is still comparable to TextRunner(Banko et al. 2007). Therefore, in contrast to the argument proposed in the related work(Banko et al. 2007)(Etzioni et al. 2011), this experiment shows that the heavy linguistic tasks such as dependency parsing doesn’t cause performance drawback in big document, assuming the average number of sentences in document do not exceed 5,593.

# Conclusion

This paper introduces an open domain information extraction system for Indonesian text using basic NLP pipelines and combination of heuristics and machine learning models. The system is able to extract meaningful domain-independent relations from Indonesian sentences to be used as document representation or document understanding task. Additionally, the source code and datasets are published openly[[11]](#footnote-56) to improve research reproducibility.

In the future, we plan to improve the performance of our system finding better heuristics for triple candidates generator to reduce the negative samples. We also plan adding more training data for triple selector to improve the precision and recall score. We also need to create dataset for triple candidates generator and token expander in order to properly evaluate further improvement of both components. We also consider adding confidence level in the output of every phases (NLP pipelines, candidate generator, triple selector, token expander) and including them as features and/or heuristics may also improve the overall performance of the system.

Angeli, Gabor, Melvin Johnson Premkumar, and Christopher D Manning. 2015. “Leveraging Linguistic Structure for Open Domain Information Extraction.” In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (Acl 2015)*.

Auer, Sören, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. “Dbpedia: A Nucleus for a Web of Open Data.” *The Semantic Web*. Springer, 722–35.

Banko, Michele, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. 2007. “Open Information Extraction from the Web.” In *IJCAI*, 7:2670–6.

Breiman, Leo. 2001. “Random Forests.” *Machine Learning* 45 (1). Springer: 5–32.

Chang, Chih-Chung, and Chih-Jen Lin. 2011. “LIBSVM: A Library for Support Vector Machines.” *ACM Transactions on Intelligent Systems and Technology (TIST)* 2 (3). ACM: 27.

Chen, Danqi, and Christopher D Manning. 2014. “A Fast and Accurate Dependency Parser Using Neural Networks.” In *EMNLP*, 740–50.

Etzioni, Oren. 2011. “Search Needs a Shake-up.” *Nature* 476 (7358). Nature Publishing Group: 25–26.

Etzioni, Oren, Anthony Fader, Janara Christensen, Stephen Soderland, and others. 2011. “Open Information Extraction: The Second Generation.” In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Exner, Peter, and Pierre Nugues. 2014. “REFRACTIVE: An Open Source Tool to Extract Knowledge from Syntactic and Semantic Relations.” In *LREC*, 2584–9.

Fader, Anthony, Stephen Soderland, and Oren Etzioni. 2011. “Identifying Relations for Open Information Extraction.” In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1535–45. Association for Computational Linguistics.

Fan, Rong-En, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. “LIBLINEAR: A Library for Large Linear Classification.” *Journal of Machine Learning Research* 9 (Aug): 1871–4.

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. 2005. “Incorporating Non-Local Information into Information Extraction Systems by Gibbs Sampling.” In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 363–70. Association for Computational Linguistics.

Hinton, Geoffrey E. 1989. “Connectionist Learning Procedures.” *Artificial Intelligence* 40 (1-3). Elsevier: 185–234.

Kohavi, Ron, and others. 1995. “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.” In *Ijcai*, 14:1137–45. 2. Stanford, CA.

Manning, C, T Grow, T Grenager, J Finkel, and J Bauer. n.d. “PTBTokenizer.”

Nair, Vinod, and Geoffrey E Hinton. 2010. “Rectified Linear Units Improve Restricted Boltzmann Machines.” In *Proceedings of the 27th International Conference on Machine Learning (Icml-10)*, 807–14.

Schmitz, Michael, Robert Bart, Stephen Soderland, Oren Etzioni, and others. 2012. “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*, 523–34. Association for Computational Linguistics.

Suhartono, Derwin. 2014. “Lemmatization Technique in Bahasa: Indonesian.” *Journal of Software* 9 (5): 1203.

Toutanova, Kristina, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. “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*, 173–80. Association for Computational Linguistics.

Wasserman, Daniel. 2015. “Grid Search Optimization.”

1. Resource Data Format W3C <https://www.w3.org/RDF/> [↑](#footnote-ref-22)
2. Stanford Core NLP <https://stanfordnlp.github.io/CoreNLP> [↑](#footnote-ref-24)
3. Stanford Open IE <https://nlp.stanford.edu/software/openie.html> [↑](#footnote-ref-27)
4. CONLL-U format description <http://universaldependencies.org/format.html> [↑](#footnote-ref-34)
5. Penn Treebank 3 <https://catalog.ldc.upenn.edu/LDC99T42> [↑](#footnote-ref-36)
6. UD Indonesian dataset <https://github.com/UniversalDependencies/UD_Indonesian> [↑](#footnote-ref-38)
7. Indonesian Lemmatizer <https://github.com/davidchristiandy/lemmatizer> [↑](#footnote-ref-40)
8. Indonesian NER <https://github.com/yusufsyaifudin/indonesia-ner> [↑](#footnote-ref-42)
9. Indonesian word embedding <https://github.com/yohanesgultom/id-openie/blob/master/data/parser-id.embed> [↑](#footnote-ref-44)
10. scikit-learn: machine learning in Python <http://scikit-learn.org> [↑](#footnote-ref-49)
11. Paper source code <https://github.com/yohanesgultom/id-openie> [↑](#footnote-ref-56)