BAB II
KAJIAN PUSTAKA

Klasifikasi teks umumnya menggunakan pendekatan *bag-of-words* dimana *term* dianggap independen satu sama lain. Pendekatan dengan model seperti ini memiliki kerugian diantaranya adalah:

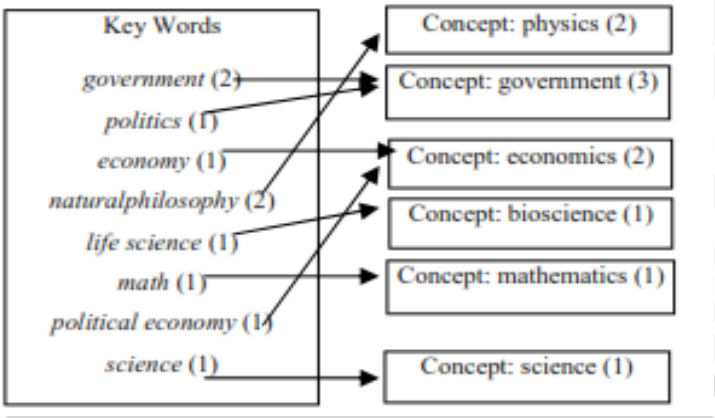
- Tidak terdapat hubungan konspetual antar *term* sehingga klasifikasi teks hanya terbatas pada mencari kesamaaan bentuk kata saja.
- Besarnya dimensi ruang vektor untuk merepresentasikan dokumen.

Penelitian yang dilakukan oleh Zakaria Elberrichi pada tahun 2008 menyatakan bahwa dua masalah pada pendekatan *bag-of-words* tersebut. Ontologi *WordNet* digunakan untuk menangani masalah hubungan antar kata sementara pendekatan statistik dengan menggunakan χ_2 (*chi-square*) multivariat untuk mengurangi dimensi dari ruang vektor. Pada penelitian tersebut pendekatan yang dilakukan adalah menggabungkan *term* dengan sinonim nya untuk membentuk representasi teks yang baru. Untuk membuat representasi teks yang baru, diperlukan 4 langkah yaitu:

- Pemetaan *term* menjadi *concepts* dan memilih strategi pemetaan yang digunakan.
- Menerapkan strategi untuk mencegah disambiguitas kata.
- Menerapkan strategi untuk kata yang berhipernim.
- Menerapkan strategi untuk pemilihan fitur (*feature selection*)

1. Pemetaan *term* menjadi *concepts*.

   Proses dari pemetaan *term* menjadi *concepts* dapat dilihat pada gambar 2.1. Misal terdapat teks yang terdiri dari kata-kata seperti pada gambar 2.1 dimana angka di dalam tanda kurung menunjukkan jumlah kemunculan kata tersebut di dalam teks. Kata-kata tersebut lalu dipetakan dengan *concepts* nya pada ontologi yaitu *WordNet.* Frekuensi kemunculan dari kata yang telah digabungkan tadi juga ikut digabungkan yang disebut dengan *concept frequency*.



Gambar 2.1 Ilustrasi Pemetaan *term* menjadi *concepts*

Sumber : [ELB-08]

2. Strategi untuk mencegah disambiguitas kata.

   Setiap kata memiliki berberapa arti sehingga kata tersebut dipetakan pada *concept* yang berbeda. Untuk mengatasi hal tersebut salah satu metode sederhana yang dapat digunakan adalah *First Concept*. Metode ini mengasumsikan bahwa ontologi *WordNet* akan memberikan *concepts list* secara berurut dimana urutan paling atas adalah *concept* yang paling umum digunakan.

3. Strategi untuk kata yang memiliki hipernim.

   Strategi yang digunakan untuk kata yang memiliki hipernim adalah dengan strategi *Adding Hypernyms*. Strategi ini menambahkan *concept frequency* untuk setiap *concepts* pada teks dengan frekuensi hiponim yang muncul.

4. Menerapkan strategi untuk pemilihan fitur (*feature selection*).

   Pemilihan fitur untuk representasi teks pada ruang vektor merupakan hal yang penting untuk dilakukan. Salah satu alasan mengapa pemilihan fitur merupakan hal yang penting karena pada banyak algoritma pembelajaran (*learning*) kompleksitasnya bergantung tidak hanya pada banyaknya jumlah data latih yang digunakan tetapi juga jumlah fitur nya. Salah satu metode yang digunakan untuk pemilihan fitur adalah χ_2 (*chi-square*) multivariat.

2.1 Klasifikasi Dokumen

Klasifikasi dokumen adalah bidang penelitian dalam perolehan informasi yang mengembangkan metode untuk menentukan atau mengkategorikan suatu

dokumen ke dalam satu atau lebih kelompok yang telah dikenal sebelumnya secara otomatis berdasarkan isi dokumen. Klasifikasi dokumen bertujuan untuk mengelompokkan dokumen yang tidak terstruktur ke dalam kelompok-kelompok yang menggambarkan isi dari dokumen. Dokumen dapat berupa teks seperti artikel berita [BAS-10].

Klasifikasi teks atau *text categorization* secara garis besar terbagi dalam tiga kelompok yaitu yang pertama adalah klasifikasi berbasis statistik. Metode yang digunakan pada kelompok ini adalah *Naïve Bayesian, K-Nearest Neighbor, Category Center Vector, Support Vector Machine,* dan model *Maximum Entropy*. Kelompok kedua adalah klasifikasi teks berbasiskan keterkaitan dan metode yang digunakan adalah jaringan syaraf tiruan. Kelompok ketiga adalah klasifikasi teks berbasiskan *rule*atau aturan. Metode yang digunakan adalah *decision tree* [XIA-09].

Tiga komponen utama dari klasifikasi teks adalah *preprocessing* data, pembangunan *classifier*, dan klasifikasi dokumen. *Preprocessing* data adalah mengubah representasi dokumen yang nantinya akan digunakan sebagai data latih, proses validasi, dan proses klasifikasi. Pembangunan *classifier* merupakan proses pembelajaran dokumen uji terhadap dokumen latih dan klasifikasi dokumen adalah menempatkan dokumen ke dalam kelas tertentu [GUO-04].

2.2 Berita

Dokumen atau teks yang digunakan pada klasifikasi teks salah satunya adalah dokumen berita. Berita (*news*) adalah laporan mengenai suatu peristiwa atau kejadian yang terbaru (aktual). Laporan mengenai fakta-fakta yang aktual, menarik perhatian, dinilai penting, atau luar biasa. Kecuali itu, masih banyak batasan lain mengenai berita. Beberapa batasan yang terkenal adalah kata *news* berasal dari kata *new* yang berarti ‘baru’ dan nilai yang ditekankan di sini adalah kebaruan (aktualitas). Kata *news* bila dijabarkan huruf pembentuknya adalah empat mata penjuru mata angin yaitu *N-North, E-East, W-West, S-South*. Artinya, sebuah berita menghimpun segala keterangan dari mana pun, dari berbagai sumber, dari keempat penjuru mata angin. Jenis-jenis berita diantaranya adalah *hard news* yang berisi

laporan peristiwa politik, ekonomi, masalah social, dan kriminalitas. Berita mengenai olahraga, kesenian, hiburan, hobi, elektronika dan sebagainya bisa dikategorikan sebagai *soft news* [BUD-05].

2.3 *Data Mining*

Klasifikasi dokumen berita merupakan salah satu kegunaan dari *Data Mining*. *Data Mining* adalah proses menemukan dan menggali pengetahuan yang baru dan berpotensi berguna dari data. *Data mining* juga dikenal dengan sebutan *Knowledge Discovery in Database (KDD)*. *Data mining* adalah bidang interdisipliner seperti system basis data, statistic, *machine learning*, visualisasi data dan pengambilan informasi. *Data mining* memiliki dua tujuan utama yaitu prediksi dan deskripsi. Prediksi mencakup penggunaan beberapa variabel atau *fields* pada database untuk memprediksi nilai prediksi dari suatu variabel. Tujuan deskripsi dari *data mining* mengkhususkan pada pencarian pola yang mendeskripsikan data yang bisa diinprestasi manusia [ELS-05].

Data mining adalah salah satu bidang yang berkembang sangat cepat pada industri komputer. Salah satu kelebihan *data mining* adalah banyaknya metode dan teknik yang bisa diaplikasikan pada sekumpulan masalah. Keberhasilan dari penggunaan *data mining* bergantung dari kemampuan, kreativitas dari designer data mining itu sendiri. Esensi dari data mining adalah memecahkan seperti memecahakan sebuah *puzzle* [KAN-03].

2.4 *Text Mining*

Text mining adalah penemuan informasi baru secara otomatis yang bersumber dari teks. *Text mining* dimulai dari penggalian fakta dan kejadian dari sumber yang berupa teks lalu kemudian membentuk hipotesis baru yang lebih lanjut akan dikaji menggunakan metode *Data Mining*. *Text mining* dikenal juga dengan sebutan *Knowledge Discovery from Text (KDT)*, yang mengacu pada proses penggalian pola dari database yang sangat besar untuk tujuan penemuan pengetahuan [BEN-01]. Fungsi dari *Text Mining* diantaranya adalah sebagai berikut:

- Pengambilan informasi yang relevan dari dokumen menggunakan *Natural Language (NL), Information Retrieval (IR)* dan algoritma matriks berasosiasi.
- Menemukan tren atau hubungan antar orang/tempat/organisasi dengan melakukan agregasi dan membandingkan informasi yang diambil dari dokumen.
- Mengklasifikasikan dan mengorganisir dokumen berdasarkan isinya.
- Mengambil dokumen berdasarkan isi dari dokumen tersebut.
- Melakukan *clustering* dokumen tersebut berdasarkan isinya.

2.5 Tahap *Preprocessing* Text

Langkah awal pada *Text Mining* adalah dokumen melalui tahap preprocessing. Tahap *preprocessing* adalah tahap dimana dokumen baik dokumen latih dan dokumen uji di olah sehingga menjadi bentuk yang dapat digunakan untuk proses klasifikasi. Proses ini terdiri dari *case folding, tokenizing, filtering, stemming* dan tahap pembobotan kata yang dilakukan secara sekuensial.

2.5.1 *Case Folding*

Case folding adalah proses mengubah semua karakter huruf pada dokumen menjadi huruf kecil. Hanya huruf ‘a’ sampai dengan ‘z’ yang diterima. Karakter selain huruf tersebut akan dihilangkan dan dianggap sebagai *delimiter* (pembatas seperti spasi).

2.5.2 *Tokenizing*

Setelah melalui proses *case folding*, langkah selanjutnya adalah proses *tokenizing. Tokenizing* adalah proses memotong sebuah urutan karakter menjadi sebuah potongan-potongan yang disebut *token*. Pada proses ini juga dilakukan pembuangan tanda baca.

2.5.3 *Filtering*

Filtering adalah proses menghilangkan kata-kata yang tidak merepresentasikan (tidak penting) dokumen sehingga kata-kata yang digunakan adalah kata-kata yang menggambarkan isi dokumen dan membedakan dokumen dengan dokumen lain yang ada pada *corpus*. Proses ini dilakukan setelah

tokenizing. Metode *filtering* yang digunakan bisa menggunakan algoritma *stoplist* (membuang kata yang kurang penting) atau *wordlist* (menyimpan kata penting).

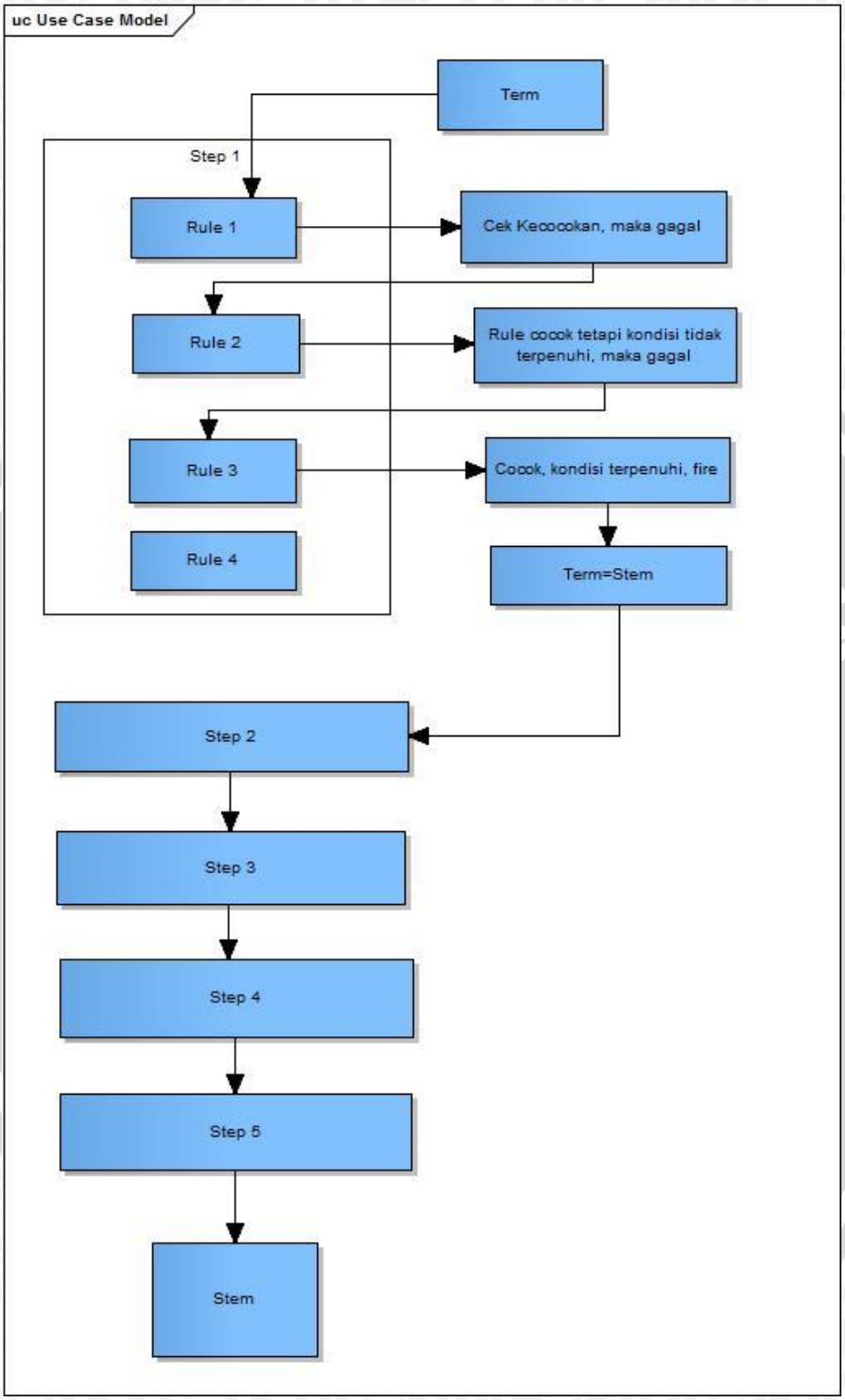
Stoplist atau *stopword* adalah kata-kata yang tidak deskriptif yang dapat dibuang dalam dokumen. Contoh *stopwords* pada bahasa Inggris adalah "*a*", "*same*", "*there*", "*in*", "*and*", dan sebagainya. Daftar *stopwords* yang digunakan dapat dilihat pada lampiran 1. Daftar *stopwords* yang digunakan adalah berasal dari http://www.ranks.nl/resources/stopwords.html.

2.5.4 *Stemming*

Stemming adalah proses pemotongan berbagai imbuhan atau *affixes* seperti awalan (*prefixes*), sisipan (*infixes*), akhiran (*suffixes*) dan kombinasi awalan dan akhiran (*confixes*). *Stemming* dilakukan setelah proses *tokenizing*. Tujuan dari *stemming* ini adalah mengurangi bentuk jamak atau bentuk turunan yang berhubungan dengan kata dasar yang umum [MAN-09].

Algoritma yang umum digunakan untuk *stemming* bahasa Inggris adalah *Porter Stemmer*. Algoritma ini dibuat oleh M.F.Porter pada tahun 1980. Secara garis besar algoritma ini melakukan penghilangan akhiran *morphological* dan *infleksional* yang umum dari bahasa Inggris [POR-80].

Algoritma *Porter-Stemmer* terdiri dari lima langkah linier. Pada setiap langkah terdapat beberapa *rule* dan kondisi untuk penghapusan akhiran. Jika aturan *suffix* sesuai dengan *term*, maka kondisi yang ada di dalam rule tersebut akan di cek kondisi mana yang memenuhi, jika sudah memenuhi kondisi maka *rule* akan di-*fire*. Contoh dari kondisi di dalam *rule* yaitu jumlah karakter vokal yang diikuti oleh karakter konsonan pada *stem* harus lebih besar daripada *rule* yang akan dijalankan. Ilustrasi dari algoritma *porter-Stemmer* dapat dilihat pada gambar 2.2. [POR-80]



Gambar 2.2 Ilustrasi proses algoritma *Porter-Stemmer*

Sumber : [POR-80]

Langkah-langkah algoritma *Porter-Stemmer* adalah sebagai berikut:

1. *Step* 1a : Menghilangkan *plural suffixation*.
2. *Step* 1b : Menghapus *verbal inflection*.
3. *Step* 1b1 : Mengubah *term* yang berimbuhan *–ed* dan *–ing*
4. *Step* 1c : Mengubah akhiran *–y* menjadi *-i*
5. *Step* 2 : mengganti akhiran *–ational* menjadi *–ate*, *-tional* menjadi *–tion*, *-enci* menjadi *–ence*, dsb
6. *Step* 3 : pemotongan terhadap *stem* yang berakhiran *–icate*, *-ative*, *-alize*, *-iciti*, *-ical*, *-ful*, dan *–ness*
7. *Step* 4 : penghapusan akhiran *–al*, *-ance*, *-ence*, *-er*, *-ic*, dsb
8. *Step* 5a : pemotongan terhadap huruf *–e*,
9. *Step* 5b : reduksi terhadap huruf konsonan dobel.

Aturan pemilihan *suffix* (akhiran) pada setiap *step* akan dijelaskan pada tabel 2.1 sampai tabel 2.9.

Tabel 2.1 *Stemming Step* 1a

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
NULL	*sses*	*ss*	*caresses -> caress*
NULL	*ies*	*i*	*ponies -> poni*
			ties -> tie
NULL	*ss*	*ss*	*caress -> caress*
NULL	*s*	*NULL*	*cats -> cat*

Tabel 2.2 *Stemming Step* 1b

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>0)	*eed*	*ee*	*feed -> feed*
			agreed -> agree
*(*v*)*	*ed*	*NULL*	*plasteres -> plaster*
			bled -> bled
*(*v*)*	*ing*	*NULL*	*motoring -> motor*
			sing -> sing

Tabel 2.3 *Stemming Step* 1b1

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
NULL	*at*	*ate*	*conflat(ed) -> conflate*
NULL	*bl*	*ble*	*troubl(ing) -> trouble*
NULL	*iz*	*ize*	*siz(ed) -> size*
*(*d and not (*<L> or *<S> or *<Z>))*	*NULL*	*single letter*	*hopp(ing) -> hop*
			tann(ed) -> tan
			fall(ing) -> fall
			hiss(ing) -> hiss
			fizz(ed) -> fizz
*(m=1 and *o)*	*NULL*	*e*	*fail(ing) -> fail*
			fil(ing) -> file

Tabel 2.4 *Stemming Step* 1c

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
*(*v*)*	*y*	*i*	*happy -> happi*
			sky -> sky

Tabel 2.5 *Stemming Step 2*

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>0)	*ational*	*ate*	*relational -> relate*
(m>0)	*tional*	*tion*	*conditional -> condition*
			rational -> rational
(m>0)	*enci*	*ence*	*valenci -> valence*
(m>0)	*anci*	*ance*	*hesitanci -> hesitance*
(m>0)	*izer*	*ize*	*digitizer -> digitize*
(m>0)	*abli*	*able*	*conformabli -> conformable*
(m>0)	*alli*	*al*	*radicalli -> radical*
(m>0)	*ently*	*ent*	*differently -> different*
(m>0)	*eli*	*e*	*vileli -> vile*
(m>0)	*ousli*	*ous*	*analogousli -> analogous*
(m>0)	*ization*	*ize*	*vietnamization -> vietnamize*
(m>0)	*ation*	*ate*	*predication -> predicate*
(m>0)	*ator*	*ate*	*operator -> operate*
(m>0)	*alism*	*al*	*feudalism -> feudal*
(m>0)	*iveness*	*ive*	*decisiveness -> decisive*
(m>0)	*fullness*	*ful*	*hopefulness -> hopefull*
(m>0)	*ousness*	*ous*	*callousness -> callous*
(m>0)	*aliti*	*al*	*formality -> formal*
(m>0)	*iviti*	*ive*	*sensitivity -> sensitive*
(m>0)	*bility*	*ble*	*sensibility -> sensible*

Tabel 2.6 *Stemming Step* 3

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>0)	*icate*	*ic*	*triplicate -> triplic*
(m>0)	*ative*	*NULL*	*formative -> form*
(m>0)	*alize*	*ai*	*formalize -> formal*
(m>0)	*iciti*	*ic*	*electricity -> electric*
(m>0)	*ical*	*ic*	*electrical -> electric*
(m>0)	*ful*	*NULL*	*hopeful -> hope*
(m>0)	*ness*	*NULL*	*goodness -> good*

Tabel 2.7 *Stemming Step* 4

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>1)	*al*	*NULL*	*revival -> reviv*
(m>1)	*ance*	*NULL*	*allowance -> allow*
(m>1)	*ence*	*NULL*	*inference -> infer*
(m>1)	*er*	*NULL*	*airliner -> airlin*
(m>1)	*ic*	*NULL*	*gyroscopic -> gyroscop*
(m>1)	*able*	*NULL*	*adjustable -> adjust*
(m>1)	*ible*	*NULL*	*defensible -> defens*
(m>1)	*ant*	*NULL*	*irritant -> irrit*
(m>1)	*ement*	*NULL*	*replacement ->replac*
(m>1)	*ment*	*NULL*	*adjustment -> adjust*
(m>1)	*ent*	*NULL*	*dependent -> depend*
(m>1)	*ion*	*NULL*	*adoption -> adopt*
(m>1)	*ou*	*NULL*	*homologou -> homolog*
(m>1)	*ism*	*NULL*	*communism -> commun*
(m>1)	*ate*	*NULL*	*activate -> activ*

(m>1)	*iti*	*NULL*	*angularity -> angular*
(m>1)	*ous*	*NULL*	*homologous -> homolog*
(m>1)	*ive*	*NULL*	*effective -> effect*
(m>1)	*ize*	*NULL*	*bowdlerize -> bowdler*

Tabel 2.8 *Stemming Step* 5a

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
(m>1)	*e*	*NULL*	*probate -> probat*
			rate -> rate
*(m=1 and not *o)*	*e*	*NULL*	*cease -> ceas*

Tabel 2.9 *Stemming Step* 5b

Kondisi	**Akhiran**	**Pengganti**	**Contoh**
*(m=1 and not *d and *<L>)*	*NULL*	*single letter*	*controll ->control*
			roll -> roll

Keterangan:

m : Ukuran (*measure*) dari sebuah stem berdasarkan urutan vokal-konsonan

*<*X*>* : berarti *stem* berakhir dengan huruf X

v : berarti *stem* mengandung sebuah vokal

**d* : berarti *stem* diakhiri dengan konsonan dobel

**o* : berarti *stem* diakhiri dengan konsonan – vokal – konsonan, berurutan, di mana konsonan akhir bukan w, x, atau y

2.5.5 Ontologi

Proses ontologi dilakukan setelah *term* melalui proses stemming. Kata "ontologi" telah dikenal pada bidang filsafat sebagai subjek dari keberadaan. Pada konteks kecerdasan buatan, ontologi berarti spesifikasi eksplisit dari konseptualisasi yang saling berbagi. Ontologi digunakan untuk berbagi pengetahuan. Ontologi meningkatan informasi dan pengertian. Ontologi memiliki peran penting pada informasi berbasis komputer yang bersifat heterogen. Ontologi memiliki peran penting pada *web* generasi kedua yang oleh Tim Berners-Lee sebut sebagai *"Semantic Web"*. Mesin pencari atau *search engine* akan menemukan halaman dengan kata yang secara sintaksis berbeda tetapi memiliki kesamaan secara semantik [ELS-05].

Ontologi adalah sebuah deskripsi formal tentang sebuah konsep secara eksplisit dalam sebuah domain, property dari setiap konsep beserta dengan batasannya. Sebuah konsep di ontologi dapat memiliki objek (*instance*). Secara teknis, ontologi direpresentasikan dalam bentuk *class, property, facet*, dan *instances*. *Class* menerangkan konsep atau makna dari suatu domain. *Class* adalah kumpulan dari elemen dengan sifat yang sama. Sebuah *class* bisa memiliki sub class yang menerangkan konsep yang lebih spesifik [BAS-10].

2.5.6 *WordNet*

Implementasi ontologi pada klasifikasi teks dapat dibantu dengan menggunakan *WordNet* sebagai basis ontologi. *WordNet* adalah database leskikal untuk bahasa Inggris. Kata benda, kata kerja, kata sifat, dan kata keterangan dikumpulkan menjadi sekumpulan kata yang bersinonim dan dinamakan *synsets*. *WordNet* menyerupai thesaurus, yang mengumpulkan kata berdasarkan maknanya. Tetapi, ada beberapa perbedaan pada *WordNet*. Pertama, *WordNet* menghubungkan tidak hanya antar bentuk kata, *string* dari huruf, tetapi secara spesifik menghubungkan makna dari kata tersebut. Kedua, *WordNet* memberikan nama hubungan semantik antar kata, dimana pengelompokkan kata pada thesaurus tidak mengikuti banyak pola.

WordNet terdiri dari dari hubungan semantik sebagai berikut :

- *Synonymy* adalah relasi dasar pada *WordNet*, karena WordNet menggunakan himpunan sinonim yang disebut *synsets* untuk merepresentasikan *senses* kata. *Synonymy* merupakan relasi simetris antar kata.
- *Antonymy* adalah lawan kata dan merupakan hubungan semantik simetris antar kata khususnya pada saat pengelompokkan arti dari kata sifat dan kata keterangan.
- *Hyponymy* (sub-kelas) dan kebalikannya *hypernymy* (super-kelas) adalah hubungan transitif antar *synsets.*
- *Meronymy* (bagian-kelas) dan kebalikannya *holonymy* (keseluruhan-kelas) adalah hubungan semantik yang kompleks pada *WordNet*.
- *Troponymy* adalah untuk kata kerja dimana pada *hyponymy* adalah kata benda.
- *Entailment* adalah hubungan antar kata kerja.

Pada *WordNet,* sebuah kata direpresentasikan string dari karakter ASCII. *WordNet* terdiri dari lebih dari 118.000 kata yang berbeda dan lebih dari 90.000 *senses* [MIL-93].

Pada penelitian ini hubungan semantik yang digunakan adalah synonymy. Sekumpulan kata yang saling bersinonim digabungkan dalam sebuah *synset.* Definisi dari relasi *synonymy* sendiri adalah dua buah ekspresi atau kata dikatakan saling bersinonim jika kata tersebut saling menggantikan dalam suatu kalimat maka tidak akan mengubah makna dari kalimat tersebut. Contoh dari *synset* dari kata *person* adalah [*individual, someone, somebody, mortal, human, drunk person*]. Jika ada suatu kalimat yang memiliki kata *person* di dalamnya kemudian kata *person* diganti dengan *synset* nya maka kalimat tersebut tidak akan berubah makna.

2.5.7 *RiTaWordNet*

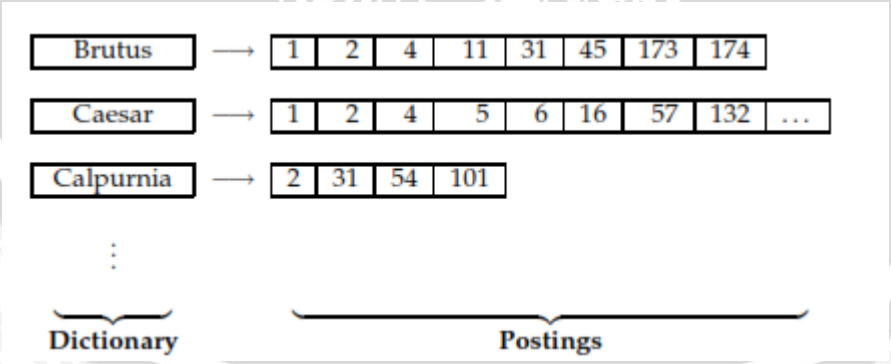
Jika implementasi dari algoritma klasifikasi teks menggunakan bahasa pemrograman *Java* dengan *WordNet* sebagai basis ontologi nya, maka dibutuhkan *interface* yang menghubungkan database *WordNet* dengan aplikasi yang menggunakan bahasa pemrograman Java. Terdapat beberapa *interface WordNet*

untuk bahasa pemrograman *Java* dan *interface* yang digunakan pada penelitian ini adalah *RiTaWordNet*. Dokumentasi API (*Application Program Interface*) dari *RiTaWordNet* beserta cara penggunaanya cukup lengkap dan dapat diakses melalui www.rednoise.org/rita/wordnet/documentation/

2.5.8 *Dictionary Construction*

Proses *Dictionary Construction* adalah proses konversi dokumen teks menjadi vektor fitur. *Term* yang dimasukkan ke dalam vektor fitur adalah *term* yang sudah melalui proses *Stemming*. Setiap fitur di dalam vektor berkorespondensi dengan kata pada *dictionary* [GUO-04].

Metode yang biasa digunakan untuk *Dictionary Construction* adalah *Inverted Index* atau disebut juga *Inverted File*. *Inverted index* terdiri dari dua bagian yaitu *term list* dan *posting list*. *Term-term* yang ada dari dokumen latih disebut *term* list. Istilah kamus atau *dictionary* adalah struktur data yang digunakan sementara sekumpulan *term* disebut *vocabulary*. Untuk setiap *term*, akan terbentuk *list* yang menyimpan pada dokumen mana *term* tersebut muncul. Setiap *item* dari *list* tersebut yang menyimpan jumlah *term* tersebut muncul pada sebuah dokumendisebut *posting*. *List* dari *posting-posting* tersebut disebut *porsing list* atau *inverted list* [MAN-09]. Ilustrasi dari *inverted index* dapat dilihat pada gambar 2.3



Gambar 2.3 Ilustrasi *Inverted Index*

Sumber : [MAN-09]

2.5.9 *Feature Selection*

Ruang vektor yang terbentuk berdasarkan hasil dari *Dictionary Construction* akan memiliki ruang fitur yang besar dimensinya dikarenakan banyaknya term yang ada di dalam kumpulan dokumen. Tiap ruang fitur didefinisikan sebagai dimensi vektor. Ruang fitur tersebut berisi *term-term* unik yang terdapat pada dokumen-dokumen yang jumlahnya bisa mencapai seratus ribuan untuk *corpus* sedang. Dimensi yang banyak tersebut akan mempengaruhi performa dari klasifikasi teks tersebut. Untuk mengatasi hal tersebut maka perlu dilakukan pengurangan dimensi vektor tetapi tanpa mengorbankan akurasi dari klasifikasi tersebut. *Feature Selection* adalah sebuah proses pemilihan *subset* fitur dari fitur asli [YIM-02].

Berdasarkan ada atau tidaknya informasi label, *feature selection* dapat dibedakan menjadi dua jenis yaitu *supervised feature selection* dimana dibutuhkannya informasi label dan *unsupervised feature* dimana informasi label tidak dibutuhkan [YIM-02].

Salah satu metode *unsupervised feature selection* adalah *documen frequency thresholding*. *Document frequency* adalah jumlah kemunculan *term* tertentu dalam sejumlah dokumen. Metode ini menghitung frekuensi dari setiap *term* unik pada dokumen latih dan menghilangkan *term* tersebut dari ruang fitur jika frekuensinya di bawah *threshold* (batas) yang telah ditentukan. Asumsi yang digunakan adalah *term* yang jarang muncul tidak akan mempengaruhi performa klasifikasi secara keseluruhan. Selain itu, penghilangan *term* yang jarang muncul ini juga akan mengurangi dimensi dari ruang fitur. Peningkatan akurasi juga memungkinkan untuk terjadi jika *term* yang jarang muncul tersebut merupakan *term* yang dapat menyebabkan *noise*. DF *thresholding* adalah teknik paling sederhana untuk pengurangan kata dengan kompleksitas komputasi berbanding linear dengan jumlah dokumen latih [YIM-02].

2.5.10 *Feature Weight* (Pembobotan)

Setelah dimensi vektor dikurangi pada proses *Feature Selection*, maka vektor dokumen tersebut dapat dihitung bobot tiap *term* nya untuk proses

klasifikasi. Feature *weighting* (pembobotan) adalah proses memberikan bobot pada tiap-tiap *term.* Metode yang paling banyak digunakan adalah *Term Frequency-Inverse Document Frequency* (TFIDF). Fungsi dari *term weighting* ini yang dikenal dengan IDF ini diperkenalkan oleh Karen S,Jones pada tahun 1972 [ROB-03].

TFIDF biasa digunakan pada model ruang vektor pada dua metode kategorisasi teks yaitu KNN dan SVM. Fungsi dari TFIDF adalah memberi bobot pada setiap komponen vektor. *Term Frequency* adalah pembobotan *term* berdasarkan perhitungan jumlah *term* yang muncul pada suatu dokumen. Semakin tinggi nilai TF (*Term frequency*) maka semakin penting pula *term* tersebut untuk mendeskripsikan suatu dokumen[SOU-03].

Inverse Document Frequency (IDF) adalah proses mengukur *term* yang jarang muncul pada *corpus*. Penilaiannya menggunakan seluruh dokumen latih yang digunakan. Jika sebuah *term* sering muncul, maka *term* tersebut tidak bisa dianggap sebagai *term* yang mewakili dokumen tersebut. Sebaliknya, jika *term* yang jarang muncul pada corpus, maka *term* tersebut dikatakan memiliki hubungan dengan dokumen. TFIDF mengalikan nilai TF dan IDF. Rumus untuk menghitung TFIDF dapat dilihat pada persamaan 2-1 sementara rumus untuk menghitung IDF dapat dilihat pada persamaan 2-2. [SOU-03]

$$w_{ij} = tf_{ij} \, . \, idf_i \qquad (2\text{-}1)$$

Dimana,

w_{ij} = bobot *term* i pada dokumen j.

tf_{ij} = frekuensi *term* i pada dokumen j.

$$idf_i = log_2\left(\frac{N}{df_i}\right) \qquad (2\text{-}2)$$

Dengan

N = jumlah keseluruhan dokumen.

df_i = banyaknya dokumen yang mempunyai *term* i.

Sehingga persamaan bobot tfidf dapat dituliskan seperti pada persamaan 2-3 dengan menggunakan logaritma basis 2.

$$w_{ij} = tf_{ij} \, . \, log_2 \left(\frac{N}{df_i}\right) \quad (2\text{-}3)$$

2.6 *Classifier Construction*

Proses klasifikasi dilakukan setelah setiap *term* pada dokumen dihitung bobotnya. Tahap pertama dari proses klasifikasi ini adalah *Classifier Construction* . *Classifier Construction* adalah komponen utama pada kategorisasi teks. Peran dari komponen ini adalah untuk membangun *classifier* dengan melakukan pembelajaran pada dokumen yang sudah ada dan nantinya akan digunakan untuk mengklasifikasikan dokumen yang belum diketahui kelasnya [GUO-04].

2.6.1 Model Ruang Vektor (*Vector Space Model*)

Term dan bobot dari *term* tersebut direpresentasikan dalam model ruang vektor. Model ruang vektor diperkenalkan oleh Gerard Salton pada tahun 1960. Setiap dokumen direpresentasikan sebagai fitur vektor dari kemunculan *term* pada semua dokumen. Model ruang vektor ini merupakan bagian penting pada *information retrieval*. Operasi vektor yang dilakukan pada model ruang vektor ini adalah *dot product*. Sekumpulan dokumen bisa dipandang sebagi sekumpulan vektor pada ruang vektor, yang berarti ada satu sumbu untuk setiap *term*. Untuk mengukur kemiripan dari dua dokumen *d1* dan *d2* misalnya, rumus *similarity* yang digunakan adalah *cosine similarity* [MAN-09].

Penggunaan *cosine similarity* dalam menghitung kemiripan dokumen memiliki keuntungan dalam efisiensi waktu apabila vektor dokumen ditransformasikan ke dalam vektor satuan, dibandingkan dengan metode lain seperti *jaccard, euclidean distance* dan *pearson correlation.* Metode ini juga bisa mengatasi efek dari panjang dokumen yang berpengaruh jika dua dokumen dengan isi yang sangat mirip bisa menjadi berbeda karena panjang vektor yang berbeda pula [MAN-09]. Rumus dari *cosine similarity* dapat dilihat pada persamaan 2-4 yaitu sebagai berikut:

$$CosSim(q, d_j) = \frac{\vec{d}_j \cdot \vec{q}}{\left|\vec{d}_j\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t}(w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}} \quad (2\text{-}4)$$

Dimana :

– $CosSim(q, d_j)$ = *cossine similarity* dokumen uji q terhadap dokumen latih j

– w_{ij} = bobot *term* i pada dokumen latih j

– w_{iq} = bobot *term* i pada dokumen uji q

- t = banyaknya *term*

Pembilang dari persamaan diatas adalah *dot product* (*inner product*) dari dokumen uji dan dokumen latih, sementara penyebut nya adalah hasil *eucledian distance* antar vektor tersebut.

2.6.2 Algoritma *Bubble Sort*

Nilai dari *cosine simmilarity* setiap dokumen latih dengan dokumen uji diurutkan dari nilai yang dari yang terbesar sampai dengan terkecil. Proses pengurutan nilai tersebut dilakukan dengan algoritma sorting. Salah satu algoritma *sorting* yang sederhana adalah algoritma *bubble sort*. Nama algoritma *bubble sort* diambil dari ide dasar algoritma ini yang bekerja berdasarkan prinsip gelembung. Teknik *bubble sort* menganalogikan setiap angka sebagai sebuah gelembung. Angka yang lebih kecil naik ke atas sama seperti gelembung yang selalu akan mengambang di atas permukaan air. *Pseudocode* dari algoritma *bubble sort* dapat dilihat pada gambar 2.4 [SET-08].

```
for i=1 to (N-1) do
      for j=N  to i-1 do
          if A[j-1]>A[j] then
               swap (A[j-1],A[j])
           end if
        end for
end for
```

Gambar 2.4 Pseudocode algoritma *Bubble Sort*

Sumber : [Kajian Pustaka]

Dalam setiap perulangan ke-`i`, algoritma akan "menggelembungkan" bilangan terkecil ke-`i` hingga posisi ke-`i` dalam *array*. Penggelembungan dimulai dari elemen terakhir *array*. Setiap pasang elemen yang berdekatan akan dibandingkan. Jika elemen yang di belakang lebih kecil, maka dilakukan penukaran. Dengan cara ini, bilangan terkecil pasti akan "naik terbawa" hingga ke posisi sebenarya [SET-08].

2.6.3 Algoritma *K-Nearest Neighbor* (KNN)

Algoritma *k nearest neighbor* (KNN) adalah algoritma pembelajaran berdasarkan nilai similaritas dan dikenal sangat efektif menangani masalah pada berbagai macam domain termasuk kategorisasi teks. Misalnya ada sebuah dokumen uji d_1, maka algoritma *k-NN* akan menemukan sejumlah *k* tetangga terdekat pada dokumen uji. Setelah menentukan *k*, maka nilai dari *cosine similarity* yang telah diurutkan dengan algoritma *bubble sort* akan diambil sejumlah *k* teratas. Untuk menentukan kategori dari dokumen uji digunakan sistem *scoring* [GUO-04].

Paramater *k* pada KNN bisa menggunakan *k* yang bernilai ganjil seperti *k* = 3 dan *k* = 5, hingga mencapai sejumlah data latih. Algoritma ini juga dapat memberi bobot dengan melakukan "scoring" kategori dari nilai *cosine similarity* antar vektor dokumen tersebut [MAN-09]. Pemberian skor untuk setiap kategori pada dokumen latih dapat dilihat pada persamaan 2-5. Persamaan 2-6 merupakan pemberian bobot dari setiap kategori dokumen.

$$score(q, c_i) = \sum_{d_j \epsilon KNN(q)} Sim(q, d_j) \,.\, \delta(d_j, c_i) \qquad (2\text{-}5)$$

Dimana :

- $score(q, c_i)$= skor dokumen uji q terhadap kategori i
- $d_j \epsilon$ KNN(q) = dokumen latih d_j yang berada pada kumpulan tetangga terdekat (*nearest neighbor*) dari dokumen uji q.
- $Sim(q, d_j)$ = *cossine similarity* antara dokumen uji q dengan dokumen latih d_j
- $\delta(d_j, c_i) = \begin{cases} 1 & d_j \epsilon c_i \\ 0 & d_j \notin c_i \end{cases}$ (2-6)
- c_i = kategori/kelas i

2.7 Evaluasi

Pada penelitian *text categorization* evaluasi dilakukan ketika proses klasifikasi apakah metode yang digunakan telah mengklasifikasi secara benar.Evaluasi ini biasanya membutuhkan sebuah matrik yang disebut dengan *matrix confusion*. *Matrix confusion* berisi informasi tentang klasifikasi yang aktual dan terprediksi yang dilakukan oleh sistem. Pada penelitian ini metode evaluasi yang digunakan adalah *recall*, *precision*, dan *F1-Measure* [SEB-02].

Recall adalah jumlah dokumen yang terklasifikasi dengan benar oleh sistem dibagi dengan jumlah dokumen yang seharusnya bisa dikenali sistem. *Precision* adalah jumlah jumlah dokumen yang diklasifikasikan dengan benar oleh sistem dibagi dengan jumlah keseluruhan klasifikasi yang dilakukan oleh sistem.*F1-measure* merupakan nilai yang mewakili kinerja keseluruhan sistem dan merupakan penggabungan nilai *recall* dan *precision.* Tabel 2.10 menggambarkan *matrix confusion* dan rumus untuk menghitung *precision* dapat dilihat pada persamaan 2-7. Rumus untuk menghitung *recall* dan *f1-measure* dapat dilihat pada persamaan 2-8 dan 2-9 [DES-09].

Tabel 2.10 Kesesuaian Hasil Kategori

		Hasil klasifikasi dari ahli	
		YES	NO
Hasil klasifikasi dari sistem	YES	*True positive (TP)*	*False Positive (FP)*
	NO	*False Negative (FN)*	*True Negative (TN)*

$$Precision = \frac{TP}{TP+FP} \quad (2\text{-}7)$$

$$Recall = \frac{TP}{TP+FN} \quad (2\text{-}8)$$

$$F1 - Measure = \frac{2\ x recall x precision}{recall+precision} \quad (2\text{-}9)$$

Dimana :

- ***True positive*** : Contoh positif dikenali dengan benar sebagai positif, misalnya dokumen *d* merupakan kategori A dan dikenali sebagai kategori A.
- ***False positive*** : Contoh negatif dikenali sebagai positif, misalnya dokumen *d* bukan kategori A akan tetapi dikenali sebagai kategori A.
- ***True negative*** : Contoh negatif dikenali dengan benar sebagai negatif, misalnya dokumen *d* bukan kategori A dan dikenali sebagai bukan kategori A.
- ***False negative*** : Contoh positif dikenali sebagai negatif, misalnya dokumen *d*merupakan kategori A tetapi dikenali bukan sebagai kategori A.

Setelah menghitung nilai *precision, recall,* dan *f1-measure* dari setiap kategori, langkah selanjutnya adalah menghitung *Macroveraged F1*. *Macroveraged F1* digunakan untuk mengevaluasi keseluruhan kinerja algoritma pada dataset yang diberikan. *Macroaveraged F1* menghitung nilai *F1* tiap kategori dan kemudian menghitung nilai rata – ratanya [GUO-04].Definisi dari *macroaveraged* adalah *precision* , *recall* dan *F1-measure* dievaluasi secara lokal pada tiap kategori dan kemudian dievaluasi secara global dengan cara menghitung rata-rata hasil *precision* , *recall* dan *F1-measure* dari kategori – kategori yang berbeda [SEB-02]. *Macroaveraged precision*, *recall* dan *F1* dapat dirumuskan pada persamaan 2-10 sampai dengan 2-12 :

$$macroaveraged\ precision = \frac{\sum_{i=1}^{m} P(i)}{m} \qquad (2\text{-}10)$$

$$macroaveraged\ recall = \frac{\sum_{i=1}^{m} R(i)}{m} \qquad (2\text{-}11)$$

$$macroaveraged\ F1 = \frac{\sum_{i=1}^{m} F1(i)}{m} \qquad (2\text{-}12)$$

Dimana :

m : Banyaknya kategori/kelas

P(i) : Nilai *precision* untuk kategori ke i

R(i) : Nilai *recall* untuk kategori ke i

F1(i) : Nilai *F1 measure* untuk kategori ke i