Automatic Open Domain Information Extraction from Indonesian Text

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

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

***Abstract*—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 an NLP pipeline, rule-based candidates generator, rule-based 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.**

1. Introduction

Open domain information extraction (open IE) is a paradigm that facilitates domain-independent discovery of triple relations from text document [**?**]. 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 [**?**]. In more linguistic term, the arguments are also referred as subject and object while relation are referred as predicate [**?**]. The example of this extraction is described in Figure 1.

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 some possible output of open domain information extraction for Indonesian text

As described in Table 1, 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] like KE [**?**] [**?**]. Although mapping to existing relation schema is required in real word task such as slot filling [**?**], 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 [**?**] and information retrieval [**?**].

1. Resource Data Format W3C https://www.w3.org/RDF/

TABLE 1. GENERAL COMPARISON BETWEEN TRADITIONAL INFORMATION, OPEN DOMAIN INFORMATION AND KNOWLEDGE EXTRACTION

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 [**?**]. 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 an 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] 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.

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[3] which is an implementation of novel open IE system that outperforms Ollie in TAC-KBP 2013 Slot Filling task [**?**].

TextRunner is designed for massive size of open-domain web documents by avoiding heavy linguistic tasks and used inverted index to store extraction result [**?**]. 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 millions of web documents. Additionally, it also uses redundancy assessor to estimate probability of a triple based on its occurrence in a document.

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

R2A2 is a system built to fix argument extraction problem in ReVerb [**?**]. Instead of using heuristics to extract the arguments, it uses a

2. Stanford Core NLP https://stanfordnlp.github.io/CoreNLP

3. Stanford Open IE https://nlp.stanford.edu/software/openie.html

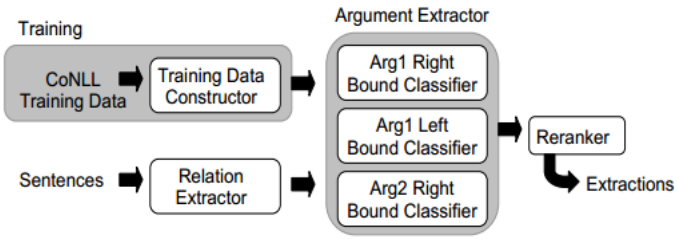

Figure 2. ArgLearner architecture training and extraction architecture

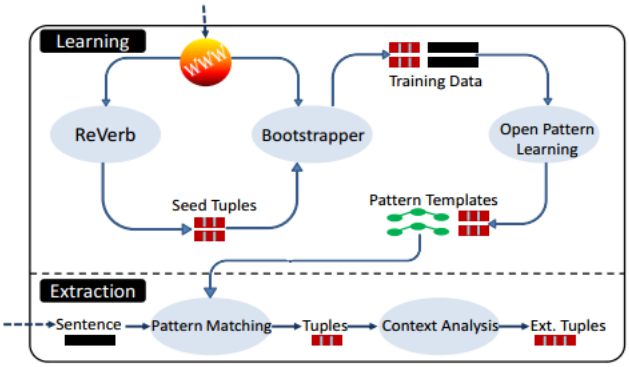

Figure 3. Ollie labeling and extraction architecture

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) [**?**] 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 described by 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 [**?**]. This system is implemented in **Stanford OpenIE** which is also integrated in the populer open source suites, Stanford Core NLP.

3. Proposed System

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

1) **Label**: sentences 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) [**?**] [**?**] [**?**]schmitz2012open, we choose to follow the method in recent research [**?**] 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 [**?**], 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 4, 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.

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[4] 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 in the pipeline are described below:

4. CoNLL-U format description http://universaldependencies.org/format.html

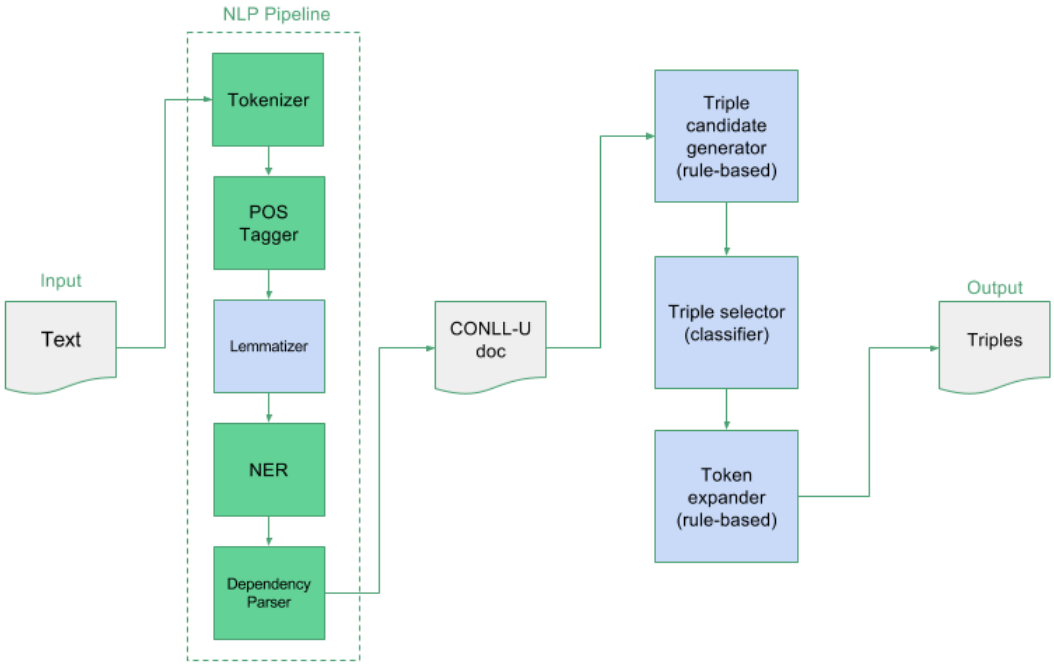

Figure 4. Indonesian open domain information extraction flowchart

1) Tokenizer
We use default tokenizer provided by Stanford Core NLP, `PTBTokenizer` [**?**], which mimics Penn Treebank 3 tokenizer[5]. 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` [**?**] with Indonesian universal POS tag dataset which we convert from dependency parsing dataset[6]. 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 [**?**] 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]. We use lemma as

5. Penn Treebank 3 https://catalog.ldc.upenn.edu/LDC99T42

6. UD Indonesian dataset https://github.com/UniversalDependencies/UD_Indonesian

7. Indonesian Lemmatizer https://github.com/davidchristiandy/lemmatizer

one of the features for NER classifier.

4) Named-Entity Recognizer (NER)
Stanford NLP `CRFClassifier` [**?**], 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 model 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].

5) Dependency Parser
We relied on Stanford NLP `nndep.DependencyParser` [**?**], to annotate dependency relation of each token in the sentence. We train this transition-based neural network model using an Indonesian universal dependencies dataset of 5,036 sentences and 3,093 Indonesian word embedding[9] (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 5. The document becomes an input for next model, the triple candidate generator which is described in Section 3.2. 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.

8. Indonesian NER https://github.com/yusufsyaifudin/indonesia-ner

9. Indonesian word embedding https://github.com/yohanesgultom/id-openie/blob/master/data/parser-id.embed

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. Example of CoNLL-U sentence annotation format

3.2. Triple Candidate Generator

Triple candidate 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 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 [**?**], we cannot use them directly as training data for triple selector/classifier.

For example, applying the rules to an annotated sentence in Figure 5 will generate these 17 triples candidates where only five of them are valid triples (check-marked):

- (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) ✓

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

- (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 candidate 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 candidate 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 [**?**], an ensemble method that aggregate classification results from multiple decision trees, as the model for the classifier. We use the Scikit-Learn[10] 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 [**?**], an exhaustive search algorithm to find optimal hyper-parameters for a certain evaluation metric. We use this algorithm to find the best F1 score for Random Forest classifier using dataset described in Section 3.2.

We extract 17 features described in Table 3 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 [**?**] [**?**]. Every nominal feature 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.

During the train step, we use the dataset 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.

10. scikit-learn: machine learning in Python http://scikit-learn.org

TABLE 3. TRIPLE SELECTOR FEATURES

#	Triple 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

3.4. Token Expander

Instead of using lightweight noun phrase chunker [**?**], 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 [**?**] 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 4 and Table 5 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 3.2 into:

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

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

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 seem to make sense before expanded to *(Sembungan, terletak di, kecamatan Kejajar)*.

4. Experiments

In this research, we report two experiments. The first one shows the performance comparison

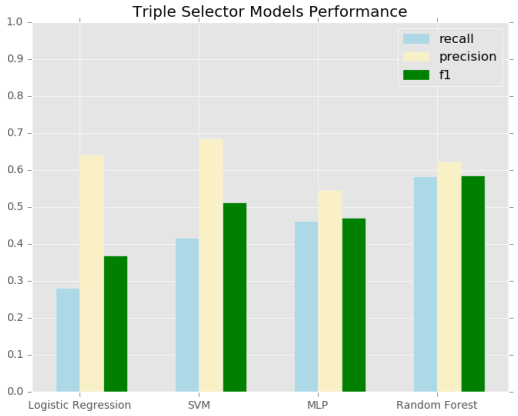

Figure 6. Triple selector models performance comparison chart

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 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 [**?**] (linear model)
2) Polynomial Support Vector Machine (SVM) [**?**] (nonlinear model)
3) Multi-Layer Perceptron (MLP) [**?**] with 2 hidden layers (20 and 10 ReLU [**?**] neurons)
4) Random Forest [**?**] (ensemble decision trees)

We use the manually annotated triple selector dataset described in Section 3.2 to cross-validate [**?**] (k-Fold with $k = 3$) the four classifiers. Since open IE systems requires both precision and recall [**?**], we choose F1 score to determine the best classifier for triple selector. The result of this experiment is shown by Figure 6 and Table 6 where Random Forest achieves the highest F1 score 0.58.

In the second experiment, we evaluate the performance of our system by extracting triples

TABLE 6. TRIPLE SELECTOR MODELS PERFORMANCE

Model	**P**	**R**	**F1**
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	3.050
138	429	11.3	0.082
5,593	19,403	78.6	0.014

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 7 shows that the lowest execution time (or fastest execution time) is 0.014 seconds when processing document of 5,593 sentences.

5. Analysis

The first experiment shows that all classifiers are still having problem learning the pattern of triples when cross-validated using $k = 3$ 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 candidate generator. To solve this issue, we plan to annotate more sentences to increase the coverage and improve the efficiency of triple candidate 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 candidate 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 1 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 candidate 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 [**?**]. Therefore, in contrast to the argument proposed in the related work [**?**] [**?**], 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.

6. 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] to improve research reproducibility.

In the future, we plan to improve the performance of our system finding better heuristics for triple candidate 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 candidate 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.

11. Paper source code https://github.com/yohanesgultom/id-openie