BAB 2
LANDASAN TEORI

2.1 Deteksi Kesalahan Gramatikal

Deteksi kesalahan kata (*Grammatical Error Detection* / GED) adalah proses otomatis yang bertujuan mengidentifikasi bagian-bagian dalam teks yang tidak sesuai dengan aturan tata bahasa, seperti kesalahan pada penggunaan preposisi, bentuk kerja, artikel, maupun kesesuaian subjek-predikat. Contohnya, dalam tulisan pembelajar bahasa, dapat ditemukan kekeliruan pada penggunaan konjungsi, kata kerja, preposisi, atau artikel [25][26]. Proses Grammatical Error Detection menjadi langkah awal sebelum tahap koreksi dan dapat diterapkan untuk meningkatkan kualitas tulisan secara keseluruhan [27].

Grammatical Error Detection juga memiliki peran penting dalam bidang Natural Language Processing (NLP), karena kemampuan mengenali kesalahan tata bahasa membantu sistem memahami cara manusia menggunakan bahasa dalam dunia nyata [28]. Tantangan utamanya mencakup mendeteksi kesalahan yang tersembunyi serta perbedaan gaya penggunaan bahasa dari penulis dengan berbagai tingkat kemampuan berbahasa [29]. Dengan demikian, GED tidak hanya berfokus pada kesalahaan ejaan atau kata yang tidak dikenal, tetapi juga mendeteksi pelanggaran struktur atau penggunaan bahasa yang tidak sesuai konteks penulisan [28].

2.2 Partikel dalam Bahasa Indonesia

Menurut Kamus Besar Bahasa Indonesia (KBBI), partikel dalam Bahasa Indonesia adalah kelas kata yang memiliki makna gramatikal namun tidak memiliki makna leksikal jika berdiri sendiri, sehingga keberadaannya hanya berdampak dalam kaitannya dengan kata lain dalam kalimat (Kembikbud, 2023). Bentuk partikel umumnya pendek dan tidak berubah (tidak mengalami afiksasi atau infleksi) serta berfungsi sebagai penegas, pembentuk kalimat tanya, ataupun sebagai penghubung yang menetapkan struktur kalimat. Dalam penulisan resmi, kaidah pemakaian dan penulisan partikel seperti -lah, -kah, -tah, pun, dan per telah diatur secara jelas oleh Pedoman Umum Ejaan Bahasa Indonesia (Kementrian Pendidikan, Kebudayaan, Riset, dan Teknologi, 2023).

2.3 Natural Language Preprocessing (NLP)

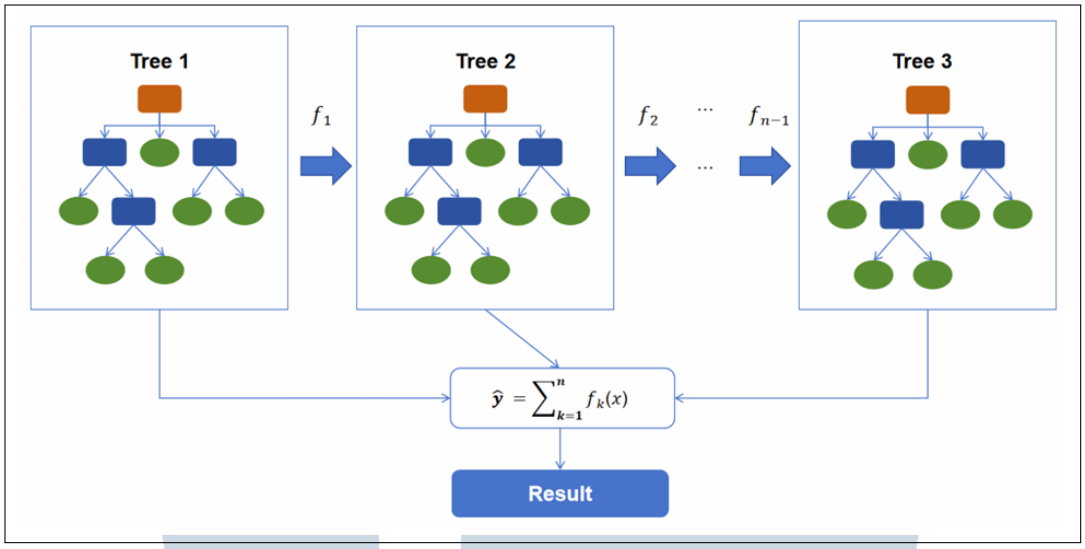
Natural Language Processing (NLP) adalah cabang dari kecerdasan buatan (Artificial Intelligence) yang berfokus pada interaksi antara komputer dan bahasa manusia. Tujuan utama NLP adalah memungkinkan mesin untuk memahami, menafsirkan, dan menghasilkan bahasa alami dengan cara yang bermakna dan bermanfaat [30][31]. Bidang ini menggabungkan linguistik, ilmu komputer, dan pembelajaran mesin untuk memproses teks atau suara dalam berbagai aplikasi seperti chatbot, analisis sentimen, penerjemahan otomatis, deteksi kesalahan tata bahasa, dan ekstraksi informasi [30]. Secara umum, NLP berusaha menjembatani cara manusia berkomunikasi secara alami dengan cara komputer memproses data berbasis logika dan angka [32].

Dalam konteks Natural Language Processing, salah satu penerapannya adalah kemampuan untuk memahami struktur dan kaidah tata bahasa. Bagian teks yang tidak sesuai dengan aturan bahasa dapat dikenali melalui proses Grammatical Error Detection (GED), seperti kesalahan penggunaan kata kerja, preprosisi, atau konjungsi [29][33]. Dengan demikian, konsep Natural Language Processing menjadi langkah fundamental untuk membantu proses evaluasi maupun perbaikan teks secara otomatis.

2.4 Extreme Gradient Boosting (XGBoost)

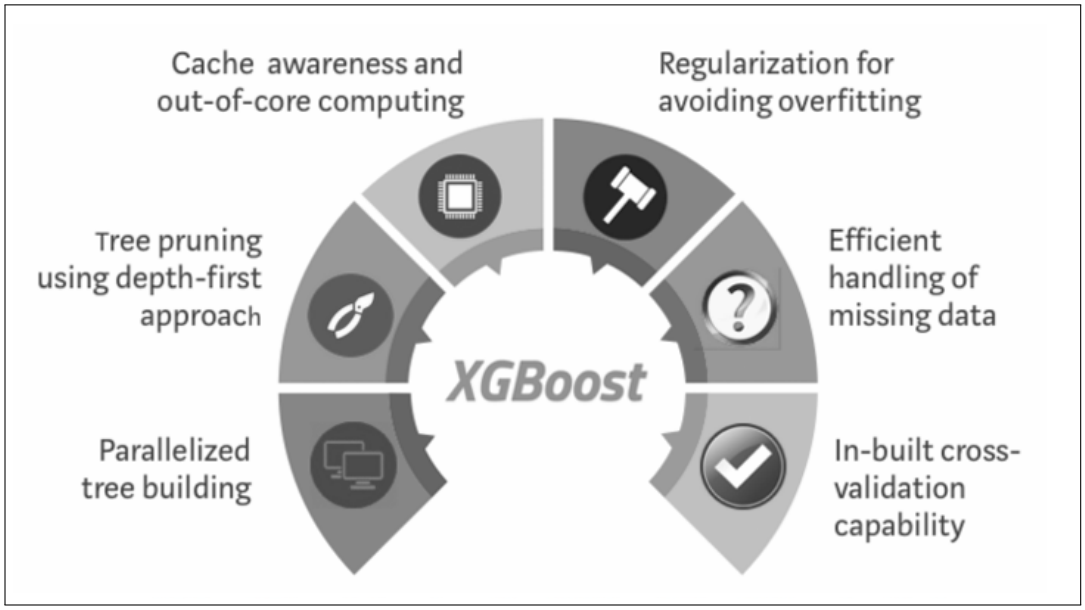
Extreme Gradient Boosting (XGBoost) adalah algoritma machine learning berbasis pohon keputusan yang dikembangkan oleh Tianqi Chen dan Carlos Guestrin pada tahun 2016 [34]. Algoritma ini bekerja dengan mengoptimalkan proses pembelajaran melalui serangkaian pohon keputusan yang dibangun secara bertahap, di mana setiap pohon berusaha memperbaiki kesalahan prediksi yang dibuat oleh pohon sebelumnya [34].

XGBoost dikenal karena kemampuannya menghasilkan model prediktif yang kuat, efisien, serta mampu menangani dataset berukuran besar dan kompleks dengan baik. Dalam konteks klasifikasi dan regresi, XGBoost menggunakan prinsip boosting, yaitu teknik yang menggabungkan sejumlah model lemah (weak learners) untuk membentuk model kuat (strong learner). Setiap iterasi akan berfokus pada kesalahan dari iterasi sebelumnya sehingga model secara bertahap memperbaiki performanya [35].



Gambar 2.1. XGBoost
[36]

XGBoost menerapkan konsep *Gradient Boosting Decision Tree* (GBDT) dengan sejumlah fitur utama yang menjadikannya unggul dibanding metode boosting tradisional.



Gambar 2.2. Features of XGBoost Algorithm
[37]

Figure 2 menunjukkan bahwa XGBoost memiliki beberapa komponen inti yang berperan penting dalam proses pembelajaran, yaitu:

1. *Cache Awareness and Out-of-Core Computing*: Fitur *out-of-core computing* memungkinkan XGBoost menangani dataset yang lebih besar dari kapasitas memori utama (RAM) dengan cara membaca data secara bertahap dari penyimpanan.

2. *Regularization for Avoiding Overfitting*: XGBoost memiliki mekanisme *regularization* (L1 dan L2) yang dimasukkan ke dalam fungsi objektifnya untuk menghindari *overfitting*.

3. *Efficient Handling of Missing Data*: XGBoost secara otomatis menangani nilai hilang (*missing values*) tanpa memerlukan imputasi manual. Selama proses pelatihan, algoritma menentukan jalur terbaik (*default direction*) untuk setiap nilai yang hilang berdasarkan hasil pelatihan sebelumnya.

4. *In-built Cross-Validation Capability*: XGBoost memiliki kemampuan *cross-validation* bawaan yang memungkinkan evaluasi performa model dilakukan secara otomatis pada setiap iterasi pelatihan untuk memperoleh hasil yang optimal.

5. *Parallelized Tree Building*: Proses pembentukan pohon dapat dilakukan secara paralel dan mempercepat waktu komputasi.

6. *Tree Pruning Using Depth-First Approach*: XGBoost akan memangkas cabang pohon yang tidak memberikan kontribusi signifikan terhadap peningkatan akurasi.

XGBoost menggunakan *XGBClassifier()* untuk membangun model klasifikasi atau *XGBRegressor()* untuk regresi. Proses pelatihan dilakukan menggunakan fungsi *fit()*, sedangkan prediksi dilakukan melalui *predict()*. Evaluasi performa model dapat dilakukan dengan fungsi *score()* untuk menghitung metrik seperti *accuracy*, *precision*, dan *recall*. Selain itu, atribut *feature_importances_* membantu mengidentifikasi fitur yang paling relevan dalam pengambilan keputusan model [38].

Dalam *XGBClassifier*, *objective function* diatur ke *binary:logistic* untuk klasifikasi biner atau *multi:softmax* untuk klasifikasi multikelas. Fungsi ini mengoptimalkan *log loss* sebagai *loss function*, yang bertujuan untuk meminimalkan kesalahan prediksi probabilitas dan memberikan penalti besar terhadap kesalahan yang memiliki tingkat keyakinan tinggi tetapi salah. XGBoost juga mendukung regularisasi L1 (Lasso) dan L2 (*Ridge*) untuk mencegah

overfitting. Parameter *reg_alpha* digunakan untuk L1 yang menekan beberapa bobot fitur menjadi nol, sedangkan *reg_lambda* digunakan untuk L2 yang mengurangi besaran bobot tanpa menekannya menjadi nol, membantu mencegah *overfitting* [38][39]. *Hyperparameter* seperti *n_estimators*, *learning_rate*, dan *max_depth* juga dapat disesuaikan untuk mengoptimalkan performa model lebih lanjut [38].

XGBoost meminimalkan fungsi objektif berikut [34]:

$$L^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{2.1}$$

Keterangan:

1. $L^{(t)}$: Fungsi objektif pada iterasi ke-t
2. n: Jumlah total sampel dalam dataset
3. $l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right)$: *Loss function* yang mengukur selisih antara nilai sebenarnya dan prediksi
4. y_i: Nilai sebenarnya dari target
5. $\hat{y}_i^{(t-1)}$: Prediksi pada iterasi ke-$t-1$
6. $f_t(x_i)$: Fungsi keputusan baru yang akan dihasilkan pada iterasi ke-t

Nilai gain digunakan untuk menentukan pemisahan node terbaik (splitting node). Rumusnya ditunjukkan sebagai berikut [34]:

$$L_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{2.2}$$

Keterangan:

1. $\sum_{i\in I_L} g_i$: total gradient untuk semua sampel di subset kiri (cabang kiri) setelah split dilakukan, yang mengukur total kesalahan prediksi di subset tersebut.
2. $\sum_{i\in I_R} g_i$: total gradient untuk semua sampel di subset kanan (cabang kanan) setelah split, mengukur total kesalahan prediksi di subset tersebut.
3. $\sum_{i\in I} g_i$: total gradient sebelum split dilakukan untuk semua sampel di node saat ini, yang mengukur total kesalahan prediksi sebelum pembagian.

4. $\sum_{i \in I_L} h_i$: total Hessian untuk semua sampel di subset kiri setelah split, yang mengukur kelengkungan fungsi loss di subset kiri.

5. $\sum_{i \in I_R} h_i$: total Hessian untuk semua sampel di subset kanan setelah split, mengukur kelengkungan fungsi loss di subset kanan.

6. $\sum_{i \in I} h_i$: total Hessian untuk semua sampel sebelum split dilakukan, yang mengukur kelengkungan fungsi loss di node sebelum split.

7. λ: parameter regularisasi L2 yang digunakan untuk menyusutkan bobot di node, mencegah bobot yang terlalu besar agar model tidak overfitting.

8. γ: parameter penalti untuk menambah node baru pada pohon, digunakan untuk menghindari pohon yang terlalu dalam dan terlalu kompleks.

Dengan menerapkan fungsi objektif dan regularisasi tersebut, XGBoost mampu menjaga keseimbangan antara akurasi tinggi dan kompleksitas model yang rendah. Selain itu, metode ini mendukung parallel computation, tree pruning, dan pengelolaan memori yang baik.

2.5 Rule-Based System

Sistem berbasis aturan (rule-based system) adalah pendekatan yang menentukan pola linguistik secara eksplisit seperti *regular expression* (regEx), kamus/domain lexicon, atau logika seperti *if-then* yang memetakan konteks ke label [40][41]. *Rule-based system* bersifat deterministik, presisi pada pola yang didefinisikan, serta tidak bergantung pada data berlabel dalam jumlah besar sehingga cocok untuk *low-resource language* [42]. Namun sistem berbasis aturan (rule-based system) memiliki kelemahan utama karena bergantung pada aturan buatan yang terbatas, menyebabkan kesulitan menangani ambiguitas dan variasi dalam penggunaan bahasa alami [43].

Dalam penelitian modern, *rule-based system* digunakan bersama algoritma *machine learning* untuk membentuk pendekatan hibrida. Model ini memanfaatkan keunggulan aturan linguistik untuk mendeteksi pola yang eksplisit sekaligus kemampuan model statistik dalam menangani variasi konteks bahasa [42]. Pendekatan hibrida telah terbukti meningkatkan kinerja sistem deteksi kesalahan tata bahasa dan ekstraksi informasi, terutama dalam bahasa yang kompleks atau

memiliki sumber daya terbatas seperti Bahasa Indonesia [3]. Sehingga, *rule-based system* tetap menjadi komponen penting dalam pengembangan sistem Natural Language Processing (NLP) yang akurat dan dapat dijelaskan [29].

2.6 Evaluasi Model

Evaluasi model adalah proses untuk mengukur seberapa baik model yang dibangun dalam tugas klasifikasi (atau regresi) dalam memprediksi label pada data. Tujuan utamanya adalah untuk menilai seberapa akurat dan andal performa model, membandingkan beberapa model atau konfigurasi seperti perbedaan algoritma atau *hyperparameter* berdasarkan metrik yang objektif, serta menentukan apakah model siap digunakan ke data nyata. Sebagai dasar, evaluasi klasifikasi menggunakan *confusion matrix* yang menyediakan jumlah *True Positives* (TP), *True Negatives* (TN), *False Positives* (FP), dan *False Negatives* (FN) dari hasil prediksi model [44][45].

A Akurasi (Accuracy)

Akurasi adalah proporsi prediksi benar (baik positif maupun negatif) terhadap seluruh prediksi. Perhitungan akurasi dapat menggunakan rumus berikut [44]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

B Presisi (Precision)

Presisi mengukur sejauh mana prediksi positif model benar-benar positif. Perhitungan presisi dapat menggunakan rumus berikut [44]:

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

C Recall

Recall mengukur sejauh mana model berhasil mendeteksi seluruh kejadian positif. Recall dapat diukur dengan rumus berikut [44]:

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

D F1-Score

F1-Score adalah rata-harmonik antara presisi dan recall, yang memberi keseimbangan antara kedua metrik tersebut. Rumus dari F1-Score adalah sebagai berikut [44]:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.6}$$

Dengan menggunakan F1-Score, evaluasi model dapat dilakukan secara keseluruhan yang memperhitungkan trade-off antara precision dan recall.