

BAB 3

LANDASAN TEORI

3.1 Twitter API

Application Programming Interface (API) merupakan fungsi-fungsi/perintah-perintah untuk menggantikan bahasa yang digunakan dalam *system calls* dengan bahasa yang lebih terstruktur dan mudah dimengerti oleh programmer. Fungsi yang dibuat dengan menggunakan API tersebut kemudian akan memanggil *system calls* sesuai dengan sistem operasinya. Tidak tertutup kemungkinan nama dari *system calls* sama dengan nama di API.

Pada awalnya perusahaan *Summize* yang menyediakan fasilitas mencari data di *Twitter*. Kemudian perusahaan *Summize* ini diakuisisi dan diganti merek menjadi *Twitter Search* sehingga *Search API* terpisah sebagai entitas sendiri. *Twitter API* terdiri dari 3 (tiga) bagian yaitu:

a. Search API.

Search API dirancang untuk memudahkan user dalam mengelola *query search* di konten *Twitter*. User dapat menggunakannya untuk mencari *tweet* lebih berdasarkan *keyword* khusus atau mencari *tweet* lebih spesifik berdasarkan *username Twitter*. *Search API* juga menyediakan akses pada data *Trending Topic*.

b. Representational State Transfer (REST) API.

REST API memperbolehkan *developer* untuk mengakses inti dari *Twitter* seperti *timeline*, *status update* dan informasi *user*. *REST API* digunakan dalam membangun sebuah aplikasi *Twitter* yang kompleks yang memerlukan inti dari *Twitter*.

c. Streaming API.

Streaming API digunakan *developer* untuk kebutuhan yang lebih intensif seperti melakukan penelitian dan analisis data. *Streaming API* dapat menghasilkan aplikasi yang dapat mengetahui statistik status *update*, *follower* dan lain sebagainya.

Dalam penelitian ini, bagian Twitter API yang digunakan adalah *REST API*.

3.2 Text Mining

Text mining merupakan penambangan teks atau secara luas didefinisikan sebagai proses pengetahuan intensif dimana pengguna berinteraksi dengan koleksi dokumen dari waktu ke waktu dengan menggunakan seperangkat alat analisis (All Farizi, 2015). Menurut (Zulianto, 2013) *text mining* merupakan penambangan data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen.

Dengan *text mining* tugas-tugas yang berhubungan dengan penganalisaan teks dengan jumlah yang besar, penemuan pola serta penggalian informasi yang mungkin berguna dari suatu teks dapat dilakukan. Dalam *text mining* untuk mendapatkan pola dari suatu teks, sumber-sumber data yang akan diolah adalah dari koleksi dokumen. Dan pola-pola menarik tersebut tidak ditemukan diantara catatan *database* yang sudah diformalisasi melainkan dalam data tekstual yang tidak terstruktur di dalam

koleksi dokumen-dokumen tersebut (Feldman & Sanger, 2007).

Secara garis besar dalam melakukan implementasi *text mining* terdiri dari dua tahap besar yaitu *preprocessing* dan *processing*) (Anggaradana, 2013).

3.2.1 Text Preprocessing

Text preprocessing adalah tahapan untuk mempersiapkan teks menjadi data yang akan mengalami pengolahan pada tahapan berikutnya. Inputan awal pada proses ini adalah berupa dokumen utuh (Mustaghfiri, 2011). *Text preprocessing* pada penelitian ini terdiri dari beberapa tahapan, yaitu: proses *case folding*, proses pemecahan kalimat menjadi kata (*tokenizing*), proses *filtering* kata dengan menghilangkan kata *stopword*, dan proses *stemming*.

a. Case Folding

Case folding adalah tahapan proses mengubah semua huruf dalam teks dokumen menjadi huruf kecil semua, serta menghilangkan karakter selain a-z dan dianggap sebagai delimiter.

b. Pemecahan Kalimat

Pemecahan kalimat yaitu proses memecah string teks dokumen yang panjang menjadi kumpulan kalimat-kalimat. Dalam memecah dokumen menjadi kalimat-kalimat menggunakan fungsi *explode()*, dengan tanda spasi " " sebagai delimiter untuk memotong string dokumen. Dengan menghilangkan tanda-tanda

tersebut dokumen akan terpotong menjadi kata.

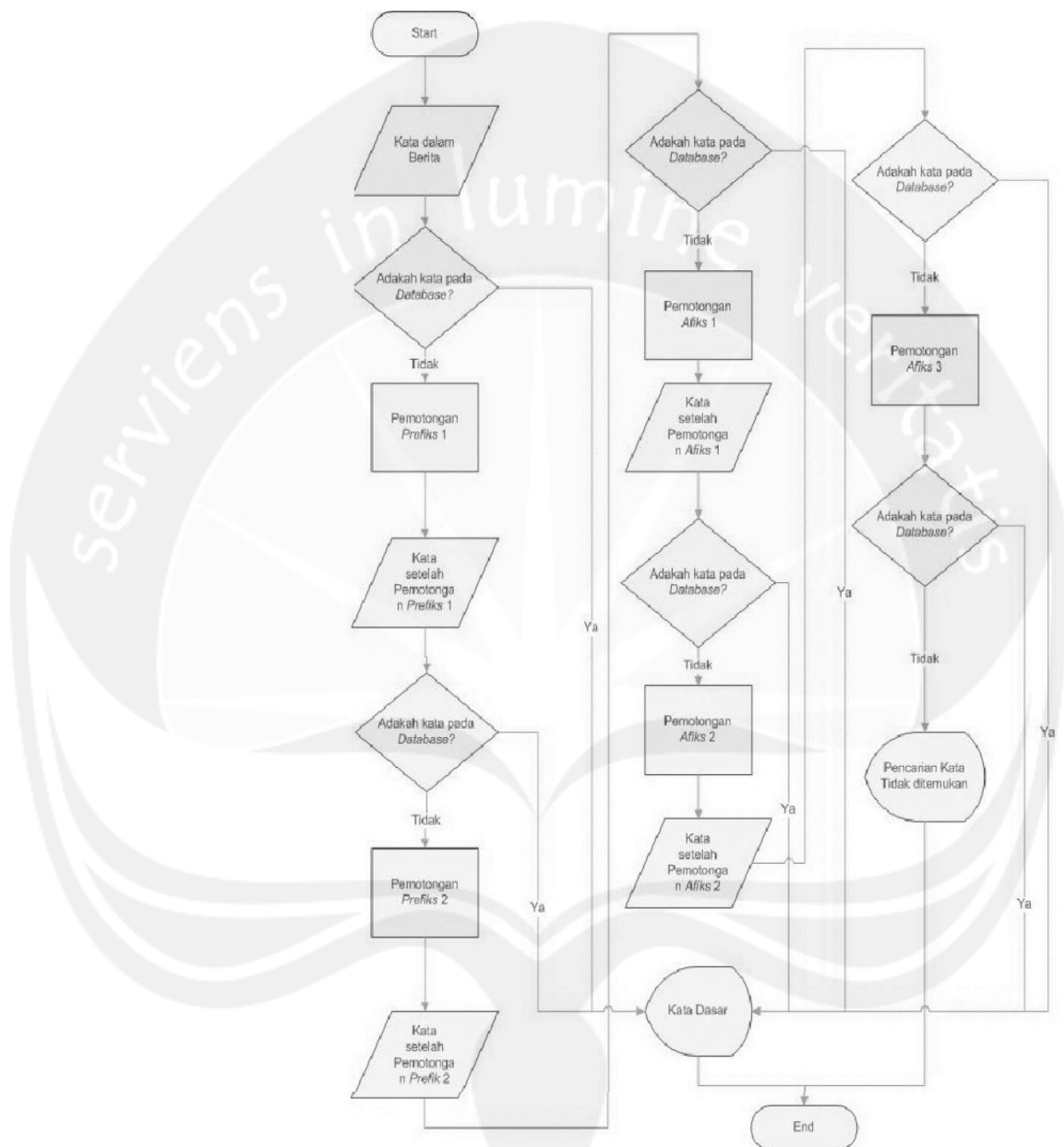
c. *Filtering kata / stopword*

Filtering merupakan proses penghilangan *stopword*. *Stopword* adalah kata-kata yang sering kali muncul dalam dokumen namun artinya tidak deskriptif dan tidak memiliki keterkaitan dengan tema tertentu. Didalam bahasa Indonesia *stopword* adalah kata penghubung dan dapat disebut sebagai kata tidak penting, misalnya "di", "oleh", "pada", "sebuah", "karena" dan lain sebagainya.

d. *Stemming*

Stemming adalah proses mencari akar (*root*) kata dari tiap token kata yaitu dengan pengembalian suatu kata berimbuhan ke bentuk dasarnya (*stem*). Beberapa algoritma yang telah dikembangkan untuk proses *stemming* diantaranya Algoritma Porter (Bahasa Indonesia dan Inggris) dan Algoritma Nazief & Adriani untuk teks berbahasa Indonesia. Dalam hal ini, penelitian yang dilakukan (Agusta, 2009) menunjukkan algoritma Nazief & Adriani memiliki tingkat akurasi yang lebih tinggi dalam proses *stemming* untuk Bahasa Indonesia dibandingkan algoritma Porter. Dalam penelitian ini penulis menggunakan algoritma Nazief & Adriani untuk proses

stemming. Flow chart algoritma Nazief & Adriani dapat dilihat pada gambar 3.1.



Gambar 3.1 Flow Chart Algoritma Nazief & Adriani (All Farizi, 2015).

3.2.2 Processing

Tahap *processing* adalah tahap terpenting dari seluruh proses *text mining*. Tahap ini berusaha menemukan pola atau pengetahuan dari keseluruhan teks. Teknik yang di gunakan pada tahap ini adalah dengan melakukan pembobotan (*weighting*) terhadap *term* dari hasil tahap *preprocessing*. Setiap *term* di berikan bobot sesuai dengan skema pembobotan yang di pilih, baik itu pembobotan lokal, global atau kombinasi keduanya. Banyak aplikasi menerapkan pembobotan kombinasi berupa perkalian bobot lokal *term frequency* dan global *inverse document frequency* yang ditulis dengan TFIDF.

3.3 TF-IDF

TF-IDF (*Term Frequency Inverse Document Frequency*) merupakan metode yang digunakan untuk menentukan nilai frekuensi sebuah kata di dalam sebuah dokumen atau artikel dan juga frekuensi di dalam banyak dokumen. Perhitungan ini menentukan seberapa relevan sebuah kata di dalam sebuah dokumen (Evan, 2014). TFIDF adalah sebuah algoritma yang umumnya digunakan untuk pengolahan data besar (Kamath, 2014).

Algoritma *TF-IDF* melakukan pemberian bobot pada setiap kata kunci disetiap kategori untuk mencari kemiripan kata kunci dengan kategori yang tersedia. Sebelum melakukan pembobotan maka akan dilakukan lima tahap pencarian *text preprocessing* yaitu pemecahan kalimat, *case folding*, *tokenizing*, *filtering*, dan *stemming*, lalu selanjutnya dilakukan proses menghitung

bobot *TF-IDF*, bobot *query relevance* dan bobot *similarity* (Marlinda & Rianto, 2013).

Berdasarkan penelitian-penelitian sebelumnya, yang membahas tentang penerapan metode *TF-IDF*. Penulis menemukan banyak terdapat variasi formula dalam mengimplementasikan metode *TF-IDF* pada pembobotan kata. Nilai *TF-IDF* meningkat secara proporsional berdasarkan jumlah atau banyaknya kata yang muncul pada dokumen, tetapi diimbangi dengan frekuensi kata dalam korpus. Variasi dari skema pembobotan *TF-IDF* sering digunakan oleh mesin pencari sebagai alat utama dalam mencetak nilai (*scoring*) dan peringkat (*ranking*) sebuah relevansi dokumen yang diberikan *user*.

TF-IDF pada dasarnya merupakan hasil dari perhitungan antara *TF* (*Term Frequency*) dan *IDF* (*Inverse Document Frequency*). Banyak cara untuk menentukan nilai yang tepat dari kedua statistik yang ada. Dalam kasus *term frequency* $tf(t, d)$, cara yang paling sederhana adalah dengan menggunakan *raw frequency* di dalam dokumen, yaitu berapa kali *term* t muncul di dokumen d . Jika menyatakan *raw frequency* t sebagai $f(t, d)$, maka skema tf yang sederhana adalah $tf(t, d) = f(t, d)$.

Nilai *idf* sebuah *term* (kata) dapat dihitung menggunakan persamaan sebagai berikut:

$$IDF = \log_{10}\left(\frac{D}{dfi}\right)$$

D adalah jumlah dokumen yang berisi *term* (t) dan dfi adalah jumlah kemunculan (frekuensi) kata terhadap D . Adapun algoritma yang digunakan untuk menghitung bobot (W) masingmasing dokumen terhadap kata kunci (*query*), yaitu :

$$W_{d,t} = tf_{d,t} * IDF_t$$

Keterangan :

d = dokumen ke-d

t = kata ke-t dari kata kunci

W = bobot dokumen ke-d terhadap kata ke-t

tf = term frekuensi/frekuensi kata

Setelah bobot (W) masing-masing dokumen diketahui, maka dilakukan proses pengurutan (*sorting*) dimana semakin besar nilai W, semakin besar tingkat kesamaan (*similarity*) dokumen tersebut terhadap kata yang dicari, demikian pula sebaliknya.

3.4 **Framework CodeIgniter**

Framework adalah sekumpulan perintah atau fungsi dasar yang dapat membantu menyelesaikan proses-proses yang lebih kompleks. Sedangkan *CodeIgniter* merupakan salah satu *open source framework* yang digunakan oleh script pemrograman web PHP (*PHP Hypertext Preprocessor*) dalam mengembangkan aplikasi web dinamis dengan dasar kerja CRUD (*Create, Read, Update, Delete*). Metode yang digunakan oleh *framework CodeIgniter* disebut *Model - View - Controller* atau yang disingkat dengan sebutan MVC.

MVC memisahkan antara logika pemrograman dengan presentasi. Hal ini dapat terlihat dari adanya minimalisir *script* presentasi (*HTML, CSS, JavaScript*, dan sebagainya) yang dipisahkan dari *PHP script*. Didalam folder *CodeIgniter*, MVC dapat kita temukan dalam folder *application*. *CodeIgniter* juga menjadi salah satu *framework* pilihan yang memungkinkan developer untuk

membuat sebuah aplikasi web dengan karakter pengembangan RAD (*Rapid Application Development*), yang memungkinkan untuk digunakan dan dikembangkan menjadi aplikasi lain yang lebih kompleks.

CodeIgniter terdiri dari file-file pustaka (*library*), kelas-kelas, dan infrastruktur *run-time* yang terinspirasi oleh *framework Ruby on Rails*. *CodeIgniter* juga banyak digunakan oleh para programmer yang memilih untuk bekerja dengan struktur yang rapi dan padat tanpa kehilangan fleksibilitas pengembangan *framework* (Fajriyah, 2011).