BAB 2
LANDASAN TEORI

2.1 Gangguan Tidur (Sleep Disorder)

Tidur merupakan proses biologis fundamental yang dibutuhkan tubuh untuk mempertahankan *homeostasis* fisiologis dan kesehatan secara menyeluruh. Secara fisiologis, tidur diatur oleh mekanisme sirkadian yang dikendalikan oleh *suprachiasmatic nuclei* di hipotalamus serta mekanisme *homeostasis* yang melibatkan akumulasi adenosin selama periode terjaga [4]. Arsitektur tidur yang normal terdiri dari dua fase utama yang bergantian secara teratur, yaitu fase *Non-Rapid Eye Movement* (NREM) yang terbagi menjadi tiga tahap (N1, N2, dan N3) serta fase *Rapid Eye Movement* (REM), dengan durasi tiap siklus kurang lebih 90 menit dan berlangsung sebanyak 4 hingga 6 siklus dalam semalam [4, 18]. Selama proses tidur, tubuh menjalankan fungsi-fungsi penting seperti pemulihan fisik, pengaturan sistem imun, konsolidasi memori, serta penstabilan emosional [4, 19].

Gangguan tidur secara medis didefinisikan sebagai kondisi yang ditandai oleh perubahan patologis dalam kualitas, durasi, maupun pola tidur yang tidak sesuai dengan kebutuhan fisiologis tubuh, sehingga berdampak negatif pada fungsi kesehatan individu secara menyeluruh [4, 20]. Kondisi ini bukan sekadar keluhan sementara, melainkan merupakan permasalahan klinis yang apabila tidak ditangani dapat memicu berbagai komplikasi jangka panjang pada sistem kardiovaskular, metabolik, kognitif, maupun kesehatan mental individu [4, 6]. Gangguan tidur diklasifikasikan ke dalam beberapa jenis berdasarkan karakteristik gejala dan mekanisme patofisiologisnya. *Insomnia* merupakan ketidakmampuan memulai atau mempertahankan tidur yang berdampak pada fungsi siang hari, sementara *sleep apnea* ditandai oleh obstruksi saluran napas berulang selama tidur yang menyebabkan hipoksemia intermiten dan fragmentasi tidur [4, 12]. *Narkolepsi* adalah gangguan neurologis kronis akibat defisiensi *hypocretin* yang menyebabkan kantuk berlebihan di siang hari disertai kataplekси, sedangkan *hipersomnia* ditandai oleh rasa kantuk berlebihan meskipun durasi tidur malam sudah mencukupi [12, 20]. *Restless Leg Syndrome* (RLS) merupakan gangguan sensorimotorik berupa dorongan kuat menggerakkan kaki akibat sensasi tidak nyaman yang memburuk di malam hari, dan *Circadian Rhythm Disorder* terjadi ketika jam biologis internal

tidak selaras dengan siklus lingkungan eksternal [18, 20]. Selain keenam gangguan tersebut, kondisi tanpa gangguan tidur atau *No Disorder* mengacu pada pola tidur normal yang dalam penelitian ini berfungsi sebagai kelas pembanding [12].

Gangguan tidur dipengaruhi oleh kombinasi faktor risiko demografis, fisiologis, dan gaya hidup yang saling berinteraksi. Usia, jenis kelamin, obesitas, serta kondisi medis seperti hipertensi dan penyakit kardiovaskular merupakan faktor risiko utama yang secara konsisten dikaitkan dengan berbagai jenis gangguan tidur [4, 6]. Di sisi lain, faktor gaya hidup seperti tingkat stres yang tinggi, kurangnya aktivitas fisik, konsumsi alkohol, dan paparan cahaya layar elektronik sebelum tidur turut memperburuk kualitas tidur melalui gangguan pada ritme sirkadian dan penekanan produksi melatonin [18, 19]. Dampak dari gangguan tidur yang tidak ditangani mencakup peningkatan risiko penyakit jantung dan *stroke* [6], gangguan metabolik berupa resistensi insulin dan obesitas [4], serta penurunan fungsi kognitif yang dapat mempercepat proses neurodegeneratif dan meningkatkan risiko demensia [9].

2.2 Machine Learning

Machine learning merupakan cabang dari kecerdasan buatan (*artificial intelligence*) yang memungkinkan sistem komputer untuk belajar secara otomatis dari data berlabel (*labeled data*) dan meningkatkan kinerjanya tanpa perlu diprogram secara eksplisit untuk setiap tugas tertentu [21]. Proses pembelajaran ini melibatkan dua tahap utama, yaitu tahap pelatihan (*training*), yakni proses algoritma mempelajari pola dari data masukan beserta label yang bersesuaian, dan tahap pengujian (*testing*), proses model yang telah dibangun digunakan untuk memprediksi label dari data baru yang belum pernah dilihat sebelumnya [21, 22]. Berdasarkan ketersediaan label data, *machine learning* dibedakan menjadi beberapa paradigma, yaitu *supervised learning*, *unsupervised learning*, *semi-supervised learning*, dan *reinforcement learning*. *Supervised learning* merupakan paradigma yang paling banyak digunakan dalam konteks klasifikasi medis karena setiap sampel data telah dilengkapi dengan label kelas yang diketahui [22].

Salah satu tugas utama dalam supervised learning adalah klasifikasi, yaitu proses memetakan setiap sampel data ke dalam satu dari beberapa kategori diskret berdasarkan nilai fitur-fitur yang dimiliki [21]. Ketika jumlah kelas yang diprediksi lebih dari dua, tugas ini disebut klasifikasi multiclass, yang memiliki kompleksitas lebih tinggi dibandingkan klasifikasi biner karena model harus

membangun decision boundary yang memisahkan setiap kelas secara simultan dan akurat [22]. Berbagai algoritma supervised learning seperti Random Forest, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), dan Logistic Regression telah terbukti efektif dalam menangani tugas klasifikasi pada berbagai domain, dengan performa yang bervariasi tergantung pada karakteristik dataset yang digunakan [22].

2.3 Data Preprocessing

Data Preprocessing merupakan tahapan awal yang menentukan kualitas masukan sebelum data digunakan untuk melatih model klasifikasi. Data mentah yang berasal dari berbagai sumber umumnya belum siap digunakan secara langsung karena mengandung nilai yang hilang, pencilan, format kategorikal yang belum terkodekan, rentang nilai antar fitur yang tidak seragam, serta distribusi kelas yang timpang. Kondisi tersebut apabila dibiarkan dapat menurunkan performa dan keandalan model, sehingga diperlukan serangkaian teknik praproses untuk mengubah data mentah menjadi representasi numerik yang bersih dan konsisten.

2.3.1 Handling Mussing Value

Nilai hilang (*missing value*) adalah kondisi ketika sebuah atribut tidak memiliki nilai pada sampel tertentu, yang dapat terjadi akibat kesalahan pencatatan, penggabungan data dari sumber berbeda, maupun ketiadaan pengukuran. Keberadaan nilai hilang perlu ditangani karena sebagian besar algoritma pembelajaran tidak dapat memproses data yang tidak lengkap. Strategi penanganannya disesuaikan dengan karakteristik masing-masing atribut, yaitu penghapusan atribut apabila proporsi nilai hilangnya terlalu besar, atau pengisian nilai (*imputation*) apabila informasi yang tersisa masih memadai untuk diestimasi.

Salah satu pendekatan imputasi yang umum digunakan pada atribut numerik adalah imputasi berbasis median, karena median lebih tahan terhadap pengaruh nilai ekstrem dibandingkan rata-rata. Pada data dengan label kelas, imputasi dapat dilakukan secara per kelas (*per-class median imputation*), yakni nilai hilang diisi dengan median dari sampel yang berada pada kelas yang sama, sehingga distribusi antar kelas tetap terjaga dan bias yang timbul akibat pengisian nilai dapat ditekan [12].

2.3.2 Deteksi dan Penanganan Outlier

Dalam sebuah data, *outlier* adalah nilai yang angkanya sangat berbeda atau tidak wajar jika dibandingkan dengan sebagian besar data lainnya. Jika dibiarkan, nilai yang tidak wajar ini bisa mengganggu proses belajar model. Akibatnya, model bisa memberikan tebakan yang salah karena keliru dalam mengenali pola data yang sebenarnya. Untuk mencari nilai *outlier* pada data angka, penelitian ini menggunakan metode *Interquartile Range* (IQR). Metode ini bekerja dengan cara menghitung jarak antara kuartil ketiga (Q_3) dan kuartil pertama (Q_1). Rumus IQR dituliskan sebagai berikut.

$$IQR = Q_3 - Q_1 \tag{2.1}$$

Berdasarkan rumus tersebut, sebuah nilai dianggap sebagai *outlier* jika angkanya lebih kecil dari batas bawah ($Q_1 - 1{,}5 \times IQR$) atau lebih besar dari batas atas ($Q_3 + 1{,}5 \times IQR$). Biasanya, data yang menjadi *outlier* akan langsung dihapus begitu saja. Namun, agar jumlah data latih tidak berkurang, penelitian ini menggunakan teknik pembatasan nilai (*capping*). Dengan teknik ini, nilai yang terlalu ekstrem tidak dibuang, melainkan diubah menjadi sama dengan nilai batas terdekatnya. Langkah ini dipilih agar model tetap bisa memproses data dengan baik tanpa harus kehilangan informasi dari data tersebut [14].

2.3.3 Standardisasi Fitur Numerik

Dalam sebuah data, atribut angka sering kali diukur dengan satuan dan rentang nilai yang berbeda-beda. Jika dibiarkan, atribut yang angkanya jauh lebih besar bisa mendominasi proses belajar model dan menutupi atribut lain yang angkanya lebih kecil [23]. Oleh karena itu, standardisasi dilakukan untuk menyamakan skala dari semua atribut tersebut. Penelitian ini menggunakan metode *Z-score*, yaitu metode yang mengubah nilai data sehingga pada akhirnya nilai rata-rata (*mean*) data menjadi nol dan simpangan bakunya (*standard deviation*) menjadi satu [24]. Rumus *Z-score* dituliskan sebagai berikut.

$$z = \frac{x - \mu}{\sigma} \tag{2.2}$$

Berdasarkan rumus tersebut, x adalah nilai asli data, μ adalah rata-rata dari seluruh data pada atribut tersebut, dan σ adalah nilai simpangan bakunya. Hasil dari perhitungan ini, yaitu z, menunjukkan seberapa jauh sebuah titik data berbeda dari nilai rata-ratanya [23]. Jika dibandingkan dengan metode normalisasi Min-Max yang memaksa data masuk ke dalam rentang angka 0 hingga 1, pendekatan *Z-score* jauh lebih disukai. Alasannya, metode ini tetap mempertahankan bentuk sebaran asli dari data dan lebih kebal terhadap gangguan dari nilai *outlier* [23].

Secara teori, algoritma seperti Random Forest dan XGBoost sebenarnya cukup kuat dan tidak terlalu terpengaruh oleh perbedaan skala angka pada data [24]. Namun, langkah standardisasi ini tetap wajib dilakukan. Hal ini karena pada tahap akhir dari metode *stacking*, hasil prediksi dari kedua algoritma tersebut akan digabungkan oleh Logistic Regression. Kinerja Logistic Regression sangat bergantung pada skala data. Jika skalanya tidak disamakan, pembobotan di dalam model bisa menjadi tidak seimbang dan pada akhirnya menurunkan kualitas prediksi [24]. Selain itu, untuk mencegah kebocoran informasi (*data leakage*) yang bisa membuat hasil pengujian menjadi bias, nilai rata-rata dan simpangan baku hanya dihitung dari data latih saja. Angka hasil hitungan tersebut barulah dipakai untuk ikut mengubah skala pada data uji [24].

2.3.4 Handling Class Imbalance

Ketidakseimbangan kelas (*class imbalance*) pada data terjadi ketika jumlah data pada satu kategori jauh lebih banyak dibandingkan kategori lainnya. Jika dibiarkan, model akan cenderung berpihak pada kelas yang datanya lebih banyak dan mengabaikan kelas yang datanya sedikit. Akibatnya, penilaian kinerja model bisa menjadi keliru apabila hanya melihat nilai akurasinya saja [25]. Untuk mengatasi masalah ini, salah satu teknik yang paling sering diandalkan adalah *Synthetic Minority Oversampling Technique* (SMOTE). SMOTE bekerja dengan cara membuat data buatan baru khusus untuk kelas yang jumlahnya sedikit. Cara ini jauh lebih aman dari sekadar menyalin ulang data secara acak yang biasanya akan membuat model terlalu hafal dengan data latih (*overfitting*). Data buatan ini diciptakan dengan cara menarik garis pembatas antara sebuah titik data dengan salah satu data tetangga terdekatnya (*k-nearest neighbors*). Rumus untuk membuat data baru ini dituliskan sebagai berikut.

$$x_{baru} = x_i + \lambda \left(x_{zi} - x_i \right) \tag{2.3}$$

Berdasarkan rumus tersebut, x_i adalah titik data asli yang jumlahnya sedikit, x_{zi} adalah salah satu tetangga terdekatnya, dan λ adalah angka acak antara 0 hingga 1 yang menentukan letak posisi data buatan tersebut di sepanjang garis penghubung kedua titik [26]. Dengan cara ini, SMOTE berhasil menambah keberagaman data tanpa harus menjiplak data yang sudah ada persis sama. Hasilnya, model bisa mengenali pola dari data kelas minoritas dengan jauh lebih baik [25]. Sebagai langkah kehati-hatian, penambahan data buatan menggunakan SMOTE ini diwajibkan hanya berlaku pada data latih saja. Data uji sama sekali tidak boleh dilibatkan dalam proses penyeimbangan ini. Aturan ini sangat penting untuk mencegah terjadinya kebocoran informasi (*data leakage*) agar hasil pengujian akhir model tetap objektif dan jujur [26].

2.3.5 Feature Engineering

Dalam membangun model klasifikasi, informasi yang paling penting sering kali tidak terlihat secara langsung pada data mentah, melainkan tersembunyi di balik hubungan antar variabel datanya. Oleh karena itu, diperlukan tahapan rekayasa fitur (*feature engineering*), yaitu proses menciptakan atribut baru dari data yang sudah ada berdasarkan pemahaman terhadap topik penelitian. Langkah ini bertujuan agar hubungan tersembunyi tersebut memiliki nilai angka yang jelas dan bisa dipelajari oleh model [27]. Sebagai contoh, ketimbang hanya memasukkan dua angka tekanan darah secara terpisah, selisih dari kedua angka tersebut dapat dihitung untuk mendapatkan sebuah nilai baru yang secara medis jauh lebih berguna.

Pada data yang berbentuk tabel, rekayasa fitur biasanya dilakukan lewat operasi matematika sederhana seperti pengurangan, pembagian, atau perkalian antar atribut untuk melihat bagaimana interaksinya [27]. Penggabungan ini sangat penting karena sebuah atribut kadang baru terlihat dampaknya terhadap hasil prediksi jika dihubungkan dengan atribut lain, sesuatu yang pasti akan terlewatkan jika atribut tersebut hanya dianalisis satu per satu. Sebagai catatan, proses rekayasa fitur ini berbeda dengan seleksi fitur. Rekayasa fitur bertugas menciptakan atribut yang sebelumnya belum ada, sedangkan seleksi fitur bertugas membuang atribut yang tidak berguna. Di dalam praktiknya, kedua langkah ini berjalan berurutan dengan atribut baru diciptakan terlebih dahulu, barulah kemudian seluruh kumpulan atribut tersebut diseleksi.

2.4 Feature Selection

Dalam sebuah data, tidak semua atribut selalu berguna untuk membantu proses prediksi. Beberapa atribut kadang hanya menambah beban perhitungan tanpa memberikan informasi baru, yang pada akhirnya justru berisiko membuat model jadi *overfitting*. Oleh karena itu, tahapan seleksi fitur (*feature selection*) dilakukan untuk menyaring atribut yang paling berhubungan dengan target prediksi agar model menjadi lebih sederhana, cepat, namun tetap akurat [28]. Secara umum, teknik penyaringan ini terbagi menjadi tiga kelompok utama, yaitu metode *filter* yang menilai kegunaan fitur murni dari perhitungan statistik datanya, metode *wrapper* yang menguji coba berbagai kombinasi fitur dengan melatih model berulang kali, serta metode *embedded* yang menyaring fitur secara otomatis bersamaan dengan proses pelatihan model itu sendiri [28]. Pada penelitian ini, penyaringan dilakukan dengan menggabungkan kekuatan dari metode *filter* dan metode *embedded*.

Pendekatan *filter* yang digunakan adalah *Mutual Information*, yaitu teknik perhitungan statistik yang mengukur seberapa banyak informasi dari sebuah fitur yang bisa membantu menjelaskan variabel targetnya. Semakin besar angka yang dihasilkan, semakin kuat pula hubungan antara fitur dan target tersebut. Salah satu keunggulan utama dari metode ini adalah kemampuannya untuk mendeteksi hubungan yang rumit dan tidak lurus (*non-linear*) tanpa bergantung pada model tertentu [28]. Rumus *Mutual Information* antara sebuah fitur X dan target Y dituliskan sebagai berikut.

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)\,p(y)} \tag{2.4}$$

Berdasarkan rumus tersebut, $p(x,y)$ adalah nilai peluang munculnya fitur X dan target Y secara bersamaan. Sementara itu, $p(x)$ dan $p(y)$ adalah nilai peluang munculnya masing-masing variabel secara terpisah. Semakin tinggi hasil perhitungan $I(X;Y)$ ini, semakin besar pula peran atribut tersebut dalam membantu model menebak kelas target dengan benar. Sementara itu, pendekatan *embedded* yang digunakan adalah *Random Forest Feature Importance*. Metode ini bekerja dengan cara melihat seberapa sering sebuah fitur berhasil merapikan dan memisahkan data ke dalam kelas yang benar saat model pohon keputusan sedang dibangun [29]. Atribut yang secara konsisten mampu memisahkan data dengan bersih pada ribuan pohon yang ada akan otomatis mendapatkan skor kepentingan

yang tinggi.

Penelitian ini sengaja menggabungkan hasil penilaian dari kedua metode tersebut untuk mendapatkan urutan fitur yang jauh lebih stabil dan meyakinkan. Penggabungan dilakukan dengan cara menghitung nilai rata-rata peringkat (*rank averaging*). Melalui cara ini, sebuah atribut hanya akan menempati posisi teratas jika atribut tersebut memang dinilai penting oleh kedua metode sekaligus. Pada akhirnya, jumlah atribut yang dipertahankan akan disesuaikan dengan prinsip kesederhanaan (*parsimony*), yaitu memilih jumlah fitur sesedikit mungkin namun tetap memberikan tingkat keakuratan klasifikasi yang tinggi.

2.5 Ensemble Learning

Ensemble learning merupakan pendekatan dalam *machine learning* yang menggabungkan prediksi dari beberapa model pembelajaran (*base learner*) untuk menghasilkan keputusan akhir yang lebih akurat dan andal dibandingkan dengan menggunakan satu model tunggal [30]. Konsep dasar dari pendekatan ini berpijak pada prinsip bahwa kumpulan model yang beragam dan saling melengkapi cenderung menghasilkan generalisasi yang lebih baik, karena kesalahan prediksi dari satu model dapat dikoreksi oleh model lainnya [30]. Secara umum, terdapat tiga teknik utama dalam *ensemble learning*, yaitu *bagging*, *boosting*, dan *stacking*. *Bagging* (*Bootstrap Aggregating*) membangun sejumlah model secara paralel dari subset data yang diambil secara acak dengan penggantian (*bootstrap sampling*), kemudian menggabungkan prediksinya melalui mekanisme *majority voting* untuk klasifikasi atau rata-rata untuk regresi, Random Forest merupakan implementasi paling representatif dari teknik ini [30]. *Boosting* membangun model secara sekuensial dengan setiap model baru difokuskan untuk memperbaiki kesalahan yang dibuat oleh model sebelumnya, sehingga menghasilkan *ensemble* yang semakin adaptif terhadap sampel sulit, XGBoost merupakan salah satu algoritma *boosting* yang paling banyak digunakan saat ini [30].

Di antara ketiga teknik tersebut, *stacking* (*stacked generalization*) memiliki mekanisme yang berbeda secara fundamental, yaitu dengan melatih sebuah model tingkat kedua yang disebut *meta-learner* untuk mempelajari cara terbaik menggabungkan prediksi dari model-model *base learner* yang telah dilatih sebelumnya [31]. Pendekatan ini memungkinkan *meta-learner* untuk memanfaatkan kekuatan komplementer dari masing-masing *base learner*, sehingga menghasilkan prediksi akhir yang lebih optimal [31]. Dalam konteks

prediksi penyakit, teknik *stacking* telah terbukti menghasilkan akurasi tertinggi dibandingkan *bagging* dan *boosting*, dengan *stacking* menunjukkan performa terbaik dalam 19 dari 23 studi yang ditinjau di berbagai domain klasifikasi penyakit [31].

2.6 Random Forest

Random Forest (RF) adalah algoritma *ensemble learning* berbasis *bagging* yang diperkenalkan oleh Breiman (2001), bekerja dengan membangun sejumlah besar *decision tree* secara paralel dari subset data yang diambil secara acak melalui teknik *bootstrap sampling*, kemudian menggabungkan seluruh prediksi pohon melalui mekanisme *majority voting* untuk menghasilkan keputusan akhir [29]. Setiap pohon dalam RF dibangun menggunakan subset fitur yang dipilih secara acak pada setiap node, sehingga menghasilkan pohon-pohon yang beragam dan saling tidak berkorelasi, karakteristik ini yang menjadikan RF lebih tahan terhadap *overfitting* dibandingkan *decision tree* tunggal [29, 32]. RF juga mampu mengukur tingkat kepentingan setiap fitur (*feature importance*) berdasarkan kontribusinya dalam mengurangi ketidakmurnian data di seluruh pohon, sehingga sangat berguna untuk analisis fitur dalam klasifikasi gangguan tidur *multiclass* pada penelitian ini [29, 32]. Cara kerja *Random Forest* dapat dijabarkan melalui empat rumus utama sebagai berikut.

1. *Bootstrap Sampling*

   Setiap pohon ke-t dilatih menggunakan subset data D_t yang diambil secara acak dengan penggantian (*with replacement*) dari dataset asli D berukuran n.

   $$D_t = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \sim D \text{ (with replacement)} \tag{2.5}$$

   Dengan cara ini, setiap pohon hanya melihat sekitar 63% data unik, sementara sisanya berfungsi sebagai data validasi internal (*out-of-bag*).

2. *Gini Impurity*

   Pada setiap node, pemilihan fitur terbaik untuk *split* dilakukan berdasarkan nilai *Gini Impurity* terendah, yang mengukur seberapa sering sampel yang dipilih secara acak akan salah diklasifikasikan.

$$Gini(t) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.6}$$

Dengan p_i adalah proporsi sampel kelas i pada node t, dan c adalah jumlah kelas. Nilai $Gini = 0$ menunjukkan node yang sepenuhnya murni (*pure node*).

3. *Majority Voting*

   Prediksi akhir RF ditentukan berdasarkan kelas yang paling banyak dipilih oleh seluruh pohon.

$$\hat{y} = \arg\max_{c} \sum_{t=1}^{T} \mathbf{1}[h_t(x) = c] \tag{2.7}$$

   Dengan T adalah jumlah total pohon, $h_t(x)$ adalah prediksi pohon ke-t untuk sampel x, dan $\mathbf{1}[\cdot]$ adalah fungsi indikator bernilai 1 jika kondisi terpenuhi [29].

4. *Feature Importance*

   Tingkat kepentingan fitur X_j dihitung berdasarkan rata-rata penurunan *Gini Impurity* yang disebabkan oleh fitur tersebut di seluruh pohon dan seluruh node.

$$FI(X_j) = \frac{1}{T} \sum_{t=1}^{T} \sum_{s \in S_t(X_j)} \Delta Gini(s) \tag{2.8}$$

   Dengan $S_t(X_j)$ adalah himpunan semua node pada pohon t yang melakukan *split* menggunakan fitur X_j, dan $\Delta Gini(s)$ adalah penurunan *Gini Impurity* pada node s.

Untuk memberikan gambaran menyeluruh mengenai alur kerja algoritma, keseluruhan tahapan pelatihan *Random Forest* mulai dari pengambilan sampel *bootstrap* hingga pembentukan keputusan akhir melalui *majority voting* dirangkum dalam bentuk *pseudocode* pada Kode 2.1.

```
ALGORITHM: Random_Forest(D, T, m)
INPUT:
    - D = {(xi, yi)} untuk i = 1,...,n (dataset latih)
```

```
    - T : jumlah pohon dalam ensemble
    - m : jumlah fitur acak yang dievaluasi pada tiap split (m << p)
OUTPUT:
    - H(x) : fungsi prediksi akhir ensemble
BEGIN
    FOR t = 1 TO T DO
        // 1. Bootstrap sampling
        Dt <- ambil n sampel acak dari D dengan pengembalian

        // 2. Bangun pohon keputusan ht dari Dt
        REPEAT untuk tiap node yang belum murni:
            pilih m fitur secara acak dari total p fitur
            tentukan fitur & titik split terbaik (Gini Impurity minimum)
            pecah node menjadi dua child node
        UNTIL kriteria henti terpenuhi

        simpan pohon ht
    END FOR

    // 3. Agregasi prediksi seluruh pohon (majority voting)
    H(x) <- argmax_c SUM(t=1..T) indikator(ht(x) = c)

    RETURN H(x)
END
```

Kode 2.1: *Pseudocode* pelatihan algoritma Random Forest

2.7 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) adalah algoritma *ensemble learning* berbasis *boosting* yang dikembangkan oleh Chen dan Guestrin (2016), bekerja dengan membangun *decision tree* secara sekuensial, dengan setiap pohon baru dilatih untuk memperbaiki kesalahan residual dari pohon sebelumnya [33]. Berbeda dengan *Random Forest* yang membangun pohon secara paralel dan independen, XGBoost secara adaptif memusatkan perhatian pada sampel yang sulit diklasifikasikan di setiap iterasi, sehingga menghasilkan model yang semakin akurat seiring bertambahnya jumlah pohon [33, 34]. XGBoost juga dilengkapi dengan mekanisme regularisasi L1 dan L2 yang mencegah *overfitting*, serta mendukung *parallel processing* yang membuatnya jauh lebih efisien secara

komputasi dibandingkan algoritma *boosting* konvensional [34]. Cara kerja XGBoost dapat dijabarkan melalui empat rumus utama sebagai berikut.

1. Fungsi Objektif

   XGBoost meminimalkan fungsi objektif yang terdiri dari *loss function* dan regularisasi.

$$\mathscr{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{2.9}$$

   Dengan $l(y_i, \hat{y}_i)$ adalah *loss function* antara nilai aktual y_i dan prediksi $\hat{y}_i$, $\Omega(f_k)$ adalah regularisasi pohon ke-k, dan K adalah jumlah total pohon.

2. Prediksi Akhir XGBoost

   Prediksi dibangun secara *additive* dan iteratif, yaitu model pada setiap iterasi memperbarui prediksi sebelumnya dengan menambahkan kontribusi pohon baru yang telah diskalakan oleh laju pembelajaran (*learning rate*) η.

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta\, f_t(x_i) \tag{2.10}$$

   Setelah seluruh K iterasi selesai, prediksi akhir merupakan akumulasi prediksi awal $\hat{y}_i^{(0)}$ beserta kontribusi seluruh pohon.

$$\hat{y}_i = \hat{y}_i^{(0)} + \eta \sum_{k=1}^{K} f_k(x_i) \tag{2.11}$$

   Dengan $f_k(x_i)$ adalah prediksi pohon ke-k untuk sampel x_i, sedangkan η merupakan laju pembelajaran yang mengendalikan besar kontribusi tiap pohon guna menekan risiko *overfitting*. Nilai η yang kecil membuat proses pembelajaran lebih bertahap sehingga model cenderung lebih stabil.

3. *Gradient* dan *Hessian*

   Pada setiap iterasi, XGBoost menggunakan ekspansi Taylor orde kedua untuk mengaproksimasi *loss function*, dengan *gradient* g_i dan *hessian* h_i sebagai berikut.

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}), \quad h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \tag{2.12}$$

Dengan g_i adalah turunan pertama dan h_i adalah turunan kedua dari *loss function* terhadap prediksi pada iterasi sebelumnya.

4. *Gain* (Pemilihan *Split*)

   XGBoost memilih fitur terbaik untuk *split* berdasarkan nilai *Gain* tertinggi.

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{2.13}$$

   Dengan G_L, G_R adalah jumlah *gradient* pada node kiri dan kanan, H_L, H_R adalah jumlah *hessian* pada node kiri dan kanan, λ adalah regularisasi L2, dan γ adalah ambang batas minimum *gain* untuk melakukan *split*.

Keseluruhan proses pelatihan XGBoost, mulai dari inisialisasi, perhitungan *gradient* dan *hessian*, pembentukan pohon berdasarkan *gain*, hingga pembaruan model dengan *learning rate*, dirangkum pada Kode 2.2.

```
ALGORITHM: XGBoost(D, K, eta, lambda, gamma)
INPUT:
    - D = {(xi, yi)} untuk i = 1,...,n (dataset latih)
    - K      : jumlah pohon (iterasi)
    - eta    : learning rate
    - lambda : regularisasi L2
    - gamma  : ambang minimum gain untuk split
OUTPUT:
    - F(x) : fungsi prediksi akhir ensemble
BEGIN
    // 1. Inisialisasi prediksi awal dengan nilai konstan
    F0(x) <- nilai konstan awal

    FOR t = 1 TO K DO
        // 2. Hitung gradient & hessian tiap sampel dari prediksi
   sebelumnya
        FOR i = 1 TO n DO
            gi <- turunan pertama loss terhadap prediksi ke-(t-1)
            hi <- turunan kedua loss terhadap prediksi ke-(t-1)
        END FOR
```

```
20
21          // 3. Bangun pohon ft dengan memaksimalkan Gain
22          REPEAT untuk tiap node:
23              hitung Gain tiap kandidat split
24              pilih split dengan Gain tertinggi
25              JIKA Gain < 0 hentikan percabangan (pruning)
26          UNTIL kriteria henti terpenuhi
27
28          // 4. Perbarui model dengan learning rate
29          Ft(x) <- F(t-1)(x) + eta * ft(x)
30      END FOR
31      RETURN F(x) = FK(x)
32  END
```

Kode 2.2: *Pseudocode* pelatihan algoritma XGBoost

2.8 Stacking Ensemble

Stacking ensemble atau *stacked generalization* adalah teknik *ensemble learning* yang menggabungkan prediksi dari beberapa model *base learner* yang heterogen melalui sebuah model tingkat kedua yang disebut *meta-learner*, dengan tujuan menghasilkan prediksi akhir yang lebih optimal dibandingkan model tunggal maupun teknik *ensemble* sederhana seperti *bagging* dan *boosting* [35]. Berbeda dengan *bagging* dan *boosting* yang menggunakan model-model homogen, *stacking* secara sengaja memilih *base learner* yang beragam dan saling melengkapi agar *meta-learner* dapat mempelajari cara terbaik menggabungkan kekuatan masing-masing model [30, 35].

Mekanisme *stacking* terdiri dari dua level utama. Pada Level 0, setiap *base learner* dilatih secara independen menggunakan data latih dan menghasilkan prediksi yang kemudian dijadikan fitur baru (*meta-features*). Pada Level 1, *meta-learner* dilatih menggunakan *meta-features* tersebut untuk menghasilkan prediksi akhir [35]. Dalam penelitian ini, *stacking ensemble* dibangun dengan *Random Forest* dan XGBoost sebagai *base learner* pada Level 0. Kedua model tersebut dipilih karena saling komplementer, *Random Forest* berbasis *bagging* yang membangun pohon secara paralel, sementara XGBoost berbasis *boosting* yang membangun pohon secara sekuensial. Sementara itu, *Logistic Regression* digunakan sebagai *meta-learner* pada Level 1 karena kemampuannya menghasilkan probabilitas kelas yang terkalibrasi dengan baik pada tugas klasifikasi *multiclass*

[16, 30, 35]. Penjelasan lebih lanjut mengenai *Logistic Regression* sebagai *meta-learner* diuraikan pada subbab berikutnya.

Cara kerja *stacking ensemble* secara formal dapat dijabarkan melalui dua rumus utama sebagai berikut.

1. *Meta-features*

   Misalkan terdapat K buah *base learner* $h_1, h_2, \ldots, h_K$ pada Level 0. Setiap *base learner* h_k menghasilkan prediksi terhadap sampel x, dan gabungan seluruh prediksi tersebut membentuk vektor *meta-features* $z(x)$ yang menjadi masukan bagi *meta-learner*.

   $$z(x) = [h_1(x), h_2(x), \ldots, h_K(x)] \tag{2.14}$$

   Pada penelitian ini, *base learner* yang digunakan adalah *Random Forest* dan XGBoost, sehingga *meta-features* tersusun dari probabilitas kelas yang dihasilkan oleh kedua model tersebut.

2. Prediksi Akhir Stacking

   Meta-learner g dilatih menggunakan *meta-features* $z(x)$ untuk menghasilkan prediksi akhir.

   $$\hat{y} = g\big(z(x)\big) = g\big(h_1(x), h_2(x), \ldots, h_K(x)\big) \tag{2.15}$$

   Dengan g merupakan *meta-learner* yang pada penelitian ini menggunakan *Logistic Regression*. Agar prediksi *base learner* yang menjadi *meta-features* tidak berasal dari data yang sama dengan data pelatihannya sendiri, prediksi Level 0 dibentuk melalui skema validasi silang (*cross-validation*), sehingga potensi kebocoran informasi (*data leakage*) dapat dihindari.

Keseluruhan alur pelatihan *stacking ensemble*, mulai dari pembentukan *meta-features* melalui validasi silang hingga pelatihan *meta-learner*, dirangkum pada Kode 2.3.

```
ALGORITHM: Stacking_Ensemble(D, base_learners, meta_learner, V)
INPUT:
    - D = {(xi, yi)} untuk i = 1,...,n (dataset latih)
```


Klasifikasi Gangguan Tidur..., Fikri Naufal Andrasito, Universitas Multimedia Nusantara

```
    - base_learners = {h1, ..., hK} (model Level 0)
    - meta_learner  = g (model Level 1)
    - V : jumlah fold cross-validation
OUTPUT:
    - model stacking terlatih
BEGIN
    // 1. Bentuk meta-features via cross-validation (hindari data
   leakage)
    bagi D menjadi V fold
    FOR tiap fold v = 1 TO V DO
        latih tiap base learner pada (D tanpa fold v)
        prediksi fold v -> simpan sebagai meta-features z pada
   baris fold v
    END FOR

    // 2. Latih meta-learner pada meta-features
    Z <- gabungan seluruh meta-features z(xi)
    latih meta_learner g pada (Z, y)

    // 3. Latih ulang seluruh base learner pada keseluruhan D
    FOR tiap base learner hk DO
        latih hk pada seluruh D
    END FOR
    RETURN stacking(base_learners, meta_learner)
END
```

Kode 2.3: *Pseudocode* pelatihan Stacking Ensemble

2.9 Logistic Regression

Berbeda dengan algoritma Random Forest dan XGBoost yang bertugas sebagai *base learner*, penelitian ini menugaskan Logistic Regression sebagai *meta-learner* pada tahap akhir dari arsitektur *stacking*. Algoritma ini dipilih untuk menebak seberapa besar peluang sebuah data masuk ke dalam kategori tertentu berdasarkan perhitungan kombinasi fitur yang ada. Jika target yang diprediksi hanya terdiri dari dua kategori, Logistic Regression menggunakan fungsi *sigmoid* untuk mengubah hasil perhitungannya menjadi nilai peluang berupa rentang angka 0 hingga 1. Rumus fungsi *sigmoid* ini dituliskan sebagai berikut.

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{dengan} \quad z = w^T x + b \tag{2.16}$$

Berdasarkan rumus tersebut, w adalah nilai bobot, x adalah fitur masukan, b adalah nilai bias, dan z adalah hasil gabungan perhitungan linear dari keseluruhannya. Karena penelitian ini memiliki tiga kategori target gangguan tidur (No Disorder, Sleep Apnea, dan Insomnia), rumus *sigmoid* tersebut harus diperluas menggunakan fungsi *softmax*. Fungsi ini memungkinkan model untuk langsung menghitung seberapa besar peluang sebuah data masuk ke masing-masing dari ketiga kategori tersebut secara bersamaan. Rumusnya adalah sebagai berikut.

$$P(y = k|x) = \frac{e^{w_k^T x}}{\sum_{j=1}^{C} e^{w_j^T x}} \tag{2.17}$$

Pada persamaan di atas, C mewakili total jumlah kelas yang ada, w_k adalah bobot untuk kelas ke-k, dan $P(y = k|x)$ adalah angka peluang yang menyatakan seberapa mungkin data x tersebut benar-benar merupakan anggota dari kelas k. Pada akhirnya, kelas dengan nilai peluang tertinggilah yang akan ditetapkan sebagai tebakan akhir dari model. Untuk bisa memberikan prediksi yang akurat, model akan terus belajar memperbaiki nilai bobot w tersebut dengan cara menekan tingkat kesalahannya sekecil mungkin melalui metode optimasi matematika tertentu.

Ada beberapa alasan kuat mengapa Logistic Regression sangat cocok digunakan sebagai *meta-learner*. Pertama, hasil prediksi dari Random Forest dan XGBoost sebelumnya sudah berupa angka peluang (probabilitas). Karena Logistic Regression secara alami memang dirancang untuk memproses angka peluang, algoritma ini bisa langsung menggabungkan hasil dari kedua model tersebut tanpa memerlukan penyesuaian format yang rumit. Kedua, sebagai model yang cara kerjanya cukup sederhana, Logistic Regression memiliki risiko yang jauh lebih kecil untuk terjebak menghafal data latih (*overfitting*) saat menerima jumlah fitur yang sedikit. Pendekatan ini telah terbukti sangat efektif pada studi kasus medis lain yang juga sukses menggabungkan Random Forest dan XGBoost menggunakan Logistic Regression [36]. Ketiga, algoritma ini terbukti mampu menghasilkan tebakan peluang yang sangat stabil dan terukur, sehingga hasil akhir prediksi penyakit gangguan tidur pada penelitian ini menjadi jauh lebih meyakinkan dan bisa dipertanggungjawabkan kebenarannya.

2.10 Evaluasi Model Klasifikasi

Evaluasi model merupakan tahap krusial dalam *pipeline machine learning* untuk mengukur sejauh mana model yang dibangun mampu melakukan generalisasi terhadap data yang belum pernah dilihat sebelumnya [37]. Dasar dari seluruh metrik evaluasi klasifikasi adalah *confusion matrix*, yaitu tabel yang merangkum hasil prediksi model dengan membandingkan kelas yang diprediksi terhadap kelas aktual; setiap sel merepresentasikan jumlah sampel yang termasuk dalam kombinasi kelas aktual dan kelas prediksi tertentu [37]. Dari *confusion matrix* diturunkan empat nilai dasar, yaitu *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN), yang menjadi fondasi perhitungan metrik *accuracy*, *precision*, *recall*, dan *F1-score* [22, 37].

Dalam konteks klasifikasi *multiclass* seperti pada penelitian ini yang mengklasifikasikan tiga kelas gangguan tidur, yaitu *No Disorder*, *Sleep Apnea*, dan *Insomnia*, penggunaan *accuracy* secara tunggal tidak cukup untuk merepresentasikan performa model. Hal ini disebabkan *accuracy* dapat menyesatkan pada data dengan distribusi kelas yang tidak seimbang, karena model yang cenderung memihak kelas mayoritas tetap dapat memperoleh nilai *accuracy* yang tinggi meskipun kemampuannya mengenali kelas minoritas rendah [38]. Oleh karena itu, digunakan pendekatan *weighted averaging* pada setiap metrik guna memperhitungkan ketidakseimbangan distribusi antar kelas, dilengkapi dengan metrik *Area Under the ROC Curve* (ROC-AUC) yang lebih tahan terhadap ketidakseimbangan kelas [38], serta skema validasi silang *StratifiedKFold* dengan 5 *fold* untuk memastikan setiap *fold* mempertahankan proporsi kelas yang representatif [12, 22, 37].

Cara kerja masing-masing metrik evaluasi dijabarkan melalui rumus-rumus berikut.

1. *Confusion Matrix*

   Untuk klasifikasi *multiclass* dengan c kelas, *confusion matrix* berukuran $c \times c$ dengan elemen CM_{ij} merepresentasikan jumlah sampel kelas aktual i yang diprediksi sebagai kelas j.

$$CM = \begin{bmatrix} CM_{11} & \cdots & CM_{1c} \\ \vdots & \ddots & \vdots \\ CM_{c1} & \cdots & CM_{cc} \end{bmatrix} \tag{2.18}$$

Klasifikasi Gangguan Tidur..., Fikri Naufal Andrasito, Universitas Multimedia Nusantara

Nilai diagonal CM_{ii} merepresentasikan prediksi yang benar untuk kelas i, sementara nilai di luar diagonal merepresentasikan kesalahan klasifikasi.

2. *Accuracy*

   Accuracy mengukur proporsi sampel yang diklasifikasikan dengan benar dari seluruh sampel.

$$Accuracy = \frac{\sum_{i=1}^{c} CM_{ii}}{\sum_{i=1}^{c}\sum_{j=1}^{c} CM_{ij}} = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.19}$$

   Dengan c adalah jumlah kelas. Perlu dicatat bahwa pada data dengan kelas yang tidak seimbang, nilai *accuracy* yang tinggi tidak selalu mencerminkan performa yang baik pada seluruh kelas, sehingga metrik ini perlu dilengkapi dengan metrik lain seperti *precision*, *recall*, *F1-score*, dan ROC-AUC [38].

3. *Precision* (Weighted)

   Precision mengukur proporsi prediksi positif yang benar-benar positif, dihitung secara *weighted* untuk setiap kelas.

$$Precision_{weighted} = \sum_{i=1}^{c} w_i \cdot \frac{CM_{ii}}{\sum_{j=1}^{c} CM_{ji}} \tag{2.20}$$

   Dengan $w_i = \frac{n_i}{n}$ adalah bobot kelas i berdasarkan jumlah sampelnya, n_i adalah jumlah sampel kelas i, dan n adalah total sampel.

4. *Recall* (Weighted)

   Recall mengukur proporsi sampel positif aktual yang berhasil diidentifikasi dengan benar.

$$Recall_{weighted} = \sum_{i=1}^{c} w_i \cdot \frac{CM_{ii}}{\sum_{j=1}^{c} CM_{ij}} \tag{2.21}$$

   Dengan w_i adalah bobot kelas i yang sama dengan perhitungan *weighted precision*.

5. *Weighted F1-Score*

   Weighted F1-Score adalah rata-rata harmonik tertimbang antara *precision* dan *recall* untuk setiap kelas.

$$F1_{weighted} = \sum_{i=1}^{c} w_i \cdot \frac{2 \times Precision_i \times Recall_i}{Precision_i + Recall_i} \tag{2.22}$$

   Dengan w_i adalah bobot kelas i berdasarkan jumlah sampelnya.

6. *Area Under the ROC Curve* (ROC-AUC)

   Kurva *Receiver Operating Characteristic* (ROC) menggambarkan kemampuan model dalam membedakan antar kelas dengan memplot *True Positive Rate* (TPR) terhadap *False Positive Rate* (FPR) pada berbagai ambang batas keputusan (*threshold*). ROC-AUC menyatakan luas area di bawah kurva tersebut, dengan nilai yang lebih tinggi menunjukkan kemampuan diskriminasi model yang lebih baik.

$$TPR = \frac{TP}{TP+FN}, \qquad FPR = \frac{FP}{FP+TN} \tag{2.23}$$

   Karena ROC-AUC pada dasarnya dirumuskan untuk klasifikasi biner, penerapannya pada klasifikasi *multiclass* tiga kelas pada penelitian ini menggunakan strategi *One-vs-Rest* (OvR), yaitu menghitung ROC-AUC untuk setiap kelas dengan memperlakukan satu kelas sebagai kelas positif dan kelas lainnya sebagai kelas negatif. Mengingat distribusi kelas pada penelitian ini tidak seimbang, hasil ROC-AUC tiap kelas kemudian digabungkan secara *weighted*, sejalan dengan pendekatan *weighted averaging* yang digunakan pada metrik *Precision*, *Recall*, dan F1-*Score*.

$$AUC_{OvR} = \sum_{i=1}^{c} w_i \cdot AUC_i \tag{2.24}$$

   Dengan AUC_i adalah nilai ROC-AUC ketika kelas i diperlakukan sebagai kelas positif terhadap gabungan kelas lainnya, dan $w_i = \frac{n_i}{n}$ adalah bobot kelas i berdasarkan jumlah sampelnya, sama dengan pembobotan pada *Precision* dan *Recall*. ROC-AUC dipilih sebagai pelengkap metrik evaluasi karena

sifatnya yang relatif konsisten dan tidak terlalu terpengaruh oleh perubahan distribusi kelas, sehingga cocok digunakan pada penelitian ini yang datanya tidak seimbang antar kelas [38].

7. *StratifiedKFold Cross-Validation*

   Pada *StratifiedKFold* dengan *k fold*, dataset D dibagi menjadi k subset yang masing-masing mempertahankan proporsi kelas yang sama dengan dataset asli. Performa akhir dihitung sebagai rata-rata dari k iterasi.

$$\overline{Acc} = \frac{1}{k}\sum_{i=1}^{k} Acc_i \tag{2.25}$$

   Dengan Acc_i adalah akurasi pada iterasi ke-i, dan $k = 5$.

2.11 Penelitian Terdahulu

Penelitian-penelitian terdahulu yang berkaitan dengan klasifikasi gangguan tidur menggunakan *machine learning* telah memberikan kontribusi penting dalam membangun fondasi ilmiah bagi penelitian ini. Berbagai pendekatan algoritma telah diujicobakan, mulai dari model tunggal seperti *Random Forest* dan XGBoost, hingga metode *ensemble* yang menggabungkan beberapa model sekaligus. Tinjauan terhadap penelitian-penelitian tersebut disajikan pada Tabel 2.1 berikut.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Tabel 2.1. Penelitian Terdahulu

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
1	Classification of Sleep Disorders using Random Forest on Sleep Health and Lifestyle Dataset	Hidayat (2023)	Kaggle *Sleep Health and Lifestyle* (374 sampel, 3 kelas)	*Random Forest*	88,00%	RF berhasil mengklasifikasikan 3 kelas (*No Disorder*, *Insomnia*, *Sleep Apnea*) pada dataset gaya hidup, namun tidak membandingkan dengan algoritma *ensemble* lain dan tidak menangani ketidakseimbangan kelas secara eksplisit.
2	Predictive Power of XGBOOST BILSTM Model: A Machine-Learning Approach for Accurate Sleep Apnea Detection using Electronic Health Data	Javeed et al. (2023)	Data rekam medis elektronik (*EHR*) — deteksi biner *Sleep Apnea*	XGBoost, BiLSTM	98,50%	Kombinasi XGBoost dan BiLSTM mengungguli metode tradisional pada data klinis EHR. Performa tinggi disebabkan oleh kekayaan fitur klinis PSG; tidak dapat dibandingkan langsung dengan dataset gaya hidup.
Lanjut ke halaman berikutnya...						

Tabel 2.1 *Tabel Lanjutan Penelitian Terdahulu*

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
3	Potential of Machine Learning for Predicting Sleep Disorders: A Comprehensive Analysis of Regression and Classification Models	Alazaidah et al. (2023)	Kaggle *Sleep Health and Lifestyle* + dataset tambahan (multiclass)	RF, SVM, KNN, *Decision Tree*, *Logistic Regression*, *Naive Bayes*	92,00%	Evaluasi komprehensif 29 model klasifikasi menunjukkan RF terbaik dengan akurasi 92%. Penelitian ini tidak menggunakan SMOTE sehingga hasil bisa bias terhadap kelas mayoritas; tidak mengeksplorasi *stacking ensemble*.
4	Sleep Quality Prediction from Wearables using Convolution Neural Networks and Ensemble Learning	Kilic et al. (2023)	*NetHealth* — data *wearable* (prediksi kualitas tidur biner)	CNN, *Ensemble Learning* (RF, XGBoost, *Voting*)	82,40%	Model *ensemble* secara konsisten mengungguli model tunggal pada data *wearable*. XGBoost dan RF menjadi kontributor terbaik. Tugas prediksi kualitas tidur berbeda dari klasifikasi jenis gangguan tidur.

Lanjut ke halaman berikutnya...

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 2.1 *Tabel Lanjutan Penelitian Terdahulu*

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
5	Classification of Sleeping Position using Enhanced Stacking Ensemble Learning	Xu et al. (2024)	Dataset posisi tidur berbasis sensor akselerometer (klasifikasi posisi, bukan gangguan)	*Stacking Ensemble* (XGBoost + LR *meta-learner*)	94,48%	*Stacking* yang menggabungkan XGBoost melampaui seluruh model *ensemble* pembanding. Meskipun domainnya berbeda (posisi tidur), penelitian ini membuktikan efektivitas *stacking* pada klasifikasi multiclass domain tidur.
6	Identification of Sleep Disorder Using Machine Learning	Saini & Chandel (2024)	Kaggle *Sleep Health and Lifestyle* (3 kelas)	*Decision Tree*, RF, SVM, KNN, *Naive Bayes*, Logistik	89,00%	RF mencapai performa tertinggi di antara enam algoritma yang diuji. Tidak menggunakan teknik *oversampling* dan tidak mengeksplorasi *stacking ensemble* maupun *hyperparameter tuning* terstruktur.
Lanjut ke halaman berikutnya...						

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 2.1 *Tabel Lanjutan Penelitian Terdahulu*

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
7	Insomnia Detection using Machine Learning	Bhartia et al. (2025)	Dataset deteksi *insomnia* biner (bukan multiclass)	LR, *Decision Tree*, RF, SVM, XGBoost	93,00%	XGBoost menunjukkan akurasi tertinggi pada deteksi *insomnia* biner. Studi terbatas pada dua kelas sehingga tidak mencerminkan kompleksitas klasifikasi multiclass yang mencakup *Sleep Apnea* dan *No Disorder* secara bersamaan.
8	Sleep Disorder Prediction System Using Machine Learning	Islam et al. (2025)	Kaggle *Sleep Health and Lifestyle* (3 kelas)	RF, XGBoost, SVM, KNN, LR, *Naive Bayes*	91,70%	RF menghasilkan akurasi tertinggi 91,7% di antara enam algoritma. Penelitian ini tidak mengeksplorasi penggabungan model melalui *stacking ensemble* dan tidak melaporkan metrik F1 per kelas secara rinci.

Lanjut ke halaman berikutnya...

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 2.1 *Tabel Lanjutan Penelitian Terdahulu*

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
9	Predicting Sleep Disorders Using Machine Learning: A Comparative Analysis	Goodyear & Alsammani (2025)	Kaggle *Sleep Health and Lifestyle* (3 kelas)	LR, *Decision Tree*, RF, KNN, *Naive Bayes*, SVM, *Linear Regression*	94,38%	RF mencapai performa terbaik dengan *precision* 0,96 dan AUC 0,98. Fitur dominan meliputi usia, BMI, dan tekanan darah. Tidak menggunakan *feature engineering* maupun *stacking ensemble*.
10	A Comprehensive Study of Sleep Disorder Classification using XGBoost Algorithm	Muthulingam et al. (2025)	Kaggle *Sleep Health and Lifestyle* (3 kelas)	XGBoost	96,67%	XGBoost menunjukkan performa sangat tinggi pada dataset tunggal Kaggle. Tidak ada pembanding dengan model lain maupun *stacking*, dan tidak ada penanganan ketidakseimbangan kelas yang dilaporkan secara eksplisit.

Lanjut ke halaman berikutnya...

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 2.1 *Tabel Lanjutan Penelitian Terdahulu*

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
11	Machine Learning with Ensemble Stacking Model for Automated Sleep Staging using Dual-Channel EEG Signal	Satapathy et al. (2021)	Sleep-EDF dan Sleep Expanded-EDF (sinyal EEG dual-channel, 5 kelas *sleep staging*)	*Stacking Ensemble* (RF, SVM, *Naive Bayes*, KNN sebagai *base learner*)	91,10%	Model *stacking ensemble* mencapai akurasi 91,10% pada *sleep staging* 5 kelas dari sinyal EEG. Usia terbukti sebagai fitur diskriminatif penting. Meskipun berbeda domain (EEG, bukan gaya hidup), penelitian ini membuktikan efektivitas *stacking* pada klasifikasi tidur multiclass.
12	Application of Various Machine Learning Techniques to Predict Obstructive Sleep Apnea Syndrome Severity	Han & Oh (2023)	Data klinis 4.014 pasien Samsung Medical Center (kuesioner, pengukuran fisik, PSG — 4 kelas *severity* OSAS)	XGBoost, LightGBM, CatBoost, RF + *clustering* + *Bayesian hyperparameter tuning*	91,11%	LightGBM terbaik untuk prediksi *severity* OSAS dengan *feature engineering* medis dan *clustering*. Dataset klinis yang kaya menghasilkan performa tinggi; tidak dapat dibandingkan langsung dengan dataset gaya hidup. Menunjukkan pentingnya *feature engineering* pada data tidur.

Lanjut ke halaman berikutnya...

Tabel 2.1 *Tabel Lanjutan Penelitian Terdahulu*

No	Judul	Penulis	Dataset	Metode	Akurasi	Hasil Ringkasan
13	Advanced Sleep Disorder Detection using Multi-Layered Ensemble Learning and Advanced Data Balancing Techniques	Monowar et al. (2025)	Kaggle *Sleep Health and Lifestyle* (374 sampel, 3 kelas)	*Multi-layer Ensemble*: RF, SVM, KNN, XGBoost + *Voting Classifier* + SMOTE	96,88%	*Ensemble* berlapis mencapai akurasi 96,88% dengan SMOTE. Namun, SMOTE diterapkan sebelum *train-test split* sehingga berpotensi *data leakage*. Dataset identik dengan DS2 penelitian ini, menjadikannya pembanding langsung yang paling relevan.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA