BAB II

TINJAUAN PUSTAKA

2.1 Sistem Rekomendasi

Sistem rekomendasi (*recommender system*) adalah sebuah sistem yang mengusulkan informasi dan menyediakan fasilitas yang diinginkan pengguna dalam membuat suatu keputusan (Ricci, 2011). Sistem rekomendasi telah berkembang pada awal 1990-an sebagai respon dari membanjirnya informasi yang tersedia di internet (Rani dkk., 2014). Rani dkk. (2014) menyatakan sistem rekomendasi berbeda dengan sistem pencarian *keyword* karena sistem rekomendasi berusaha untuk mencari *items* yang cocok dengan selera pengguna dan kualitas yang diharapkan pengguna.

Menurut Sase dkk. (2015), sistem rekomendasi telah menjadi aplikasi yang penting pada *e-commerce* dan akses informasi karena kemampuannya memberikan saran dan memangkas banyaknya informasi sehingga pengguna diarahkan kepada *items* yang paling memenuhi kebutuhan dan preferensi mereka. Beberapa contoh aplikasi yang menerapkan sistem rekomendasi adalah Amazon.com yang menyediakan rekomendasi buku, CD, dan produk lainnya (Linden dkk., 2003), MovieLens yang memberikan rekomendasi film, dan Pandora yang memberikan rekomendasi musik (Sase dkk., 2015).

2.2 Content-based Filtering

Content-based Filtering merupakan pendekatan yang memungkinkan pengguna mendapatkan rekomendasi berdasarkan pemilihan *item* yang dilakukan sebelumnya dan relasinya dengan *item* pada *database*. Pendekatan *Content-based*

Filtering efektif digunakan pada pencarian *item* berbasis teks berdasarkan topiknya (Li & Kim, 2002).

Pendekatan *Content-based Filtering* juga memiliki beberapa kelebihan jika dibandingkan dengan pendekatan *Collaborative Filtering* (Lops dkk., 2011), yaitu sebagai berikut.

a. *User independence*

Content-based Filtering menggunakan *ratings* yang telah diberikan oleh pengguna aktif untuk membuat profilnya sendiri. Hal ini berbeda dengan pendekatan *Collaborative Filtering* yang memerlukan *rating* dari pengguna lain untuk menemukan *"nearest neighbors"* dari pengguna aktif.

b. *Transparency*

Penjelasan mengenai bagaimana sistem rekomendasi bekerja dapat disediakan menggunakan fitur daftar konten secara eksplisit atau deskripsi mengenai apa yang menyebabkan suatu *item* bisa masuk ke dalam daftar rekomendasi. Fitur tersebut yang menjadi indikator apakah pengguna dapat mempercayai sebuah rekomendasi atau tidak. Sedangkan metode *Collaborative Filtering* tidak memungkinkan hal ini karena rekomendasi diberikan oleh pengguna yang tidak dikenal yang memiliki kesamaan selera dengan *item* tersebut.

c. *New item*

Metode *Content-based Filtering* memungkinkan untuk merekomendasikan *items* yang belum diberikan *rating* oleh pengguna manapun. Oleh karena itu, pengguna terbebas dari *first-rater problem*. Masalah ini merupakan masalah utama pada pendekatan *Collaborative Filtering* karena pendekatan tersebut sangat

bergantung dari preferensi pengguna untuk memberikan rekomendasi. Jika *item* baru tidak mendapatkan *rating* dari pengguna lain, sistem tidak akan merekomendasikan *item* tersebut.

2.3 Text Mining

Text mining digunakan untuk mendeskripsikan teknik dari *data mining* yang secara otomatis menemukan sesuatu hal yang berguna atau sebuah pengetahuan baru dari sebuah teks yang tidak terstruktur (Han & Kamber, 2000). Kunci dari proses ini adalah menggabungkan informasi yang berhasil diekstraksi dari berbagai sumber (Tan, 1999). Tujuan dari *text mining* yaitu mendapatkan informasi yang bermanfaat dari kumpulan dokumen yang ada. Selain itu, Handayani (2014) menyatakan *text mining* dapat membantu permasalahan seperti pemrosesan, pengorganisasian atau pengelompokan dan menganalisis teks yang tidak terstruktur dalam jumlah besar.

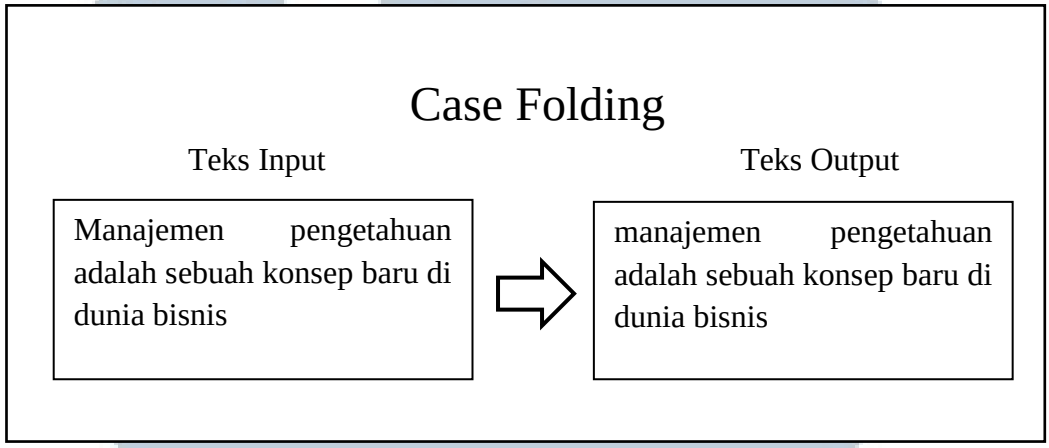
2.4 Tahapan Preprocessing Data

Teks yang akan dilakukan proses *text mining*, pada umumnya memiliki beberapa karakteristik yaitu memiliki dimensi yang tinggi, terdapat *noise* pada data, dan terdapat struktur teks yang tidak baik. Oleh karena itu, pada proses *text mining*, terdapat beberapa tahapan awal (*preprocessing*) yang perlu dilakukan, yaitu *case folding*, *tokenizing*, *filtering*, *stemming*, dan *analyzing* (Handayani, 2014).

1. Case Folding

Case folding adalah proses yang pertama kali dilakukan dalam rangkaian perancangan klasifikasi dokumen teks. Proses ini merupakan proses di mana kata-kata di dalam dokumen atau kalimat akan diubah menjadi huruf kecil (a sampai z)

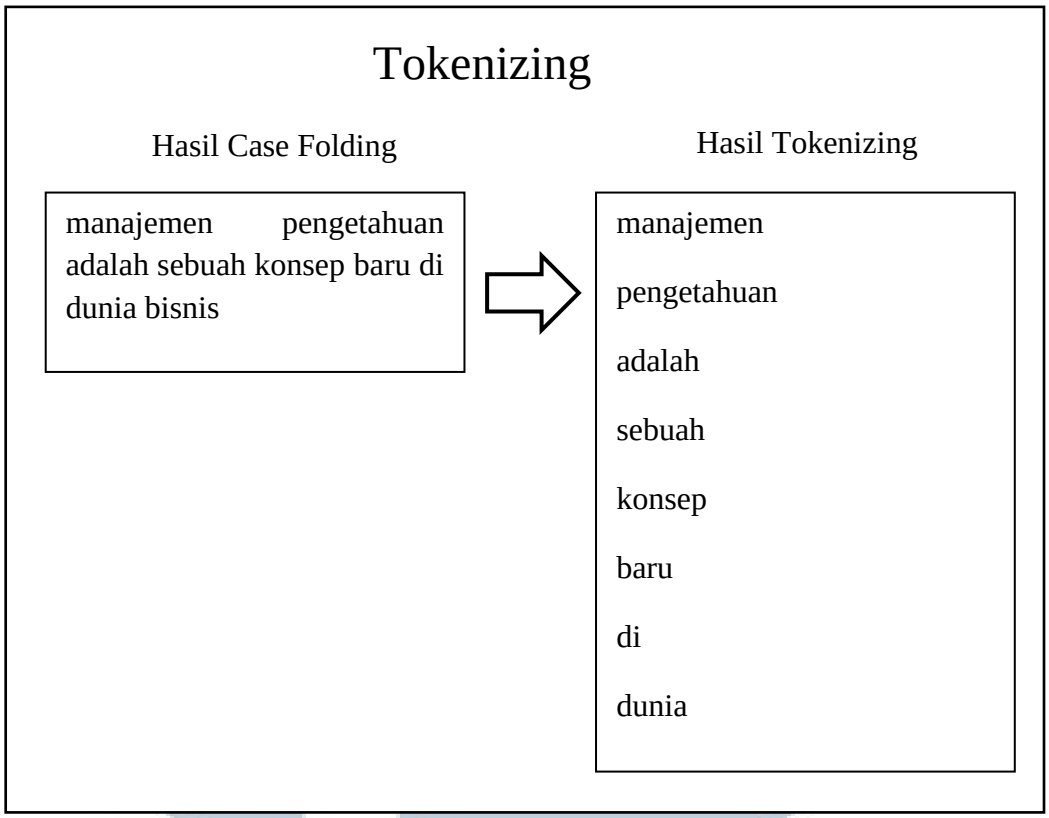
dan menghilangkan tanda baca. Karakter lain selain huruf akan dianggap *delimiter* sehingga karakter tersebut akan dihilangkan. Hal ini dilakukan untuk mencegah terjadinya *noise* pada saat pengambilan informasi. Untuk selanjutnya, hasil dari *case folding* nantinya akan digunakan untuk proses *tokenizing*.



Gambar 2.1 Contoh Case Folding
(Handayani, 2014)

2. Tokenizing

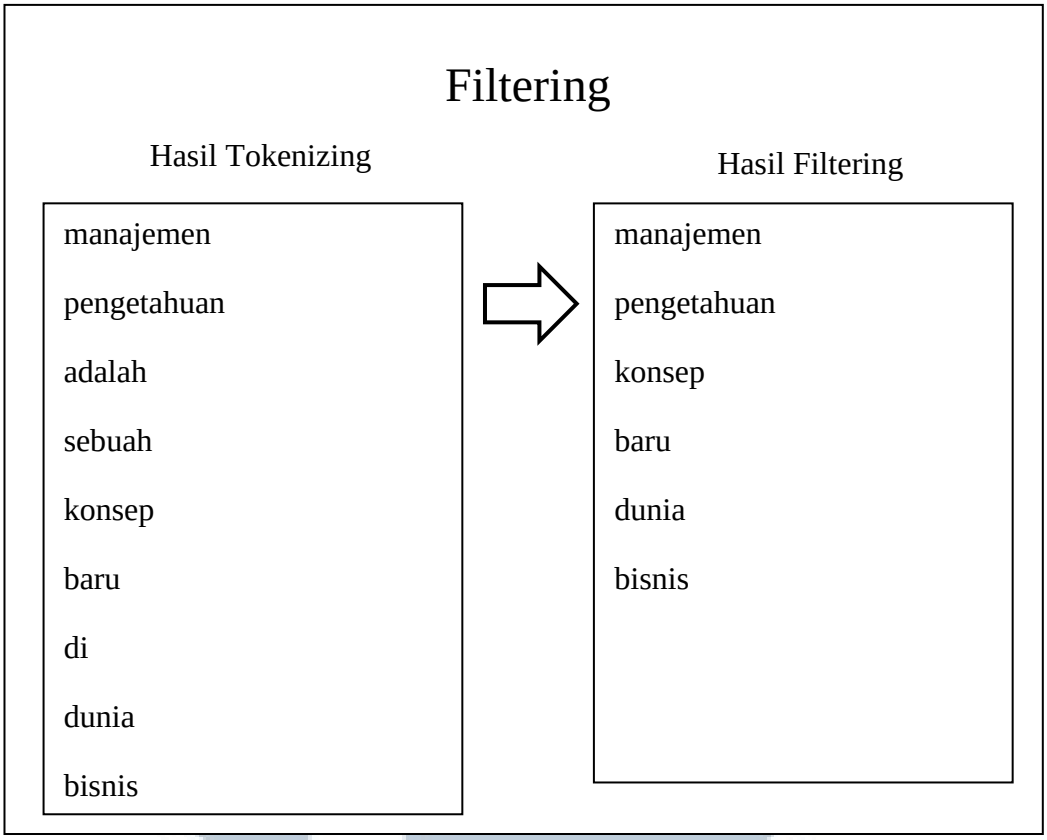
Proses *tokenizing* merupakan proses yang dilakukan setelah melakukan proses *case folding*. Pada tahap ini dilakukan pemotongan *string input* berdasarkan tiap kata yang menyusunnya. Hasil pemrosesan akan berupa kata yang disebut dengan *token*/*term*. *Term* ini nantinya akan disimpan ke dalam *database* untuk dilakukan *indexing* saat melakukan pencarian.



Gambar 2.2 Contoh Tokenizing
(Handayani, 2014)

3. Filtering

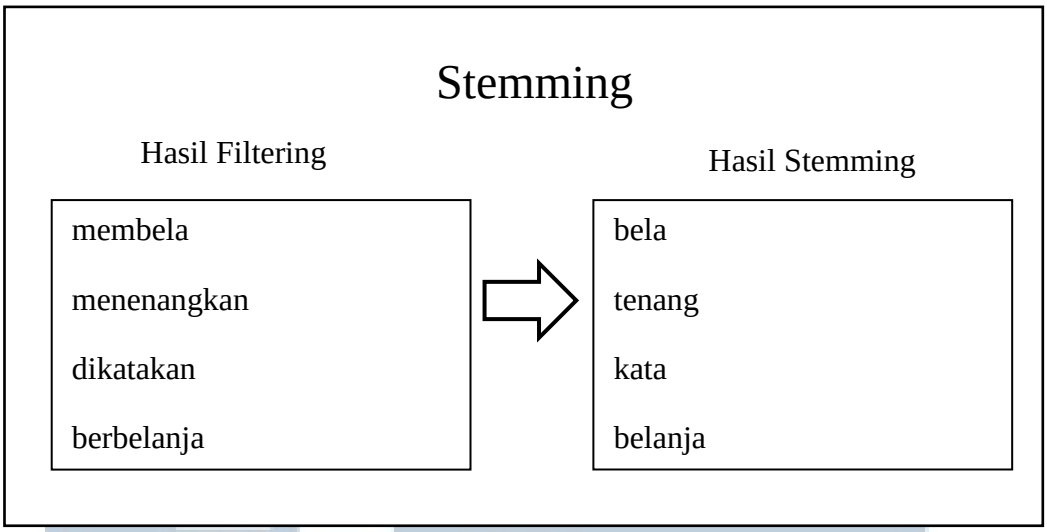
Filtering atau *parsing* merupakan proses pengambilan kata-kata penting dari hasil *token*. Tahap *filtering* dapat dilakukan dengan menghapus *stoplist/stopword* (membuang kata yang kurang penting). *Stopword* adalah kata-kata yang sering muncul dalam teks dalam jumlah besar dan dianggap tidak memiliki makna. Pada tahap ini, kata-kata yang merupakan *stopword* akan dihilangkan. *Stopword* ini dapat berupa kata penghubung, kata depan, dan kata pengganti, contohnya seperti "yang", "di", "dan", "ke", dan "dari". Tujuan dari proses ini adalah mengurangi volume kata sehingga hanya kata-kata penting saja yang terdapat pada dokumen.



Gambar 2.3 Contoh Filtering
(Handayani, 2014)

4. Stemming

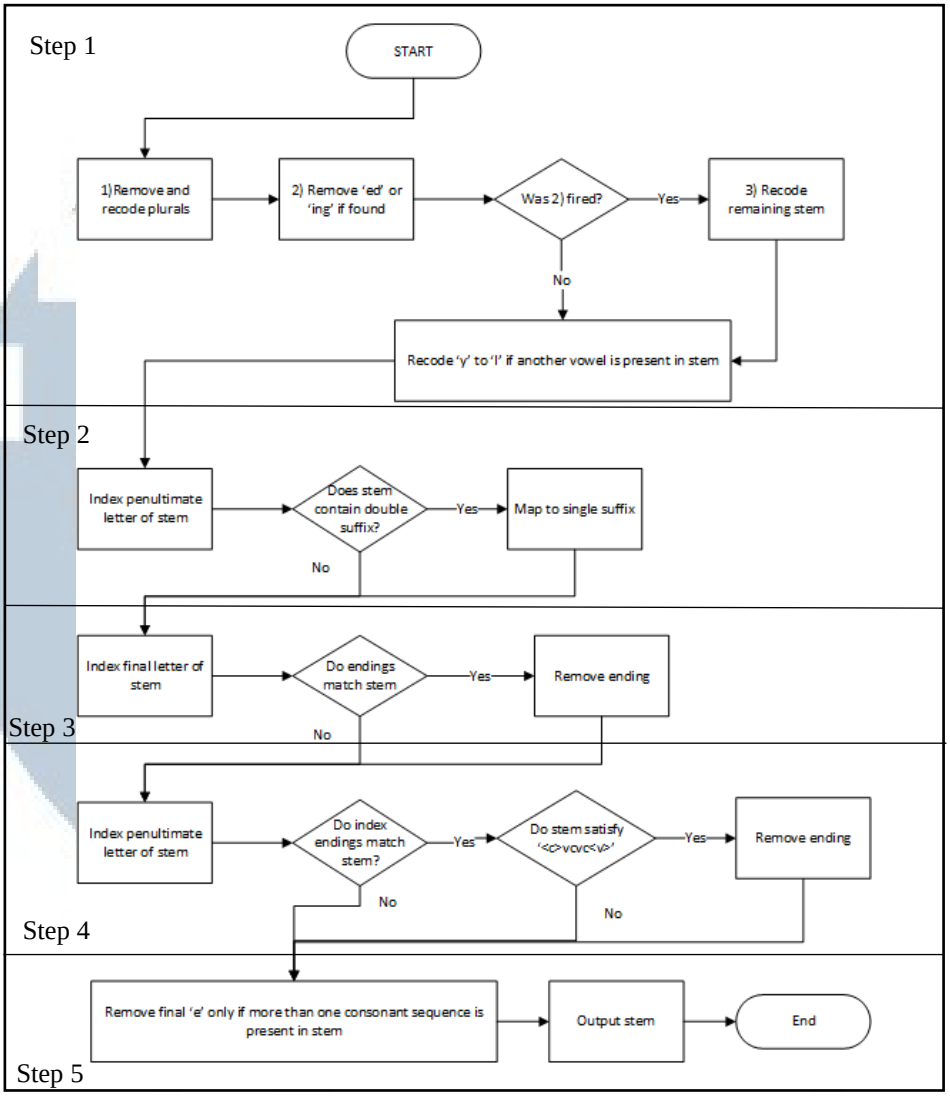
Proses *stemming* merupakan proses untuk mencari kata dasar dari kata yang sudah mengalami proses *filtering*. Pencarian kata dasar sebuah kata dapat memperkecil hasil indeks tanpa harus menghilangkan makna. Proses *stemming* dilakukan dengan menghilangkan semua imbuhan baik yang terdiri dari awalan (*prefix*), akhiran (*suffix*), sisipan (*infix*), bentuk perulangan, dan kombinasi antara awalan dan akhiran (*confix*). Tujuan dari proses ini adalah mengurangi variasi kata yang mempunyai kata dasar yang sama.



Gambar 2.4 Contoh Stemming
(Handayani, 2014)

2.5 Algoritma Porter Stemmer

Menurut Zaman (2014), algoritma *porter stemmer* dirancang dengan aturan bahwa *suffix* pada bahasa Inggris umumnya merupakan kombinasi *suffix* kecil dan sederhana. Tiap langkah proses pemenggalan kata dilakukan secara serial yang menyimulasikan proses kata infleksional dan derivasional. Pada tiap langkah, *suffix* tertentu dihilangkan dengan aturan substitusi. Aturan substitusi diterapkan ketika sejumlah kondisi terpenuhi. Sebagai contoh, kondisi yang sederhana antara lain panjang minimum hasil *stem* (jumlah urutan huruf vokal dan konsonan). Panjang minimum ini disebut dengan hasil (*measure*). Kondisi sederhana lainnya adalah *stem* dilakukan jika *stem* berakhir dengan huruf konsonan atau jika *stem* terdapat huruf vokal. Bila semua kondisi pada suatu aturan sesuai maka aturan tersebut diterapkan sehingga berakibat pada pengurangan *suffix*. Jika kondisi pada suatu aturan tidak sesuai maka dicoba kondisi untuk aturan lainnya sehingga aturan tersebut cocok atau aturan pada langkah tersebut tidak bisa digunakan. Gambar 2.5 menunjukkan langkah dan alur kerja algoritma *porter stemmer* (Zaman, 2014)



Gambar 2.5 Algoritma Porter Stemmer
(Ali & Ibrahim, 2012)

Porter (1980) menyatakan algoritma *porter stemmer* pada *step* 1 dibagi menjadi tiga, yaitu *step* 1a, *step* 1b, dan *step* 1c. Pada Gambar 2.5, proses "*Remove and recode plurals*" merupakan *step* 1a, proses "*Remove 'ed' or 'ing' if found" merupakan step 1b*". Kemudian, proses "*Recode 'Y' to 'I' if another vowel is present in stem*" merupakan *step* 1c.

2.6 Algoritma Porter Stemmer pada Bahasa Indonesia

Sebelum masuk ke dalam algoritma ini, sebaiknya diketahui bahwa bahasa Indonesia memiliki morfologi atau strukur kata infleksional dan derivasional.

Menurut Tala (2003), infleksional adalah strukur sederhana yang diikuti oleh imbuhan yang tidak mempengaruhi kata dasar. Strukur ini dibagi menjadi 2, yaitu *suffix* partikel dan kata ganti. Partikel berfungsi untuk memberikan penekanan pada kata. Macam-macam dari partikel adalah "-lah", "-kah", "-tah", dan "-pun". Kata ganti terdiri dari "-ku", "-mu", dan "-nya". Akhiran "-ku" sebagai kata ganti ia, "-mu" sebagai kata ganti kamu, dan "-nya" sebagai kata ganti ia.

Derivasional adalah struktur sederhana yang diikuti imbuhan yang dapat mempengaruhi kata dasar. Derivasional pada strukur bahasa Indonesia terdiri dari *prefix*, *suffix*, dan *confix*. *Prefix* yang sering muncul antara lain "ber-", "di-", "ke-", "meng-", "per-", dan "ter-". *Suffix* derivasional antara lain "-i", "-kan", dan "-an". *Confix* merupakan gabungan dari *prefix* dan *suffix*, di mana *prefix* dan *suffix* disisipkan bersama kata turunan yang baru.

Tabel 2.1 Pasangan Confix yang Tidak Diperbolehkan

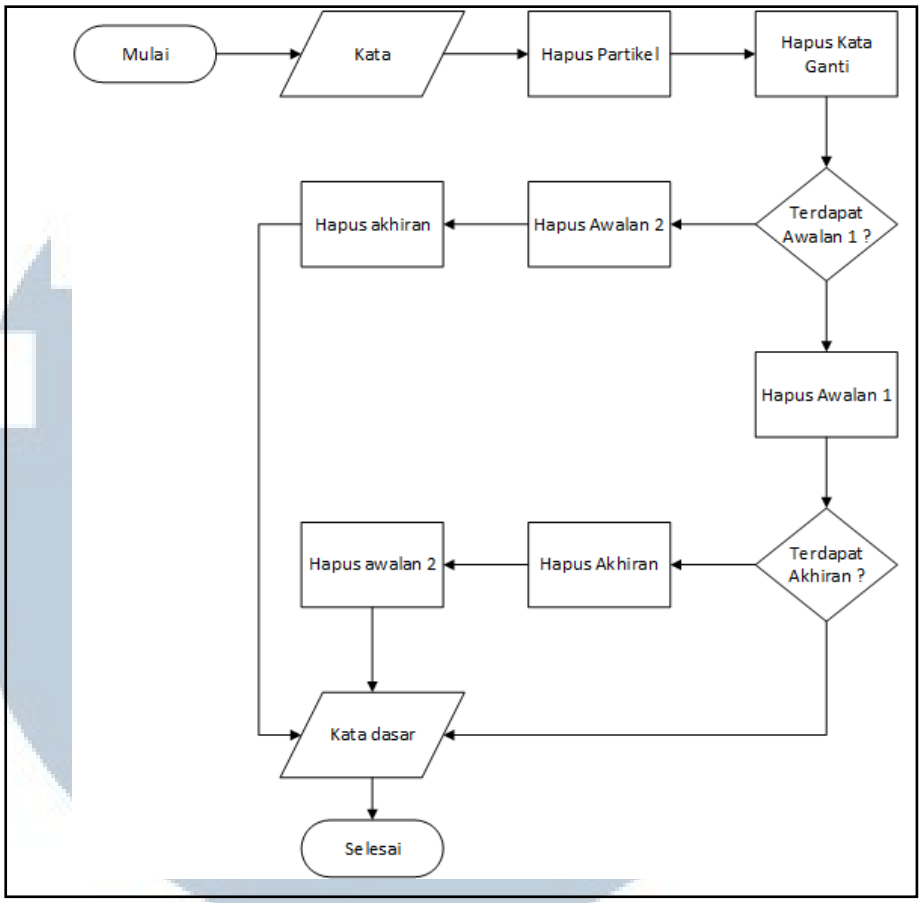
Prefix	**Suffix**
ber	i
di	an
ke	i | kan
meng	an
peng	i | kan
ter	an

Dari penjelasan di atas, dapat disimpulkan struktur derivasional didefinisikan seperti pada Gambar 2.6.

Derivasional = *prefix* | *suffix* | *confix* | *multiple prefixes*
Di mana:
Prefix = *prefix* + kata dasar
Suffix = kata dasar + *suffix*
Confix = *prefix* + kata dasar + *suffix*
Multiple prefix = (*prefix* 1 + *prefix* 2) | (*prefix* + *confix*)

Gambar 2.6 Struktur Morfologi Kata Derivasional pada Bahasa Indonesia

Menurut Zaman (2014), struktur morfologi kata Bahasa Indonesia yang terdiri dari kombinasi struktur derivasional dan infleksional yang kecil dan sederhana, sesuai dengan dengan ide dasar dari algoritma *porter stemmer*. Oleh karena itu, urutan langkah-langkah pada algoritma *porter stemmer* dapat disesuaikan dengan struktur pada kata Bahasa Indonesia sehingga dapat mengurangi tingkat kompleksitas struktur kata sehingga menghasilkan kata dasar yang benar (Zaman & Winarko, 2011). Penyesuaian tersebut meliputi perubahan pada sekumpulan aturan-aturan dan kondisi pengukuran. Algoritma *porter stemmer* hanya dapat melakukan pengurangan *suffix* (*suffix stripping*) maka beberapa aturan tambahan diperlukan untuk menangani pengurangan *prefix*, *confix*, dan juga pengejaan yang mengalami peluluhan huruf pertama pada kata dasar (Tala, 2003). Gambar 2.6 menunjukkan langkah dan alur kerja algoritma *porter stemmer* yang telah dimodifikasi untuk Bahasa Indonesia (Zaman, 2014).



Gambar 2.7. Algoritma Porter Stemmer Bahasa Indonesia (Zaman, 2014)

2.7 Term Frequency-Inverse Document Frequency (TF-IDF)

Term Frequency-Inverse Document Frequency merupakan teknik dalam memberikan bobot hubungan suatu *term* terhadap sebuah dokumen (Robertson, 2004). TF-IDF bekerja dengan memberikan bobot pada suatu *term* dalam sebuah dokumen (Satya & Murthy, 2012). Metode ini bekerja dengan menggabungkan dua konsep untuk perhitungan bobot, yaitu frekuensi kemunculan sebuah *term* di dalam sebuah dokumen (*tf*) dan inversi frekuensi dokumen (*idf*) yang mengandung kata tersebut (Anggiharto, 2013). Anggiharto (2013) menyatakan bahwa frekuensi kemunculan kata di dalam dokumen yang diberikan menunjukkan seberapa penting kata tersebut di dalam dokumen tersebut. Jumlah frekuensi dokumen yang mengandung kata tersebut menunjukan seberapa umum kata tersebut. Semakin

sedikit jumlah dokumen yang mengandung *term* yang dimaksud maka nilai pada *idf* akan semakin besar (Uden & V., 2011). Berikut rumus untuk penghitungan *term frequency* (Yates & Neto, 1999).

$$tf_{i,j} = f_{i,j} \quad \ldots(2.1)$$

Keterangan:

a. $tf_{i,j}$adalah bobot suatu *term* pada suatu dokumen

b. $f_{i,j}$ adalah banyaknya kemunculan *term* pada suatu dokumen.

$$idf_i = \log(\frac{N}{df_i}) \quad \ldots(2.2)$$

Perhitungan bobot dari *term* tertentu dalam sebuah dokumen menggunakan perkalian nilai *tf* dan *idf* menunjukkan bahwa deskripsi terbaik dari dokumen adalah *term* yang banyak muncul dalam dokumen tersebut dan sangat sedikit muncul pada dokumen yang lain. Perhitungan bobot *term frequency-inverse document frequency* adalah sebagai berikut (Yates & Neto, 1999).

$$W_{i,j} = tf_{i,j} log \frac{N}{df_i} \quad \ldots(2.3)$$

Keterangan:

a. $W_{i,j}$ adalah bobot dokumen.

b. N adalah jumlah dokumen yang terambil oleh sistem.

c. $tf_{i,j}$ adalah banyaknya kemunculan *term* t_i pada dokumen d_j.

d. df_i adalah banyaknya dokumen dalam koleksi di mana *term* t_i muncul di dalamnya.

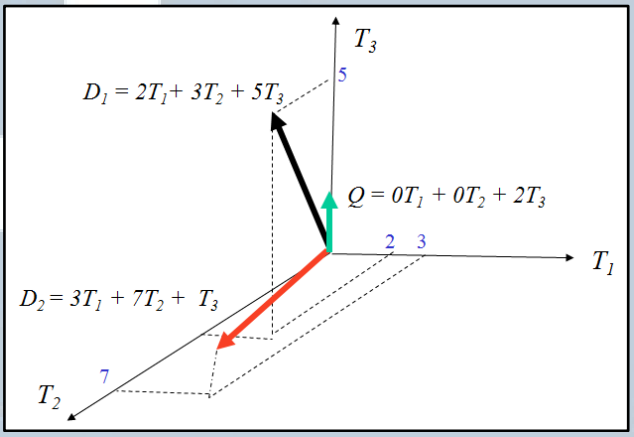
2.8 Vector Space Model

Vector Space Model (VSM) merupakan suatu metode yang digunakan untuk mengukur tingkat kedekatan atau kesamaan (*similarity*) *term* dengan cara pembobotan pada *term* (Amin, 2012). Dokumen diasumsikan sebagai sebuah vektor-vektor yang memiliki jarak (*magnitude*) dan arah (*direction*). Dalam metode ini, sebuah *term* direpresentasikan dengan sebuah dimensi dari ruang vektor. *Term* yang digunakan umumnya berdasarkan kepada *term* yag ada pada *query* atau *keyword*. Relevansi sebuah dokumen ke sebuah *query* didasarkan pada similaritas di antara vektor dokumen dan vektor *query* (Yates & Neto, 1999). Perhitungan kesamaan antara vektor *query* dengan vektor dokumen dilihat dari sudut yang paling kecil. Pada *Vector Space Model*:

a. Kamus data (*vocabulary*) merupakan kumpulan semua *term* berbeda yang tersisa dari dokumen setelah *preprocessing* dan mengandung *t term index*. Kumpulan *terms* ini membentuk suatu ruang vektor.
b. Setiap *term i* di dalam dokumen atau *query j*, diberikan suatu bobot (*weight*) bernilai real W_{ij}.
c. Dokumen dan *query* diekspresikan sebagai vektor *t* dimensi dj = (W_1, W_2, …, W_{tj}) dan terdapat *n* dokumen di dalam koleksi, yaitu *j* = 1, 2, …, *n*.

Gambar 2.8 merupakan contoh dari model ruang vektor tiga dimensi untuk 2 dokumen di mana D adalah dokumen, Q adalah *query*, dan T adalah *term* yang menjadi dimensi dari VSM. D_1 mempunyai susunan *term* $2T_1 + 3T_2 + 5T_3$, D_2 memiliki $3T_1 + 7T_2 + T_3$, dan query $Q = 0T_1 + 0T_2 + 2T_3$. D_1 digambarkan sebagai

vektor berarah berdasarkan *terms* penyusunnya. Dari dokumen-dokumen dan *query* tersebut, sudut antara *query* dengan tiap dokumen akan menentukan nilai kedekatan suatu dokumen dengan *query* yang dimasukkan. Semakin kecil sudut maka semakin besar tingkat kemiripan.



Gambar 2.8 Vector Space Model
(Annisa dkk., 2014)

Dalam *Vector Space Model*, koleksi dokumen direpresentasikan sebagai sebuah matriks *term document* (matriks *term frequency*). Setiap sel dalam matriks bersesuaian dengan bobot yang diberikan dari suatu *term* dalam dokumen yang ditentukan. Nilai nol menunjukkan *term* tersebut tidak ada dalam dokumen. Gambar 2.9 menunjukkan matriks *term document* dengan *n* dokumen dan *t term* (Amin, 2012).

	T_1	T_2	T_3	$T_{...}$	T_t
D_1	W_{11}	W_{21}	W_{31}	$\square_{...}$	T_{t1}
D_2	W_{12}	W_{22}	W_{32}	$\square_{...}$	T_{t2}
D_3	W_{13}	W_{23}	W_{33}	$\square_{...}$	T_{t3}
$D_{...}$	$\square_{...}$	$\square_{...}$	$\square_{...}$	$\square_{...}$	$\square_{\square}$
D_n	W_{1n}	W_{2n}	W_{3n}	$\square_{...}$	T_{tn}

Gambar 2.9 Matriks *term document*

Metode *Vector Space Model* memiliki beberapa tahap yaitu menghitung bobot dokumen dengan TF-IDF, menghitung jarak tiap *query* dan dokumen, menghitung dot produk, menghitung similaritas, dan membuat *ranking*. Rumus perhitungan jarak pada *query* adalah sebagai berikut (Amin, 2012).

$$|q| = \sqrt{\Sigma_{i=1}^{t} (W_{i,q})^2} \quad \ldots(2.4)$$

Keterangan:

a. $|q|$ adalah jarak *query*

b. $W_{i,q}$ adalah bobot *query* dokumen ke-*i*

Perhitungan jarak *query* $|q|$ dilakukan dengan tujuan mendapatkan jarak *query* dari bobot *query* dokumen ($W_{i,q}$) yang terambil oleh sistem. Sedangkan untuk perhitungan jarak pada dokumen, rumus yang digunakan adalah sebagai berikut (Amin, 2012).

$$|d_j| = \sqrt{\Sigma_{i=1}^{t} (W_{i,j})^2} \quad \ldots(2.5)$$

Keterangan:

a. $|d_j|$ adalah jarak dokumen

b. $W_{i,j}$ adalah bobot dokumen ke-*i*

Perhitungan jarak dokumen $|d_j|$ dilakukan dengan tujuan untuk mendapatkan jarak dokumen dari bobot dokumen-dokumen $W_{i,j}$ yang terambil oleh sistem.

2.9 Cosine Similarity

Cosine similarity merupakan metode pengukuran tingkat kemiripan yang didapatkan dari perbandingan hasil perkalian *cosine angle* 2 buah vektor. *Cosine* 0 adalah 1 dan kurang dari 1 terhadap nilai *angle* yang lain maka 2 buah vektor

dikatakan mirip ketika nilai *cosine similarity* adalah 1 (Saputra & Muttaqin, 2013). Perhitungan *cosine similarity* ditunjukkan oleh rumus berikut.

$$Sim\ (q, d_j)\ = \frac{q \times d_j}{|q| \times |d_j|} = \frac{\Sigma_{i=1}^{n}\ W_{i,q} \times W_{i,j}}{\sqrt{\Sigma_{i=1}^{t}\ (W_{i,q})^2} \times \sqrt{\Sigma_{i=1}^{t}\ (W_{i,j})^2}} \quad \ldots(2.6)$$

Keterangan:

a. $Sim\ (q, d_j)$ adalah similaritas antara *query* dan dokumen.

b. $|q|$ adalah jarak *query*

c. $|d_j|$ adalah jarak dokumen

d. $W_{i,j}$ adalah bobot dokumen ke-*i*

e. $W_{i,q}$ adalah bobot *query* dokumen ke-*i*

Similaritas antara *query* dan dokumen atau berbanding lurus terhadap hasil perkalian jumlah bobot *query* (q) dengan bobot dokumen (d_j) dan berbanding terbalik terhadap perkalian dari akar jumlah kuadrat q ($|q|$) dengan akar jumlah kuadrat dokumen $|d_j|$ (Amin, 2012). Amin (2012) menyatakan bahwa perhitungan similaritas menghasilkan bobot dokumen yang mendekati nilai 1 atau menghasilkan bobot dokumen yang lebih besar dibandingkan dengan nilai yang dihasilkan dari perhitungan *inner product*.

2.10 Precision, Recall, dan F-Measure

Menurut Hasugian (2006), dalam penilaian relevansi ada dua hal penting yang umumnya digunakan dalam mengukur kemampuan suatu sistem dalam memanggil dokumen sesuai dengan istilah yang diformulasikan, yaitu *precision* dan *recall*. Penggunaan perhitungan *precision* dan *recall* dapat menggunakan Tabel 2.2.

Tabel 2.2 Perhitungan Precision dan Recall (Hasugian, 2006)

	Relevant	Not Relevant	Total
Retrieved	A	B	A + B
Not Retrieved	C	D	C + D
Total	A + C	B + D	

Recall adalah perbandingan antara jumlah *records* relevan yang didapatkan dengan jumlah keseluruhan *records* relevan pada *database*.

$$R = \frac{Number\ of\ relevant\ items\ retrieved}{Total\ number\ of\ relevant\ items\ in\ collection} \quad \ldots(2.7)$$

Precision adalah perbandingan antara jumlah *records* relevan yang didapatkan dengan jumlah keseluruhan *records* yang relevan maupun yang tidak relevan.

$$P = \frac{Number\ of\ relevant\ items\ retrieved}{Total\ number\ of\ items\ retrieved} \quad \ldots(2.8)$$

Hasugian (2006) menyatakan *recall* sebenarnya sulit untuk diukur karena jumlah seluruh dokumen yang relevan dalam *database* sangat besar. Oleh karena itu, *precision* yang dijadikan salah satu ukuran yang digunakan untuk menilai keefektifan suatu sistem pencarian. F-*Measure* sering disebut F1 score adalah *harmonic mean* atau nilai rataan harmonis antara perhitungan *precision* dan *recall* (Hripsack dan Rothschild, 2005). F-*Measure* dapat dihitung menggunakan rumusan berikut (Hripsack dan Rothschild, 2005).

$$F1 = 2 * \frac{precision * recall}{precision + recall} \quad \ldots(2.9)$$

Nilai F-*Measure* dapat dihitung dari 0 hingga 1 atau persentase 0% - 100%.