Penerapan *Regular Expression* dalam *Predictive Text* dengan Metode N-Gram

Ryo Richardo 13519193
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ryorichardo06@gmail.com

***Abstract*—Perkembangan dunia teknologi pada saat ini sudah mencapai tahap yang lebih kompleks. Teknologi, khususnya dalam dunia perangkat lunak saat ini sudah semakin pintar dan meniru cara berpikir manusia. Salah satu perkembangan perangkat lunak ini adalah *predictive text,* yaitu perangkat lunak yang dapat memprediksi kata yang akan dituliskan sesuai bahasa manusia sehari-hari. Metode implementasi *predictive text* yang paling terkenal adalah metode N-Gram, dimana perangkat lunak memprediksi kata yang akan dituliskan berdasarkan beberapa kata sebelumnya. Untuk menerapkan program *predictive text* ini, kita dapat memanfaatkan algoritma pencocokan string, terutama *regular expression*.**

Keywords—predictive text; N-Gram, regular expression

I. PENDAHULUAN

Dewasa ini, perkembangan teknologi dalam dunia digital sudah mengalami perubahan yang sangat cepat jika dibandingkan dengan era-era sebelumnya. Kecanggihan teknologi dapat menyediakan segala informasi dan kebutuhan kita dalam perangkat yang berukuran hanya segenggaman tangan kita saja. Ya, dengan sebuah *smartphone* dan koneksi internet yang memadai, sebagian besar kebutuhan manusia dapat terpenuhi tanpa bantuan alat yang lain. Perkembangan dunia digital ini sangat menakjubkan karena berhasil menggantikan fungsi dari beragam benda seperti jam, buku, telepon, bahkan komputer yang pada jaman dahulu berukuran sangat besar dan berharga sangat mahal.

Semakin berkembangnya dunia digital juga berpengaruh dalam perkembangan inovasi-inovasi baru yang dapat dilakukan oleh seorang *software* engineer saat ini. Salah satu inovasi yang cukup baru dalam dunia perangkat lunak adalah algoritma *predictive text*. Sesuai dengan namanya, algoritma *predictive text* adalah algoritma yang dapat memprediksi kata apa yang akan kita tulis berikutnya ketika kita sedang menulis pesan. Fitur dengan algoritma *predictive text* sering kita jumpai ketika sedang menulis *email* maupun ketika mengetikkan pesan singkat melalui *keyboard* virtual dalam *smatphone* kita. Fitur ini biasanya muncul dalam bentuk rekomendasi kata yang dapat kita pilih untuk melengkapi pesan yang sedang kita ketik. Algoritma *predictive text* sangat bermanfaat untuk mempercepat waktu mengetik serta mengurangi kemungkinan adanya *typo* (salah ketik) dalam teks.

Walaupun terlihat sederhana, perkembangan algoritma *predictive text* bukanlah suatu hal yang mudah. Untuk membangun algoritma *predictive text* dibutuhkan *dataset* dalam jumlah besar dan data pola pengetikkan kata penggunanya. Beragam metode juga sudah diciptakan untuk menyempurnakan algoritma *predictive text* ini. Dalam makalah ini, saya akan menjelaskan bagaimana algoritma *predictive text* bekerja serta mengimplementasikannya dalam sebuah program sederhana.

II. DASAR TEORI

A. String

String adalah salah satu tipe data yang digunakan dalam *programming* untuk merepresentasikan teks. Secara harafiah, string merupakan himpunan character (tipe data, bukan alfabet) termasuk spasi dan angka. Sebagai contoh, kata "informatika" dan frasa "saya adalah mahasiswa semester 4 informatika" adalah string. Biasanya, sebuah string dituliskan diantara tanda kutip dua ("") agar dapat dibedakan dengan angka atau nama variabel.

Sebuah string memiliki beberapa komponen di dalamnya. Asumsikan terdapat sebuah string S dengan panjang m yang berisi:

$$S = x_0x_1\ldots x_{m-1}$$

Sebuah prefix dari string S adalah substring yang beranggotakan semua character pada S[0…*k*]. sebuah suffix dari string S adalah substring yang beranggotakan semua character pada S[*k*…*m-1*]. Dalam pengertian ini, *k* adalah sembarang indeks yang berada diantara 0 dan *m-1*. Sebagai contoh, sebuah string "informatika" memiliki prefix "i", "in", "inf", "info", "infor", "inform", "informa", "informat", "informati", "informatika" dan suffix "a", "ka", "ika", "tika", "atika", "matika", "rmatika", "ormatika", "formatika", "nformatika", "informatika".

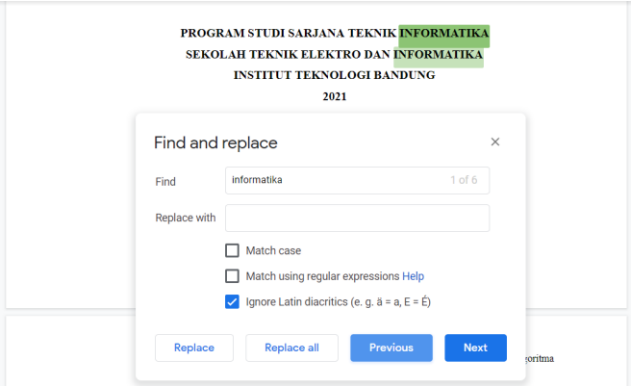
B. Pencocokan String

Pencocokan string, yang dalam Bahasa Inggris disebut *string matching* / *pattern matching* adalah sebuah algoritma yang berfungsi menemukan suatu pola string tertentu di dalam sebuah teks. Dalam algoritma pencocokan string terdapat beberapa istilah sebagai berikut:

- T: teks (text), sebuah string dengan panjang *n* character.

- P: pola (pattern), sebuah string dengan panjang *m* character (asumsi *m* < *n*) yang akan dicari di dalam teks.

Salah satu aplikasi algoritma pencocokan string adalah untuk mencari sebuah kata di dalam sebuah dokumen. Sebagai contoh, berikut adalah aplikasi pencocokan string dalam fitur *find and replace* di *website* Google Docs.



Gambar 1. Fitur *find and search* pada Google Docs
(sumber: Dokumen Pribadi)

Pada gambar tersebut, pola "informatika" ditemukan dalam dokumen sebanyak 6 kali. Selain pencarian pola dalam teks, algoritma pencocokan string juga dapat diimplementasikan dalam *web search engine*, analisis citra, serta pencarian rantai DNA dalam keilmuan Bioinformatika.

C. *Regular Expression*

Regular Expression atau yang biasa disingkat regex adalah notasi standar untuk mendeskripsikan sebuah pola yang terdiri dari beberapa character atau string. Keunggulan regex adalah dapat mencari string dalam pola tertentu, tidak harus berupa pola eksak seperti algoritma pencocokan string lainnya. Regex sudah menjadi standar dalam banyak *tools* dan bahasa pemrograman yang umum digunakan.

Berikut adalah macam-macam format karakter yang digunakan untuk membangun regex:

- String literal

  Bentuk paling dasar untuk membangun regex. Dapat dibangun oleh semua karakter kecuali karakter yang memiliki fungsi khusus. Misalnya regex "informatika" akan cocok dengan string "informatika".

- Metakarakter

  Metakarakter adalah karakter khusus yang tidak termasuk dalam string literal. Metakarakter terdiri atas karakter ., |, *, +, ?, ^, -, =, $, !, \, <, >, (,), [,], {, }. Fungsi dari masing-masing karakter akan dijelaskan kemudian.

- Karakter *any*

  Karakter titik (.) mewakilkan keberadaan karakter apapun kecuali baris baru (\n). Misalnya regex ".amu" akan cocok dengan string "kamu", "jamu", dan string lainnya yang sejenis.

- Operator *or*

  Untuk mengaplikasikan operator *or* dalam regex, kita dapat menggunakan karakter (|). Contohnya regex "aku|kamu" akan cocok dengan string "aku" atau "kamu".

- *Quantifiers*

  Regex mendukung pencocokan string dengan *quantifier*, yaitu banyak perulang pola dalam string. Karakter khusus yang terlibat dalam pembentukan *quantifier* adalah (*, +, ?, {, }).

  - Karakter (*) menunjukkan kemunculan pola di depannya sebanyak nol atau banyak. Contohnya regex "a*" akan cocok dengan string "" (kosong), "a", "aaaaa", dan seterusnya.
  - Karakter (+) meunjukkan kemunculan pola di depannya sebanyak satu atau banyak. Contohnya regex "b+" akan cocok dengan string "b", "bbb", dan seterusnya.
  - Karakter (?) menunjukkan kemunculan pola di depannya sebanyak nol atau satu. Contohnya regex (c?) akan cocok dengan string "" (kosong) atau "c".
  - Pasangan karakter ({}) yang diisi dengan angka dapat menghitung jumlah kemunculan pola secara spesifik. Contohnya regex "d{3}" akan cocok dengan string "ddd", sedangkan regex "e{1,3}" akan cocok dengan string "e", "ee", atau "eee".

- *Classes*

  Regex juga mendukung adanya kelas atau himpunan karakter tertentu. Karakter khusus yang terlibat dalam pembentukkan *class* adalah ([,], ^, -, &).

  - Pasangan karakter ([]) mendefinisikan sebuah kelas. Contohnya regex "[abc]" akan cocok dengan karakter "a", "b", atau "c".
  - Karakter (^) yang digunakan dalam pasangan karakter ([]) menghasilkan negasi dari kelas di dalamnya. Contohnya regex "[^abc]" akan cocok dengan semua karakter selain karakter "a", "b", dan "c".
  - Karakter (-) yang dihimpit dengan karakter literal menghasilkan karakter dalam *range* yang didefinisikan. Contohnya regex "[a-c]" akan cocok dengan karakter "a", "b", atau "c".
  - *Class* mendukung pencarian karakter dari gabungan 2 himpunan. Contohnya regex "[a-c[g-i]]" akan cocok untuk karakter a sampai c atau g sampai i.
  - *Class* juga mendukung pencarian karakter dari irisan dan subtraksi 2 himpunan dengan memanfaatkan karakter (&). Contohnya regex "[a-c&&[b-d]]" akan cocok dengan karakter "b" atau "c" yang merupakan irisan kedua himpunan tersebut. Regex "[a-d&&[^bc]]" akan cocok

dengan karakter “a” atau “d” yang merupakan subtraksi kedua himpunan tersebut.

- *Predefined character classes*

  Regex memiliki kelas *predefined* untuk kelas-kelas karakter yang digunakan secara umum sehingga tidak perlu didefinisikan ulang lagi secara spesifik.

  - Regex “.” berarti semua karakter
  - Regex “\d” berarti digit ([0-9])
  - Regex “\D” berarti non digit ([^0-9])
  - Regex “\s” berarti *whitespace* ([\t\n\x0B\f\r])
  - Regex “\S” berarti non *whitespace*
  - Regex “\w” berarti karakter *word* ([a-zA-Z_0-9])
  - Regex “\W” berarti karakter non *word*

- *Boundary matchers*

  Boundary matchers digunakan untuk mencari pola pada posisi tertentu, seperti di awal atau akhir string.

  - Karakter (^) digunakan untuk mendeteksi pola di awal string. Contohnya regex “^\d+” dapat mendeteksi pola angka sebanyak 1 atau lebih digit di awal string.
  - Karakter ($) digunakan untuk mendeteksi pola di akhir string. Contohnya regex “\.$” dapat mendeteksi karakter titik di akhir string.
  - Regex “\b” digunakan untuk mendeteksi kata pada string. Contohnya regex “\bw+\b” dapat mendeteksi semua kata dalam string.
  - Regex “\B” digunakan untuk mendeteksi batas bukan kata.
  - Regex “\G” digunakan untuk mendeteksi akhir dari kecocokan sebelumnya.
  - Regex “\z” digunakan untuk mendeteksi akhir dari input.
  - Regex “\Z” digunakan untuk mendeteksi akhir dari input tapi untuk *final terminator* jika ada.

- *Groups*

  Regex mendukung pengelompokan beberapa karakter menjadi sebuah pola tertentu. *Group* didefinisikan dengan menggunakan pasangan karakter (()). *Group* biasanya digunakan untuk melakukan operasi-operasi sebelumnya terhadap string. Contohnya regex “(abc)+” akan cocok dengan pola “abc” yang berjumlah satu atau banyak.

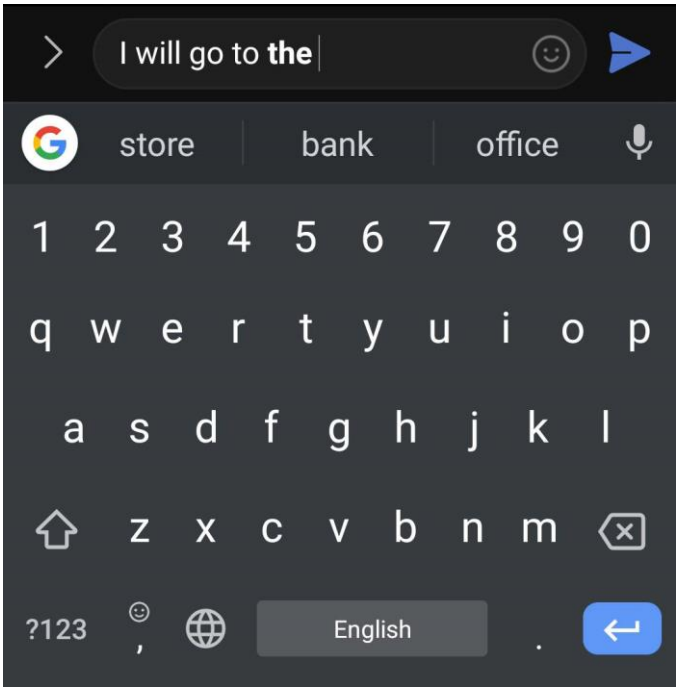
- *Escaped characters*

  Untuk mengenali string yang mengandung karakter khusus, dapat digunakan karakter *backslash* (\) sebagai *escaped character*. Contohnya regex “\.” akan cocok dengan karakter “.”, regex “\\” akan cocok dengan karakter “\\”, dan lain-lain. Karakter (\) juga memiliki fungsi lain untuk mendefinisikan *tab* (\t), *newline* (\n), *return* (\r), kode hex 2 digit (\xhh), *Unicode* 4 digit (\uhhhh), dan *Unicode* 8 digit (\uhhhhhhhh).

D. *Predictive Text*

Predictive text adalah salah satu perkembangan teknologi yang memfasilitasi proses mengetik di perangkat *mobile* dengan cara memberi sugesti kata yang mungkin akan diketik oleh pengguna. Prediksi yang diberikan berdasarkan pada konteks dari kata-kata yang sebelumnya sudah diketikkan. *Predictive text* dapat membantu pengguna untuk mempercepat proses pengetikkan dengan cara langsung memilih prediksi kata yang diberikan daripada harus mengetikkan kata tersebut jika prediksi yang diberikan sesuai dengan keinginan pengguna.

Berikut adalah contoh *predictive text* dalam bahasa Inggris pada Google Keyboard yang dimiliki perangkat Android.



Gambar 2. *Predictive text* dalam Google Keyboard
(sumber: Dokumen Pribadi)

Pada gambar tersebut terlihat setelah mengetikkan “I will go to the”, Google Keyboard memberikan prediksi kata “store”, “bank”, dan “office”. Pengguna dapat langsung menekan kata tersebut jika prediksi tersebut sesuai dengan apa yang ingin diketikkan pengguna.

E. *N-Gram*

Secara harafiah, istilah N-Gram hanyalah berarti rangkaian kalimat atau frasa yang terdiri dari N kata. Sebagai contoh, frasa “saya mahasiswa informatika” merupakan contoh 3-Gram, sedangkan frasa “strategi algoritma” merupakan contoh 2-Gram.

Dalam pembangunan sistem *predictive text*, algoritma yang digunakan untuk memprediksi kata adalah *N-Gram*

Probabilities. Sesuai namanya, algoritma ini memanfaatkan konsep probabilitas kata yang akan muncul setelah frasa tertentu berdasarkan *dataset*. *N-Gram Probabilities* dapat dihitung dengan rumus:

$$Probability = P(w|h) = \frac{P(w \cap h)}{P(h)}$$

Dimana *w* adalah kata yang akan diprediksi dan *h* adalah frasa yang terbentuk sebelum *w*. Jumlah kata yang berada dalam *h* bergantung pada jenis N-Gram yang digunakan. Sebagai contoh, jika kita menggunakan 3-gram, maka *h* akan terdiri dari 2 kata, dan seterusnya.

Untuk mendukung sistem *predictive text* dengan algoritma N-Gram, dibutuhkan *dataset* berupa percakapan sehari-hari dalam jumlah banyak. Pemrosesan algoritma dilakukan dengan cara mengekstrak *dataset* untuk setiap N kata, yang kemudian setiap kata yang muncul setelah N kata tersebut akan ditampilkan sebagai prediksi kata selanjutnya.

III. Implementasi dan Pengujian

A. Implementasi Program

Implementasi *predictive text* dibuat dengan menggunakan bahasa pemrograman Python. Untuk mendukung proses pencarian pola, dibutuhkan *library* regex yang sudah ada dalam *package* Python. Selain itu, sebagai *dataset* sumber pencarian kata, digunakan *dataset* campuran teks berbahasa Indonesia dari berita, *website*, dan lain lain. *Dataset* bersumber dari https://wortschatz.uni-leipzig.de/en/download/Indonesian.

Implementasi *predictive text* dengan algoritma N-Gram dapat dilihat dalam gambar berikut.

```
import re

# Path dataset
path = "data\\ind_mixed_2012_300K-sentences.txt"

# Input
n = int(input("Masukkan n: "))
string = str(input("Masukkan teks: "))

# Regex untuk mengambil karakter alfabet saja
regex = re.compile("[^a-zA-Z \.]")

# Pemrosesan string input
string_alpha_only = regex.sub("", string)
string_no_multi_spaces = re.sub(" +", " ",
string_alpha_only)
string_lowered = string_no_multi_spaces.lower()
m = len(string_lowered.split())
string_last_n = string_lowered.split()[(m-n+1):]
string_final = " ".join(string_last_n)

# Regex untuk mengambil kata berikutnya dari dataset
search_regex = string_final + " (\w+)"

# Inisialisasi list prediksi kata
found_words = []

# Pencarian prediksi kata
with open(path, encoding="utf-8") as dataset:
    for line in dataset:
        alpha_only = regex.sub("", line)
        no_multi_spaces = re.sub(" +", " ", alpha_only)
        lowered = no_multi_spaces.lower()
        found_words += re.findall(search_regex, lowered)

# Pengurutan prediksi kata berdasarkan akurasi
count_all = len(found_words)
sorted_found_words = sorted(found_words, key =
found_words.count, reverse = True)
words = []
count_words = []

# Menampung setiap kata unik serta jumlah kemunculannya
for word in sorted_found_words:
    if word in words:
        count_words[words.index(word)] += 1
    else:
        words.append(word)
        count_words.append(1)


# Output
print("Prediksi kata:")
for i in range (3):
    print(str(i+1) + ". " + words[i] + " , akurasi " +
str(round((count_words[i] / count_all * 100), 2)) + "%")
```

Gambar 3. Implementasi *predictive text* dengan algoritma N-Gram
(sumber: Dokumen Pribadi)

Dalam program tersebut, dibutuhkan input berupa jenis N-Gram yang digunakan, yaitu antara 2, 3, atau 4. Kemudian, program meminta input string yang terdiri dari kata-kata yang ingin diprediksi kata berikutnya. Program memberikan output hingga 3 prediksi paling akurat berdasarkan *dataset*. Apabila pola string tidak ditemukan dalam *dataset*, maka program tidak mengeluarkan apapun. *Source code* lengkap implementasi program tersebut dapat dilihat di laman berikut.

https://github.com/ryorichardo/N-Gram-NLP

B. Hasil Pengujian

Sebagai hasil pengujian, dibutuhkan beberapa *test case* untuk menguji setiap jenis N-Gram yang digunakan. Untuk setiap hasil pencarian, ditampilkan tingkat akurasi kata yang dihitung menggunakan rumus *N-Gram Probabilites*.

- Prediksi 2-Gram

  Prediksi 2-Gram akan mengambil kata terakhir dari sebuah string untuk memprediksi kata berikutnya yang akan diketik.

```
Masukkan jenis N gram (range 2 sampai 4): 2
Masukkan teks: hari ini
Prediksi kata:
1. adalah , akurasi 4.39%
2. tidak , akurasi 2.6%
3. akan , akurasi 2.26%
```

Gambar 4. Pengujian pertama metode 2-Gram
(sumber: Dokumen Pribadi)

Pada pengujian pertama, diberikan masukkan string "hari ini" untuk diprediksi kata berikutnya yang akan diketik. Program berhasil memberikan 3 prediksi terakurat berdasarkan *dataset* yang merupakan kata "adalah", "tidak", dan "akan".

```
Masukkan jenis N gram (range 2 sampai 4): 2
Masukkan teks: pergi
Prediksi kata:
1. ke , akurasi 34.68%
2. dan , akurasi 4.08%
3. dari , akurasi 3.37%
```

Gambar 5. Pengujian kedua metode 2-Gram
(sumber: Dokumen Pribadi)

Pada pengujian kedua, diberikan masukkan string "pergi" untuk diprediksi kata berikutnya yang akan diketik. Program berhasil memberikan 3 prediksi terakurat berdasarkan *dataset* yang merupakan kata "ke", "dan", dan "dari".

- Prediksi 3-Gram

Prediksi 3-Gram akan mengambil 2 kata terakhir dari sebuah string untuk memprediksi kata berikutnya yang akan diketik.

```
Masukkan jenis N gram (range 2 sampai 4): 3
Masukkan teks: hari ini
Prediksi kata:
1. saya , akurasi 4.48%
2. aku , akurasi 4.17%
3. dan , akurasi 3.86%
```

Gambar 6. Pengujian pertama metode 3-Gram
(sumber: Dokumen Pribadi)

Pada pengujian pertama, diberikan masukkan string "hari ini" untuk diprediksi kata berikutnya yang akan diketik. Program berhasil memberikan 3 prediksi terakurat berdasarkan *dataset* yang merupakan kata "saya", "aku", dan "dan".

```
Masukkan jenis N gram (range 2 sampai 4): 3
Masukkan teks: mereka pergi
Prediksi kata:
1. ke , akurasi 25.0%
2. dan , akurasi 8.33%
3. mengunyah , akurasi 5.56%
```

Gambar 7. Pengujian kedua metode 3-Gram
(sumber: Dokumen Pribadi)

Pada pengujian kedua, diberikan masukkan string "mereka pergi" untuk diprediksi kata berikutnya yang akan diketik. Program berhasil memberikan 3 prediksi terakurat berdasarkan *dataset* yang merupakan kata "ke", "dan", dan "mengunyah".

- Prediksi 4-Gram

Prediksi 4-Gram akan mengambil 3 kata terakhir dari sebuah string untuk memprediksi kata berikutnya yang akan diketik.

```
Masukkan jenis N gram (range 2 sampai 4): 4
Masukkan teks: Nampaknya pada hari ini
Prediksi kata:
1. kita , akurasi 11.11%
2. dan , akurasi 8.33%
3. tidak , akurasi 5.56%
```

Gambar 8. Pengujian pertama metode 4-Gram
(sumber: Dokumen Pribadi)

Pada pengujian pertama, diberikan masukkan string "Nampaknya pada hari ini" untuk diprediksi kata berikutnya yang akan diketik. Program berhasil memberikan 3 prediksi terakurat berdasarkan *dataset* yang merupakan kata "kita", "dan", dan "tidak".

```
Masukkan jenis N gram (range 2 sampai 4): 4
Masukkan teks: mereka pergi ke
Prediksi kata:
1. sangkal , akurasi 11.11%
2. mesjid , akurasi 11.11%
3. ladang , akurasi 11.11%
```

Gambar 9. Pengujian kedua metode 4-Gram
(sumber: Dokumen Pribadi)

Pada pengujian kedua, diberikan masukkan string "mereka pergi ke" untuk diprediksi kata berikutnya yang akan diketik. Program berhasil memberikan 3 prediksi terakurat berdasarkan *dataset* yang merupakan kata "sangkal", "mesjid", dan "ladang".

C. Analisis

Dari 6 pengujian untuk 3 metode Gram yang berbeda, terlihat bahwa semakin besar metode yang digunakan, prediksi yang diberikan oleh program akan semakin mendekati penggunaan kata dalam Bahasa Indonesia yang sering kita jumpai. Hal ini terjadi karena semakin banyak kata yang digunakan dalam pencarian, maka semakin spesifik juga hasil pencarian yang ditemukan dalam *dataset*. Namun, penggunaan kata dalam jumlah banyak juga memiliki kelemahan yaitu semakin berkurangnya hasil yang ditemukan dalam *dataset* sehingga ada kemungkinan pencarian tidak memberikan hasil apapun. Untuk mengatasi hal tersebut, kita bisa menggunakan *dataset* dengan ukuran yang lebih besar dan berisi ekstraksi kalimat dalam Bahasa Indonesia yang lebih beragam agar program dapat memberikan prediksi kata yang lebih akurat.

IV. KESIMPULAN DAN SARAN

Algoritma pencocokan string, terutama dengan *Regular Expression* dapat membantu memecahkan permasalahan *predictive text* dengan metode *N-Gram Probabilities*. Dengan bantuan *dataset* yang berisi kumpulan kalimat Bahasa Indonesia yang lengkap, permasalahan *predictive text* dapat diselesaikan dengan hasil prediksi yang cukup akurat.

Untuk pengembangan program *predictive text* dengan hasil yang lebih akurat, penulis menyarankan untuk menambahkan *dataset* yang lebih lengkap dan mencangkup kalimat berbahasa Indonesia dari berbagai macam sumber, seperti berita, *website*, hingga percakapan sehari-hari. Penggunaan *dataset* yang semakin lengkap akan membantu program dalam memprediksi kata yang akan dituliskan.

VIDEO LINK AT YOUTUBE

https://youtu.be/CaKpi2GIO2E

UCAPAN TERIMA KASIH

Puji syukur dan terima kasih penulis ucapkan kepada Tuhan Yang Maha Esa atas segala berkat dan pertolongan-Nya sehingga penulis dapat menyelesaikan makalah ini dengan baik. Selain itu penulis juga mengucapkan terima kasih kepada:

1) Bapak Dr. Ir. Rinaldi Munir, MT., Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc., dan Ibu Dr. Masayu Leylia Khodra, ST., MT., atas bimbingannya khususnya Bapak Rinaldi selaku dosen IF2211 Strategi Algoritma di kelas K-04 yang telah memberikan ilmunya kepada penulis sehingga makalah ini dapat diselesaikan.

2) Keluarga, teman, serta semua orang yang mendukung penulis dalam menyelesaikan makalah ini.

REFERENSI

[1] Kapadia, Sashank. *"LanguageModels: N-Gram"*. https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9. Diakses pada 8 Mei 2021 pukul 09.48 WIB.

[2] Munir, Rinaldi. *"Pencocokan string (String matching/pattern matching)"*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses pada 7 Mei 2021 pukul 06.59 WIB.

[3] Srinidhi, Sunny. *"Understanding World N-grams and N-gram Probability in Natural Language Processing"*. https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058. Diakses pada 8 Mei 2021 pukul 09.36 WIB.

[4] Wibisono, Yudi, dan Masayu Leylia Khodra. *"Pengantar Regular Expression"*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf. Diakses pada 7 Mei 2021 pukul 17.47 WIB.

[5] Wigmore, Ivy. *"Definition Predictive Text"*. https://whatis.techtarget.com/definition/predictive-text. Diakses pada 9 Mei 2021 pukul 11.35 WIB.

[6] https://techterms.com/definition/string. Diakses pada 7 Mei 2021 pukul 06.47 WIB.

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.

Tangerang, 11 Mei 2021

Ryo Richardo 13519193