BAB II

KERANGKA TEORETIK, KERANGKA BERPIKIR DAN HIPOTESIS PENELITIAN

2.1. Kerangka Teoretik

2.1.1. Plagiarisme

Dalam kata Latin, kata plagiarisme adalah *plagiarius* yang artinya merampok, membajak. Secara singkat, plagiarisme adalah tindakan menyerahkan (*submitting*) atau menyajikan (*presenting*) ide atau kata/kalimat orang lain tanpa menyebutkan sumbernya (Sastroasmoro, 2006: 240).

Plagiarisme atau disebut juga plagiat merupakan penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri. Tindakan plagiat ini dapat dianggap sebagai tindak pidana karena mencuri hak cipta orang lain. Di dalam dunia pendidikan, pelaku plagiat (disebut juga plagiator) dapat dikenakan hukuman berat, seperti dikeluarkan dari sekolah/universitas.

Untuk memudahkan dalam memahami plagiarisme, dibuat pengklasifikasian mengenai plagiarisme. Menurut Sastroasmoro (2006: 240), klasifikasi atau jenis-jenis plagiarisme tersebut diantaranya:

1) Klasifikasi berdasarkan aspek yang dicuri:
    1) Plagiarisme ide

        Plagiarisme ide merupakan kategori plagiat yang menjiplak ide/gagasan orang lain tanpa mencantumkan nama penggagasnya. Plagiasrisme ini biasanya terjadi dalam pembuatan tema, judul atau

pembahasan sebuah karya tulis seperti skripsi. Penjiplakan ide memang cenderung lebih mudah terjadi karena ide atau tema bersifat luas. Namun, mengakui ide orang lain tanpa mencantumkan sumber juga disebut plagiarisme ide. Dalam karya tulis ilmiah, plagiarisme ide sering dihubungkan dengan laporan hasil penelitian yang replikatif.

2) Plagiarisme isi (data penelitian)

Plagiarisme isi (data) penelitian merupakan tindakan membuat data yang tidak ada menjadi seolah ada dan mengubah data, dengan maksud agar sesuai dengan yang dikehendaki oleh peneliti, karena peneliti tidak mempunyai data atau datanya tidak sesuai dengan yang dikehendaki. Atau dengan kata lain, plagiarisme kategori ini merupakan tindakan membuat data yang sebenarnya belum diteliti atau malah tidak ada sama sekali dan menjadikan seolah-olah data itu nyata. Plagiarisme ini mirip dengan manipulasi data, mengubah data penelitian dengan tujuan agar data tersebut membuktikan teori yang sudah ada.

3) Plagiarisme kata demi kata

Plagiarisme kata demi kata merupakan plagiarisme yang dilakukan dengan mengubah isi karya tulis, yaitu dengan mengganti sebuah kecil kalimat, mengganti satu paragraf atau lebih, atau meniru seluruh isinya sama persis meskipun ditulis dalam bahasa yang berbeda. Plagiarisme jenis ini merupakan plagiarisme yang paling mudah untuk dideteksi.

4) Plagiarisme total

Plagiarisme total merupakan plagiarisme yang dilakukan dengan menjiplak karya orang lain tanpa melakukan perubahan sedikitpun. Karya tulis yang dibuat merupakan salinan yang dibuat sama persis dengan karya orang lain. Hal ini menjadikan plagiarisme jenis ini merupakan jenis plagiarisme yang paling berat.

2) Klasifikasi berdasarkan sengaja atau tidaknya plagiarisme

Plagiarisme sengaja adalah mencuri atau menjiplak karya orang lain dengan sengaja (sudah) mempunyai niat dalam hati untuk berbuat demikian) untuk kepentingannya sendiri, seperti agar bisa lulus ujian, atau meningkatkan jabatan.

Plagiarisme yang tidak disengaja adalah plagiarisme yang terjadi karena ketidaktahuan (*ignorancy*). Ketidaktahuan ini biasanya terjadi dalam hal menggunakan dokumentasi, teknik mengutip karya tulis dan parafrase kalimat yang keliru. Meskipun plagiarisme ini terjadi dengan tidak disengaja, namun sanksi yang dikenakan sama seperti plagiarisme yang disengaja, hanya saja dapat dicegah dengan mengikuti panduan cara menghindari plagiarisme.

3) Klasifikasi berdasarkan proporsi atau persentase kata, kalimat, dan paragraf

1) Plagiarisme ringan

Plagiarisme ringan adalah plagiarisme yang jumlah persentase kata, kalimat, dan paragraf yang dijiplak/ditiru kurang dari 30 persen.

2) Plagiarisme sedang

Plagiarisme sedang adalah plagiarisme yang jumlah persentase kata, kalimat, dan paragraf yang dijiplak/ditiru berkisar antara 30-70 persen.

3) Plagiarisme berat

Plagiarisme berat adalah plagiarisme yang jumlah persentase kata, kalimat, dan paragraf yang dijiplak/ditiru lebih dari 70 persen.

4) Klasifikasi berdasarkan pola plagiarisme

1) Plagiarisme kata demi kata (*word for word plagiarizing*)

Plagiarisme kata demi kata adalah pola plagiarisme dengan melakukan penjiplakan sebagian kecil kalimat, bisa juga paragraf, maupun seluruh isi dari makalah meskipun ditulis dengan bahasa lain. Jenis plagiarisme ini merupakan jenis plagiarisme yang paling mudah diidentifikasi.

2) Plagiarisme mosaik

Plagiarisme mosaik adalah pola plagiarisme yang penjiplakannya tidak dilakukan kata demi kata, namun diselang-seling atau disisipkan. Penulis meminjam kata, frase, atau kalimat dari penulis lain, kemudian menyambungnya dengan kata, frase, atau kalimat dari penulis lain tanpa memberikan rujukan, sehingga memberi kesan kalimat tersebut adalah kalimat asli penulis.

2.1.2. Algoritma Levenshtein *distance*

Algoritma Levenshtein *distance* dinamakan sesuai penemunya yaitu Vladimir Levenshtein. Dia menemukan algoritma tersebut pada tahun 1965. Pada saat itu, dia berhasil menemukan jarak (*distance*) antara dua *input* string.

Yang dimaksud dengan jarak (*distance*) adalah jumlah modifikasi yang dibutuhkan untuk mengubah suatu bentuk string ke bentuk string yang lain. Sebagai contoh hasil *output* algoritma ini, string "tessi" dan "messi" memiliki jarak (*distance*) 1 karena untuk mengubah string "tessi" menjadi "messi" hanya diperlukan satu operasi saja. Dalam kasus dua string diatas, string "tessi" dapat menjadi "messi" hanya dengan melakukan satu operasi *substitution* karakter 't' dengan karakter 'm' pada string "tessi".

Dalam teori informasi dan ilmu komputer, algoritma Leventshtein *distance* merupakan matriks yang digunakan untuk mengukur perbedaan jarak antara dua sekuens (Janowski dan Mohanty, 2010: 259). Hasil perhitungan Levenshtein *distance* antara dua string ditentukan berdasarkan jumlah minimum perubahan yang diperlukan untuk melakukan transformasi dari satu bentuk string ke bentuk string lain. Misalnya jika string sumber adalah "hello" dan string target adalah "hallo" maka hasil perhitungan Levenshtein *distance* (LD) = 1. Hal tersebut berarti dibutuhkan sebuah operasi (*substitution*) untuk mengubah string sumber menjadi sama dengan string target. Operasi yang dilakukan dan diperbolehkan digunakan dalam menentukan Levenshtein *distance* ini ada 3 macam operasi (Andhika, 2010: 1), yaitu:

1) *Insertion* (penyisipan)

Insertion atau penyisipan adalah operasi penyisipan sebuah karakter kedalam string tertentu. Misalnya penyisipan sebuah karakter ‘a’ kedalam string “kla” tepat setelah karakter ‘k’. Setelah operasi *insertion* dilakukan, string “kla” berubah menjadi “kala”.

2) *Deletion* (penghapusan)

Deletion adalah operasi penghilangan atau penghapusan sebuah karakter tertentu dari sebuah string. Misalnya menghapus karakter ‘s’ pada string “senam”. Setelah operasi *deletion* dilakukan, string berubah menjadi “enam”.

3) *Substitution* (penukaran)

Substitution adalah operasi penukaran sebuah karakter pada string tertentu dengan karakter lain. Misalnya, menukarkan karakter ‘b’ pada string “baru” dengan karakter baru ‘p’. Setelah operasi *substitution* dilakukan, string akan menjadi “paru”.

2.1.2.1. Cara Mendapatkan Nilai Jarak (*distance*)

Gambar 2.1. berikut ini menunjukkan *pseudocode* dari algoritma Levenshtein *distance* yang digunakan untuk mencari nilai jarak (*distance*) antara dua *input* string, dimana m adalah panjang dari string pertama, dan n adalah panjang dari string kedua.

```
Function
LevDistance(input s:string[1..m], t: string[1..n])
Deklarasi
i, j = integer;
d[0..m, 0..n] = integer;
Algoritma
for i from 0 to m {perbandingan dengan kosong}
     d[i, 0] = i;
for j from 0 to n {perbandingan dengan kosong}
    d[0, j] = j;
for j from 1 to n{
     for i from 1 to m{
          if s[i] = t[j] then
            d[i, j] = d[i-1, j-1]
          else
            d[i, j] = minimum(
                              d[i-1, j-1]+1,
                              d[i-1, j]+1,
                              d[i, j-1]+1
                         )
      }
}
return d[m, n]
```

Gambar 2.1. *Pseudocode* algoritma Levenshtein *distance*

Fungsi algoritma Levenshtein *distance* pada Tabel 2.1 berikut menggunakan *input* dua buah string dan menghasilkan nilai jarak (*distance*)-nya. Seperti yang telah disebutkan sebelumnya, nilai jarak (*distance*) merupakan nilai yang

menunjukkan jumlah modifikasi minimum yang harus dilakukan untuk melakukan pengubahan string yang satu ke string yang lain.

Berikut adalah ilustrasi matriks pencari Levenshtein *distance* antara dua string yaitu “penjara” dan “jarah”. Untuk mengubah string "jarah“ menjadi “penjara” diperlukan 4 operasi, yaitu:

1) Menyisipkan karakter ‘p’

   jarah → pjarah

2) Menyisipkan karakter ‘e’

   pjarah → pejarah

3) Menyisipkan karakter ‘n’

   pejarah → penjarah

4) Menghapus karakter ‘h’

   penjarah → penjara

Representasi matriks-nya ditunjukkan dalam tabel berikut:

Tabel 2.1. Matriks Levenshtein *distance*

		p	**e**	**n**	**j**	**a**	**r**	**a**
	0	1	2	3	4	5	6	7
j	1							
a	2							
r	3							
a	4							
h	5							

2.1.2.2. Menghitung Persentase Dugaan Awal Kemiripan Isi Teks

Output yang dikeluarkan oleh aplikasi adalah nilai persentase dugaan awal kemiripan isi teks antar dokumen yang dibandingkan. Nilai ini disebut juga

plagiarized value. Plagiarized value diperoleh setelah melalui penghitungan dengan algoritma Levenshtein *distance*. Namun sebelum dihitung *plagiarized value*-nya, kedua dokumen yang dibandingkan harus melalui tahap *preprocessing*. Langkah-langkah dalam menentukan *plagiarized value* dugaan awal kemiripan isi teks adalah (Liaqat dan Ahmad, 2011: 11):

1) Setelah dilakukan penghitungan dari kedua string tersebut menggunakan algoritma Levenshtein *distance,* maka algoritma ini akan memberikan angka d*istance* yang merupakan nilai perbedaan dari kedua string. Misalkan: *String* sumber = *CS*, *String* target = *ST*, *Diff = distance*
2) Setelah didapatkan *distance* dari kedua string tersebut, maka formula untuk menghitung dugaan awal kemiripan (*plagiarized value*) kedua string dapat ditentukan sebagai berikut:

$$Plagiarized\ value = \left\{1 - \frac{Diff}{Max(CS, ST)}\right\} * 100$$

Keterangan:

Max(CS, ST) merupakan nilai yang paling panjang yang diberikan dari perbandingan *CS* dan *ST*. Sedangkan *plagiarized value* adalah persentase dugaan awal kemiripan antara string yang dibandingkan.

Contoh kasus:

CS = plagiarismdetection

ST = plagiarism

Setelah dilakukan perhitungan dengan Levenshtein *distance,* misalnya didapat:

Diff = 9

Max(CS, ST) = 19 (panjang dari *CS*)

Maka perhitungan selanjutnya adalah:

$$\text{Plagiarized Value} = \left\{1 - \frac{Diff}{Max(CS, ST)}\right\} * 100$$

$$= \left(1 - \frac{9}{19}\right) * 100$$

$$= 53$$

Hasil *plagiarized value* diatas adalah 53. Hal tersebut mengindikasikan dugaan awal kemiripan isi teks sebesar 53% antara string yang dibandingkan.

2.1.3. *Text Mining*

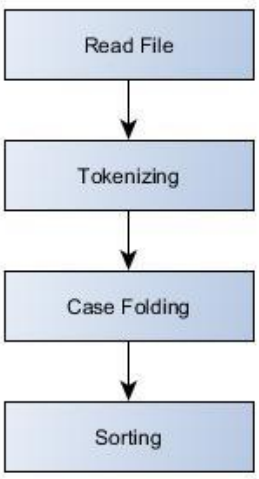
Dalam buku The Text Mining Handbook, *text mining* didefinisikan sebagai proses mendapatkan informasi secara intensif dimana pengguna berinteraksi dengan koleksi-koleksi dokumen menggunakan tools analisis. Secara umum, proses-proses pada *text mining* mengadopsi proses *data mining*. Tujuan dari *text mining* adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada *text mining* adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Proses-proses yang ada pada text mining diantaranya pemrosesan awal teks (*text preprocessing*), penemuan pola (*pattern discovery*), transformasi teks (*text transformation*), dan pemilihan fitur (*feature selection*).

Struktur data yang baik dapat memudahkan proses komputerisasi secara otomatis. Pada *text mining*, informasi yang akan digali berisi informasi-informasi yang strukturnya sembarang. Oleh karena itu, diperlukan proses pengubahan bentuk menjadi data yang terstruktur sesuai kebutuhan untuk proses selanjutnya. Salah satu tahap implementasi dari *text mining* adalah tahap *preprocessing*. Tahap *preprocessing* adalah tahap dimana aplikasi melakukan seleksi data yang akan

diproses pada setiap dokumen. Proses *preprocessing* ini meliputi *read file, tokenizing, case folding, filtering, stemming*, dan *sorting*. Kemudian tahap yang selanjutnya adalah melakukan *processing*. Tahap ini merupakan tahap inti dimana setiap kata akan diolah dengan algoritma tertentu sehingga mempunyai bobot terhadap setiap dokumen yang akan diseleksi. Tahap ini sering disebut juga *analyzing*, yang meliputi penemuan pola (*pattern discovery*), transformasi teks (*text transformation*), dan pemilihan fitur (*feature selection*).

2.1.4. Ekstraksi Dokumen

Teks yang akan dilakukan proses *text mining*, pada umumnya memiliki beberapa karakteristik diantaranya adalah memiliki dimensi yang tinggi, terdapat *noise* pada data, dan terdapat struktur teks yang tidak baik. Dalam mempelajari suatu data teks, diperlukan tahap *preprocessing* yang dilakukan secara umum pada dokumen, yaitu *read file, tokenizing, case folding, filtering,* dan *stemming*. Namun pada skripsi ini proses *filtering* dan *stemming* tidak disertakan karena alasan tertentu. Gambar 2.2 berikut adalah tahap dari *preprocessing* pada skripsi ini:

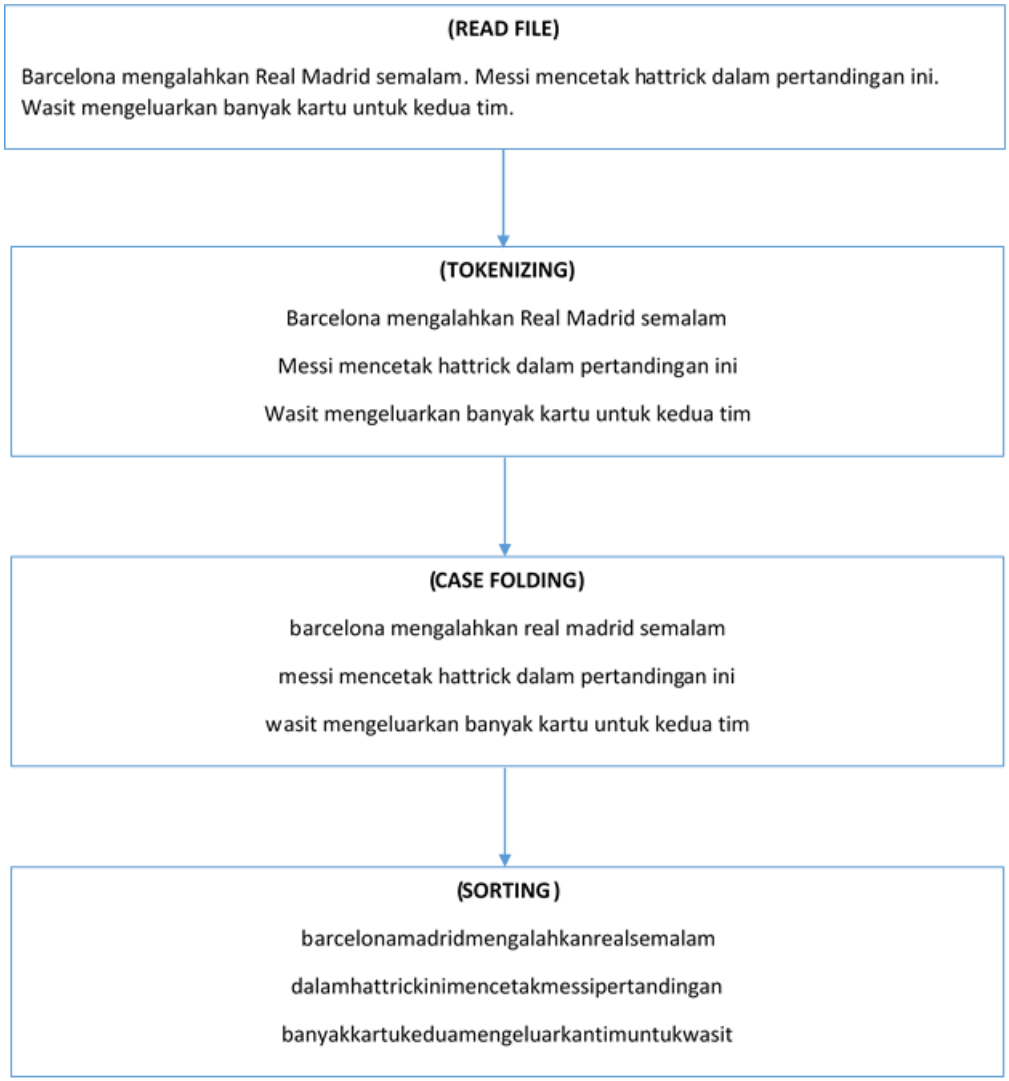


Gambar 2.2 Tahap *preprocessing*

2.1.4.1. *Read file, Tokenizing, Case Folding, Sorting*

Read file merupakan tahap membaca isi teks dari sebuah *file*/dokumen. *Tokenizing* adalah tahap pemotongan string *input* berdasarkan tiap kalimat yang menyusunnya.

Selanjutnya, *case folding* adalah proses mengubah semua huruf dalam dokumen menjadi huruf kecil. Sedangkan, *sorting* adalah pengurutan kata secara *ascending*/menaik sehingga pencocokkan string dokumen dilakukan pada data yang sudah terurut. Gambar 2.3 berikut adalah contoh dari proses *read file*, *tokenizing*, *case folding* dan *sorting*.

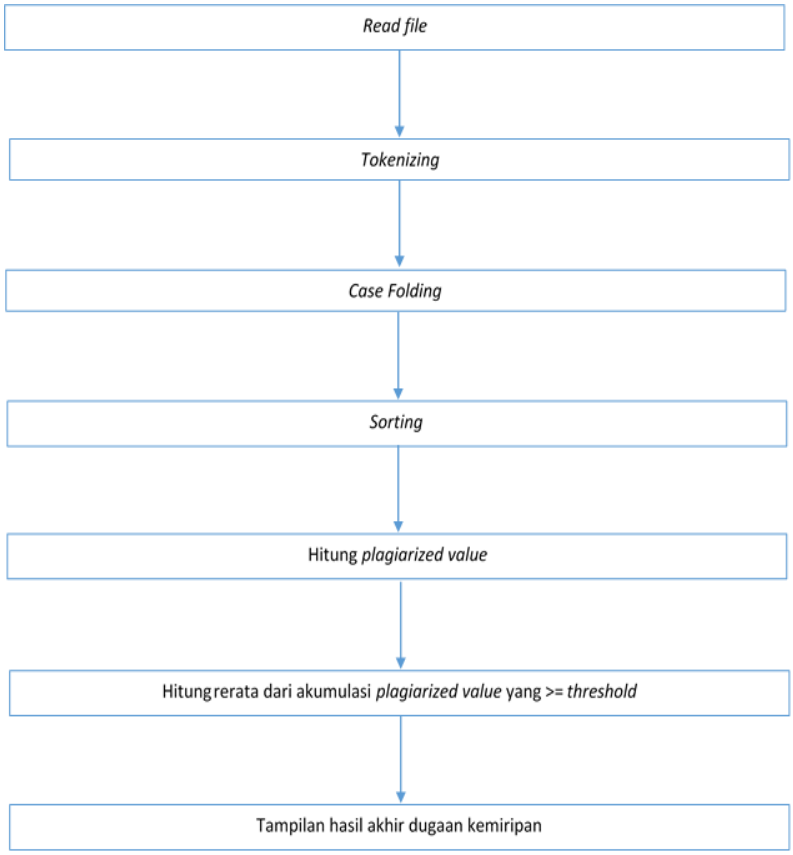


Gambar 2.3 Proses *read file*, *tokenizing*, *case folding*, dan *sorting*

2.2. Kerangka Berpikir

Algoritma Levenshtein *distance* merupakan salah satu algoritma *text similarity* yang dapat mengukur jarak perbedaan antar string yang dibandingkan. Oleh karena itu, sebelum mengimplementasi algoritma Levenshtein *distance* pada aplikasi, isi teks dari dokumen yang akan dibandingkan harus diekstrak terlebih dahulu untuk menghasilkan *token-token* string dari masing-masing dokumen teks. Kemudian *token-token* string antar dokumen teks dibandingkan satu per satu

menggunakan algoritma Levenshtein *distance*. Hasilnya digunakan untuk menghitung *plagiarized value* yang kemudian diakumulasi dan dicari reratanya. *Output* yang dihasilkan berupa persentase kemiripan isi teks antar dokumen yang dibandingkan. Kemiripan isi teks pada penelitian ini maksudnya adalah kemiripan dari kata-kata yang digunakan dalam penyusunan tiap-tiap kalimat pada dokumen teks yang dibandingkan menggunakan cara-cara penghitungan yang telah dijabarkan sebelumnya. Adapun pengertian dokumen teks pada penelitian adalah dokumen tugas siswa/mahasiswa yang berbasis teks. Blok perangkat lunak aplikasi pendeteksi plagiarisme berdasarkan kemiripan isi teks yang mengimplementasi algoritma Levenshtein *distance* ditunjukkan oleh gambar 2.4:



Gambar 2.4 Blok perangkat lunak

2.3. Hipotesis Penelitian

Berdasarkan teori-teori yang telah disebutkan diatas, dapat ditarik sebuah hipotesis bahwa algoritma Levenshtein *distance* dapat diimplementasi pada aplikasi yang dapat menghitung kemiripan isi teks dokumen tugas siswa dan mahasiswa sehingga dapat membantu pendeteksian dugaan awal plagiarisme, terutama plagiarisme kata demi kata (*word for word plagiarizing*).