BAB II
LANDASAN TEORI

2.1 Struktur *Morfologi* Bahasa Indonesia

Morfologi adalah ilmu yang mempelajari seluk beluk kata serta pengaruh perubahan-perubahan bentuk kata terhadap golongan dan arti kata, atau dengan kata lain dapat dikatakan bahwa *morfologi* mempelajari seluk-beluk bentuk kata serta fungsi perubahan-perubahan bentuk kata itu (Ramlan, 1997). *Morfologi* kata bahasa Indonesia bisa terdiri dari struktur *infleksional* dan *derivasional*. *Infleksional* adalah struktur yang paling sederhana yang dinyatakan dalam penambahan *sufiks* dimana tidak mempengaruhi arti sebenarnya dari kata dasar yang dilekati (Tala, 2003). *Sufiks infleksional* dapat dibagi menjadi dua jenis yaitu:

1. *Sufiks* (*-lah, -kah, -pun, -tah*). *Sufiks* ini sebenarnya adalah partikel yang tidak mempunyai arti. Keberadaannya pada suatu kata adalah untuk penekanan. Contoh :
   dia + kah → diakah
   duduk + lah → duduklah
2. *Sufiks* (*-ku, -mu, -nya*). *Sufiks* ini berfungsi sebagai kata ganti kepunyaan. Contoh :
   tas + ku → tasku
   buku + mu → bukumu

Sufiks-sufiks diatas dapat melekat pada kata dasar secara bersama-sama. Adapun aturan urutannya adalah *sufiks* pada jenis kedua selalu diletakkan sebelum *sufiks* jenis pertama. Sehingga struktur *morfologi* pada kata *infleksional* adalah : *Infleksional* = (kata dasar +kata ganti) | (kata dasar + partikel) | (kata dasar + kata ganti + partikel)

Penambahan *sufiks infleksional* tidak akan merubah bentuk dasar dari kata berimbuhan. Dengan kata lain, tidak ada penghilangan atau peleburan kata dasar pada kata berimbuhan. Kata dasar dapat ditentukan dengan mudah

pada struktur *infleksional*. Struktur *derivasional* dalam bahasa Indonesia terdiri dari *prefiks, sufiks* dan kombinasi dari keduanya. *Prefiks* yang sering dipakai adalah : *ber-, di-, ke-, meng-, peng-, per-, ter-*. Contoh penggunaan *prefiks* adalah :

ber + lari → berlari

di + makan → dimakan

ke + kasih → kekasih

meng + ambil → mengambil

peng + atur → pengatur

per + lebar → perlebar

ter + baca → terbaca

Beberapa *prefiks* seperti *ber-*, *meng-*, *peng-*, *per-*, *ter-* mungkin akan berubah menjadi beberapa bentuk yang berbeda. Bentuk dari setiap *prefiks* bergantung pada karakter pertama dari kata dasar yang dilekatinya. Tidak seperti struktur *infleksional*, pada struktur *derivasional* pengucapan kata mungkin berubah setelah adanya penambahan *prefiks*. Seperti contoh menyapu yang terdiri dari *prefiks meng-* dan kata dasar sapu. *Prefiks* meng- berubah menjadi *meny-* dan karakter pertama dari kata dasar mengalami peleburan. *Sufiks derivasional* adalah *-i, -kan, -an*. Contoh penggunaan *sufiks derivasional* adalah :

gula + i → gulai

makan + an → makanan

sampai + kan → sampaikan

Berbeda dengan penggunaan *prefiks*, penambahan *sufiks* tidak akan mengubah bentuk dasar dari suatu kata. Seperti disebutkan sebelumnya, struktur *derivasional* juga terdiri dari *konfiks*, yaitu gabungan dari *prefiks* dan *sufiks* yang melekat secara bersama-sama pada suatu kata. Contoh :

Per + main + an → permainan

Ke + kalah + an → kekalahan

Ber + jatuh + an → berjatuhan

Meng + ambil + i → mengambili

Struktur *morfologi* pada kata *derivasional* adalah : *Derivasional* = (*prefiks* + kata dasar) | (kata dasar + *sufiks*) | (*prefiks* + kata dasar + *sufiks*) | (*prefiks* 1 + *prefiks* 2 + kata dasar) | (*prefiks* 1 + *prefiks* 2 + kata dasar + *sufiks*).

Struktur lain yang mungkin terjadi dalam *morfologi* bahasa Indonesia adalah penambahan *sufiks infleksional* pada struktur *derivasional*, yang dinamakan *multiple suffiks* atau *sufiks* bertingkat. Sehingga dapat disimpulkan secara umum struktur *morfologi* kata bahasa Indonesia adalah : Struktur *morfologi* = [*prefiks* 1] + [*prefiks* 2] + kata dasar + [*sufiks*] + [kata ganti] + [partikel].

2.2 *Prepocessing*

Prepocessing merupakan proses awal yang akan mengubah data masukan menjadi data yang sesuai dan siap untuk diproses. *Prepocessing* terdiri dari *case folding, tokenizing, stopword removal* dan *stemming*.

2.2.1 *Case Folding*

Case folding adalah proses penyamaan huruf dengan mengubahnya menjadi huruf kecil (*lowercase*). Pada kata yang dimasukkan terdapat huruf besar (kapital) dan kecil. Perbedaan bentuk huruf pada kata tersebut akan menganggu proses selanjutnya maka dari itu diperlukan proses *case folding* untuk merubah kata menjadi huruf kecil semua.

2.2.2 Pemisahan rangkaian kata (*Tokenization*)

Tahap *tokenizing* yakni tahap pemotongan string inputan berdasarkan kata yang menyusunnya. Pada dasarnya proses *tokenizing* adalah pemenggalan kalimat menjadi kata.

2.2.3 Penyaringan (*Filtration*) / *Stopword removal*

Penyaringan atau *Filtration* merupakan pengambilan ‘kata’ penting dari hasil token, menggunakan algoritma *stoplist* (membuang kata yang kurang penting) atau *wordlist* (menyimpan kata penting). ‘*Stoplist*’ atau ‘*Wordlist*’ adalah kata-kata yang tidak deskriptif yang dapat dibuang dalam pendekatan *bag of word*. Kata yang tidak penting adalah hasil parsing dicek

dengan kamus (kumpulan kata) *stopword*. Jika parsing ada yang sama dengan *stopword* maka akan dibuang atau dihapus.

2.2.4 *Stemming*

Stemming merupakan sebuah cara untuk menghilangkan imbuhan seperti berupa awalan dan akhiran agar didapatkan kata dasar (*root word*) (Novitasari, 2016). Proses *stemming* memiliki aturan atau *algoritma* yang berbeda dalam menghilangkan imbuhan disetiap bahasanya, seperti bahasa inggris memiliki perbedaan aturan penggunaan tata bahasa dengan bahasa indonesia.

2.2.4.1 *Stemming* Nazief & Adriani

Algoritma Nazief & Adriani memperhatikan kemungkinan adanya partikel-partikel yang mungkin mengikuti suatu kata berimbuhan. Sehingga kita dapat melihat pada rumus untuk *algoritma* ini yaitu adanya penempatan *possesive pronoun* dan juga partikel yang mungkin ada pada suatu kata berimbuhan (Agusta, 2009).

Algoritma Nazief & Adriani yang dibuat oleh Bobby Nazief dan Mirna Adriani ini memiliki tahap-tahap sebagai berikut (Agusta, 2009) :

1. Cari kata yang akan di*stemming* dalam kamus kata dasar. Jika ditemukan maka diasumsikan kata adalah kata dasar. Maka *algoritma* berhenti.
2. Hilangkan *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”). Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “-nya”), jika ada.
3. Hilangkan *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. Hilangkan *Derivation Prefix* (“di-”, “ke-”, “se-”, “me-”, “be”, “pe-”, “te-”) dengan iterasi maksimum sebanyak 3 kali .
   a. Langkah 4 berhenti jika :
      1. Terjadi kombinasi awalan dan akhiran.
      2. Awalan yang dideteksi saat ini sama dengan awalan yang dihilangkan sebelumnya.
      3. Tiga awalan telah dihilangkan.

   Tabel 2.1 Kombinasi Awalan-Akhiran yang tidak diizinkan Awalan Akhiran yang tidak diizinkan

Awalan	**Akhiran yang tidak diizinkan**
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan
te-	-an

   b. Identifikasi tipe awalan dan hilangkan. Tipe awalan ada 2 yaitu:
      1. Standart : “di-”, “ke-”, “se-” dapat langsung dihilangkan dari kata
      2. Kompleks : “me-”, “be”, “pe-”, “te-” adalah tipe awalan yang dapat *bermorfologi* sesuai kata dasar yang mengikutinya. Oleh karena itu, gunakan aturan pada Tabel 2.1 untuk mendapatkan pemenggalan yang tepat.
   c. Cari kata yang telah dihilangkan awalannya ini di dalam kamus. Apabila tidak ditemukan, maka langkah 4 diulangi kembali. Apabila ditemukan maka keseluruhan proses selesai.
5. Apabila setelah langkah 4 kata dasar masih belum ditemukan, maka proses *recoding* dilakukan dengan mengacu pada aturan pada Tabel 2.3 *Recoding* dilakukan dengan menambahkan karakter *recoding* di awal

kata yang dipenggal. Karakter *recoding* adalah huruf kecil setelah tanda hubung (‘-’) dan terkadang berada sebelum tanda kurung. Sebagai contoh, kata “menangkap” (aturan 15), setelah dipenggal menjadi “nangkap”. Karena tidak *valid*, maka *recoding* dilakukan dan menghasilkan kata “tangkap”.

Tabel 2.2 Aturan Pemenggalan awalan *Stemmer* Nazief & Adriani

Aturan	**Format Kata**	**Pemenggalan**
1	berV...	ber-V... | be-rV...
2	berCAP...	ber-CAP... dimana C!=“r“ & P!=“er“
3	berCAerV...	ber-CaerV... dimana C!=“r“
4	belajar	bel-ajar
5	beC1erC2...	be-C1erC2... dimana C1!={“r“|“l“}
6	terV...	ter-V... | te-rV...
7	terCerV...	ter-CerV... dimana C!=“r“
8	terCP...	ter-CP... dimana C!=“r“ dan P!=“er“
9	teC_1erC_2	$teC_1erC_{2...}$ dimana C_1!=’r’
10	me{l|r|w|y}V...	me-{l|r|w|y}V...
11	mem{b|f|v}...	mem-{b|f|v}...
12	mempe{r|l}...	mem-pe...
13	mem{rV|V}...	me-m{rV|V}... | me-p{rV|V}...
14	men{c|d|j|z}...	men-{c|d|j|z}...
15	menV...	me-nV... | me-tV
16	meng{g|h|q}...	meng-{g|h|q}...
17	mengV...	meng-V... | meng-kV...
18	menyV...	meny-sV…
19	mempV...	mem-pV... dimana V!=„e“
20	pe{w|y}V...	pe-{w|y}V...
21	perV...	per-V... | pe-rV...
23	perCAP	per-CAP... dimana C!=“r“ dan P!=“er“
24	perCAerV...	per-CAerV... dimana C!=“r“

25	pem{b|f|V}...	pem-{b|f|V}...
26	pem{rV|V}...	pe-m{rV|V}... | pe-p{rV|V}...
27	pen{c|d|j|z}...	pen-{c|d|j|z}...
28	penV...	pe-nV... | pe-tV...
29	peng{g|h|q}...	peng-{g|h|q}...
30	pengV...	peng-V... | peng-kV...
31	penyV...	peny-sV…
32	pelV...	pe-lV... kecuali “pelajar” yang menghasilkan “ajar”
33	peCerV...	per-erV... dimana C!={r|w|y|l|m|n}
34	peCP...	pe-CP... dimana C!={r|w|y|l|m|n} dan !=“er“
Keterangan simbol huruf : C : huruf konsonan V : huruf vokal A : huruf vokal atau konsonan P : partikel atau fragmen dari suatu kata, misalnya “er”		

6. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai kata dasar. Proses selesai.

Tipe awalan ditentukan melalui langkah-langkah berikut:

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 2.2 Hapus awalan jika ditemukan.

Tabel 2.3 Cara Menentukan Tipe Awalan Untuk awalan “te-”

Following Characters				**Tipe Awalan**
Set 1	**Set 2**	**Set 3**	**Set 4**	
“-r-“	“-r-“	-	-	None
“-r-“	-	-	-	Ter-luluh
“-r-“	Not (vowel or “-r-“)	“-er-“	Vowel	Ter
“-r-“	Not (vowel or “-r-“)	“-er-“	Not vowel	Ter-
“-r-“	Not (vowel or “-r-“)	Not “-er-“	-	Ter
Not (vowel or “-r-“)	“-er-”	Vowel	-	None
Not (vowel or “-r-“)	“-er-“	Not vowel	-	None

Table 2.4 Jenis awalan berdasarkan tipe awalannya

Tipe Awalan	Awalan yang harus dihapus
di-	di-
ke-	ke-
se-	se-
te-	te-
ter-	ter-
ter-luluh	Ter

Berikut contoh-contoh aturan yang terdapat pada awalan sebagai pembentuk kata dasar :

1. **Awalan SE-**

   Se + semua konsonan dan vokal tetap tidak berubah Contoh :

   - Se + bungkus = sebungkus

- Se + nasib = senasib
- Se + arah = searah
- Se + ekor = seekor

2. **Awalan ME-**

   Me + vokal (a,i,u,e,o) menjadi sengau “meng” Contoh :

   - Me + inap = menginap
   - Me + asuh = mengasuh
   - Me + ubah = mengubah
   - Me + ekor = mengekor
   - Me + oplos = mengoplos

   Me + konsonan b menjadi “mem” Contoh :

   - Me + beri = member
   - Me + besuk = membesuk

   Me + konsonan s menjadi “meny” (luluh) Contoh :

   - Me + sapu = menyapu
   - Me + satu = menyatu

   Me + konsonan t menjadi “men” (luluh) Contoh :

   - Me + tanama = menanam
   - Me + tukar = menukar

   Me + konsonan (l,m,n,r,w) menjadi tetap “me” Contoh :

   - Me + lempar = melempar
   - Me + masak = memasak
   - Me + naik = menaik
   - Me + rawat = merawat
   - Me + warna = mewarna

3. **Awalan KE-**

   Ke + semua konsonan dan vokal tetap tidak berubah Contoh :

   - Ke + bawa = kebawa
   - Ke + atas = keatas

4. **Awalan PE-**

   Pe + konsonan (h,g,k) dan vokal menjadi “per” Contoh :

   - Pe + hitung + an = perhitungan
   - Pe + gelar + an = pergelaran
   - Pe + kantor + = perkantoran

   Pe + konsonan “t” menjadi “pen” (luluh) Contoh :

   - Pe + tukar = penukar
   - Pe + tikam = penikam

   Pe + konsonan (j,d,c,z) menjadi “pen” Contoh :

   - Pe + jahit = penjahit
   - Pe + didik = pendidik
   - Pe + cuci = pencuci
   - Pe + zina = penzina

   Pe + konsonan (b,f,v) menjadi “pem” Contoh :

   - Pe + beri = pemberi
   - Pe + bunuh = pembunuh

   Pe + konsonan “p” menjadi “pem” (luluh) Contoh :

   - Pe + piker = pemikir
   - Pe + potong = pemotong

   Pe + konsonan “s” menjadi “peny” (luluh) Contoh :

   - Pe + siram = penyiram
   - Pe + sabar = penyabar

   Pe + konsonan (l,m,n,r,w,y) tetap tidak berubah Contoh :

   - Pe + lamar = pelamar
   - Pe + makan = pemakan
   - Pe + nanti = penanti
   - Pe + wangi = pewangi

2.3 *Jaro Winkler Distance*

JaroWinkler Distance merupakan sebuah *algoritma* untuk mengukur kesamaan antara dua *string*, biasanya *algoritma* ini digunakan di dalam pendeteksian duplikat. Semakin tinggi nilai *Jaro-Winkler Distance* untuk dua buah *string*, maka semakin mirip *string* tersebut. Nilai normalnya adalah 0 yang menandakan tidak adanya kesamaan dan 1 yang menandakan adanya kesamaan. *Algoritma Jaro-Winkler distance* memiliki kompleksitas waktu *quadratic runtime complexity* yang sangat efektif pada *string* pendek dan dapat bekerja lebih cepat dari *algoritma edit distance*. Dasar dari algoritma ini memiliki tiga bagian:

1. Menghitung panjang *string*
2. Menemukan jumlah karakter yang sama didalam dua *string*
3. Menemukan jumlah *transposisi* (Ana Kurniawati & Rahman, 2010).

Pada *algoritma Jaro* digunakan rumus untuk menghitung jarak (*dj*) antara dua *string* yaitu S_1 dan S_2 adalah

$$dj = \frac{1}{3} x \left(\frac{m}{s1} + \frac{m}{s2} + \frac{m-t}{m} \right) \quad (2.1)$$

Keterangan :

m = Jumlah karakter yang sama persis

|*S1*| = Panjang *string* 1

|*S2*| = Panjang *string* 2

t = Jumlah *transposisi*

Jarak teoritis dua buah karakter yang dikatakan sama dapat dibenarkan jika tidak melebihi :

$$\frac{\max(|s1|,|s2|)}{2} < -1 \quad (2.2)$$

Jika mengacu kepada nilai yang dihasilkan oleh *algoritma Jaro-Winkler* maka nilai jarak maksimalnya adalah 1 menandakan kesamaan *string* yang dibandingkan mencapai seratus persen atau sama persis. s_1 digunakan sebagai acuan untuk urutan di dalam mencari *transposisi*.

Transposisi adalah karakter yang sama dari *string* yang dibandingkan akan tetapi tertukar urutannya. Sebagai contoh, dalam membandingkan kata CRATE dengan TRACE, bila dilihat seksama maka dapat dikatakan semua karakter yang ada di s_1 dengan karakter yang ada di s_2 ada dan sama, tetapi dengan urutan yang berbeda. Dengan mengganti C dan T, dapat dilihat perubahan kata CRATE menjadi TRACE. Pertukaran dua elemen *string* inilah adalah contoh nyata dari *transposisi* yang dijelaskan. Dalam pencocokkan DwAyNE dan DuANE memiliki urutan yang sama D-A-N-E, jadi tidak ada *transposisi*.

Jaro-Winkler *distance* menggunakan *prefix scale* (p) yang memberikan tingkat penilaian yang lebih, dan *prefix length* (l) yang menyatakan panjang awalan yaitu panjang karakter yang sama dari *string* yang dibandingkan sampai ditemukannya ketidaksamaan. Bila *string* s_1 dan *string* s_2 dibandingkan, maka *Jaro-Winkler distancenya* adalah (W.E, 2006):

$$dw = dj + (l \; x \; p \; (1 - dj) \qquad (2.3)$$

Keterangan :

dj = *Jaro distance* untuk *string* s_1 dan s_2

l = Panjang *prefiks* umum di awal *string.* (panjang karakter yang sama sebelum ditemukan ketidaksamaan *max* 4)

p = konstanta *scaling factor*. Nilai standar untuk konstanta ini menurut Winkler *p=0.1*

Berikut ini adalah contoh pada perhitungan *Jaro-Winkler distance*. Jika *string* s_1 B A R T H A dan s_2 B A R H T A maka :

m = 6

s1 = 6

s2 = 6

Karakter yang tertukar hanyalah T dan H. Maka

t = 1

Maka nilai *Jaro distance* adalah:

$$dj = \frac{1}{3} x \left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6} \right) = 0{,}944$$

Kemudian bila diperhatikan susunan s_1 dan s_2 terdapat kesaaman pada awal kata yaitu BAR, dapat diketahui nilai l = 3 dan dengan nilai konstan p = 0.1, maka nilai *Jaro-Winkler distance* adalah :

$$dw = 0{,}944 + (3 \; x \; 0{,}1 \; (1 - 0{,}944) = 0{,}961$$

2.4 Akurasi

Tujuan evaluasi adalah untuk menghasilkan perbaikan pada proses pengambilan informasi. Sebuah proses evaluasi memerlukan proses pengujian mengenai hasil dari sistem. Hal itu perlu dilakukan untuk mendapatkan akurasi dari perhitungan yang dilakukan. Pengujian dilakukan dengan menghitung nilai akurasi berdasarkan relevansi kesesuaian kata yang ditampilkan dengan yang diinputkan. Nilai akurasi adalah kemampuan untuk mendapatkan hasil yang sedekat mungkin dengan hasil yang sesungguhnya. Berikut Tabel 2.5 parameter untuk menghitung akurasi.

Tabel 2.5 Parameter menghitung akurasi

Keterangan	**Relevan**	**Tidak Relevan**
Terambil	True (T)	False (F)

Rumus untuk menghitung Akurasi:

$$Akurasi = \frac{RW}{W} \; x \; 100 \; \% \qquad (2.4)$$

Keterangan :

RW = jumlah kata yang ter-*stem*

W = jumlah kata yang di*stemming* dengan benar

2.5 Penelitian Sebelumnya

Penelitian sebelumnya dilakukan oleh Agung Prasetyo, Wiga Maulana Baihaqi, dan Iqbaluddin Syam Had (Agung Prasetyo, 2018) yang telah melakukan penelitian "Algoritma Jaro-Winkler *Distance*: Fitur *Autocorrect* Dan *Spelling Suggestion* Pada Penulisan Naskah Bahasa Indonesia Di Bms Tv". Pada penelitian ini disimpulkan bahwa penerapan *jaro winkler distance* untuk fitur *autocorrect* dan *spelling suggestion* yang telah dibuat dapat menangani kesalahan penulisan ejaan kata pada penulisan naskah bahasa indonesia. Dari hasil pengujian terhadap 60 kata yang terdiri dari berbagai kesalahan penulisan ejaan, fitur *autocorrect* dan *spelling sugesstion* dapat menangani kesalahan penulisan ejaan pada 49 kata dengan baik. Sedangkan pada kata lainnya terjadi kesalahan dalam menampilkan saran ejaan.

Penelitian lainnya dilakukan oleh Andita Dwiyoga Tahitoe dan Diana Purwitasari (Purwitasari, 2010) dari Institut Teknologi Sepuluh November telah melakukan penelitian yaitu "Modifikasi Enhanced Confix Stripping Stemmer Untuk Bahasa Indonesia dengan Metode Corpus Based Stemming" pada penelitian tersebut mampu memperbaiki kesalahan *overstemming* dan *understemming* yang dilakukan oleh algoritma Enhanced Confix Stripping *Stemmer*. Metode Enhanced Confix Stripping adalah penggabungan pengembangan dari dua mteode sebelumnya yaitu Nazief-Adriani dan Arifin. Penggunaan metode *corpus based stemming* dapat digunakan untuk memilih hasil stemming yang tepat berdasarkan koleksi dokumen yang digunakan.