BAB 2
LANDASAN TEORI

Pada penelitian ini digunakan beberapa teori dan metode yaitu Penyakit Jantung, Machine Learning, Supervised Learning, Ensemble Learning, Boosting, LightGBM, dan Confusion Matrix.

2.1 Penyakit Jantung

Penyakit Jantung adalah sebuah kondisi penyakit pada pembuluh darah utama ketika ingin menyuplai darah ke jantung. Penyakit jantung yang terjadi terdapat beberapa macam jenis yaitu penyakit jantung koroner, kebocoran katup jantung dan gagal jantung. Penyakit jantung koroner adalah kondisi dimana aliran darah ke jantung terhalang oleh penumpukan lemak di dinding pembuluh darah. Gagal jantung, di sisi lain, terjadi ketika jantung tidak mampu memompa darah dengan efisien, mengakibatkan gangguan dalam fungsi jantung. Kebocoran katup jantung, yang juga menjadi penyebab umum masalah jantung, terjadi ketika katup jantung tidak dapat berfungsi sebagaimana mestinya, seringkali akibat dari perkembangan jantung yang tidak normal [26].

2.2 *Machine Learning*

Machine Learning adalah bagian dari kecerdasan buatan (*Artificial Intelligence*) yang melatih algoritma komputer untuk membuat klasifikasi dan prediksi berdasarkan pola dalam data[27]. Pada machine learning pengalaman berupa informasi-informasi yang telah tersedia yang akan dijadikan informasi untuk menyelesaikan permasalahan kedepannya. Pada pembelajaran mesin terbagi menjadi beberapa metode seperti *Supervised learning*, *Unsupervised learning* dan *Reinforcement Learning*[28]. Pada penelitian ini menggunakan algoritma supervised learning.

2.3 *Supervised Learning*

Supervised learning adalah pendekatan fundamental dalam bidang pembelajaran mesin di mana model-model dilatih menggunakan data yang telah diberi label. Metode ini melibatkan sebuah algoritma belajar dari data input

yang ditandai dengan keluaran yang benar, memungkinkan model untuk menilai akurasinya dan menyesuaikannya sesuai kebutuhan[29]. Algoritma-algoritma *supervised learning* meningkatkan kinerjanya dengan membandingkan prediksi dengan label sebenarnya yang diberikan oleh seorang ahli manusia, mengasah kemampuannya untuk membuat prediksi yang akurat pada data baru yang belum terlihat[30]. Berbeda dengan *self-supervised* dan *unsupervised learning*, *supervised learning* bergantung pada kumpulan contoh yang diberi label secara manual, yang dapat mahal dan memakan waktu untuk diproduksi[31]. *Supervised learning* secara tradisional difokuskan pada pembelajaran induktif, di mana model mengamati contoh-contoh yang diberi label untuk mempelajari sebuah tugas, tetapi menghadapi tantangan seperti kebutuhan akan dataset yang diberi label secara ekstensif dan potensi *overfitting* terhadap data pelatihan[32].

2.4 *Ensemble Learning*

Ensemble learning adalah paradigma pembelajaran mesin di mana *multiple model*, seperti *base learners* atau *classifiers*, digabungkan secara strategis untuk meningkatkan akurasi prediksi dan kinerja generalisasi. Metode *ensemble* meningkatkan kinerja dengan menggabungkan *output* dari *multiple classifier* untuk mengurangi varians keseluruhan sehingga dapat meningkatkan akurasi [14]. *Ensemble learning* memiliki teknik seperti *bagging*, *boosting*, dan *stacking* umum digunakan untuk pembentukan *ensemble* [16]. Penerapan *ensemble learning* yang dipilih pada penelitian ini adalah *boosting*.

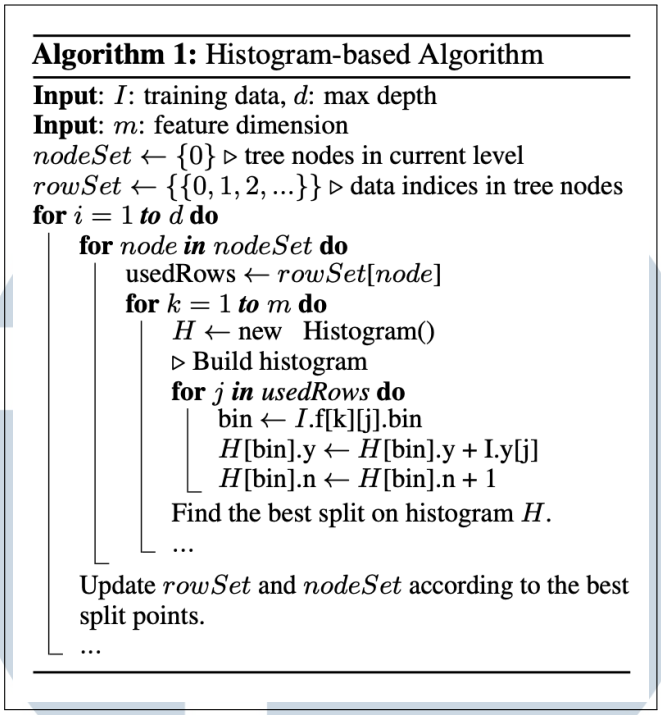
2.5 *Boosting*

Boosting merupakan teknik dalam *ensemble learning* yang dapat menggabungkan model-model yang memiliki performa yang lemah yang digunakan secara berurutan untuk meningkatkan kinerja prediksi atau klasifikasi secara bertahap. Teknik ini mempelajari klasifikasi yang diberikan, cukup dengan menggunakan aturan pembelajaran yang cukup tidak akurat (disebut "hipotesis lemah"), yang kemudian secara otomatis dikumpulkan berdasarkan algoritma peningkatan menjadi algoritma yang akurat[17]. Algoritma *boosting* terdiri dari AdaBoost, Gradient Booting, XGBoost, CatBoost, and LightGBM[18]. Algoritma *boosting* yang digunakan pada penelitian ini adalah LightGBM.

2.6 LightGBM

LightGBM adalah sebuah kerangka kerja peningkatan gradien yang menggunakan algoritma pembelajaran berbasis pohon, dirancang untuk didistribusikan dan efisien dengan kecepatan pelatihan yang lebih cepat, efisiensi yang lebih tinggi, dan kemampuan untuk menangani data berskala besar[33]. Algoritma ini melakukan komputasi *parallel* seperti *feature parallel*, *data parallel* dan *voting parallel* yang bertujuan untuk meningkatkan efisiensi komputasi[21, 2].

Pada sebuah algoritma *Gradient Boost Decision Tree* (GBDT) yang menjadi biaya utama terletak pada pembelajaran pohon keputusan, dan bagian yang paling memakan waktu dalam mempelajari pohon keputusan adalah menemukan titik pemisahan terbaik. Salah satu algoritma yang paling populer untuk menemukan titik pisah adalah algoritma *Pre-Sorted* yang menghitung semua kemungkinan titik pisah pada nilai fitur yang telah diurutkan sebelumnya. Algoritma ini sederhana dan dapat menemukan titik pemisahan yang optimal, namun tidak efisien dalam kecepatan pelatihan dan konsumsi memori. Algoritma populer lainnya adalah algoritma berbasis histogram seperti yang ditunjukkan pada Gambar 2.1. Algoritma berbasis histogram memasukkan nilai fitur berkelanjutan ke dalam wadah terpisah dan menggunakan wadah ini untuk membuat histogram fitur selama pelatihan. Karena algoritma berbasis histogram lebih efisien dalam konsumsi memori dan kecepatan pelatihan. Seperti yang ditunjukkan pada Gambar 2.1 yaitu algoritma berbasis histogram menemukan titik pemisahan terbaik berdasarkan histogram fitur. Biayanya O(data $\times$ feature) untuk pembuatan histogram dan O(bin $\times$ feature) untuk menemukan titik pisah. Karena bin biasanya jauh lebih kecil daripada data, pembuatan histogram akan mendominasi kompleksitas komputasi.



Algorithm 1: Histogram-based Algorithm

Input: I: training data, d: max depth
Input: m: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ ***to*** d **do**
  for $node$ ***in*** $nodeSet$ **do**
    usedRows $\leftarrow rowSet[node]$
    for $k = 1$ ***to*** m **do**
      $H \leftarrow$ new Histogram()
      ▷ Build histogram
      for j ***in*** *usedRows* **do**
        bin $\leftarrow I$.f[k][j].bin
        H[bin].y $\leftarrow H$[bin].y + I.y[j]
        H[bin].n $\leftarrow H$[bin].n + 1
      Find the best split on histogram H.
      ...
  Update $rowSet$ and $nodeSet$ according to the best split points.
  ...

Gambar 2.1. Algoritma *Histogram-based* [1]

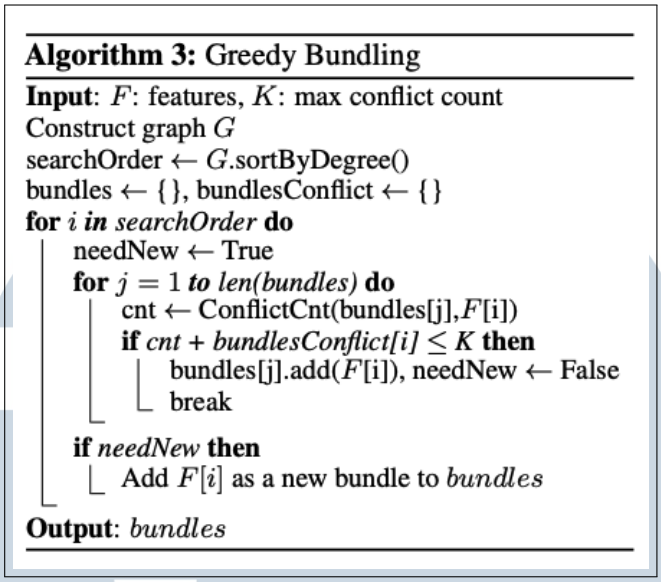
Ada dua teknik dalam *LightGBM* yaitu *Gradient-based One-Side Sampling* (GOSS) dan *Exclusive Feature Bundling* (EFB), yang dikembangkan untuk mengatasi tantangan dari jumlah sampel data yang besar dan fitur yang kompleks. GOSS secara cerdas menyimpan contoh-contoh dengan gradien yang signifikan, sementara melakukan pengambilan sampel secara acak pada contoh-contoh dengan gradien yang rendah. GOSS menyimpan semua instance dengan gradien besar dan melakukan pengambilan sampel acak pada instance tersebut dengan gradien kecil. Untuk mengkompensasi pengaruh terhadap distribusi data, saat menghitung perolehan informasi, GOSS memperkenalkan pengali konstan untuk contoh data dengan gradien kecil seperti pada Gambar 2.2. Secara khusus, GOSS pertama-tama mengurutkan contoh data berdasarkan nilai absolutnya gradien dan memilih instance a × 100% teratas. Kemudian secara acak mengambil sampel b × 100% instance dari sisa datanya. Setelah itu, GOSS memperkuat data sampel dengan gradien kecil secara konstan 1a/b saat menghitung perolehan informasi *gain*. Dengan melakukan hal ini, lebih fokus pada data yang kurang terlatih tanpa banyak mengubah distribusi data asli.

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models ← {}, fact ← $\frac{1-a}{b}$
topN ← a × len(I) , randN ← b × len(I)
for $i = 1$ ***to*** d **do**
    preds ← models.predict(I)
    g ← $loss$(I, preds), w ← {1,1,...}
    sorted ← GetSortedIndices(abs(g))
    topSet ← sorted[1:topN]
    randSet ← RandomPick(sorted[topN:len(I)], randN)
    usedSet ← topSet + randSet
    w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data.
    newModel ← L(I[usedSet], − g[usedSet], w[usedSet])
    models.append(newModel)

Gambar 2.2. Algoritma *Gradient Based One Side Sampling* (GOSS)[1]

Di sisi lain, algoritma EFB mampu menggabungkan banyak fitur eksklusif ke dalam sebuah kelompok fitur yang lebih padat, mengurangi perhitungan yang tidak perlu untuk fitur-fitur dengan nilai nol. LightGBM, yang merupakan algoritma berbasis histogram, menyederhanakan fitur-fitur kontinu menjadi nilai-nilai diskrit, memungkinkan pelatihan yang lebih cepat dengan efisiensi yang lebih tinggi serta penggunaan memori yang lebih hemat[18, 2].



Gambar 2.3. *Greedy Bundling* [1]

Pada gambar 2.3 merupakan alur sebuah untuk *bundling* fitur eksklusif. Pertama, dibangun sebuah graf dengan tepi yang diberi bobot, di mana bobotnya sesuai dengan konflik total antara fitur. Kedua, fitur-fitur diurutkan berdasarkan derajat mereka dalam grafik secara menurun. Terakhir, setiap fitur dalam daftar yang diurutkan diperiksa, dan *diassign* ke bundel yang sudah ada dengan konflik kecil (dikendalikan oleh γ), atau dibuat bundel baru. Kompleksitas waktu dari algoritma ini adalah $O(\#\text{ feature }^2)$ dan ini diproses hanya sekali sebelum pelatihan. Kompleksitas ini dapat diterima ketika jumlah fitur tidak terlalu besar, namun mungkin masih mengalami kendala jika terdapat jutaan fitur. Untuk meningkatkan efisiensi lebih lanjut, diusulkan strategi pengurutan yang lebih efisien tanpa membangun grafik: pengurutan berdasarkan jumlah nilai non-nol, yang serupa dengan pengurutan berdasarkan derajat karena lebih banyak nilai non-nol biasanya menghasilkan probabilitas konflik yang lebih tinggi. Karena hanya mengubah strategi pengurutan dalam algoritma tersebut, detail dari algoritma baru tersebut dihilangkan untuk menghindari duplikasi [1].

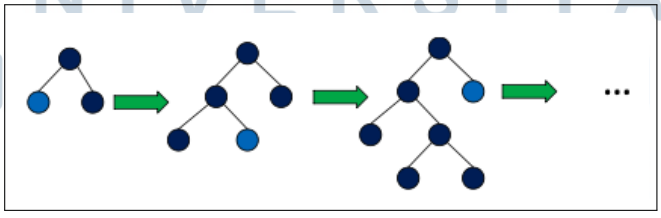
Algorithm 4: Merge Exclusive Features

Input: $numData$: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for f ***in*** F **do**
    totalBin += f.numBin
    binRanges.append(totalBin)
newBin ← new Bin(numData)
for $i = 1$ ***to*** $numData$ **do**
    newBin[i] ← 0
    for $j = 1$ ***to*** *len(F)* **do**
        if $F[j].bin[i] \neq 0$ **then**
            newBin[i] ← F[j].bin[i] + binRanges[j]

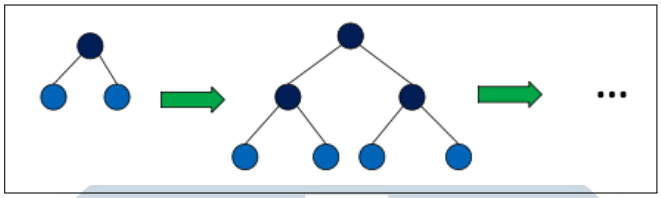
Output: $newBin, binRanges$

Gambar 2.4. *Merge Exclusive Features* [1]

Tahap selanjutnya, dibutuhkan pendekatan yang efektif untuk menggabungkan fitur dalam bundel yang sama untuk mengurangi kompleksitas pelatihan seperti pada gambar 2.4. Hal yang paling penting adalah memastikan bahwa nilai fitur asli masih dapat diidentifikasi dalam bundel fitur. Karena algoritma berbasis histogram menyimpan bin diskrit daripada nilai fitur kontinu, kita dapat membuat bundel fitur dengan menempatkan fitur eksklusif di bin yang berbeda. Ini dapat dilakukan dengan menambahkan pergeseran ke nilai asli dari fitur. Misalnya, pertimbangkan dua fitur dalam satu bundel fitur. Awalnya, fitur A memiliki rentang dari [0, 10), sedangkan fitur B memiliki rentang [0, 20). Dengan menambahkan pergeseran sebesar 10 ke nilai fitur B, fitur yang disesuaikan sekarang memiliki rentang dari [10, 30). Akibatnya, penggabungan fitur A dan B menjadi memungkinkan, menggunakan bundel fitur dengan rentang [0, 30) untuk menggantikan fitur asli A dan B [1].



Gambar 2.5. *Leaf-wise tree growth in LightGBM* [2]



Gambar 2.6. *Level-wise tree growth in other boosting algorithms* [2]

Salah satu karakteristik yang membuat algoritma LightGBM berbeda dari algoritma boosting pohon lainnya adalah cara pembagian pohon yang dilakukan secara *"leafwise"*, seperti yang ditunjukkan dalam Gambar 2.5 dengan pemisahan terbaik, sedangkan algoritma *boosting* lainnya membagi pohon secara *"depthwise"* atau *"levelwise"*, seperti yang ditunjukkan dalam Gambar 2.6, bukan *"leafwise"*. Ketika tumbuh di daun yang sama dalam LightGBM, algoritma *"leafwise"* dapat mengurangi *loss* lebih banyak daripada algoritma *"levelwise"* dan, oleh karena itu, menghasilkan akurasi yang jauh lebih baik, yang tidak terpenuhi oleh algoritma boosting lainnya [2].

2.7 *Confusion Matrix*

Kinerja sebuah model dapat dievaluasi menggunakan nilai yang didapatkan oleh *confusion matrix*.

	Predicted positive	Predicted negative
Actual positive	True positives TP	False negatives FN
Actual negative	False positives FP	True negatives TN

Gambar 2.7. *Confusion Matrix* 2x2 [3]

Hasil yang didapatkan seperti pada tabel 2.7 berupa jumlah *True Positives* (TP), *True Negatives* (TN), *False Positives* (FP), dan *False Negatives* (FN) yang terkait dengan kelas yang diprediksi [3, 34].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

$$F\text{-}Measure = 2 * \frac{PrecisionRecall}{Precision + Recall} \tag{2.4}$$

Dari hasil *confusion matrix* tersebut dapat dihasilkan beberapa *evaluation mectric* yaitu *Accuracy*, *Precision*, *Recall* dan *F1-Score*. *Accuracy* menunjukkan proporsi prediksi pengklasifikasi yang benar (kasus positif dan negatif), dan semakin dekat nilainya ke 1, semakin baik klasifikasinya. *Precision* menunjukkan proporsi sampel yang diprediksi oleh pengklasifikasi yang termasuk dalam kelas positif, dan semakin dekat nilainya ke 1, semakin baik klasifikasinya. *Recall* menunjukkan proporsi kategori dengan benar diprediksi positif oleh pengklasifikasi, dan semakin mendekati satu nilainya maka semakin baik pula hasil klasifikasinya. *F1-score* adalah metrik evaluasi yang menggabungkan presisi (precision) dan recall dalam satu nilai tunggal. Selain akurasi yang membandingkan klasifikasi positif dan negatif didapatkan metode lain yang bisa digunakan AUC dan ROC. AUC didefinisikan sebagai luas di bawah kurva ROC. Kurva ROC adalah kurva karakteristik operasi penerima, apabila area di bawah kurva semakin luas maka semakin baik pengklasifikasi yang dilakukan[20].