Jurnal "LOG!K@" , Jilid 6, No. 2, 2016, Hal. 131 - 143
ISSN 1978 – 8568

ANALISIS KINERJA ALGORITMA *LEVENSHTEIN DISTANCE* DALAM MENDETEKSI KEMIRIPAN DOKUMEN TEKS

B. P. Pratama[1)] dan S. A. Pamungkas[2)]
[1)] Program Studi Matematika, Fakultas Sains dan Teknologi,
UIN Syarif Hidayatullah Jakarta
Email: bp.pratama16@gmail.com
[2)] Badan Pengkajian dan Penerapan Teknologi (BPPT) Jakarta

Abstract: Recently, plagiarism in text documents is the one of academic problems. Generally, acts of plagiarism is done by changing the structure of words in a sentence and the sentences structure in a paragraph, as well as insertion, deletion, or subtitution of the word.In this study, we constructed a system that able to detect the level of similarity in the text documents using Levenshtein distance algorithm by adding case folding, tokenizing, stopword removal, stemming, and sorting. The process of matching strings in this algorithm produce a distance value that determines the percentage of similarity scores. Analysis of using the stopword removal, stemming, and sorting is done to see the effect of the algorithm Levenshtein distance performance. Two sets of data and the real data are used in the simulation. The first data set is done by changing the structure of words in a sentence, the second data set is done by changing the sentence structure in a paragraph, and the real data consist of the abstract of research journals. The simulation shows that the word sorting process gives a big effect in Levenshtein distance algorithm. The best result in the first data set is shown by process of using the stopword removal, stemming, and sorting as simultaneously. The best result in the second data set is shown by process of using the stopword and stemming combined with sorting. The best result in the real data is shown in stemming-sorting process.
Keywords: plagiarism, Levenshtein distance, stopword removal, Stemming, sorting.

Abstrak: Akhir-akhir ini, plagiarisme pada dokumen teks merupakan salah satu permasalahan akademik yang semakin meningkat. Secara umum, tindak plagiarisme dilakukan dengan mengubah struktur kata dalam kalimat dan struktur kalimat dalam paragraf, serta melakukan penyisipan, penghapusan, atau penggantian kata. Pada penelitian ini, dibangun sebuah sistem yang mampu mendeteksi tingkat kemiripan antar dokumen teks menggunakan algoritma *Levenshtein distance* dengan menambahkan proses *case folding*, *tokenizing*, *stopword removal*, *stemming*, dan *sorting*. Proses pencocokan *string* pada algoritma ini dapat menghasilkan nilai *distance* yang menjadi penentu persentase bobot *similarity*. Analisa penggunaan *stopword removal*, *stemming*, dan *sorting* dilakukan untuk melihat pengaruhnya terhadap kinerja algoritma *Levenshtein distance*. Simulasi algoritma ini dilakukan terhadap dua data set dan satu data real. Pada data set 1 dilakukan pengubahan struktur kata dalam kalimat, pada data set 2 pengubahan struktur kalimat dalam paragraf, dan pada data real tidak dilakukan pengubahan apapun karena data real merupakan dokumen abstrak dari jurnal penelitian. Hasil simulasi menunjukkan bahwa penggunaan *sorting* sangat berpengaruh bagi algoritma *Levenshtein distance*. Hasil terbaik pada data set 1 ditunjukan pada proses yang menggunakan *stopword removal*, *stemming*, dan *sorting* sekaligus. Hasil terbaik pada data set 2 diperlihatkan pada proses yang

menggunakan *stopword* dan *stemming* yang digabungkan dengan *sorting*. Hasil terbaik pada data real diperlihatkan pada proses *stemming-sorting*.
Kata kunci: plagiarisme, *Levenshtein distance*, *Stopword removal*, *Stemming*, *sorting*.

PENDAHULUAN

Teknologi dalam bidang informasi telah banyak dimanfaatkan kelebihannya untuk mempermudah suatu pekerjaan menjadi lebih efektif dan lebih cepat. Namun, hal ini tidak serta-merta memberikan dampak positif. Salah satu wujud nyata dampak negatif dari perkembangan teknologi adalah terjadinya tindakan mengambil sebagian atau seluruh ide seseorang berupa teks dokumen tanpa mencantumkan sumber pengambilan, yang sering disebut dengan plagiarisme. Plagiarisme adalah tindakan penyalahguanaan, pencurian/perampasan, penerbitan, pernyataan, atau menyatakan sebagai milik sendiri sebuah pikiran, ide, tulisan atau ciptaan yang sebenarnya milik orang lain.

Tindak plagiarisme secara perlahan dapat ditekan dan untuk mendukung penekanan tindak plagiarisme ini, maka dibutuhkan adanya sistem yang memudahkan dalam mendeteksi dan mengukur kemiripan dokumen. Selain dapat mengetahui tindak plagiarisme, pengukuran kemiripan dokumen ini dapat membantu dalam pengelompokan dokumen. Sebagian besar kasus plagiarisme ditemukan dibidang akademisi, berupa essai, jurnal, laporan penelitian, dan sebagainya. Deteksi plagiarisme dilakukan dengan membandingkan sebuah dokumen dengan dokumen lainnya. Tingkat kesamaan dokumen tersebut akan menjadi dasar pendeteksian plagiarisme.

Terdapat tiga metode dalam pendeteksian plagiarisme, yaitu metode pencarian kata kunci, perbandingan teks lengkap dan dokumen fingerprinting. Pada metode pencarian kata kunci, metode ini mencari kesamaan kata-kata yang sering muncul dalam suatu dokumen dan kemudian akan dibandingkan dengan kata-kata yang sering muncul pada dokumen lain. Sedangkan pada metode perbandingan teks lengkap adalah dengan membandingkan seluruh isi teks. Pada metode dokumen fingerprinting yaitu dengan mengubah struktur kalimat menjadi sebuah angka-angka yang kemudian dibandingkan nilainya dengan dokumen lain yang sudah diubah juga ke dalam bentuk angka-angka.

Beberapa penelitian telah dilakukan untuk mendeteksi plagiarisme. Pandawa [6] menggunakan algoritma Winowwing untuk mendeteksi kemiripan isi dokumen teks. Algoritma tersebut menggunakan metode fingerprinting. Nafik [5] menggunakan algoritma Levenshtein distance untuk mencari nilai edit distance untuk penilaian jawaban esai. Pada penelitian tersebut dilakukan uji coba untuk melihat kemampuan membandingkan kemiripan jawaban esai dengan kunci jawaban, kemudian dilihat pengaruh stemming terhadap perubahan presentase yang dihasilkan dan hasilnya dibandingkan dengan hasil penilaian secara manual.

Pada penelitian ini, penulis merancang sebuah aplikasi deteksi kemiripan dokumen berbasis web yang dibuat untuk membandingkan satu dokumen dengan dokumen lain. Perbedaan penelitian ini dengan penelitian sebelumnya adalah uji coba menggunakan algoritma Leveshtein distance preprocessing untuk mendeteksi kemiripan dokumen teks menggunakan metode pencarian kata kunci dengan menambahkan proses *converting*, *tokenizing*, *stopword removal*, *stemming* dan *sorting*. Kemudian hasilnya dibandingkan dengan algoritma Levenshtein distance standard, yaitu tanpa proses stopword removal,

stemming dan sorting, penggunaan database untuk menyimpan kata berimbuhan atau term beserta kata dasarnya pada tabel stem, dan menyimpan kata-kata yang sering muncul pada tabel stop.

TINJAUAN PUSTAKA

String metric adalah matriks berbasis karakter atau tekstual yang dapat menghasilkan nilai kesamaan atau ketidaksamaan dari dua buah teks *string* untuk proses perbandingan dan penyamaan. *String metric* biasanya digunakan dalam deteksi kecurangan, analisa *fingerprint*, deteksi plagiarisme, analisis DNA dan RNA, data *mining*, dan lain-lain. Beberapa algoritma yang berdasarkan kepada *string metric* adalah Smith-Waterman, Levenshtein *Distance*, TF/IDF, dan lain-lain. *String matching* merupakan metode yang digunakan *string metric* dalam menemukan hasil kesamaan atau ketidaksamaan dari teks yang diberikan [3].

Salah satu langkah penting dalam proses deteksi kemiripan adalah *preprocessing*, yaitu pemrosesan isi dokumen sebelum dibandingkan. Tujuannya adalah untuk menyeragamkan kata dan menghilangkan *noise* yang terdiri dari imbuhan, angka, simbol dan karakter yang tidak relevan, serta kata-kata yang tidak penting [1].

Tahap-tahap *preprocessing* adalah *Case Folding* atau *Converting*, *Tokenizing*, *Stopword Removal*, *Stemming*, dan *Sorting*. *Case folding* atau *converting* adalah proses mengubah semua huruf dalam dokumen menjadi huruf kecil, agar sistem mampu menyamakan tiap karakter dari dokumen satu dengan dokumen lainnya. *Tokenizing* adalah tahap pengeliminasian simbol dan karakter yang tidak relevan dihilangkan, seperti tanda baca, angka, dan lain-lain. *Stopword Removal* adalah tahap pengeliminasian kata-kata yang tidak penting dari hasil *tokenizing*, seperti ‘yang’, ‘di’, ‘ke’, ‘pada’, dan lain-lain. Penghapusan kata-kata ini dimaksudkan untuk lebih mempermudah pembandingan dokumen. *Stemming* adalah proses untuk mencari *root* kata (kata dasar) dari kata berimbuhan hasil proses *stopword removal*. Pada tahap ini dilakukan proses pengembalian berbagai bentukan kata ke dalam suatu representasi yang sama. Tahap ini kebanyakan dipakai untuk teks berbahasa Inggris dan lebih sulit diterapkan pada teks berbahasa Indonesia. *Sorting* adalah proses mengurutkan data baik secara menaik (*ascending*) atau secara menurun (*descending*). Data yang diurut bisa bertipe numerik maupun karakter [7].

Algoritma Levenshtein *Distance*

Levenshtein *distance* adalah sebuah matriks *string* yang digunakan untuk mengukur perbedaan atau jarak (*distance*) antara dua *string*. Nilai *distance* antara dua *string* ini ditentukan oleh jumlah minimum dari operasi-operasi perubahan yang diperlukan untuk melakukan transformasi dari suatu *string* menjadi *string* lainnya. Operasi-operasi tersebut adalah penyisipan (*insertion*), penghapusan (*deletion*), atau penukaran (*subtitution*). Levenshtein *distance* merupakan salah satu algoritma yang dapat digunakan dalam mendeteksi kemiripan antara dua *string* yang berpotensi melakukan tindak plagiarisme [9].

Operasi-Operasi pada Levenshtein *Distance*

Pada algoritma Levenshtein *distance*, terdapat tiga macam operasi yang dapat dilakukan yaitu [4]:

1. Operasi Penyisipan Karakter (*Insertion*)

   Operasi penyisipan karakter berarti menyisipkan karakter ke dalam suatu *string*. Contohnya *string* ‘disrit’ menjadi *string* ‘diskrit’, dilakukan penyisipan karakter ‘k’ di

akhir *string*. Penyisipan karakter tidak hanya dilakukan di tengah *string*, namun bisa disisipkan diawal maupun disisipkan diakhir *string*. Ilustrasi:

String 1	d	i	s	k	r	i	t
String 2	d	i	s	-	r	i	t
insertion				k			

2. Operasi Penghapusan Karakter (*Deletion*)

   Operasi penghapusan karakter dilakukan untuk menghilangkan karakter dari suatu *string*. Contohnya *string* 'matematikan' karakter terakhir dihilangkan sehingga menjadi *string* 'matematika'. Pada operasi ini dilakukan penghapusan karakter 'n'. Ilustrasi:

String 1	m	A	t	E	m	a	t	i	k	a	-
String 2	m	A	t	E	m	a	t	i	k	a	n
Deletion											n

3. Operasi Penukaran Karakter (*Subtitution*)

   Operasi penukaran karakter merupakan operasi menukar sebuah karakter dengan karakter lain. Contohnya penulis menuliskan *string* 'gimpunan' menjadi 'himpunan'. Dalam kasus ini karakter 'g' yang terdapat pada awal *string*, diganti dengan huruf 'h'. Ilustrasi:

String 1	H	i	M	p	u	n	a	n
String 2	G	i	M	p	u	n	a	n
subtitution	H							

Langkah-Langkah Algoritma Levenshtein *Distance*

Algoritma Levenshtein *distance* berjalan mulai dari pojok kiri atas sebuah *array* dua dimensi (matriks) yang telah diisi sejumlah karakter *string* awal dan *string* target. Entri-entri pada matriks tersebut merepresentasikan nilai terkecil dari transformasi *string* awal menjadi *string* target. Entri yang terdapat pada ujung kanan bawah matriks adalah nilai *distance* yang menggambarkan jumlah perbedaan dua *string* [4]. Berikut ini adalah langkah-langkah algoritma Levenshtein *distance* dalam mendapatkan nilai *distance* [2]:
Misalkan S = *String* Awal, dan T = *String* Target

Langkah 1: Inisialisasi
a) Hitung panjang S dan T, misalkan m dan n
b) Buat matriks berukuran 0...m baris dan 0...n kolom
c) Inisialisasi baris pertama dengan 0...n
d) Inisialisasi kolom pertama dengan 0...m

Langkah 2: Proses
a) Periksa S[i] untuk $1 < i < n$
b) Periksa T[j] untuk $1 < j < m$
c) Jika S[i] = T[j], maka entrinya adalah nilai yang terletak pada tepat didiagonal atas sebelah kiri, yaitu d[i,j] = d[i-1,j-1]
d) Jika S[i] ≠ T[j], maka entrinya adalah d[i,j] minimum dari:
   - Nilai yang terletak tepat diatasnya, ditambah satu, yaitu d[i,j-1]+1
   - Nilai yang terletak tepat dikirinya, ditambah satu, yaitu d[i-1,j]+1
   - terletak pada tepat didiagonal atas sebelah kirinya, ditambah satu, yaitu d[i-1,j-1]+1

Langkah 3: Hasil entri matriks pada baris ke-i dan kolom ke j, yaitu d[i,j]
Langkah 2 diulang hingga entri d[m,n] ditemukan.

Algoritma Levenshtein distance yang dibantu preprocessing dapat diimplementasikan dalam bahasa pemrograman dengan pseudocode yang ditunjukan pada Gambar 1.

```
int LevenshteinDistance (char s[1…m], char t[1…n])
    //d is a table with m+1 rows and n+1 columns
    declare int d[0…m, 0…n]
        for i from 0 to m
            d[i, 0] := i
        for j from 0 to n
            d[0, j] := j
        for i from 1 to m
             for j from 1 to n
                  if s[i] = t[j] then d[i,j] := d[i-1,j-1]
                      else
                  d[i, j] := minimum(
                      d[i-1, j] +1, //deletion
                      d[i, j-1] +1, //insertion
                      d[i-1, j-1] +1) //substitution
        return d[m, n]
```

Gambar 1 *Pseudocode* Algoritma Levenshtein *Distance* []

Bobot *Similarity*

Levenshtein *distance* melakukan perhitungan bobot *similarity* setelah mendapatkan nilai *distance* dari dua dokumen yang dibandingkan. Kemudian menggunakan suatu persamaan dalam menentukan bobot *similarity*, yaitu [8]:

$$Bobot\ Similarity = \left(1 - \frac{d[m,n]}{Max(S,T)}\right) * 100\%$$

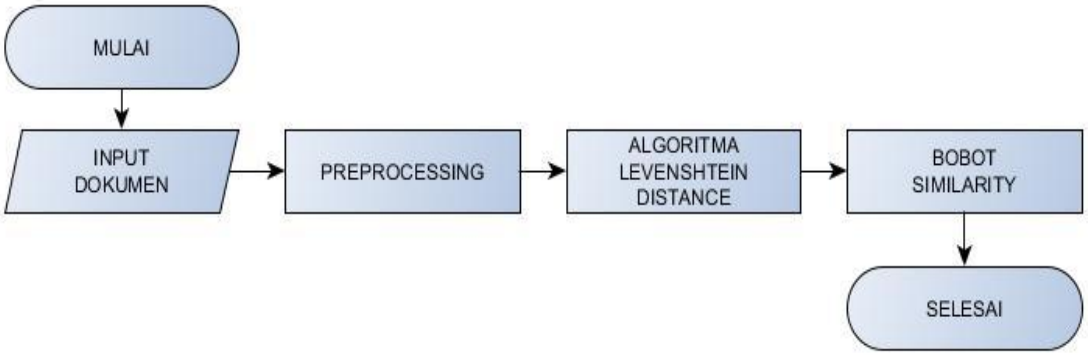
dengan d[m,n] adalah nilai *distance*, terletak pada baris ke m dan kolom ke n, S adalah panjang *string* awal, T adalah panjang *string* target, dan Max(S,T) adalah panjang *string* terbesar antara *string* awal dan *string* target.

Bobot *similarity* diasumsikan pada rentang 0 (nol) hingga 100 (seratus) persen, yang artinya nilai 100 adalah nilai maksimum yang menunjukan bahwa dua kata adalah sama identik. Pendekatan ini mampu digunakan untuk mengukur bobot *similarity* antar dua *string* berdasarkan susunan karakter.

METODOLOGI PENELITIAN

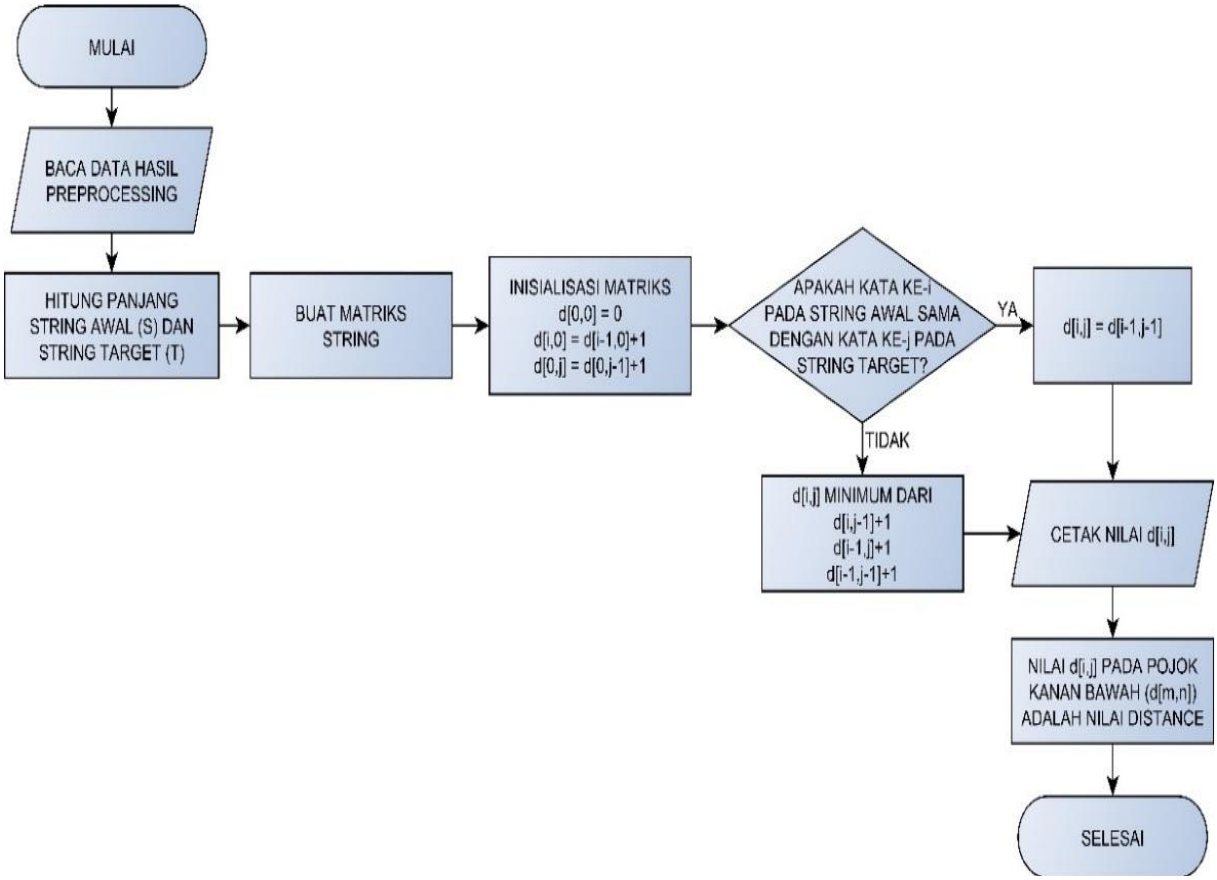
Data Penelitian

Data yang digunakan dalam uji coba deteksi kemiripan dokumen ini adalah dokumen teks berbahasa indonesia tanpa gambar maupun tabel, data *stopword*, dan data *stemming*. Data *stopword* diambil dari http://hikaruyuuki.lecture.ub.ac.id. Data *stemming* dibuat secara manual. Setelah data terkumpul, dilakukan pengolahan data untuk mendapatkan bobot kemiripan. Gambar 2 memperlihatkan *flowchart* dalam mendapatkan bobot kemiripan (*similarity*).



Gambar 2. *Flowchart* Pencarian Bobot *Similarity*

Data hasil *preprocessing* diolah untuk mendapatkan nilai *distance* yang kemudian digunakan untuk menghitung bobot *similarity* antar dokumen. Gambar 3 memperlihatkan tahap-tahap dalam proses algoritma, di tunjukan pada *flowchart* berikut:



Gambar 3. *Flowchart* Levenshtein *Distance*

HASIL DAN PEMBAHASAN

Kinerja Algoritma Levenshtein *distance*

Untuk mengetahui bagaimana kinerja algoritma Levenshtein dalam mendeteksi kemiripan dokumen teks, maka dilakukan percobaan menggunakan dua buah data *training* yang didapatkan dari internet (tabel 1).

Tabel 1. *Input* Dokumen

Dokumen 1	Information Retrieval: PreProcessing dengan PHP Preprocessing merupakan tahapan sangat penting dalam fase indexing pada suatu sistem temu-balik informasi (Information Retrieval). Kode program berikut memperlihatkan langkah-langkah sederhana dalam preprocessing terutama stop word removal dan stemming. Program ditulis dalam PHP sehingga mudah dipahami dan langsung dapat dijalankan (asal di taruh di web server). Daftar stop word dan stem dimasukkan ke dalam suatu array sehingga tidak memerlukan database. Tulisan lain memperlihatkan preprocessing yang melibatkan database MySQL.
Dokumen 2	Information Retrieval: Preprocessing dengan PHP+MySQL Tulisan sebelumnya memperlihatkan langkah-langkah preprocessing menggunakan PHP dimana daftar stop word dan term stem disimpan di dalam array. Kali ini, sebagaimana tutorial kuliah IR kemarin malam, saya sertakan kode program, masih dengan PHP dimana teks yang akan diproses dan daftar term stem tersebut dimasukkan ke dalam database MySQL bernama dbstbi yang di dalamnya terdapat 3 tabel, yaitu tbberita, tbstem dan tbindex. Sementara, tbindex tidak digunakan, kali ini.

K1/K2		array	daftar	database	database	fase	indexing	informasi	information	information	jalan	kode	langkah	langkah	langsung	libat	masuk	mudah	mysql	paham	php	php	preprocessing	preprocessing	preprocessing	preprocessing	program	program	removal	retrieval	retrieval	sederhana	server	sistem	stem	stemming	stop	stop	tahap	taruh	temu	tulis	tulis	web	word	word
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
array	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
daftar	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
daftar	3	2	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
database	4	3	2	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
dbstbi	5	4	3	2	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
information	6	5	4	3	3	3	3	4	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
ir	7	6	5	4	4	4	4	4	5	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
kemarin	8	7	6	5	5	5	5	5	5	6	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
kode	9	8	7	6	6	6	6	6	6	6	7	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
kuliah	10	9	8	7	7	7	7	7	7	7	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
langkah	11	10	9	8	8	8	8	8	8	8	8	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
langkah	12	11	10	9	9	9	9	9	9	9	9	8	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
malam	13	12	11	10	10	10	10	10	10	10	10	9	8	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
masuk	14	13	12	11	11	11	11	11	11	11	11	10	9	9	9	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
mysql	15	14	13	12	12	12	12	12	12	12	12	11	10	10	10	10	10	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
mysql	16	15	14	13	13	13	13	13	13	13	13	12	11	11	11	11	11	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
nama	17	16	15	14	14	14	14	14	14	14	14	13	12	12	12	12	12	11	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
php	18	17	16	15	15	15	15	15	15	15	15	14	13	13	13	13	13	12	12	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
php	19	18	17	16	16	16	16	16	16	16	16	15	14	14	14	14	14	13	13	12	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
php	20	19	18	17	17	17	17	17	17	17	17	16	15	15	15	15	15	14	14	13	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
preprocessing	21	20	19	18	18	18	18	18	18	18	18	17	16	16	16	16	16	15	15	14	13	12	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
preprocessing	22	21	20	19	19	19	19	19	19	19	19	18	17	17	17	17	17	16	16	15	14	13	12	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
program	23	22	21	20	20	20	20	20	20	20	20	19	18	18	18	18	18	17	17	16	15	14	13	13	13	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
proses	24	23	22	21	21	21	21	21	21	21	21	20	19	19	19	19	19	18	18	17	16	15	14	14	14	14	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
retrieval	25	24	23	22	22	22	22	22	22	22	22	21	20	20	20	20	20	19	19	18	17	16	15	15	15	15	15	15	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
serta	26	25	24	23	23	23	23	23	23	23	23	22	21	21	21	21	21	20	20	19	18	17	16	16	16	16	16	16	16	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
simpan	27	26	25	24	24	24	24	24	24	24	24	23	22	22	22	22	22	21	21	20	19	18	17	17	17	17	17	17	17	17	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
stem	28	27	26	25	25	25	25	25	25	25	25	24	23	23	23	23	23	22	22	21	20	19	18	18	18	18	18	18	18	18	18	19	19	20	21	22	23	24	25	26	27	28	29	30	31	32
stem	29	28	27	26	26	26	26	26	26	26	26	25	24	24	24	24	24	23	23	22	21	20	19	19	19	19	19	19	19	19	19	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
stop	30	29	28	27	27	27	27	27	27	27	27	26	25	25	25	25	25	24	24	23	22	21	20	20	20	20	20	20	20	20	20	20	20	20	20	21	22	23	24	25	26	27	28	29	30	31
tabel	31	30	29	28	28	28	28	28	28	28	28	27	26	26	26	26	26	25	25	24	23	22	21	21	21	21	21	21	21	21	21	21	21	21	21	22	23	24	25	26	27	28	29	30	31	32
tbberita	32	31	30	29	29	29	29	29	29	29	29	28	27	27	27	27	27	26	26	25	24	23	22	22	22	22	22	22	22	22	22	22	22	22	22	22	23	24	25	26	27	28	29	30	31	32
tbindex	33	32	31	30	30	30	30	30	30	30	30	29	28	28	28	28	28	27	27	26	25	24	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	24	25	26	27	28	29	30	31	32
tbindex	34	33	32	31	31	31	31	31	31	31	31	30	29	29	29	29	29	28	28	27	26	25	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	25	26	27	28	29	30	31	32
tbstem	35	34	33	32	32	32	32	32	32	32	32	31	30	30	30	30	30	29	29	28	27	26	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	26	27	28	29	30	31	32	33
teks	36	35	34	33	33	33	33	33	33	33	33	32	31	31	31	31	31	30	30	29	28	27	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	27	28	29	30	31	32	33
term	37	36	35	34	34	34	34	34	34	34	34	33	32	32	32	32	32	31	31	30	29	28	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27	27	28	29	30	31	32
term	38	37	36	35	35	35	35	35	35	35	35	34	33	33	33	33	33	32	32	31	30	29	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	28	29	30	31	32
tulis	39	38	37	36	36	36	36	36	36	36	36	35	34	34	34	34	34	33	33	32	31	30	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	28	28	29	29	29	30	31
tutorial	40	39	38	37	37	37	37	37	37	37	37	36	35	35	35	35	35	34	34	33	32	31	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	29	29	29	30	30	32
word	41	40	39	38	38	38	38	38	38	38	38	37	36	36	36	36	36	35	35	34	33	32	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	30	30	30	29	30

Gambar 4. Pengisian Nilai Levenshtein *Distance* Keseluruhan

Setelah melalui *preprocessing* pada kedua data tersebut, kemudian dilakukan proses pencarian nilai *distance* menggunakan algoritma Levenshtein *distance* dimana dalam perhitungannya menggunakan *string metric* (matriks *string*). Bobot kemiripan diperoleh setelah mendapatkan nilai *distance*. Pengisian nilai Levenshtein *distance* ditampilkan pada Gambar 4. Pada gambar 4, nilai yang ditunjukan pada elemen matriks paling akhir d[m,n] = d[45,41] adalah nilai *distance* yang didapat dari perbandingan dua *string* ini. Nilai *distance* ini menentukan besarnya bobot kemiripan dokumen.

Hasil Uji Coba

Hasil uji coba ditampilkan pada Tabel 2. Bobot *similarity* yang ditampilkan merupakan bobot *simlarity* antar dokumen yang mengalami pembulatan keatas. Dalam sistem sebenarnya ditampilkan hasil *preprocessing*, matriks *string*, kategori kemiripan, waktu proses *preprocessing* dan algoritma, serta memberikan hasil akhir berupa bobot *similarity* yang mengalami pembulatan keatas. Setiap uji coba data menggunakan nilai ambang batas (*threshold*) default 50%.

Tabel 2. Hasil Uji Coba Tingkat Akurasi

		Bobot Similarity							
		SP	SM	SO	SP.SM	SM.SO	SP.SO	SP.SM.SO	NONE
Data Set 1	A1-A2	47	36	73	48	77	83	92	34
	A1-B1	12	7	16	13	19	17	23	7
	A1-B2	12	9	18	13	24	15	22	9
	A2-B1	12	9	17	14	21	14	21	9
	A2-B2	15	10	20	16	24	14	21	9
	B1-B2	47	33	77	48	81	88	91	32
Data Set 2	C1-C2	37	41	100	40	100	100	100	40
	C1-D1	5	8	16	5	15	5	7	8
	C1-D2	0	6	16	0	15	5	7	6
	C2-D1	3	7	16	3	15	5	7	7
	C2-D2	5	7	16	5	15	5	7	7
	D1-D2	65	71	100	65	100	100	100	71
Data Real	E1-E2	6	5	8	6	10	7	8	5
	E1-E3	21	11	27	21	32	22	25	10
	E1-E4	7	8	9	8	11	7	14	7
	E2-E3	2	6	18	4	21	11	19	5
	E2-E4	7	7	21	8	26	20	24	6
	E3-E4	8	9	8	9	25	18	22	9

Pada Tabel 2, hasil uji coba pengaruh *stopword removal*, *stemming*, dan *sorting* diperlihatkan dalam tiga jenis data uji dengan menggunakan algoritma Levenshtein *distance*. Nilai *threshold* digunakan untuk menentukan kategori kemiripan yaitu ‘Mirip’ atau ‘Tidak Mirip’. Jika bobot *similarity* kurang dari nilai *threshold*, maka kategori kemiripannya adalah ‘Tidak Mirip’. Berdasarkan uji coba tingkat akurasi yang dilakukan dengan *threshold* 50%, dokumen yang termasuk dalam kategori ‘Mirip’ ditandai dengan sel yang berwarna biru.

Pada Tabel 3, hasil uji coba waktu proses diperlihatkan dalam tiga jenis data uji. Masing-masing data uji memperlihatkan rata-rata waktu proses untuk *preprocessing* dan algoritma pada setiap dokumen yang dibandingkan. Berdasarkan uji coba, waktu proses sistem dapat dengan cepat mengolah data uji dengan durasi kurang dari 1 detik.

Tabel 3 Hasil Uji Coba Waktu Proses

		Rata-Rata Waktu Proses							
		SP	SM	SO	SP.SM	SM.SO	SP.SO	SP.SM.SO	NONE
Data Set 1	Preprocessing	0,0320	0,0828	0,0017	0,1047	0,1061	0,0290	0,1019	0,0015
	Algoritma	0,0493	0,1437	0,1978	0,0556	0,2668	0,0455	0,0544	0,2214
Data Set 2	Preprocessing	0,0314	0,0726	0,0012	0,0780	0,1065	0,0253	0,0800	0,0012
	Algoritma	0,0218	0,0650	0,0815	0,0178	0,0999	0,0178	0,0172	0,0979
Data Real	Preprocessing	0,0379	0,1285	0,0020	0,1206	0,1476	0,0313	0,1281	0,0016
	Algoritma	0,1015	0,4758	0,5441	0,1397	0,5605	0,0949	0,1451	0,4913

Keterangan: SP: *Stopword Removal*, SM: *Stemming*, SO: *Sorting*, NONE: Tanpa *Stopword Removal*, *Stemming*, dan *Sorting*

Skenario Uji Coba

Perbandingan hasil persentase kemiripan menggunakan algoritma Levenshtein distance tanpa stopword removal, stemming, dan sorting dengan algoritma Levenshtein distance yang dibantu stopword removal, stemming, dan sorting dapat dimanfaatkan untuk melihat akurasi serta kinerja dari algoritma tersebut dalam mendeteksi kemiripan dokumen. Berdasarkan data set, dalam uji coba tingkat akurasi dilakukan dua jenis pengujian sekaligus, yaitu:

1. Uji coba dengan mengubah struktur kalimat
   Uji coba ini dimaksudkan untuk melihat tingkat akurasi algoritma Levenshtein distance dalam membandingkan dua jenis dokumen yang sama tetapi telah mengalami pengubahan struktur paragraf, yaitu mengubah paragraf deduktif menjadi induktif atau sebaliknya. Sedangkan untuk pengubahan struktur kata di dalam kalimat dapat dilakukan dengan mengubah kalimat aktif menjadi pasif atau sebaliknya. Misalnya tiga kalimat dibawah ini:
   a. Rabu kemarin Tiyo menghadiri seminar matematika
   b. Seminar matematika dihadiri Tiyo rabu kemarin
   c. Tiyo menghadiri seminar matematika rabu kemarin
2. Uji coba stopword removal, stemming, dan sorting
   Uji coba ini dimaksudkan untuk melihat kemampuan dari stopword removal, stemming, dan sorting yang bertugas membantu kinerja algoritma Levenshtein distance.

Selain uji coba tingkat akurasi, dilakukan juga uji coba waktu proses. Uji coba ini dimaksudkan untuk mengamati secara eksplisit waktu proses kinerja algoritma Levenshtein distance, baik yang di uji menggunakan bantuan stopword removal, stemming, dan sorting maupun tidak. Untuk mempermudah dalam melihat hubungan akurasi dengan waktu proses kinerja algoritma, maka digunakan data set yang sama dalam uji coba. Uji coba data real dilakukan dengan membandingan empat buah abstrak yang diperoleh dari internet. Dalam pengujian data real, setiap dokumen adalah murni sebuah abstrak tanpa dilakukan pengubahan struktur paragraf maupun kalimat. Selanjutnya sama seperti uji coba data set, pada data real dilakukan uji coba stopword removal, stemming, sorting, dan uji coba waktu proses. Berikut ini adalah rincian jenis data yang digunakan:

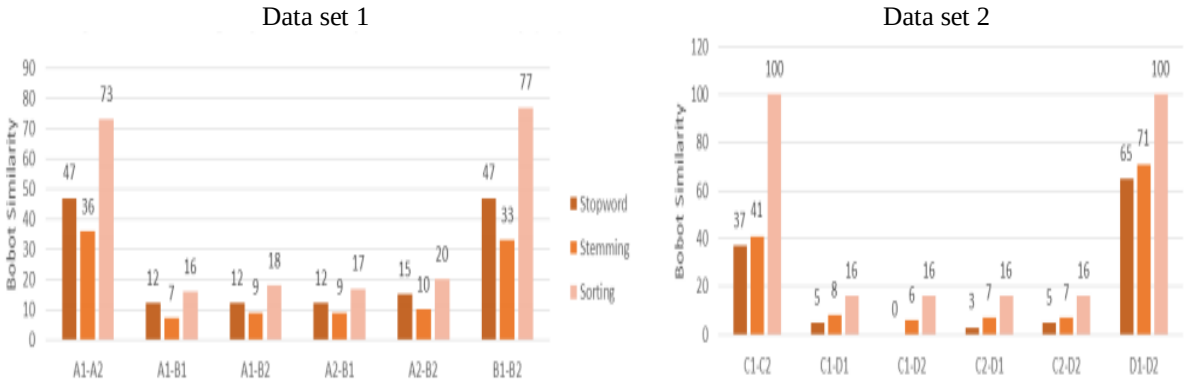
1. Data set 1:
   A1: Dokumen utama yang berisi sejarah dari batu lapis lazuli
   A2: Dokumen A1 yang mengalami manipulasi struktur kata dalam kalimat
   B1: Dokumen utama yang berisi kandungan dari batu lapis lazuli
   B2: Dokumen B1 yang mengalami manipulasi struktur kata dalam kalimat
2. Data set 2:
   C1: Dokumen utama yang berisi zat-zat pada cat

C2: Dokumen C1 yang mengalami manipulasi struktur kalimat di dalam paragraf
D1: Dokumen utama yang berisi molekul zat cair
D2: Dokumen D1 yang mengalami manipulasi struktur kalimat di dalam paragraf

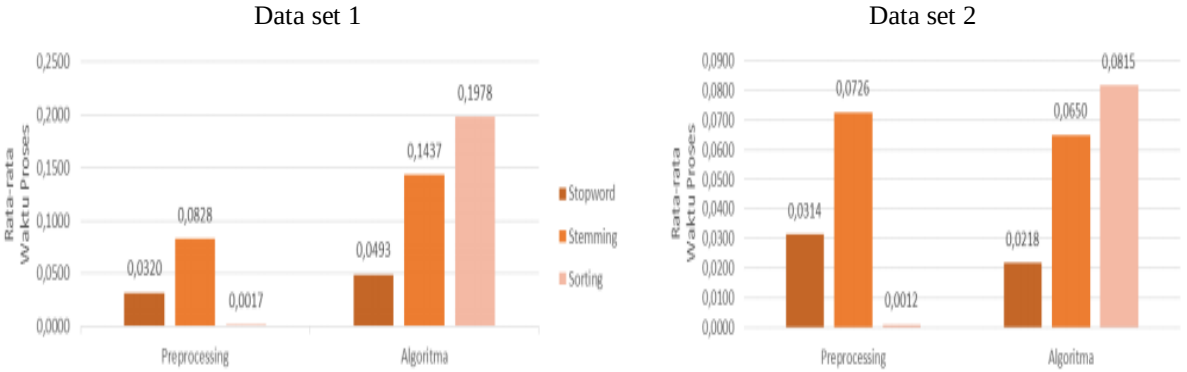
3. Data real:
   E1: Abstrak penelitian dengan judul 'Pemanfaatan Teknik Supervised Untuk Klasifikasi Teks Bahasa Indonesia'
   E2: Abstrak penelitian dengan judul 'Sistem Klasifikasi Dan Pencarian Jurnal Dengan Menggunakan Metode Naive Bayes Dan Vector Space Model'
   E3: Abstrak penelitian dengan judul 'Klasifikasi Otomatis Dokumen Berita Kejadian Berbahasa Indonesia Menggunakan Metode Naive Bayes'
   E4: Abstrak penelitian dengan judul 'Kalsifikasi Dokumen Naskah Dinas Menggunakan Algoritma Term Frequency – Inversed Document Frequency Dan Vector Space Model'

Analisa Akhir dari Kinerja Algoritma Levenshtein

Pengamatan dilakukan dengan menganalisa perbandingan output keakurasian dan waktu proses pada kedua data set. Hasil uji coba, pengaruh penggunaan stopword saja, stemming saja, dan sorting saja pada kedua data set dapat dilihat pada Gambar 2. Pengaruh penggunaan stopword saja, stemming saja, dan sorting saja terhadap waktu proses pada kedua data set dapat dilihat pada Gambar 3.

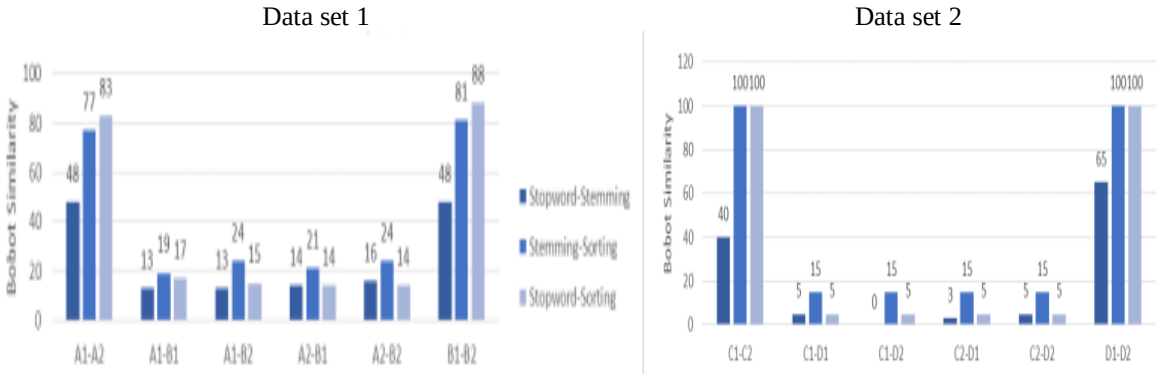


Gambar 2. Pengaruh penggunaan stopword saja, stemming saja, dan sorting saja terhadap bobot simalarity.

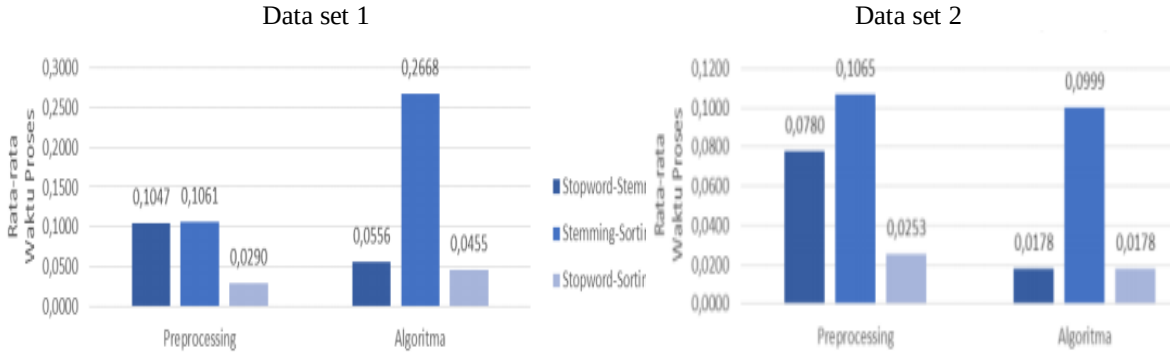


Gambar 3. Pengaruh penggunaan stopword saja, stemming saja, dan sorting saja terhadap waktu proses (dalam detik).

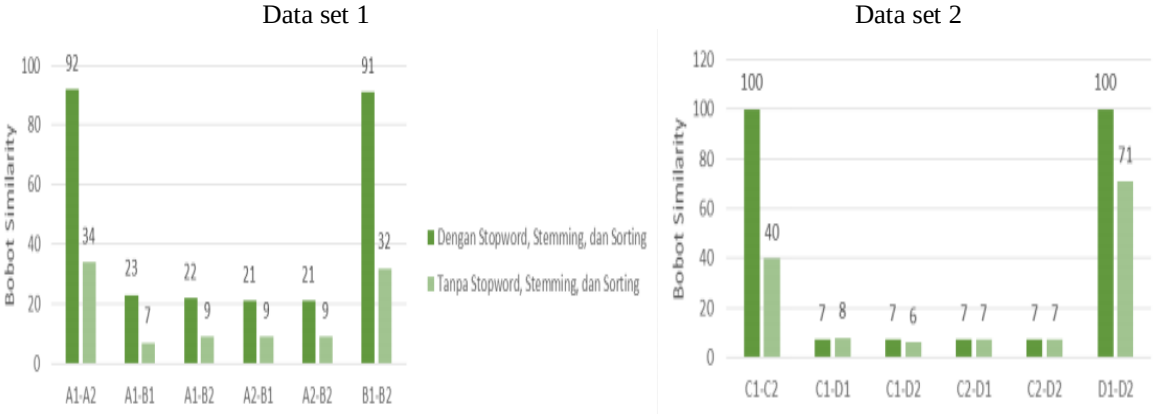
Gambar 4 menyajikan grafik pengaruh penggunaan stopword-stemming, stemming-sorting, dan stopword-sorting terhadap bobot similarity. Grafik pada Gambar 5 memperlihatkan pengaruh penggunaan stopword-stemming, stemming-sorting, dan stopword-sorting terhadap waktu proses (dalam detik). Gambar 6 mempelihatkan pengaruh penggunaan stopword, stemming, dan sorting secara sekaligus terhadap bobot similarity dan Gambar 7 memperlihatkan pengaruh penggunaan stopword, stemming, dan sorting secara sekaligus terhadap waktu proses (dalam detik).



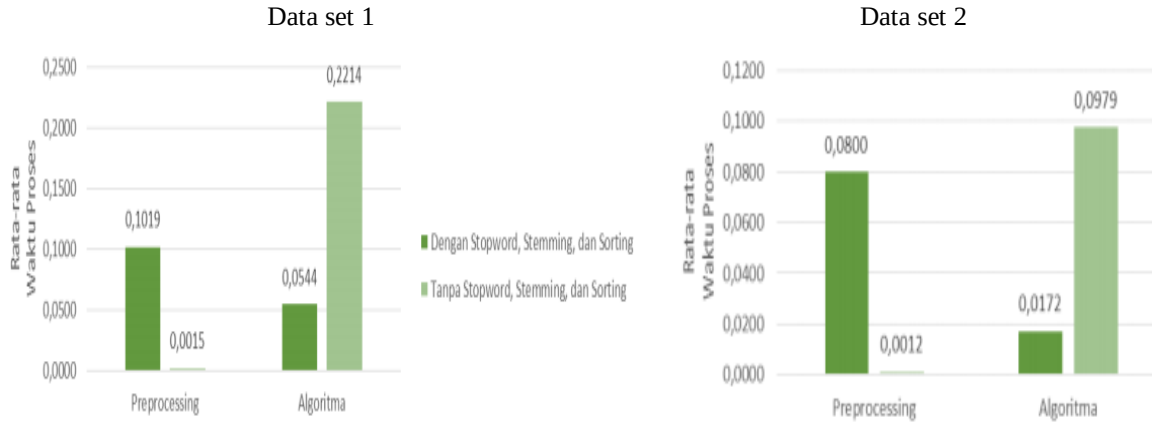
Gambar 4. Pengaruh penggunaan stopword-stemming, stemming-sorting, dan stopword-sorting terhadap bobot similarity.



Gambar 5. Pengaruh penggunaan stopword-stemming, stemming-sorting, dan stopword-sorting terhadap waktu proses (dalam detik).



Gambar 6. Pengaruh penggunaan stopword, stemming, dan sorting secara sekaligus terhadap bobot similarity.



Gambar 7. Pengaruh penggunaan stopword, stemming, dan sorting secara sekaligus terhadap waktu proses (dalam detik).

Hasil analisa pada uji coba data set 1 menunjukan bahwa penggunaan *stopword*, *stemming*, dan *sorting* sekaligus, menghasilkan bobot *similarity* yang lebih tinggi dibandingkan dengan penggunaan *stopword*, *stemming*, dan *sorting* secara terpisah. Pada uji coba data set 2 menunjukan bahwa baik penggunaan *stopword* maupun *stemming*, dapat menghasilkan bobot *similarity* yang lebih tinggi ketika digabungkan dengan proses *sorting*.

Pada uji coba data real menunjukan bahwa bobot *similarity* tertinggi dihasilkan dengan penggunaan *stemming-sorting*. Hasil tersebut disebabkan oleh nilai *distance* yang kecil, yang merupakan akibat dari banyaknya kata-kata umum pada masing-masing dokumen uji yang tidak tereliminasi karena tidak adanya proses *stopword*.

Pada akhirnya, penggunaan *sorting* menjadi sangat berpengaruh bagi algoritma Levenshtein *distance* dalam mendeteksi kemiripan dokumen teks. Besar atau kecilnya bobot *similarity* yang dihitung dengan algoritma Levenshtein *distance*, ditentukan berdasarkan isi pada setiap dokumen yang ingin diperiksa kemiripannya. Disarankan untuk tetap menggunakan proses *stopword*, *stemming*, dan *sorting* sekaligus agar *similarity* antar dokumen menjadi lebih terspesifikasi.

Hasil uji coba waktu proses memperlihatkan adanya keserupaan pada masing-masing proses. Artinya, sistem dengan algoritma Levenshtein *distance* memiliki kestabilan waktu dalam memroses perbandingan dokumen. Cepat atau lambatnya proses perbandingan, ditentukan dengan jumlah kata yang terdapat pada setiap dokumen. Banyaknya kata-kata yang tersimpan dalam *database*, ikut menentukan cepat atau lambatnya proses perbandingan. Kata-kata pada *database* bisa bertambah atau berkurang sesuai keinginan *user*. Semakin besar jumlah kata pada dokumen dan semakin bertambah kata pada *database*, maka proses perbandingan akan semakin lambat.

Pada akhirnya, durasi waktu proses menunjukan hasil yang sangat singkat dalam membandingkan dokumen, yaitu kurang dari 1 detik. Algoritma Levenshtein *distance* dapat mengenyampingkan durasi waktu proses pada saat membandingkan dokumen. Hal ini dikarenakan dalam mendeteksi kemiripan yang diutamakan adalah bobot *similarity* yang dihasilkan memiliki kecenderungan plagiarisme atau tidak.

KESIMPULAN

Berdasarkan serangkaian uji coba yang dilakukan dalam penelitian ini, dapat disimpulkanan bahwa:

1 Algoritma Levenshtein *distance* dapat diimplementasikan untuk mendeteksi kemiripan suatu dokumen teks dengan dokumen teks lainnya dengan cara pembentukan suatu matriks *string* untuk mendapatkan nilai *distance*, kemudian melakukan perhitungan bobot *similarity* antar dokumen teks berdasarkan nilai *distance* tersebut.

2 Implementasi algoritma Levebshtein *distance* menjadi lebih mudah dengan memanfaatkan program aplikasi (dalam hal ini menggunakan bahasa pemrograman PHP), karena proses deteksi kemiripan tidak dilakukan secara manual, namun dibantu oleh program aplikasi.

3 Penggunaan *sorting* menjadi sangat berpengaruh bagi algoritma Levenshtein *distance* dalam mendeteksi kemiripan dokumen teks. Dengan kata lain, Algoritma Levenshtein *distance* akan bekerja lebih baik jika kedua dokumen yang dibandingkan mempunyai urutan kata yang sama.

4 Penggunaan proses *stopword*, *stemming*, dan *sorting* sekaligus dapat memberikan hasil yang lebih spesifik. Artinya, dalam mendeteksi kemiripan dokumen teks, akan lebih baik jika dilakukan pengeliminasian kata-kata yang umum, dan megubah kata berimbuhan menjadi kata dasar.

REFERENSI

[1] Februariyanti, Herny. 2013. *Perancangan Pengindeks Kata pada Dokumen Teks Menggunakan Aplikasi Berbasis Web*. Jurnal Teknologi Informasi DINAMIK Volume 18 Nomor 2, Program Studi Sistem Informasi, Universitas Stikubank.

[2] Haldar, Rishin. dan Mukhopadhyay, D. 2011. *Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach*. Diakses tanggal 07 April 2015, dari Cornell University Library (http://arxiv.org/abs/1101.1232).

[3] Hultberg, J. dan J.P. Helger. 2007. *Seminar Course in Algorithms – Project Report*.

[4] Junedy, Richard. 2014. *Perancangan Aplikasi Deteksi Kemiripan Isi Dokumen Teks dengan Menggunakan Metode Leveshtein Distance*. Jurnal Pelita Informatika Budi Darma Vol. VII No.2, Jurusan Teknik Informatika, STMIK Budi Darma, Medan.

[5] Nafik, M.Z., Indriati dan A. Ridok. (2014). *Sistem Penilaian Otomatis Jawaban Esai Menggunakan Algoritma Levenshtein Distance*. Repositori Jurnal Mahasiswa PTIIK UB Vol 3 No. 4. Universitas Brawijaya, Malang.

[6] Pandawa, J.P. 2015. *Perancangan Sistem Deteksi Kemiripan Dokumen Menggunakan Algoritma Winnowing*. Skripsi Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Syarif Hidayatullah, Jakarta.

[7] Sjukani, Moh. 2013. *Algoritma (Algoritma dan Struktur Data 1) dengan C, C++, dan Java*. Jakarta: Mitra Wacana Media.

[8] Winarsono, D., D.O. Siahaan dan U. Yuhana. 2009. *Sistem Penilaian Otomatis Kemiripan Kalimat Menggunakan Syntactic Semantic Similarity pada Sistem E-Learning*. Jurnal Ilmiah KURSOR Menuju Solusi Teknologi Informasi Volume 5 Nomor 2, Jurusan Teknik Informatika, ITS, Surabaya.

[9] Zhan Su, Byung-Ryul Ahn, Ki-yol Eom, Min-koo Kang, Jin-Pyung Kim, dan Moon-Kyun Kim. 2008. *Plagiarism Detection Using the Levenshtein Distance and Smith-Waterman Algorithm*. The 3rd Intetnational Conference on Innovative Computing Information and Control, Department of Artificial Intelligence, University of Sungkyunkwan, Cheoncheon dong, Jangan-gu, Suwon, Korea. Diakses tanggal 10 April 2015, dari http://ieeexplore.ieee.org.