BAB 2
LANDASAN TEORI

Diabetes mellitus, atau yang lebih umum disebut diabetes, merupakan penyakit metabolik yang ditandai oleh hiperglikemia tanpa pengobatan, dengan etiologi yang beragam, termasuk kelainan produksi dan kinerja insulin, serta masalah metabolisme karbohidrat, lipid, dan protein. Penyakit ini dapat menyebabkan berbagai komplikasi serius, seperti ”retinopati, nefropati, dan neuropati” [7], serta meningkatkan risiko kondisi lain seperti penyakit jantung, obesitas, dan infeksi menular seperti TBC. Gejala diabetes sering kali meliputi ”poliuria, polidipsia, polifagia” [7], dan penurunan berat badan tanpa penyebab jelas, tetapi dalam beberapa kasus, individu mungkin tidak menunjukkan gejala apa pun. Tanda klinis yang lebih parah, seperti ”ketoasidosis” [7], dapat mengakibatkan dehidrasi, ketidaksadaran, dan bahkan kematian jika tidak ditangani dengan tepat.

Selain faktor gaya hidup seperti pola makan tidak sehat dan kurang olahraga, faktor genetik juga memainkan peran penting dalam perkembangan diabetes, terutama diabetes tipe 2. Penelitian menunjukkan bahwa adanya riwayat keluarga dengan diabetes secara signifikan meningkatkan risiko seseorang mengembangkan penyakit ini [8]. Gen-gen tertentu mempengaruhi cara tubuh memproses gula, menghasilkan insulin, dan merespons insulin. Beberapa mekanisme genetik yang terlibat meliputi kerusakan pada gen yang memproduksi insulin, yang dapat menyebabkan kekurangan insulin dan memicu diabetes tipe 1, serta gangguan pada gen yang mengkode reseptor insulin, yang menyebabkan resistensi insulin dan diabetes tipe 2 [8]. Selain itu, polimorfisme genetik, yaitu variasi genetik umum pada populasi, juga telah dikaitkan dengan peningkatan risiko diabetes tipe 2 [8]. Telaah ini menekankan pentingnya deteksi dini, manajemen diabetes, serta pemahaman peran genetik untuk mencegah komplikasi yang lebih serius [8].

Parameter `random_state` digunakan untuk mengontrol proses pengacakan pada algoritma *machine learning* agar hasil model menjadi konsisten setiap kali dijalankan. Pada algoritma seperti *Random Forest*, pengacakan terjadi saat pemilihan sampel data (*bootstrap sampling*) dan subset fitur untuk membangun pohon keputusan. Tanpa menggunakan `random_state`, hasil evaluasi seperti akurasi, precision, recall, dan f1-score dapat berubah-ubah setiap kali model dilatih ulang. Sebaliknya, dengan menentukan nilai `random_state` (contoh: `random_state=42`), proses pengacakan menjadi tetap, sehingga hasil evaluasi

model akan selalu sama meskipun dijalankan

Random Forest adalah algoritma pembelajaran mesin yang memprediksi hasil dengan membagi variabel secara biner dan mengatasi keterbatasan decision tree dalam data kompleks. Algoritma ini menggunakan beberapa pohon keputusan yang dibangun dari pengacakan sampel pelatihan (*bootstrap sampling*) dan subset acak variabel prediktor. Penggabungan hasil dilakukan dengan *majority voting* untuk klasifikasi atau rata-rata untuk regresi, meningkatkan akurasi secara keseluruhan. Penelitian ini menggunakan algoritma Random Forest untuk tugas klasifikasi, di mana hasil yang diprediksi berupa kategori, yaitu apakah seseorang mengidap diabetes atau tidak. Dengan pendekatan ansambel ini, Random Forest terbukti lebih akurat dibandingkan decision tree dalam berbagai tugas klasifikasi [9].

Random Forest Classifier tersedia dalam *library* scikit-learn, sebuah pustaka populer untuk pembelajaran mesin di Python. Random Forest merupakan sebuah *meta estimator* yang membangun sejumlah pohon keputusan (*decision tree classifiers*) pada berbagai sub-sampel dari dataset dan menggunakan rata-rata untuk meningkatkan akurasi prediktif serta mengendalikan *overfitting*. Dengan fungsi `RandomForestClassifier`, pengguna dapat dengan mudah mengimplementasikan model klasifikasi secara efisien, memanfaatkan parameter yang fleksibel untuk disesuaikan demi mengoptimalkan performa model[10].

Random Forest memiliki beberapa parameter penting yang dapat disesuaikan untuk mengoptimalkan performa model:

- **n_estimators**: Menentukan jumlah pohon yang dibangun dalam model. Semakin banyak pohon yang digunakan, semakin besar kemungkinan model akan tahan terhadap overfitting, tetapi juga semakin banyak waktu komputasi yang dibutuhkan untuk menyelesaikannya[11].

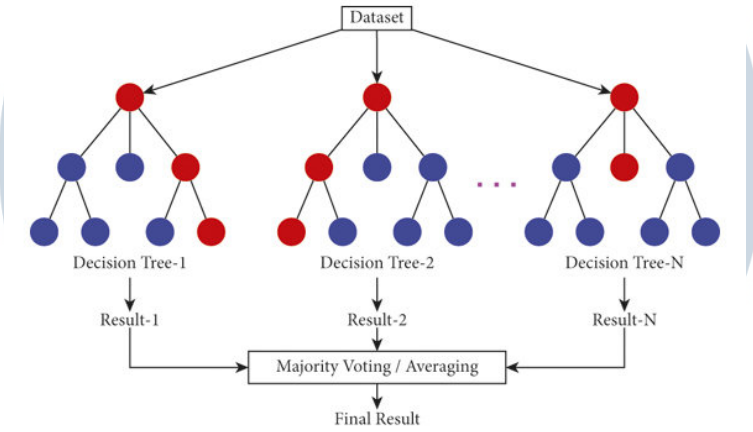
- **max_depth**: Mengatur kedalaman maksimum setiap pohon. Kedalaman yang lebih besar dapat meningkatkan kompleksitas model[11].

- **min_samples_split**: Menentukan jumlah minimum sampel untuk membagi sebuah node, yang mengontrol pembentukan pohon agar tidak terlalu kompleks.

- **min_samples_leaf**: Jumlah sampel terkecil yang diperlukan untuk setiap daun (simpul terakhir) pohon. Ini membantu mencegah pembentukan daun dengan jumlah sampel yang terlalu kecil, yang dapat menyebabkan overfitting[11].

- **random_state**: Mengontrol proses pengacakan, memastikan hasil pelatihan konsisten dan mudah direplikasi.

Gambar 2.1 menunjukkan bagaimana setiap pohon dibangun menggunakan subset data (*bootstrap sample*) dan bagaimana prediksi akhir dihasilkan dari penggabungan seluruh pohon.



Gambar 2.1. Ilustrasi Pohon Keputusan dalam Random Forest

Extreme Gradient Boosting atau XGBoost dikembangkan oleh Chen dan Guestrin pada tahun 2016. XGBoost adalah salah satu metode *boosting*, yaitu teknik ansambel yang menggabungkan beberapa pohon keputusan (*decision trees*) secara berurutan, di mana pembangunan setiap pohon baru didasarkan pada kesalahan yang dihasilkan oleh pohon sebelumnya. Metode ini dirancang untuk meningkatkan akurasi dengan memperbaiki kesalahan yang terus berkurang pada setiap iterasi.[12] Istilah *eXtreme* dalam XGBoost merujuk pada peningkatan kinerja dan efisiensi yang dioptimalkan melalui teknik komputasi canggih, seperti paralelisasi yang memungkinkan eksekusi beberapa pohon secara bersamaan, pengelolaan memori yang lebih baik, serta penggunaan algoritma pemangkasan pohon untuk mengurangi waktu komputasi dan overfitting [13]. *Gradient* dalam XGBoost mengacu pada proses pembaruan bobot menggunakan metode *gradient descent*, yaitu teknik optimasi berbasis turunan pertama dari fungsi loss. Gradien yang dimaksud adalah turunan parsial dari fungsi loss terhadap prediksi model pada setiap data, yang menunjukkan arah dan besaran perubahan yang diperlukan untuk mengurangi kesalahan prediksi (*error*). Setiap pohon yang baru dibangun akan menggunakan informasi gradien ini untuk mengarahkan perubahan pada pembelajaran selanjutnya[13]. Sementara itu, *Boosting* menggambarkan proses

iteratif di mana setiap pohon keputusan baru dibangun untuk memperbaiki kesalahan yang dihasilkan oleh pohon sebelumnya. Pada iterasi awal, pohon pertama menghasilkan prediksi awal yang lemah dengan inisialisasi probabilitas tertentu, biasanya berupa rata-rata dari label target untuk regresi atau pembobotan yang seimbang untuk klasifikasi. Kesalahan prediksi yang dihasilkan kemudian dihitung menggunakan fungsi loss (misalnya, *log loss* atau *MSE*), dan gradien dari loss tersebut digunakan untuk memperbarui bobot setiap data[13]. Data yang salah diklasifikasikan atau memiliki error tinggi akan diberi bobot lebih besar pada iterasi berikutnya, sehingga pohon selanjutnya lebih fokus pada memperbaiki kesalahan tersebut. Proses ini berlanjut secara iteratif dengan memperbarui setiap pohon menggunakan pendekatan *gradient boosting*, di mana kontribusi masing-masing pohon dikalibrasi dengan menyesuaikan bobot berdasarkan gradien error yang dihasilkan. Dengan pendekatan ini, XGBoost mampu menghasilkan kumpulan pohon keputusan yang kuat dan akurat dalam berbagai tugas klasifikasi maupun regresi.

XGBClassifier adalah kelas dalam pustaka XGBoost yang digunakan untuk melakukan klasifikasi. Kelas ini mengimplementasikan algoritma Extreme Gradient Boosting (XGBoost), yang dirancang untuk meningkatkan akurasi model klasifikasi dengan membangun model secara bertahap. XGBClassifier sangat efisien dalam menangani dataset besar dan sering digunakan dalam kompetisi pembelajaran mesin karena performanya yang tinggi dan kemampuannya untuk mengatasi berbagai jenis data.

Dalam penggunaannya, terdapat beberapa parameter penting yang perlu diperhatikan. Salah satunya adalah `use_label_encoder`, yang mengontrol penggunaan label encoder internal yang sebelumnya digunakan dalam XGBoost. Parameter ini diatur ke `False` untuk menghindari peringatan terkait penggunaan label encoder yang sudah tidak direkomendasikan lagi, sehingga membantu menjaga kompatibilitas dengan versi terbaru dari pustaka[14]. Selanjutnya, `eval_metric` adalah parameter yang menentukan metrik evaluasi yang akan digunakan selama pelatihan model. Dengan mengatur `eval_metric` ke metrik yang sesuai, seperti `logloss`, pengguna dapat memantau performa model selama pelatihan, yang berguna untuk mengidentifikasi apakah model mengalami overfitting atau tidak. Log loss dipilih karena kemampuannya untuk mengukur seberapa baik model memprediksi probabilitas kelas, memberikan penalti yang lebih besar untuk prediksi yang salah. Parameter lain yang penting adalah `random_state`, yang digunakan untuk mengatur seed acak. Dengan mengatur

`random_state`, pengguna dapat memastikan bahwa hasil yang diperoleh dapat direproduksi, yang penting dalam eksperimen dan pengujian model.

Selain itu, terdapat beberapa parameter penting lainnya yang dapat digunakan untuk membantu dalam performa pembuatan model dengan XGBClassifier[15].

- **n_estimators**: Banyaknya pohon yang digunakan untuk proses klasifikasi untuk membangun didalam model.
- **max_depth**: Untuk kedalaman pohon. Kedalaman lebih besar memungkinkan model menangkap pola yang lebih kompleks.
- **learning_rate** (eta): Membantu mempersingkat langkah dalam pembaruan model.
- **subsample**: Menentukan proporsi data yang digunakan untuk membangun setiap pohon.
- **colsample_bytree**: Mengontrol proporsi fitur pada data *training* yang digunakan untuk membangun setiap pohon.
- **gamma**: Pengurangan loss minimum yang dibutuhkan untuk membagi simpul daun pada pohon individu.
- **scale_pos_weight**: Digunakan untuk mengatasi masalah ketidakseimbangan kelas dengan memberikan bobot lebih pada kelas yang kurang terwakili.
- **random_state**: Mengontrol proses pengacakan, memastikan hasil pelatihan konsisten dan mudah direplikasi.

Train-test split adalah metode yang digunakan dalam pengembangan dan evaluasi model machine learning. Dalam metode ini, data yang tersedia dibagi menjadi dua bagian: training set dan testing set. Training set, yang mencakup 80% dari data, digunakan untuk melatih model machine learning dengan mempelajari pola dan hubungan antar variabel, sehingga model dapat memprediksi hasil dengan akurasi tinggi. Sementara itu, testing set, yang mencakup 20% dari data, digunakan untuk menguji akurasi model yang telah dilatih. Data dalam testing set tidak digunakan untuk melatih model, sehingga hasil prediksi model dapat dianggap independen dan digunakan untuk mengevaluasi kinerja model secara lebih akurat. Dengan menggunakan train-test split, model machine learning dapat diuji secara

lebih objektif dan akurat, serta dapat mengetahui seberapa baik model dapat memprediksi hasil tanpa melihat data yang digunakan untuk melatih model[16].

Uji perbandingan akurasi bertujuan untuk menentukan seberapa akurat algoritma Random Forest dan XGBoost dalam memprediksi penyakit diabetes dengan mempertimbangkan berbagai skenario. Langkah-langkahnya adalah sebagai berikut:

1. **Pembagian Data:** Dataset dibagi menjadi training set dan testing set menggunakan berbagai rasio (misalnya, 70:30, 80:20) untuk mengevaluasi dampak pembagian data terhadap hasil model. Metode stratified split digunakan untuk menangani distribusi label yang tidak seimbang.

2. **Penanganan Ketidakseimbangan Data:** Teknik oversampling (SMOTE) atau undersampling diterapkan pada training set untuk mengatasi ketidakseimbangan label (imbalanced data) agar performa model lebih adil terhadap kelas minoritas.

3. **Pelatihan Model:** Model dilatih menggunakan training set dengan tuning hyperparameter melalui metode grid search. Parameter yang disesuaikan pada Random Forest dan XGBoost meliputi jumlah pohon, kedalaman maksimal pohon, dan parameter lainnya sesuai kebutuhan model. Kombinasi parameter terbaik dipilih berdasarkan hasil validasi silang.

4. **Pengujian Model:** Model yang telah dilatih diuji menggunakan testing set. Prediksi dibandingkan dengan nilai sebenarnya untuk menghitung metrik evaluasi, termasuk akurasi, precision, recall, dan f1-score.

5. **Evaluasi dan Perbandingan:** Performa kedua algoritma dibandingkan berdasarkan:

    - Akurasi, precision, recall, dan f1-score.
      Berikut adalah tabel 2.1 adalah tabel confusion matrix untuk menghitung hasil evaluasi model

	Prediksi Positif	Prediksi Negatif
Aktual Positif	TP	FN
Aktual Negatif	FP	TN

Tabel 2.1. Confusion Matrix untuk Prediksi Diabetes

Berikut adalah rumus yang digunakan untuk mengukur hasil evaluasinya

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

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

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.4}$$

Dimana:
TP: True Positive (Benar Positif),
TN: True Negative (Benar Negatif),
FP: False Positive (Salah Positif),
FN: False Negative (Salah Negatif).

- Pengaruh pemilihan fitur terhadap hasil model.
- Efektivitas penanganan ketidakseimbangan data.
- Waktu proses pembuatan model.