BAB II

LANDASAN TEORI

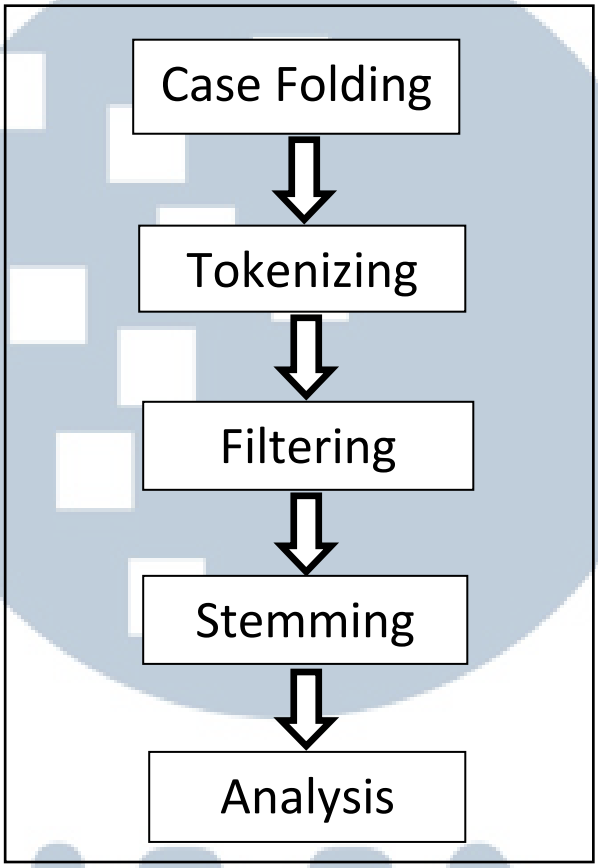
2.1 Text Mining

Text mining digunakan untuk mendeskripsikan teknik dari *data mining* yang secara otomatis menemukan sesuatu hal yang berguna atau sebuah pengetahuan baru dari sebuah teks yang tidak terstruktur (Han dan Kamber, 2000). Secara garis besar teknik ini bekerja untuk menemukan kata-kata yang mewakili isi dari suatu dokumen, kemudian kata tersebut akan dicarikan kesamaan satu dengan lainnya untuk digunakan sebagai bahan analisis. Algoritma dalam text mining dibuat untuk dapat mengenali data yang sifatnya semi terstruktur misalnya sinopsis, abstrak maupun isi dari dokumen-dokumen (Gupta dan Lehal, 2009). Ada beberapa tahapan proses *text mining* dalam mengolah data *input* agar lebih mudah dipahami pada saat melakukan analisis yang disebut tahap *preprocessing*. Tahapan ini akan dijabarkan dan dijelaskan pada subbab berikutnya.

2.2 Tahapan Preprocessing Data

Proses *text mining* dilakukan dalam beberapa tahapan awal (*preprocessing*) diantaranya adalah *tokenizing*, *filtering*, *stemming*, *tagging*, dan *analyzing* (Herwansyah, 2009). Proses *text mining* untuk teks bahasa Indonesia hanya sampai pada tahap *stemming* saja sebelum lanjut ke tahap *analyzing*. Tahap *tagging* tidak digunakan pada teks berbahasa Indonesia karena kata dalam bahasa Indonesia tidak

mempunyai bentuk lampau (Hatta dkk., 2010). Urutan tahapan *preprocessing* data dapat dilihat pada Gambar 2.1.



Gambar 2.1 Tahapan *Prepocessing* Data (Sumber: Hatta dkk., 2010)

1. Case Folding dan Tokenizing

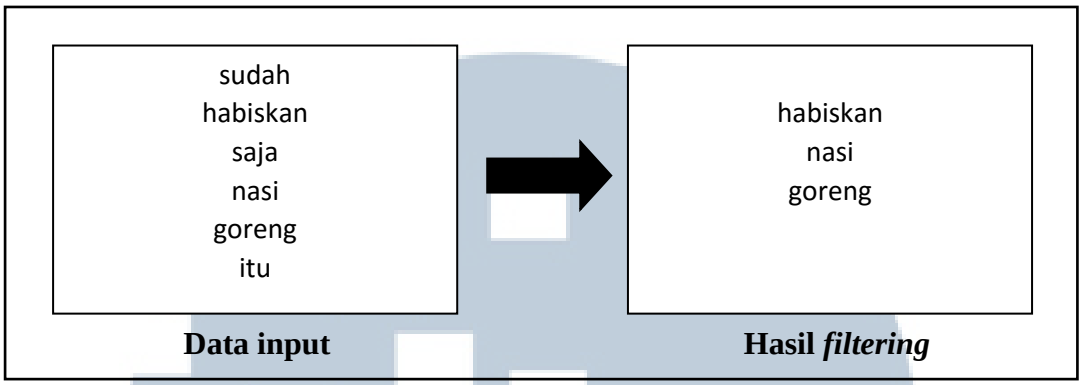
Case folding adalah tahap mengubah semua huruf dalam dokumen menjadi huruf kecil, selain itu karakter non-huruf juga akan dihilangkan (Imbar, 2014). Perubahan menjadi huruh kecil dan menghilangkan delimeter bertujuan untuk pemerataan format kalimat ke dalam bentuk yang sama dan sederhana agar memudahkan proses *tokenizing*. Proses menghilangkan delimiter-delimiter (pembatas) non-huruf, seperti titik (.), koma (,), seru (!), tanya (?), spasi dan karakter angka. *Tokenizing* adalah tahap pemotongan *string* input berdasarkan tiap kata yang menyusunnya (Maulana, 2015). Sebagai contoh diberikan sebuah kalimat

"Nama saya adalah Arvin !", maka hasil dari tahap *case folding* dan *tokenizing* akan menghasilkan pemecahan kalimat menjadi beberapa *string* (nama; saya; adalah; arvin).

2. Filtering

Tahap selanjutnya adalah *filtering*, tahap ini akan melakukan penyaringan terhadap hasil kata dari tahap *tokenizing*. Ada 2 teknik penyaringan yang dimaksud, yaitu yang pertama membuang kata-kata yang tidak penting dari tabel atau dikenal dengan istilah *stopword list* dan yang kedua adalah menyimpan kata-kata yang diangkap penting atau dikenal dengan istilah *word list*. *Stopword list* adalah sebuah penyaringan kata yang tidak layak untuk dijadikan sebagai pembeda atau sebagai kata kunci, sedangkan *word list* adalah daftar kata yang mungkin digunakan sebagai kata kunci dalam pencarian dokumen (Wicaksono, 2012). *Stopword list* menghasilkan hasil akhir dalam berbagai kemungkinan data tanpa dibatasi oleh bidang atau klasifikasi tertentu. Berbeda dengan *word list* yang hasil akhirnya telah ditentukan karena sifat dari *word list* yang menyimpan kata penting pada setiap kemunculan sehingga akan menggukan tempat penyimpanan yang lebih besar dan mempengaruhi kecepatan proses. Ada beberapa contoh kata yang termasuk dalam *stopword list* bahasa Indonesia (Kabul, 2012).

a. Kata penghubung (sesudah, selesai, sebelum)
b. Kata tugas (bagi, dari, dengan, pada)
c. Kata keterangan (sangat, hanya, lebih)
d. Kata bilangan (beberapa, banyak, sedikit)
e. Kata ganti (kami, mereka, kita, itu)
f. Dan lain sebagainya



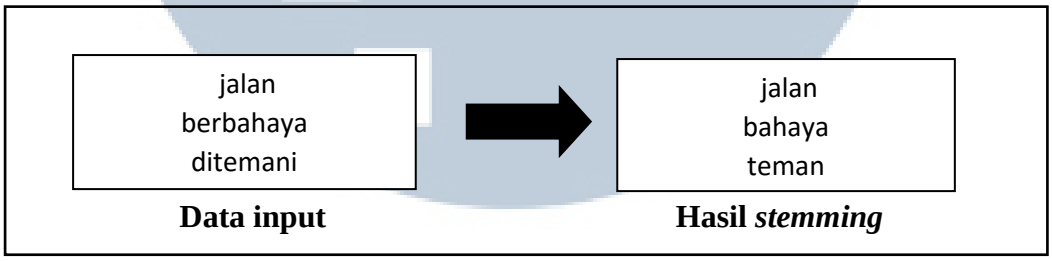
Gambar 2.2 Proses *Filtering* (Sumber: Wicaksono, 2012)

Berdasarkan Gambar 2.2, ada beberapa contoh kata yang termasuk ke dalam kata tidak penting (*stop word*), seperti "sudah", "saja", dan "itu". Menentukan kata yang termasuk ke dalam *stopword list* dapat berasal dari berbagai sumber tergantung konteks tujuan penelitian, seperti fokus pada kata-kata yang bersifat ilmiah atau kata kunci yang dapat mewakili sebuah kelompok. Kesalahan penulisan, angka, kata yang bersifat umum akan dimasukkan ke dalam *stopword list*.

3. Stemming

Stemming adalah proses untuk mencari kata dasar (*root*) pada hasil dari proses *filtering* (Utomo, 2013). Banyak algoritma yang dapat digunakan dalam melakukan proses *stemming* di antaranya algoritma Suffix-Stripping, algoritma Lemmatisation, algoritma Stohastic, algoritma Vega, dan sebagainya. Namun, kebanyakan algoritma di atas digunakan untuk proses *stemming* teks bahasa Inggris lalu bagaimana dengan teks bahasa Indonesia. Proses *stemming* pada teks berbahasa Indonesia lebih rumit atau kompleks karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan *root word* dari sebuah kata (Lestiyanto, 2014). Oleh karena itu, diperlukan algoritma yang cocok untuk dokumen teks berbahasa

Indonesia. Berdasarkan penelitian dan perbandingan algoritma lain, ditemukan ada dua algoritma yang banyak digunakan dan cocok untuk melakuakan proses *stemming* pada teks berbahasa Indonesia, yaitu algoritma *Porter* dan algoritma Nazief Adriani. Melihat dari performa keduanya, dinyatakan pada penelitian (Agusta, 2009) bahwa *stemming* dokumen teks berbahasa Indonesia menggunakan algoritma Nazief Adriani memiliki keakuratan yang lebih tinggi dibandingakan dengan algoritma *Porter*. Contoh hasil keluaran dari tahap ini dapat dilihat pada Gambar 2.3. Penjelasan algoritma Nazief Adriani akan dijelaskan pada subbab berikutnya.



Gambar 2.3 Proses *Stemming* (Sumber: Budi dkk., 2006)

4. Analyzing

Tahap *analyzing* meruapakan langkah terakhir setelah mendapatkan kata yang lolos dari tahap *case folding*, *tokenizing*, *filtering*, dan *terming*. Kata yang lolos sampai tahap ini disebut sebagai *term*. Kemudian *term* ini akan dihitung kemunculannya setiap ada *input* dari dokumen dan disimpan dalam sebuah penampung yang digunakan untuk penentuan pola.

2.3 Algoritma Nazief Adriani

Sebelum masuk ke dalam algoritma ini, sebaiknya diketahui bahwa bahasa Indonesia memiliki morfologi atau struktur kata infleksional dan derivasional. Menurut Tala (2003) infleksional adalah struktur kata yang diikuti dengan imbuhan tanpa merubah kata dasar. Contoh infleksional dapat ditemui pada sebuah kata dengan akhiran -kah, -lah, -tah, -pun, -ku, -mu, dan –nya. Derivasional adalah struktur kata dengan tambahan imbuhan dan merubah kata dasar. Struktur derivasional dalam bahasa Indonesia terdiri dari prefix, suffix, dan konfix. Menurut seorang ahli Chaer (1994), prefiks adalah afiks yang diimbuhkan dimuka bentuk dasar, sedangkan sufiks adalah suatu afiks yang diimbuhkan pada posisi akhir bentuk dasar. Contoh prefix dapat ditemukan pada kata yang diawali dengan imbuhan per-, ter-, ber-, di-, ke-, te-, se-, be-, me-, pe-, meng-, dan peng-. Contoh suffix dapat ditemukan pada kata yang diakhiri dengan imbuhan -kan, -i, dan –an. Konfix adalah sebuah kata hasil gabungan dari prefix dan suffix.

Algoritma Nazief Adriani

Input : Kata
Output : Kata Dasar

Langkah – langkah :

1. Cek apakah kata yang di-*input* ada di dalam kamus, jika terdapat dalam kamus proses berhenti dan jika tidak ada lanjut ke langkah 2.
2. Membuang *Inflection Suffix*, kemudian cek kata di dalam kamus, jika terdapat dalam kamus proses berhenti dan jika tidak ada lanjut ke langkah 3.
3. Membuang *Derivation Suffix*, kemudian cek kata di dalam kamus, jika terdapat dalam kamus proses selesai dan jika tidak ada lanjut ke langkah 4.
4. Membuang *Derivation Prefix*, kemudian cek kata di dalam kamus, jika terdapat dalam kamus proses berhenti dan jika tidak ada, kembalikan kata dalam bentuk semula kemudian jadikan kata tersebut sebagai *rootword* baru dan proses selesai.

Gambar 2.4 Langkah Proses Algoritma Nazief Adriani
(Sumber: Agusta, 2009)

Algoritma Nazief Adriani ditemukan oleh Bobby Nazief dan Mirna Adriani. Algoritma ini juga didasari dengan morfologi bahasa Indonesia dengan pengelompokan imbuhan awalan (prefix), sisipan (infix), akhiran (suffix), dan gabungan (konfix). Kamus Bahasa Indonesia dan proses penyusunan kembali kata hasil *stemming* yang berlebih (*recording*) menjadi bagian penting untuk menentukan keberhasilan sebuah tahap *stemming*. Berikut langkah-langkah proses *stemming* Nazief Adriani.

1. Melakukan pencarian kata hasil *tokenizing* dalam kamus yang tersimpan di dalam *databse* apakah ditemukan atau tidak. Jika ditemukan, maka algoritma selesai. Sebaliknya jika tidak ditemukan, maka lanjut ke langkah 2.
2. Membuang imbuhan akhir infleksi atau *Inflection Suffix* (“-lah”, “- kah”,“-ku”,”-mu”, dan ”-nya”). Contoh kata yang dapat ditemui pada tahap ini adalah “percayalah”, “melupakanmu”, “siapakah”, dan sebagainya. Jika ditemukan imbuhan akhir di atas, maka kembalikan dalam bentuk kata yang baru. Jika tidak ada, kembalikan kata dalam bentuk awal. Cek kembali kata yang dikembalikan di dalam kamus. Jika ditemukan, algoritma berhenti dan jika belum, lanjut ke langkah 3.
3. Membuang imbuhan akhir derivasi atau *Derivation Suffix* (“-i”, “-an”, atau “-kan”). Contoh kata yang dapat ditemui pada tahap ini adalah “kekinian”, “mainan”, “menyanggupi”, dan sebagainya. Jika ditemukan imbuhan akhir di atas, maka kembalikan dalam bentuk kata yang baru. Jika tidak ada, kembalikan kata dalam bentuk awal. Cek kembali kata yang

dikembalikan di dalam kamus. Jika ditemukan, algoritma berhenti dan jika tidak ditemukan, lanjut ke langkah 4.

4. Membuang imbuhan awal derivasi atau *Derivation Prefix* (“di-”, “ke-”, “se-”, “me-”, “be-”, “pe-”, “te-”). Contoh kata yang dapat ditemui pada tahap ini adalah “menyapu”, “pembatas”, “berkembang”, dan sebagainya. Tahap ini merupakan tahap dengan proses yang lebih panjang dari sebelumnya karena terdapat banyak sekali kata dasar yang mengalami peluluhan atau perubahan akibat bertemu huruf awalan tertentu sehingga diperlukan sebuah peraturan untuk menentukan kata dasar awal. Berikut beberapa contoh perubahan kata dasar.

   a. Awalan Se-

      (Se + kosonan dan vokal tidak berubah)

      Contoh : se + ekor = seekor, se + hati = sehati

   b. Awalan Me-

      (Me + vokal a,i,u,e,o menjadi “meng-“)

      Contoh : me + ubah = mengubah, me + ekor = mengekor

      (Me + kosonan ‘b’ menjadi “mem”)

      Contoh : me + beli = membeli, me + buncit = membuncit

      (Me + kosonan ‘s’ menjadi “meny-“)

      Contoh : me + suap = menyuap, me + supir = menyupir

      (Me + kosonan ‘t’ menjadi “men-“)

      Contoh : me + tuang = menuang, me + tuker = menukar

(Me + kososnan l, m, n, r, w tetap menggunakan “me-”)

Contoh : me + rawat = merawat, me + masak = memasak, me + warna = mewarna, me + lemar = melempar, me + naik = menaik

c. Awalan Ke-

(Ke + kosonan dan vokal tidak berubah)

Contoh : ke + bawa = kebawa, ke + luar = keluar

d. Awalan Pe-

(Pe + kosonan h, g, k dan vokal a, i, u, e, o menjadi “per”)

Contoh : pe + gelar + an = pergelaran, pe + kantor + an = perkantoran

(Pe + kosonan ‘t’ menjadi “pen”)

Contoh : pe + tukar = penukar, pe + tangkap = penangkap

(Pe + kosonan j, d, c, z menjadi “pen”)

Contoh : pe + jahit = penjahit, pe + didik = pendidik, pe + cuci = pencuci, pe + zina = penzina

(Pe + kosonan b, f, v menjadi “pem”)

Contoh : pe + ber = pemberi, pe + bunuh = pembunuh

(Pe + kosonan ‘p’ menjadi “pem”)

Contoh : pe + piker = pemikir, pe + pangkas = pemangkas

(Pe + kosonan ‘s’ menjadi “peny-”)

Contoh : pe + siram = penyiram, pe + sabar = penyabar

(Pe + kosonan l, m, n, r, w, y dan vokal tidak berubah)

Contoh : pe + lupa = pelupa, pe + makan = pemakan, pe + nanti = penanti, pe + rusak = perusak, pe + wangi = pewangi

Penghapusan dan berubahan kata dasar ke dalam bentuk yang benar, didasarkan pada ketentuan di atas. Proses tidak akan dilanjutkan, apabila kata yang masuk tergolong dalam awalan dan akhiran yang di larang sesuai dengan Tabel 2.1. Tahap 4 ini juga memiliki kesamaan dengan tahap sebelumnya, yaitu setelah melakukan penghapusan dan perubahan kata dan kata tersebut terdapat di dalam kamus, maka fungsi akan mengembalikan kata terbaru dan jika tidak ditemukan, maka yang dikembalikan adalah kata awal sebelum dilakukan penghapusan dan perubahan. Cek kembali kata yang dikembalikan fungsi di dalam kamus. Jika ditemukan, algoritma berhenti dan jika tidak ditemukan, lanjut ke langkah tahap 5.

Table 2.1 Kombinasi Awalan dan Akhiran yang Tidak Diizinkan
(Sumber: Agusta, 2009)

Awalan	Akhiran (tidak diizinkan)
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan
te-	-an

5. Tahap ini adalah tahap pengembangan untuk mengoptimalkan algoritma Nazief Adriani yang tidak memiliki aturan pada reduplikasi. Reduplikasi yang dimaksud adalah kata yang ditulis berulang bisa sama atau berbeda, tetapi memiliki satu arti. Contoh kata reduplikasi, seperti “hal-hal”, “kupu-kupu”, “rumah-rumah”, dan sebagainya. Cara mengetahui kata dasar dari reduplikasi ini adalah melakukan cek dengan mengambil potongan kata menggunakan panjang *string* dan dicari apakah potongan kata tersebut ada pada kamus. Jika ditemukan, maka kata baru dikembalikan dan algoritma

berhenti, sedangkan jika tidak ditemukan, maka kata tersebut diasumsikan sebagai kata dasar baru (*root word*) dan algoritma berakhir.

2.4 Analisis Cluster

Analisis Cluster sendiri dibagi menjadi 2 metode yang berbeda, yaitu metode *hierarchical* (hirarki) dan *non-hierarchical* (partisi). Menurut Setiawan (2014) metode hirarki adalah teknik *clustering* yang digunakan saat jumlah *cluster* yang diinginkan belum diketahui, sedangkan metode partisi adalah teknik *clustering* yang digunakan saat jumlah *cluster* sudah ditentukan atau kata lain membagi data dari n objek ke dalam k *cluster* yang sudah ditentukan terlebih dahulu. Metode *hierarchial clustering* sendiri terdiri dari *single lingkage method*, *complete linkage method*, *average linkage method*, *centroid linkage method*, *ward method,* dan *median method*. Metode *non-hierarchial* terdiri dari K-Means, K-Medoids, Fuzzy-C-Mean, dan berbagai pengembangan metode lain.

2.5 Metode K-Medoids

Metode K-Medoids adalah sebuah teknik *clustering* yang bertujuan untuk mencari hasil terbaik dari semua objek yang ada di dalam *dataset* sebagai perwakilan karakteristiknya (Zeidat, 2004).

Metode K-Medoids

Input : Data, Jumlah *Cluster* (K)
Output : Anggota data set pada Cluster K

Langkah – langkah :
1. Tentukan jumlah cluster K dan secara acak tentukan data yang akan menjadi *medoids* untuk setiap cluster K.
2. Bandingkan dan hitung jarak antar *medoids* dengan *non-medoids*. Cari perbandingan jarak yang paling dekat dengan Manhattan Distance lalu jumlahkan untuk mencari *total cost*.
3. Ambil acak pada cluster K yang bukan medoids dari data set dan tukar dengan salah salah satu *medoids*.
4. Ulangi langkah 2 dan hitung *total cost* baru. Jika *total cost* baru lebih kecil dari nilai sebelumnya, ganti medoids lama dengan yang baru. Jika lebih besar, batalkan pertukaran medoids. Ulangi hingga tidak ada perubahan *medoids* lagi.

Gambar 2.5 Langkah Proses Metode K-Medoids
(Sumber: Park dan Jun, 2009)

Penjelasan untuk cara kerja dari metode K-Medoids akan di simulasikan pada sebuah contoh sederhana berikut ini.

Sebuah *cluster* memliki data set dengan 5 objek yang terbagi dalam 2 *cluster* (k = 2).

X1	3	1
X2	6	4
X3	3	6
X4	1	2
X5	9	3

Gambar 2.6 Data Set Awal 2 *Cluster* (Sumber: Park dan Jun, 2009)

1. Asumsikan X1 (c1) dan X4 (c2) terpilih menjadi *medoids*, sehingga titik pusat c1 = (3,1) dan c2 = (1,2).
2. Hitung jarak antara titik pusat dengan objek yang dekat dengan *medoids*. Hitung *cost* dengan menggunakan Manhattan Distance. Cara perhitungan ini akan dijelaskan pada subbab berikutnya. Hasil perhitungan biaya *cost*

dengan jarak terdekat dapat dilihat pada Gambar 2.12 dengan latar belakang berwarna kuning.

Cost (jarak) ke c1					
i	c1		Objek Data (Xi)		Cost (jarak)
2	3	1	6	4	**6**
3	3	1	3	6	**5**
5	3	1	9	3	**8**

Cost (jarak) ke c2					
i	c2		Objek Data (Xi)		Cost (jarak)
2	1	2	6	4	7
3	1	2	3	6	6
5	1	2	9	3	9

Gambar 2.7 Perhitungan Data Set Awal 2 *Cluster* (Sumber: Park dan Jun, 2009)

Cluster yang terbentuk hasil dari perhitungan di atas:

Cluster1 = {(3,1)(6,4)(3,6)(9,3)}

Cluster 2 = {(1,2)}

Diketahui bahwa objek (6,4) (3,6) (9,3) adalah jarak terdekat untuk masing-masing *cluster*, maka dihitung total *cost* dari jumlah jarak. Hitung total *cost* degan persamaan rumus di bawah ini.

$$cost(x, c) = \sum_{i=1}^{d} |x_i - c_i| \quad \ldots(2.1)$$

Keterangan :

x = objek data

c = *medoids*

d = dimensi dari objek (contoh ini menggunakan dua dimensi)

total *cost* = {*cost*((3,1).(6,4)) + *cost*((3,1),(3,6)) + *cost*((3,1),(9,3))}

= (6 + 5 + 8)

= 19

3. Pilih secara acak satu *non-medoids* (O').

Asumsikan O' = X5(9,3) sehingga *medoids* sekarang adalah c1(3,1) dan O'(9,3).

4. Ulangi langkah 1 untuk menghitung total *cost*.

Cost (jarak) ke c1					
i	c1		Objek Data (Xi)		Cost (jarak)
2	3	1	6	4	6
3	3	1	3	6	5
4	3	1	2	1	1

Cost (jarak) ke O'					
i	O'		Objek Data (Xi)		Cost (jarak)
2	9	3	6	4	4
3	9	3	3	6	9
4	9	3	2	1	9

Gambar 2.8 Perhitungan Data Set dengan *Non-Medoids* 2 *Cluster*
(Sumber: Park dan Jun, 2009)

total *cost* = {*cost*((3,1).(3,6)) + *cost*((3,1),(2,1))} + {*cost*((9,3),(6,4))}

= (5 + 1) + (4)

= 6 + 4

= 10

Bandingkan antara total *cost* c1,c2 dengan total *cost* c1,O'. Ternyata total *cost* c1,O' lebih kecil dari pada total *cost* c1,c2 (10 < 19), maka O' menjadi *medoids* baru menggantikan c2 yang lama. Lakukan kembali langkah 3 hingga tidak ada perubahan lagi. Apabila total *cost* baru lebih besar atau

sama dengan total *cost* lama, maka batalkan pertukaran *medoids* lama dengan O'.

2.6 Manhattan Distance

Manhattan Distance atau dikenal dengan nama City Block Distance adalah cara menghitung jarak yang telah ditempuh antara dua buah objek. Perhitungan ini dilakukan secara tegak lurus antar jarak objek dalam sebuah dimensi. Berdasarkan penelitian sebelumnya, menyatakan metode Manhattan Distance memiliki waktu proses yang lebih singkat dibandingkan metode perhitungan lain, yaitu Euclidean Distance (Gautama, 2015). Persamaan metode Manhattan Distance seperti di bawah ini.

$$d = \sum_{i=1}^{n} |X_i - Y_i| \quad \ldots(2.2)$$

Keterangan :

d = total jarak tempuh

n = dimensi dari objek

X = koordinat objek pertama

Y = koordinat objek kedua

Perhitungan di atas berdasarkan penjumlahan jarak selisih antara dua buah objek dan hasil yang didapatkan bersifat mutlak (Sinwar dan Kaushik, 2014). Berikut contoh cara menghitung 2 objek menggunakan Manhattan Distance. Diketahui 2 buah objek dengan koordinat masing-masing A(6,-5,1) dan B (3,2,9). Masukan koordinat kedalam rumus Manhattan Distance dan mendapatkan hasil sebesar 18.

d = |6 – 3| + |-5 -2| + |1 – 9|

= 3 + 7 + 8

= 18

2.7 Purity

Purity adalah salah satu metode yang dapat digunakan untuk menguji *cluster*. Metode ini menguji apakah anggota yang paling banyak terdapat dalam *cluster* sudah sesuai dengan dengan kelompokya (Handoyo dkk., 2014). Pengujian dikatakan baik apabila hasil *purity* mendekati nilai 1, sedangkan dikatakan buruk bila hasil *purity* mendekati nilai 0. Hasil yang mendekati 1 mengindikasikan bahwa banyak dokumen yang telah sesuai atau benar pengelompokannya (Prilianti dan Wijaya, 2014). Persamaan metode *purity* per *cluster* seperti di bawah ini.

$$purity\ (j) = \frac{1}{n_j} \sum_{i=1}^{k} max_j |c_i \cap t_j| \quad \ldots(2.3)$$

Sedangkan persamaan metode *purity* untuk semua *cluster* seperti di bawah ini.

$$purity = \sum_{l=1}^{k} \frac{n_j}{n} purity\ (j) \quad \ldots(2.4)$$

Keterangan :

n_j = total jumlah objek per *cluster*

n = total jumlah objek

k = banyaknya *cluster*

c_i = *cluster* yang terdapat di dalam C

t_j = klasifikasi yang memiliki perhitungan maksimum untuk *cluster* c_i

2.8 Likert Scale

Aplikasi yang telah berhasil dibangun memerlukan sebuah pengujian penerimaan dari berbagai aspek terhadap aplikasi tersebut. Hal tersebut berkaitan dengan respon dari pengguna lain atau *user* yang menggunakan aplikasi. Salah satu pengujian yang dapat digunakan, yaitu menggunakan metode penelitian survei. Penelitian survei adalah penelitian yang mengambil sampel dari suatu populasi dan menggunakan kuesioner sebagai alat pengumpulan data pokok (Singarimbun dkk., 1995). Kuesioner yang digunakan harus memiliki perwakilan dari aspek yang ingin diuji dan dapat diolah menjadi nilai statistik. Menurut Mclver dan Carmines (1981) Likert Scale adalah metode yang terdiri dari kumpulan *item* yang memiliki tingkat penilaian terhadap sikap suatu objek yang diberikan pada kelompok subjek dan mereka diminta untuk memberikan pernyataan setuju atau tidak setuju berdasarkan tingkat penilaian. Berdasarkan pernyataan di atas, metode Likert Scale dapat digunakan untuk menyusun sebuah kuesioner yang berbobot. Setiap pernyataan yang digunakan harus dapat mewakili semua tingkat nilai dari pilihan jawaban. Skala likert menggunakan beberapa butir pertanyaan untuk mengukur perilaku individu dengan merespon 5 titik pilihan pada setiap butir pertanyaan, sangat setuju, setuju, tidak memutuskan, tidak setuju, dan sangat tidak setuju (Likert, 1932).

Menurut Laugwitz, Held, dan Schrepp (2008) ada 6 pembagian aspek kategori dalam menyusun pertanyaan kuesioner, seperti *attractiveness*, *perspicuity*, *efficiency*, *dependability*, *stimulation*, dan *novelty*. Aspek *attractiveness* berhubungan dengan daya tarik seseorang terhadap apa yang sedang mereka coba atau rasakan. Aspek *perspicuity* berhubungan dengan kemudahan dalam

pembelajaran atau pemahaman. Aspek *efficiency* berhubungan dengan kecepatan dan organisir. Aspek *dependability* berhubungan dengan hal dapat diandalkan dan keamanan. Aspek *stimulation* berhubungan dengan seberapa besar daya dorong yang dapat dimunculkan dari pengalaman yang dirasakan. Aspek *novelty* berhubungan dengan sesuatu hal yang baru, baik dari segi inovasi atau kreativitas.

2.9 CodeIgniter

CodeIgniter adalah aplikasi *open source* dan merupakan salah satu *framework* dari PHP yang mengusung metode berbasis *Model*, *View*, dan *Controller* (MVC) (Thomas, 2008). *Model* adalah bagian yang berhubungan langsung dengan database dan berfungsi untuk memanipulasi data (*select*, *insert*, *update*, dan *delete*). *View* adalah bagian yang menampilkan hasil atau mempresentasikan visual kepada *user*. *Controller* adalah bagian penghubung yang mangatur segala bentuk *input* dan *output* antara *view* dan *model*.

Beberapa kelebihan yang dimiliki oleh *framework* CodeIgniter (Mado, 2013).

1. Memiliki performa yang sangat cepat.
2. Konfigurasi yang minim.
3. Komunitas pengguna yang sudah berkembang pesat.
4. Dokumentasi yang lengkap.