



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 ekstraksi informasi (*information extraction*) otomatis dari teks atau dokumen digital dari berbagai domain (*open domain*). Sayangnya, setiap sistem *open domain information extraction* (*open IE*) yang ada saat ini hanya berlaku untuk satu bahasa tertentu saja dan belum ada sistem *open IE* untuk bahasa Indonesia yang dipublikasikan. Pada penelitian ini Penulis memperkenalkan 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 *supervised 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* dengan 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 kualitas *dataset* yang masih rendah.

Kata Kunci:

information extraction, open domain, natural language processing, supervised learning, 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 supervised-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 low quality of dataset.

Keywords:

information extraction, open domain, natural language processing, supervised learning, Indonesian language

DAFTAR ISI

HALAMAN JUDUL	i
ABSTRAK	ii
Daftar Isi	iv
Daftar Gambar	vi
Daftar Tabel	vii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Permasalahan	2
1.2.1 Definisi Permasalahan	2
1.2.2 Batasan Permasalahan	2
1.3 Tujuan dan Manfaat	3
1.4 Sistematika Penulisan	4
2 TINJAUAN PUSTAKA	5
2.1 Penelitian Terkait	5
2.2 <i>Open Domain Information Extraction</i>	7
2.3 <i>Natural Language Processing</i>	8
2.3.1 <i>Tokenization</i>	9
2.3.2 <i>Part of Speech Tagging</i>	9
2.3.3 <i>Lemmatization</i>	10
2.3.4 <i>Named-Entity Recognition</i>	10
2.3.5 <i>Dependency Parsing</i>	10
2.3.6 <i>CoNLL-U</i>	11
2.4 <i>Supervised Learning</i>	11
2.4.1 <i>Logistic Regression</i>	12
2.4.2 <i>Support Vector Machine</i>	13
2.4.3 <i>Multi-Layer Perceptron</i>	14
2.4.4 <i>Random Forest</i>	15

3	METODE PENELITIAN	16
3.1	Studi Literatur	16
3.2	Rancangan dan Implementasi Sistem	16
3.2.1	NLP Pipeline	17
3.2.2	<i>Triple Candidate Generator</i>	20
3.2.3	<i>Triple Selector</i>	22
3.2.4	<i>Token Expander</i>	24
3.3	Evaluasi dan Analisis	26
3.3.1	Evaluasi	26
3.3.1.1	Eksperimen Model <i>Triple Selector</i>	26
3.3.1.2	Eksperimen Waktu Ekstraksi <i>Triple</i>	27
3.3.2	Analisis	27
4	HASIL DAN ANALISIS	28
4.1	Evaluasi	28
4.2	Analisis	30
5	PENUTUP	34
5.1	Kesimpulan	34
5.2	Saran	35
	Daftar Referensi	36
	LAMPIRAN	1
	Lampiran 1: Kode Sumber Program Utama	2
	Lampiran 2: Kode Sumber <i>NLP Pipeline</i>	3
	Lampiran 3: Kode Sumber Pustaka Utama	6
	Lampiran 4: Kode Sumber Pelatihan <i>Triple selector</i>	11
	Lampiran 5: Daftar <i>POS Tag</i> dan <i>Dependency Relation</i> CoNLL-U	14

DAFTAR GAMBAR

1.1	Contoh input dan output yang diharapkan dari sistem open IE untuk bahasa Indonesia	2
2.1	Proses pelatihan dan ekstraksi ARGLEARNER	6
2.2	Proses <i>labeling</i> dan ekstraksi pada OLLIE	7
2.3	Contoh <i>input</i> dan <i>output POS tagging</i>	9
2.4	Contoh <i>input</i> dan <i>output NER</i>	10
2.5	Contoh hasil pemetaan (titik merah dan biru) fungsi <i>logistic regression</i> dari fitur x ke kelas y yang dapat dipisahkan oleh fungsi logistik/ <i>sigmoid</i> (garis hijau) (sumber: https://florianhartl.com)	12
2.6	Contoh fungsi linier (garis hijau) dari SVM yang memisahkan dua kelompok data dua dimensi (titik merah dan biru) menggunakan dua <i>support vector</i> (sumber: https://florianhartl.com)	13
2.7	Visualisasi MLP dengan <i>input layer</i> $\{x_1, x_2\}$, dua <i>hidden layer</i> $\{y_1, y_2, y_3\}$, $\{z_1, z_2\}$ dan satu <i>output layer</i> $\{y\}$ (sumber: Theodoridis (2015))	15
2.8	Visualisasi <i>random forest</i> yang memprediksi kelas k untuk data x berdasarkan voting hasil klasifikasi setiap <i>tree</i> $\{k_1, k_2, \dots, k_b\}$ (sumber: http://www.scirp.org)	15
3.1	Indonesian open domain information extraction flowchart	17
3.2	Contoh format CoNLL-U untuk sebuah kalimat	20
4.1	Diagram hasil eksperimen perbandingan model <i>supervised learning</i> untuk <i>triple selector</i>	29
4.2	Diagram hasil eksperimen perbandingan model <i>supervised learning</i> untuk <i>triple selector</i> dengan menggunakan data latih sebagai data uji	31

DAFTAR TABEL

2.1	Perbandingan antara <i>information extraction</i> tradisional (IE), <i>open domain extraction</i> (open IE) dan <i>knowledge extraction</i> (KE)	8
3.1	Tahapan penelitian	16
3.2	Aturan pembangkitan kandidat <i>triple</i>	21
3.3	Fitur klasifikasi <i>triple selector</i>	24
3.4	Aturan ekspansi untuk <i>token</i> subjek atau objek	25
3.5	Aturan ekspansi untuk <i>token</i> predikat	25
4.1	Hasil eksperimen perbandingan model <i>supervised learning</i> untuk <i>triple selector</i>	30
4.2	Waktu eksekusi sistem <i>open IE end-to-end</i>	30
4.3	Hasil eksperimen perbandingan model <i>supervised learning</i> untuk <i>triple selector</i> dengan menggunakan data latih sebagai data uji . . .	31

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, salah satu cara paling sederhana untuk mengekstraksi nama penulis dari berita elektronik 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 untuk bahasa Inggris (Banko et al., 2007; Schmitz et al., 2012; Angeli et al., 2015). 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 dikembangkan untuk bahasa Inggris 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.

<p>Input</p> <p>”Sembungan adalah sebuah desa yang terletak di kecamatan Kejajar, kabupaten Wonosobo, Jawa Tengah, Indonesia.”</p> <p>Output</p> <ol style="list-style-type: none"> 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* (satu relasi dan dua argumen/entitas) dari satu atau lebih kalimat 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 *supervised 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 mengimbangi kinerja sistem sistem *open IE* untuk bahasa Inggris pada penelitian terkait.

1.3 Tujuan dan Manfaat

Tujuan dan manfaat dari penelitian ini adalah:

Tujuan

1. Merancang dan mengimplementasikan sistem *open IE* untuk teks bahasa Indonesia.
2. Mengumpulkan dan membangun *dataset* yang diperlukan oleh sistem *open IE* bahasa Indonesia.
3. Mencari model *supervised learning* yang sesuai sistem *open IE* 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, evaluasi dan analisis 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

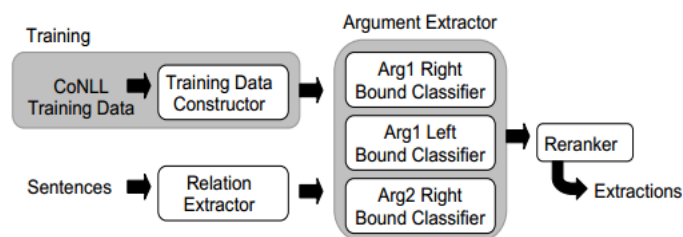
Sejak pertama kali diperkenalkan pada tahun 2007 (Banko et al., 2007), sudah ada beberapa penelitian mengenai *open IE* untuk bahasa Inggris yang dipublikasikan. Sistem *open IE* yang pertama diperkenalkan adalah TEXTRUNNER (Banko et al., 2007). Sistem ini kemudian dikembangkan oleh sistem-sistem dari 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 inkohereni 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* berbasis 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*) *dataset* 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 mengganti 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* yang membentuk (subjek, predikat, ob-

¹Stanford Core NLP <https://stanfordnlp.github.io/CoreNLP/>

jek) ini ternyata berlaku umum untuk semua dokumen yang berisi teks bahasa natural sehingga dapat diterapkan pada dokumen 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 Nagues, 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 *dataset* untuk *classifier* baik secara manual atau otomatis.
2. Belajar (*learn*): melatih *classifier* untuk mengekstrak himpunan *triple* dari kalimat menggunakan *dataset* 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

²Resource Data Format W3C <https://www.w3.org/RDF/>

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*, *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 probabili-

tas 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 dihasilkan sesuai urutan 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-U³ adalah format anotasi berbasis *token* (*token-based*) yang menggunakan himpunan *POS tag* dan *dependency relation* yang berlaku untuk banyak bahasa atau universal (Nivre et al., 2016) (terlampir). CoNLL-U merupakan pengembangan dari CoNLL-X, format yang disepakati dalam *Conference on Computational Natural Language Learning* ke sepuluh dan juga dijadikan format standar oleh *dependency parser* pada *Stanford Core NLP*. Setiap *token* pada CoNLL-U memiliki sepuluh kolom atau atribut:

1. ID: Indeks *token* dalam kalimat (mulai dari 1)
2. FORM: Bentuk asli dari *token*
3. LEMMA: Bentuk kamus dari *token*
4. UPOSTAG: *POS tag* universal
5. XPOSTAG: *POS tag* spesifik untuk bahasa tertentu
6. FEATS: Fitur tambahan atau spesifik untuk bahasa tertentu
7. HEAD: ID *token* yang merupaka kepala (*head*) dari *token* ini (0 untuk akar atau *head* kalimat)
8. DEPREL: *dependency relation* universal
9. DEPS: *dependency graph* khusus atau spesifik untuk bahasa tertentu
10. MISC: Anotasi tambahan yang belum tercakup di anotasi lainnya, contoh: *named-entity*

2.4 Supervised Learning

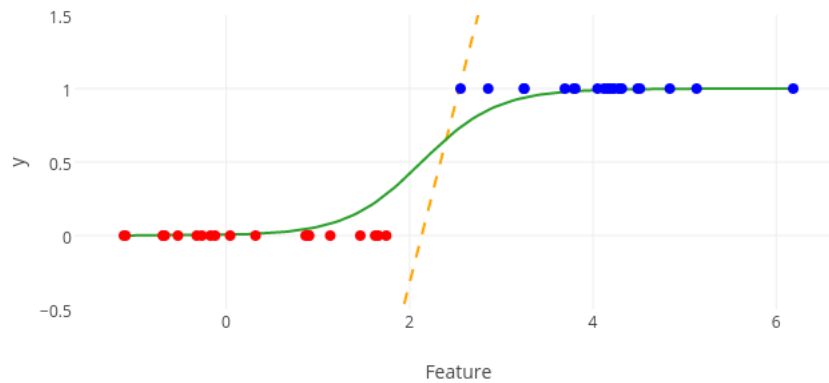
Supervised learning adalah teknik *machine learning* yang mempelajari pola dari *dataset* yang telah diberi label atau dikelompokkan (Mohri et al., 2012). Metode *supervised learning* dapat dibagi menjadi dua, yaitu deskriptif (*descriptive learning*) dan generatif (*generative learning*). Pada *descriptive 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*,

³CoNLL-U <http://universaldependencies.org/format.html>

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>)

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

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

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

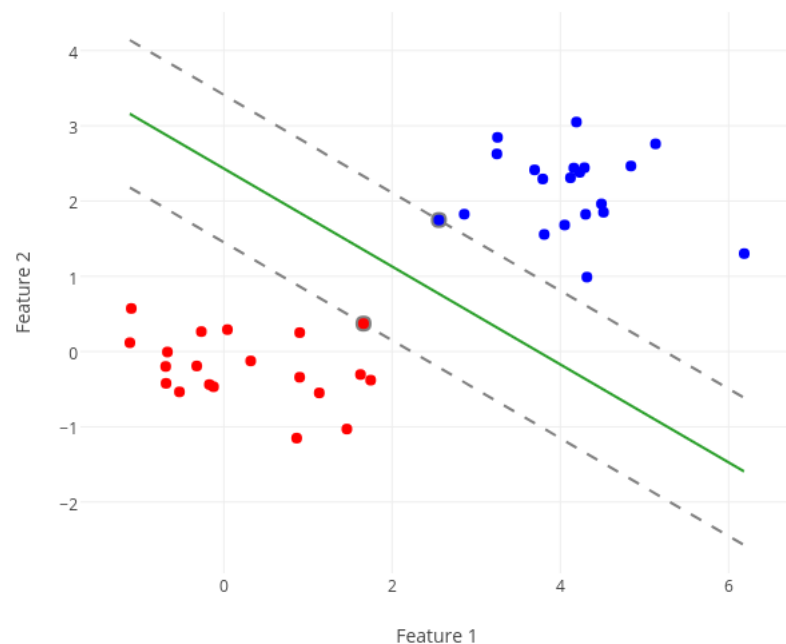
$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} L(\theta) \quad (2.3)$$

dimana, θ = bobot

α = 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 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>)

$$K(x, y) = (x^T y + c)^d \quad (2.4)$$

di mana, x = data atau fitur,

y = kelas atau label,

d = derajat polinomial,

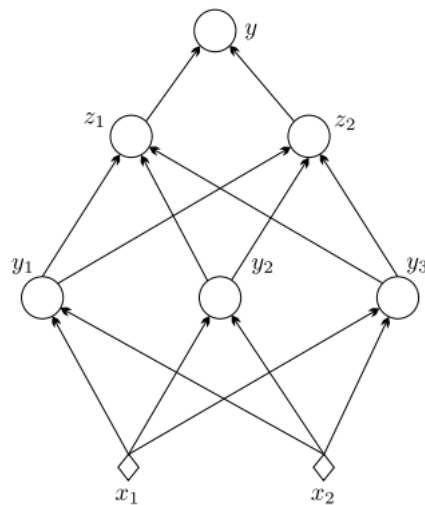
c = konstanta

2.4.3 Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) atau *feed-forward neural network* adalah pemodelan klasifikasi nonlinier berbasis 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*: $f(x) = \max(0, x)$ (Nair and Hinton, 2010)

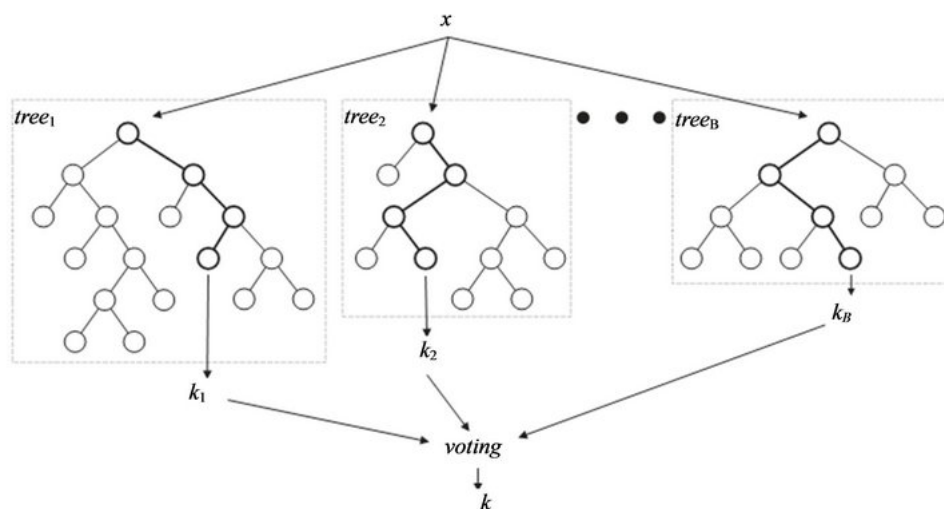
MLP dilatih dengan menyesuaikan bobot secara iteratif menggunakan algoritma *gradient descent* dan *backpropagation* (Theodoridis, 2015).



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* $\{k_1, k_2, \dots, k_b\}$ (sumber: <http://www.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, 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 <i>Open IE</i>
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 pencari¹² 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.

3.2 Rancangan dan Implementasi Sistem

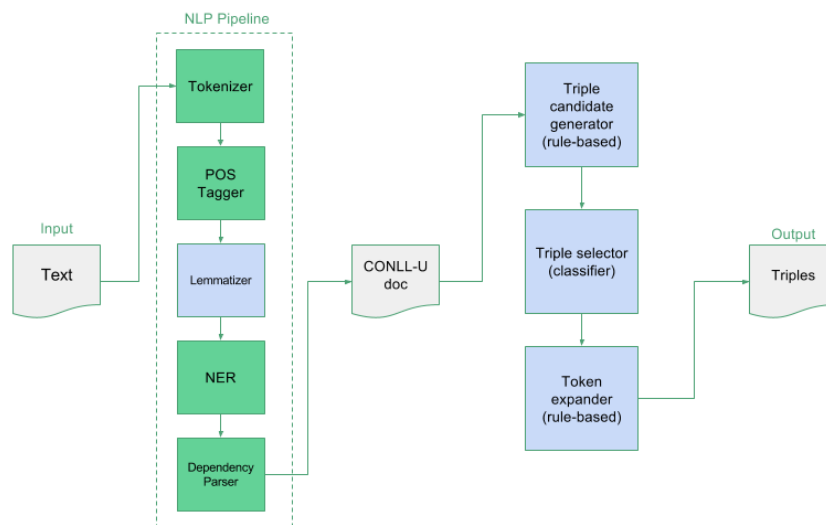
Pada tahap ini Penulis merancang sistem *open IE* untuk bahasa Indonesia yang mengadaptasi beberapa teknik pada sistem *open IE* pada penelitian terkait. Ran-

¹Database Jurnal Universitas Indonesia <http://remote-lib.ui.ac.id>

²Google Scholar <https://scholar.google.co.id/>

cangan 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 *dataset* untuk untuk *triple selector* dengan menganotasi manual kandidat *triple* yang dihasilkan oleh *triple candidate generator* dan *NLP pipeline*.
2. Belajar (*learn*): melatih *triple selector* untuk mengekstrak himpunan *triple* dari kalimat menggunakan *dataset* dari fase Label. Hasil dari fase ini adalah model yang dipakai pada fase berikutnya.
3. Ekstrak (*extract*): mengekstrak himpunan *triple* dari kalimat menggunakan *NLP pipeline*, *triple candidate generator*, *token expander* dan *triple selector* yang telah dilatih pada fase Belajar. Alur kerja pada fase ini ditunjukkan pada Gambar 3.1.



Gambar 3.1: Indonesian open domain information extraction flowchart

3.2.1 NLP Pipeline

NLP pipeline adalah modul yang berisi serangkaian *NLP task* yang menganotasi kalimat bahasa Indonesia dan menyimpannya sebagai dokumen dengan format CoNLL-U. Modul ini menerima dokumen teks yang berisi satu atau lebih kalimat yang dipisahkan oleh karakter baris baru (*newline*) dan menghasilkan dokumen teks

berisi kalimat yang telah dipotong menjadi *token* dan diberi anotasi dengan format CoNLL-U. Rangkaian ini diimplementasikan menggunakan pustaka *Stanford Core NLP*, seperti yang ditunjukkan pada berkas `DependencyParser.java` pada lampiran, dan didistribusikan dalam format *Java Archive* (JAR) sehingga mudah diintegrasikan dengan modul lain. *NLP task* yang terdapat pada rangkaian ini adalah sebagai berikut:

1. *Tokenizer*

Tokenizer yang digunakan pada rangkaian ini adalah yang disediakan pustaka *Stanford Core NLP*, `PTBTokenizer` (Manning et al., 2014). *Tokenizer* berbasis aturan (*rule-based*) ini mengikuti *tokenizer* yang digunakan untuk menghasilkan *Penn Treebank 3*³ (*treebank* bahasa Inggris). Meskipun *tokenizer* ini menyediakan opsi untuk menyesuaikan proses dengan bahasa lain, di penelitian ini kami hanya menggunakan opsi standar untuk memotong kalimat berdasarkan *whiteline* untuk mendapatkan *token*.

Part of Speech Tagger (*POS tagger*) yang digunakan pada rangkaian ini adalah, `MaxentTagger` (Toutanova et al., 2003), yang juga merupakan bagian dari pustaka *Stanford Core NLP*. *POS tagger* berbasis *multi-class logistic regression* ini dilatih dengan *dataset* yang diperoleh dengan mengekstraksi *POS tag* dari 5,036 kalimat *treebank universal dependency* (UD) bahasa Indonesia⁴. Hasil pengujian model *POS tagger* ini, menggunakan 559 kalimat lain dari sumber yang sama, mencapai akurasi per token **93.68%** dan akurasi per kalimat **63.91%** (seluruh token dalam kalimat dianotasi dengan *POS tag* yang benar).

2. *Lemmatizer*

Lemmatizer yang digunakan pada rangkaian ini diadaptasi dari *lemmatizer* bahasa Indonesia berbasis aturan (Suhartono, 2014) dan diberi nama `IndonesianLemmaAnnotator`. Adaptasi dilakukan dengan melakukan perubahan berikut:

- Kemampuan untuk memproses tidak hanya kata tapi juga kalimat
- Peningkatan kecepatan dengan penggunaan *in-memory database*
- Meningkatkan *reusability* dengan implementasi ulang menggunakan Java serta integrasi dengan pustaka *Stanford Core NLP*

³*Penn Treebank 3* <https://catalog.ldc.upenn.edu/LDC99T42>

⁴UD *treebank* Indonesia https://github.com/UniversalDependencies/UD_Indonesian

Lemmatizer ini mencapai akurasi **99%** saat diuji dengan 5.638 pasangan kata dan *lemma* bahasa Indonesia dari Suhartono (2014).

3. *Named-Entity Recognizer (NER)*

Named-entity recognizer (NER) yang digunakan dalam rangkaian ini adalah *CRFClassifier* (Finkel et al., 2005) dari pustaka *Stanford Core NLP*. NER berbasis *Conditional Random Field (CRF) sequence models* ini dilatih dan diuji menggunakan *dataset* yang didapatkan dari dua sumber, yaitu dari Fakultas Ilmu Komputer, Universitas Indonesia dan dari repositori kode publik⁵. *dataset* ini berisi kalimat-kalimat yang setiap *token*-nya yang relevan sudah diberi anotasi dari lima kelas: *Person*, *Organization*, *Quantity* dan *Time*. Hasil pelatihan dengan 3,535 kalimat dan pengujian dengan 426 kalimat adalah model yang mencapai rata-rata presisi 0.86, *recall* 0.85 dan *F₁-score* **0.86**.

4. *Dependency Parser*

Dependency parser yang digunakan dalam rangkaian ini adalah salah satu modul dalam pustaka *Stanford Core NLP*, yaitu *nndep.DependencyParser* (Chen and Manning, 2014). *Dependency parser* ini berbasiskan jaringan syaraf tiruan (*artificial neural network*) yang mempelajari pola transisi antar *token* dalam kalimat dalam membentuk *dependency tree*. *dataset* yang digunakan untuk melatih dan menguji *dependency parser* ini diperoleh dari *treebank universal dependency (UD)* bahasa Indonesia (sama dengan yang digunakan untuk *POS tagger*). Model yang dihasilkan dengan melatih *dependency parser* menggunakan 5,036 kalimat bahasa Indonesia ini mencapai nilai **70%** UAS (*Unlabeled Attachment Score*) dan **46%** LAS (*Labeled Attachment Score*) ketika diuji dengan 559 kalimat.

Estimasi kinerja dari modul *NLP pipeline* ini dihitung dari rata-rata kinerja *POS tagger* (*sentence accuracy*), *NER* (*F₁-score*) dan *dependency parser* (LAS), yaitu **65.30%**. Kinerja *tokenizer* dan *lemmatizer* tidak diperhitungkan karena dianggap sudah terwakili oleh *NLP task* yang lain. Hasil dari *NLP pipeline* ini adalah dokumen berisi anotasi setiap kalimat dengan format CoNLL-U seperti contoh pada Gambar 3.2. Dokumen ini menjadi input bagi modul *triple candidate generator* yang akan dijelaskan berikutnya.

⁵NER Indonesia <https://github.com/yususfsyaifudin/indonesia-ner>

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: Contoh format CoNLL-U untuk sebuah kalimat

3.2.2 Triple Candidate Generator

Modul *triple candidate generator* berfungsi untuk mengekstrak kandidat *triple* dari dokumen CoNLL-U yang dihasilkan oleh *NLP pipeline*. Modul ini menggunakan sejumlah aturan berbasis *POS tag* dan *dependency relation* yang ditampilkan pada Tabel 3.2 untuk mengekstrak kandidat *triple* dari tiap kalimat pada dokumen. Berbeda dengan TEXTRUNNER (Banko et al., 2007) yang menghasilkan hanya menentukan kandidat yang valid secara otomatis, kandidat yang dihasilkan modul ini tidak semuanya valid sehingga diperlukan pelabelan oleh manusia (pada fase Label) atau pelabelan otomatis oleh *classifier* (pada fase Extract) seperti pada STANFORD OPEN IE (Angeli et al., 2015). Modul ini diimplementasikan menggunakan bahasa pemrograman *opensource Python*⁶ dengan fungsi utama `extract_triples_by_combinations` pada berkas `tripletools.py` yang disertakan pada lampiran.

⁶Python <https://www.python.org>

Tabel 3.2: Aturan pembangkitan kandidat *triple*

Jenis	Kondisi
Subjek	<p><i>POS tag token</i> termasuk (PROPN, NOUN, PRON, VERB)</p> <p><i>Token</i> bukan termasuk ("yang", "adalah")</p> <p><i>Dependency</i> dari <i>token</i> bukan termasuk ("compound", "name")</p> <p><i>Dependency</i> dari <i>token</i> bukan termasuk ("compound", "name") tapi berjarak > 2 dari <i>head</i>-nya</p>
Predikat	<p>Posisi <i>token</i> setelah Subjek</p> <p><i>POS tag token</i> termasuk (VERB, AUX)</p>
Objek	<p>Posisi <i>token</i> setelah Subjek dan Predikat</p> <p><i>POS tag token</i> termasuk (PROPN, NOUN, PRON, VERB)</p> <p><i>Token</i> bukan termasuk ("yang", "adalah")</p> <p><i>Dependency</i> dari <i>token</i> bukan termasuk ("compound", "name")</p> <p><i>Dependency</i> dari <i>token</i> bukan termasuk ("compound", "name") tapi berjarak > 2 dari <i>head</i>-nya</p>

Contoh aplikasi aturan *triple candidate generator* pada contoh dokumen CoNLL-U pada Gambar 3.2 akan menghasilkan 17 kandidat *triple* di mana hanya 5 di antaranya merupakan kandidat yang valid (ditandai dengan centang (✓)):

- (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)

Untuk melatih modul *triple selector* yang dapat memilih kandidat *triple* yang valid, dibangun *dataset* dengan melakukan pelabelan manual pada 1,611 kandidat *triple* (132 positif dan 1,479 negatif) yang dihasilkan *triple candidate generator* dari 42 kalimat berformat CoNLL-U. Himpunan kalimat tersebut merupakan sebagian dari *dataset universal dependency* Indonesia yang ditambahkan anotasi *named-entity* secara manual.

Pada fase Ekstrak, *triple candidate generator* juga digunakan untuk menghasilkan kandidat *triple* dari dokumen CoNLL-U yang tidak berlabel seperti yang digambarkan pada Gambar 3.1. Hasil dari modul ini kemudian akan diseleksi oleh *triple selector* yang telah dilatih pada fase Belajar. Lebih jauh mengenai *triple selector* akan dijelaskan di subbab berikutnya.

3.2.3 Triple Selector

Modul *triple selector* adalah sebuah *supervised-learning classifier* yang dilatih untuk menyeleksi kandidat *triple* yang dihasilkan oleh *triple candidate generator*. Sebagai contoh, jika diberikan input 17 kandidat *triple* yang disebutkan pada subbab 3.2.2, modul ini akan mengambil lima kandidat *triple* yang diberi tanda centang (✓) dan mengabaikan yang lainnya.

Metode yang digunakan untuk membangun *classifier* pada modul ini adalah *random forest* (Breiman, 2001), yang merupakan metode *bagging* terhadap sejumlah *decision tree*. Implementasi *random forest* yang digunakan pada modul ini berasal dari pustaka *scikit-learn*⁷ seperti yang ditunjukkan pada berkas

⁷scikit-learn <http://scikit-learn.org>

`classifier.py` (untuk fase Belajar) dan `extract_triples.py` (untuk fase Ekstrak) pada lampiran. Konfigurasi yang digunakan pada model *random forest* pada modul ini adalah:

- Kriteria percabangan (`criterion`): *Gini Impurity*
- Jumlah sampel minimal untuk membuat cabang (`min_samples_split`): 5
- Jumlah fitur maksimum (`max_features`): 4 (akar dari jumlah total fitur)
- Kedalaman maksimum (`max_depth`): 8
- Jumlah pohon (`n_estimators`): 20
- Bobot kelas (`class_weight`): *balanced* (sesuai rasio kelas pada *dataset*)

Konfigurasi ini ditemukan dengan menggunakan algoritma *grid search* (Wasserman, 2015), algoritma pencarian menyeluruh (*exhaustive*) terhadap sejumlah himpunan *hyperparameter* untuk mengoptimalkan metrik evaluasi tertentu. Algoritma ini digunakan untuk mencari konfigurasi yang menghasilkan F_1 score terbaik untuk *random forest* dengan data yang ada.

Untuk melakukan klasifikasi, 17 fitur berbasis *POS tag*, *named-entity* dan *dependency relation* diekstrak dari masing-masing kandidat *triple* dengan rincian pada Tabel 3.3. Berbeda dengan TEXTRUNNER atau REVERB (Banko et al., 2007) (Etzioni et al., 2011) yang lebih memilih menggunakan *shallow syntactic features*, *classifier* pada sistem ini menggunakan *heavy linguistic features* seperti *dependency relation* untuk mengoptimalkan *precision* dan *recall*.

Tabel 3.3: Fitur klasifikasi *triple selector*

#	Fitur Klasifikasi
1	<i>POS tag</i> dari <i>token</i> Subjek
2	<i>Dependency relation</i> dari <i>token</i> Subjek
3	<i>POS tag</i> dari <i>head token</i> Subjek
4	<i>Named-entity</i> dari <i>token</i> Subjek
5	Jarak Subjek ke <i>token</i> Predikat
6	<i>Dependency relation</i> dari <i>token</i> Subjek ke Predikat
7	<i>POS tag</i> dari <i>token</i> Predikat
8	<i>Dependency relation</i> dari <i>token</i> Predikat
9	<i>POS tag</i> dari <i>head token</i> Predikat
10	Jumlah <i>dependents token</i> Predikat
11	<i>POS tag</i> dari <i>token</i> Objek
12	<i>Dependency relation</i> dari <i>token</i> Objek
13	<i>POS tag</i> dari <i>head token</i> Objek
14	<i>Named-entity</i> dari <i>token</i> Objek
15	Jumlah <i>dependents</i> dari <i>token</i> Objek
16	Jarak dari <i>token</i> Objek ke predikat
17	<i>Dependency relation token</i> Objek ke Predikat

Pada fase Belajar, data kandidat *triple* yang telah diberi label di fase Label diekstrak menjadi 17 fitur yang dijelaskan di Tabel 3.3 dan dinormalisasi menggunakan pustaka `StandardScaler` dari *skit-learn*. *Dataset* berisi fitur yang telah dinormalisasi tersebut dipakai untuk melatih *classifier* dan hasilnya juga disimpan dalam berkas biner (*binary file*) untuk digunakan pada fase Ekstrak.

3.2.4 *Token Expander*

Modul *token expander* adalah berfungsi mengekspansi *token* pada *triple* menjadi kata, kata majemuk atau frase sehingga makna *triple* menjadi lebih jelas. Contoh ekspansi *token* terhadap lima kandidat *triple* yang valid pada subbab 3.2.2 adalah:

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

- (Sembungan, terletak di, Indonesia)

Jika TEXTRUNNER menggunakan *noun phrase chunker* (Banko et al., 2007) untuk menemukan frase sebagai kandidat argumen (subjek atau objek), *token expander* menggunakan 11 aturan berbasis *POS tag*, *named-entity* dan *dependency relation* yang dirinci pada Tabel 3.4. Perbedaan lain dengan *TextRunner* adalah modul ini digunakan juga untuk mengekspansi negasi *token* predikat dengan aturan pada Tabel 3.5. Modul ini menelusuri setiap *dependent* dari sebuah *token* dan memutuskan apakah akan (1) melakukan ekspansi (*expand*) ke *dependent* tersebut, (2) mengabaikan (*ignore*) *dependent* tersebut, atau (3) membuang (*remove*) *dependent* tersebut. Sekalipun memiliki tujuan dan teknik yang berbeda dengan *clause selector* pada STANFORD OPEN IE (Angeli et al., 2015), modul ini sama-sama membentuk frase dengan menelusuri *dependent* dan memutuskan apakah sebuah *dependent* merupakan bagian dari frase yang sama atau berbeda.

Tabel 3.4: Aturan ekspansi untuk *token* subjek atau objek

#	Kondisi untuk <i>token</i> subjek atau objek	Aksi
1	Jika relasi <i>dependent</i> di antara()["compound", "name", "amod"]	Ekspansi
2	Jika <i>dependent</i> memiliki <i>named-entity</i> yang sama dengan <i>token</i>	Ekspansi
3	Jika <i>dependent</i> dan token berada dalam kutipan (<i>quote</i>)	Ekspansi
4	Jika <i>token</i> merupakan <i>root</i> kalimat	Abaikan
5	Jika <i>POS tag</i> <i>dependent</i> CONJ atau termasuk simbol (";", ")/)	Abaikan
6	Jika <i>POS tag</i> <i>dependent</i> termasuk ("VERB", "ADP")	Abaikan
7	Jika <i>dependent</i> memiliki <i>dependent</i> dengan <i>POS tag</i> "ADP"	Abaikan
8	Jika <i>POS tag</i> <i>dependent</i> di antara ("CONJ", "ADP") dan berada di depan frase	Buang
9	Jika <i>dependent</i> merupakan tanda kurung yang tidak lengkap	Buang
10	Jika <i>dependent</i> merupakan kata "yang" diakhir frase	Buang
11	Lain-lain	Abaikan

Tabel 3.5: Aturan ekspansi untuk *token* predikat

#	Kondisi untuk <i>token</i> predikat	Aksi
1	Jika <i>dependent</i> adalah "tidak"	Ekspansi
2	Lain-lain	Abaikan

Pada fase Label, *token expander* digunakan untuk mengekspansi kandidat *triple* yang dihasilkan *triple candidate generator* sehingga lebih mudah dimengerti manusia. Sedangkan pada fase Ekstrak, modul ini mengekspansi *triple* yang telah dipilih oleh *triple selector* sehingga maknanya lebih jelas. Implementasi dari modul ini dibuat dengan *Python*, yaitu pada fungsi `flatten_node` dalam berkas `tripletools.py` yang juga dilampirkan.

3.3 Evaluasi dan Analisis

3.3.1 Evaluasi

Evaluasi sistem *open IE* ini akan difokuskan pada modul *triple selector* karena modul ini ditopang oleh modul *triple candidate generator* dan *token expander* yang, bersama dengan *triple selector*, merupakan pusat kontribusi dari penelitian ini. Selain karena tidak banyak kontribusi yang diberikan melalui modul *NLP pipeline*, kinerja dari modul ini tidak dievaluasi secara khusus melainkan hanya akan dievaluasi secara tidak langsung pada evaluasi efisiensi sistem *open IE* ini. Evaluasi pada penelitian ini akan dibagi menjadi dua eksperimen:

3.3.1.1 Eksperimen Model *Triple Selector*

Pada eksperimen ini akan dibandingkan kinerja empat buah model *supervised learning* untuk melakukan klasifikasi *triple* untuk menentukan apakah *random forest* (Wasserman, 2015) adalah model yang paling cocok. Empat buah model tersebut adalah *linear logistic regression* (Fan et al., 2008), *polynomial support vector machine* (SVM) (Chang and Lin, 2011), *multi-layer perceptron* (MLP) (Hinton, 1989) dan *random forest* sendiri. Keempat model ini akan dilatih dan diuji dengan metode *k-fold cross-validation* menggunakan *dataset* yang telah dijelaskan pada subbab 3.2.2. Karena *dataset* yang digunakan memiliki rasio kelas positif dan negatif yang tidak seimbang, maka digunakan $k = 3$ pada *cross-validation* untuk mencegah adanya potongan (*fold*) *dataset* yang hanya terdiri dari kelas negatif.

Metrik yang akan dibandingkan pada eksperimen ini adalah *precision*, *recall* dan F_1 score hanya untuk kelas positif (*triple valid*) karena pada *open IE* data negatif tidak diperlukan. Karena untuk *task* klasifikasi *triple* ini *precision* dan *recall* sama-sama penting, maka metrik yang dipandang lebih penting adalah rerata dari dua metrik tersebut, yaitu F_1 -score (Angeli et al., 2015). Persamaan untuk menghitung *precision*, *recall* dan F_1 -score untuk data positif ditunjukkan secara berurutan pada persamaan 3.1, 3.2 dan 3.3.

$$precision_+ = \frac{|selected\ valid\ triples|}{|selected\ triples|} \quad (3.1)$$

$$recall_+ = \frac{|selected\ valid\ triples|}{|valid\ triples|} \quad (3.2)$$

$$F_1 = 2 \cdot \frac{precision_+ \cdot recall_+}{precision_+ + recall_+} \quad (3.3)$$

3.3.1.2 Eksperimen Waktu Ekstraksi *Triple*

Pada eksperimen ini diukur waktu eksekusi total sistem *open IE* yang diajukan pada tiga dokumen input yang memiliki jumlah kalimat yang bervariasi. Tujuan dari eksperimen ini adalah mengukur efisiensi sistem secara umum dan membandingkannya dengan penelitian terkait. Dokumen yang digunakan sebagai input pada eksperimen ini adalah:

1. Dokumen berisi 2 kalimat dari *dataset* pelatihan
2. Dokumen berisi 138 kalimat dari proposal penelitian
3. Dokumen berisi 5,593 dari *dataset* pelatihan

3.3.2 Analisis

Analisis akan dilakukan terhadap hasil eksperimen yang telah dijelaskan di atas dengan tujuan mengukur pencapaian pada penelitian ini relatif terhadap penelitian-penelitian terkait, memaparkan alasan di balik hasil eksperimen yang diperoleh serta mencari alternatif perbaikan atau peningkatan yang dapat dilakukan pada penelitian ini ke depannya.

BAB 4

HASIL DAN ANALISIS

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

4.1 Evaluasi

Dua eksperimen pada penelitian ini dilakukan pada *notebook* dengan sistem operasi *Ubuntu 15.04 64-bit*, prosesor *Intel Core i7 5500U (dual cores)*, RAM DDR3 8 GB dan penyimpanan SSD 250 GB. Program yang digunakan untuk melakukan eksperimen pertama adalah `classifier.py` sedangkan eksperimen yang kedua menggunakan program `extract_triples.py` (terlampir).

Pada eksperimen pertama, empat model *supervised learning* dilatih dan diuji menggunakan data kandidat *triple* yang sudah diberikan label, diekstraksi menjadi 17 fitur dan dinormalisasi. Metode yang digunakan untuk melatih dan menguji adalah *k-fold cross-validation* (Kohavi et al., 1995) dengan $k = 3$. Empat model yang dibandingkan beserta dengan konfigurasinya adalah sebagai berikut:

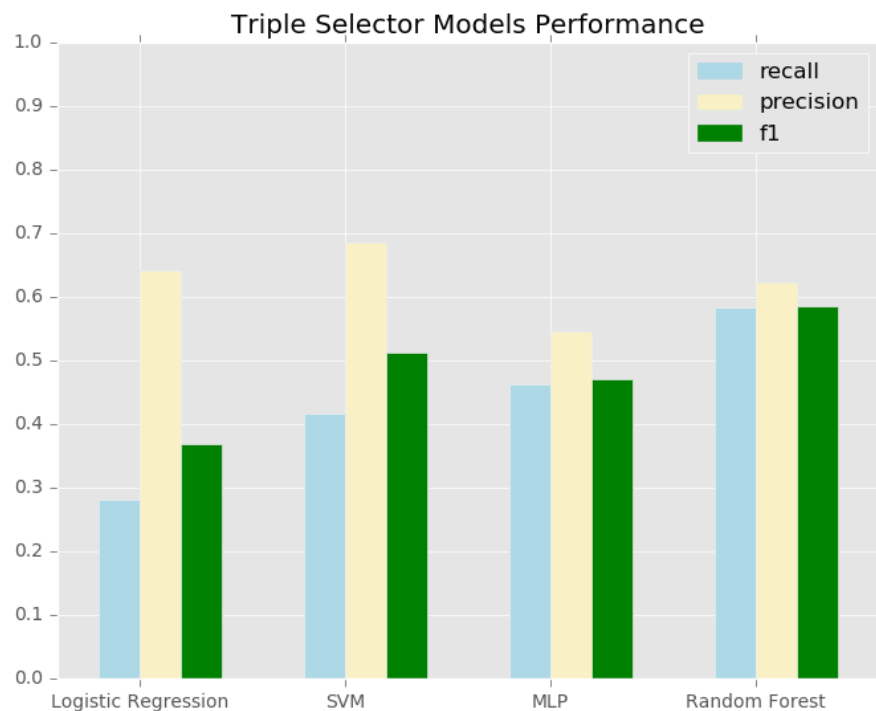
1. Linear Logistic Regression
 - Solver (cost function): `liblinear`
 - Penalty (regularizer): `l2`
2. Polynomial Support Vector Machine (SVM)
 - Kernel: `poly`
 - Degree: `5`
3. ReLU Multi-Layer Perceptron (MLP)
 - Hidden layers: `(20, 10)`
 - Activation: `relu` (Nair and Hinton, 2010)
 - Max. iteration: `1000`
4. Random Forest
 - Max. depth: `8`
 - Number of estimators: `20`

- Min. samples split: 5
- Criterion: gini (Mingers, 1989)
- Max. features: auto (pembulatan akar dari jumlah total fitur)
- Class weight: balanced (sesuai rasio kelas)

Eksperimen ini dilakukan dengan menjalankan program `classifier.py` (di direktori yang sama dengan pustaka utama `tripletools.py`) dengan input *dataset* fitur yang sudah dinormalisasi dengan format *comma separated value* (CSV) `triple-selector.train.csv` pada *terminal*:

```
$ python classifier.py --best triple-selector.train.csv
```

Hasil dari eksperimen pertama ini dapat dilihat pada Tabel 4.1 dan visualisasinya pada Gambar 4.1 di mana *random forest* mencapai nilai *recall* dan F_1 tertinggi, yaitu **0.58**. Sedangkan nilai *precision* tertinggi, **0.68**, dicapai oleh SVM.



Gambar 4.1: Diagram hasil eksperimen perbandingan model *supervised learning* untuk *triple selector*

Tabel 4.1: Hasil eksperimen perbandingan model *supervised learning* untuk *triple selector*

Model	<i>Precision</i>	<i>Recall</i>	F_1
Logistic Regression	0.64	0.28	0.36
SVM	0.68	0.41	0.51
MLP	0.54	0.46	0.47
Random Forest	0.62	0.58	0.58

Pada eksperimen kedua, sistem *open IE* dipakai untuk mengekstrak *triple* dari tiga buah dokumen dengan ukuran bervariasi. Metrik utama pada eksperimen ini adalah waktu total eksekusi yang dipakai untuk menghitung waktu eksekusi rata-rata per kalimat. Sebagai tambahan, diamati juga jumlah *triple* yang dapat diekstraksi dari tiap dokumen. Eksperimen ini dilakukan dengan menjalankan program utama `extract_triples.py` untuk setiap dokumen (`doc1.txt`, `doc2.txt`, `doc3.txt`) yang berisi satu kalimat per baris:

```
$ python extract_triples.py -f tsv doc1.txt
$ python extract_triples.py -f tsv doc2.txt
$ python extract_triples.py -f tsv doc3.txt
```

Hasil eksperimen tersebut ditunjukkan pada Tabel 4.2 di mana waktu eksekusi rata-rata per kalimat **0.014 detik/kalimat** dicapai untuk ukuran dokumen terbesar 5,593 kalimat. Dapat dilihat bahwa rata-rata waktu yang dibutuhkan untuk memproses satu kalimat semakin menurun seiring dengan bertambahnya jumlah kalimat pada dokumen. Rata-rata jumlah *triple* yang diekstraksi dari setiap dokumen adalah **3.3 triple/kalimat**.

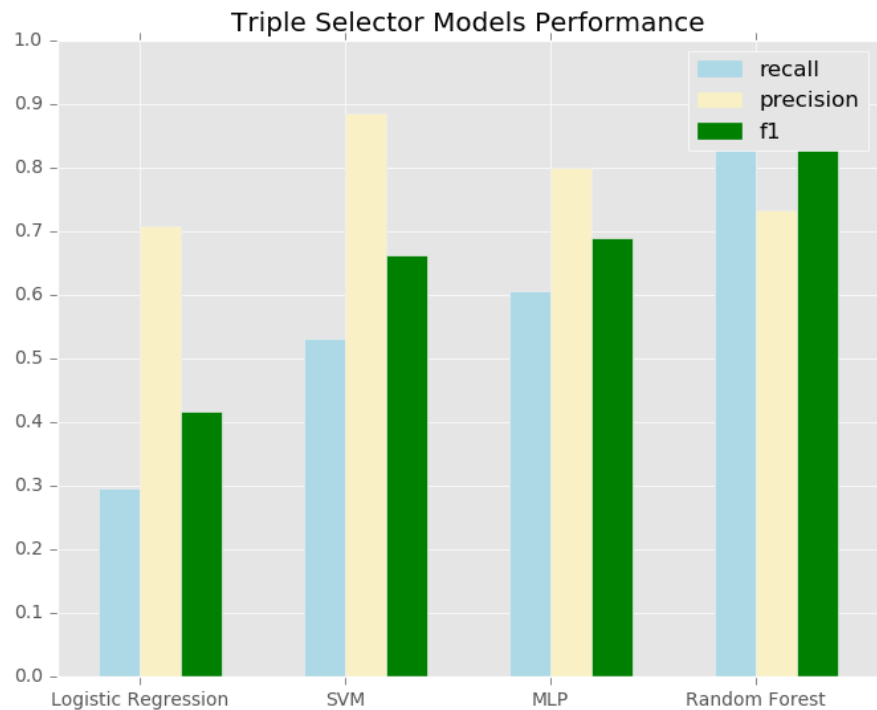
Tabel 4.2: Waktu eksekusi sistem *open IE end-to-end*

Jumlah kalimat	<i>Triple</i>	Total (detik)	Per kalimat (detik)
2	7	6.1	0.800
138	429	11.3	0.082
5,593	19,403	78.6	0.014

4.2 Analisis

Hasil eksperimen pertama di mana nilai F_1 tertinggi hanya 0.58, mengindikasikan bahwa semua model mengalami kesulitan untuk mempelajari pola *triples* dari *dataset* yang diberikan. Kemungkinan penyebab hasil ini adalah masalah pada

model (pemilihan fitur atau algoritma) atau kualitas *dataset* yang digunakan (konflik pola atau ketidaklengkapan pola *dataset*). Untuk memastikan penyebab dari hasil eksperimen pertama ini, Penulis melakukan eksperimen tambahan yaitu menguji tiap model *triple selector* menggunakan *dataset* latih (data yang sama). Hasil cukup baik yang ditunjukkan pada Gambar 4.2 dan Tabel 4.3, di mana F_1 tertinggi **0.83**, *recall* tertinggi **0.96** dan *precision* tertinggi **0.88**, menunjukkan bahwa fitur yang dipilih dan model yang digunakan tidak memiliki masalah (kecuali model linier, *logistic regression*). Oleh karena itu Penulis berargumen bahwa masalah utama terdapat pada *dataset* yang digunakan, yaitu tidak cukupnya pola $\frac{2}{3}$ data yang dipakai melatih untuk mengenali pola sisa $\frac{1}{3}$ data yang dipakai untuk menguji.



Gambar 4.2: Diagram hasil eksperimen perbandingan model *supervised learning* untuk *triple selector* dengan menggunakan data latih sebagai data uji

Tabel 4.3: Hasil eksperimen perbandingan model *supervised learning* untuk *triple selector* dengan menggunakan data latih sebagai data uji

Model	<i>Precision</i>	<i>Recall</i>	F_1
Logistic Regression	0.70	0.29	0.41
SVM	0.88	0.53	0.66
MLP	0.80	0.60	0.68
Random Forest	0.73	0.96	0.83

Selain disebabkan oleh kurangnya jumlah kalimat yang dianotasi, permasalahan pada *dataset triple selector* ini juga tentu dipengaruhi oleh kemampuan *triple candidate generator* untuk menghasilkan jumlah kandidat *triple* valid (data positif) yang sebanding jumlahnya dengan kandidat yang tidak valid (data negatif). Penulis berpendapat bahwa, selain menambah data, ada minimal dua solusi yang dapat dilakukan untuk meningkatkan kualitas *dataset*, yaitu:

1. Mengekstrak *triple* implisit dari kalimat

Module *triple candidate generator* pada penelitian ini baru menangani *triple* yang memiliki struktur yang eksplisit sehingga jumlah data positif sangat sedikit. Dengan menambah pola *triple* yang dapat dibangkitkan, *dataset* akan lebih seimbang dan memiliki pola lebih banyak (Schmitz et al., 2012). Contoh *triple* eksplisit yang perlu ditangani lebih jauh:

- *Triple (kecamatan Kejajar, terletak di, Jawa Tengah)* dari kalimat asal "Sembungan adalah sebuah desa yang terletak di kecamatan Kejajar, kabupaten Wonosobo, Jawa Tengah, Indonesia."
- *Triple (Sukarno, adalah, Presiden)* dari kalimat asal "Presiden pertama Indonesia Sukarno lahir di Surabaya."

2. Mengurangi ekstraksi *triple* invalid dari kalimat

Rasio perbandingan data positif dan negatif pada *dataset* adalah 1:11. Hal ini menunjukkan bahwa proses pembangkitan kandidat *triple* ini masih bisa dibuat lebih efisien. Salah satu teknik yang bisa digunakan adalah membuat aturan yang lebih spesifik atau melatih *classifier* untuk mengekstrak frase *self-contained* (Angeli et al., 2015).

Hal menarik yang ditemukan dari hasil eksperimen pertama adalah *random forest*, yang mewakili *ensemble classifier*, merupakan pemodelan yang paling cocok dibandingkan pemodelan linier (*logistic regression*), nonlinier dengan optimisasi margin (SVM) dan jaringan syaraf tiruan (MLP). Penulis menyimpulkan bahwa dibutuhkan pemodelan yang keseimbangan antara *precision* dan *recall*-nya relatif mudah disesuaikan untuk module *triple selector*. Sekalipun tidak memiliki *precision* setinggi SVM, *Random forest* lebih unggul karena penyesuaian jumlah dan kedalaman *tree* memudahkan penyeimbangan *precision* dan *recall* yang menghasilkan F_1 yang paling baik. Potensi SVM yang mampu mencapai *precision* yang paling tinggi ini juga mungkin bisa dimanfaatkan dengan melakukan *bagging* (Breiman, 1996) SVM dan *random forest* untuk meningkatkan kinerja lebih jauh.

Sebagai tambahan, hasil eksperimen kedua menunjukkan bahwa waktu rata-rata 0.014 detik/kalimat yang dibutuhkan sistem *open IE* untuk mengekstrak *triple* dari dokumen yang berukuran sedang (5,593 kalimat), cukup sebanding dengan sistem lain seperti TEXTRUNNER yang membutuhkan 0.036 detik/kalimat (Banko et al., 2007). Hal ini menunjukkan bahwa penggunaan fitur *heavy linguistic* cukup efisien juga digunakan pada dokumen yang berukuran sedang. Kekurangan dari sistem ini tentu ada pada ekstraksi pada dokumen dengan kalimat sedikit. Selain itu juga kedepannya perlu dikaji sejauh skalabilitas sistem ini dengan melakukan ekstraksi pada dokumen yang lebih besar.

BAB 5

PENUTUP

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

5.1 Kesimpulan

Melalui penelitian ini telah diajukan rancangan sistem *open IE* untuk bahasa Indonesia yang menggunakan *NLP pipeline* dan kombinasi model heuristik dan *supervised learning*. Sekalipun belum mencapai akurasi yang tinggi, implementasi sistem ini mampu mengekstraksi *triple* dari teks atau dokumen bahasa Indonesia secara otomatis dalam waktu yang sebanding dengan sistem dari penelitian terkait. Pada penelitian ini juga dibangun *dataset* untuk seleksi *triple* dan dikumpulkan himpunan *dataset* untuk *NLP task* bahasa Indonesia yang dapat digunakan untuk penelitian terkait. Semua kode sumber dan *dataset* penelitian ini juga dipublikasikan pada repositori publik¹ untuk memudahkan replikasi. Kesimpulan yang dapat diambil berdasarkan evaluasi dan analisis dalam penelitian ini adalah:

1. Kombinasi *NLP pipeline* dasar (*POS tagging*, *lemmatization*, *NER* dan *dependency parsing*) berbasis *Universal Dependency*, model heuristik dan *supervised learning* dapat melakukan *open domain information extraction (open IE)* dalam format *triple* (subjek, predikat, objek) dari teks bahasa Indonesia secara otomatis.
2. Model *supervised-learning* yang paling sesuai untuk melakukan seleksi *triple* berdasarkan fitur berbasis *POS tag*, *named-entity* dan *dependency relation* adalah *random forest*, yang merupakan *ensemble classifier*. Model ini mencapai nilai F_1 0.58, yang lebih tinggi dari tiga model linier dan nonlinier lain yang diujikan.
3. Sistem *open IE* yang diajukan dapat melakukan ekstraksi 19,403 *triple* dari dokumen yang terdiri atas 5,593 kalimat bahasa Indonesia dalam waktu 78.6 detik atau 0.014 detik/kalimat. Dapat disimpulkan bahwa sistem ini cukup efisien untuk digunakan pada dokumen berukuran sedang.

¹Repositori penelitian github.com/yohanesgultom/id-openie

5.2 Saran

Berdasarkan hasil analisis, berikut adalah saran pengembangan penelitian ini ke depannya:

1. Memperbaiki kualitas *dataset* untuk melatih *triple selector* dengan menambah lebih banyak data.
2. Mengembangkan *triple candidate generator* untuk bisa mengekstraksi kandidat *triple* implisit dan mengurangi kandidat *triple* yang invalid.
3. Menggunakan kombinasi antara *ensemble classifier* seperti *random forest* dan *classifier* berpresisi tinggi seperti SVM sebagai *triple selector* untuk meningkatkan *precision* dan F_1 score.
4. Melakukan pengujian sistem secara keseluruhan dengan dokumen yang lebih besar serta membangun *dataset* untuk bisa mengevaluasi keseluruhan sistem secara lebih *reliable*.

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.
- Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine learning*, 3(4):319–342.
- 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`

```
1 import os
2 import csv
3 import argparse
4 import subprocess
5 import numpy as np
6 import json
7 from sys import platform
8 from sklearn.externals import joblib
9 from sklearn.preprocessing import StandardScaler
10 from tripletools import (
11     vectorize,
12     parse_conllu_file,
13     extract_triples_by_combinations,
14     get_best_features
15 )
16 from pprint import pprint
17
18 # choose script based on OS (windows or *nix)
19 DEPPARSE_SCRIPT = 'bin' + os.sep + 'id-openie'
20 if platform == 'win32':
21     DEPPARSE_SCRIPT += '.bat'
22
23
24 def write_json(triples, y, out):
25     count = 0
26     grouped = {}
27     for i in range(y.shape[0]):
28         if y[i] == 1:
29             triple = triples[i]
30             if triple[1] not in grouped:
31                 grouped[triple[1]] = {}
32             if triple[2] not in grouped[triple[1]]:
33                 grouped[triple[1]][triple[2]] = {}
34             if triple[3] not in grouped[triple[1]][triple[2]]:
35                 grouped[triple[1]][triple[2]][triple[3]] = {}
36             count += 1
37     out.write(json.dumps(grouped) + '\n')
38     return count
39
40
41 def write_tsv(triples, y, out):
42     writer = csv.writer(out, delimiter='\t', quoting=csv.QUOTE_NONE, quotechar='')
43     count = 0
44     for i in range(y.shape[0]):
45         if y[i] == 1:
46             writer.writerow(triples[i])
47             count += 1
48     return count
49
50
51 def extract(conllu_file, classifier, out, format='tsv', scaler=None):
52     X = []
53     triples = []
54     for index, s, s_header in parse_conllu_file(conllu_file):
55         for first, second, third, subj, pred, obj in extract_triples_by_combinations(s, s_header):
56             X.append(vectorize(first, second, third))
57             triples.append((first['sentence_id'], subj, pred, obj))
58     X = np.array(X, dtype='float32')
59     # apply best features selection
60     X = X[:, get_best_features()]
61     # scale if scaler is available
62     if scaler:
63         X = scaler.transform(X)
64     y = classifier.predict(X)
65     # write output
66     if format == 'tsv':
```



```

67         return write_tsv(triples, y, out)
68     else: # format == 'json'
69         return write_json(triples, y, out)
70
71
72 if __name__ == '__main__':
73
74     if os.path.isfile(DEPPARSE_SCRIPT):
75         parser = argparse.ArgumentParser(description='Extract triples from Indonesian text')
76         parser.add_argument('input_file', help='Input file containing 1 (one) Indonesian sentence per line')
77         parser.add_argument('-m', '--model_file', help='Triples classifier model file', default='triples-
classifier-model.pkl')
78         parser.add_argument('-s', '--scaler_file', help='Triples classifier scaler file', default='triples-
classifier-scaler.pkl')
79         parser.add_argument('-o', '--output_file', help='Output file containing triples')
80         parser.add_argument('-f', '--output_format', help='Output file format', choices=['json', 'tsv'],
default='json')
81         args = parser.parse_args()
82         args.output_file = args.output_file if args.output_file else 'triples.' + args.output_format
83
84         # dependency parsing
85         print('Parsing dependency tree..')
86         depparse_output = os.path.basename(args.input_file) + '.conllu'
87         subprocess.call([DEPPARSE_SCRIPT, '-f', args.input_file])
88
89         # extract triples
90         classifier = joblib.load(args.model_file)
91         scaler = joblib.load(args.scaler_file)
92         with open(args.output_file, 'wb') as out:
93             count = extract(depparse_output, classifier, out, args.output_format, scaler=scaler)
94
95         print('{} triple(s) extracted'.format(count))
96         print('Triples saved in ' + args.output_file)
97     else:
98         print('File not found: ' + DEPPARSE_SCRIPT)

```

LAMPIRAN 2: KODE SUMBER *NLP PIPELINE*

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

```
1 package id.nlp.depparser;
2
3 import edu.stanford.nlp.ling.CoreAnnotations;
4 import edu.stanford.nlp.pipeline.*;
5 import edu.stanford.nlp.semgraph.SemanticGraph;
6 import edu.stanford.nlp.semgraph.SemanticGraphCoreAnnotations;
7 import edu.stanford.nlp.trees.ud.CoNLLUDocumentWriter;
8 import edu.stanford.nlp.trees.ud.ExtendedCoNLLUDocumentWriter;
9 import edu.stanford.nlp.util.CoreMap;
10 import edu.stanford.nlp.util.PropertiesUtils;
11 import net.sourceforge.argparse4j.ArgumentParsers;
12 import net.sourceforge.argparse4j.inf.ArgumentParser;
13 import net.sourceforge.argparse4j.inf.ArgumentParserException;
14 import net.sourceforge.argparse4j.inf.Namespace;
15
16 import java.io.File;
17 import java.io.IOException;
18 import java.sql.SQLException;
19 import java.util.ArrayList;
20 import java.util.List;
21 import java.util.Properties;
22
23 import static edu.stanford.nlp.pipeline.Annotator.*;
24
25 public class DependencyParser {
26
27     static final String TAGGER_MODEL = "tagger-id.universal.model";
28     static final String NER_MODEL = "ner-id.model.ser.gz";
29     static final String PARSER_MODEL = "parser-id.conllu.model.gz";
30     static final int NUM_THREADS = 1;
31     static final String OUTPUT_FORMAT = "conllu";
32
33     AnnotatorPool annotatorPool;
34     Properties props;
35     StanfordCoreNLP pipeline;
36
37     public DependencyParser() throws SQLException, IOException, ClassNotFoundException {
38         this(TAGGER_MODEL, NER_MODEL, PARSER_MODEL, NUM_THREADS);
39     }
40
41     public DependencyParser(
42         String taggerModel,
43         String nerModel,
44         String parserModel,
45         int numThreads
46     ) throws SQLException, IOException, ClassNotFoundException {
47
48         // Create the Stanford CoreNLP pipeline
49         this.props = PropertiesUtils.asProperties(
50             "annotators", "tokenize,ssplit,pos,lemma,ner,depparse",
51             "ner.model", nerModel,
52             "ner.useSUTime", "false",
53             "pos.model", taggerModel,
54             "depparse.model", parserModel,
55             "splitter.nomodel", "true",
56             "ignore_affinity", "true",
57             "outputFormat", OUTPUT_FORMAT,
58             "threads", String.valueOf(numThreads)
59         );
60
61         // Create annotator pools
62         this.annotatorPool = new AnnotatorPool();
63         AnnotatorImplementations annotatorImplementations = new IndonesianAnnotatorImplementations();
64         annotatorPool.register(STANFORD_TOKENIZE, AnnotatorFactories.tokenize(props, annotatorImplementations)
65     );
```

```

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`

```
1 import itertools
2 import csv
3 import argparse
4
5
6 BEST_FEATURES = [0, 1, 2, 3, 5, 6, 10, 11, 12, 14, 17, 18, 19, 20, 21, 22, 23] # F1 0.586
7 # BEST_FEATURES = [0, 1, 2, 3, 5, 6, 10, 11, 12, 17, 18, 19, 20, 21, 22, 23] # F1 0.579
8 # 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,
9     25, 26] # F1 0.547
10
11 # constants
12 conllu = ['ID', 'FORM', 'LEMMA', 'UPOSTAG', 'XPOSTAG', 'FEATS', 'HEAD', 'DEPREL', 'DEPS', 'MISC']
13 postag = ['', 'ADJ', 'ADP', 'ADV', 'AUX', 'CCONJ', 'DET', 'INTJ', 'NOUN', 'NUM', 'PART', 'PRON', 'PROPN', 'PUNCT', 'SCONJ', 'SYM', 'VERB', 'X', 'CONJ']
14 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']
15 entity = ['', 'PERSON', 'LOCATION', 'ORGANIZATION', 'TIME', 'QUANTITY', 'OTHER']
16
17 # extraction RULES
18 subject_object_candidates_pos = ['PROPN', 'NOUN', 'PRON', 'VERB']
19 predicate_candidates_pos = ['VERB', 'AUX']
20 non_subject_object_candidates_form = ['yang', 'adalah']
21 non_predicate_candidates_form = ['yang']
22 num_siblings = 1 # bigram
23
24
25 def extract_triples_by_root_children(conllu_s, header):
26     """
27     Extract features (triples) for clustering from sentence (conllu_s)
28     by combining sentence root/header with 2 of its children
29     """
30     # find all direct branches of header
31     direct_branches = []
32     for id, row in conllu_s.iteritems():
33         # children (direct branches of header)
34         if (
35             row['head'] == header['id'] and
36             row['upostag'] in subject_object_candidates_pos and
37             row['form'] not in non_subject_object_candidates_form
38         ):
39             direct_branches.append(id)
40
41     # yield triples combinations
42     if len(direct_branches) > 1:
43         for combi in itertools.combinations(direct_branches, 2):
44             first = None
45             third = None
46             if combi[0] < header['id'] and header['id'] < combi[1]:
47                 first = conllu_s[combi[0]]
48                 third = conllu_s[combi[1]]
49             elif combi[1] < header['id'] and header['id'] < combi[0]:
50                 first = conllu_s[combi[1]]
51                 third = conllu_s[combi[0]]
52
53             if first and third:
54                 second = conllu_s[header['id']]
55                 yield (first, second, third)
56
57
58 def extract_triples_by_combinations(conllu_s, header):
59     """
```

```

60     Extract features (triples) for clustering from sentence (conllu_s)
61     by enumerating all possible triple combination of word
62     """
63     num_tokens = len(conllu_s)
64     # sentence start from 1
65     for i in range(1, num_tokens - 2):
66         first = conllu_s[i]
67         # RULES for Subject
68         if (
69             first['upostag'] in subject_object_candidates_pos and
70             first['form'] not in non_subject_object_candidates_form and
71             (first['deprel'] not in ['compound', 'name'] or first['head_distance'] > 2)
72         ):
73             for j in range(i + 1, num_tokens - 1):
74                 second = conllu_s[j]
75                 # RULES for Predicate
76                 if (
77                     second['upostag'] in predicate_candidates_pos
78                 ):
79                     for k in range(j + 1, num_tokens):
80                         third = conllu_s[k]
81                         # RULES for Object
82                         if (
83                             third['upostag'] in subject_object_candidates_pos and
84                             third['form'] not in non_subject_object_candidates_form and
85                             (third['deprel'] not in ['compound', 'name'] or third['head_distance'] > 2) and
86                             (third['upostag'] not in predicate_candidates_pos or first['upostag'] not in
87                             predicate_candidates_pos)
88                         ):
89                             s = first['flatten_s']
90                             p = second['flatten_p']
91                             if third['nearest_adp_id']:
92                                 p += ' ' + conllu_s[third['nearest_adp_id']]['form']
93                             o = third['flatten_o']
94                             yield (first, second, third, s, p, o)
95
96 def extract_triples_by_children_combination(conllu_s, header):
97     """
98     Extract features (triples) for clustering from sentence (conllu_s)
99     by combining sentence predicate nodes with 2 of their children
100     """
101     for k, v in conllu_s.items():
102         # RULES for Subject, Predicate and Object
103         if (
104             v['upostag'] in predicate_candidates_pos and
105             v['form'] not in non_predicate_candidates_form
106         ):
107             for first, second, third in extract_triples_by_root_children(conllu_s, v):
108                 yield (first, second, third)
109
110
111 def trace_children_pos(child_pos_list, parent_pos, node, s):
112     """
113     Find parent that has parent_pos pos tag and has one child of child_pos
114     """
115     parent_pos_list = [parent_pos] if parent_pos not in ['NOUN', 'PROPN'] else ['NOUN', 'PROPN']
116     if node['deprel'] == 'root' or node['upostag'] not in parent_pos_list:
117         return None
118     else:
119         # find child with upostag == child_pos
120         for child_id in node['children']:
121             if s[child_id]['upostag'] in child_pos_list:
122                 return s[child_id]
123
124         # if not found try to search on node's parent
125         return trace_children_pos(child_pos_list, parent_pos, s[node['head']], s)
126
127
128 def remove_token_if_first(field, values, tokens):
129     while (tokens and tokens[0][1][field] in values):
130         tokens.pop(0)
131
132
133 def remove_token_if_last(field, values, tokens):
134     while (tokens and tokens[-1][1][field] in values):
135         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
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 [
169                 dengan perbaikan desain], [Beberapa bulan sebelum] (Rose)
170                 continue
171             expanded.update(expand_node(v, s))
172
173     return expanded
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
189             in sorted_nodes])
190         ids = [k for k, v in sorted_nodes]
191     elif expand_as.lower() in ['p']:
192         text = node['form'] if not mark_head else '({})'.format(node['form'])
193         ids = [node['id']]
194
195     # EXPAND RULES
196     negation_node = [s[c_id] for c_id in node['children'] if s[c_id]['form'].lower() == 'tidak']
197     if negation_node:
198         text = negation_node[0]['form'] + ' ' + text
199         ids = [negation_node[0]['id']] + ids
200
201     return text, ids
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

```

```

210 Flatten subject/object candidates
211 """
212 for k, v in s.iteritems():
213     # get head pos tag
214     s[k]['head_upostag'] = s[v['head']]['upostag'] if v['head'] > 0 else ''
215     # get siblings pos tags
216     before = v['id'] - num_siblings
217     s[k]['before_upostag'] = [s[i]['upostag'] if i > 0 else '' for i in range(before, v['id'])]
218     after = v['id'] + num_siblings + 1
219     s[k]['after_upostag'] = [s[i]['upostag'] if i < len(s) else '' for i in range(after - num_siblings,
220 after)]
221     # get children id
222     if k in children:
223         sorted_children = sorted(children[k])
224         s[k]['children'] = sorted_children
225
226 # loop once more to flatten as children is required
227 for k, v in s.iteritems():
228     if v['upostag'] in subject_object_candidates_pos:
229         s[k]['flatten_s'], s[k]['flatten_s_id'] = flatten_node(s[k], s, expand_as='s', mark_head=mark_head
230 )
231         s[k]['flatten_o'], s[k]['flatten_o_id'] = flatten_node(s[k], s, expand_as='o', mark_head=mark_head
232 )
233
234 # trace ADP node to parents to be inherited
235 if v['head'] > 0:
236     nearest_adp_node = trace_children_pos(['ADP'], v['upostag'], v, s)
237     if nearest_adp_node:
238         s[k]['nearest_adp_id'] = nearest_adp_node['id']
239
240 if v['upostag'] == 'VERB':
241     s[k]['flatten_p'], s[k]['flatten_p_id'] = flatten_node(s[k], s, expand_as='p', mark_head=mark_head
242 )
243
244
245
246 def get_neighbour_upostag(position, token):
247     key = position + '_upostag'
248     if position not in ['before', 'after'] or not token[key]:
249         return postag.index('')
250     return postag.index(token[key][0])
251
252
253 def get_next_upostag(token):
254     return get_neighbour_upostag('after', token)
255
256
257 def get_prev_upostag(token):
258     return get_neighbour_upostag('before', token)
259
260
261 def vectorize(first, second, third):
262     """
263     Convert a triple's member to feature vector
264     """
265     distance_first_second = abs(first['id'] - second['id'])
266     distance_second_third = abs(second['id'] - third['id'])
267     first_is_child_of_second = 1 if first['id'] in second['children'] else 0
268     third_is_child_of_second = 1 if third['id'] in second['children'] else 0
269
270     vector = []
271     vector.append(postag.index(first['upostag']))
272     vector.append(deprel.index(first['deprel']))
273     vector.append(postag.index(first['head_upostag']))
274     vector.append(entity.index(first['entity']))
275     vector.append(len(first['children']))
276     vector.append(distance_first_second)
277     vector.append(first_is_child_of_second)
278     vector.append(get_prev_upostag(first))
279     vector.append(get_next_upostag(first))
280     vector.append(1 if first['nearest_adp_id'] else 0)
281
282     vector.append(postag.index(second['upostag']))
283     vector.append(deprel.index(second['deprel']))
284     vector.append(postag.index(second['head_upostag']))
285     vector.append(entity.index(second['entity']))
286     vector.append(len(second['children']))
287     vector.append(get_prev_upostag(second))
288     vector.append(get_next_upostag(second))

```



```

283     vector.append(postag.index(third['upostag']))
284     vector.append(deprel.index(third['deprel']))
285     vector.append(postag.index(third['head_upostag']))
286     vector.append(entity.index(third['entity']))
287     vector.append(len(third['children']))
288     vector.append(distance_second_third)
289     vector.append(third_is_child_of_second)
290     vector.append(get_prev_upostag(third))
291     vector.append(get_next_upostag(third))
292     vector.append(1 if third['nearest_adp_id'] else 0)
293
294     return vector
295
296
297 def parse_conllu_file(conllu_file, mark_head=False):
298     with open(conllu_file, 'rb') as csvfile:
299         reader = csv.reader(csvfile, delimiter='\t', quoting=csv.QUOTE_NONE)
300         s = {}
301         children = {}
302         s_header = None
303         index = 0
304         for row in reader:
305             if len(row) > 0:
306                 id = int(row[conllu.index('ID')])
307                 head_id = int(row[conllu.index('HEAD')])
308                 deprel = row[conllu.index('DEPREL')].split(':')[0] # ignore sub relation
309                 obj = {
310                     'id': id,
311                     'sentence_id': index,
312                     'form': row[conllu.index('FORM')],
313                     'upostag': row[conllu.index('UPOSTAG')],
314                     'head': head_id,
315                     'head_distance': abs(head_id - id) if head_id > 0 else 0,
316                     'deprel': deprel if deprel != '_' else 'root',
317                     'head_upostag': '',
318                     'before_upostag': [],
319                     'after_upostag': [],
320                     'flatten_s': row[conllu.index('FORM')],
321                     'flatten_p': row[conllu.index('FORM')],
322                     'flatten_o': row[conllu.index('FORM')],
323                     'flatten_s_id': [id],
324                     'flatten_p_id': [id],
325                     'flatten_o_id': [id],
326                     'entity': row[conllu.index('MISC')] if row[conllu.index('MISC')] != '_' else '',
327                     'children': [],
328                     'nearest_adp_id': None
329                 }
330                 s[id] = obj
331                 # map children
332                 if obj['head'] != 0:
333                     if obj['head'] not in children:
334                         children[obj['head']] = []
335                     if id not in children[obj['head']]:
336                         children[obj['head']].append(id)
337                 # find root header
338                 s_header = obj if obj['head'] == 0 else s_header
339             else:
340                 set_extra_properties(s, children, mark_head)
341                 yield index, s, s_header
342                 s = {}
343                 index += 1
344                 children = {}
345             if s:
346                 # if last element not a blank
347                 set_extra_properties(s, children, mark_head)
348                 yield index, s, s_header
349
350
351 def get_best_features():
352     return BEST_FEATURES

```

LAMPIRAN 4: KODE SUMBER PELATIHAN *TRIPLE* *SELECTOR*

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

```
1 import argparse
2 import collections
3 import numpy as np
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.model_selection import GridSearchCV
6 from sklearn.ensemble import RandomForestClassifier
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.neural_network import MLPClassifier
10 from sklearn.externals import joblib
11 from sklearn.model_selection import cross_val_score
12 from sklearn.metrics import precision_recall_fscore_support
13 from tripletools import get_best_features
14 import matplotlib.pyplot as plt
15 import matplotlib.patches as mpatches
16
17
18 plt.style.use('ggplot')
19
20 experiments = [
21     {
22         'name': 'Logistic Regression',
23         'model': LogisticRegression(),
24         'params': [
25             {
26                 'solver': ['liblinear'],
27                 'penalty': ['l2'],
28                 'random_state': [77]
29             },
30         ]
31     },
32     {
33         'name': 'SVM',
34         'model': SVC(),
35         'params': [
36             {
37                 'kernel': ['poly'],
38                 'degree': [5],
39                 'random_state': [77]
40             },
41         ]
42     },
43     {
44         'name': 'MLP',
45         'model': MLPClassifier(max_iter=1000),
46         'params': [
47             {
48                 'hidden_layer_sizes': [(20, 10)],
49                 'random_state': [77]
50             },
51         ]
52     },
53     {
54         'name': 'Random Forest',
55         'model': RandomForestClassifier(),
56         'params': [
57             {
58                 'max_depth': [8],
59                 'n_estimators': [20],
60                 'min_samples_split': [5],
61                 'criterion': ['gini'],
62                 'max_features': ['auto'],
63                 'class_weight': ['balanced'],
```

```

64         'random_state': [77]
65     }
66 ]
67 },
68 ]
69
70
71 def cross_validate_precision_recall_fbeta(model, X, y, cv=None):
72     precision = cross_val_score(model, X, y, cv=cv, scoring='precision').mean()
73     recall = cross_val_score(model, X, y, cv=cv, scoring='recall').mean()
74     fbeta_list = cross_val_score(model, X, y, cv=cv, scoring='f1')
75     fbeta = fbeta_list.mean()
76     fbeta_min = fbeta_list.min()
77     fbeta_max = fbeta_list.max()
78     fbeta_std = fbeta_list.std()
79     return precision, recall, fbeta, fbeta_min, fbeta_max, fbeta_std
80
81
82 def plot_model_performance_comparison(experiments):
83     fig, ax = plt.subplots()
84
85     # Example data
86     x_data = []
87     y_dict = {
88         'precision': {'color': '#f9f1c5', 'data': []},
89         'recall': {'color': 'lightblue', 'data': []},
90         'f1': {'color': 'green', 'data': []},
91     }
92     for exp in experiments:
93         x_data.append(exp['name'])
94         y_dict['precision']['data'].append(exp['best_score']['precision'])
95         y_dict['recall']['data'].append(exp['best_score']['recall'])
96         y_dict['f1']['data'].append(exp['best_score']['f1'])
97
98     x = np.arange(len(x_data))
99     width = 0.20
100     i = 1
101     legend_handles = []
102     for label, y in y_dict.items():
103         ax.bar(x + width * i, y['data'], width, color=y['color'])
104         legend_handles.append(mpatches.Patch(color=y['color'], label=label))
105         i += 1
106     ax.set_xticks(x + width * 2)
107     ax.set_xticklabels(x_data)
108     ax.set_yticks(np.arange(0.0, 1.1, 0.1))
109     ax.set_title('Triple Selector Models Performance')
110
111     lgd = plt.legend(handles=legend_handles)
112     plt.show()
113
114 if __name__ == '__main__':
115     parser = argparse.ArgumentParser(description='Train triples classifier')
116     parser.add_argument('dataset_path', help='Dataset path')
117     parser.add_argument('-o', '--output_path', help='Output model path', default='triples-classifier-model.pkl')
118     parser.add_argument('-s', '--scaler_output_path', help='Output scaler path', default='triples-classifier-scaler.pkl')
119     parser.add_argument('-b', '--best', help='search parameters that gives best model', action='store_true')
120     parser.add_argument('--nocv', help='no cross-validation. training accuracy only', action='store_true')
121     args = parser.parse_args()
122
123     # load dataset
124     dataset = np.genfromtxt(args.dataset_path, delimiter=',', dtype='float32')
125     total_features = dataset.shape[1] - 1
126
127     # feature selection
128     selected_features = get_best_features()
129     print('Total features: {}'.format(total_features))
130     print('Selected features: {} ({}).format(selected_features, len(selected_features)))
131
132     X = dataset[:, selected_features]
133     y = dataset[:, -1]
134     scaler = StandardScaler().fit(X)
135     X = scaler.transform(X)
136     joblib.dump(scaler, args.scaler_output_path)
137
138     # collect dataset statistics

```

```

139 counter = collections.Counter(y)
140 print(counter)
141 pos = counter[1] * 1.0 / (counter[0] + counter[1])
142 neg = 1.0 - pos
143
144 # exhaustive best parameters search
145 cv = None
146 print('')
147 if args.best:
148     best_score = 0.0
149     best_model = None
150     count = 0
151     for experiment in experiments:
152         search = GridSearchCV(
153             estimator=experiment['model'],
154             param_grid=experiment['params'],
155             scoring='f1',
156             cv=cv
157         )
158         search.fit(X, y)
159         if args.nocv:
160             y_pred = search.best_estimator_.predict(X)
161             precision, recall, fbeta, support = precision_recall_fscore_support(y, y_pred, average='binary')
162             fbeta_min = fbeta_max = fbeta_std = fbeta
163         else:
164             precision, recall, fbeta, fbeta_min, fbeta_max, fbeta_std =
165             cross_validate_precision_recall_fbeta(search.best_estimator_, X, y)
166             print(search.best_estimator_)
167             print('Precision: {} \n Recall: {} \n F1 avg: {} \n F1 min: {} \n F1 max: {} \n F1 std: {} \n'.format(
168                 precision,
169                 recall,
170                 fbeta,
171                 fbeta_min,
172                 fbeta_max,
173                 fbeta_std,
174             ))
175             experiment['best_model'] = best_model
176             experiment['best_score'] = {'precision': precision, 'recall': recall, 'f1': fbeta}
177             # replace current best model if the score is higher
178             if search.best_score_ > best_score:
179                 best_score = search.best_score_
180                 best_model = search.best_estimator_
181                 count += 1
182             print('----- Result -----')
183             print('Best models: {} (F1 = {})'.format(best_score, type(best_model).__name__))
184             model = best_model
185
186             # show plot
187             plot_model_performance_comparison(experiments)
188
189 else:
190     model = RandomForestClassifier(max_depth=8, class_weight='balanced', n_estimators=20,
191                                   min_samples_split=5, max_features='auto', random_state=77)
192
193     # cross validate best model to compare score
194     precision, recall, fbeta, fbeta_min, fbeta_max, fbeta_std = cross_validate_precision_recall_fbeta(
195         model, X, y)
196     print('Precision: {} \n Recall: {} \n F1 avg: {} \n F1 min: {} \n F1 max: {} \n F1 std: {} \n'.format(
197         precision,
198         recall,
199         fbeta,
200         fbeta_min,
201         fbeta_max,
202         fbeta_std,
203     ))
204
205 # save model to file
206 joblib.dump(model, args.output_path)
207 print('Model saved to {}'.format(args.output_path))
208 print('Scaler saved to {}'.format(args.scaler_output_path))

```

LAMPIRAN 5: DAFTAR *POS TAG* DAN *DEPENDENCY RELATION* CONLL-U

POS tag

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

Dependency Relation

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

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