Pemanfaatan Approximate String Matching Menggunakan Jaro-Winkler Distance untuk Mengidentifikasi Typographical Error pada Keyword-Based Chatbot

Christian Wibisono / 13516147
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516147@std.stei.itb.ac.id
christian.wibisono7@gmail.com

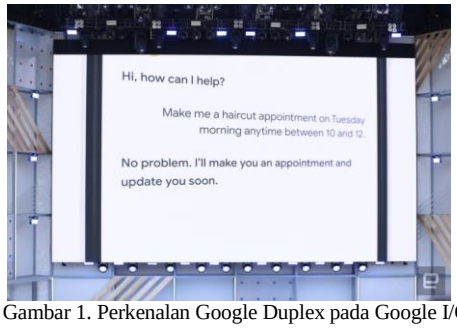
***Abstract*—Kesalahan pengetikan (Typographical Error) dapat mengakibatkan kata kunci yang menjadi dasar perintah Keyword-Based Chatbot menjadi tidak dikenali. Hal tersebut dapat ditangani dengan mengembangkan sistem untuk mengidentifikasi kesalahan pegetikan. Metode *approximate string matching* merupakan salah satu metode yang bisa diterapkan dengan memanfaatkan jarak Jaro-Winkler untuk mengatasi permasalahan ini untuk membuat chatbot yang lebih 'cerdas' dan adaptif untuk memahami perintah yang disampaikan oleh pengguna. Makalah ini membahas tentang analisis algoritma Jaro-winkler untuk mengidentifikasi *typographical error* saat melakukan percakapan dengan *keyword-based* chatbot.**

Keywords—approximate string matching, typographical error, jaro-winkler,chatbot

I. PENDAHULUAN

Alan Turing, matematikawan asal Inggris pada tahun 1950 pernah mengajukan sebuah pertanyaan "Apakah mesin dapat bepikir?" (Turing, 2008). Sejak munculnya pertanyaan itu, banyak ilmuwan komputer dunia berlomba-lomba untuk memecahkan persoalan tersebut. Semenjak itulah bidang keilmuan Inteligensia Buatan mulai berkembang. Salah satu penerapan paling sederhana dari Inteligensia buatan adalah Chatbot atau biasa disebut juga chatter robot. Chatbot adalah teknologi yang membuat interaksi antara manusia dan komputer dapat menjadi sealamiah mungkin Pengukuran yang biasa dilakukan untuk menguji kealamiahan interaksi antara sebuah chatbot dengan manusia adalah dengan melakukan Turing Test. Chatbot pertama di dunia yang dikenal bernama ELIZA, dikembangkan oleh Weizenbaum, seorang professor asal MIT (Massachussets Institute of Technology) pada tahun 1966. ELIZA ini menjadi inspirasi bagi ilmuwan komputer di dunia lain untuk terus mengembangkan Chatbot hingga pada akhirnya muncul terobosan yaitu ALICE (Artificial Linguistic Internet Computer Entity) yang berhasil memenangkan Loebner's *annual instantiation of Turing's Test for machine intelligence.*

Pada tahun 2018, chatbot diproyeksikan menjadi teknologi paling prospektif untuk berbagai keperluan bisnis. Perkembangan AI didukung kemajuan teknologi Natural Language Processing menjadikan chatbot semakin populer dan menjadi opsi logis bagi industri untuk mengembangkan produk yang mereka miliki. Ditambah lagi banyaknya open messaging API yang mendukung para pengembang untuk mengembangkan chatbot mereka sendiri secara mudah. Berita terhangat di topik Chatbot adalah diperkenalkannya chatbot buatan Google yaitu Duplex dalam ajang Google I/O di Amerika Serikat pekan lalu. Google Duplex adalah teknologi baru kecerdasan buatan yang mampu meniru suara dan percakapan manusia untuk melakukan berbagai hal lewat telepon genggam yang dimiliki pengguna.



Gambar 1. Perkenalan Google Duplex pada Google I/O

Dalam proses pengembangan chatbot sederhana, salah satu permasalahan yang paling sering ditemui adalah *typographical error*. Galat ini menyebabkan *keyword-based* chatbot tidak dapat menginterpretasikan teks perintah yang diberikan pengguna karena kata yang salah dalam penulisan tersebut tidak terdapat pada basis data chatbot. Kasus ini sesungguhnya menurunkan tingkat *user experience* dari chatbot ini sendiri karena *nature* dari manusia yang juga sering melakukan kesalahan pengetikan.

Dalam makalah ini penulis akan menganalisis penerapan algoritma Jaro-Winkler untuk mengidentifikasi *typographical error* pada *keyword-based* chatbot sederhana. Proses pemahaman chatbot sederhana terhadap *typographical error* dapat dikembangkan dengan pendekatan *string metrics* Jaro-Winkler. String metrics dapat digunakan untuk mengetahui

seberapa besar kesamaan suatu teks perintah dengan basis data teks yang dimiliki oleh sebuah chatbor, Dengan demikian, chatbot dapat mengidentifikasi kata terdekat kemudian memberikan respon yang tepat walaupun terjadi kesalahan penulisan *(typo).* Diharapkan dengan pengembangan metode ini, chatbot sederhana yang dibangun dapat menjadi lebih responsif dan adaptif dalam menginterpretasikan teks perintah dari pengguna.

II. DASAR TEORI

A. *Keyword-Based Chatbot*

Chatbot adalah sebuah program komputer yang dirancang untuk menyimulasikan sebuah percakapan atau komunikasi yang interaktif kepada pengguna (manusia) melalui bentuk teks, suara, dan atau visual. Percakapan yang terjadi antara komputer dengan manusia merupakan bentuk respon dari program yang telah dideklarasikan pada database program pada komputer. Kemampuan komputer dalam menyimpan banyaknya data tanpa melupakan satu pun informasi yang disimpannya digabungkan dengan kepraktisan bertanya pada sumber informasi langsung dibandingkan dengan mencari informasi sendiri serta kemampuan learning yang dimilikinya menyebabkan chatbot adalah customer service yang handal.

Respon yang dihasilkan merupakan hasil pemindaian kata kunci pada inputan pengguna dan menghasilkan respon balasan yang dianggap paling cocok, atau pola kata-kata yang dianggap paling mendekati dan pada umumnya menggunakan pendekatan Natural Language Processing (NLP). Natural Language Processing merupakan salah satu tujuan jangka panjang dari Artificial Intelligence (kecerdasan buatan) yaitu pembuatan program yang memiliki kemampuan untuk memahami bahasa manusia. Pada prinsipnya bahasa alami adalah suatu bentuk representasi dari suatu pesan yang ingin dikomunikasikan antar manusia. Bentuk utama representasinya adalah berupa suara/ucapan,tetapi sering pula dinyatakan dalam bentuk tulisan.

Keyword-based chatbot adalah chatbot sederhana dimana pengguna mengetikkan sebuah kata kunci atau frasa dan bot akan melakukan *string matching* dengan respons kata kunci yang tersedia dalam basis datanya. Kelemahan dari bot ini adalah bahwa string yang diinputkan pengguna harus tepat sama karena akan dilakukan *exact matching*. Bot ini tidak dapat mendeteksi *misspelled words* atau *slang*. Keuntungannya adalah bot ini tidak akan memberikan respon yang tidak pernah diinputkan secara manual oleh pembuat.

B. *String Matching*

Pencocokan *string* (*string matching*) secara garis besar dapat dibedakan menjadi dua (Sagita & Prasetiyowati, 2012) yaitu: (1) *Exact string matching*, merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Misalnya, kata obat akan menunjukkan kecocokan hanya dengan kata obat. (2) *Inexact string matching* atau *fuzzy string matching*, merupakan pencocokan *string* secara samar yaitu pencocokan *string* dimana string yang dicocokkan memiliki kemiripan namun keduanya memiliki susunan karakter yang berbeda (mungkin jumlah atau urutannya) tetapi *string* tersebut memiliki kemiripan baik kemiripan tekstual/penulisan atau kemiripan ucapan. Pencocokan *string* berdasarkan kemiripan tekstual/penulisan meliputi jumlah karakter, susunan karakter dalam dokumen disebut sebagai *approximate string matching* sedangkan pencocokan *string* berdasarkan kemiripan ucapan dari segi pengucapan disebut sebagai *phonetic string matching*. Berikut ini contoh dari *approximate string matching* dan *phonetic string matching*: (a) panitea dengan panitia, memiliki jumlah karakter yang sama tetapi ada karakter yang berbeda. Jika perbedaan karakter ini dapat ditoleransi sebagai sebuah kesalahan penulisan maka dua *string* tersebut dikatakan cocok. (b) obat dengan obad dari tulisan berbeda tetapi dalam pengucapannya mirip sehingga dua *string* tersebut dianggap cocok. Contoh yang lain adalah obat, dengan obbat, obaat, obatt, obate.

C. *String Metric*

String metric atau *similarity metric* adalah kelas matriks berbasis tekstual yang dapat menghasilkan nilai kesamaaan atau ketidaksamaan dari dua teks string untuk proses perbandingan dan penyamaan. String metric biasanya digunakan dalam deteksi kecurangan, analisa fingerprint, deteksi plagiarisme, ontology merging, analisis DNA , analisis RNA, analisis image, database deduplication, data mining, Web interfaces, dan sebagainya. Beberapa algoritma yang berdasarkan kepada string metric diantaranya adalah Levenshtein *distance*, TF/IDF, Needleman-Wunsch *distance*, Jaro-Winkler *distance*, dan sebagainya. Dari algoritma yang telah disebutkan di atas Jaro-Winkler *distance* memiliki ketepatan yang baik di dalam pencocokan string yang relafif pendek. Untuk dapat mendukung kinerja dari algoritma Jaro-Winkler *distance* maka dilakukan proses tokenizing terlebih dahulu terhadap dokumen-dokumen yang akan digunakan

D. *Jaro Winkler Distance*

Jaro-Winkler *distance* adalah merupakan varian dari Jaro *distance* metrik yaitu sebuah algoritma untuk mengukur kesamaan antara dua string, biasanya algoritma ini digunakan di dalam pendeteksian duplikat. Semakin tinggi Jaro-Winkler *distance* untuk dua string, semakin mirip dengan string tersebut. Jaro-Winkler *distance* terbaik dan cocok untuk digunakan dalam perbandingan string singkat seperti nama orang. Skor normalnya seperti 0 menandakan tidak ada kesamaan, dan 1 adalah sama persis. Algoritma Jaro-Winkler *distance* memiliki kompleksitas waktu quadratic runtime complexity yang sangat efektif pada string pendek dan dapat bekerja lebih cepat dari algoritma edit *distance*. Dasar dari algoritma ini memiliki tiga bagian:

1. Menghitung panjang string,
2. Menemukan jumlah karakter yang sama di dalam dua string
3. Menemukan jumlah transposisi

.Pada algoritma Jaro digunakan rumus untuk menghitung jarak (dj) antara dua string yaitu s1 dan s2 adalah

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

dimana :
m = jumlah karakter yang sama persis
$|s_1|$ = panjang string 1
$|s_2|$ = panjang String 2
t = jumlah transposisi

Jarak teoritis dua buah karakter yang disamakan dapat dibenarkan jika tidak melebihi:

$$\left(\frac{max\left(|s_1|, |s_2|\right)}{s} \right) < -1 \quad (2)$$

Akan tetapi bila mengacu kepada nilai yang akan dihasilkan oleh algoritma Jaro-Winkler maka nilai jarak maksimalnya adalah 1 yang menandakan kesamaan string yang dibandingkan mencapai seratus persen atau sama persis. Biasanya s1 digunakan sebagai acuan untuk urutan di dalam mencari transposisi. Yang dimaksud transposisi di sini adalah karakter yang sama dari string yang dibandingkan akan tetapi tertukar urutannya.. Sebagai contoh, dalam membandingkan kata CRATE dengan TRACE, bila dilihat seksama maka dapat dikatakan semua karakter yang ada di s1 ada dan sama dengan karakter yang ada di s2 tetapi dengan urutan yang berbeda. Dengan mengganti C dan T, dapat dilihat perubahan kata CRATE menjadi TRACE. Pertukaran dua elemen string inilah adalah contoh nyata dari transposisi yang dijelaskan. Dalam pencocokkan DwAyNE dan DuANE memiliki urutan yang sama D-A-N-E, jadi tidak ada transposisi.. Jaro-Winkler *distance* menggunakan prefix scale (p) yang memberikan tingkat penilaian yang lebih, dan prefix length (l) yang menyatakan panjang awalan yaitu panjang karakter yang sama dari string yang dibandingkan sampai ditemukannya ketidaksamaan. Bila string s1 dan s2 yang diperbandingkan, maka Jaro-Winkler *distance*nya (dw) adalah:

$$dw = dj + (lp\ (1 - dj))$$

dimana :
dj = Jaro *distance* untuk strings s1 dan s2
l = panjang prefiks umum di awal string nilai maksimalnya 4 karakter(panjang karakter yg sama sebelum ditemukan ketidaksamaan max 4)
p = konstanta scaling factor. Nilai standar untuk konstanta ini menurut Winkler adalah p = 0,1.

E. Typographical Error

Typographical error merupakan kesalahan yang terjadi pada saat proses mengetik teks dan dapat mengubah arti dari suatu kata bahkan arti dari suatu kalimat. Istilah ini mencakup kesalahan karena kegagalan mekanis atau slip tangan atau jari, dan juga timbul akibat ketidaktahuan penulis seperti kesalahan ejaan. Typographical *error* dapat disebabkan oleh, misalnya, jari menekan dua tombol keyboard yang berdekatan secara bersamaan. Typographical *error* ini bervariasi mulai dari kesalahan ketik biasa sampai kesalahan dalam tatanan bahasa yang digunakan atau bahkan pengertian dari kata tersebut. Kesalahan-kesalahan tersebut dikategorikan ke dalam 2 jenis yaitu non-word *error* dan real-word *error*. Non-word *error* adalah *error* yang tidak terdapat makna didalamnya sedangkan pada real-word *error*, kata yang tertulis bernilai benar atau bisa disebut mempunyai arti dalam kamus namun tidak dimaksudkan dalam kalimat tersebut maupun mempunyai arti yang berbeda dan bahkan kalimat tersebut memiliki tata bahasa yang salah (Naradhipa et al., 2011). Terdapat beberapa jenis *typographical error* yaitu insertion, deletion serta substitution. Dalam mendeteksi *typographical error* pada teks dibutuhkan suatu aplikasi disebut spelling checker. Spellin checker melakukan proses pengecekan terhadap pengejaan kata-kata untuk mendeteksi adanya kata yang mengalami kesalahan ejaan dan juga memberikan suggestion berupa kata-kata kandidat (Soleh dan Purwarianti, 2011).

III. Penerapan Algoritma Jaro-Winkler Distance Pada Chatbot

Pada bagian ini, penulis akan membahas mengenai implementasi Jaro-Winkler Distance untuk mengidentifikasi *typographical error* pada chatbot dimulai dari mendefinisikan persoalan, membuat kode program, dan menentukan solusi dari permasalahan ini.

A. Definisi Persoalan

Diberikan dua buah string A dan B, kita harus menentukan indeks kemiripan (*similarity*) dari kedua string tersebut. Indeks kemiripan dan Jarak Levenshtein dapat dihitung menggunakan persamaan 1 dan 2 dari bagian II. Diberikan juga fungsi pembatas yang digunakan untuk kasus *typographical error* adalah 0.8 dan minimal memiliki nilai m = 2 artinya hanya dua string yang memiliki nilai jarak Jaro-Winkler yang lebih dari/sama dengan 0,8 yang diklasifikasikan sebagai *typo*.

B. Perhitungan Algoritma Jaro-Winkler

Kasus 1

Diberikan string s_1 MARTHA dan s_2 MARHTA maka:

s1	M	A	R	**T**	**H**	A
matches	0	1	2	4	3	5
s2	M	A	R	**H**	**T**	A

m = 6
s_1 = 6
s_2 = 6
Dapat kita lihat bahwa pada indeks ke 3 dan ke -4 karakter T dan H tertukar. Hal ini menunjukkan bahwa terjadi 2 half transposition. Oleh sebab itu,

t = 1.
Maka nilai Jaro *distance* adalah:

$$d_j = \frac{1}{3} \times \left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6} \right) = 0.944$$

Kemudian bila diperhatikan susunan s1 dan s2 dapat diketahui nilai l = 3, dan dengan nilai konstan p = 0.1. Maka nilai Jaro-Winkler *distance* adalah:

$$dw = 0.944 + (3 \times 0.1\ (1 - 0.944)) = 0.961$$

Kasus 2
Jika string s1 DWAYNE dan s2 DUANE maka:
m = 4
s1 = 6
s2 = 5
t = 0, hal ini dikarenakan tidak ada karakter yang sama tapi tertukar urutannya. Karakter seperti D, A, N, E dianggap dalam urutan yang sama. Maka nilai Jaro *distance* adalah:

$$d_j = \frac{1}{3} \times \left(\frac{4}{6} + \frac{6}{5} + \frac{4-1}{4} \right) = 0.822$$

Kemudian bila diperhatikan susunan s1 dan s2 dapat diketahui nilai *l =1*, dan dengan nilai konstan *p= 0.1*. Maka nilai Jaro-Winkler *distance* adalah:

$$dw = 0.822 + (1 \times 0.1\ (1 - 0.8.22)) = 0.961$$

Kasus 3
Diberikan string yang memiliki panjang yang berbeda yaitu, s1 adalah JONES dan s2 adalah JOHNSON

s1	J	O	**N**	**E**	S		
Matches	0	1	3	-	5		
s2	J	O	**H**	N	S	O	N

m = 4
s1 = 5
s2 = 7
Dari kata JONES dan JOHNSON diketahui bahwa tidak terjadi transposisi maka nilai t = 0

$$dj = 1/3 * (4/5 + 4/7 + (4 - 0) /4) = 0.790$$

String JONES dan JOHNSON hanya cocok pada 2 karakter pertamanya yaitu JO, sehingga jaro Winkler distancenya adalah:

$$dw = 0.790 + 0.1 \text{ x } 2 \text{ x } (1.0 - 0.790) = 0.832$$

C. *Implementasi Algoritma Jaro-Winkler*

Penulis mengimplementasikan algoritma Jaro-Winkler menggunakan bahasa pemrograman Python 3. Kompleksitas waktu algoritma Jaro-Winkler yang merupakan salah satu algoritma *Edit Distance* adalah **O(mn)**. Implementasi lengkap kode bisa dilihat pada kode sumber berikut ini

```
def jaro_winkler(first, second, winkler=True,
winkler ajustment=True, scaling=0.1):
    print('Calculating similarity from {} and {}
:'.format(first,second))
    print('Scaling factor: {}'.format(scaling))

    jaro = calculate score(first, second)
    cl = min(len(get prefix(first, second)), 4)

    if all([winkler, winkler ajustment]):
        return round((jaro + (scaling * cl * (1.0 -
jaro))) * 100.0) / 100.0

    return jaro

def match char(first, second):
    common = []
    limit = math.floor(min(len(first), len(second))
/ 2)

    for i, l in enumerate(first):
        left, right = int(max(0, i - limit)),
int(min(i + limit + 1, len(second)))
        if l in second[left:right]:
            common.append(l)
            second = second[0:second.index(l)] + '*'
+ second[second.index(l) + 1:]

    return ''.join(common)


def transposition(first, second):
    return math.floor(len([(f, s) for f, s in
zip(first, second) if not f == s]) / 2.0)

def calculate_score(first, second):
    shorter, longer = first.lower(), second.lower()

    if len(first) > len(second):
        longer, shorter = shorter, longer

    m1 = match char(shorter, longer)
    m2 = match char(longer, shorter)

    if len(m1) == 0 or len(m2) == 0:
        return 0.0

    return (float(len(m1)) / len(shorter) +
            float(len(m2)) / len(longer) +
            float(len(m1) - transposition(m1, m2)) /
len(m1)) / 3.0


def get_index(first, second):
    if first == second:
        return -1

    if not first or not second:
        return 0

    max len = min(len(first), len(second))
    for i in range(0, max_len):
        if not first[i] == second[i]:
            return i

    return max_len
```

```
def get_prefix(first, second):
    if not first or not second:
        return ""

    index = get index(first, second)
    if index == -1:
        return first

    elif index == 0:
        return ""

    else:
        return first[0:index]
```

D. Implementasi Pada Chatbot

Chatbot yang akan dibuat oleh penulis dibangun diatas platform LINE Messaging API memanfaatkan program Jaro-Winkler yang telah dibuat penulis. Tujuan utama dari implementasi ini adalah agar respon yang diberikan oleh chatbot bisa tetap menginterpretasikan perintah yang sebenarnya merupakan intensi dari pengguna. Chatbot akan mengeksekusi perintah yang memiliki tingkat kemiripan tertinggi dengan batas jarak jaro winkler 0.8.

```
def identify_error(words):
    for word in words:
        score =
jaro winkler(word[0],word[1],scaling=0.1)
        print('Score : {}'.format(score))
        if score < 0.8 :
            print('Verdict: Not Similar\n')
        else :
            print('Verdict: Similar\n')
```

IV. Hasil Pengujian Dan Analisis

Untuk melakukan pengujian, penulis terlebih dahulu melakukan pengujian beberapa kata kunci yang telah terdefinisi pada chatbot pada program jaro-winkler.
Adapun input 7 kata kunci yang akan dibandingkan adalah sebagai berikut:

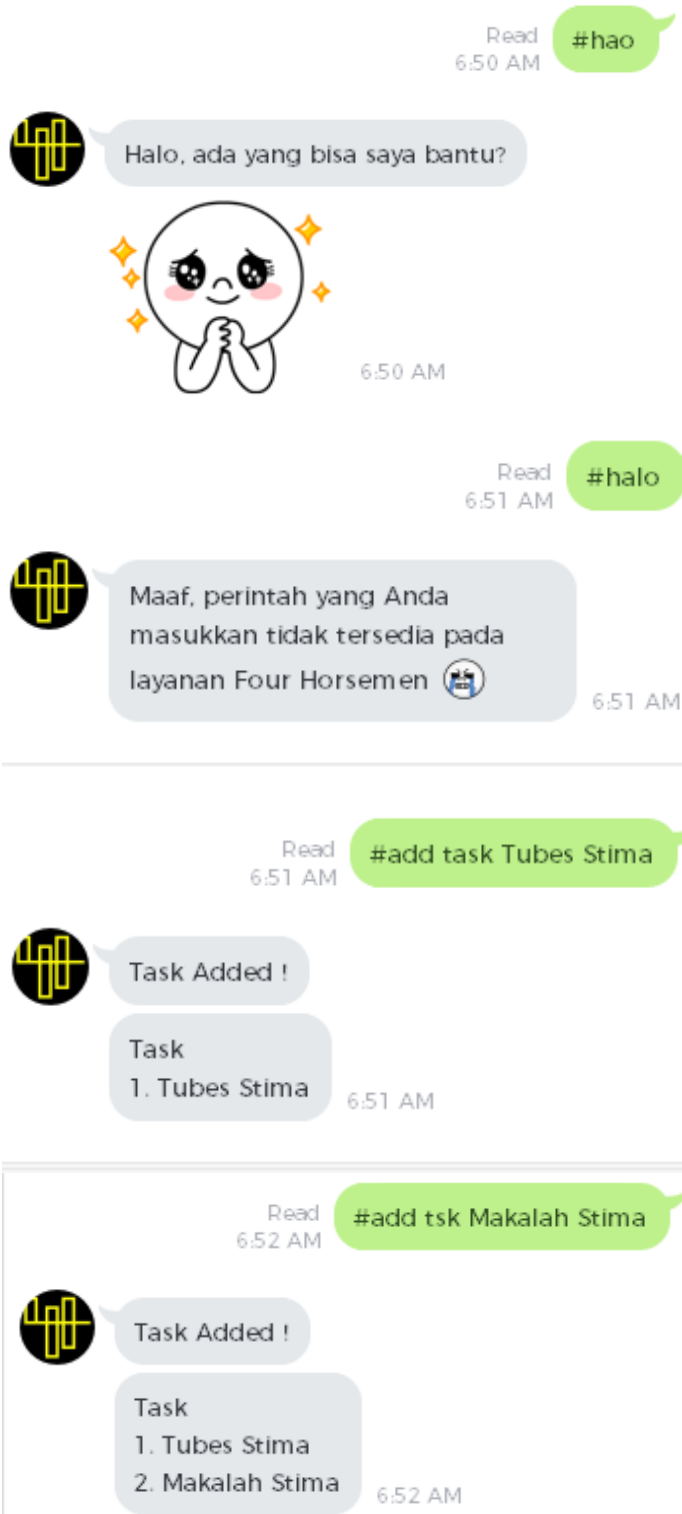
Kata Pertama	Kata Kedua
hai	hao
hai	halo
task	tsk
remove	remive
remove	rmf
Stima	Stims
Stima	Stima

Hasil penghitungan Jaro-Winkler Distance dengan *scaling factor* 0.1 didapatkan sebagai berikut:

```
E:\ITB\CODE\Semester 4\Stima>py jaro-winkler.py
Calculating similarity from hai and hao :
Scaling factor: 0.1
Score : 0.82
Verdict: Similar

Calculating similarity from hai and halo :
Scaling factor: 0.1
Score : 0.78
Verdict: Not Similar

Calculating similarity from task and tsk :
Scaling factor: 0.1
Score : 0.92
Verdict: Similar

Calculating similarity from remove and remive :
Scaling factor: 0.1
Score : 0.92
Verdict: Similar

Calculating similarity from remove and rmf :
Scaling factor: 0.1
Score : 0.7
Verdict: Not Similar

Calculating similarity from Stima and Stims :
Scaling factor: 0.1
Score : 0.92
Verdict: Similar

Calculating similarity from Stima and Stima :
Scaling factor: 0.1
Score : 1.0
Verdict: Similar
```

Dari hasil tersebut didapatkan bahwa dengan nilai batas *lower-bound* 0.8 ditemukan bahwa hao adalah hasil *typographical error* dari hai sedangkan halo bukan merupakan *typographical error* dari hai. Tsk juga merupakan hasil error dari task dengan tipe deletion. Remive juga teridentifikasi merupakan hasil error dari remove, tetapi rmf bukan merupakan error dari remove. Hal serupa berlaku untk Stima dan Stims yang memiliki kemiripan hanya substitusi antara a dan s saja sehingga diidentifikasi sebagai *typo*. Untuk pengujian terakhir diujikan kata Stima dengan Stima yang sama secara eksak dan diperoleh nilai 1.0.

Berikut hasil percakapan dengan chatbot yang telah diperlengkapi fitur untuk pendeteksian *typographical error*.



Berdasarkan hasil pengujian pada chatbot yang dibuat oleh penulis, dapat dilihat bahwa kata kunci yang diujikan telah berhasil diinterpretasikan oleh bot dengan benar. Bot memberikan respon yang sesuai dengan kata kunci yang telah terdefinisi dalam basis data. Dibuktikan pula bot berhasil menampilkan pesan error apabila kata-kata tidak dianggap sebagai kesalahan pengetikan.

V. KESIMPULAN

Algoritma *string matching* dapat dikembangkan dalam berbagai persoalan kehidupan. Salah satu penerapannya adalah pendeteksian kesalahan pengetikan. Algoritma Jaro-Winkler Distance dapat digunakan untuk mengidentifikasi *Typographical Error* dengan menggunakan fungsi constraint dan fungsi pembatas jarak Jaro Winkler yang mengukur kemiripan dua buah string. Pengidentifikasian kesalahan pengetikan ini didasari oleh algoritma pencocokan string dengan metode *Approximate String Matching*. Jaro Winkler bekerja dengan baik ketika string yang digunakan panjangnya relatif pendek karena memiliki kompleksitas waktu O(mn) yang berarti akan turut meningkat bersamaan dengan panjang string.

Algorima ini cukup sederhana dan cepat dalam hal eksekusi. Pertimbangan utama yang harus dipikirkan ketika menggunakan algoritma ini adalah algoritma ini hanya

berfokus pada *common character* saja sehingga akurasinya sebenarnya bisa ditingkatkan dengan mempertimbangkan *non-common character*. Algoritma Jaro Winkler juga baik untuk mendeteksi jenis kesalahan penulisan berupa penghapusan huruf. penggantian huruf, dan penukaran huruf.

Pengembangan dapat dilakukan dengan menggunakan algoritma yang lebih mutakhir lainnya seperti klasifikasi menggunakan *machine learning* dan *natural language processing* sehingga permasalahan kesalahan pengetikan ini bisa diselesaikan dengan respons bahasa yang lebih alami lagi.

UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Kuasa atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini dengan baik dan pada waktu yang tepat. Penulis mengucapkan terima kasih kepada kedua orang tua serta teman-teman yang terus memberikan dukungan baik secara moral maupun doa. Ucapan terima kasih turut penulis sampaikan kepada Bapak Rinaldi Munir, selaku dosen dari mata kuliah Strategi Algoritma yang telah membimbing penulisan makalah ini. Akhir kata, penulis memohon maaf apabila terdapat kekurangan dan kesalahan kata dalam makalah ini. Penulis berharap makalah ini dapat digunakan sebaik-baiknya dan dikembangkan sehingga lebih menghasilkan manfaat bagi masyarakat luas

REFERENCES

[1] Kurniawati, Anna. Implementasi Algoritma Jaro-Winkler Distance untuk Membandingkan Kesamaan Dokumen Berbahasa Indonesia

[2] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algortima. Program Studi Teknik Informatika ITB.

[3] Identifikasi Kesalahan Penulisan Kata. https://www.researchgate.net/publication/323365722_Identifikasi_Kesalahan_Penulisan_Kata_Typographical_Error_pada_Dokumen_Berbahasa_Indonesia_Menggunakan_Metode_N-gram_dan_Levenshtein_Distance [accessed May 14 2018].

[4] Rochmawati, Yeny, Kusumaningrum, Retno. Universitas Diponegoro. Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks. 2015

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Mei 2018

Christian Wibisono - 13516147