BAB 2
LANDASAN TEORI

2.1. Coreference Resolution

Tujuan dari *Coreference Resolution* adalah untuk mengidentifikasi kesetaraan antar entitas serta antara tanda penunjuknya, yaitu kata ganti dan entitas yang dikenali dalam fase *Named Entity Recognition* [3]. *Coreference Resolution* adalah sebagian dari banyaknya integran aplikasi seperti perangkum teks, menjawab dan mengekstraksi informasi terstruktur dari teks [3]. *Coreference Resolution* mencoba untuk mencluster istilah atau frasa yang merujuk pada entitas yang sama. Istilah atau frasa adalah kata ganti atau entitas yang telah dikenali dalam fase *Named Entity Recognition* [3].

Mari perhatikan kutipan kalimat dari novel "Surga Sungsang" berikut untuk mengilustrasikan bagaimana *coreference resolution* bekerja.

Ahmad bin Fulan membeli mobil baru dari Udin bin Fulan. Setelah pembayaran lunas diterima oleh Udin, maka sekarang mobil sudah menjadi miliknya.

Dan di bawah ini bentuk yang sudah dimodifikasi sebagai bentuk untuk memperjelas penggambaran terhadap data masukan.

Ahmad bin Fulan$_1$ *membeli mobil baru dari* ***Udin bin Fulan***$_2$*. Setelah pembayaran lunas diterima oleh* ***Udin***$_3$*, maka sekarang mobil sudah menjadi milik* ***nya***$_4$.

Coreference resolution nantinya harus dapat mengidentifikasi yang mana istilah atau frasa yang menjadi kata ganti terhadap suatu entitas. Proses ini nanti harus dapat mengidentifikasi bahwa ***Udin***$_3$ merujuk kepada ***Udin bin Fulan***$_2$ dan -***nya***$_4$ mengacu kepada ***Ahmad bin Fulan***$_1$. Sehingga sistem harus dapat membuat

dua *cluster*, yaitu {Ahmad bin Fulan$_1$ dan -nya$_4$} dan {Udin bin Fulan$_2$ dan Udin$_3$}.

Meskipun sudah banyak penelitian mengenai *coreference resolution*, namun mayoritas bertumpu pada kasus Bahasa Inggris. Maka, peneliatan ini akan memanfaatkan *coreference resolution* untuk mengidentifikasi relasi atau *coreference* yang terdapat dalam novel berbahasa Indonesia.

2.1. Text Mining

Text mining adalah salah satu bidang yang sampai saat ini masih berkembang dengan pesat, dengan tugasnya dalam mengekstraksi atau mengumpulkan informasi yang bermakna dari teks alami suatu bahasa. Ini dapat diartikan sebagai proses menganalisis suatu teks untuk kemudian diekstrak informasi-informasi yang berguna dari teks tersebut untuk tujuan terentu [8]. Dalam budaya modern, teks adalah salah satu media dalam pertukaran informasi, dibanding dengan *database*, teks tidak terstuktur, memiliki bermacam-macam bentuk, dan lebih sulit ditangani menggunakan algoritma tertentu [8].

Pada kasus ini, yaitu *text mining* sumber data yang berupa teks tidak memiliki struktur yang jelas dan memiliki bermacam bentuk, sehingga disebut sebagai *unstructured data.* Maka dari itu, butuh proses untuk membuat data menjadi lebih terstuktur sehingga ekstraksi informasi dari teks akan lebih mudah, tepat, dan sangat penting dalam proses *text mining*. Sumber data yang digunakan, yaitu novel berbahasa Indonesia merupakan *unstructured data,* sehingga butuh proses untuk membuat data menjadi lebih terstuktur. Salah satunya adalah dengan diawali oleh *preprocessing*, yang mana nanti akan menghasilkan fitur yang lebih represenatif dibanding sumber data novel berbahasa Indonesia yang belum dipersiapkan dan masih tidak berstruktur.

2.2. Preprocessing

Proses *preprocessing* merupakan tahap dimana sumber data masukan diolah kembali sebelum kemudian diproses lebih lanjut dan dijadikan bahan ajar. Pada kasus teks novel berbahasa Indonesia, *text mining* mempunyai sumber data yang masih berbentuk *unstructured data*, sehingga dibutuhkannya proses yang merubah bentuknya menjadi data yang terstruktur. Proses ini nantinya akan

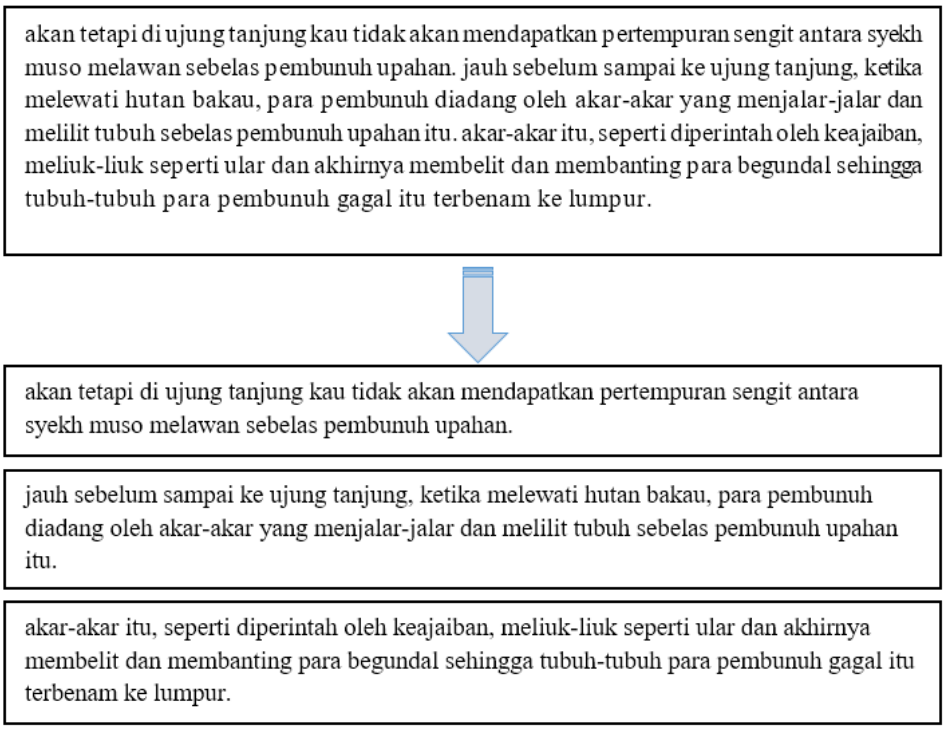
menyeragamkan *case* data masukkan menjadi *lowercase* atau *uppercase* semua, menghilangkan

tanda baca yang tidak diperlukan, kemudian membuat token dari data masukkan tersebut, sehingga data lebih bersih, terstruktur, dan dapat diolah lebih lanjut.

Pada penelitian ini, tahap *preprocessing* yang diterapkan terdiri dari *Tokenizing, POS Tagging, Named Entity, Case Folding, Filtering,* dan *Stemming.*

2.1.1. Tokenisasi Kalimat

Proses tokenisasi adalah proses dimana string dipotong menjadi beberapa bagian dengan melihat *delimiter*nya, seperti tipe kapitalisasi, keberadaan digit, tanda baca, karakter special, dan sebagainya. Pemecahan dokumen menjadi kata-kata tunggal dilakukan dengan cara men-*scan* dokumen dan setiap kata akan teridentifikasi atau terpisahkan dengan kata lainnya oleh *delimiter* [9]. Tokenisasi kalimat adalah proses dimana data *input*-an dibagi menjadi beberapa token sesuai dengan jumlah kalimat menggunakan *delimiter* '.' pada teks *input*-an. Contoh dari tokenisasi kalimat dapat dilihat pada Gambar 2.1 di bawah. Gambar tersebut memperlihatkan data masukan yang merupakan *string* teks yang terdiri dari beberapa kalimat dan dipisahkan oleh '.' pada akhir kalimatnya.



Gambar 2.1 Contoh Tokenisasi Kalimat

2.1.1. Tokenisasi Kata

Tokenisasi kata adalah proses dimana data *input*-an dibagi menjadi beberapa token sesuai dengan jumlah kata menggunakan *delimiter* spasi pada teks *input*-an. Tokenisasi kata merupakan pemecahan kalimat menjadi kata-kata tunggal dan hasil dari tokenisasi akan digunakan dalam tahap transformasi teks. Contoh dari tokenisasi kata dapat dilihat pada Gambar 2.2 di bawah. Gambar tersebut memperlihatkan data masukan yang merupakan *string* kalimat yang terdiri dari beberapa kata dan dipisahkan oleh spasi pada setiap kalimatnya.

akan tetapi di ujung tanjung kau tidak akan mendapatkan pertempuran sengit antara syekh muso melawan sebelas pembunuh upahan

akan	tetapi	di	ujung	tanjung
kau	tidak	akan	mendapatkan	pertempuran
sengit	antara	syekh	muso	melawan
sebelas	pembunuh	upahan.		

Gambar 2.2 Contoh Tokenisasi Kata

2.1.1. POS Tagging

Part of Speech Tagging (POS-Tagging) adalah suatu proses memberikan label pada kelas kata pada suatu kata dalam kalimat secara otomatis serta hasil dari *POS Tagging* ini berpengaruh besar terhadap keluaran dari proses *Parsing* [10].

Sudah banyak penelitian yang dilakukan terhadap *POS Tagging* dengan kasus Bahasa Indonesia. Metode-metode yang digunakan dalam penelitian pun sudah beragam dan memiliki akurasi yang cukup tinggi, diantaranya Brill Tagger dengan akurasi 98%, *Conditional Random Field (CRF)* dan *Maximum Entropy Method* dengan akurasi 97,57%, dan *Hidden Markov Model (HMM)* dengan akurasi sebesar 96,5% [5]. Berikut ini adalah contoh kalimat yang akan ditentukan kelas katanya.

"Saya pergi ke Bandung."

Setelah diproses oleh *POS Tagger*, kemudian setiap token di dalam kata tersebut akan diberikan tag sesuai dengan masing-masing kategorinya menjadi.

"Saya/PRP pergi/VB ke/IN Bandung/NN"

Label PRP digunakan untuk *"Personal Pronoun"* atau kata ganti orang, seperti saya, kami, kita, anda, kamu, mereka, kalian, dan dia. VB digunakan untuk *"Verb"* atau kata kerja, seperti pergi, makan, dan minum. Label IN digunakan untuk "Preposisi" seperti di, ke, dan pada. Dan label NN digunakan untuk kata benda.

Pada penelitian ini, *library* yang digunakan untuk mendeteksi *part-of-speech tag* pada proses *training* dan *testing* adalah menggunakan *library* di dalam *python* bernama *SpaCy* dari *explosion.ai* yang sudah dilatih secara manual dengan korpus data latih yang dihimpun dan dirapikan sendiri oleh penulis, khusus untuk menangani kasus *coreference resolution* dalam novel bahasa indonesia.

2.1.1. Named Entity

Named Entity adalah satu dari beberapa komponen utama dari *Information Extraction* yang bertujuan agar sistem mendeteksi entitas bernama atau *named-entity* pada suatu teks masukan. *Named Entity* umumnya digunakan untuk mendeteksi entitas seperti nama orang, lokasi, organisasi, dan waktu pada suatu dokumen yang menjadi data masukan. Berikut adalah contoh dalam suatu kalimat:

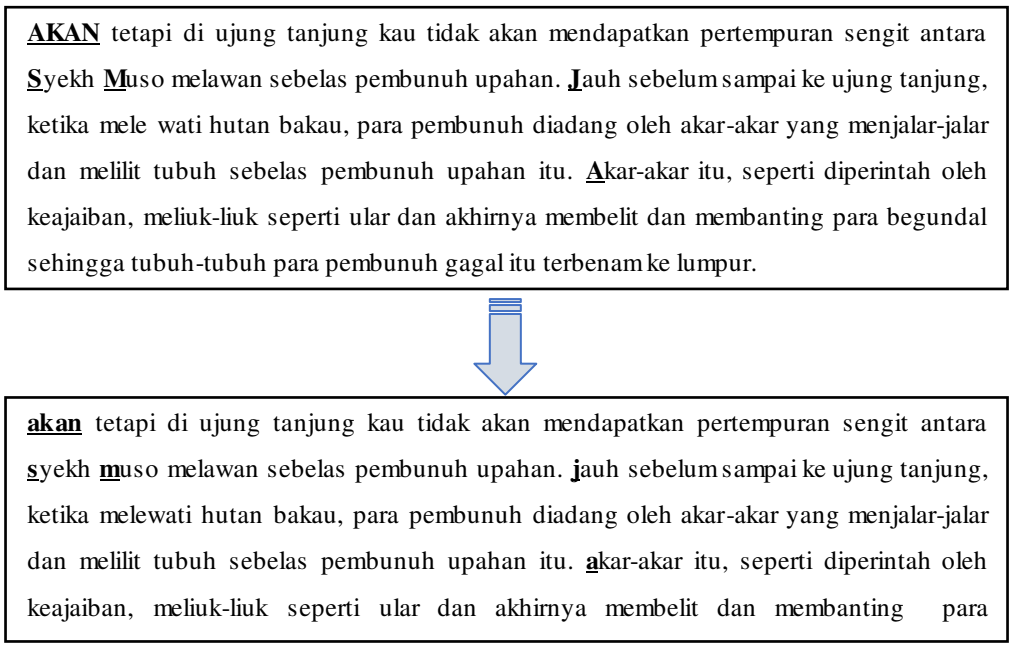
"20.37: Banjir setinggi 60-80 cm di Jln Pegangsaan, sekitar bengkel motor Jakarta, hati-hati bagi pengendara"

Setelah proses *named entity* dijalankan, maka akan didapatkan entitas dari teks tersebut. "20.37" bertipe waktu; "banjir" bertipe kejadian; "60-80 cm" bertipe ukuran; "Jln pegangsaan", "bengkel motor", dan "Jakarta" bertipe lokasi. Dapat dilihat dari contoh di atas bahwa *named entity* akan mendeteksi kemudian mengkategorikan masing-masing kata pada teks tersebut ke dalam tipe yang sesuai. Setelah setiap kata dalam kalimat terkategori dengan label entitas masing-masing, maka selanjutnya setiap entitas akan digabungkan menjadi satu token. Sehingga, "60-80 cm" dan "bengkel motor" yang sebelumnya berjumlah masing-masing 2 token, akan menjadi 1 token, begitu pula entitas lainnya.

Pada penelitian ini, *library* yang digunakan untuk mendeteksi *named entity* pada proses *training* dan *testing* adalah menggunakan *library* di dalam *python* bernama *SpaCy* dari *explosion.ai* yang sudah dilatih secara manual dengan korpus data latih yang dihimpun dan dirapikan sendiri oleh penulis, khusus untuk menangani kasus *coreference resolution* dalam novel bahasa indonesia.

2.1.1. Case Folding

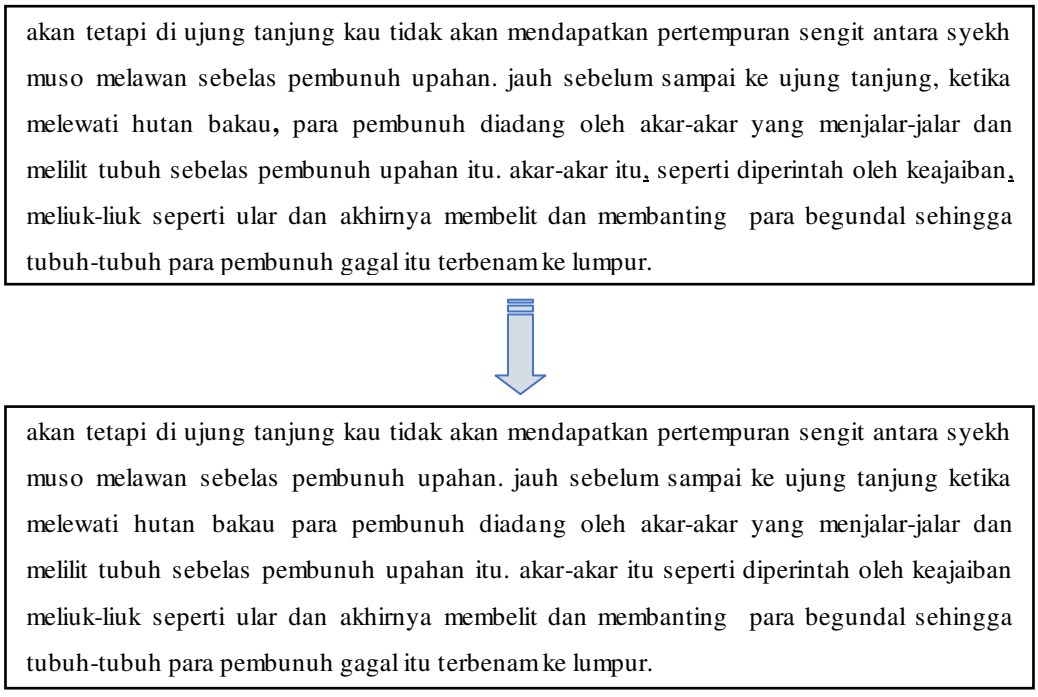
Case folding digunakan untuk menyeragamkan seluruh teks ke dalam case yang seragam, baik itu menjadi huruf kecil (*lowercase*) atau huruf besar (*uppercase*) [8]. *Case folding* yang digunakan pada penelitian ini adalah penyeragaman menjadi *lowercase*. Contoh dari *case folding* dapat dilihat pada Gambar 2.3 di bawah. Gambar tersebut memperlihatkan data masukan yang memiliki beragam tipe *case* kemudian memiliki data keluaran berupa teks yang seluruhnya adalah *lowercase*.



Gambar 2.3 Contoh *Case Folding*

2.1.1. Filtering

Filtering adalah proses menghilangkan elemen yang tidak diperlukan dari dokumen, karena dianggap tidak menjadi bobot utama bagi proses klasifikasi dan hanya akan menambah beban ketika melakukan *training* model dan pengujian. Proses ini memutuskan istilah mana yang harus digunakan untuk merepresentasikan dokumen sehingga dapat menggambarkan keseluruhan dokumen dan dapat membedakan dokumen satu dengan dokumen lainnya pada koleksi [8]. Sehingga selain karakter 'a' sampai 'z', '-', ',', dan spasi akan dihilangkan. Contoh dari *filtering* dapat dilihat pada Gambar 2.4 di bawah. Gambar tersebut memperlihatkan data masukan yang memiliki beragam karakter seperti '.' yang kemudian dihilangkan dan hanya menyisakan karakter 'a' sampai 'z', '-', ',', dan spasi. Sehingga, memiliki data keluaran berupa teks yang seluruhnya adalah hanya mengandung karakter-karakter yang sudah disebutkan.



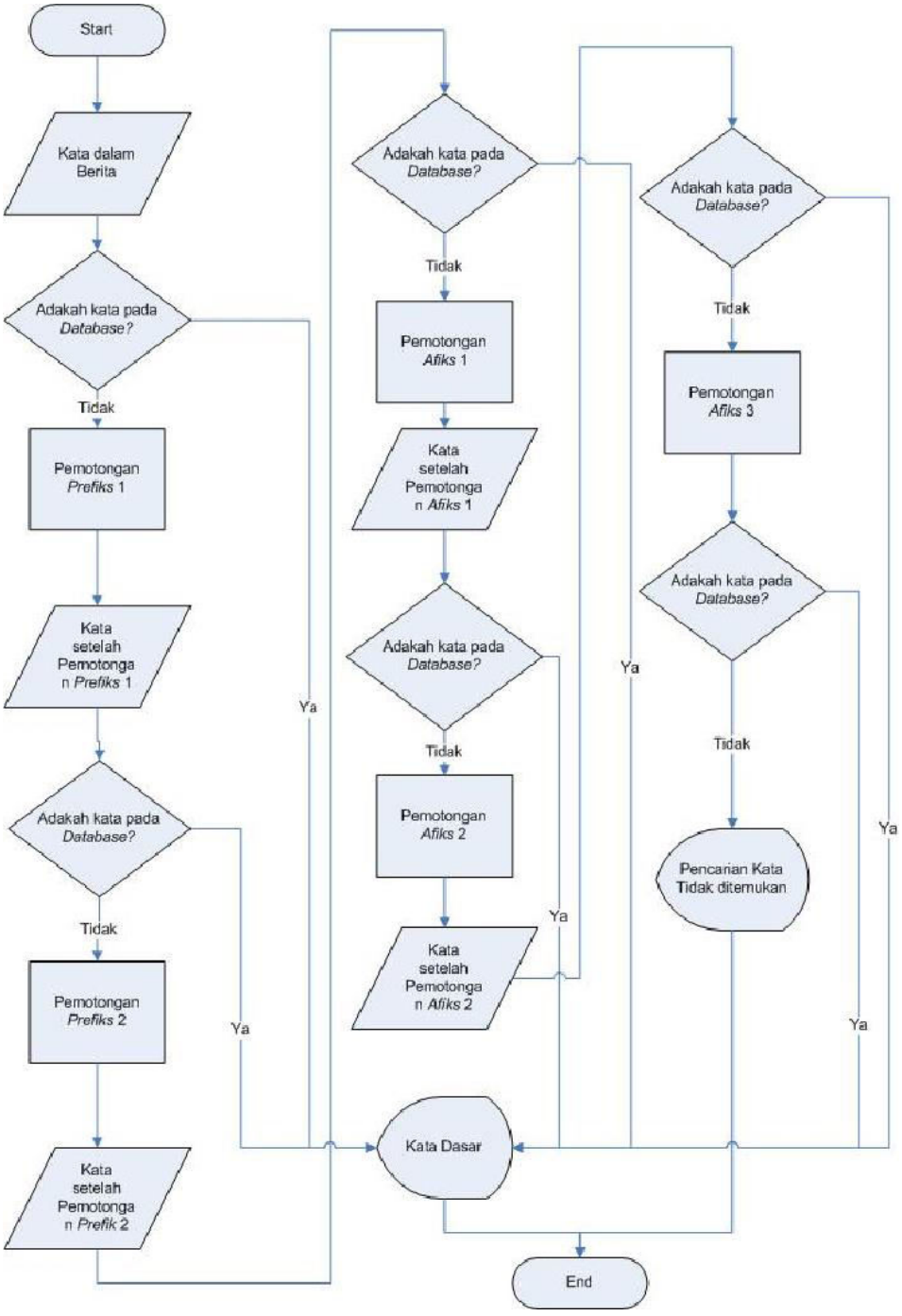
Gambar 2.4 Contoh *Filtering*

2.1.1. Stemming

Stemming merupakan suatu proses yang mentransformasi kata-kata yang berimbuhan ke kata-kata dasarnya (*root word*) [11]. Sebagai contoh, kata "menyenangkan'" dan "menyenangi" dapat dikelompokkan menjadi satu kata dasar, yaitu kata senang. Proses *stemming* yang dilakukan pada dokumen berbahasa Indonesia berbeda dengan Bahasa Inggris dan Bahasa lainnya. Seperti pada Bahasa Inggris proses yang diperlukan untuk *stemming* hanya perlu menghilangkan sufiks atau imbuhan di akhir. Sedangkan, pada dokumen atau teks Bahasa Indonesia, selain sufiks, kita pun perlu untuk mendeteksi prefiks, konfiks, dan juga infiks untuk dihilangkan. Pada umumnya, kata dasar pada Bahasa Indonesia terdiri dari kombinasi:

Prefiks 1 + Prefiks 2 + Kata Dasar + Sufiks 3 + Sufiks 2 + Sufiks 1

Sehingga dapat digambarkan menggunakan *flowchart* seperti Gambar 2.5 di bawah ini:



Gambar 2.5 *Flowchart* algoritma *stemming* Nazief dan Adriani [12]

Pada penelitian ini, salah satu algoritma yang digunakan adalah algoritma Nazief dan Adriani yang terdapat di dalam *library* PySastrawi [13]. Algoritma dari Nazief dan Adriani memiliki aturan imbuhan sendiri dengan model sebagai berikut[5]:

[[[AW+]AW+]AW+] Kata-Dasar [[+AK][+KK][+P]]

Dimana:

AW : Awalan

AK : Akhiran

KK : Kata Ganti kepunyaan

P : Partikel

Algoritma Nazief & Adriani yang dibuat oleh Bobby Nazief dan Mirna Adriani ini memiliki tahap-tahap sebagai berikut [14]:

1. Pertama cari kata yang akan di*stem* dalam kamus kata dasar. Jika ditemukan maka diasumsikan kata adalah *root word*. Maka algoritma berhenti.
2. *Inflection Suffixes* ("-lah", "-kah", "-ku", "-mu", atau "-nya") dibuang. Jika berupa *particles* ("-lah", "-kah", "-tah" atau "-pun") maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* ("-ku", "-mu", atau "-nya"), jika ada.
3. Hapus *Derivation Suffixes* ("-i", "-an" atau "-kan"). Jika kata ditemukan di kamus, maka algoritma berhenti. Jika tidak maka ke langkah 3a.
    a. Jika "-an" telah dihapus dan huruf terakhir dari kata tersebut adalah "-k", maka "-k" juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 3b.
    b. Akhiran yang dihapus ("-i", "-an" atau "-kan") dikembalikan, lanjut ke langkah 4.
4. Hapus *Derivation Prefix*. Jika pada langkah 3 ada sufiks yang dihapus maka pergi ke langkah 4a, jika tidak pergi ke langkah 4b.
    a. Periksa tabel kombinasi awalan-akhiran yang tidak diijinkan. Jika ditemukan maka algoritma berhenti, jika tidak maka pergi ke langkah 4b.

b. *For i = 1 to 3*, tentukan tipe awalan kemudian hapus awalan. Jika *root word* belum juga ditemukan lakukan langkah 5, jika sudah maka

algoritma berhenti. Catatan: jika awalan kedua sama dengan awalan pertama maka algoritma berhenti.

1. Melakukan *Recoding*.
2. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai *root word*. Proses selesai.

Tipe awalan ditentukan melalui langkah-langkah berikut [14]:

1. Jika awalannya adalah: "di-", "ke-", atau "se-" maka tipe awalannya secara berturut-turut adalah "di-", "ke-", atau "se-".
2. Jika awalannya adalah "te-", "me-", "be-", atau "pe-" maka dibutuhkan sebuah proses tambahan untuk menentukan tipe awalannya.
3. Jika dua karakter pertama bukan "di-", "ke-", "se-", "te-", "be-", "me-", atau "pe-" maka berhenti.
4. Jika tipe awalan adalah "none" maka berhenti. Jika tipe awalan adalah bukan "none" maka awalan dapat dilihat pada Tabel 2.1. Hapus awalan jika ditemukan.

Tabel 2.1 Kombinasi awalan akhiran yang tidak diijinkan

Awalan	Akhiran yang Tidak Diijinkan
be-	-i
di-	-an
ke-	-i, -kan
Se-	-i, -kan

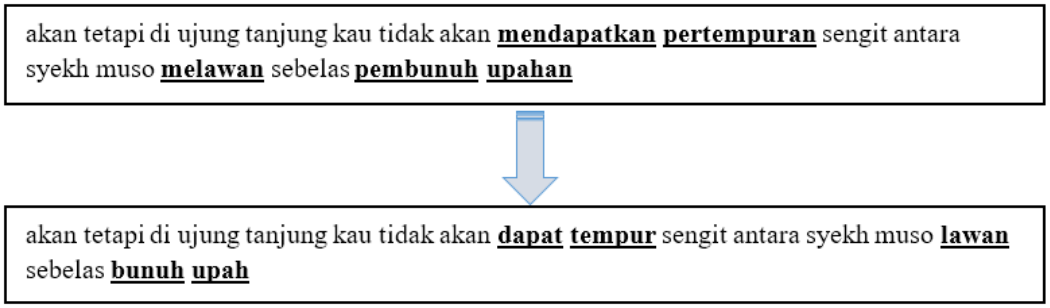
Tabel 2.2 Cara menentukan tipe awalan untuk awalan "te-"

Following Characters				**Tipe Awalan**
Set 1	Set 2	Set 3	Set 4	
'r'	'r'			none
'-r-'	vowel	-	-	ter-luluh
'-r-'	not (vowel or '-r-')	'-er-'	vowel	ter
'-r-'	not (vowel or '-r-')	'-er-'	not vowel	ter-
not (vowel or '-r-')	not (vowel or '-r-')	not '-er-'	-	ter
not (vowel or '-r-')	'-er-'	vowel	-	none
not (vowel or '-r-')	'-er-'	not vowel	-	te

Tabel 2.3 Jenis awalan berdasarkan tipe awalannya

Tipe Awalan	**Awalan yang Harus Dihapus**
di-	di-
ke-	ke-
se-	se-
te-	te-
ter-	ter-
ter-luluh	ter

Sebagai contoh mari kita lihat pada Gambar 2.6 di bawah.



Gambar 2.6 Contoh Tahap *Stemming*

Pada penelitian ini, untuk proses *stemming* menggunakan *library PySastrawi* yang sudah dimodifikasi pada tahap *stemming*, yang kemudian diberi nama *PySastrawiDio*. Secara keseluruhan *library* ini menggunakan algoritma Nazief dan Adriani serta algoritma *Enhanced Confix Stripping Stemmer* yang kemudian dibalut dengan *Object Oriented Design* untuk meningkatkan kualitas kode [13].

2.1. Ekstraksi Fitur

Ekstraksi fitur adalah sebuah proses dimana sumber data masukan diekstraksi karakteristik dan informasinya untuk nantinya dipergunakan ke dalam perhitungan lebih lanjut. Informasi yang diambil berupa informasi penting dari suatu data untuk proses klasifikasi nantinya.

Dalam *Machine Learning* dan untuk membangung modelnya, maka dibutuhkan fitur untuk menentukan apakah dua pasang kata merupakan koreferensi satu dengan yang lainnya atau bukan. Pada peneletian sebelumnya yang berjudul "*Research of Noun Phrase Coreference Resolution*" menerapkan beberapa fitur untuk mendeteksi koreferensi, diantaranya adalah *Distance, String Match, Alias, I-pronoun, J-pronoun, Demonstrative Noun Phrase, Semantic Class, I-propernoun, J-propernoun, I-arg0, I-arg1, J-arg0, J-arg1,* dan *similarity*. Semua fitur nantinya akan diekstrak menjadi nilai yang akan digunakan sebagai ciri untuk tahap pengklasifikasian [6]. Berikut penjelasan singkat dari fitur yang akan digunakan pada penelitian ini.

Fitur untuk mendeteksi kata-i

a. *i-Pronoun (i-pron)* yaitu mendeskripsikan nilai i yang dideteksi oleh *POS tagger* sebagai PRP.
b. *i-Propernoun Phrase (i-prop)* yaitu mendeskripsikan nilai i yang dideteksi *POS tagger* sebagai NNP.

Fitur untuk mendeskripsikan kata-j

a. *j-Pronoun (j-pron)* yaitu mendeskripsikan nilai j yang dideteksi oleh *POS tagger* sebagai PRP.
b. *j-Propernoun Phrase (j-prop)* yaitu mendeskripsikan nilai j yang didteksi *POS tagger* sebagai NNP.

Fitur untuk relasi antar kata-i dan kata-j

a. *Distance of Words (DOW)* yaitu mendeskripsikan jarak antar kata i dengan kata j, nilai ini akan didapat dari jarak antara kata-j dikurangi jarak kata-i lalu dibagi dengan jumlah kata pada dokumen teks tersebut.

$$DOW = \frac{jarak\ kata\ i - jarak\ kata\ j}{jumlah\ kata\ dalam\ dokumen\ teks}$$

b. *Distance of Sentence (DOS)* yaitu mendeskripsikan jarak antar kalimat i dengan jakar antar kalimat j, nilai ini akan didapat dari jarak kalimat j dikurangi jarak kalimat i, lalu dibagi dengan jumlah kalimat pada dokumen teks tersebut.

$$DOS = \frac{jarak\ kalimat\ i - jarak\ kalimat\ j}{jumlah\ kalimat\ dalam\ dokumen\ teks}$$

c. *String Match* yaitu mendeskripsikan string pada kata-i dan kata-j,

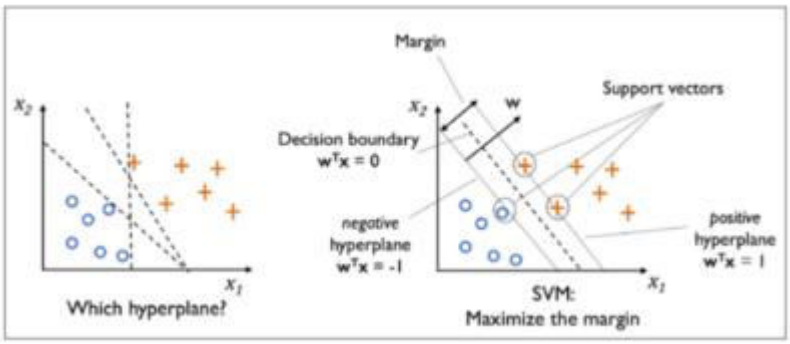
$$String\ match = \begin{cases} 1, kata\ i = kata\ j \\ 0, kata\ i \neq kata\ j \end{cases}$$

Perhatikan contoh di bawah ini

1. Jika kata-I = “dia” dan kata-j = “dia”, maka nilai *String Match* = 1.
2. Jika kata-I = “makan” dan kata-j = “dia”, maka nilai *String Match* = 0.

2.1. Support Vector Machine

Support Vector Machine (SVM) merupakan salah satu metode *Machine Learning* yang bertujuan untuk mengubah data teks menjadi *vector*. *SVM* populer digunakan untuk klasifikasi dan regresi. Konsep *SVM* secara sederhana dapat dikatan sebagai usaha untuk mencari *hyperplane* terbaik yang nantinya berfungsi sebagai pemisah dua buah kelas pada input *space, hyperplane* pemisah dapat ditemukan dengan mengukur *margin*, *hyperplane* adalah garis batas pemisah data antar kelas, sedangkan margin adalah jarak terdekat antara *hyperplane* dengan data dari masing-masing kelas. *Subset data training* yang paling dekat disebut dengan *Support Vector* [15]. Berikut adalah ilustrasi dari metode *SVM* pada Gambar 2.7.



Gambar 2.7 Ilustrasi *Hyperplane*

Dapat dilihat beberapa data yang merupakan anggota dari dua buah kelas data, yaitu +1 dan -1. Data yang tergabung pada kelas -1 disimbolkan dengan bentuk lingkaran, sedangkan data pada kelas +1 disimbolkan dengan tanda tambah. Garis putus-putus pada skema sebelah kanan menunjukkan *hyperplane* terbaik, yaitu yang terletak tepat ditengah kedua kelas, sedangkan data lingkaran dan data tanda tambah yang dilewati garis batas margin (garis solid) adalah

support vector. Usaha dalam mencari lokasi *hyperplane* terbaik ini merupakan inti dari proses pelatihan menggunakan metode *SVM*.

Data yang tersedia dinotasikan sebagai $\overrightarrow{x_i} \in R^d$ sedangkan label masing-masing dinotasikan $y_i \in \{-1, +1\}$ untuk i = 1,2,… ,l , yang mana l adalah banyaknya data. Diasumsikan kedua kelas -1 dan +1 dapat terpisah secara sempurna oleh *hyperplane* berdimensi d, yang didefinisikan:

$$\overrightarrow{w_i}.\overrightarrow{x_i} + b = 0 \text{ …… (2.1)}$$

Pattern i yang ternasuk kelas -1 (sampel negatif) dapat dirumuskan sebagai pattern yang memenuhi pertidaksamaan:

$$\overrightarrow{w_i}.\overrightarrow{x_i} + b \leq -1 \text{ …… (2.2)}$$

Sedangkan pattern i yang termasuk kelas +1 (sampel positif) memenuhi pertidaksamaan:

$$\overrightarrow{w_i}.\overrightarrow{x_i} + b \geq +1 \text{ …… (2.3)}$$

Margin terbesar dapat ditemukan dengan memaksimalkan nilai jarak antara *hyperplane* dan titik terdekatnya, yaitu $1/\|\vec{w}\|$. Hal ini dapat dirumuskan sebagai *Quadratic Programming (QP) Problem*, yaitu mencari titik minimal persamaan (2.4) dengan memperhatikan *constraint* persamaan (2.8).

$$min\ \tau\ (w) = \frac{1}{2}\ \|\vec{w}\| \text{ …… (2.4)}$$

$$y_i(x_i.w + b) - 1 \geq 0, \forall_i \text{ …… (2.5)}$$

Permasalahan ini dapat dipecahkan dengan berbagai teknik komputasi, diantaranya dengan *Lagrange Multiplier*.

$$L(w, b, \alpha) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i\big((x_i.w_i + b) - 1\big)\right)$$

$$(i = 1,2,3, \dots, l) \text{ …… (2.6)}$$

α_i adalah *Lagrange Multiplier*, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan (2.6) dapat dihitung dengan meminimalkan L terhadap $\vec{w}$ dan b, dan memaksimalkan L terhadap α_i. Dengan memperhatikan sifat bahwa pada titik optimal gradient L = 0, persamaan (2.6) dapat dimodifikasi sebagai maksimalisasi *problem* yang hanya mengandung α_i, sebagaimana persamaan (2.7) berikut.

Maksimasi:

$$\sum_{i=1}^{l} \alpha_i - \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j x_i x_j \ldots\ldots\ldots (2.7)$$

Dengan *Constraint*:

$$\alpha_i \geq 0 (i = 1,2,3, \ldots, l) \sum_{i=1}^{l} \alpha_i y_i \ldots\ldots\ldots (2.8)$$

Dari perhitungan ini diperoleh α_i yang kebanyakan bernilai positif. Data yang berkolerasi dengan α_i yang positif inilah yang disebut *support vector*. [16]

Penjelasan di atas berdasarkan asumsi bahwa kedua belah kelas dapat terpisah secara sempurna oleh *hyperplane*. Akan tetapi, pada umumnya dua belah kelas pada *input space* tidak dapat terpisah secara sempurna (*non linear separable*). Hal ini menyebabkan *constraint* pada persamaan (2.8) tidak dapat terpenuhi, sehingga optimisasi tidak dapat dilakukan. Untuk mengatasi masalah ini, SVM dirumuskan dengan memperkenalkan teknik *softmargin*.

Dalam *softmargin*, persamaan (2.5) dimodifikasi dengan memasukkan *slack variable* $\zeta_i (\zeta > 0)$ sebagai berikut:

$$y_i (x_i . w + b) \geq 1 - \zeta_i \forall_i \ldots\ldots\ldots (2.9)$$

Dengan demikian persamaan (2.4) diubah menjadi:

$$min\ \tau\ (w) = \frac{1}{2} \|\vec{w}\|^2 + \mathrm{C} \sum_{i=1}^{l} \zeta_i \ldots\ldots\ldots (2.10)$$

Parameter C dipilih untuk mengontrol *tradeoff* antara *margin* dan *error* klasifikasi ζ. Nilai C yang besar berarti akan memberikan *penalty* yang lebih besar terhadap eror klasifikasi tersebut. [16]

Untuk menyelesaikan *problem non-linear*, *SVM* dimodifikasi dengan memasukkan fungsi kernel. Dalam *non-linear SVM*, pertama-tama data dipetakan oleh fungsi $\Phi(\vec{x})$ ke ruang vektor yang berdimensi lebih tinggi. *Hyperplane* yang memisahkan kedua kelas tersebut dapat dikontruksikan. Selanjutnya fungsi Φ *(trick-phi*) memetakan tiap data pada *input space* tersebut ke ruang vektor baru yang berdimensi lebih tinggi (dimensi 3), sehingga kedua kelas dapat dipisahkan secara *linear* oleh sebuah *hyperplane.*

Selanjutnya proses pembelajaran pada SVM dalam menemukan titik-titik *support vector*, hanya bergantung pada *dot product* dari data yang sudah ditransformasikan pada ruang baru yang berdimensi lebih tinggi, yaitu $\Phi(\overrightarrow{x_i}).\Phi(\overrightarrow{x_j})$. Karena umumnya transformasi Φ ini tidak diketahui, dan sangat sulit untuk dipahami secara mudah, maka perhitungan *dot product* dapat digantikan dengan fungsi *kernel* $K(\overrightarrow{x_i},\overrightarrow{x_j})$ yang mendefinisikan secara implisit transformasi Φ. Hal ini disebut sebagai *Kernel Trick*, yang dirumuskan sebagai berikut.

$$K(\overrightarrow{x_i},\overrightarrow{x_j}) = \Phi(\overrightarrow{x_i}).\Phi(\overrightarrow{x_j}) \text{ ..... (2.12)}$$

$$f(\Phi(\vec{x})) = \vec{w}.\Phi(\vec{x}) + b \text{ ..... (2.13)}$$

$$= \sum_{i=1,X\ i\in SV}^{n} \alpha_i y_i \Phi(\vec{x}).\Phi(\overrightarrow{x_i}) + b \text{ ..... (2.14)}$$

$$= \sum_{i=1,X\ i\in SV}^{n} \alpha_i y_i K(x, x_i) + b \text{ ..... (2.15)}$$

Syarat sebuah fungsi untuk menjadi fungsi *kernel* adalah memenuhi teorema Mercer yang menyatakan bahwa matriks kernel yang dihasilkan harus bersifat *positive semi-definite*. Fungsi *kernel* yang umum digunakan adalah sebagai berikut:

a. Kernel Linear

$K(x, y) = x.y$ ..... (2.16)

b. Polynomial

$K(x, y) = (x.y + c)^d$ ..... (2.17)

c. Radial Basis Function (RBF)

$K(\vec{x}, \vec{y}) = exp\left(-\frac{\|\vec{x}-\vec{y}\|^2}{2\sigma^2}\right)$ ..... (2.18)

d. Sigmoid

$K(\vec{x}, \vec{y}) = \tan(\mathrm{K} < \vec{x}.\vec{y} > +\vartheta)$ ..... (2.19)

Pada penelitian ini, kernel yang digunakan adalah *kernel Radial Basis Function (RBF)*. Bentuk *primal form* yang tadinya sangat susah untuk dipecahkan, akan dirubah ke dalam bentuk *dual form* yang hanya akan mengandung nilai α. Berbagai algoritma telah dikembangkan untuk mencari nilai α tersebut. Akan tetapi, algoritma-algoritma tersebut memerlukan waktu yang lama, apalagi jika dipakai untuk data yang berukuran besar, karena algoritma tersebut menggunakan *numerical quadratic programming* sebagai *inner loop*. [16]

Oleh karena itu, muncul algoritma-algoritma yang dapat menangani masalah pemecahan nilai α dalam proses *training* tersebut. Salah satunya dengan metode sekuantial.

2.1. Akurasi

Pada penelitian ini tingkat akurasi dari klasifikasi *coreference resolution* dengan metode SVM dihitung dengan rumus sebagai berikut.

$$Akurasi = \frac{Jumlah\ data\ yang\ terklasifikasi\ dengan\ benar}{Jumlah\ sampel\ dari\ data\ uji} \times 100\% \text{ ..... (2.20)}$$

2.2. Black Box Testing

Pengujian sistem adalah pengujian program perangkat lunak yang lengkap dan terintegrasi. Pengujian perangkat lunak dapat dibagi menjadi dua, yaitu *white box*

dan *black box*. *Black box testing* merupakan metode pengujian perangkat lunak yang digunakan untuk menguji perangkat lunak tanpa perlu mengetahui struktur internal dari perangkat lunak tersebut, sehingga *Black box* dikenal juga dengan pengujian fungsional.

Pengujian *Black box* memiliki kelebihan seperti:

a. Akses kode tidak diperlukan.
b. Efisien untuk segmen kode yang relatif besar.
c. Pemisahan antara perspektif pengguna dan pengembang.

2.1. Pemodelan Sistem

Pemodelan sistem merupakan proses pengembangan model abstrak dari sistem, di mana persyaratan non-fungsional dibahas, dengan tiap model memperlihatkan pandangan berbeda dari sistem. Kemudian yang digunakan dalam pemodelan sistem pada penelitian tugas akhir ini adalah sebagai berikut.

2.1.1. Flowchart

Flowchart adalah penggambaran secara grafik dari langkah-langkah dan urutan prosedur dari suatu program. *Flowchart* menolong analis dan programmer untuk memecahkan masalah ke dalam segmen-segmen yang lebih kecil dan menolong dalam menganalisis alternatif-alternatif lain dalam pengoperasian. *Flowchart* akan mempermudah penyelesaian suatu masalah khususnya masalah yang perlu dipelajari dan dievaluasi lebih lanjut. [5]

Dalam penulisan *Flowchart* dikenal dua model, yaitu Sistem *Flowchart* dan Program *Flowchart*.

1. Sistem *Flowchart* yaitu bagan yang memperlihatkan alur kerja yang sedang dilakukan di dalam sistem secara keseluruhan. Sistem *flowchart* menjelaskan urutan dari prosedur-prosedur yang ada di dalam suatu sistem. Artinya, *Flowchart* ini merupakan deskripsi secara grafik dari urutan prosedur-prosedur yang terkombinasi dan membentuk suatu sistem.
2. *Flowchart* Sistem menghasilkan program *flowchart*, atau keterangan yang lebih rinci tentang setiap langkah program atau prosedur yang dilaksanakan. *Flowchart*

program digunakan untuk menggambarkan urutan pekerjaan yang dilakukan dalam suatu prosedur.

2.1.1. Diagram Konteks

Diagram konteks adalah diagram yang terdiri dari proses-proses yang menggambarkan ruang lingkup sistem. Diagram konteks adalah level tertinggi dari DFD, yang menggambarkan semua masukan atau keluaran dari sistem. Dia akan memberikan gambaran umum tentang keseluruhan sistem. Sistem dibatasi oleh batas-batas (yang dapat diwakili oleh garis putus-putus). Dalam diagram konteks, hanya ada satu proses, dan tidak ada yang harus disimpan dalam diagram konteks.[2]

2.1.2. Data Flow Diagram

Data Flow Diagram (DFD) adalah representasi grafis yang menggambarkan arus informasi dan transformasi informasi yang digunakan sebagai data dari input dan output. DFD adalah alat untuk merancang sistem yang berorientasi pada aliran data dengan konsep yang dapat didekomposisi [5]. Berikut ini adalah komponen-komponen dari *Data Flow Diagram*.

a. Komponen *Terminator* atau Entitas Luar

   Terminator menunjukkan entitas eksternal yang berkomunikasi dengan sistem yang sedang dikembangkan.

b. Komponen Proses

   Komponen proses adalah deskripsi dari bagian sistem yang mengubah *input* menjadi *output*.

c. Komponen *Data Store*

   Komponen ini digunakan untuk memodelkan paket data atau kumpulan penyimpanan data dalam bentuk *file* atau *database* pada sistem komputer.

d. Komponen *Data Flow* atau Arus Data

   Suatu *data flow* atau Aliran data diwakili oleh panah yang menunjukkan arah masuk dan keluarnya proses. Aliran data digunakan untuk

menjelaskan perpindahan data atau informasi dari satu bagian sistem ke bagian lain.

2.1. Bahasa Pemrograman

Bahasa pemrograman adalah kumpulan-kumpulan kode yang selanjutnya disebut dengan *syntax* dan identik dengan masing-masing bahasa pemrograman. Bahasa pemrograman ditulis sesuai dengan struktur, metode, dan kaidah yang dimiliki oleh masing-masing bahasa pemrograman. Komputer kemudian akan menerjemahkannya ke dalam bahasa yang dimengerti oleh mesin atau biasa disebut dengan angka biner.

Pada penelitian kali ini bahasa pemrograman yang akan digunakan adalah bahasa pemrograman Python, karena *environment* kerja dari Python sendiri sangat cocok untuk manipulasi data seperti untuk *Machine Learning, Big Data, Data Mining*, dan keperluan *Data Science* lainnya.

2.1.1. Python

Python merupakan salah satu Bahasa pemrograman yang popular di kalangan ilmu komputer, khususnya untuk bidang *Artificial Intelligence* dan juga *Machine Learning*. Karena, kemudahan *syntax* yang dimiliki oleh *Python* dan juga kecepatan serta performa untuk memproses data, maka *Python* dipilih menjadi bahasa pemrograman utama dalam penelitian ini. Beberapa *library* yang digunakan untuk menunjang produktivitas penelitian ini antara lain adalah *Numpy, Pandas, SpaCy, PySastrawiDio* (*PySastrawi* yang sudah dimodifikasi di tahap *stemming), sklearn* atau *Sci-kit learn, Matplotlib, NLTK, pickle,* dan *itertools.*

Pada penelitian ini, versi *Python* yang digunakan adalah *Python* versi 3.7.9 yang akan digunakan sebagai bahasa pemrograman utama dalam membangun sistem untuk *Coreference Resolution* ini.

2.2. Software Pendukung

Pada penelitian ini dibutuhkan beberapa *software* pendukung lainnya. Berikut ini adalah *software* pendukung yang dibutuhkan.

2.2.1. Web Browser

Web browser adalah *software* yang bisa menjelajah, menyajikan, maupun mengambil konten-konten, baik itu konten teks, gambar, hingga video yang ada di berbagai sumber informasi pada jaringan

World Wide Web (WWW) atau biasa disebut sebagai internet. Terdapat banyak pilihan dari *web browser* itu sendiri, di antaranya adalah Google Chrome, Mozilla Firefox, Opera, Safari, dan *web browser* lainnya.

Pada penelitian ini *web browser* yang akan digunakan untuk *training* dan *testing* dari *coreference resolution* ini adalah Google Chrome.

2.1.1. Jupyter Notebook

Jupyter adalah organisasi *non-profit* untuk mengembangkan *software* interaktif dalam berbagai bahasa pemrograman. *Notebook* adalah satu *software* buatan *Jupyter*, adalah aplikasi *web open-source* yang memungkinkan Anda membuat dan berbagi dokumen interaktif yang berisi kode *live*, persamaan, visualisasi, dan teks naratif yang kaya. [17]

Pada penelitian ini, *Jupyter Notebook* digunakan sebagai salah satu *text editor* untuk menuliskan kode-kode program *Python* serta memvisualisasikan data hasil olah pada sistem ini.

2.1.2. Visual Studio Code

VSCode adalah sebuah *text editor* yang bersifat *open source* yang dibuat oleh Microsoft untuk sistem operasi *multiplatform*, yang artinya tersedia untuk Linux, Mac, dan Windows. *Text editor* ini sangat andal dan ringan serta relatif *friendly* dalam penggunaannya. VSCode secara langsung mendukung bahasa pemrograman yang sangat banyak, di antaranya adalah *PHP, Javascript, Python,* dan bahasa pemrograman lainnya. Selain bahasa pemrograman, *text editor* ini juga mendukung bahasa markup seperti *HTML* dan lainnya.

Pada penelitian ini, VSCode dipergunakan sebagai *text editor* untuk menuliskan beberapa kode program *Python* sebagai alteranatif dari Jupyter Notebook pada Web Browser.