UNIVERSITAS INDONESIA

OPEN DOMAIN INFORMATION EXTRACTION OTOMATIS DARI TEKS BAHASA INDONESIA

TESIS

YOHANES GULTOM
1506706345

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI MAGISTER ILMU KOMPUTER
DEPOK
JUNI 2017

UNIVERSITAS INDONESIA

OPEN DOMAIN INFORMATION EXTRACTION OTOMATIS DARI TEKS BAHASA INDONESIA

TESIS
Diajukan sebagai salah satu syarat untuk memperoleh gelar
Magister Ilmu Komputer

YOHANES GULTOM
1506706345

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI MAGISTER ILMU KOMPUTER
DEPOK
JUNI 2017

ABSTRAK

Nama	:	Yohanes Gultom
Program Studi	:	Magister Ilmu Komputer
Judul	:	Open Domain Information Extraction Otomatis dari Teks Bahasa Indonesia

Banyaknya jumlah dokumen digital yang tersedia saat ini sudah melebihi kapasitas manusia untuk memprosesnya secara manual. Hal ini mendorong munculnya kebutuhan akan metode ekstrasi informasi (*information extraction*) otomatis dari teks atau dokumen digital dari berbagai domain (*open domain*). Sayangnya, sistem *open domain information extraction* (*open IE*) yang ada saat ini hanya berlaku untuk bahasa tertentu saja. Selain itu belum ada sistem *open IE* untuk bahasa Indonesia yang dipublikasikan. Pada penelitian ini Penulismemperkenalkan sebuah sistem untuk mengekstraksi relasi antar entitas dari teks bahasa Indonesia dari berbagai domain. Sistem ini menggunakan sebuah NLP *pipeline*, pembangkit kandidat *triple* (*triple candidates generator*) dan pengembang token (*token expander*) berbasis aturan serta pemilih *triple* berbasis *machine learning*. Setelah melakukan *cross-validation* terhadap empat kandidat model: *logistic regression*, SVM, MLP dan *Random Forest*, Penulismenemukan bahwa *Random Forest* adalah *classifier* yang terbaik untuk dijadikan *triple selector* denan skor F1 0.58 (*precision* 0.62 dan *recall* 0.58). Penyebab utama skor yang masih rendah ini adalah aturan pembangkitan kandidat yang masih sederhana dan cakupan pola *dataset* yang masih rendah.

Kata Kunci:
information extraction, *open domain*, *natural language processing*, bahasa Indonesia

ABSTRACT

Name	:	Yohanes Gultom
Program	:	Magister Ilmu Komputer
Title	:	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 an NLP pipeline, 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.

Keywords:
information extraction, open domain, natural language processing, Indonesian language

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Di masa sekarang ketersediaan dokumen digital berbahasa natural seperti berita, jurnal dan buku elektronik (*e-book*) sudah sangat banyak dan terus meningkat dengan cepat karena didorong oleh meningkatnya pemanfaatan komputer, *smartphone* dan *internet*. Jumlah dokumen digital tersebut telah melampaui batas kemampuan manusia untuk memproses secara manual sehingga menimbulkan kebutuhan akan proses otomatis untuk melakukannya (Banko et al., 2007). Salah satu proses yang dikembangkan adalah *information extraction* (IE) yang secara selektif menyusun dan mengkombinasikan data yang ditemukan di dalam teks atau dokumen menjadi informasi(Cowie and Lehnert, 1996).

Meskipun *IE* sudah mampu manusia untuk memproses dokumen digital dengan lebih efisien, metode yang digunakan umumnya hanya berlaku untuk kelompok dokumen yang homogen atau berada dalam satu domain (*closed-domain*). Hal ini terjadi karena umumnya teknik yang dipakai dibuat sedemikian rupa untuk memanfaatkan pola tertentu pada teks atau dokumen(Cowie and Lehnert, 1996). Sebagai contoh untuk mengekstraksi nama penulis dari berita elektronik, salah satu cara paling mudah adalah mencari nama orang di awal atau akhir dokumen. Cara yang sama tidak bisa digunakan untuk mencari nama penulis dari dokumen lain seperti jurnal karena struktur dokumen yang berbeda. Hal ini mendorong berkembangnya metode lain yang mampu mengekstraksi informasi dari berbagai domain (*open domain*) yang disebut *open domain information extraction* (*open IE*) (Banko et al., 2007).

Seiring dengan berkembangnya waktu, beberapa sistem *open IE* sudah dikembangkan (Schmitz et al., 2012) untuk bahasa Inggris. Bahkan penelitian terkait melaporkan kesuksesan aplikasi open IE untuk *task question answering* (Fader et al., 2011) dan *information retrieval* (Etzioni, 2011). Akan tetapi karena sistem open IE menggunakan satu atau lebih *task natural language processing* (NLP) dan aturan/heuristik yang hanya berlaku untuk bahasa tertentu, maka sistem yang berkembang tidak dapat dipakai untuk memproses teks atau dokumen dalam bahasa lain seperti bahasa Indonesia. Oleh karena itu dalam penelitian ini, Penulis memperkenalkan sistem open IE untuk bahasa Indonesia.

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)

Gambar 1.1: Contoh input dan output yang diharapkan dari sistem open IE untuk bahasa Indonesia

Sistem open IE yang Penulis ajukan bertujuan untuk mengekstrak sejumlah *triple* dari satu atau lebih teks bahasa Indonesia seperti contoh pada Gambar 1.1. Sistem ini terdiri dari sebuah NLP *pipeline*, pembangkit kandidat *triple* (*triple candidate generator*), pengembang token (*token expander*) dan sebuah model *machine learning* untuk memilih *triple* (*triple selector*). Untuk melatih model *triple selector* tersebut, Penulis juga membuat dataset berisi 1.611 kandidat *triple* bahasa Indonesia yang valid dan yang tidak valid. Sistem ini diharapkan dapat menjadi referensi dalam pengembangan open IE untuk bahasa Indonesia dan juga digunakan untuk kebutuhan aplikasi yang lebih kompleks seperti pendeteksian plagiarisme, *question answering* dan *knowledge extraction*.

1.2 Permasalahan

Pada bagian ini akan dijelaskan mengenai definisi permasalahan yang ingin diselesaikan pada penelitian ini serta batasan yang ditetapkan.

1.2.1 Definisi Permasalahan

Permasalahan yang ditemukan dan ingin diselesaikan pada penelitian ini:

1. Bagaimana merancang sistem *open IE* yang cocok untuk bahasa Indonesia?
2. Bagaimana implementasi sistem *open IE* tersebut?

1.2.2 Batasan Permasalahan

Batasan permasalahan pada penelitian ini adalah:

1. Penelitian ini hanya berfokus untuk menghasilkan *triple* yang eksplisit secara sintaktik. Contoh *triple* yang eksplisit dari kalimat "*Universitas Indonesia berada di Depok, Jawa Barat, Indonesia*" adalah *(Universitas Indonesia, terletak di, Depok)*. Sedangkan *triple* yang implisit seperti *(Depok, terletak di, Jawa Barat)* belum ditangani pada penelitian ini.

2. Proses dibatasi pada dokumen teks bahasa Indonesia yang setiap barisnya hanya berisi satu kalimat. Praproses yang dibutuhkan untuk menggubah dokumen dari format yang berbeda tidak dibahas di penelitian ini.

3. Algoritma *tokenization* yang dipakai pada penelitian ini menggunakan aturan untuk bahasa Inggris sehingga belum menangani *token* khusus untuk bahasa Indonesia ("Ny.", "Dra.", "dkk.", dsb.).

4. Penelitian ini tidak berfokus untuk mencapai kinerja sistem yang sebanding dengan sistem open IE untuk bahasa Inggris pada penelitian terkait.

1.3 Tujuan dan Manfaat

Tujuan dan manfaat dari penelitian ini adalah:

Tujuan

1. Merancang sistem open IE untuk teks bahasa Indonesia.

2. Mengimplementasikan sistem open IE untuk teks bahasa Indonesia.

Manfaat

1. Menghasilkan sistem *open IE* yang dapat digunakan untuk mengekstrak entitas relasi dan argumen/entitas dalam format *triple* dari teks bahasa Indonesia

2. Memberikan acuan untuk pengembangan sistem *open IE* untuk bahasa Indonesia

3. Memberikan kontribusi terhadap perkembangan sumber daya bahasa (*language resources*) Indonesia

1.4 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN
  Bab ini akan menjelaskan mengenai latar belakang permasalahan, rumusan masalah, tujuan, manfaat dan batasan penelitian.

- Bab 2 TINJAUAN PUSTAKA
  Bab ini akan menjelaskan landasan teori yang digunakan pada penelitian ini serta memaparkan kajian pustaka terhadap penelitian-penelitian terkait.

- Bab 3 METODE PENELITIAN
  Bab ini akan menjelaskan mengenai tahapan, rancangan & implementasi sistem, pengumpulan & pengolahan data dan teknik evaluasi yang digunakan pada penelitian ini.

- Bab 4 HASIL DAN ANALISIS
  Bab ini akan menjelaskan tentang hasil eksperimen dan analisis hasil eksperimen.

- Bab 5 PENUTUP
  Bab ini akan menjelaskan tentang kesimpulan dari penelitian yang telah dilakukan dan saran untuk penelitian berikutnya.

BAB 2
TINJAUAN PUSTAKA

Pada bab ini dijelaskan mengenai penelitian terkait dan berbagai dasar teori yang menunjang penelitian ini.

2.1 Penelitian Terkait

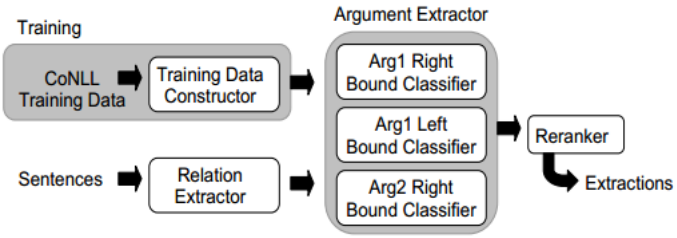
Sejak pertama kali diperkenalkan pada tahun 2007 (Banko et al., 2007), sudah ada beberapa penelitian mengenai *open IE* untuk bahasa Inggris yang dipublikasikan. Sistem *open IE* yang pertama diperkenalkan adalah TEXTRUNNER (Banko et al., 2007). Sistem ini kemudian dikembangkan oleh sistem-sistem dari penelitian berikutnya yaitu (secara berurutan) REVERB (Fader et al., 2011), R2A2 (Etzioni et al., 2011) dan kemudian OLLIE (Schmitz et al., 2012) . Selain itu, salah satu penelitian terbaru juga memperkenalkan sistem *open IE* baru, STANFORD OPEN IE, yang berhasil mengungguli kinerja OLLIE dalam TAC-KBP 2013 *Slot Filling task* (Angeli et al., 2015).

Sistem *open IE* yang pertama diperkenalkan adalah TEXTRUNNER. Sistem ini didesain untuk mengekstrak informasi secara efisien dari halaman-halaman *web* di internet yang jumlahnya sangat besar dan memiliki domain yang berbeda-beda (Banko et al., 2007). Informasi yang diekstrak merupakan *tuple* $t = (e_i, r_{i,j}, e_j)$ di mana $r_{i,j}$ adalah relasi antara entitas e_i dan e_j dalam sebuah kalimat. TEXTRUNNER terdiri dari tiga modul utama (Banko et al., 2007) yaitu: (1) *Self-Supervised Learner*, modul yang melatih sebuah *naive bayes classifier* (NBC) untuk mengenali kandidat *triple* yang valid tanpa memerlukan campur tangan manusia (*self-supervised*), (2) *Single-Pass Extractor*, modul yang mengekstrak sejumlah kandidat *triple* dari setiap kalimat dan menyimpan kandidat yang dianggap valid oleh *classifier*, dan (3) *Redundancy-based Assessor*, modul yang menghitung probabilitas kemunculan *triple* dalam satu dokumen. Sistem ini mampu mengekstrak informasi per kalimat dengan akurasi rata-rata 88% dan mampu memproses 9 juta halaman *web* dalam 68 *CPU hours* (Banko et al., 2007).

REVERB adalah sistem *open IE* yang dikembangkan untuk memperbaiki dua masalah pada pendahulunya, TEXTRUNNER. Masalah yang ingin diselesaikan oleh REVERB adalah inkoherensi hasil ekstraksi *incoherent extractions* dan hasil ekstraksi yang tidak informatif *uninformative extractions* (Fader et al., 2011). Un-

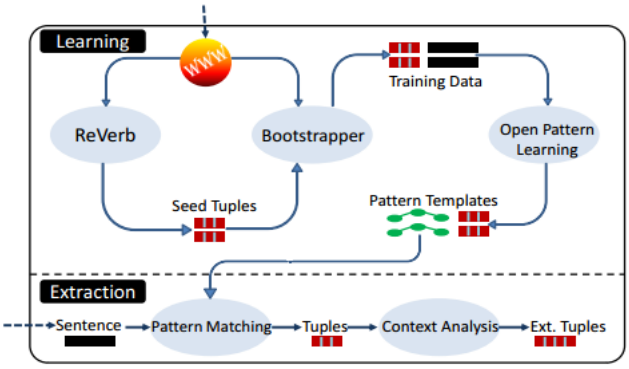
tuk mengekstrak *triple* $t = (e_i, r_{i,j}, e_j)$, sistem ini menggunakan dua algoritma utama, yaitu (1) *Relation Extraction*, algoritma yang mengekstrak relasi $r_{i,j}$ menggunakan pembatasan sintaktik dan leksikal yang menyelesaikan dua masalah tersebut, dan (2) *Argument Extraction*, algoritma yang mencari entitas e_i dan e_j yang dihubungkan oleh relasi $r_{i,j}$ menggunakan heuristik. REVERB menerima *input* berupa kalimat yang telah dianotasi POS-nya % potongan frase kata bendanya (NP *chunk*) dan menghasilkan *output* sejumlah *triple*. Dari hasil pengujian yang dilakukan, REVERB mencapai *precision* dan *recall* yang hampir dua kali lebih baik dari TEXTRUNNER (Fader et al., 2011).

Jika REVERB memperbaiki masalah pada ekstraksi relasi, R2A2 berfokus untuk memperbaiki ekstraksi argumen/entitas (Etzioni et al., 2011). Jika REVERB hanya menggunakan aturan atau heuristik untuk mengekstraksi argumen (Fader et al., 2011), R2A2 menggunakan modul berbasis *machine learning*, ARGLEARNER. Modul ini menerima relasi dan kalimat sebagai *input* dan mengembalikan dua buah argumen sebagai *output*. Modul ini menggunakan tiga buah *classifier* berbasiskan REPTREE (Hall et al., 2009) dan *sequence labeling* CRF (McCallum, 2002) untuk mengekstrak argumen dari kalimat melalui proses yang ditunjukkan pada Gambar 2.1 (Etzioni et al., 2011).



Gambar 2.1: Proses pelatihan dan ekstraksi ARGLEARNER

Penelitian berikutnya memperkenalkan OLLIE (*Open Language Learning for Information Extraction*) (Schmitz et al., 2012) yang menjadikan REVERB sebagai salah satu modulnya. OLLIE menggunakan REVERB untuk mencari sejumlah (*open pattern*)/*template* sebagai panduan untuk mengekstrak *triple* dari kalimat. Perbedaan lain sistem ini dengan pendahulunya adalah relasi yang diekstrak tidak hanya dari kata kerja (*verb*) tetapi bisa juga diekstrak secara implisit dari kata benda (*noun*), kata sifat (*adjective*) (Schmitz et al., 2012). Selain itu OLLIE juga menambahkan modul untuk melakukan analisis dan penambahan informasi kontekstual pada hasil ekstraksi sehingga presisi lebih tinggi. Dua modul utama ini diajukan untuk memperbaiki kekurangan dari REVERB yaitu pembatasan relasi hanya pada kata kerja (*verb*) dan pengabaian konteks kalimat (Schmitz et al., 2012). Proses pelabelan (*labeling*) data latih dan ekstraksi OLLIE ditunjukkan pada Gambar 2.2.



Gambar 2.2: Proses *labeling* dan ekstraksi pada OLLIE

Salah satu riset terbaru memperkenalkan model sistem *open IE* yang menggganti penggunaan banyak *open pattern/template* untuk mengekstrak *triple* pada OLLIE (Schmitz et al., 2012) dengan hanya enam pola atomik (*atomic patterns*) (Angeli et al., 2015). Enam pola atomik itu digunakan untuk mengekstrak *triple* dari klausa yang *self-contained* dan *maximally compact*. Modul ekstraktor *inter-clauses*, yang menggunakan *multinomial logistic regression classifier*, bertanggungjawab menghasilkan klausa yang *self-contained* (independen secara sintaktik dan semantik), dan modul ekstraktor *intra-clause*, yang menggunakan model *natural logic* (MacCartney and Manning, 2007), mengubahnya menjadi klausa yang *maximally compact* (tidak mengandung kata redundan). Model sistem ini diimplementasikan dalam STANFORD OPEN IE, yang merupakan bagian dari kakas NLP *opensource*, *Stanford Core NLP*[1].

2.2 *Open Domain Information Extraction*

Open domain information extraction (*open IE*) adalah proses ekstraksi informasi dari dokumen dalam format *triple* (x, r, y) di mana r adalah relasi antara dua buah argumen/entitas x dan y (Banko et al., 2007; Etzioni et al., 2011). Relasi pada *triple* diambil dari kata kerja (*verb*) (Banko et al., 2007; Fader et al., 2011) (contoh: kalimat "*Jakarta is the capital of Indonesia*" mengandung *triple* ("Jakarta", "is the capital of", "Indonesia")) atau dari kata lain yang secara implisit merupakan kata kerja (Schmitz et al., 2012) (contoh: "*Indonesian President Joko Widodo was born in Surakarta*" mengandung *triple* ("Joko Widodo", "be", "president")). Sedangkan argumen atau entitas yang diekstrak selalu merupakan frase (*noun phrase*) seperti yang juga terlihat di contoh. Format *triple* ini ternyata berlaku umum untuk semua dokumen yang berisi teks bahasa natural sehingga dapat diterapkan pada dokumen

[1]Stanford Core NLP https://stanfordnlp.github.io/CoreNLP/

dari berbagai domain. Format *triple* yang digunakan *open IE* memiliki kemiripan dengan format yang lazim digunakan pada *knowledge extraction* (KE), yaitu *Resource Data Format* (RDF)[2] (Auer et al., 2007; Exner and Nugues, 2014). Namun, perbedaannya adalah *triple* pada *open IE* umumnya tidak mengikuti seluruh spesifikasi RDF dan tidak memiliki himpunan ontologi tetap. Ringkasan perbandingan antara open IE dan KE ditunjukkan pada Tabel 2.1.

Tabel 2.1: Perbandingan antara *information extraction* tradisional (IE), *open domain extraction* (open IE) dan *knowledge extraction* (KE)

Aspek	**IE**	**Open IE**	**KE**
Domain	Tertutup	Terbuka	Terbuka
Format	Tergantung domain	Triples	RDF Triples
Ontologi	Tidak tersedia	Opsional	Wajib

Meskipun menggunakan modul dan teknik yang berbeda-beda, model sistem *open IE* umumnya menjalankan proses yang dapat dibagi menjadi tiga langkah/fase (Etzioni et al., 2011):

1. Label (*label*): membangun data latih untuk *classifier* baik secara manual atau otomatis.

2. Belajar (*learn*): melatih *classifier* untuk mengekstrak himpunan *triple* dari kalimat menggunakan data dari fase Label.

3. Ekstrak (*extract*): mengekstrak himpunan *triple* dari kalimat menggunakan *classifier* yang telah dilatih pada fase Belajar

Hasil ekstraksi *open IE* berguna untuk berbagai *task* seperti *question answering*, *slot filling* (Etzioni et al., 2011), *common sense knowledge acquiring* (Singh et al., 2002) dan *information retrieval* (Etzioni, 2011). Selain itu, jika dilihat sebagai representasi teks atau dokumen, himpunan *triple* dari *open IE* dapat digunakan sebagai fitur untuk klasifikasi dan *clustering* teks atau dokumen.

2.3 *Natural Language Processing*

Pemrosesan bahasa natural atau *natural language processing* (NLP) tidak bisa dipisahkan dari information extraction (Banko et al., 2007; Fader et al., 2011; Etzioni et al., 2011; Angeli et al., 2015). Semua model sistem *open IE* juga selalu membutuhkan informasi yang dihasilkan oleh *task* NLP seperti *part of speech tagging*,

[2]Resource Data Format W3C https://www.w3.org/RDF/

dependency parsing dan *named-entity recognition*. Informasi tersebut digunakan sebagai variabel dalam heuristik *open IE* dan juga sebagai fitur untuk *classifier*.

2.3.1 *Tokenization*

Tokenization adalah *task* NLP yang bertujuan memotong kalimat atau frase menjadi kata-kata (*tokens*) (Manning et al., 2008). Ini merupakan *task* yang paling dasar dan diperlukan sebelum dapat menjalankan *task* lainnya seperti *lemmatization*, *POS tagging*, dsb. Untuk bahasa yang ditulis secara horizontal dan setiap katanya dipisahkan oleh spasi seperti Inggris dan Indonesia, dapat digunakan algoritma berbasis aturan (*rule-based*) yang cukup sederhana (Manning et al., 2014), yaitu memotong kalimat di antara spasi dan memisahkan tanda baca sebagai *token*. Contoh *tokenization* dari kalimat ”Ibu pergi ke pasar.” adalah senarai *token* (”Ibu”, ”pergi”, ”ke”, ”pasar”, ”.”). Dalam implementasinya pada bahasa tertentu, algoritma tersebut juga disesuaikan untuk menjalankan proses yang berbeda pada *token* tertentu misalnya gelar atau singkatan yang diikuti titik (”dr.”, ”Dra.”, ”Ir.”, dsb.).

2.3.2 *Part of Speech Tagging*

Part of speech (POS) *tagging* adalah *task* NLP yang bertujuan menentukan *POS tag* atau jenis setiap kata pada kalimat (Jurafsky, 2000). Contoh *POS tag* dasar adalah kata benda (*noun*), kata kerja (*verb*), kata sifat (*adjective*) dst. Gambar 2.3 menunjukkan contoh *POS tagging* terhadap kalimat sederhana. *POS tag* dapat digunakan juga oleh *NLP task* yang lain seperti *dependency parsing* dan *named-entity recognition*.

Input: ”Ibu pergi ke pasar.”
Output: (Ibu, *noun*) (pergi, *verb*) (ke, *preposition*) (pasar, *noun*) (., *punctuation*)

Gambar 2.3: Contoh *input* dan *output POS tagging*

Algoritma *POS tagging* umumnya dapat dikelompokkan menjadi dua: berbasis aturan (*rule-based*) dan berbasis stokastik (*stochastic-based*) (Jurafsky, 2000). Salah satu algoritma yang menjadi *state-of-the-art* adalah *maximum-entropy-based POS tagger* (berbasis stokastik) yaitu *tagger* yang mempelajari model probabilitas kondisional *log-linear* (*logistic regression*) menggunakan metode *maximum entropy*.

2.3.3 *Lemmatization*

Lemmatization adalah *task* NLP yang bertujuan mengubah kata imbuhan ke bentuk *lemma* atau bentuk kamus (Suhartono, 2014). Sekalipun memiliki tujuan yang mirip dengan *stemming*, *lemmatization* tidak selalu menghasilkan kata dasar karena menggunakan analisis kosakata dan morfologi yang dapat menghindari terbuangnya *derivational affixes* (Manning et al., 2008). Jika dilakukan *stemming* dan *lemmatization* pada *token* ”penjahit” maka yang dihasikan sesuai urutan adalah adalah ”jahit” dan ”penjahit”. Hal ini bermanfaat untuk mengurangi terbuangnya informasi yang berguna. Algoritma yang dilaporkan efektif untuk bahasa Indonesia adalah algoritma berbasis aturan penghapusan imbuhan (*affixes*) dan pencarian kamus (*dictionary lookup*) (Suhartono, 2014).

2.3.4 *Named-Entity Recognition*

Named-entity recognition (NER) adalah *task* NLP yang mengenali jenis entitas dari *token* pada kalimat. Jenis entitas yang umumnya dikenali contohnya *Person* (nama orang), *Location* (nama lokasi), *Organization* (nama organisasi atau kelompok), dsb. Algoritma *state-of-the-art* untuk NER adalah yang berbasis stokastik seperti *Conditional Random Field* (CRF) dengan fitur-fitur berbasis morfologi, leksikal dan ortografik.

Input: ”Ibu Budi tinggal di Solo.”
Output: (Ibu) (Budi, *Person*) (tinggal) (di) (Solo, *Location*) (.)

Gambar 2.4: Contoh *input* dan *output* NER

2.3.5 *Dependency Parsing*

Dependency parsing adalah *task* NLP yang memetakan dan mengenali pohon hubungan antar *token* dalam kalimat. Masing-masing *token* dapat memiliki satu atau lebih *token* yang bergantung padanya (*dependents*) tapi hanya bisa memiliki satu kepala (*head*) atau tidak memiliki kepala sama sekali. Salah satu algoritma yang menjadi *state-of-the-art* untuk *dependency parsing* adalah algoritma berbasis jaringan syaraf tiruan (*neural network*) yang mempelajari transisi antar *token* (Chen and Manning, 2014).

2.3.6 CoNLL-U

CoNLL-U adalah format anotasi yang dikembangkan berdasarkan CoNLL-X, format yang disepakati dalam *Conference on Computational Natural Language Learning* ke sepuluh, yang menggunakan himpunan *POS tag* dan *dependency relation* yang berlaku untuk banyak bahasa atau universal (Nivre et al., 2016).

Himpunan *POS tag* yang dipakai pada CoNLL-U adalah:

1. ADJ: *adjective*
2. ADP: *adposition*
3. ADV: *adverb*
4. AUX: *auxiliary*
5. CCONJ: *coordinating conjunction*
6. DET: *determiner*
7. INTJ: *interjection*
8. NOUN: *noun*
9. NUM: *numeral*
10. PART: *particle*
11. PRON: *pronoun*
12. PROPN: *proper noun*
13. PUNCT: *punctuation*
14. SCONJ: *subordinating conjunction*
15. SYM: *symbol*
16. VERB: *verb*
17. X: *other*

Sementara himpunan *dependency relation* yang dipakai adalah:

1. acl: *clausal modifier of noun (adjectival clause)*
2. advcl: *adverbial clause modifier*
3. advmod: *adverbial modifier*
4. amod: *adjectival modifier*
5. appos: *appositional modifier*
6. aux: *auxiliary*
7. case: *case marking*
8. cc: *coordinating conjunction*
9. ccomp: *clausal complement*
10. clf: *classifier*
11. compound: *compound*
12. conj: *conjunct*
13. cop: *copula*
14. csubj: *clausal subject*
15. dep: *unspecified dependency*
16. det: *determiner*
17. discourse: *discourse element*
18. dislocated: *dislocated elements*
19. expl: *expletive*

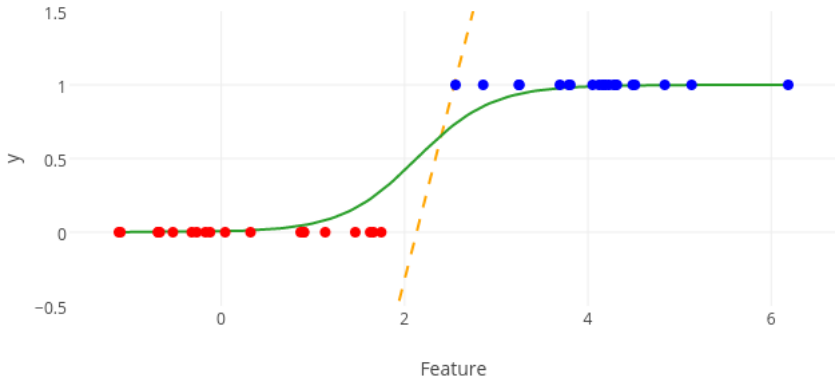
20. fixed: *fixed multiword expression*
21. flat: *flat multiword expression*
22. goeswith: *goes with*
23. iobj: *indirect object*
24. list: *list*
25. mark: *marker*
26. nmod: *nominal modifier*
27. nsubj: *nominal subject*
28. nummod: *numeric modifier*
29. obj: *object*
30. obl: *oblique nominal*
31. orphan: *orphan*
32. parataxis: *parataxis*
33. punct: *punctuation*
34. reparandum: *overridden disfluency*
35. root: *root*
36. vocative: *vocative*
37. xcomp: *open clausal complement*

2.4 *Supervised Learning*

Supervised learning adalah teknik *machine learning* yang mempelajari pola dari data yang telah diberi label atau dikelompokkan (Mohri et al., 2012). Metode *supervised learning* dapat dibagi menjadi dua, yaitu deskriptif (*descriptive learning*) dan generatif (*generative learning*). Pada *descriptive learning* mencari fungsi untuk memetakan data x ke label y atau probabilitas posterior (*posterior probability*) $p(y|x)$ (contoh: *logistic regression*, *support vector machine*, *multi-layer perceptron*, dsb.) sedangkan *generative learning* mencari probabilitas gabungan (*joint probability*) $p(x,y)$ lebih dulu sebelum menggunakan *Bayes Rules* untuk menghitung $p(y|x)$ (contoh: *naive bayes classifier*, *decision tree*, dsb.) (Ng and Jordan, 2002). Penelitian ini membandingkan empat buah model klasifikasi biner yang dihasilkan oleh metode-metode berikut:

2.4.1 *Logistic Regression*

Logistic regression adalah metode pemodelan deskriptif yang mencari fungsi hipotesis yang memetakan data x ke kelas y yang dapat dipisahkan fungsi logistik/*sigmoid* (2.1) sesuai kelasnya $\{0,1\}$ (Theodoridis, 2015) seperti visualisasi pada Gambar 2.5. Fungsi hipotesis dihasilkan dengan mencari bobot θ yang dapat meminimumkan *cost function* (2.2) menggunakan algoritma *gradient descent*.



Gambar 2.5: Contoh hasil pemetaan (titik merah dan biru) fungsi *logistic regression* dari fitur x ke kelas y yang dapat dipisahkan oleh fungsi logistik/*sigmoid* (garis hijau) (sumber: https://florianhartl.com)

$$\sigma(t) = \frac{1}{1+e^{-t}} \tag{2.1}$$

di mana t adalah fungsi hipotesis, $t = \theta^T x$

$$L(\theta) = -\sum_{n=1}^{N} (y_n \ln \sigma(t) + (1-y_n)\ln(1-\sigma(t))) \tag{2.2}$$

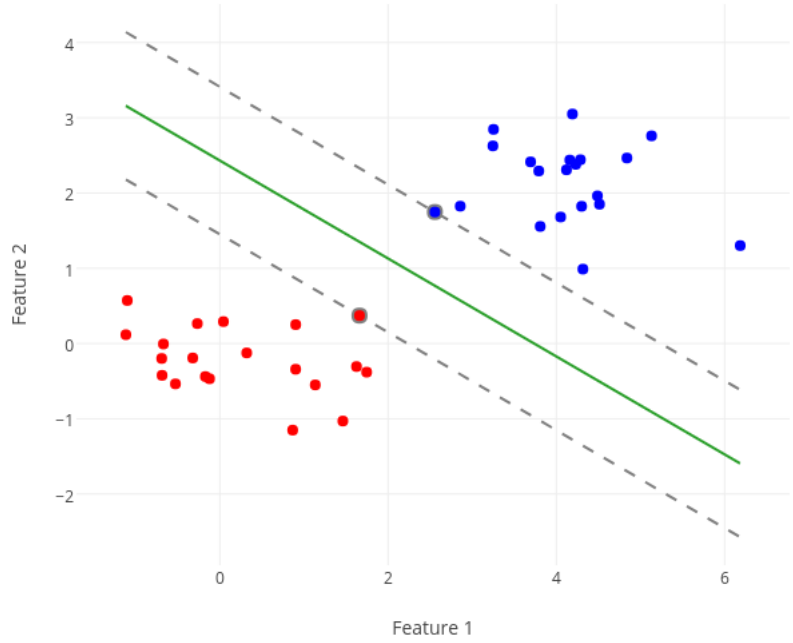
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} L(\theta) \tag{2.3}$$

$dimana, \theta =$ bobot

$\alpha = learning\ rate$

2.4.2 *Support Vector Machine*

Support vector machine (SVM) merupakan pemodelan yang mencari fungsi *hyperplane* yang memisahkan data sesuai kelasnya dengan menggunakan *decision boundary* yang memiliki jarak optimal dengan *hyperplane* (Theodoridis, 2015) seperti pada Gambar 2.6. Untuk memisahkan data yang tidak terpisahkan secara linier (*non-linearly separable*), dapat digunakan fungsi *kernel* untuk memetakan data sehingga bisa bisa dipisahkan secara linier. Salah satu fungsi *kernel* yang umum digunakan pada *task* NLP adalah *kernel* polinomial (2.4) (Joachims, 1998).



Gambar 2.6: Contoh fungsi linier (garis hijau) dari SVM yang memisahkan dua kelompok data dua dimensi (titik merah dan biru) menggunakan dua *support vector* (sumber: https://florianhartl.com)

$$K(x,y) = (x^{\mathsf{T}}y + c)^d \tag{2.4}$$

di mana, x = data atau fitur,

y = kelas atau label,

d = derajat polinomial,

c = konstanta

2.4.3 *Multi-Layer Perceptron*

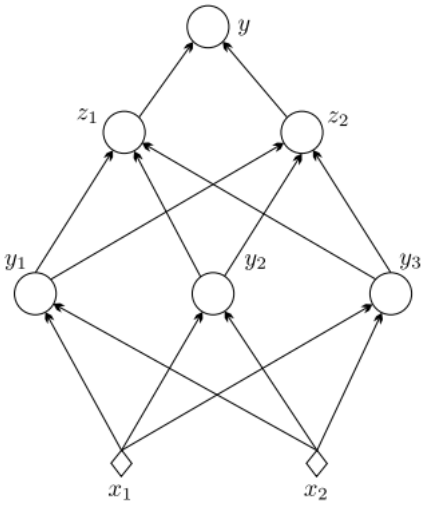
Multi-Layer Perceptron (MLP) atau *feed-forward neural network* adalah pemodelan klasifikasi nonlinier berbasiskan jaringan syaraf tiruan (*perceptron*) yang memiliki lebih dari satu *hidden layer* yang berisi sejumlah neuron (Theodoridis, 2015) seperti yang divisualisasikan pada Gambar 2.7. Nilai *output* dari suatu neuron ditentukan oleh *input* x, bobot (*weight*) w, *bias* b dan fungsi aktivasi f, $o(\vec{x}) = f(\vec{w} \cdot \vec{x} + \vec{b})$ (Mitchell, 1997). Contoh fungsi aktivasi yang bisa digunakan (Mitchell, 1997) adalah:

1. Fungsi *sign* : $f(x) = 1$ if $x > 0$ selain itu -1
2. Fungsi *sigmoid/logistic* : $f(x) = \frac{1}{1+e^{-x}}$

3. Fungsi *tanh*: $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$

4. Fungsi *rectifier*: $f(x) = \max(0,x)(NairandHinton,\ 2010)$

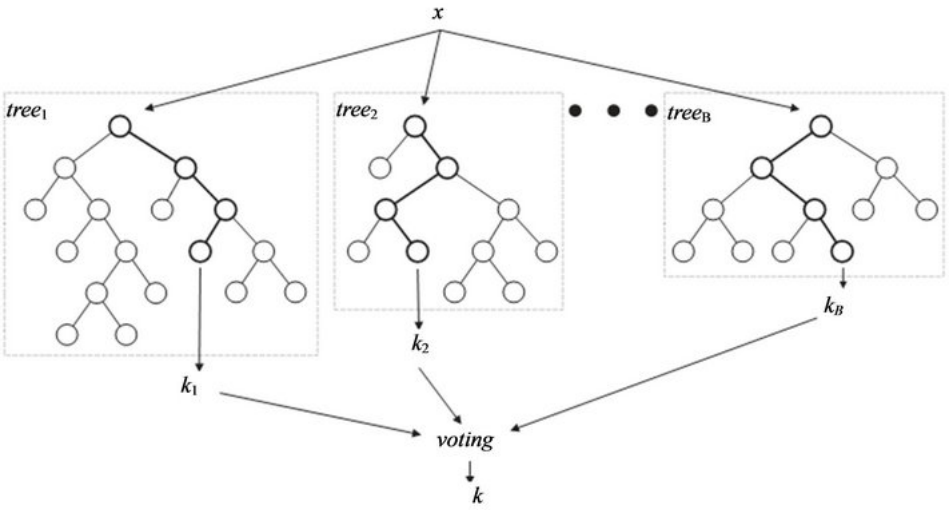
MLP dilatih dengan menyesuaikan bobot secara iteratif menggunakan algoritma *gradient descent* dan *backpropagation* (Theodoridis, 2015).



Gambar 2.7: Visualisasi MLP dengan *input layer* $\{x_1, x_2\}$, dua *hidden layer* $\{\{y_1, y_2, y_3\}, \{z_1, z_2\}\}$ dan satu *output layer* $\{y\}$ (sumber: Theodoridis (2015))

2.4.4 *Random Forest*

Random forest adalah metode *bagging* lebih dari satu varian *decision tree* (*forest*) dengan pemilihan fitur yang acak (*random*) (Breiman, 2001). *Bagging* sendiri adalah metode klasifikasi berdasarkan voting lebih dari satu varian *classifier* dengan tujuan meningkatkan kemampuan generalisasi (Breiman, 1996). Sedangkan *decision tree* adalah pemodelan klasifikasi generatif yang membangun serangkaian tes terhadap data/fitur untuk menolak kemungkinan kelas sampai hanya tersisa satu kelas (Theodoridis, 2015). Visualisi *random forest* ditunjukkan pada Gambar 2.8.



Gambar 2.8: Visualisasi *random forest* yang memprediksi kelas k untuk data x berdasarkan voting hasil klasifikasi setiap *tree* $\{k_1, k_2, .., k_b\}$ (sumber: http://wwww.scirp.org)

BAB 3
METODE PENELITIAN

Pada bab ini dijelaskan mengenai tahapan penelitian, seperti yang ditunjukkan pada Tabel 3.1, yang meliputi studi literatur, perancangan dan implementasi sistem, pengumpulan data serta evaluasi dan analisis.

Tabel 3.1: Tahapan penelitian

Tahapan	**Alat**	**Hasil**
Studi literatur	Mesin pencari buku dan jurnal elektronik	Latar belakang masalah, rumusan masalah, rangkuman penelitian terkait dan ide rancangan sistem
Perancangan dan pengimplentasian sistem	Java, Python, Git, editor kode	Sistem Open IE
Pengumpulan data	Python, Java	Dataset dan model *classifier*
Evaluasi dan analisis	Python	Tabel hasil, diagram hasil, kesimpulan dan saran

3.1 Studi Literatur

Pada tahap ini Penulis mengumpulkan dan menelaah dokumen ilmiah seperti *paper* dan artikel elektronik terkait *open IE* untuk memahami topik ini secara lebih mendalam dan mengetahui pencapaian penelitian-penelitian terkait. Pencarian dilakukan digunakan menggunakan mesin pencari[1][2] jurnal dan artikel ilmiah elektronik nasional dan internasional. Hasil penelaahan ini berupa latar belakang dan rumusan masalah yang dituangkan pada bab 1, rangkuman dan perbandingan sistem *open IE* pada bab 2, serta ide rancangan sistem *open IE* untuk bahasa Indonesia yang akan dijelaskan pada subbab berikutnya.

[1]Database Jurnal Universitas Indonesia http://remote-lib.ui.ac.id
[2]Google Scholar https://scholar.google.co.id/

3.2 Rancangan dan Implementasi Sistem

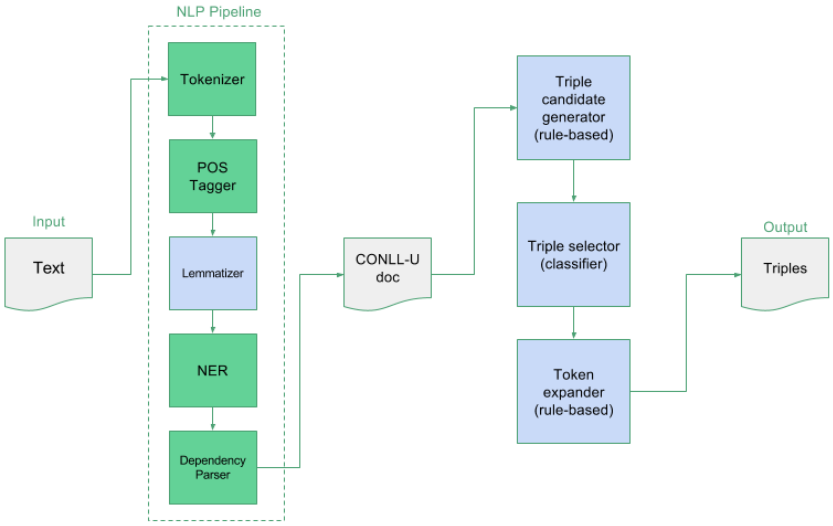
As shown in the flowchart Figure 3.1, our system is composed of four main components: **NLP pipeline**, **triple candidate generator**, **triple selector** and **token expander**. Each of them are explained further in following subsections.

Pada tahap ini Penulis merancang sistem *open IE* untuk bahasa Indonesia yang mengadaptasi beberapa teknik pada sistem *open IE* pada penelitian terkait. Rancangan sistem ini berisi empat modul utama, seperti yang ditunjukkan pada Gambar 3.1, yaitu **NLP pipeline**, **triple candidate generator**, **triple selector** dan **token expander**. Terdapat tiga fase atau langkah untuk melakukan ekstraksi *triple* menggunakan sistem ini:

1. Label (*label*): membangun data latih untuk untuk *triple selector* dengan menganotasi manual hasil ekstraksi dari *triple candidate generator*

2. Belajar (*learn*): melatih *classifier* untuk mengekstrak himpunan *triple* dari kalimat menggunakan data dari fase Label.

3. Ekstrak (*extract*): mengekstrak himpunan *triple* dari kalimat menggunakan *classifier* yang telah dilatih pada fase Belajar



Gambar 3.1: Indonesian open domain information extraction flowchart

3.2.1 NLP Pipeline

The NLP pipeline is a series of NLP tasks that annotates one or more sentences and saves them in CONLL-U[3] 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` (Manning et al., 2014), which mimics Penn Treebank 3 tokenizer[4]. 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[5]. 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:

   - Ability to process a sentence instead of a token
   - Usage of in-memory database to speed up dictionary lookup
   - Reimplementation in Java language and integration with Stanford Core NLP annotator API to improve reusability

   This lemmatizer yields **99%** accuracy when tested using dataset of 5,638

[3]CONLL-U format description http://universaldependencies.org/format.html
[4]Penn Treebank 3 https://catalog.ldc.upenn.edu/LDC99T42
[5]UD Indonesian dataset https://github.com/UniversalDependencies/UD_Indonesian

token-lemma pairs[6]. We use lemma as one of the features for NER classifier.

4. Named-Entity Recognizer (NER)

   Stanford NLP `CRFClassifier` (Finkel et al., 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[7].

5. Dependency Parser

   We relied on Stanford NLP `nndep.DependencyParser` (Chen and Manning, 2014), to annotate dependency relation of each token in 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 embedding[8] (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 3.2. The document becomes an input for next model, the triple candidate generator which is described in Section 3.2.2. 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.

[6]Indonesian Lemmatizer https://github.com/davidchristiandy/lemmatizer

[7]Indonesian NER https://github.com/yusufsyaifudin/indonesia-ner

[8]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	_	_

Gambar 3.2: Example of CONLL-U sentence annotation format

3.2.2 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 3.2 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 (Banko et al., 2007), we cannot use them directly as training data for triple selector/classifier.

For example, applying the rules to an annotated sentence in Figure 3.2 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) ✓
- (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 3.1).

3.2.3 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 3.2.2, 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[9] 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

[9]scikit-learn: machine learning in Python http://scikit-learn.org

Tabel 3.2: 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

- 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 3.2.2.

We extract 17 features described in Table 3.3 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 (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.

Tabel 3.3: 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

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.

3.2.4 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 et al., 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 3.4 and Table 3.5 to determine whether to: (1) expand a token to its dependent, (2) ig-

Tabel 3.4: 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

nore the dependent or (3) remove the token itself. For example, token expander will expand check-marked triples in Section 3.2.2 into:

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

Tabel 3.5: 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)*.

3.3 Pengumpulan Data

Kamus lemma https://github.com/davidchristiandy/lemmatizer (5,638 kata uji)

Dataset POS tagging https://github.com/UniversalDependencies/UD_Indonesian (5,036 kalimat latih + 559 kalimat uji)

Dataset NER Kelas NLP Fasilkom, UI, 2016 https://github.com/yohanesgultom/knowledge-extractor-id (1,700 kalimat latih + 426 kalimat uji, 3 kelas entitas: Person, Organization, Location) https://github.com/yusufsyaifudin/indonesia-ner (1,835 kalimat latih, 5 kelas entitas: Person, Organization, Location, Quantity, Time)

Dataset Dependency parsing https://github.com/UniversalDependencies/UD_Indonesian (5,036 kalimat latih + 559 kalimat uji)

Dataset triple selector, anotasi manual dari sebagian dataset dependency parsing

3.4 Evaluasi dan Analisis

Untuk mengevaluasi kinerja dari algoritma *supervised learning* dilakukan proses validasi silang (*cross validation*) menggunakan data yang sudah diketahui kelasnya.

BAB 4
HASIL DAN ANALISIS

Pada bab ini dijelaskan hasil evaluasi dan analisis dari penelitian ini.

4.1 Evaluasi

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 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 RegressionFan et al. (2008) (linear model)
2. Polynomial Support Vector Machine (SVM)Chang and Lin (2011) (nonlinear model)
3. Multi-Layer Perceptron (MLP)Hinton (1989) with 2 hidden layers (20 and 10 ReLUNair and Hinton (2010) neurons)
4. Random ForestWasserman (2015) (ensemble decision trees)

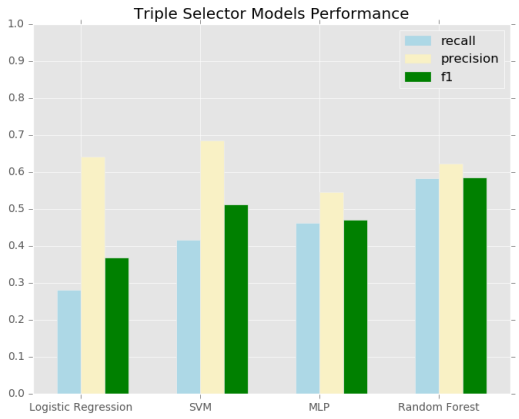
We use the manually annotated triple selector dataset described in Section **??** to cross-validateKohavi et al. (1995) (k-Fold with $k = 3$) the four classifiers. Since open IE systems requires both precision and recallAngeli et al. (2015), we choose F1 score to determine the best classifier for triple selector. The result of this experiment is shown by Figure 4.1 and Table 4.1 where Random Forest achieves the highest F1 score 0.58.

Tabel 4.1: 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**

Tabel 4.2: 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



Gambar 4.1: Triple selector models performance comparison chart

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

4.2 Analisis

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 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 **??** 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 TextRunnerBanko et al. (2007). Therefore, in contrast to the argument proposed in the related workBanko 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.

BAB 5
PENUTUP

Pada bab ini dijelaskan kesimpulan penelitian ini dan saran untuk pengembangan penelitian di masa depan.

5.1 Kesimpulan

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[1] to improve research reproducibility.

5.2 Saran

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.

[1]Paper source code https://github.com/yohanesgultom/id-openie

DAFTAR REFERENSI

Angeli, G., Premkumar, M. J., and Manning, C. D. (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, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735.

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the web. In *IJCAI*, volume 7, pages 2670–2676.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Cowie, J. and Lehnert, W. (1996). Information extraction. *Communications of the ACM*, 39(1):80–91.

Etzioni, O. (2011). Search needs a shake-up. *Nature*, 476(7358):25–26.

Etzioni, O., Fader, A., Christensen, J., Soderland, S., et al. (2011). Open information extraction: The second generation. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Exner, P. and Nugues, P. (2014). Refractive: An open source tool to extract knowledge from syntactic and semantic relations. In *LREC*, pages 2584–2589.

Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874.

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

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial intelligence*, 40(1-3):185–234.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142.

Jurafsky, D. (2000). *Speech & language processing*. Pearson Education India.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA.

MacCartney, B. and Manning, C. D. (2007). Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 193–200. Association for Computational Linguistics.

Manning, C., Grow, T., Grenager, T., Finkel, J., and Bauer, J. (2014). Ptbtokenizer.

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

McCallum, A. K. (2002). Mallet: A machine learning for language toolkit.

Mitchell, T. M. (1997). *Machine learning. 1997*, volume 45.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2:841–848.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016). Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666.

Schmitz, M., Bart, R., Soderland, S., Etzioni, O., et al. (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*, pages 523–534. Association for Computational Linguistics.

Singh, P., Lin, T., Mueller, E., Lim, G., Perkins, T., and Li Zhu, W. (2002). Open mind common sense: Knowledge acquisition from the general public. *On the move to meaningful internet systems 2002: CoopIS, DOA, and ODBASE*, pages 1223–1237.

Suhartono, D. (2014). Lemmatization technique in bahasa: Indonesian. *Journal of Software*, 9(5):1203.

Theodoridis, S. (2015). *Machine learning: a Bayesian and optimization perspective*. Academic Press.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (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*, pages 173–180. Association for Computational Linguistics.

Wasserman, D. (2015). Grid search optimization.

LAMPIRAN

LAMPIRAN 1: KODE SUMBER PROGRAM UTAMA

Kode sumber program utama (*main program*) `extract_triples.py`

```
import os
import csv
import argparse
import subprocess
import numpy as np
import json
from sys import platform
from sklearn.externals import joblib
from sklearn.preprocessing import StandardScaler
from tripletools import (
    vectorize,
    parse_connlu_file,
    extract_triples_by_combinations,
    get_best_features
)
from pprint import pprint

# choose script based on OS (windows or *nix)
DEPPARSE_SCRIPT = 'bin' + os.sep + 'id-openie'
if platform == 'win32':
    DEPPARSE_SCRIPT += '.bat'


def write_json(triples, y, out):
    count = 0
    grouped = {}
    for i in range(y.shape[0]):
        if y[i] == 1:
            triple = triples[i]
            if triple[1] not in grouped:
                grouped[triple[1]] = {}
            if triple[2] not in grouped[triple[1]]:
                grouped[triple[1]][triple[2]] = {}
            if triple[3] not in grouped[triple[1]][triple[2]]:
                grouped[triple[1]][triple[2]][triple[3]] = {}
            count += 1
    out.write(json.dumps(grouped) + '\n')
    return count


def write_tsv(triples, y, out):
    writer = csv.writer(out, delimiter='\t', quoting=csv.QUOTE_NONE, quotechar='')
    count = 0
    for i in range(y.shape[0]):
        if y[i] == 1:
            writer.writerow(triples[i])
            count += 1
    return count


def extract(conllu_file, classifier, out, format='tsv', scaler=None):
    X = []
    triples = []
    for index, s, s_header in parse_connlu_file(conllu_file):
        for first, second, third, subj, pred, obj in extract_triples_by_combinations(s, s_header):
            X.append(vectorize(first, second, third))
            triples.append((first['sentence_id'], subj, pred, obj))
    X = np.array(X, dtype='float32')
    # apply best features selection
    X = X[:, get_best_features()]
    # scale if scaler is available
    if scaler:
        X = scaler.transform(X)
    y = classifier.predict(X)
    # write output
    if format == 'tsv':
```

```
        return write_tsv(triples, y, out)
    else:  # format == 'json'
        return write_json(triples, y, out)


if __name__ == '__main__':

    if os.path.isfile(DEPPARSE_SCRIPT):
        parser = argparse.ArgumentParser(description='Extract triples from Indonesian text')
        parser.add_argument('input_file', help='Input file containing 1 (one) Indonesian sentence per line')
        parser.add_argument('-m', '--model_file', help='Triples classifier model file', default='triples-classifier-model.pkl')
        parser.add_argument('-s', '--scaler_file', help='Triples classifier scaler file', default='triples-classifier-scaler.pkl')
        parser.add_argument('-o', '--output_file', help='Output file containing triples')
        parser.add_argument('-f', '--output_format', help='Output file format', choices=['json', 'tsv'], default='json')
        args = parser.parse_args()
        args.output_file = args.output_file if args.output_file else 'triples.' + args.output_format

        # dependency parsing
        print('Parsing dependency tree..')
        depparse_output = os.path.basename(args.input_file) + '.conllu'
        subprocess.call([DEPPARSE_SCRIPT, '-f', args.input_file])

        # extract triples
        classifier = joblib.load(args.model_file)
        scaler = joblib.load(args.scaler_file)
        with open(args.output_file, 'wb') as out:
            count = extract(depparse_output, classifier, out, args.output_format, scaler=scaler)

        print('{} triple(s) extracted'.format(count))
        print('Triples saved in ' + args.output_file)
    else:
        print('File not found: ' + DEPPARSE_SCRIPT)
```

LAMPIRAN 2: KODE SUMBER *NLP PIPELINE*

Kode sumber utama *NLP pipeline*: `DepdendencyParser.java`

```java
package id.nlp.depparser;

import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.pipeline.*;
import edu.stanford.nlp.semgraph.SemanticGraph;
import edu.stanford.nlp.semgraph.SemanticGraphCoreAnnotations;
import edu.stanford.nlp.trees.ud.CoNLLUDocumentWriter;
import edu.stanford.nlp.trees.ud.ExtendedCoNLLUDocumentWriter;
import edu.stanford.nlp.util.CoreMap;
import edu.stanford.nlp.util.PropertiesUtils;
import net.sourceforge.argparse4j.ArgumentParsers;
import net.sourceforge.argparse4j.inf.ArgumentParser;
import net.sourceforge.argparse4j.inf.ArgumentParserException;
import net.sourceforge.argparse4j.inf.Namespace;

import java.io.File;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import static edu.stanford.nlp.pipeline.Annotator.*;

public class DependencyParser {

    static final String TAGGER_MODEL = "tagger-id.universal.model";
    static final String NER_MODEL = "ner-id.model.ser.gz";
    static final String PARSER_MODEL = "parser-id.conllu.model.gz";
    static final int NUM_THREADS = 1;
    static final String OUTPUT_FORMAT = "conllu";

    AnnotatorPool annotatorPool;
    Properties props;
    StanfordCoreNLP pipeline;

    public DependencyParser() throws SQLException, IOException, ClassNotFoundException {
        this(TAGGER_MODEL, NER_MODEL, PARSER_MODEL, NUM_THREADS);
    }

    public DependencyParser(
            String taggerModel,
            String nerModel,
            String parserModel,
            int numThreads
    ) throws SQLException, IOException, ClassNotFoundException {

        // Create the Stanford CoreNLP pipeline
        this.props = PropertiesUtils.asProperties(
                "annotators", "tokenize,ssplit,pos,lemma,ner,depparse",
                "ner.model", nerModel,
                "ner.useSUTime", "false",
                "pos.model", taggerModel,
                "depparse.model", parserModel,
                "splitter.nomodel", "true",
                "ignore_affinity", "true",
                "outputFormat", OUTPUT_FORMAT,
                "threads", String.valueOf(numThreads)
        );

        // Create annotator pools
        this.annotatorPool = new AnnotatorPool();
        AnnotatorImplementations annotatorImplementations = new IndonesianAnnotatorImplementations();
        annotatorPool.register(STANFORD_TOKENIZE, AnnotatorFactories.tokenize(props, annotatorImplementations)
    );
```

```
65          annotatorPool.register(STANFORD_SSPLIT, AnnotatorFactories.sentenceSplit(props,
        annotatorImplementations));
66          annotatorPool.register(STANFORD_POS, AnnotatorFactories.posTag(props, annotatorImplementations));
67          annotatorPool.register(STANFORD_LEMMA, AnnotatorFactories.lemma(props, annotatorImplementations));
68          annotatorPool.register(STANFORD_NER, AnnotatorFactories.nerTag(props, annotatorImplementations));
69          annotatorPool.register(STANFORD_DEPENDENCIES, AnnotatorFactories.dependencies(props,
        annotatorImplementations));
70
71          // Create pipeline
72          this.pipeline = new IndonesianStanfordCoreNLP(this.props, annotatorPool);
73      }
74
75      /**
76       * Parse text
77       * @param text
78       * @return
79       */
80      public String parse(String text) {
81          StringBuilder result = new StringBuilder();
82          Annotation doc = pipeline.process(text);
83          List<CoreMap> sentences = doc.get(CoreAnnotations.SentencesAnnotation.class);
84          CoNLLUDocumentWriter conllUWriter = new ExtendedCoNLLUDocumentWriter();
85          for (CoreMap sentence : sentences) {
86              SemanticGraph sg = sentence.get(SemanticGraphCoreAnnotations.BasicDependenciesAnnotation.class);
87              if (sg != null) {
88                  result.append(conllUWriter.printSemanticGraph(sg)).append("\n");
89              }
90          }
91          return result.toString();
92      }
93
94      /**
95       * Parse input file(s)
96       * @param inputFiles
97       * @param outputDir
98       * @throws IOException
99       */
100     public void parse(List<File> inputFiles, String outputDir) throws IOException, SQLException,
        ClassNotFoundException {
101         // override existing pipeline
102         if (!props.containsKey("outputDirectory")) {
103             props.setProperty("outputDirectory", outputDir);
104             this.pipeline = new IndonesianStanfordCoreNLP(this.props, this.annotatorPool);
105         }
106         pipeline.processFiles(inputFiles);
107     }
108
109     public static void main(String args[]) {
110
111         // parse arguments
112         ArgumentParser parser = ArgumentParsers.newArgumentParser("DependencyParser").defaultHelp(true).
        description("Generate CONLL-U dependency tree from Indonesian text");
113         parser.addArgument("-t", "--text").help("Text input to parse");
114         parser.addArgument("-f", "--file").nargs("*").help("File input to parse");
115         parser.addArgument("-o", "--outputDir").setDefault(".").help("Output directory");
116
117         Namespace ns = null;
118         try {
119             ns = parser.parseArgs(args);
120         } catch (ArgumentParserException e) {
121             parser.handleError(e);
122             System.exit(1);
123         }
124
125         String text = ns.getString("text");
126         List<String> files = ns.<String> getList("file");
127         String outputDir = ns.getString("outputDir");
128         try {
129             if (text != null) {
130                 System.out.println(new DependencyParser().parse(text.trim()));
131             } else if (files != null) {
132                 List<File> fileList = new ArrayList<>();
133                 List<String> outputFiles = new ArrayList<>();
134                 String sep = System.getProperty("file.separator");
135                 for (String file:files) {
136                     File fileObj = new File(file);
137                     fileList.add(fileObj);
```

```
138                     outputFiles.add(outputDir + sep + fileObj.getName() + "." + OUTPUT_FORMAT);
139                 }
140                 new DependencyParser().parse(fileList, outputDir);
141                 System.out.println("File(s) created:");
142                 for (String outputFile:outputFiles) {
143                     System.out.println(outputFile);
144                 }
145             } else {
146                 System.err.println("No input provided");
147             }
148         } catch (Exception e) {
149             e.printStackTrace();
150         }
151     }
152 }
```

LAMPIRAN 3: KODE SUMBER PUSTAKA UTAMA

Kode sumber pustaka utama (*main library*) yang berisi kode sumber untuk *Triple Candidates Generator*, *Token Expander* dan *Triple Selector* `tripletools.py`

```
import itertools
import csv
import argparse


BEST_FEATURES = [0, 1, 2, 3, 5, 6, 10, 11, 12, 14, 17, 18, 19, 20, 21, 22, 23]  # F1 0.586
# BEST_FEATURES = [0, 1, 2, 3, 5, 6, 10, 11, 12, 17, 18, 19, 20, 21, 22, 23]   # F1 0.579
# BEST_FEATURES = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
    25, 26]  # F1 0.547


# constants
conllu = ['ID', 'FORM', 'LEMMA', 'UPOSTAG', 'XPOSTAG', 'FEATS', 'HEAD', 'DEPREL', 'DEPS', 'MISC']
postag = ['', 'ADJ', 'ADP', 'ADV', 'AUX', 'CCONJ', 'DET', 'INTJ', 'NOUN', 'NUM', 'PART', 'PRON', 'PROPN', '
    PUNCT', 'SCONJ', 'SYM', 'VERB', 'X', 'CONJ']
deprel = ['', 'acl', 'advcl', 'advmod', 'amod', 'appos', 'aux', 'case', 'cc', 'ccomp', 'clf', 'compound', '
    conj', 'cop', 'csubj', 'dep', 'det', 'discourse', 'dislocated', 'expl', 'fixed', 'flat', 'goeswith', '
    iobj', 'list', 'mark', 'nmod', 'nsubj', 'nummod', 'obj', 'obl', 'orphan', 'parataxis', 'punct', '
    reparandum', 'root', 'vocative', 'xcomp', 'nsubjpass', 'name', 'dobj', 'neg', 'mwe', 'csubjpass']
entity = ['', 'PERSON', 'LOCATION', 'ORGANIZATION', 'TIME', 'QUANTITY', 'OTHER']

# extraction RULES
subject_object_candidates_pos = ['PROPN', 'NOUN', 'PRON', 'VERB']
predicate_candidates_pos = ['VERB', 'AUX']
non_subject_object_candidates_form = ['yang', 'adalah']
non_predicate_candidates_form = ['yang']
num_siblings = 1  # bigram


def extract_triples_by_root_children(conllu_s, header):
    """
    Extract features (triples) for clustering from sentence (conllu_s)
    by combining sentence root/header with 2 of its children
    """
    # find all direct branches of header
    direct_branches = []
    for id, row in conllu_s.iteritems():
        # children (direct branches of header)
        if (
            row['head'] == header['id'] and
            row['upostag'] in subject_object_candidates_pos and
            row['form'] not in non_subject_object_candidates_form
        ):
            direct_branches.append(id)

    # yield triples combinations
    if len(direct_branches) > 1:
        for combi in itertools.combinations(direct_branches, 2):
            first = None
            third = None
            if combi[0] < header['id'] and header['id'] < combi[1]:
                first = conllu_s[combi[0]]
                third = conllu_s[combi[1]]
            elif combi[1] < header['id'] and header['id'] < combi[0]:
                first = conllu_s[combi[1]]
                third = conllu_s[combi[0]]

            if first and third:
                second = conllu_s[header['id']]
                yield (first, second, third)


def extract_triples_by_combinations(conllu_s, header):
    """
```

```
    Extract features (triples) for clustering from sentence (conllu_s)
    by enumerating all possible triple combination of word
    """
    num_tokens = len(conllu_s)
    # sentence start from 1
    for i in range(1, num_tokens - 2):
        first = conllu_s[i]
        # RULES for Subject
        if (
            first['upostag'] in subject_object_candidates_pos and
            first['form'] not in non_subject_object_candidates_form and
            (first['deprel'] not in ['compound', 'name'] or first['head_distance'] > 2)
        ):
            for j in range(i + 1, num_tokens - 1):
                second = conllu_s[j]
                # RULES for Predicate
                if (
                    second['upostag'] in predicate_candidates_pos
                ):
                    for k in range(j + 1, num_tokens):
                        third = conllu_s[k]
                        # RULES for Object
                        if (
                            third['upostag'] in subject_object_candidates_pos and
                            third['form'] not in non_subject_object_candidates_form and
                            (third['deprel'] not in ['compound', 'name'] or third['head_distance'] > 2) and
                            (third['upostag'] not in predicate_candidates_pos or first['upostag'] not in
     predicate_candidates_pos)
                        ):
                            s = first['flatten_s']
                            p = second['flatten_p']
                            if third['nearest_adp_id']:
                                p += ' ' + conllu_s[third['nearest_adp_id']]['form']
                            o = third['flatten_o']
                            yield (first, second, third, s, p, o)


def extract_triples_by_children_combination(conllu_s, header):
    """
    Extract features (triples) for clustering from sentence (conllu_s)
    by combining sentence predicate nodes with 2 of their children
    """
    for k, v in conllu_s.items():
        # RULES for Subject, Predicate and Object
        if (
            v['upostag'] in predicate_candidates_pos and
            v['form'] not in non_predicate_candidates_form
        ):
            for first, second, third in extract_triples_by_root_children(conllu_s, v):
                yield (first, second, third)


def trace_children_pos(child_pos_list, parent_pos, node, s):
    """
    Find parent that has parent_pos pos tag and has one child of child_pos
    """
    parent_pos_list = [parent_pos] if parent_pos not in ['NOUN', 'PROPN'] else ['NOUN', 'PROPN']
    if node['deprel'] == 'root' or node['upostag'] not in parent_pos_list:
        return None
    else:
        # find child with upostag == child_pos
        for child_id in node['children']:
            if s[child_id]['upostag'] in child_pos_list:
                return s[child_id]

        # if not found try to search on node's parent
        return trace_children_pos(child_pos_list, parent_pos, s[node['head']], s)


def remove_token_if_first(field, values, tokens):
    while (tokens and tokens[0][1][field] in values):
        tokens.pop(0)


def remove_token_if_last(field, values, tokens):
    while (tokens and tokens[-1][1][field] in values):
        tokens.pop(-1)
```

```
136
137
138 def remove_token_if_first_or_last(field, values, tokens):
139     remove_token_if_first(field, values, tokens)
140     remove_token_if_last(field, values, tokens)
141
142
143 def expand_node(node, s):
144     """
145     Expand node to its children as dict
146     """
147     expanded = {node['id']: node}
148     has_quote = False
149     # EXPAND RULES
150
151     for k in node['children']:
152         v = s[k]
153         if v['deprel'] in ['compound', 'name', 'amod']:
154             expanded.update(expand_node(v, s))
155         elif v['entity'] and v['entity'] == node['entity'] and abs(v['id'] - node['id']) == 1:
156             expanded.update(expand_node(v, s))
157         elif has_quote:
158             expanded.update(expand_node(v, s))
159         elif node['deprel'] == 'root':     # [Sembungan adalah sebuah] (desa) [.]
160             continue
161         else:
162             if v['form'] in ['\'', '"']:  # (" Lelaki dan Telaga ")
163                 has_quote = True
164             if (v['upostag'] in ['CONJ'] or v['form'] in [',', '/']):  # (kecamatan) Kejajar [, kabupaten
      Wonosobo]
165                 break
166             if v['upostag'] in ['VERB', 'ADP']:  # (helm) Brodie [yang dipakai]
167                 continue
168             if v['children'] and 'ADP' in [s[i]['upostag'] for i in v['children']]:  # (Stahlhelm) Jerman [
      dengan perbaikan desain], [Beberapa bulan sebelum] (Rose)
169                 continue
170             expanded.update(expand_node(v, s))
171
172     return expanded
173
174
175 def flatten_node(node, s, expand_as='o', mark_head=False):
176     """
177     Expand node and its branches to clause string
178     """
179     if expand_as.lower() in ['s', 'o']:
180         expanded = expand_node(node, s)
181         sorted_nodes = sorted(expanded.items())
182
183         # EXPAND RULES
184         remove_token_if_first_or_last('upostag', ['CONJ', 'ADP'], sorted_nodes)
185         remove_token_if_first('form', [')'], sorted_nodes)
186         remove_token_if_last('form', ['(', 'yang'], sorted_nodes)
187
188         text = ' '.join([v['form'] if not mark_head or k != node['id'] else '({})'.format(v['form']) for k, v
      in sorted_nodes])
189         ids = [k for k, v in sorted_nodes]
190     elif expand_as.lower() in ['p']:
191         text = node['form'] if not mark_head else '({})'.format(node['form'])
192         ids = [node['id']]
193
194         # EXPAND RULES
195         negation_node = [s[c_id] for c_id in node['children'] if s[c_id]['form'].lower() == 'tidak']
196         if negation_node:
197             text = negation_node[0]['form'] + ' ' + text
198             ids = [negation_node[0]['id']] + ids
199
200     return text, ids
201
202
203 def flatten_conllu_sentence(conllu_s):
204     return ' '.join([token['form'] for token in conllu_s.values()])
205
206
207 def set_extra_properties(s, children, mark_head=False):
208     """
209     Retrieve head's pos tag
```

```
    Flatten subject/object candidates
    """
    for k, v in s.iteritems():
        # get head pos tag
        s[k]['head_upostag'] = s[v['head']]['upostag'] if v['head'] > 0 else ''
        # get siblings pos tags
        before = v['id'] - num_siblings
        s[k]['before_upostag'] = [s[i]['upostag'] if i > 0 else '' for i in range(before, v['id'])]
        after = v['id'] + num_siblings + 1
        s[k]['before_upostag'] = [s[i]['upostag'] if i < len(s) else '' for i in range(after - num_siblings,
     after)]
        # get children id
        if k in children:
            sorted_children = sorted(children[k])
            s[k]['children'] = sorted_children

    # loop once more to flatten as children is required
    for k, v in s.iteritems():
        if v['upostag'] in subject_object_candidates_pos:
            s[k]['flatten_s'], s[k]['flatten_s_id'] = flatten_node(s[k], s, expand_as='s', mark_head=mark_head
     )
            s[k]['flatten_o'], s[k]['flatten_o_id'] = flatten_node(s[k], s, expand_as='o', mark_head=mark_head
     )
            # trace ADP node to parents to be inherited
            if v['head'] > 0:
                nearest_adp_node = trace_children_pos(['ADP'], v['upostag'], v, s)
                if nearest_adp_node:
                    s[k]['nearest_adp_id'] = nearest_adp_node['id']
        if v['upostag'] == 'VERB':
            s[k]['flatten_p'], s[k]['flatten_p_id'] = flatten_node(s[k], s, expand_as='p', mark_head=mark_head
     )


def get_neigbour_upostag(position, token):
    key = position + '_upostag'
    if position not in ['before', 'after'] or not token[key]:
        return postag.index('')
    return postag.index(token[key][0])


def get_next_upostag(token):
    return get_neigbour_upostag('after', token)


def get_prev_upostag(token):
    return get_neigbour_upostag('before', token)


def vectorize(first, second, third):
    """
    Convert a triple's member to feature vector
    """
    distance_first_second = abs(first['id'] - second['id'])
    distance_second_third = abs(second['id'] - third['id'])
    first_is_child_of_second = 1 if first['id'] in second['children'] else 0
    third_is_child_of_second = 1 if third['id'] in second['children'] else 0

    vector = []
    vector.append(postag.index(first['upostag']))
    vector.append(deprel.index(first['deprel']))
    vector.append(postag.index(first['head_upostag']))
    vector.append(entity.index(first['entity']))
    vector.append(len(first['children']))
    vector.append(distance_first_second)
    vector.append(first_is_child_of_second)
    vector.append(get_prev_upostag(first))
    vector.append(get_next_upostag(first))
    vector.append(1 if first['nearest_adp_id'] else 0)

    vector.append(postag.index(second['upostag']))
    vector.append(deprel.index(second['deprel']))
    vector.append(postag.index(second['head_upostag']))
    vector.append(entity.index(second['entity']))
    vector.append(len(second['children']))
    vector.append(get_prev_upostag(second))
    vector.append(get_next_upostag(second))

```

```
    vector.append(postag.index(third['upostag']))
    vector.append(deprel.index(third['deprel']))
    vector.append(postag.index(third['head_upostag']))
    vector.append(entity.index(third['entity']))
    vector.append(len(third['children']))
    vector.append(distance_second_third)
    vector.append(third_is_child_of_second)
    vector.append(get_prev_upostag(third))
    vector.append(get_next_upostag(third))
    vector.append(1 if third['nearest_adp_id'] else 0)

    return vector


def parse_connlu_file(conllu_file, mark_head=False):
    with open(conllu_file, 'rb') as csvfile:
        reader = csv.reader(csvfile, delimiter='\t', quoting=csv.QUOTE_NONE)
        s = {}
        children = {}
        s_header = None
        index = 0
        for row in reader:
            if len(row) > 0:
                id = int(row[conllu.index('ID')])
                head_id = int(row[conllu.index('HEAD')])
                deprel = row[conllu.index('DEPREL')].split(':')[0]  # ignore sub relation
                obj = {
                    'id': id,
                    'sentence_id': index,
                    'form': row[conllu.index('FORM')],
                    'upostag': row[conllu.index('UPOSTAG')],
                    'head': head_id,
                    'head_distance': abs(head_id - id) if head_id > 0 else 0,
                    'deprel': deprel if deprel != '_' else 'root',
                    'head_upostag': '',
                    'before_upostag': [],
                    'after_upostag': [],
                    'flatten_s': row[conllu.index('FORM')],
                    'flatten_p': row[conllu.index('FORM')],
                    'flatten_o': row[conllu.index('FORM')],
                    'flatten_s_id': [id],
                    'flatten_p_id': [id],
                    'flatten_o_id': [id],
                    'entity': row[conllu.index('MISC')] if row[conllu.index('MISC')] != '_' else '',
                    'children': [],
                    'nearest_adp_id': None
                }
                s[id] = obj
                # map children
                if obj['head'] != 0:
                    if obj['head'] not in children:
                        children[obj['head']] = []
                    if id not in children[obj['head']]:
                        children[obj['head']].append(id)
                # find root header
                s_header = obj if obj['head'] == 0 else s_header
            else:
                set_extra_properties(s, children, mark_head)
                yield index, s, s_header
                s = {}
                index += 1
                children = {}
        if s:
            # if last element not a blank
            set_extra_properties(s, children, mark_head)
            yield index, s, s_header


def get_best_features():
    return BEST_FEATURES
```

LAMPIRAN 4: KODE SUMBER PELATIHAN *TRIPLE SELECTOR*

Kode sumber pelatihan dan perbandingan *Triple Selector* `classifier.py`

```
import argparse
import collections
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.externals import joblib
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_recall_fscore_support
from tripletools import get_best_features
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches


plt.style.use('ggplot')

experiments = [
    {
        'name': 'Logistic Regression',
        'model': LogisticRegression(),
        'params': [
            {
                'solver': ['liblinear'],
                'penalty': ['l2'],
                'random_state': [77]
            },
        ]
    },
    {
        'name': 'SVM',
        'model': SVC(),
        'params': [
            {
                'kernel': ['poly'],
                'degree': [5],
                'random_state': [77]
            },
        ]
    },
    {
        'name': 'MLP',
        'model': MLPClassifier(max_iter=1000),
        'params': [
            {
                'hidden_layer_sizes': [(20, 10)],
                'random_state': [77]
            }
        ]
    },
    {
        'name': 'Random Forest',
        'model': RandomForestClassifier(),
        'params': [
            {
                'max_depth': [8],
                'n_estimators': [20],
                'min_samples_split': [5],
                'criterion': ['gini'],
                'max_features': ['auto'],
                'class_weight': ['balanced'],
```

```
                'random_state': [77]
            }
        ]
    },
]


def cross_validate_precision_recall_fbeta(model, X, y, cv=None):
    precision = cross_val_score(model, X, y, cv=cv, scoring='precision').mean()
    recall = cross_val_score(model, X, y, cv=cv, scoring='recall').mean()
    fbeta_list = cross_val_score(model, X, y, cv=cv, scoring='f1')
    fbeta = fbeta_list.mean()
    fbeta_min = fbeta_list.min()
    fbeta_max = fbeta_list.max()
    fbeta_std = fbeta_list.std()
    return precision, recall, fbeta, fbeta_min, fbeta_max, fbeta_std


def plot_model_performance_comparison(experiments):
    fig, ax = plt.subplots()

    # Example data
    x_data = []
    y_dict = {
        'precision': {'color': '#f9f1c5', 'data': []},
        'recall': {'color': 'lightblue', 'data': []},
        'f1': {'color': 'green', 'data': []},
    }
    for exp in experiments:
        x_data.append(exp['name'])
        y_dict['precision']['data'].append(exp['best_score']['precision'])
        y_dict['recall']['data'].append(exp['best_score']['recall'])
        y_dict['f1']['data'].append(exp['best_score']['f1'])

    x = np.arange(len(x_data))
    width = 0.20
    i = 1
    legend_handles = []
    for label, y in y_dict.items():
        ax.bar(x + width * i, y['data'], width, color=y['color'])
        legend_handles.append(mpatches.Patch(color=y['color'], label=label))
        i += 1
    ax.set_xticks(x + width * 2)
    ax.set_xticklabels(x_data)
    ax.set_yticks(np.arange(0.0, 1.1, 0.1))
    ax.set_title('Triple Selector Models Performance')

    lgd = plt.legend(handles=legend_handles)
    plt.show()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Train triples classifier')
    parser.add_argument('dataset_path', help='Dataset path')
    parser.add_argument('-o', '--output_path', help='Output model path', default='triples-classifier-model.pkl
      ')
    parser.add_argument('-s', '--scaler_output_path', help='Output scaler path', default='triples-classifier-
      scaler.pkl')
    parser.add_argument('-b', '--best', help='search parameters that gives best model', action='store_true')
    parser.add_argument('--nocv', help='no cross-validation. training accuracy only', action='store_true')
    args = parser.parse_args()

    # load dataset
    dataset = np.genfromtxt(args.dataset_path, delimiter=',', dtype='float32')
    total_features = dataset.shape[1] - 1

    # feature selection
    selected_features = get_best_features()
    print('Total features: {}'.format(total_features))
    print('Selected features: {} ({})'.format(selected_features, len(selected_features)))

    X = dataset[:, selected_features]
    y = dataset[:, -1]
    scaler = StandardScaler().fit(X)
    X = scaler.transform(X)
    joblib.dump(scaler, args.scaler_output_path)

    # collect dataset statistics
```

```
    counter = collections.Counter(y)
    print(counter)
    pos = counter[1] * 1.0 / (counter[0] + counter[1])
    neg = 1.0 - pos

    # exhaustive best parameters search
    cv = None
    print('')
    if args.best:
        best_score = 0.0
        best_model = None
        count = 0
        for experiment in experiments:
            search = GridSearchCV(
                estimator=experiment['model'],
                param_grid=experiment['params'],
                scoring='f1',
                cv=cv
            )
            search.fit(X, y)
            if args.nocv:
                y_pred = search.best_estimator_.predict(X)
                precision, recall, fbeta, support = precision_recall_fscore_support(y, y_pred, average='binary
    ')
                fbeta_min = fbeta_max = fbeta_std = fbeta
            else:
                precision, recall, fbeta, fbeta_min, fbeta_max, fbeta_std =
    cross_validate_precision_recall_fbeta(search.best_estimator_, X, y)
            print(search.best_estimator_)
            print('Precision: {}\nRecall: {}\nF1 avg: {}\nF1 min: {}\nF1 max: {}\nF1 std: {}\n'.format(
                precision,
                recall,
                fbeta,
                fbeta_min,
                fbeta_max,
                fbeta_std,
            ))
            experiment['best_model'] = best_model
            experiment['best_score'] = {'precision': precision, 'recall': recall, 'f1': fbeta}
            # replace current best model if the score is higher
            if search.best_score_ > best_score:
                best_score = search.best_score_
                best_model = search.best_estimator_
            count += 1
        print('--------------- Result ----------------')
        print('Best models: {} (F1 = {})'.format(best_score, type(best_model).__name__))
        model = best_model

        # show plot
        plot_model_performance_comparison(experiments)

    else:
        model = RandomForestClassifier(max_depth=8, class_weight='balanced', n_estimators=20,
    min_samples_split=5, max_features='auto', random_state=77)

        # cross validate best model to compare score
        precision, recall, fbeta, fbeta_min, fbeta_max, fbeta_std = cross_validate_precision_recall_fbeta(
    model, X, y)
        print('Precision: {}\nRecall: {}\nF1 avg: {}\nF1 min: {}\nF1 max: {}\nF1 std: {}\n'.format(
            precision,
            recall,
            fbeta,
            fbeta_min,
            fbeta_max,
            fbeta_std,
        ))

    # save model to file
    joblib.dump(model, args.output_path)
    print('Model saved to {}'.format(args.output_path))
    print('Scaler saved to {}'.format(args.scaler_output_path))
```