BAB 2
LANDASAN TEORI

2.1 Bahasa Pemrograman Go

Go atau Golang diciptakan pada bulan September 2007 oleh Google yang dipelopori oleh Robert Griesemer, Rob Pike, dan Ken Thompson dan diumumkan kepada publik pada bulan November 2009. Bahasa pemrograman ini dirancang dengan tujuan untuk menjadi efisien dalam proses kompilasi dan eksekusi, serta efektif dalam menulis program yang dapat diandalkan [6]. Dalam pengembangannya, Go menghindari fitur yang memperumit kode dan tidak dapat efisien, seperti halnya bahasa pemrograman C, sehingga Go dapat menghasilkan output yang optimal dengan waktu minimal. Fitur yang diandalkan dalam bahasa pemrograman ini antara lain konkurensi yang efisien dan pendekatan yang sangat fleksibel terhadap abstraksi data dan pemrograman berorientasi objek.

Go memiliki kelebihan karena mengimpelemntasikan beberapa kelebihan atau keuntungan dari bahasa pemrograman pendahulunya tanpa menghilangkan kesederhanaan pada bahasa pemrograman ini. Berikut fitur utama yang meliputi [7]:

1. Sintaks sederhana, Go memiliki sintaks yang sederhana dan mudah dipahami seperti halnya dengan C. Hal ini memudahkan programmer untuk lebih cepat mempelajari dan menggunakan bahasa pemrograman ini.

2. Dukungan konkurensi, Go memiliki dukungan konkurensi bawaan, yang membuatnya mudah untuk menulis program yang dapat menjalankan banyak tugas secara bersamaan.

3. *Garbage Collection*, Go memiliki *Garbage Collection* yang efisien untuk membantu alokasi dan deallokasi memori untuk menghindari resiko *memory leaks*.

4. Waktu kompilasi yang relatif cepat, dikarenakan Go memiliki kompiler yang cepat untuk mengkompilasi kode, hal ini membuatnya ideal untuk proyek berskala besar.

5. Dukungan lintas *platform*, Go hadir dengan pustaka standar yang komprehensif yang menyediakan berbagai dukungan untuk para

pengembang.

Namun Go memiliki beberapa kekurangan yang di sebabkan oleh prinsip kesederhanaannya yang dibangun pada bahasa pemrograman ini seperti, tidak memiliki konversi numerik secara implisit, tidak ada konstruktor dan desktruktor, tidak ada *default parameter*, tidak ada *inheritance* dan tidak ada *thread-local storage*. Dengan beberapa pengembangan yang dilakukan pada proses pembentukan bahasa pemrograman ini dan didasarkan pada kelebihan yang dimiliki oleh beberapa bahasa pemrograman pendahulunya, Golang memiliki *run time* yang cenderung lebih cepat dibandingkan pendahulunya, yang salah satu penyebabnya adalah konsep kesederhanaan yang diusung pada bahasa pemrograman ini.

2.2 Algoritma Cosine Similiarity

Cosine similarity digunakan untuk membangun graf pada suatu dokumen. *Cosine similarity* pada sistem Pengoreksian Otomatis berfungsi untuk mengukur kesamaan antar dua vektor dokumen antar sebuah jawaban yang di *input* dengan opsi jawaban yang tersedia. Berikut rumus perhitungan bobot antar dokumen yang digunakan dalam implementasi nya [8]:

$$\text{cosine similarity} = \frac{\sum_{\text{token} \in \text{tokens1}} \text{freq1}[\text{token}] \cdot \text{freq2}[\text{token}]}{\sqrt{\sum_{\text{token} \in \text{tokens1}} \text{freq1}[\text{token}]^2} \cdot \sqrt{\sum_{\text{token} \in \text{tokens2}} \text{freq2}[\text{token}]^2}} \tag{2.1}$$

Penjelasan dari diagram rumus 2.1 adalah sebagai berikut:

- **tokens1** dan **tokens2** merupakan kumpulan kata yang muncul pada masing-masing kalimat yang disimpan dalam bentuk *array*.
- **freq1** dan **freq2** merupakan *map* yang menampung frekuensi kemunculan token pada masing-masing kalimat.
- **freq1[token]** dan **freq2[token]** adalah frekuensi kemunculan token pada masing-masing kalimat.

Keuntungan utama dari metode *Cosine Similarity* adalah tidak terpengaruh oleh panjang-pendeknya suatu dokumen [9]. Secara umum, perhitungan metode ini didasarkan pada ruang vektor ukuran kesamaan. Namun algoritma ini pun memiliki

kekurangan yaitu semakin banyaknya dokumen yang akan dicari maka semakin banyak waktu yang dibutuhkan [10].

2.3 Term Frequency-Inverse Document Frequency (Algoritma TF-IDF)

algoritma ini merupakan salah satu metode pembobotan perhitungan statistik untuk mengetahui seberapa penting sebuah kata dan seberapa sering kata tersebut tampak pada sebuah dokumen [11]. Untuk mengetahui sebuah nilai pada setiap token pada sebuah dokumen algoritma ini memiliki formula sebagai berikut [12]:

$$TF(t,d) = \frac{\text{jumlah kemunculan kata } t \text{ dalam dokumen } d}{\text{total jumlah kata dalam dokumen } d} \tag{2.2}$$

Term Frequency (TF) merupakan frekuensi terjadinya suatu kata *(term)* dalam sebuah dokumen yang relevan [13]. Semakin sering suatu kata muncul dalam sebuah dokumen, maka semakin besar nilai kesesuaian-nya.

$$IDF(t,D) = \log\left(\frac{\text{total jumlah dokumen dalam kumpulan dokumen } D}{\text{jumlah dokumen yang mengandung kata } t}\right) \tag{2.3}$$

Inverse Document Frequency (IDF) merupakan perhitungan bagaimana suatu kata *(term)* terdistribusi secara keseluruhan dalam kumpulan dokumen [13]. IDF menunjukkan keterkaitan suatu kata dalam kumpulan dokumen, semakin kecil sebuah dokumen mengandung kata *(term)* yang dimaksudkan maka semakin besar nilai IDF.

$$TF-IDF(t,d,D) = \text{TF}(t,d) \times \text{IDF}(t,D) \tag{2.4}$$

TF-IDF menghasilkan vektor representasi untuk setiap dokumen dalam kumpulan dokumen, dimana setiap dimensi vektor merupakan bobot TF-IDF dari suatu kata *(term)* [13].

Penjelasan dari diagram rumus 2.4 adalah sebagai berikut:

- ***t*** merupakan kata dalam sebuah dokumen (term).
- ***d*** merupakan dokumen.
- ***D*** merupakan kumpulan dokumen

- ***TF(t,d)*** mengukur seberapa sering **kata *(t)*** muncul dalam **dokumen *(d)***
- ***IDF(t,D)*** mengukur seberapa umum atau jarangnya **kata *(t)*** dalam kumpulan **dokumen *(d)***
- ***TF-IDF(t,d,D*** mengukur bobot untuk untuk **kata *(t)*** dalam **dokumen *(d)*** dalam **kumpulan dokumen *(D)***

Representasi vektor dari hasil perhitungan TF-IDF dari setiap dokumen dapat membantu perhitungan *Cosine Similiarity* untuk menilai kemiripan antar dokumen. Pada penilitian sebelumnya algoritma ini berhasil mendapatkan nilai akurasi paling tinggi dibanding metode lainnya, terlihat pada tabel 2.1.

Tabel 2.1. Perbandingan Algoritma yang Dilakukan pada Penelitian "An optimized lstm-based augmented language model (flstm-alm) using fox algorithm for automatic essay scoring prediction"

Algoritma	RMSE	Percent RMSE
Cosine	0.5158	20.63
Jaccard	0.4138	16.55
TF-IDF	0.4274	17.10
TF-IDF Jaccard	0.3921	15.68
LSA	0.5368	21.47

2.3.1 Mean Square Error (MSE)

Mean Square Error (MSE) adalah salah satu metrik yang digunakan untuk mengukur seberapa baik sebuah model memprediksi atau memperkirakan nilai yang diinginkan [14]. MSE dihitung dengan mengambil rata-rata dari kuadrat selisih antara nilai yang diprediksi dan nilai sebenarnya. Rumus MSE adalah sebagai berikut:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.5}$$

- n adalah jumlah observasi dalam dataset.
- y_i adalah nilai sebenarnya dari observasi ke-i.
- $\hat{y}_i$ adalah nilai yang diprediksi oleh model untuk observasi ke-i.

- $\sum$ adalah simbol untuk menjumlahkan nilai dari 1 hingga n.

MSE memberikan ukuran kesalahan rata-rata kuadrat antara nilai yang diprediksi dan nilai sebenarnya. Nilai MSE yang lebih rendah menunjukkan bahwa model memiliki kesalahan prediksi yang lebih kecil.

2.3.2 Root Mean Square Error (RMSE)

Root Mean Square Error (RMSE) adalah akar kuadrat dari MSE dan memberikan ukuran kesalahan yang berada dalam satuan yang sama dengan data asli [14]. RMSE sering digunakan karena lebih mudah diinterpretasikan dibandingkan dengan MSE. Berikut formula yang digunakan:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.6}$$

- n adalah jumlah observasi dalam dataset.
- y_i adalah nilai sebenarnya dari observasi ke-i.
- $\hat{y}_i$ adalah nilai yang diprediksi oleh model untuk observasi ke-i.
- $\sum$ adalah simbol untuk menjumlahkan nilai dari 1 hingga n.
- $\sqrt{\cdot}$ adalah operasi akar kuadrat.

RMSE memberikan indikasi bahwa semakin rendah nilai RMSE, semakin tinggi tingkat akurasi model, sementara semakin tinggi nilai RMSE, semakin rendah tingkat akurasi. Dengan demikian, metodologi ini sangat relevan untuk mengevaluasi tingkat akurasi dari algoritma yang digunakan dalam penelitian ini, yang bertujuan untuk menilai seberapa baik algoritma tersebut dapat mengoreksi jawaban siswa pada pertanyaan esai.

2.4 Black Box

Pengujian dilakukan menggunakan pendekatan black box, dimana algoritma ini merupakan pengujian yang dilakukan dengan mengamati hasil eksekusi dari suatu sistem [15]. Pengujian ini berfokus terhadap peniliain secara fungsional dari fitur yang telah di bangun pada suatu sistem. Dalam pengujian ini, beberapa parameter yang diuji meliputi:

- **Variasi Jawaban Siswa:** Sejumlah variasi jawaban siswa dibuat untuk menguji respons algoritma terhadap beragam struktur dan gaya penulisan.
- **Jumlah Kata:** Pengujian dilakukan dengan memvariasikan jumlah kata dalam jawaban siswa untuk mengamati bagaimana algoritma bereaksi terhadap tingkat kompleksitas yang berbeda.
- **Kesalahan Umum:** algoritma diuji dengan memasukkan kesalahan umum yang sering terjadi dalam penulisan siswa, seperti ejaan yang salah, tata bahasa yang tidak tepat, atau kesalahan konsep.
- **Perbandingan dengan Jawaban Benar:** Respons algoritma dibandingkan dengan jawaban yang benar untuk mengukur tingkat akurasi dan keberhasilannya dalam memperbaiki jawaban-jawaban yang salah.

Dengan parameter-parameter pengujian tersebut, pengujian menggunakan black box memberikan gambaran yang komprehensif tentang kinerja algoritma dalam mengoreksi jawaban siswa.