Tugas Pengenalan Pola - Preprocessing
Baskara 16/398499/PA/17460

1. Algoritma Nazief Andriani

Algoritma Nazief & Adriani adalah salah satu algoritma yang digunakan dalam tahap stemming.

Algortima Nazief & Adriani merupakan algoritma untuk mengubah kata yang memiliki sufiks, prefix dan atau konfiks menjadi bentuk kata dasar. Algoritma Nazief & Adriani digunakan dalam penelitian ini sebagai algoritma pendukung dalam proses penentuan nilai kemiripan pada dokumen teks. Terdapat beberapa metode dalam menentukan kemiripan antar dua objek. Algoritma Nazief & Adriani dapat mengenal kesamaan makna dari setiap kata, dimana kata yang memiliki sufiks, prefiks dan konfiks diubah menjadi bentuk dasar.

Algoritma stemming untuk bahasa yang satu berbeda dengan algoritma stemming untuk bahasa lainnya. Proses stemming pada teks berbahasa Indonesia lebih rumit/kompleks karena terdapat variasi imbuhan yang harus dibuang untuk mendapatkan root word (kata dasar) dari sebuah kata. Pada umumnya kata dasar pada bahasa Indonesia terdiri dari kombinasi:

Prefiks 1 + Prefiks 2 + Kata dasar + Sufiks 3 + Sufiks 2 + Sufiks 1

Konjungsi adalah Algoritma stemming Nazief dan Adriani dikembangkan berdasarkan aturan morfologi Bahasa Indonesia yang mengelompokkan imbuhan menjadi awalan (prefix), sisipan (infix), akhiran (suffix) dan gabungan awalan akhiran (confixes). Algoritma ini menggunakan kamus kata dasar dan mendukung recoding, yakni penyusunan kembali kata-kata yang mengalami proses stemming berlebih. Aturan morfologi Bahasa Indonesia mengelompokkan imbuhan ke dalam beberapa kategori sebagai berikut:

1. Inflection suffixes yakni kelompok akhiran yang tidak merubah bentuk kata dasar. Sebagai contoh, kata “duduk” yang diberikan akhiran “-lah” akan menjadi “duduklah”. Kelompok ini dapat dibagi menjadi dua:
    a. Particle (P) atau partikel yakni termaksud di dalamnya “-lah”, “kah”, “tah” dan “pun”
    b. Possessive pronoun (PP) atau kata genti kepunyaan, termaksud di dalamnya “-ku”,”-mu” dan “-nya”.
2. Derivation suffixes (DS) yakni kumpulan akhiran asli Bahasa Indonesia yang secara langsung ditambahkan pada kata dasar yaitu akhiran “-i”, “-kan”, dan “-an”.
3. Derivation prefixes (DP) yakni kumpulan awalan yang dapat langsung diberikan pada kata dasar murni, atau pada kata dasar yang sudah mendapatkan penambahan sampai dengan 2 awalan. Termaksud di dalamnya adalah:
    a. Awalan yang dapat bermorfologi (“me-“, “be-“, “pe-“ dan “te”).
    b. Awalan yang tidak bermorfologi (“di-“, “ke-“ dan “se-“).

Tahapan Algoritma Nazief Andriani:

1. Cari kata dalam kamus jika ditemukan maka diasumsikan bahwa kata tersebut adalah kata dasar. Algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 2.
2. Hilangkan inflectional suffixes bila ada. Dimulai dari inflectional particle (“-lah”, “-kah”, “-tah” dan“-pun”), kemudian possessive pronoun (“-ku”, “-mu” dan “-nya”). Cari kata pada kamus jika ditemukan algoritma berhenti, jika kata tidak tidak ditemukan dalam kamus lakukan langkah 3.
3. Hilangkan derivation suffixes (“-an”, “-i” dan “-kan”). Jika akhiran “-an” dihapus dan ditemukan akhiran “-k”, maka akhiran “-k” dihapus. Cari kata pada kamus jika ditemukan algoritma berhenti, jika kata tidak tidak ditemukan maka lakukan langkah 4.
4. Pada langkah 4 terdapat tiga iterasi.
    a. Iterasi berhenti jika :
        i. Ditemukannya kombinasi awalan yang tidak diizikan berdasarkan awalan
        ii. Awalan yang dideteksi saat ini sama dengan awalan yang dihilangkan sebelumnya.
        iii. Tiga awalan telah dihilangkan.
    b. Identifikasikasikan tipe awalan dan hilangkan. Awalan terdiri dari dua tipe:
        i. Standar (“di-“, “ke-“, “se-”) yang dapat langsung dihilangkan dari kata.
        ii. Kompleks (“me-“, “be-“, “pe-“, “te”) adalah tipe-tipe awalan yang dapat bermorfologi sesuai kata dasar yang mengikutinya. Oleh karena itu dibutuhkan aturan pada tabel 2.4 untuk mendapatkan hasil pemenggalan yang tepat.
        iii. Cari kata yang telah dihilangkan awalannya. Apabila tidak ditemukan, maka langkah 4 diulang kembali. Apabila ditemukan, maka algoritma berhenti.
5. Apabila setelah langkah 4 kata dasar masih belum ditemukan, maka proses recording dilakukan dengan mengacu pada aturan tabel 2.4. Recording dilakukan dengan menambahkan karakter recording di awal kata yang dipenggal. Pada tabel 2.4, karakter recording adalah huruf kecil setelah tanda hubung (‘-‘) dan terkadang berada sebelum tanda kurung. Sebagai contoh, kata “menangkap” (aturan 15) pada tabel 2.4, setelah dipenggal menjadi “nangkap”. Karena tidak valid, maka recording dilakukan dan menghasilkan kata “tangkap”.
6. Jika semua langkah gagal, maka input kata yang diuji pada algoritma ini di anggap sebagai kata dasar.

Contoh:

- Se + bungkus = sebungkus
- Me + inap = menginap
- Me + satu = menyatu

2. Jaro-Winkler Algorithm

Jaro-Winkler distance adalah suatu ukuran yang dapat digunakan untuk mengukur jarak antara dua buah string.

Jaro distance antara dua buah kata merupakan jumlah minimum transposisi untuk tiap karakter yang dibutuhkan untuk mengubah suatu kata menjadi kata yang lain.

Jaro Distance antara string S1 dan S2 didefinisikan sebagai berikut:

$$d_j = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)$$

dj : Jarro Distance
m : Jumlah karakter yang sama
t : setengah jumlah transposisi
|s1| : panjang string S1
|s2| : panjang string S2

Jaro-Winkler distance menggunakan skala prefix p yang memberi rating skor yang lebih kepada string yang cocok dari awal untuk suatu set prefix dengan panjang l. Berikut definisi dari Jaro-Winkler distance:

$$d_w = d_j + (\ell p(1 - d_j))$$

P adalah suatu faktor skala konstan untuk jumlah skor pada sting yang memiliki prefix yang sama. Nilai standar yang digunakan adalah p=0.1.

L adalah panjang dari prefx yang sama pada bagian awal string.

Contoh:

“martha” dan “marhta”.

m = 6, t = 2/2 =1, |s1| = 6, |s2| = 6

dj = (⅓) (6/6 + 6/6 + (6–1)/6) = ⅓ 17/6 = 0,944
Jaro distance = 94,4%

l = 3 (“mar”)

dw = 0,944 + ((0,1*3)(1–0,944)) = 0,944 + 0,3*0,056 = 0,961
Jaro-Winkler distance = 96,1%

3. Implementasi Preprocessing Terhadap Data Twitter
    a. Stemming
       Stemming dilakukan menggunakan library python Sastrawi, dengan algoritma Nazief-Andriani.
    b. Tokenization
       Tokenization dilakukan dengan menggunakan fungsi word_tokenization() dari library nltk. word_tokenization() berfungsi untuk mengubah string menjadi sebuah list.
    c. Case Folding
       Case Folding dilakukan dengan menggunakan fungsi lower() dari library nltk. lower() berfungsi untuk mengubah seluruh karakter pada suatu string menjadi lowercase.

```
In [1]: import pandas as pd
        import nltk
        from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

In [2]: factory = StemmerFactory()
        stemmer = factory.create_stemmer()

In [3]: df = pd.read_csv("raw_tweet.csv")

In [4]: for index, row in df.iterrows():
            # Stemming
            stemmed = stemmer.stem(row[1])
            #Tokenization
            tokens = nltk.word_tokenize(row[1])
            #Case Folding
            words = [w.lower() for w in tokens]
            row[1] = words

In [5]: df.to_csv("preprocessed_tweet.csv", encoding='utf-8', index=False)

In [ ]:
```