

BAB III

ANALISIS PENYELESAIAN MASALAH

Pada bab tiga ini akan dibahas mengenai analisis *stemming* dengan menggunakan algoritma Porter, algoritma Paice/Husk, algoritma Lovins, dan analisis teknik-teknik yang digunakan untuk membandingkan berbagai metode *stemming* tersebut.

3.1 Analisis Porter Stemmer

Terdapat lima langkah utama dalam algoritma Porter. Tiap langkah dieksekusi berurutan dan hanya dieksekusi sekali untuk setiap *term*. Kelima langkah tersebut akan dijelaskan secara lebih rinci di bagian ini.

- 1) Langkah 1
 - a) Langkah ini bertujuan untuk mereduksi *term-term* jamak menjadi bentuk tunggalnya. *Rule-rule* yang digunakan :
 - SSES → SS
 - IES → I
 - SS → SS
 - S →
 - b) Langkah ini bertujuan untuk mereduksi *term-term* dalam bentuk *continuous* atau *participle* ke *term* dasarnya. *Rule-rule* yang digunakan :
 - (m > 0) EED → EE
 - (*v*) ED →
 - (*v*) ING →

Langkah berikut ini hanya dieksekusi jika *rule* kedua atau ketiga di langkah (1b) diinvokasi. Langkah ini diperlukan untuk menyesuaikan bentuk *term* ketika suatu *term* berubah dari bentuk *continuous* atau *participle* menjadi bentuk dasarnya. Proses yang dilakukan meliputi pemetaan *term* dengan akhiran *double letter* ke *single letter* dan penambahan -E untuk beberapa *suffix*. *Rule-rule* yang digunakan :

- AT → ATE
- BL → BLE
- IZ → IZE
- (*d and not (*L or *S or *Z)) → single letter

- (m-1) and (*o) \rightarrow E

- c) Langkah ini bertujuan untuk mengganti '-Y' dengan '-I' jika terdapat huruf vokal lain dalam *term*.

Rule yang digunakan :

- (*v) Y \rightarrow I

2) Langkah 2

Langkah ini bertujuan untuk menangani suatu *term* yang memiliki akhiran ganda. Jika suatu *term* memiliki akhiran ganda, maka *term* tersebut akan direduksi sehingga menjadi *term* dengan akhiran tunggal. Pada langkah ini dilakukan pengindeksan pada *penultimate letter* (huruf kedua terkahir) dari *term* untuk membantu mempercepat pengujian kondisi. String S1 dalam langkah ini juga diurutkan berdasarkan nilai *penultimate letter*. Pemilihan *penultimate letter* ini bertujuan untuk mendapatkan distribusi yang merata terhadap nilai-nilai yang mungkin untuk string S1.

Rule-rule yang digunakan :

- | | |
|-----------------------------------|-----------------------------------|
| - (m>0) ATIONAL \rightarrow ATE | - (m>0) IZATION \rightarrow IZE |
| - (m>0) TIONAL \rightarrow TION | - (m>0) ATION \rightarrow ATE |
| - (m>0) ENCI \rightarrow ENCE | - (m>0) ATOR \rightarrow ATE |
| - (m>0) ANCI \rightarrow ANCE | - (m>0) ALISM \rightarrow AL |
| - (m>0) IZER \rightarrow IZE | - (m>0) IVENESS \rightarrow IVE |
| - (m>0) ABLI \rightarrow ABLE | - (m>0) FULNESS \rightarrow FUL |
| - (m>0) ALLI \rightarrow AL | - (m>0) OUSNESS \rightarrow OUS |
| - (m>0) ENTLI \rightarrow ENT | - (m>0) ALITI \rightarrow AL |
| - (m>0) ELI \rightarrow E | - (m>0) IVITI \rightarrow IVE |
| - (m>0) OUSLI \rightarrow OUS | - (m>0) BILITI \rightarrow BLE |

3) Langkah 3

Pada langkah ini dilakukan pengindeksan pada huruf terkahir dari *term* untuk membuang beberapa akhiran spesifik. *Rule-rule* yang digunakan :

- | | |
|---------------------------------|-------------------------------|
| - (M>0) ICTATE \rightarrow IC | - (m>0) ICAL \rightarrow IC |
| - (M>0) ATIVE \rightarrow | - (m>0) FUL \rightarrow |
| - (M>0) ALIZE \rightarrow AZ | - (m>0) NESS \rightarrow |
| - (M>0) ICITI \rightarrow IC | |

4) Langkah 4

Pada langkah ini dilakukan pengindeksan pada *penultimate letter* dari *term* untuk membuang beberapa akhiran spesifik jika *term* tersebut memiliki nilai $(m > 1)$. *Rule-rule* yang digunakan :

- | | |
|----------------------|---|
| - $(m > 1)$ AL -> | - $(m > 1)$ ENT -> |
| - $(m > 1)$ ANCE -> | - $(m > 1 \text{ and } (*S \text{ or } *T))$ ION -> |
| - $(m > 1)$ ENCE -> | - $(m > 1)$ OU -> |
| - $(m > 1)$ ER -> | - $(m > 1)$ ISM -> |
| - $(m > 1)$ IC -> | - $(m > 1)$ ATE -> |
| - $(m > 1)$ ABLE -> | - $(m > 1)$ ITI -> |
| - $(m > 1)$ IBLE -> | - $(m > 1)$ OUS -> |
| - $(m > 1)$ ANT -> | - $(m > 1)$ IVE -> |
| - $(m > 1)$ EMENT -> | - $(m > 1)$ IZE -> |
| - $(m > 1)$ MENT -> | |

5) Langkah 5

Langkah terakhir ini bertujuan untuk melakukan penyesuaian akhir terhadap *stem* yang dihasilkan.

a) Membuang huruf terakhir '-E' jika $(m > 1)$ atau $(m = 1)$ dan kondisinya bukan *o. *Rule-rule* yang digunakan :

- $(m > 1)$ E →
- $(m = 1 \text{ and not } *o)$ E →

b) Langkah ini menangani duplikasi huruf terakhir pada beberapa kata. *Rule* yang digunakan :

- $(m > 1 \text{ and } *d \text{ and } *L)$ → single letter

3.2 Analisis Paice/Husk Stemmer

Algoritma Paice/Husk terdiri dari empat langkah utama :

- 1) Pilih kelompok yang relevan
 - Jika tidak ada kelompok yang sesuai dengan kata yang sedang diproses, maka lakukan terminasi.
 - Jika ada kelompok yang sesuai, pertimbangkan *rule* pertama dari kelompok tersebut.
- 2) Periksa apakah *rule* tersebut dapat diterapkan

- Jika akhiran dari kata tidak bersesuaian dengan akhiran dari *rule* (yang ditulis terbalik), lanjutkan ke langkah 4.
 - Jika akhiran bersesuaian, jika terdapat *intact flag* dan katanya tidak *intact*, lanjutkan ke langkah 4.
 - Jika *acceptability conditions** tidak dipenuhi, lanjutkan ke langkah 4.
- 3) Terapkan *rule*
- Hapus akhiran sebanyak jumlah karkater yang dispesifikasikan, jika terdapat append string maka tambahkan ke bentuk kata yang baru.
 - Jika simbol kontinuasinya adalah ”.”, maka lakukan terminasi.
 - Jika simbol kontinuasinya adalah ”>”, kembali ke langkah 1.
- 4) Lihat *rule* lain
- Lanjutkan dengan *rule* berikutnya di tabel.
 - Jika kelompok *rule* telah berubah, maka lakukan terminasi.
 - Jika belum, kembali ke langkah 2.

* *Acceptibility conditions* terdiri dari dua kondisi :

- Jika kata dimulai dengan vokal, maka setidaknya terdapat dua huruf setelah *stemming*.
- Jika kata dimulai dengan konsonan, maka setidaknya terdapat tiga huruf setelah *stemming* dan paling sedikit salah satu dari ketiga huruf tersebut adalah vokal.

Rule – rule yang digunakan dalam algoritma Paice/Husk dapat ditemukan di bagian lampiran.

3.3 Analisis Lovins Stemmer

Algoritma Lovins terdiri dari dua langkah utama :

1) *Stemming*

- Tentukan letak awal pencarian *ending* sehingga *stem* yang dihasilkan terdiri dari dua huruf atau lebih.
- Lakukan pencarian *ending* di daftar *ending*.

- Jika *ending* ditemukan, periksa apakah *context sensitive rule* dipenuhi. Jika ya, hapus *ending* dan lanjutkan ke langkah dua. Jika tidak, maka kembali ke langkah pencarian *ending* di daftar *ending*.
 - Jika *ending* tidak ditemukan, lanjutkan ke langkah dua.
- 2) Recoding
- Hapus konsonan ganda di akhir *stem*.
 - Lakukan pencarian *stem* sisa di daftar *transformation rules*.
 - Jika ditemukan kecocokan, *recode stem* sesuai dengan *rule* yang didefinisikan.

Rule – *rule* yang digunakan dalam algoritma Lovins dapat ditemukan di bagian lampiran.

3.4 Analisis Teknik Perbandingan Algoritma Stemming

Terdapat beberapa alasan mengenai pentingnya perbandingan *stemmer*. Alasan pertama, designer sistem temu balik informasi harus memahami performansi dari tiap algoritma agar dapat memilih *stemmer* terbaik untuk digunakan dalam pembuatan index. Selain itu suatu sistem temu balik informasi juga dapat menawarkan beberapa pilihan *stemmer* sebagai cara untuk mengontrol nilai *recall* dan *precision*. Terakhir, perbandingan algoritma *stemming* juga dapat digunakan sebagai dasar untuk pembuatan algoritma yang lebih baik.

Beberapa cara dapat digunakan untuk membandingkan algoritma *stemming*, baik ditinjau dari sisi performansi maupun kekuatan dan kemiripan dari masing-masing algoritma. Kekuatan algoritma *stemming* memetakan suatu *stemmer* ke suatu nilai yang mengindikasikan seberapa besar *stemmer* tersebut mengubah *term* untuk menghasilkan *stem*. Kemiripan algoritma *stemming* memetakan n -tuples dari *stemmer* ($n \geq 2$) ke suatu nilai yang mengindikasikan kesamaan dari *stemmer*. Perlu dicatat bahwa metric kekuatan adalah fungsi dari satu *stemmer*, sedangkan metric kemiripan adalah fungsi dari dua atau lebih *stemmer*.

Dalam penyusunan Tugas Akhir ini, ketiga algoritma *stemming* yang digunakan dibandingkan dengan menggunakan tujuh kriteria, yaitu :

1. nilai *recall*
2. nilai *precision*
3. nilai *non-interpolated average precision*
4. faktor kompresi index
5. rata-rata jumlah *term* dalam suatu *conflation class*
6. banyaknya *term* yang diubah oleh *stemmer*
7. *modified Hamming distance*

Recall dinyatakan sebagai bagian dari dokumen relevan dalam dokumen yang ditemukan.:

$$recall = \frac{\text{jumlah dokumen relevan yang berhasil ditemukan}}{\text{jumlah seluruh dokumen relevan}}, \quad (a)$$

Precision dinyatakan sebagai bagian dokumen relevan yang ditemukan.

$$precision = \frac{\text{jumlah dokumen relevan yang berhasil ditemukan}}{\text{jumlah seluruh dokumen yang ditemukan}}, \quad (b)$$

Sedangkan *non-interpolated average precision* dinyatakan sebagai rata-rata *precision* untuk setiap kemunculan dokumen relevan.

Misalkan terdapat suatu hasil temu balik informasi sebagai berikut :

Rangking	Relevan
1	Ya
2	Tidak
3	Tidak
4	Ya
5	Ya
6	Tidak
7	Tidak
8	Tidak
9	Tidak
10	Tidak

$\rightarrow Precision = 1 / 1 = 1$
 $Recall = 1 / 4 = 0.25$

$\rightarrow Precision = 2 / 4 = 0.5$
 $Recall = 2 / 4 = 0.5$

$\rightarrow Precision = 3 / 5 = 0.6$
 $Recall = 3 / 4 = 0.75$

$$Non-interpolated\ average\ precision = \frac{1 + 0.5 + 0.6}{3} = 0.7.$$

Ketiganya menggambarkan performansi dari sistem temu balik informasi dengan melakukan penghitungan terhadap jumlah dokumen relevan hasil pencarian.

Faktor kompresi index adalah reduksi fraksional ukuran index yang berhasil dicapai. *Stemmer* yang lebih kuat cenderung memiliki faktor kompresi index yang lebih besar. Faktor kompresi index dihitung dengan rumus :

$$CF = \frac{(n - s)}{N}, \quad (c)$$

di mana n adalah jumlah kata dalam *corpus* dan s adalah jumlah *stem*.

Sebagai contoh, suatu *corpus* dengan 50000 kata (n) dan 40000 *stem* (s) memiliki faktor kompresi index 20%.

Faktor kompresi index dapat menggambarkan kekuatan dari tiap-tiap algoritma *stemming* yang digunakan.

Rata-rata jumlah *term* dalam suatu *conflation class* adalah nilai rata-rata jumlah *term* yang berkorespondensi dengan *stem* yang sama dalam suatu *corpus*. *Stemmer* yang kuat cenderung memiliki lebih banyak *term* dalam suatu *conflation class*.

Jumlah *term* awal dan *stem* yang berbeda dapat pula digunakan sebagai salah satu *metrics* perbandingan. *Stemmer* sering membiarkan suatu *term* tidak berubah. *Stemmer* yang lebih kuat cenderung mengubah suatu *term* lebih sering dibandingkan *stemmer* yang lemah.

Hamming *distance* diantara dua string yang memiliki panjang yang sama didefinisikan sebagai jumlah karakter yang berbeda dalam posisi yang sama. Untuk string yang memiliki panjang yang berbeda, perbedaan panjang ditambahkan ke Hamming *distance* sehingga diperoleh fungsi *modified Hamming distance*. *Modified Hamming distance* ini memperhitungkan transformasi dari akhiran *stem*. Sebagai contoh, suatu algoritma *stemming* dapat mereduksi *corpus* {try, tried, trying} menjadi *stem* tri. Rata-rata *modified Hamming distance* antara kata asal dengan *stem* adalah $(1+2+4)/3 = 2.33$ karakter, dan mediannya adalah 2.

Modified Hamming distance dapat menggambarkan kesamaan dari tiap-tiap algoritma *stemming* yang digunakan.

Pengukuran kekuatan *stemmer* dilakukan dengan hipotesis awal bahwa suatu algoritma *stemming* yang kuat cenderung meningkatkan nilai *recall*, mengurangi

precision, dan meningkatkan faktor kompresi index. Stemmer yang paling kuat ialah *stemmer* yang membuang seluruh karakter kecuali karakter pertama dari *term* yang diproses. Jumlah equivalence class untuk *stemmer* seperti ini ialah 26, dan faktor kompresi indexnya $(n-26)/n$. Stemmer yang paling lemah ialah *stemmer* yang tidak merubah satu karakterpun dari *term* yang diproses. Stemmer seperti ini akan memiliki equivalence class sebanyak jumlah *term*, dan faktor kompresi indexnya ialah 0. Pada prakteknya, nilai *recall*, *precision*, dan faktor kompresi index dari ketiga algoritma *stemming* yang digunakan (Porter, Paice/Husk, Lovins) dihitung lalu dibandingkan untuk mendapatkan metric kekuatan dari masing-masing algoritma.

Metric kemiripan *stemmer* M untuk pasangan algoritma *stemming* $A1$ dan $A2$, dengan corpus W , adalah inverse dari rata-rata *modified Hamming distance* (d) untuk seluruh kata dalam corpus, dengan kata lain

$$M(A1,A2,W) = \frac{n}{\sum_{i=1}^n d(x_i, y_i)} , \quad (d)$$

di mana n adalah ukuran dari W , dan untuk semua kata w_i dalam W , x_i adalah hasil aplikasi $A1$ untuk w_i dan y_i adalah hasil aplikasi $A2$ untuk w_i .

Invers dari nilai rata-rata digunakan agar algoritma yang lebih mirip memiliki nilai M yang lebih besar [FRA99].

Sebagai contoh, misalkan $W = \{\text{brittle, engineered, fairies}\}$ dan algoritma *stemming* $A1$ menghasilkan *stem* $\{\text{brit, engineer, fairy}\}$, dan algoritma *stemming* $A2$ menghasilkan $\{\text{britt, engineered, fairi}\}$. Maka $M(A1,A2,W)$ dihitung dengan membagi ukuran W (3) dengan jumlah dari *modified Hamming distance* antara *stem* yang dihasilkan oleh tiap *stemmer* untuk setiap kata. *Modified Hamming distance* antara *brit* - *britt* adalah 1, antara *engineer* - *engineered* adalah 2, dan antara *fairy* - *fairi* adalah 1, sehingga nilai $M(A1,A2,W)$ yang diperoleh adalah $3/(1+2+1) = 0.75$. Nilai ini dinyatakan sebagai *metric* kemiripan algoritma $A1$ dan $A2$.

Cara lain yang dapat digunakan dalam evaluasi algoritma *stemming* adalah *error counting*. Metode ini melakukan evaluasi jumlah kesalahan klasifikasi yang muncul

dalam proses *stemming*, yaitu *under-stemming* dan *over-stemming*. *Under-stemming* mengacu pada kata-kata yang seharusnya diklasifikasikan ke dalam satu kelas, tetapi pada prosesnya tidak. Hal ini menyebabkan suatu konsep tunggal tersebar di beberapa *stem* yang berbeda, sehingga cenderung akan mengurangi nilai *recall*. *Over-stemming* mengacu pada kata-kata yang seharusnya tidak diklasifikasikan ke dalam satu kelas, tetapi pada prosesnya iya. Hal ini menyebabkan arti dari *stem* menjadi kabur, dan akan mempengaruhi nilai *precision*. Pendekatan *error counting* mengacu pada fakta bahwa meskipun menguntungkan jika diperoleh *term index* yang terkompresi, hal ini hanya berlaku sampai titik tertentu. Ketika *conflation* semakin kuat, penggabungan konsep yang berbeda terjadi lebih sering. Pada titik ini, sedikit peningkatan pada nilai *recall* mengorbankan penurunan nilai *precision* yang cukup besar [PAI96].