PERBANDINGAN METODE SMOOTHING UNTUK DETEKSI DAN KOREKSI KESALAHAN KATA DALAM TEKS BERBAHASA INDONESIA

Fernando Sihole[1], Kania Evita Dewi[2]

[1,2] Teknik Informatika – Universitas Komputer Indonesia
Jl.Dipatiukur 112-113 Bandung
E-mail : nitrousfernando@gmail.com[1], kania.evita.dewi@email.unikom.ac.id[2]

ABSTRAK

Dalam Topik NLP (*Natural Language Processing*), kesalahan kata merupakan hal yang penting untuk diperhatikan. Beberapa penelitian telah dilakukan untuk mengatasi masalah kesalahan ejaan *real-word* dalam bahasa Inggris. Skripsi Muhammad Aburizal Siregar telah mengatasi masalah kesalahan ejaan *real-word* dalam bahasa Indonesia yaitu deteksi dan koreksi kesalahan kata *real-word* memiliki akurasi sebesar 11% setelah mengubah perhitungan probabilitas n-gram dengan menggunakan metode *additive smoothing*. Oleh karena itu, penelitian ini bertujuan untuk membandigkan akurasi metode deteksi dan koreksi kesalahan ejaan *real-word* dalam bahasa Indonesia menggunakan metode *smoothing Good-Turing estimate, Jelinek-Mercer, Katz smoothing, Witten-Bell dan Absolute discounting* untuk kasus deteksi dan koreksi kesalahan *real-word* dalam bahasa Indonesia. Dari hasil pengujian pada 30 artikel berita menggunakan korpus sebesar 21.5 juta didapatkan akurasi yang paling tinggi adalah metode *Absoulte Discounting* pada akurasi deteksi sebesar 80%, dan *Jelinek Mercer* pada akurasi koreksi sebesar 89% dan hasil akurasi metode *smoothing* yang paling rendah adalah metode *Good-turing* dengan akurasi deteksi dan koreksi sebesar 0%.

.

Kata kunci : kesalahan ejaan, deteksi, koreksi, *real-word*, n-gram, *confusion set.*

1. PENDAHULUAN

Pada skripsi Muhammad Aburizal Siregar telah mengatasi masalah kesalahan ejaan *real-word* dalam bahasa Indonesia berdasarkan metode n-gram lokal deteksi dan koreksi kesalahan kata *real-word* memiliki akurasi sebesar 11%[1]. setelah mengubah perhitungan probabilitas n-gram dengan menggunakan metode *additive smoothing*, nilai yang sangat kecil jika dibandingkan dengan penelitian Samanta dan Chaudhuri sebesar 80%[2]. Oleh karena itu, dalam penelitian ini akan diambil beberapa metode *smoothing* untuk kasus deteksi dan koreksi kesalahan *real-word* dalam bahasa Indonesia yang diambil dari korpus PAN-BPPT dan UI *Tagged Corpus* dan akan ditambah dengan korpus bahasa Inggris dari *Wikipedia* tahun 2016 sebesar 20 juta kata yang di *translate* ke bahasa Indonesia. Metode yang diambil adalah metode bigram dan trigram lokal yang diajukan Samanta dan Chaudhuri[2]. Metode ini dapat mendeteksi lebih dari satu kesalahan dalam satu kalimat, dengan itu penelitian ini akan membandingkan metode *smoothing Good-Turing estimate, Jelinek-Mercer, Katz smoothing, Witten-Bell dan Absolute discounting* yang sudah diteliti oleh Stanley F. Chen dan Joshua Goodman [3], penelitiannya akan digunakan untuk melihat bagaimana hasil perbandingan dari perhitungan probabilitas n-gram untuk kasus koreksi kesalahan ejaan *real-word* bahasa Indonesia.

2. ISI PENELITIAN

2.1 Analisis Proses

Analisis proses yang dilakukan adalah *preprocessing* pembuatan korpus n-gram, *preprocessing* deteksi dan koreksi kesalahan, dan proses deteksi dan koreksi kesalahan *real-word.* Berikut penjelasannya.

2.2 Analisis Data Masukan

Data masukan yang digunakan dalam penelitian ini adalah data uji, daftar kata, dan teks korpus. Berikut adalah penjelasan masing-masing data masukan.

2.2.1 Daftar Kata

Daftar kata dibutuhkan untuk membuat *confusion set*. *Confusion set* adalah himpunan kata-kata yang biasa tertukar satu sama lain dari kata tersebut. Daftar kata diambil dari *database* Kamus Besar Bahasa Indonesia yang berformat .csv. Berikut adalah contoh potongan *database* Kamus Besar Bahasa Indonesia. Daftar kata dibuat dengan mengambil kolom Kata dari *database* Kamus Besar Bahasa Indonesia di atas. Kata yang diambil hanyalah kata dasar saja, dan dilakukan *filtering* dengan menghapus tanda strip. Daftar kata tersebut kemudian disimpan dalam *database*. Daftar kata ini akan dipanggil saat sistem akan membuat *confusion set* pada tahap Deteksi dan Koreksi Kesalahan Kata.

2.3 *Preprocessing* Pembuatan Korpus N-Gram

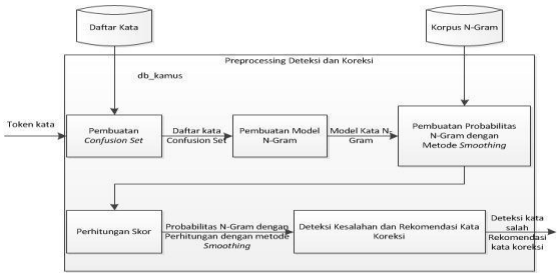
Korpus n-gram digunakan untuk menghitung nilai probabilitas suatu kata. Korpus n-gram berisi daftar n-gram dan jumlah kemunculan n-gram tersebut dalam semua dokumen-dokumen dari korpus dan data uji[2]. Langkah-langkah untuk membuat korpus n-gram adalah *case folding*, tokenisasi kalimat, *filtering* tanda baca, tokenisasi kata[4], pembuatan korpus n-gram, perhitungan korpus n-gram dan menjadi korpus n-gram,

2.4 *Preprocessing* Deteksi dan Koreksi Kesalahan

Preprocessing ini akan dilakukan untuk menyiapkan data uji agar dapat dilakukan proses deteksi dan koreksi kesalahan ejaan *real-word* pada data uji tersebut. Tahapan *preprocessing* deteksi dan koreksi kesalahan sama dengan tiga tahapan awal *preprocessing* pembuatan korpus n-gram, yaitu *case folding*, tokenisasi kalimat, dan tokenisasi kata[4].

2.5 Deteksi dan Koreksi Kesalahan Ejaan *Real -Word*

Proses deteksi dan koreksi kesalahan ejaan *real-word* berawal dengan pembuatan *confusion set* untuk setiap kata kandidat. Setelah itu dilakukan penghitungan peringkat setiap elemen dalam *confusion set* tersebut. Berdasarkan hasil peringkat, kesalahan dideteksi dan diberikan sugesti kata koreksi. Ini dilakukan secara kata per kata[2].



Gambar 1 Diagram Proses Deteksi dan Koreksi Kesalahan Kata

Dalam analisis ini kita akan memakai data uji yang telah di-*preprocessing* pada subbab 2.4 agar memudahkan analisis, beri notasi W^i[2] untuk tiap-tiap token sebagai berikut.

Tabel 1 Daftar Token Kata Data Uji

W^i	Token
W^0	_
W^1	sketsa
W^2	keadaan
W^3	sastra
W^4	eropa
W^5	sampah
W^6	akhir
W^7	abad
W^8	ke14
W^9	pertengahan
W^{10}	_

Dari token-token kata di atas, akan dicari kata-kata manakah yang merupakan kesalahan *real-word*. Jika kesalahan terdeteksi, akan diberi sugesti kata yang benar untuk mengoreksi kata yang salah tersebut.

2.5.1 Pembuatan *Confusion Set*

Pertama-tama bangkitkan *confusion set* untuk setiap token kata, yaitu himpunan kata-kata yang merupakan variasi ejaan dari token kata tersebut. Ini didapat dengan menghitung *Levenshtein distance* dari setiap token dalam daftar kata. *Levenshtein distance* adalah jumlah minimum operasi edit yang diperlukan untuk mengubah suatu kata menjadi kata yang lain. Operasi edit merupakan penambahan (*insertion*), penghapusan (*deletion*), atau penggantian (*substitution*) satu karakter dalam suatu kata. Hasil penghitungan dapat dilihat pada kolom terakhir baris terakhir matriks[5]. Dari hasil penghitungan, didapat bahwa nilai *Levenshtein distance* dari perbandingan kata "keadaan" dan "keagaan" adalah 1. Penghitungan *Levenshtein distance* dilakukan untuk semua kata dalam daftar kata kamus. Berikut adalah hasil penghitungan *Levenshtein distance* untuk kata "keadaan".

Kata-kata yang dijadikan anggota *confusion set* adalah kata-kata dalam daftar kata yang memiliki nilai *Levenshtein distance* maksimal 1. *Confusion set* dapat direpresentasikan sebagai

$$C(W^i) = \{c_1^i, c_2^i, \dots, c_j^i, \dots, c_{k_i}^i\} \quad (1)$$

dimana W^i adalah kata ke-i di dalam kalimat dan k_i adalah jumlah elemen di dalam $C(W^i)$. Berikut adalah *confusion set* untuk setiap kata dalam data uji.

Tabel 2 Hasil Pembuatan Anggota *Confusion Set* untuk Data Uji

W_i	Kata	c_j^i	Anggota	Nilai *Levenshtein*
W^1	sketsa	c_1^1	sketsa	1
W^2	keadaan	c_1^2	keadaan	0
		c_2^2	keadaban	1
		c_3^2	keagaan	1
…	…	…	…	…

2.5.2 Pembuatan Model N-Gram

Setelah itu dilakukan pembuatan model n-gram untuk setiap anggota *confusion set*[2]. Untuk setiap anggota $C(W^i)$, bangkitkan bigram kiri, bigram kanan dan trigram dengan cara mengambil kata di kiri, kanan, dan keduanya (untuk trigram) dari W^i(diambil dari tabel 3.20). Maka bigram dan trigram yang terbentuk adalah:

Bigram kiri: $W^{i-1}c_j^i$ (2)

Bigram kanan: $c_j^i W^{i+1}$ (3)

Trigram: $W^{i-1}c_j^i W^{i+1}$ (4)

dimana $1 \leq j \leq k_i$. (5)

untuk kata di awal dan di akhir kalimat, kata yang kosong ditandai dengan garis bawah (_). Pembuatan model n-gram dilakukan untuk semua anggota *confusion set* untuk setiap kata. Setelah itu lihat

jumlah kemunculan tiap bigram dan trigram dalam korpus n-gram. Korpus bigram dan trigram yang digunakan untuk mencari jumlah kemunculan adalah berdasarkan dari Tabel 3.15 dan Tabel 3.16. Jumlah kemunculan n-gram dalam korpus dilambangkan dengan

$count(c_j^i)$ untuk jumlah kemunculan unigram (6)

$count(W^{i-1}c_j^i)$ untuk jumlah kemunculan bigram kiri, (7)

$count(c_j^i W^{i+1})$ untuk jumlah kemunculan bigram kanan, (8)

$count(W^{i-1}c_j^i W^{i+1})$ untuk jumlah kemunculan trigram. (9)

Tabel-tabel berikut masing-masing menampilkan jumlah kemunculan unigram, bigram kiri, bigram kanan dan trigram pada data uji. Jika suatu n-gram tidak ditemukan dalam korpus, maka diberi nilai 0. Berikut adalah daftar kemunculan bigram kiri data uji dalam korpus n-gram. Sebagai contoh, untuk token W^5= "sampah", untuk anggota *confusion set* c_2^5 = "sampai", jumlah kemunculan bigram kirinya $W^5 c_j^5$ = "eropa sampai" pada korpus n-gram adalah 1.

Tabel 3 Daftar Jumlah Kemunculan Unigram Data Uji

c_j^0		**count**(c_j^1)
c_1^0	_	2
c_j^1		**count**(c_j^1)
c_1^1	sketsa	1
c_j^2		**count**(c_j^3)
c_1^2	keadaan	1
c_2^2	keadaban	0
c_3^2	keagaan	0
…		….

Berikut adalah daftar kemunculan unigram data uji dalam korpus n-gram. Sebagai contoh, untuk token W^5 = "sampah", untuk anggota *confusion set* c_2^5 = "sampai", jumlah kemunculan unigram $c_2^5 W^5$ = "sampai" pada korpus n-gram adalah 1.

Tabel 4 Daftar Jumlah Kemunculan Bigram Kiri Data Uji

c_j^1		**Bigram Kiri** $(W^0 c_j^1)$	**count** $(W^0 c_j^1)$
c_1^1	sketsa	_ sketsa	0
c_j^2		**Bigram Kiri** $(W^1 c_j^3)$	**count** $(W^1 c_j^3)$
c_1^2	keadaan	sketsa keadaan	1
c_2^2	keadaban	sketsa keadaban	0
c_3^2	keagaan	sketsa keagaan	0
…		…	…

Berikut adalah daftar kemunculan bigram kanan data uji dalam korpus n-gram. Sebagai contoh, untuk token W^5 = "sampah", untuk anggota *confusion set* c_2^5 = "sampai", jumlah kemunculan bigram kanannya $c_2^5 W^6$ = "sampai akhir" pada korpus n-gram adalah 1.

Tabel 5 Daftar Jumlah Kemunculan Bigram Kanan Data Uji

c_j^1		**Bigram Kanan** $(c_j^1 W^2)$	**count**$(c_j^1 W^2)$
c_1^1	sketsa	sketsa keadaan	1
c_j^2		**Bigram Kanan** $(c_j^2 W^3)$	**count**$(c_j^2 W^3)$
c_1^2	keadaan	keadaan sastra	1
c_2^2	keadaban	keadaban sastra	0
c_3^2	keagaan	keagaan sastra	0
…		…	…

Berikut adalah daftar kemunculan trigram data uji dalam korpus n-gram. Sebagai contoh, untuk token W^5 = "sampah", untuk anggota *confusion set* c_2^5 = "sampai", jumlah kemunculan trigramnya $W^4 c_2^5 W^6$ = "eropa sampai akhir" pada korpus n-gram adalah 1.

Tabel 6 Daftar Jumlah Kemunculan Trigram Data Uji

c_j^1		**Trigram** $(W^0 c_j^1 W^2)$	**count(** $W^0 c_j^1 W^2$**)**
c_1^1	sketsa	_ sketsa keadaan	0
c_j^2		**Trigram** $(W^1 c_j^2 W^3)$	**count(** $W^1 c_j^2 W^3$**)**
c_1^2	keadaan	sketsa keadaan sastra	1
c_2^2	keadaban	sketsa keadaban sastra	0
c_3^2	keagaan	sketsa keagaan sastra	0
…		…	…

Tabel 7 Jumlah Frekuensi Kemunculan Tiap Unigram, Bigram dan Trigram

r Bigram	Nr	**r** Trigram	Nr
0	20	0	15
1	42	1	41
2	1	2	0
3	0	3	0

Hasil dari perhitungan jumlah kemunculan pada pembuatan korpus dihitung tiap frekuensi nya dengan huruf "r" dan jumlah frekuensi "Nr". Saat menjumlahkan tiap frekuensi kemunculan dilihat dari saat pembuatan korpus n-gram sebelumnya, jika mencari jumlah frekuensi 0 bisa didapat dari saat deteksi dan koreksi kesalahan.

Lalu hitung juga total jumlah kemunculan n-gram untuk semua anggota *confusion set* untuk setiap kata. Total jumlah kemunculan dilambangkan dengan

$\sum_{r=1}^{k_i} count(c_r^i)$ untuk total jumlah kemunculan unigram. (10)

$\sum_{r=1}^{k_i} count(W^{i-1}c_r^i)$ untuk total jumlah kemunculan bigram kiri. (11)

$\sum_{r=1}^{k_i} count(W^{i+1}c_r^i)$ untuk total jumlah kemunculan

bigram kanan. (12)
$\sum_{r=1}^{k_i} count(W^{i-1}c_r^i W^{i+1})$ untuk total jumlah kemunculan trigram. (13)
k adalah jumlah anggota dari *confusion set* suatu kata dan r adalah urutan anggota dari *confusion set* tersebut. Berikut adalah hasil penghitungan total jumlah kemunculan bigram dan trigram untuk data uji.

Tabel 8 Total Jumlah Kemunculan *Confusion Set* Unigram dan Bigram Kiri

W^i	Token	$\sum_{r=1}^{k_i} count(c_r^i)$	$\sum_{r=1}^{k_i} count(W^{i-1}c_r^i)$
W^1	sketsa	$count(c_1^1) = 1$	$count(W^0 c_1^1) = 0$
W^2	keadaan	$count(c_1^2) + count(c_2^2) + count(c_3^2) = 1 + 0 + 0 = 1$	$count(W^1 c_1^2) + count(W^1 c_2^2) + count(W^1 c_3^2) = 1 + 0 + 0 = 1$
…	…	…	…

Tabel 9 Total Jumlah Kemunculan *Confusion Set* Bigram Kanan

W^i	Token	$\sum_{r=1}^{k_i} count(c_r^i W^{i+1})$
W^1	sketsa	$count(c_1^1 W^2) = 1$
W^2	keadaan	$count(c_1^2 W^3) + count(c_2^2 W^3) + count(c_3^2 W^3) = 1 + 0 + 0 = 1$
…	…	…

Tabel 10 Total Jumlah Kemunculan *Confusion Set* Trigram

W^i	Token	$\sum_{r=1}^{k_i} count(W^{i-1}c_r^i W^{i+1})$
W^1	sketsa	$count(W^0 c_1^1 W^2) = 0$
W^2	keadaan	$count(W^1 c_1^2 W^3) + count(W^1 c_2^2 W^3) + count(W^1 c_3^2 W^3) = 1 + 0 + 0 = 1$
…	…	…

2.5.3 Penghitungan Probabilitas N-Gram

Tujuan penulis disini untuk melihat bagaimana hasil perbandingan dari perhitungan probabilitas n-gram untuk kasus koreksi kesalahan ejaan *real-word* bahasa Indonesia dengan metode *smoothing Good-Turing estimate, Jelinek-Mercer, Katz smoothing, Witten-Bel dan Absolute discounting* yang sudah diteliti oleh Stanley F. Chen dan Joshua Goodman[3].

2.5.3.1 Metode *Smoothing Good-Turing*

Rumus metode *smoothing good-turing*[6] adalah

$\frac{1}{N}(r+1)\frac{N_{r+1}}{N_r}$. (14)

$N = \sum_r rN_r$ (15)

r adalah frekuensi kemunculan dari kata.
Nr adalah jumlah frekuensi kemunculan kata.

2.5.3.2 Metode *Smoothing Jelinek-Mercer*

Metode *jelinek-mercer*[7] ini menggunakan cara *interpolation*, yaitu berhubungan dengan n-gram di bawah tingkatanya.
Nilai λ = 0.1
Bigram Kiri

$P_1'(c_j^i|W^{i-1}) = \lambda P_1(c_j^i|W^{i-1}) + (1-\lambda) P_0(c_j^i)$ (16)

Bigram Kanan

$P_2'(c_j^i|W^{i+1}) = \lambda P_2(c_j^i|W^{i+1}) + (1-\lambda) P_0(c_j^i)$ (17)

Trigram

$P_3' = (c_j^i|W^{i-1}, W^{i+1}) = \lambda P_3(c_j^i|W^{i-1}, W^{i+1}) + (1-\lambda)(P_1(c_j^i|W^{i-1}) + P_2(c_j^i|W^{i+1})/2)$ (18)

$P_0(C_j^i)$ adalah nila probabitas *maximum likelihood* untuk uniram.
$P_1(C_j^i|W^{i-1})$ adalah nila probabitas *maximum likelihood* untuk bigram kiri.
$P_2(C_j^i|W^{i+1})$ adalah nila probabitas *maximum likelihood* untuk bigram kanan.
$P_3(W_j^i|W^{i-1}, W^{i+1})$ adalah nila probabitas *maximum likelihood* untuk trigram.

2.5.3.3 Metode *Smoothing Katz*

Berdasarkan persamaan Rumus metode *katz smoothing*[1] ada 2 opsi yaitu jika $r > 0$ dan jika $r = 0$, sama seperti metode *jelinek mercer*, perlu menggunakan nilai probabilitas *maximum likelihood.*

1. Jika $r > 0$ gunakan rumus $P_1(c_j^i|W^{i-1}) = dr$

   $dr = \frac{r^*}{r}$, $r^* = (r+1)\frac{N_{r+1}}{Nr}$. (19)

   r = frekuensi dari $count(c_j^i|W^{i-1})$
   r adalah frekuensi kemunculan dari kata.
   Nr adalah jumlah frekuensi kemunculan kata.
2. Jika $r = 0$ gunakan rumus
   Bigram Kiri $\alpha(c_j^i|W^{i-1})P_0(c_j^i)$ (20)

   $\alpha(c_j^i|W^{i-1}) = \frac{1-\sum_{w_i:count(W^{i-1}c_r^i)>0} P_1(c_j^i|W^{i-1})}{1-\sum_{w_i:count(c_r^i)>0} P_0(c_j^i)}$ (21)

   Bigram Kanan $\alpha(c_j^i|W^{i+1})P_0(c_j^i) =$ (22)

   $\frac{1-\sum_{w_i:count(W^{i+1}c_r^i)>0} P_2(c_j^i|W^{i+1})}{1-\sum_{w_i:count(c_r^i)>0} P_0(c_j^i)}$ (23)

   Trigram

   $\alpha(c_j^i|W^{i-1}, W^{i+1})(P_1(c_j^i|W^{i-1}) + P_2(c_j^i|W^{i+1})/2)$ (24)

   $\alpha(c_j^i|W^{i-1}, W^{i+1}) = \frac{1-\sum_{w_i:count(W^{i-1}, W^{i+1}c_r^i)>0} P_3(c_j^i|W^{i-1}, W^{i+1})}{1-\sum_{w_i:count(W^{i-1}c_r^i).count(W^{i+1}c_r^i)>0} (P_1(c_j^i|W^{i-1}) + P_2(c_j^i|W^{i-1}))/2}$ (25)

   $\sum_{w_i:count(c_j^i)>0} P_0(c_j^i|W^{i-1})$ adalah jumlah W_i yang memiliki jumlah kemunculan unigram yang lebih dari 0.

$\sum_{w_i:count(c_j^i w_{i-1})>0} P_1(c_j^i|W^{i-1})$ adalah jumlah W_i yang memiliki jumlah kemunculan bigram kanan yang lebih dari 0.

$\sum_{w_i:count(c_j^i w_{i+1})>0} P_2(c_j^i|W^{i+1})$ adalah jumlah W_i yang memiliki jumlah kemunculan bigram kanan yang lebih dari 0.

$\sum_{w_i:count(c_j^i w_{i-1})>0} P_3(c_j^i|W^{i-1}, W^{i+1})$ adalah jumlah W_i yang memiliki jumlah kemunculan trigram yang lebih dari 0.

$\sum_{w_i:count(W^{i-1}c_j^i),count(W^{i+1}c_j^i)>0}(P_1(c_j^i|W^{i-1}) + P_2(c_j^i|W^{i-1}))/2$

adalah jumlah W_i yang memiliki jumlah kemunculan bigram kiri dan bigram kanan yang lebih dari 0.

2.5.3.4 Metode *Smoothing Witten-Bell*

Berikut rumus untuk metode *Smoothing Witten-Bell*[7]:

1. Rumus perhitungan untuk bigram kiri

$$P_1'(c_j^i|W^{i-1}) = \frac{count(W^{i-1}c_j^i) + N_{1+}(c_j^i|W^{i-1}\cdot)\, P_1(c_j^i|W^{i-1})}{\sum_{r=1}^{k_i} count(W^{i-1}c_r^i)+0.01\times V_j + N_{1+}(c_j^i|W^{i-1}\cdot)} \quad (26)$$

$$N_{1+}(c_j^i|W^{i-1}\cdot) = |\{W^i : count(c_j^i W^{i+1}) > 0\}| \quad (27)$$

$N_{1+}(c_j^i|W^{i-1}\cdot)$ adalah kemunculan kata dari bigram kiri yang memiliki nilai kemuncula lebih dari 0.

2. Rumus perhitungan bigram kanan

$$P_2'(c_j^i|W^{i+1}) = \frac{count(W^{i+1}c_j^i) + N_{1+}(c_j^i|W^{i+1}\cdot)\, P_2(c_j^i|W^{i+1})}{\sum_{r=1}^{k_i} count(W^{i+1}c_r^i)+0.01\times V_j + N_{1+}(c_j^i|W^{i+1}\cdot)} \quad (28)$$

$$N_{1+}(c_j^i|W^{i+1}\cdot) = |\{W^i : W^i count(c_j^i W^{i+} 0\}| \quad (29)$$

$N_{1+}(c_j^i|W^{i-1}\cdot)$ adalah kemunculan kata dari bigram kanan yang memiliki nilai kemunculan lebih dari 0.

3. Rumus perhitungan untuk trigram.

$$P_3'(c_j^i|W^{i-1}, W^{i+1}) = \frac{count(W^{i-1}, W^{i+1}c_j^i) + N_{1+}(c_j^i|W^{i-1}, W^{i+1}\cdot)\, P_3(c_j^i|W^{i-1}, W^{i+1})}{\sum_{r=1}^{k_i} count(W^{i-1}, W^{i+1}c_r^i)+0.01\times V_j + N_{1+}(c_j^i|W^{i-1}, W^{i+1}\cdot)} \quad (30)$$

$$N_{1+}(c_j^i|W^{i-1}, W^{i+1}\cdot) = |\{W^i : W^i count(c_j^i W^{i-1}, W^{i+1}) > 0\}| \quad (31)$$

$N_{1+}(c_j^i|W^{i+1}\cdot)$ adalah kemunculan kata dari trigram yang memiliki nilai kemunculan lebih dari 0.

2.5.3.5 Metode *Smoothing Absolute Discounting*

Berikut rumus untuk metode *Smoothing Absolute Discounting*[7]:

1. Rumus perhitungan untuk bigram kiri.

$$P_1'(c_j^i|W^{i-1}) = \frac{\max\{count(c_j^i W^{i-1})+0.01-\delta, 0\}}{\sum_{r=1}^{k_i} count(W^{i-1}c_r^i)+0.01\times V_j} + (1-\lambda)\, P_0(c_j^i). \quad (32)$$

$$(1-\lambda) = \frac{\delta}{\sum_{r=1}^{k_i} count(W^{i-1}c_r^i)+0.01\times V_j} N_{1+}(c_j^i W^{i-1}\cdot) \quad (33)$$

$N_{1+}(c_j^i|W^{i-1}\cdot)$ adalah kemunculan kata dari bigram kiri yang memiliki nilai kemuncula lebih dari 0.

2. Rumus perhitungan untuk bigram kanan.

$$P_2'(c_j^i|W^{i+1}) = \frac{\max\{count(c_j^i W^{i+1})+0.01)-\delta, 0\}}{\sum_{r=1}^{k_i} count(W^{i+1}c_r^i)+0.01\times V_j} + (1-\lambda)\, P_0(c_j^i). \quad (34)$$

$$(1-\lambda) = \frac{\delta}{\sum_{r=1}^{k_i} count(W^{i+1}c_r^i)+0.01\times V_j} N_{1+}(c_j^i W^{i+1}\cdot) \quad (35)$$

$N_{1+}(c_j^i|W^{i-1}\cdot)$ adalah kemunculan kata dari bigram kanan yang memiliki nilai kemunculan lebih dari 0.

3. Rumus perhitungan untuk trigram

$$P_3'(c_j^i|W^{i-1}, W^{i+1}) = \frac{\max\{count(c_j^i W^{i-1}, W^{i+1})+0.01-\delta, 0\}}{\sum_{r=1}^{k_i} count(W^{i-1}, W^{i+1}c_r^i)+0.01\times V_j} + (1-\lambda)(P_1(c_j^i|W^{i-1}) + P_2(c_j^i|W^{i+1})/2) \quad (36)$$

$$(1-\lambda) = \frac{\delta}{\sum_{r=1}^{k_i} count(W^{i-1}, W^{i+1}c_r^i)+0.01\times V_j} N_{1+}(c_j^i W^{i-} \quad (37)$$

$N_{1+}(c_j^i|W^{i+1}\cdot)$ adalah kemunculan kata dari trigram yang memiliki nilai kemunculan lebih dari 0.

2.6 Pengujian

Pengujian nilai akurasi pada aplikasi deteksi dan koreksi kesalahan kata *real-word* ini dilakukan untuk untuk mengetahui sejauh mana hasil deteksi dan koreksi kesalahan kata *real-word* yang dihasilkan benar-benar tepat. Korpus n-gram yang digunakan bersumber dari korpus PAN-BPPT sebanyak total 500 ribu kata, UI-Tagged-Corpus sebanyak 1 juta kata dan korpus *Wikipedia* 2016 sebanyak 20 juta kata. Teks yang digunakan sebagai data uji adalah 30 artikel berita yang berasal dari www.detik.com dan www.kompas.co.id. Nilai akurasi didapat dengan membagi jumlah deteksi sistem dengan jumlah deteksi yang dilakukan secara manual.

Kinerja dari metode deteksi dan koreksi kesalahan ejaan *real-word* ini diukur dengan melakukan pengukuran akurasi deteksi dan akurasi koreksi. Pengukuran akurasi deteksi dilakukan dengan persamaan

$$akurasi\ deteksi = \frac{ds}{dm} \times 100\% \quad (38)$$

dimana

ds : jumlah deteksi yang dihasilkan sistem

dm : jumlah deteksi yang dihasilkan secara manual

Sedangkaran pengukuran akurasi koreksi dilakukan dengan persamaan

$$akurasi\ koreksi = \frac{kb}{kb + ks} \times 100\% \quad (38)$$

dimana

kb : jumlah koreksi benar yang dihasilkan sistem

ks : jumlah koreksi salah yang dihasilkan sistem

tabel 11 hasil akurasi deteksi dan koreksi

Metode *Smoothing*	Akurasi Deteksi	Akurasi Koreksi
Good Turing	0%	0%
Jelinek-Mercer	36%	89%
Katz	50%	24%
Witten-Bell	79%	7%
Absolute Discounting	80%	7%

Pada data uji dari 30 teks yang diuji sering terjadi kesalahan saat skor peringkat tertinggi anggota *confusion set* tidak cukup untuk menyatakan bahwa kata salah adalah salah. Batas suatu kata dinyatakan salah adalah jika skornya 0 atau kurang dari 1% skor peringkat tertinggi. Pada kasus ini skor kata salah lebih besar dari 1% skor peringkat tertinggi sehingga kata yang salah dianggap sebagai benar begitu juga dengan hasil akurasi koreksi, ada kata yang seharusnya tidak dikoreksi akan tetapi dikoreksi oleh sistem dikarenakan skor nya masih rendah dibandingkan kata yang sering muncul pada korpus.

Dari hasil akurasi deteksi dan koreksi kesalahan menggunakan 21,5 juta korpus dengan 5 metode *smoothing* dapat di bandingkan dengan jurnal milik Stanley F. Chen dan Joshua Goodman [3], bahwa

1. Metode *smothing good-turing* ini kurang bagus begitu pula dengan hasil akurasi yang kurang bagus juga yang ditunjukan pada tabel 11 dengan akurasi detesi dan koreksi sebesar 0%.
2. Metode *smothing jelinek mercer* ini bagus, begitu pula dengan hasil akurasi yang bagus juga yang ditunjukan pada tabel 11 dengan akurasi deteksi sebesar 36% dan akurasi koreksi sebesar 89%.
3. Metode *smothing katz* ini sangat bagus di data yang besar dan beragam karena itu dengan hasil akurasi yang ditunjukan pada tabel 11 dengan akurasi deteksi sebesar 50% dan akurasi koreksi sebesar 24% kurang bagus untuk korpus yang dipakai saat ini.
4. Metode *smothing witten-bell* ini sangat bagus pada data yang lumayan besar begitu pula dengan hasil akurasi yang bagus juga yang ditunjukan pada tabel 11 dengan akurasi deteksi sebesar 79% dan akurasi koreksi sebesar 7%.
5. Metode *smothing witten-bell* ini sangat bagus pada data yang lumayan besar begitu pula dengan hasil akurasi yang bagus juga yang ditunjukan pada tabel 11 dengan akurasi deteksi sebesar 79% dan akurasi koreksi sebesar 7%.

3. PENUTUP

Pada bagian ini akan menjelaskan kesimpulan dan saran dari penelitian.

3.1 Kesimpulan

Berdasarkan hasil dari pengujian akurasi deteksi dan koreksi kesalahan ejaan *Real-word*, akurasi deteksi yang tinggi adalah metode *smoothing Absolute Discounting* sebesar 80%, dan untuk akurasi koreksi yang tinggi adalah metode *smoothing Jelinek-Mercer* sebesar 89% dan akurasi paling kecil dengan akurasi deteksi dan koreksi sebesar 0% adalah metode *smoothing Good Turing.*

3.2 Saran

Untuk penelitian selanjutnya, disarankan untuk menambah ukuran korpus n-gram yang akan digunakan agar dapat menambah akurasi deteksi dan koreksi kesalahan kata *real-word*. Juga disarankan untuk menggunakan metode *smoothing* lain seperti Add-one *Smoothing* dan Kneser-Ney *Smoothing*.

DAFTAR PUSTAKA

[1] M. Aburizal S. Skripsi, Deteksi Dan Koreksi Kesalahan *Real-Word* Pada Penulisan Kata Menggunakan Metode N-Gram Dalam Teks Berbahasa Indonesia, 2017.

[2] B. B. C. ratip Samanta, *"A simple real-world error detection and correction using local word bigram and trigram," Proceedings of the Twenty-Fifth Conference on Computational Linguistics and Speech Processing*. 2013.

[3] S. F. C. a. J. Goodman, "An Empirical Study Of Smoothing Techniques For Language Modeling," *Computer Speech and Language*, 1999.

[4] F. J. D. d. R. L. M. E. Mays, *"Context based spelling correction,"*. Information Processing and Management, 27(5):527-522,, 1991

[5] M. Gilleland, *"Levenshtein Distance, in Three Flavors"*. Merriam Park Software https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/.

[6] EllisWeng, A. Owens Lecture 11: The Good-Turing Estimate, 4 March 2010.

[7] B. MacCartney, *NLP Lunch Tutorial: Smoothing*. Stanford Natural Language Processing Group, 21 April 2005.