Jurnal Ilmiah KURSOR
Menuju Solusi Teknologi Informasi
Vol. 4, No. 2, 2008
ISSN 0216 - 0544

PERANCANGAN DAN PEMBUATAN APLIKASI SMS *SPELL CHECKER* BERBAHASA INDONESIA DENGAN MENGGUNAKAN ALGORITMA *NAIVE BAYES* PADA *MOBILE DEVICE*

Tohari Ahmad*, Nunut P. Jatmiko+, Moh Safii#
Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Gedung Teknik Informatika, Kampus ITS, Jl. Raya ITS, Sukolilo, Surabaya – 60111
email : *tohari@its-sby.edu, +nunutpj@its-sby.edu, #syafii@cs.its.ac.id

Abstract

Typo error which may happen in typing text in a hand phone or other mobile devices, or typing text on a keyboard, can be avoided by using spell checker. It is actually often provided by those devices. There are some algorithms that can used to develop such aplication, for example: naive-bayes algorithm as in this research.

In this research, that algorithm is applied in the SMS spell checker in Indonesian. The analyis is done over time and memory needed to process the correct word.

Keywords: *SMS, Spell checker, J2ME*

1. Pendahuluan

Spell checker merupakan metode yang ditujukan untuk mengkoreksi ejaan kata yang salah sehingga membantu pengguna dalam proses pengetikan kata yang benar. Beberapa aplikasi yang menggunakan fitur *spell checker* diantaranya *word processor*, *email client*, *electronic dictionary* serta *search engine* [1].

Spell checker secara sederhana hanya membandingkan di tingkat kata. Artinya, aplikasi *spell checker* sudah mempunyai *dictionary* semua kata-kata yang benar. Ketika *user* mengetikkan suatu kata, maka akan dicek satu per satu apakah kata tersebut ada di *dictionary* atau tidak. Kata yang ada di *dictionary* dianggap sebagai kata yang benar. Sebaliknya jika kata yang tidak ada di *dictionary* dianggap kata yang salah[1].

Hampir semua telepon genggam sudah mempunyai fasilitas T9 yang memungkinkan pengguna untuk melakukan pengecekan kata yang dituliskan. Fasilitas ini mungkin tidak dapat disamakan dengan *spell checker* karena hanya menampilkan kata berdasarkan huruf awalan yang diketik. Bedanya dengan *spell checker* dalam penelitian ini ialah adanya metode pencarian solusi dari kata-kata yang salah.

2. Algoritma *Levenstein Distance* dan *Naive Bayes*

Salah algoritma yang dapat digunakan sebagai pembanding untuk *spell checker* ialah *levensthein distance*. Penyelesaian algoritma *levenstein distance* diperoleh dengan menemukan cara termudah mengubah suatu *string* dalam bentuk *string* yang lain. Perubahan bentuk ini dilakukan pada setiap langkah berupa proses penyisipan (*insertion*), pergantian (*subtitution*) dan penghapusan (*deletion*), dengan setiap *string* akan dibandingkan kesamaannya dalam bentuk perkalian matrik, nilai 0 akan diberikan bila sama dan 1 akan diberikan dan ditambahkan pada setiap iterasinya bila *string* tersebut berbeda [2].

Algoritma *naive bayes* digunakan dalam proses *spell checker* karena probabilitas orang menulis kata benar lebih tinggi dari pada probabilitas menulis kata salah.

Algoritma *naive bayes* diturunkan dari *bayes theorem*.

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

Keterangan:

$P(h)$: probabilitas *prior* dari hipotesis h

$P(D)$: probabilitas *prior* dari pelatihan data D

$P(h|D)$: probabilitas hipotesis h dari data pelatihan D

$P(D|h)$: probabilitas D dengan syarat h sudah terjadi

Dengan kata lain rumus di atas dapat disebut sebagai:

$$posterior = \frac{prior \times likelihood}{evidence}$$

Pada persamaan di atas, yang dicari adalah maksimum *posterior*, sehingga persamaan tersebut dapat diturunkan menjadi [3]:

$$h_{MAP} = \text{argmax} P(h \mid D)$$

$$h_{MAP} = \text{argmax} \frac{P(D \mid h)P(h)}{P(D)}$$

$$h_{MAP} = \text{argmax} P(D \mid h)P(h)$$

$$\text{argmax}_c P(w \mid c)P(c)$$

- $P(c)$ ialah probabilitas kata c didalam pelatihan data. Isi dari data pelatihan adalah kata-kata dalam bahasa Indonesia beserta probabilitasnya. Misalnya, jika ada P('saya') maka dapat dipastikan di dalam *database* pelatihan data probabilitasnya adalah $0 < p < 1$. Tetapi jika ada P('asdfhajsh') maka dapat dipastikan probabilitasnya 0 (nol) atau mendekati 0.
- $P(w \mid c)$ merupakan probabilitas w diketikkan dalam *text* dimana yang dimaksud ialah c [0, 1].

Karena *naïve bayes* menggunakan probabilitas, dan *spell checker* juga membutuhkan *dictionary* yang isinya kumpulan kata-kata benar, maka perlu dilakukan proses pelatihan data. Untuk melakukan hal tersebut pelatihan dilakukan dengan mengambil kata-kata dari forum diskusi seperti pada kaskus.com [4].

Untuk mengukur keberhasilan proses pelatihan data, kata sambung mempunyai probabilitas terbesar, seperti pada Table 1. Kata 'yang' mempunyai probabilitas terbesar dibandingkan data-data yang lain.

Tabel 1. Sepuluh kata dengan probabilitas terbesar

Kata	Probabilitas
yang	0.0188250560
di	0.0124085580
dan	0.0107357750
dengan	0.0102863700
tidak	0.0101615350
dari	0.0053678870
dapat	0.0045689460
dalam	0.0044690780
ini	0.0042693430
untuk	0.0042443760

3. Metode Pencarian Solusi

Di sini, kata yang diperoleh tidak cukup hanya diperoleh dengan proses membandingkan kata yang diinputkan *user* dengan kata yang ada di*dictionary* saja, tetapi juga perlu di*generate*

kemungkinan-kemungkinan kata yang benar. Dari Peter Norvig [5], 80% – 95 % kesalahan pengetikkan disebabkan oleh posisi satu huruf saja. Misalnya kata 'sataga' , padahal kata yang dimaksud ialah 'astaga'. Hanya dengan merubah posisi huruf 's' dengan posisi huruf 'a' , maka akan terbentuk kata yang benar., walaupun sebenarnya dapat saja karena salah posisi 2 huruf misalnya 'astgas' yang seharusnya 'astaga'. Kasus seperti itu memang ada tetapi mungkin prosentasenya kecil.

Seperti yang telah didiskusikan sebelumnya, langkah awal dari *spell checker* ialah membandingkan dengan kata yang ada di*dictionary*. Tetapi mungkin saja setelah dibandingkan ternyata tidak ditemukan kata tersebut di*dictionary*. Kesimpulan awal bahwa kata tersebut salah. Misalnya kata 'sataga' , di*dictionary* memang tidak ada kata tersebut dalam bahasa Indonesia, karena memang tidak ada artinya, tetapi dapat diperkirakan bahwa kata yang dimaksud ialah kata 'astaga'. Dan kata 'astaga' terdapat di dalam *dictionary*.

Untuk memunculkan kombinasi kata yang mungkin dari kata 'sataga' adalah menggunakan metode *deletion*, *transpositiion*, *alteration* dan *insertion*, seperti pada bagian berikut ini.

3.1. *Deletion*

Metode *deletion* digunakan untuk menghapus satu huruf dari kata yang diketikkan. Misalnya ada kata 'astagas', padahal yang dimaksud ialah 'astaga'. Dengan menghapus huruf 's' maka akan terbentuk kata 'astaga', dan kata 'astaga' tersebut dibandingkan dengan yang ada di *dictionary*. Apabila terdapat n huruf dalam satu kata dari maka akan terbentuk n kata yang baru. Sebagai contoh, kata 'astagas' , jika menggunakan metode ini hasilnya stagas, atagas, asagas, astgas, astaas, astags dan astaga.

3.2. *Transposition*

Kesalahan pengetikkan biasanya terjadi karena salah menempatkan huruf. Dan umumnya kesalahan tersebut terjadi pada tombol *keyboard* yang bersebelahan misalnya huruf 'a' dengan 's'. Kata 'astaga' , kemungkinan terbesar orang salah ketik akan menjadi 'sataga'. Dengan menggunakan metode ini posisi huruf dalam kata akan diganti, sehingga jika ada n huruf dalam satu kata akan terbentuk n-1 kata yang baru. Contohnya kata 'sataga' akan dikombinasi menjadi astaga, staaga, saatga, satgaa, sataag.

3.3. *Alteration*

Kesalahan yang mungkin juga terjadi ialah salah ketik satu huruf. Misalnya kata 'asyaga' padahal yang dimaksud ialah kata 'astaga'. Kesalahan tersebut wajar karena di *keyboard* huruf 't' bersebelahan dengan huruf 'y'. Maka untuk mengatasi hal tersebut perlu digunakan pula metode *alteration*. Jika ada n huruf dalam kata, maka akan terbentuk n*26 kata baru. Karena ada penghapusan satu huruf kemudian disisipkan karakter dari 'a' sampai 'z'. Contohnya seperti kata di atas 'asyaga', hasil metode *alteration* menjadi:

*asyaga , bsyaga , csyaga , dsyaga , esyaga , fsyaga , gsyaga , hsyaga , isyaga , jsyaga , ksyaga , lsyaga , msyaga , nsyaga , osyaga , psyaga , qsyaga , rsyaga , ssyaga , tsyaga , usyaga , vsyaga , wsyaga , xsyaga , ysyaga , zsyaga , aayaga , abyaga , acyaga , adyaga , aeyaga , afyaga , agyaga , ahyaga , aiyaga , ajyaga , akyaga , alyaga , amyaga , anyaga , aoyaga , apyaga , aqyaga , aryaga , asyaga , atyaga , auyaga , avyaga , awyaga , axyaga , ayyaga , azyaga , asaaga , asbaga , ascaga , asdaga , aseaga , asfaga , asgaga , ashaga , asiaga , asjaga , askaga , aslaga , asmaga , asnaga , asoaga , aspaga , asqaga , asraga , assaga , **astaga** , asuaga , asvaga , aswaga , asxaga , asyaga , aszaga , asyaga , asybga , asycga , asydga , asyega , asyfga , asygga , asyhga , asyiga , asyjga , asykga , asylga , asymga , asynga , asyoga , asypga , asyqga , asyrga , asysga , asytga , asyuga , asyvga , asywga , asyxga , asyyga , asyzga , asyaaa , asyaba , asyaca , asyada , asyaea , asyafa , asyaga , asyaha , asyaia , asyaja , asyaka , asyala , asyama , asyana , asyaoa , asyapa , asyaqa , asyara , asyasa , asyata , asyaua , asyava , asyawa , asyaxa , asyaya , asyaza , asyaga , asyagb , asyagc , asyagd , asyage , asyagf , asyagg , asyagh , asyagi , asyagj , asyagk , asyagl , asyagm , asyagn , asyago , asyagp , asyagq , asyagr , asyags , asyagt , asyagu , asyagv , asyagw , asyagx , asyagy , asyagz.*

Hasil setelah dilakukan *alteration* adalah 6*26 = 156 kata baru seperti di atas.

3.4. *Insertion*

Selain ketiga metode diatas, bentuk kesalahan pengetikkan mungkin juga terjadi karena kurang mengetikkan satu huruf saja. Misalnya kata 'cotoh' padahal yang dimaksud ingin mengetikkan kata 'contoh'. Untuk mengatasi kesalahan pengetikkan karena kurang satu huruf maka digunakan metode *insertion*. Metode ini akan menambahkan karakter dari 'a' sampai 'z' tanpa menghapus kata yang sudah ada. Jumlah kata setelah dilakukan proses *insertion*, jika ada n huruf dalam satu kata, akan menjadi (n +1) * 26. Merujuk pada kata 'cotoh' diatas, akan terbentuk kata baru sebanyak (5+1)* 26 = 156.

Output dari metode ini adalah:

*acotoh , bcotoh , ccotoh , dcotoh , ecotoh , fcotoh , gcotoh , hcotoh , icotoh , jcotoh , kcotoh , lcotoh , mcotoh , ncotoh , ocotoh , pcotoh , qcotoh , rcotoh , scotoh , tcotoh , ucotoh , vcotoh , wcotoh , xcotoh , ycotoh , zcotoh , caotoh , cbotoh , ccotoh , cdotoh , ceotoh , cfotoh , cgotoh , chotoh , ciotoh , cjotoh , ckotoh , clotoh , cmotoh , cnotoh , cootoh , cpotoh , cqotoh , crotoh , csotoh , ctotoh , cuotoh , cvotoh , cwotoh , cxotoh , cyotoh , czotoh , coatoh , cobtoh , coctoh , codtoh , coetoh , coftoh , cogtoh , cohtoh , coitoh , cojtoh , coktoh , coltoh , comtoh , **contoh** , cootoh , coptoh , coqtoh , cortoh , costoh , cottoh , coutoh , covtoh , cowtoh , coxtoh , coytoh , coztoh , cotaoh , cotboh , cotcoh , cotdoh , coteoh , cotfoh , cotgoh , cothoh , cotioh , cotjoh , cotkoh , cotloh , cotmoh , cotnoh , cotooh , cotpoh , cotqoh , cotroh , cotsoh , cottoh , cotuoh , cotvoh , cotwoh , cotxoh , cotyoh , cotzoh , cotoah , cotobh , cotoch , cotodh , cotoeh , cotofh , cotogh , cotohh , cotoih , cotojh , cotokh , cotolh , cotomh , cotonh , cotooh , cotoph , cotoqh , cotorh , cotosh , cototh , cotouh , cotovh , cotowh , cotoxh , cotoyh , cotozh , cotoha , cotohb , cotohc , cotohd , cotohe , cotohf , cotohg , cotohh , cotohi , cotohj , cotohk , cotohl , cotohm , cotohn , cotoho , cotohp , cotohq , cotohr , cotohs , cotoht , cotohu , cotohv , cotohw , cotohx , cotohy , cotohz.*

Dari keempat metode di atas, dapat dihitung jumlah kata yang dihasilkan dari kombinasi keempat metode di atas.

Tabel 2. Metode dan Jumlah Kata Baru

Metode	Jumlah kata
Deletion	n
Transposition	n-1
Alteration	n*26
Insertion	(n+1)*26

Untuk mengetahui jumlah totalnya , maka nilai pada Tabel 2 harus dijumlahkan, sehingga:

Total = n+ (n-1) + n*26 + ((n+1)*26)

Total = (2n-1) + 26n + 26n + 26

Total = 54n + 25

Artinya jika ada kata 'sataga' , kata-kata hasil *generate* keempat metode di atas menjadi

= 54 * (jumlah huruf 'sataga') + 25

= (54 *6)+25

= 349 kata baru

4. Uji Coba *Spell Checker* dengan Mengirimkan SMS pada *Mobile Phone*

Langkah-langkah yang dilakukan pada saat uji adalah sebagai berikut.

4.1. *Load* data

Perangkat lunak SMS *Spell Checker* ketika di-*deploy* di telepon genggam data pelatihan disimpan dalam file teks. Kemudian ketika perangkat lunak dijalankan maka data akan ditampung dalam *array*. Data yang berada di *array* ini disebut sebagai *dictionary* (kamus).

4.2. Mengetik No Tujuan

Nomor penerima tidak perlu diketik ulang, dengan menggunakan tipe PHONENUMBER untuk *textfield*, nomor telepon akan secara otomatis ditambahkan sebagai menu baru untuk membuka kontak.

4.3. Mengetik Pesan Singkat

Pesan singkat yang diketikkan dapat dieja kebenaran tiap katanya. Jika kata tersebut tidak sesuai dengan kata yang ada di kamus maka dilakukan operasi *insertion*, *transposition*, *deletion*, *alteration*. Dengan operasi tersebut maka akan tercipta himpunan solusi. Kemudian masing-masing kata dalam himpunan solusi akan dicek ulang untuk mengetahui berapa nilai probabilitasnya. Probabilitas kata-kata tersebut kemudian dihitung dengan menggunakan algoritma *naive bayes* dan akan didapatkan solusi. Semua kata yang dihitung akan diurutkan berdasarkan posterior terbesar dan akan ditampilkan dalam bentuk pilihan sehingga pengguna dapat memilih kata yang dimaksud.

4.4. Mengirim Pesan ke Nomor Tujuan

Proses pengiriman pesan ke nomor tujuan menggunakan *Wireless Messaging* API [6,7].

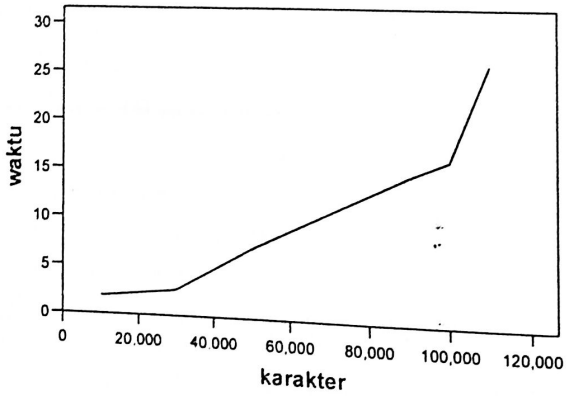
5. Hasil Uji Coba

Uji coba dan evaluasi dilakukan pada tiap proses. Tabel 3 berikut ini merupakan perbandingan jumlah karakter dengan durasi waktu yang dibutuhkan ketika data dari file teks dipindahkan ke *array*.

Tabel 3. Waktu dan Memori yang diperlukan sesuai Jumlah karakter

Jumlah karakter	Durasi *load* (detik)	Memori yang digunakan (byte)
10000	1.9	124172
30000	2.67	367160
50000	7.1	615844
90000	14.81	1096352
100000	16.44	1215732
110000	26.06	1346092

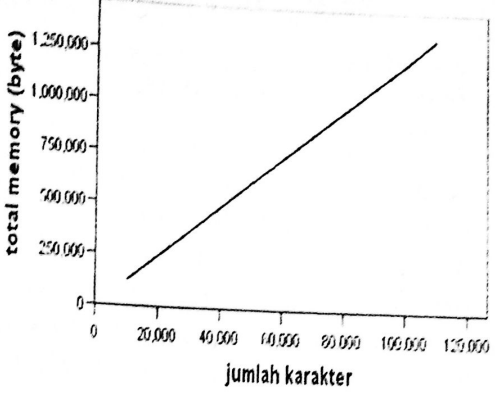
Dari data pada Tabel 3 tersebut, dapat dibuat grafik seperti pada Gambar 1. merupakan uji coba dilakukan dengan membandingkan jumlah karakter dengan durasi *load* dan membandingkan jumlah karakter dengan memori yang digunakan.



Gambar 1 Perbandingan Waktu terhadap Jumlah Karakter

Dari Tabel 3 dan Gambar 1 dapat diambil kesimpulan, bahwa proses *loading* data akan lebih cepat jika menggunakan data pelatihan dengan jumlah karakter di bawah 50000.

Sedangkan perbandingan antara kapasitas memori yang diperlukan dengan jumlah karakter, dapat dinyatakan seperti pada Gambar 2.



Gambar 2. Perbandingan Total Memori dan Jumlah Karakter

Karena bentuk grafik pada Gambar 2 adalah linear, maka persamaan fungsi dapat ditentukan. Kalau diketahui fungsinya maka dapat diketahui jika ada 'n' jumlah karakter maka berapa memori yang dibutuhkan seperti pada Tabel 4 berikut ini.

Tabel 4. Perhitungan fungsi linear total memori dan jumlah karakter

Coefficients [a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	3203.824	3206.662		.999	.374
	jumlah_karakter	12.170	.043	1.000	284.421	.000

a. Dependent Variable: total_memory

Berdasar pada Tabel 4, jika ada 'n' jumlah_karakter maka total_memory yang dibutuhkan 203.824 + (jumlah_karakter * 12.170).

Untuk menentukan jumlah memori yang diperlukan berdasarkan jumlah karakter yang ada, dapat ditentukan dengan menggunakan persamaan yang dihasilkan, seperti dicontohkan pada Tabel 5.

Dari Tabel 6, rata-rata untuk jumlah kata yang dibangkitkan adalah berbanding lurus dengan durasi waktu yang dibutuhkan untuk sampai menemukan solusi.

Tabel 5. Contoh perhitungan memori dengan menggunakan fungsi 203.824 + (jumlah_karakter * 12.170)

Jumlah Karakter	Total memory (203.824 + (jumlah karakter * 12.170))
120000	1463604
130000	1585304
140000	1707004
150000	1828704

Tabel 6. Contoh durasi dan pencarian solusi

Kalimat salah	Kalimat benar	Durasi (detik)	Jumlah kata
sya besk berangkt ke malang	**saya besok berangkat** ke malang	9.565	885 (10 solusi)
mkan dengn bpak soe nant	**makan dengan bapak sore nanti**	13.223	1205 (11 solusi)
pengrus rmah tngga saya	**pengurus rumah tangga** saya	10.155	939 (7 solusi)
belm mkan sehrian karna sbuk	**belum makan seharian karena sibuk**	15.869	1421 (17 solusi)
sia maan soe sebntar lgi	Siap makan sore sebentar lagi	13.329	1205 (27 solusi)

Tabel 7. Hasil pengujian dengan *naive bayes*

Kata benar	Kata salah	Menurut *naive bayes*
benar	bnar	benar
salah	slah	salah
payah	apyah	ayah
astaga	sataga	astaga
tunjuk	tunjku	tungku
uang	aung	uang
sendiri	sendii	sendiri
malang	lalang	lelang
manis	mans	mana
bakat	abkat	bakat
kamu	kmu	kamu
sekolah	seoklah	sekolah
bercanda	bercada	berada
kurang	kuragn	kurang
sedia	sdia	dia
percaya	percaya	percaya
laku	laky	laku
sampah	samph	sampah
umpan	mupan	mulan

Contoh penulisan kata yang salah yang dapat dikoreksi oleh sistem ini adalah dinyatakan dalam Tabel 7. Kata yang benar di kolom sebelah kiri, kata yang salah di bagian tengah, dan di bagian yang paling kanan adalah hasil dari *spell checker* menggunakan *naive bayes*. Dari total pengujian yang dilakukan, tingkat keakuratan mencapai 63.14%.

Tabel 8. Solusi berprobabilitas kata 'sdia'

Solusi berprobabilitas	Posterior
dia	3.3151082E-6
sedia	3.107884E-7

Tabel 9. Hasil pengujian dengan *levensthein*

Kata benar	Kata salah	Menurut *levenshtein*
benar	bnar	benar
salah	slah	salah
payah	apyah	ayah
astaga	sataga	setara
tunjuk	tunjku	tungku
uang	aung	gaung
sendiri	sendii	sendiri
malang	lalang	lelang
manis	mans	mana
bakat	abkat	akan
kamu	kmu	kamu
sekolah	seoklah	setelah
bercanda	bercada	berada
kurang	kuragn	kurang
sedia	sdia	dia
percaya	percaya	percaya
laku	laky	laku
sampah	samph	sampe
umpan	mupan	mapan

Hasil yang diperoleh adalah 63.14% dimungkinkan karena probabilitas kata yang dikehendaki masih belum menjadi maksimum posterior. Ini artinya, kata yang dikehendaki benar, misalnya kata sdia mempunyai himpunan solusi berprobabilitas seperti pada Tabel 9:

Kata yang dikehendaki muncul dari kata sdia ialah kata sedia. Tetapi, karena probabilitasnya, kata dia (lebih besar dari probabilitas kata sedia) yang menjadi solusi. Karena *naive bayes* berdasar pada probabilitas dan untuk mendapatkan probabilitas yang relatif tinggi harus melakukan pelatihan data terlebih dahulu, maka kata sambung, kata ganti akan mempunyai probabilitas yang relatif tinggi.

Untuk pengujian akurasi *spell checker* dengan *levenshtein distance*, penulis membuat dengan menggunakan bahasa pemrograman PHP. Dengan menggunakan data pengujian yang sama dengan algortima *naive-bayes*, hasil pengujian menggunakan metode *levensthein* dapat dituliskan seperti pada Tabel 9.

Nilai akurasinya ialah 36.84 %. Sehingga dapat disimpulkan bahwa penggunaan *naive bayes* masih lebih tinggi dengan 63.14 %, di atas akurasi *levenshtein* 36.84%.

6. Simpulan dan Saran

Dari aplikasi yang telah dibuat beserta uji coba yang telah dilakukan terhadapnya, dapat diambil kesimpulan sebagai berikut:

- Algoritma *naive bayes* dapat diterapkan pada proses *spell checker*. Ini dikarenakan kata yang benar mempunyai probabilitas yang tinggi daripada kata yang salah.
- Probabilitas kata 'yang' sebesar 0.0188250560, artinya kata sambung mempunyai probabilitas terbesar sama seperti kata 'di', 'dan' serta 'dengan'.
- Semakin lengkap kosakata semakin akurat dan banyak pilihan kata-kata yang benar tetapi berbanding terbalik dengan kecepatan *load* data.
- Jumlah kata yang dipindahkan ke *array* berbanding lurus dengan jumlah penggunaan memori.
- Semakin banyak kata yang salah maka semakin banyak pula kata baru yang dibangkitkan. Ini akan membuat proses pencarian solusi lebih lama.
- Karena grafik jumlah karakter dan jumlah total memori yang dibutuhkan linear maka dapat dihitung fungsinya, total_memory = 203.824 + (jumlah_karakter * 12.170). sehingga jika ada *n* jumlah_karakter maka dengan rumus tersebut dapat diketahui berapa total memory yang dibutuhkan untuk proses pemindahan data dari file teks ke *array*.
- Untuk aplikasi ini, penerapan algoritma *naive bayes* lebih sesuai daripada *levenshtein*.

Untuk pengembangan lebih lanjut, hal-hal yang dapat dilakukan antara lain adalah:

- Selain ejaan, dapat pula memperbaiki urutan SPO (Subyek Predikat Obyek).
- Perlu dibuat metode baru bagaimana proses *load* data lebih cepat walau dengan menggunakan ratusan ribu karakter.
- Pengembangan selanjutnya, probabilitas kata dapat di *update* dengan kosa kata yang baru.

Daftar Pustaka

[1] Wikipedia, *Spell Checker*, URL:http://en.wikipedia.org/wiki/Spell_checker, 2008.

[2] Ranjaliba, Lihardo, Mubarok, Mohammad, Syahrul, *Analisa Algoritma Levenshtein Distance untuk Penerapan Kasus Spell Checking*. Jurusan Teknik Informatika Sekolah Tinggi Telkom Bandung, 2005.

[3] Wikipedia, *Bayes theorem*, URL:http://en.wikipedia.org/wiki/Bayes_theorem, 2008

[4] http://www.kaskus.us/index.php

[5] Norvig, Peter, *How to write a spelling corrector*, URL:http://www.norvig.com/spell-correct.html, 2008

[6] Eric Giguere, *The J2ME Wireless Toolkit WMA Console* URL: http://developers.sun.com/mobility/reference/techart/index.jsp, diakses tanggal 3 Juli 2008.

[7] Hartanto, Aditya Antonius, *Java 2 Micro Edition Mobile Interface Device Programming*, Elexmedia Komputindo, Jakarta, 2003.