BAB 2

LANDASAN TEORI

2.1 Typographical Error

Typographical Error atau *typo* merupakan kesalahan yang dilakukan dalam proses pengetikan suatu kata atau kalimat sehingga terjadi kesalahan eja pada kata tersebut (Fahma et al., 2018). *Typo* umumnya terjadi dikarenakan terjadinya slip tangan ketika sedang mengetik suatu kata. Beberapa jenis *typo* yang umum terjadi antara lain, kekurangan huruf pada suatu kata (*deletion*), huruf yang salah pada suatu kata (*substitution*), dan transposisi huruf dimana seluruh huruf pada suatu kata sudah lengkap namun beberapa posisi huruf masih ada yang salah atau tertukar (*transposition*). Contoh *typo deletion* adalah kata ‘buku’ yang menjadi kata ‘buu’. Contoh *typo substitution* adalah kata ‘merah’ yang menjadi kata ‘nerah’. Contoh *typo transposition* adalah kata ‘sungai’ yang menjadi kata ‘sungia’.

2.2 Algoritma Symspell

Pada tahun 2012, Wolf Garbe menemukan algoritma *Symspell* (*Symmetric Delete Spelling Correction*) sebagai pendekatan yang lebih efisien dan membuat waktu pencarian daftar kandidat kata *typo* menjadi seribu kali lebih cepat daripada metode *dynamic programming levenshtein*, namun mengorbankan tingkat akurasi pemilihan kandidat kata *typo* dikarenakan batas jarak *edit Symspell* yang sangat rendah. Meningkatkan batas jarak *edit Symspell* terlalu tinggi akan meningkatkan waktu pencarian daftar kandidat kata *typo*. Algoritma *Symspell* mengurangi kompleksitas waktu dengan hanya melakukan operasi penghapusan saja

dibandingkan algoritma *Levenshtein* yang melakukan operasi penambahan, penghapusan, dan perubahan huruf. Berikut langkah-langkah algoritma *Symspell*,

1. Melakukan operasi penghapusan setiap huruf sebanyak satu kali pada setiap kata yang terdapat di kamus. Setiap kata hasil penghapusan huruf akan disimpan pada suatu *dataset*.
2. Untuk setiap kata *typo*, cari kata tersebut di dalam *dataset* pada tahap sebelumnya. Jika ditemukan, tambahkan kata original dalam kamus ke dalam daftar kandidat kata.
3. Lakukan penghapusan huruf dari kata *typo* tersebut dan cari kata hasil penghapusan di dalam *dataset*. Jika ditemukan, tambahkan kata original dalam kamus ke dalam daftar kandidat kata.

Misalkan salah satu kata pada kamus adalah biru dan kata *typo* yang akan dicek adalah bilu. Hasil pendekatan algoritma *Symspell* dapat dilihat pada Tabel 2.1.

Tabel 2.1 Hasil pendekatan algoritma *Symspell*

biru	bilu
iru	ilu
bru	blu
biu	**biu**
bir	bil

Berdasarkan tabel di atas, terdapat kesamaan hasil penghapusan kata yaitu kata biu. Maka kata biru yang terdapat pada kamus dimasukkan ke dalam daftar kandidat kata yang benar untuk menggantikan kata bilu.

2.3 Bayesian Network

Bayesian Network merupakan sebuah model graf asiklik berarah di mana setiap *edge* pada graf tersebut memiliki arah dan graf tersebut tidak memiliki *cycle*. Dikarenakan sifat graf yang asiklik, maka tidak ada cara untuk mengunjungi *node* yang sama lebih dari satu kali pada sekali perjalanan pada graf tersebut. *Node* pada model *Bayesian Network* merepresentasikan sebuah *event* atau kejadian. Jika terdapat sebuah *edge* yang menghubungi *node* A menuju *node* B secara langsung, maka dapat dikatakan probabilitas terjadinya kejadian B dependen terhadap probabilitas terjadinya kejadian A. Relasi ini dapat dituliskan dalam notasi $P(B \mid A)$. Dapat disimpulkan bahwa probabilitas gabungan dari kejadian A dan B dapat dinotasikan dengan rumus $P(A,\ B) = P(B \mid A) \cdot P(A)$. Untuk menghitung probabilitas terjadinya suatu kejadian yang memiliki dependensi langsung terhadap kejadian lain dapat dilakukan dengan teorema Bayes (Stephenson dan Todd A., 2000).

Definisi 1. (Bayes, 1763) Berikut rumus yang diusul oleh Thomas Bayes untuk menghitung probabilitas kejadian A terjadi jika kejadian B terjadi.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \quad (1)$$

P(A) menandakan probabilitas terjadinya kejadian A. *P*(A | B) menandakan terjadinya probabilitas terjadinya kejadian A jika kejadian B terjadi. Dalam penelitian ini, notasi A pada rumus menandakan kata yang menjadi kandidat pengganti *typo* dan B menandakan kata yang bersebelahan dengan kata *typo*. Dibutuhkan juga korpus yang akan menjadi variabel dasar pada algoritma *Bayes*. Korpus berisi seluruh kata yang sah menurut KBBI daring beserta frekuensi kata tersebut ditemukan pada proses pengumpulan data. Setiap kata pada korpus berasal

dari artikel berita daring (Kompas dan Republika) dan beberapa buku elektronik yang didapatkan dengan metode *web scraping*. Sehingga *P*(kata kandidat) merupakan probabilitas frekuensi kemunculan kata kandidat pada korpus terhadap jumlah kata pada korpus dan *P*(A | B) merupakan probabilitas munculnya kata A di depan kata B dalam suatu kalimat. Algoritma akan menghitung peluang kemunculan setiap kata dalam daftar kandidat terhadap kata B yang merupakan kata yang bersebelahan dengan kata *typo*. Kata dengan peluang kemunculan tertinggi akan dijadikan sebagai kata pengganti yang benar.

Bayesian Network digunakan pada sistem *spell checker* karena dependensi antar kejadian atau fitur mempengaruhi hasil akhir. Kata-kata yang bersebalahan satu sama lain pada suatu kalimat memiliki nilai dependensinya masing-masing sehingga dapat menjaga konteks kalimat. Berbeda dengan *Naïve Bayes Classifier* yang mengasumsikan setiap fiturnya independen terhadap satu sama lain (Rish, 2001).

2.4 Optimisasi Dynamic Programming

Dynamic programming merupakan metode optimisasi matematis yang juga dapat diaplikasikan pada pemrograman komputer (pemrograman linier) yang pertama dikembangkan oleh Richard Bellman pada tahun 1940an (Abdulla, 2018). Metode ini dapat menyelesaikan permasalahan yang kompleks dengan memecah permasalahan tersebut menjadi banyak sub-permasalahan dan menyelesaikan sub-permasalahan tersebut hanya sekali dan menyimpan hasil dari setiap sub-permasalahan dalam suatu tabel. Jika pada suatu waktu dalam algoritma ada

diperlukan hasil dari suatu sub-permasalahan yang sama dan pernah dikerjakan sebelumnya, hasil tersebut dapat langsung diambil dari tabel sub-permasalahan.

Suatu permasalahan dapat diselesaikan dengan metode *dynamic programming* jika dan hanya jika permasalahan tersebut memiliki sub-struktur yang optimal dan sub-permasalahan yang *overlap*. Sebagai contoh, permasalahan *shortest path* pada graf memiliki sub-struktur permasalahan yang optimal. Jika terdapat *node b* yang dilewati oleh salah satu jalur terpendek dari *node a* menuju *node c*, maka jalur yang ditempuh dari *node a* menuju *b* dan jalur yang ditempuh dari *node b* menuju *c* pada jalur terpendek dari *node a* menuju *c* merupakan salah satu jalur terpendek dari *node a* menuju *b* dan *node b* menuju *c*. Contoh permasalahan yang memiliki sub-permasalahan yang *overlap* dapat ditemukan pada permasalahan *fibonacci*. Pada permasalahan *fibonacci* yang diselesaikan secara rekursif, banyak sub-permasalahan yang sama yang diselesaikan berulang kali. Dengan menyimpan hasil suatu sub-permasalahan pada tabel, sub-permasalahan yang sama cukup diselesaikan hanya satu kali saja sehingga kompleksitas waktu permasalahan tersebut menjadi linier.

Metode *dynamic programming* dapat diaplikasikan pada model *Bayesian network* untuk mencari kalimat hasil koreksi kata *typo* dengan nilai probabilitas tertinggi. Probabilitas terbentuknya suatu kalimat didapat dari melakukan perkalian probabilitas kejadian setiap dua kata yang bersebelahan pada kalimat tersebut. Sebagai contoh, kalimat “aku makan nasi” memiliki nilai probabilitas sebesar probabilitas kejadian *P*(“makan” | “aku”) dikali dengan *P*(“nasi” | “makan”). Dikarenakan daftar kandidat kata untuk setiap kata *typo* dapat berjumlah sangat banyak, akan memerlukan waktu yang sangat banyak untuk mencoba semua

kombinasi kalimat yang dapat dibentuk. Dengan *dynamic programming*, waktu yang diperlukan untuk mendapatkan kalimat dengan probabilitas tertinggi dapat berkurang hingga mencapai kompleksitas waktu linier terhadap jumlah kata pada kalimat tersebut.

Optimisasi *dynamic programming* untuk mencari kalimat dengan probabilitas tertinggi dapat dimodelkan sebagai graf dengan kata pada kalimat sebagai titik atau *node* dan probabilitas kejadian antar dua kata sebagai garis atau *edge* yang menghubungkan kata pertama menuju kata kedua. Sehingga permasalahan ini dapat disimpulkan menjadi permasalahan mencari jalur terpanjang dari suatu graf asiklik berarah di mana kata pertama pada kalimat menjadi titik mulai dan kata terakhir pada kalimat menjadi titik akhir pada jalur tersebut. Berikut transisi *dynamic programming* yang diimplementasikan secara rekursif (*top-down approach*) dengan teknik *memoization* untuk mencari kalimat dengan probabilitas tertinggi.

$$dp_{i,j} = \begin{cases} 1 & if\ i = n, \\ \max\limits_{0 \le k < len(word[i+1])} P(word[i+1][k] \mid word[i][j]) * dp_{i+1,k} & otherwise \end{cases} \quad (2)$$

$dp_{i,j}$ melambangkan nilai probabilitas kalimat pada posisi kata ke-i dengan memilih kandidat kata ke-j. *word*[j] melambangkan daftar kandidat kata untuk posisi kata ke-j. *word*[j][k] melambangkan kandidat kata ke-k untuk menggantikan kata pada posisi ke-j. Variabel n melambangkan jumlah kata pada kalimat. Probabilitas tertinggi dari kombinasi kalimat yang mungkin bernilai $\max\limits_{0 \le i < len(word[0])} dp_{0,i}$. Variabel *dp* juga berperan sebagai tabel *memoization* untuk menyimpan hasil sub-permasalahan yang pernah diselesaikan sebelumnya. Implementasi *dynamic programming* dilakukan secara rekursif dikarenakan

implementasinya yang lebih mudah dan transparan dibandingkan tabular. Transisi *dynamic programming* secara rekursif dibagi menjadi dua kasus, kasus dasar dan kasus rekursif. Kasus dasar terjadi jika telah mencapai kata terakhir pada kalimat dan fungsi akan mengembalikan nilai 1 sebagai nilai probabilitas kata terakhir. Selain kasus dasar, kasus rekursif akan terjadi dan fungsi akan mengembalikan nilai probabilitas $dp_{i,j}$ yang didapatkan dari nilai probabilitas maksimal kalimat pada posisi kata ke-*i* jika memilih kandidat kata ke-*j*.