BAB 2
TINJAUAN PUSTAKA

2.1 Berita Hoaks

Berita hoaks di Indonesia didefinisikan sebagai penyebaran informasi palsu yang sengaja dibuat untuk menyesatkan dan membingungkan masyarakat, dengan tujuan mendapatkan keuntungan ekonomi atau politik. Fenomena ini sangat mengkhawatirkan di Indonesia karena memiliki potensi besar untuk mengganggu kehidupan sosial dan stabilitas politik dalam masyarakat yang multikultural. Penyebaran berita hoaks sering kali ditujukan untuk memecah belah kelompok tertentu berdasarkan agama, ras, atau etnis [18].

Berita hoaks memiliki beberapa karakteristik, seperti informasi yang sengaja disamarkan untuk mengaburkan kebenaran dan memanipulasi persepsi publik [19]. Selain itu, berita hoaks sering menyajikan konten yang tampak kredibel namun tidak memiliki sumber yang dapat diverifikasi. Dampak dari penyebaran berita palsu ini dapat menciptakan ketakutan, kebingungan, dan memecah belah masyarakat, yang pada akhirnya dapat memicu kerusuhan sosial [20]. Rendahnya literasi media di Indonesia juga memperburuk situasi ini, karena banyak orang kesulitan untuk menilai secara kritis dan membedakan antara berita palsu dan berita yang benar [18].

2.2 Machine Learning

Machine Learning (ML) merupakan subbidang dari kecerdasan buatan (AI) yang berfokus pada kemampuan komputer untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit. Dalam proses ini, ML menggunakan algoritma untuk mengidentifikasi pola dan struktur dalam data, yang kemudian digunakan untuk melakukan prediksi atau pengambilan keputusan berdasarkan informasi tersebut [21]. Hal ini menjadikan ML sebagai alat yang sangat berguna untuk berbagai aplikasi, karena memungkinkan komputer untuk melakukan analisis data secara otomatis dan mendalam.

Pada intinya, ML adalah metode analisis data yang memungkinkan komputer belajar dari data dan membangun model prediktif [21]. Algoritma ML secara otomatis belajar dan mengenali pola dari data, yang kemudian dapat digunakan untuk pengambilan keputusan di berbagai skenario. Proses ini sering kali

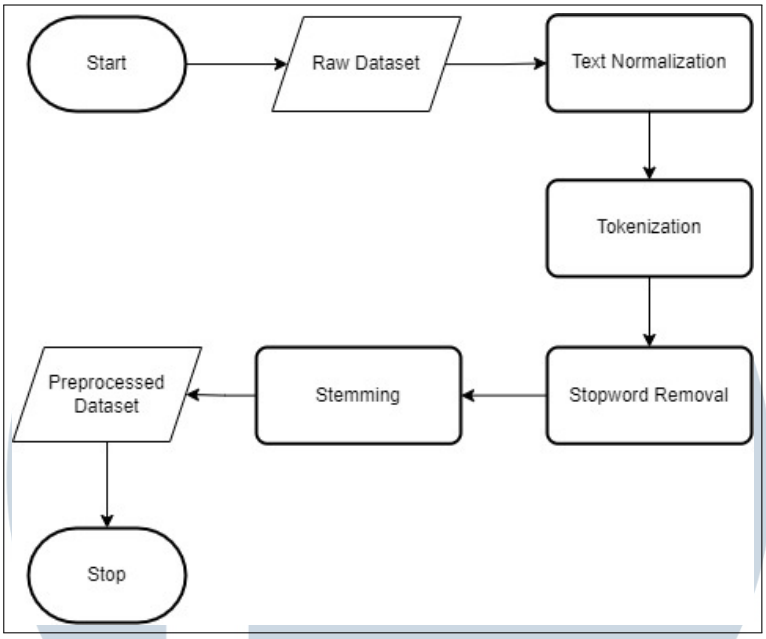
melibatkan model statistik yang digunakan untuk menemukan pola atau struktur dalam data, yang pada akhirnya diformulasikan sebagai prediksi untuk pengamatan baru [22]

Selain itu, penerapan ML sangat luas dan mencakup berbagai bidang. Dalam bidang teknik kimia, misalnya, ML digunakan untuk memprediksi energi biomassa, mengestimasi viskositas kinematik bahan bakar minyak, serta mengklasifikasikan plastik limbah [23]. Di bidang medis, ML membantu dalam pembuatan model prediktif adaptif untuk hasil klinis pasien, meskipun akurasi model ini sangat bergantung pada validasi eksternal [24]. ML juga berperan penting dalam bidang oseanografi, seperti dalam memprediksi kadar nitrogen tetap global, mendeteksi bentuk plankton, serta mengklasifikasikan mamalia laut berdasarkan data akustik [25].

Namun, terdapat sejumlah tantangan yang perlu dipertimbangkan dalam penerapan ML. Salah satunya adalah fenomena ”black box,” di mana proses di balik prediksi model sulit untuk diinterpretasikan [23]. Selain itu, model ML sering kali memerlukan jumlah data yang besar untuk pelatihan, yang dapat menjadi hambatan dalam situasi dengan keterbatasan data [23].

2.3 Text Pre-processing

Text preprocessing merupakan tahapan awal dalam analisis teks yang bertujuan untuk mempersiapkan data sebelum dilakukan analisis lebih lanjut. Langkah ini bertujuan untuk meningkatkan kualitas data sehingga dapat digunakan secara optimal dalam berbagai model analisis teks seperti klasifikasi, *sentiment analysis*, dan lain-lain. Berikut adalah tahapan-tahapan utama dalam proses *text preprocessing* yang sering digunakan dalam penelitian analisis teks.



Gambar 2.1. *Preprocessing*

2.3.1 Normalisasi Teks

Normalisasi Teks adalah proses menyederhanakan dan menstandarkan teks menjadi bentuk yang lebih seragam, sehingga memudahkan analisis lebih lanjut. Proses ini mencakup beberapa langkah, di antaranya adalah case folding, penghapusan tanda baca, penghapusan angka, dan perbaikan kata tidak baku.

A Case Folding

Case folding bertujuan untuk mengubah seluruh huruf dalam teks menjadi huruf kecil. Langkah ini dilakukan untuk memastikan konsistensi dalam teks sehingga perbedaan kapitalisasi tidak memengaruhi proses analisis. Misalnya, kata ”Jakarta”, ”JAKARTA”, dan ”jakarta” diubah menjadi ”jakarta”. Menurut Yaakov *et al.* (2020), case folding efektif untuk mengurangi redundansi dalam data dan memastikan bahwa model tidak terpengaruh oleh perbedaan kapitalisasi [26].

B Penghapusan Tanda Baca dan Simbol

Penghapusan tanda baca, angka, simbol, dan karakter khusus dilakukan untuk memastikan bahwa teks hanya berisi informasi yang relevan. Langkah ini

bertujuan untuk mengurangi gangguan yang dapat memengaruhi hasil analisis, terutama pada model berbasis statistik sederhana.

C Perbaikan Kata Tidak Baku

Kata-kata yang tidak baku seperti ”gk”, ”nggak”, atau ”tdk” sering ditemukan dalam teks mentah. Proses normalisasi ini mengganti kata-kata tersebut dengan bentuk baku, seperti ”tidak”. Menurut Algan *et al.* (2022), perbaikan kata tidak baku penting untuk meningkatkan akurasi model [27].

2.3.2 Tokenisasi

Tokenisasi adalah proses pemecahan teks menjadi unit-unit yang lebih kecil, seperti kata, frasa, atau kalimat. Proses ini merupakan tahapan penting dalam *text preprocessing* karena memengaruhi representasi data yang digunakan oleh model. Algan *et al.* (2022) menekankan bahwa tokenisasi yang tepat sangat penting untuk meningkatkan akurasi model secara keseluruhan [27].

2.3.3 Penghapusan Stop-word

Stop-words adalah kata-kata umum seperti ”dan”, ”atau”, ”yang” yang tidak memiliki makna signifikan dalam konteks analisis teks. Teknik ini bertujuan untuk fokus pada kata-kata yang lebih bermakna dalam teks, sehingga dapat meningkatkan akurasi analisis data. Hacohen-Kerner *et al.* (2020) menunjukkan bahwa penghapusan stop-words sering mengarah pada peningkatan yang signifikan dalam tugas klasifikasi teks [26].

2.3.4 Stemming

Stemming adalah teknik yang bertujuan untuk mengurangi kata-kata menjadi bentuk dasarnya atau akarnya. Proses ini dilakukan dengan memotong akhiran atau imbuhan dari suatu kata untuk mendapatkan bentuk dasar, meskipun bentuk tersebut tidak selalu merupakan kata yang valid secara linguistik. Menurut Algan *et al.* (2022), stemming membantu menyederhanakan data teks yang diolah dan meningkatkan pemahaman model terhadap konteks teks, sehingga mempermudah proses analisis lebih lanjut [27].

2.4 Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF (*Term frequency-inverse document frequency*) adalah algoritma yang dikembangkan dari metode DF. Algoritma ini menggunakan pendekatan statistik untuk menilai seberapa penting suatu kata dalam sebuah kumpulan dokumen. Pentingnya sebuah kata dihitung berdasarkan frekuensinya dalam dokumen tertentu dan sebaliknya berkurang jika kata tersebut sering muncul di banyak dokumen lain dalam kumpulan tersebut[28]. Frekuensi istilah-TF ($TF_{i,j}$) menunjukkan seberapa sering suatu kata w_i muncul dalam dokumen tertentu x_j. Hal ini dapat dihitung menggunakan rumus berikut:

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2.1}$$

Di mana $n_{i,j}$ adalah jumlah kemunculan kata w_i dalam dokumen x_j, sementara penyebutnya adalah jumlah total seluruh kata yang muncul dalam dokumen x_j. Sementara itu, *inverse document frequency* (IDF) mengukur seberapa unik suatu kata w_i di antara dokumen-dokumen dalam korpus. IDF dihitung dengan membagi jumlah total dokumen dengan jumlah dokumen yang mengandung kata tersebut, lalu mengambil logaritma dari hasilnya. Rumus IDF dituliskan sebagai berikut:

$$IDF_i = \log\left(\frac{|D|}{|\{j : w_i \in x_j\}| + 1}\right) \tag{2.2}$$

Dalam rumus ini, $|D|$ adalah total jumlah dokumen dalam korpus, sedangkan $|\{j : w_i \in x_j\}|$ adalah jumlah dokumen yang memuat kata w_i. Penambahan angka 1 pada penyebut bertujuan untuk menghindari pembagian dengan nol. Kombinasi dari nilai TF dan IDF menghasilkan skor TF-IDF untuk sebuah kata w_i, yang dihitung dengan rumus berikut:

$$(TF\text{-}IDF)_{w_i} = TF_{i,j} \times IDF_i \tag{2.3}$$

Dengan demikian, nilai TF-IDF mencerminkan pentingnya sebuah kata dalam dokumen tertentu, sekaligus memperhitungkan relevansinya dalam keseluruhan korpus.

2.5 Decision Tree

Pohon keputusan adalah algoritma pembelajaran terawasi yang secara rekursif mempartisi data menjadi subset berdasarkan nilai atribut yang berbeda hingga kriteria pemberhentian terpenuhi. Struktur seperti pohon yang dihasilkan terdiri dari simpul yang merepresentasikan keputusan atau pemisahan berdasarkan atribut tertentu [29]. Prosedur pembelajaran pohon keputusan melibatkan serangkaian langkah di mana data dibagi menjadi subset yang homogen, dengan simpul akar mewakili seluruh dataset dan simpul daun mewakili hasil atau label kelas.

Efektivitas pohon keputusan sangat bergantung pada pilihan kriteria atau aturan pemisahan yang digunakan untuk menentukan di mana pohon harus membuat pemisahan pada simpulnya. Beberapa kriteria pemisahan yang umum digunakan meliputi:

A Indeks Gini

Indeks Gini, juga disebut Ketidakmurnian Gini, mengukur probabilitas sampel yang dipilih secara acak salah diklasifikasikan jika diberi label secara acak [29]. Indeks Gini dihitung sebagai:

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2 \tag{2.4}$$

di mana S adalah himpunan sampel, n adalah jumlah kelas unik dalam himpunan, dan p_i adalah proporsi sampel dalam himpunan yang termasuk dalam kelas i.

B Information Gain

Information Gain didasarkan pada konsep entropi dari teori informasi. Metode ini menentukan efektivitas suatu atribut dalam memisahkan data pelatihan menjadi himpunan yang homogen [29]. Information Gain untuk pemisahan pada dataset S dengan atribut A dihitung sebagai:

$$IG = E(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} E(S_v) \tag{2.5}$$

di mana $E(S)$ adalah entropi dari himpunan S, $Values(A)$ adalah nilai-nilai

berbeda yang dapat diambil oleh atribut A, dan S_v adalah subset dari S di mana atribut A memiliki nilai v.

Salah satu algoritma *Decision Tree* adalah Algoritma ID3, diperkenalkan oleh Quinlan [30], melalui pendekatan *greedy search* dari atas ke bawah untuk membangun pohon keputusan. Algoritma ini memilih atribut yang memaksimalkan Information Gain pada setiap simpul. Algoritma ID3 dijelaskan dalam Algoritma 1.

Algorithm 1 ID3 Decision Tree Algorithm

Require: Training data set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$
Ensure: Decision tree T

function ID3(D)
    if D is empty **then**
        return a terminal node with default class $c_{default}$
    end if
    if all instances in D have the same class label y **then**
        return a terminal node with class y
    end if
    if the attribute set J is empty **then**
        return a terminal node with the prevalent class in D
    end if
    Select the feature f that best splits the data using information gain
    Create a decision node for f
    for all value b_i of f **do**
        Create a branch for b_i
        Let D_i be the subset of D where $x_i = b_i$
        Recursively build the subtree for D_i
        Attach the subtree to the branch for b_i
    end for
    return the decision node
end function

2.6 *Ensemble Learning*

Ensemble learning merupakan pendekatan pembelajaran mesin yang menggabungkan beberapa algoritma untuk mencapai kinerja prediksi yang lebih baik dibandingkan menggunakan algoritma tunggal. Berdasarkan [31], metode ensemble learning bekerja dengan cara menghasilkan hasil prediksi lemah dari beberapa algoritma pembelajaran mesin berdasarkan fitur yang diekstrak melalui berbagai proyeksi data, kemudian menggabungkan hasil-hasil tersebut

menggunakan berbagai mekanisme voting. Kunci keunggulan dari metode ensemble adalah kemampuannya untuk secara efektif menangkap berbagai karakteristik dan struktur data yang mendasar yang mungkin sulit dimodelkan oleh algoritma individual.

Pendekatan ini membantu mengatasi tantangan dalam menangani jenis data yang kompleks, seperti dataset yang tidak seimbang, berdimensi tinggi, dan mengandung noise. Melalui integrasi fusi data, pemodelan, dan penggalian dalam satu kerangka kerja terpadu, ensemble learning dapat mencapai akurasi, stabilitas, dan generalisasi yang lebih unggul dibandingkan pendekatan model tunggal tradisional [31]. Metode *bagging* dan *boosting* adalah contoh penerapan *ensemble learning* yang umum digunakan.

2.6.1 *Bagging*

Metode bootstrap aggregating (bagging) dikembangkan pada tahun 1994 oleh Breiman [32] untuk meningkatkan performa klasifikasi model machine learning dengan menggabungkan prediksi dari dataset training yang dibuat secara acak. Breiman berpendapat bahwa mengubah learning set dapat menyebabkan modifikasi signifikan pada prediktor yang dihasilkan; oleh karena itu bagging dapat meningkatkan akurasi [32]. Bagging mendapatkan keragaman dengan membuat replika bootstrap dari data input, di mana beberapa subset data input dipilih secara acak dengan penggantian dari dataset training asli. Prosedur bagging ditunjukkan pada Algoritma 2 [33].

Pada dasarnya, metode bagging melibatkan pembagian data training untuk setiap base learner menggunakan sampling acak untuk menghasilkan b subset berbeda yang digunakan untuk melatih b base learner. Base learner tersebut kemudian digabungkan menggunakan majority voting untuk mendapatkan classifier yang kuat [34].

Algorithm 2 Bagging (Bootstrap Aggregating)

Input: Training data $S = (x_1, y_1), ..., (x_2, y_2), ..., (x_n, y_n)$
Input: Base ML algorithm L
Input: Number of base learners T
Output: Ensemble model H
for $t = 1, ..., T$ **do**
    Generate bootstrap sample S_j from input data S
    Train base learner h_j using S_j, i.e., $h_j = L(S_j)$
end for
Output: Combine outputs of base learners
$H(x) = mode(h_1(x), ..., h_T(x))$
return H

Keuntungan utama bagging adalah kemampuannya mengurangi varians tanpa meningkatkan bias. Untuk dataset besar, bagging memiliki waktu komputasi lebih singkat karena melatih model dengan ukuran sampel kecil [35]. Namun, keterbatasannya adalah bagging meningkatkan akurasi model tanpa memperhatikan interpretabilitas. Misalnya, jika hanya satu 'tree' yang digunakan sebagai base learner, diagram 'tree' yang sesuai dan mudah diinterpretasi dapat diperoleh; karenanya, interpretabilitas diabaikan karena bagging menggunakan banyak 'decision tree'.

2.6.2 *Boosting*

Boosting pertama kali diperkenalkan dalam paper tahun 1990 oleh Schapire [36]. Boosting merupakan teknik machine learning yang mampu mengonversi weak learner menjadi strong classifier. Ini adalah jenis ensemble meta-algoritma yang digunakan untuk mengurangi bias dan varians. Weak learner adalah classifier yang kinerjanya sedikit lebih baik dari random guessing, sedangkan strong learner adalah classifier yang mencapai akurasi yang baik [37].

Ide utama di balik boosting melibatkan penerapan algoritma pembelajaran dasar secara iteratif pada versi data input yang disesuaikan [38]. Secara khusus, teknik boosting menggunakan data input untuk melatih weak learner, menghitung prediksi dari weak learner tersebut, memilih sampel training yang salah diklasifikasikan, dan melatih weak learner berikutnya dengan dataset training yang disesuaikan yang terdiri dari instance yang salah diklasifikasikan dari putaran

training sebelumnya [39].

A *AdaBoost*

AdaBoost adalah algoritma boosting yang menggunakan *weak learner* untuk membentuk *strong classifier*, dikembangkan oleh Freund dan Schapire di tahun 1995[40]. Dalam proses pembelajaran AdaBoost, *classifier* dasar dilatih menggunakan algoritma dasar seperti pohon keputusan. Bobot sampel disesuaikan berdasarkan prediksi *classifier* dan digunakan untuk melatih *classifier* berikutnya. Sampel yang salah diklasifikasikan diberi bobot lebih besar, sedangkan yang benar diberi bobot lebih kecil.

Berbagai *learner* ditambahkan secara berurutan dan diberi bobot untuk mendapatkan *strong classifier* [41]. Pada setiap iterasi, AdaBoost menetapkan bobot untuk setiap *instance* dalam set pelatihan [42]. Misalkan terdapat *m instance* pelatihan berlabel $S = (x_1, y_1), ..., (x_i, y_i), ..., (x_m, y_m)$ dengan y_i adalah label target dari sampel x_i dan $y_i \in Y = -1, +1$, maka bobot awal D_1 dari sampel x_i dan pembaruan bobot D_{t+1} dihitung dengan persamaan 2.6 dan 2.7.

$$D_1(i) = \frac{1}{n}, i = 1, 2, ..., m \tag{2.6}$$

$$D_{t+1}(i) = \frac{D_t(i)}{z_t} \exp(-\alpha_t y_i h_t(x_i)), i = 1, 2, ..., \tag{2.7}$$

$h_t(x)$ adalah *classifier* dasar, $t = 1, ..., T$ adalah jumlah iterasi, Z_t adalah faktor normalisasi, dan α_t adalah bobot dari *classifier* $h_t(x)$. α_t mengukur pentingnya $h_t(x)$ dalam prediksi akhir. *Instance* yang diprediksi salah oleh $h_t(x)$ akan memiliki bobot lebih besar pada iterasi $t + 1$. Z_t dipilih agar D_{t+1} menjadi distribusi. Perhitungan Z_t dan α_t mengikuti persamaan 2.8 dan 2.9.

$$Z_t = \sum_{t+1}^{n} D_t(i) \exp(-\alpha_t y_i h_t(x_i)) \tag{2.8}$$

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \varepsilon_t}{\varepsilon_t}) \tag{2.9}$$

ε_t adalah tingkat kesalahan dari *classifier*, dihitung menggunakan persamaan 2.10:

$$\varepsilon_t = P[h_t(x_i) \neq y_i] = \sum_{i=n}^{n} D_i(i) I[h_t(x_i) \neq y_i] \tag{2.10}$$

Setelah semua iterasi selesai, *strong classifier* akhir $H(x)$ didapatkan dengan menggabungkan prediksi dari seluruh *classifier* dasar seperti pada persamaan 2.11.

$$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{2.11}$$

Algoritma AdaBoost dirangkum dalam Algoritma 3 [33].

Algorithm 3 AdaBoost Algorithm

Input: training data $S = (x_1, y_1), ..., (x_2, y_2), ..., (x_m, y_m)$
The base algorithm L
The number of iterations T
Procedure:
Initialize the weight D_1 of sample x_i using (2.6)
for $t = 1, ..., T$ **do**
1) Train the base classifier $h_t(x)$ by minimizing ε_t
2) Calculate the weight α_t of the classifier using (2.9)
3) Update the sample weights using (2.7)
end for
Output: Apply (2.11) to combine the predictions of the base classifiers to obtain the final strong classifier $H(x)$

AdaBoost relatif mudah diimplementasikan dan *hyperparameter*-nya tidak banyak yang perlu disetel [43]. AdaBoost juga fleksibel karena dapat menggunakan beragam algoritma sebagai *learner* dasar sesuai kebutuhan aplikasi. Namun, AdaBoost sensitif terhadap *noise* data dan *outlier* akibat proses pembelajaran iteratifnya yang dapat menyebabkan *overfitting*.

2.7 *Gradient Boosting Decision Tree*

Gradient Boosting merupakan algoritma pembelajaran mesin yang mengkombinasikan beberapa base learner secara sekuensial untuk membentuk model yang kuat, dengan pohon keputusan sebagai base learner yang umum digunakan. Konsep ini pertama kali diperkenalkan oleh Breiman [44] yang memperlihatkan bahwa boosting dapat direpresentasikan sebagai masalah optimasi fungsi kerugian. Selanjutnya, Friedman [45] mengembangkan versi yang lebih komprehensif yang menjadi dasar implementasi Gradient Boosting modern.

Secara matematis, algoritma Gradient Boosting dapat dijelaskan sebagai berikut: Dengan data pelatihan $S = x_i, y_i{}_1^N$, algoritma ini berusaha memperoleh fungsi aproksimasi $\hat{F}(x)$ terhadap fungsi target $F^*(x)$ yang memetakan input x ke output y, dengan meminimalkan fungsi kerugian $L(y, F(x))$. Model dibangun secara iteratif seperti ditunjukkan pada persamaan 2.12.

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \tag{2.12}$$

dimana ρ_m adalah bobot dari fungsi $h_m(x)$ ke-m. Fungsi-fungsi tersebut adalah pohon keputusan dalam ensemble. Algoritma melakukan aproksimasi secara iteratif. Aproksimasi awal konstan $F_0(x)$ diperoleh menggunakan persamaan 2.13:

$$F_0(x) = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i, \alpha) \tag{2.13}$$

Base learner berikutnya h_m bertujuan untuk meminimalkan:

$$(\rho_m, h_m) = \arg\min_{\rho, h} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i)) \tag{2.14}$$

Setiap h_m dilatih dengan dataset baru $D = x_i, r_{mi}{}_{i=1}^N$, di mana r_{mi} adalah pseudo-residual, dihitung sebagai:

$$r_{mi} = \left[\frac{\delta L(y_i, F(x_i))}{\partial F(x_i)} \right] F(x) = Fm - 1(x) \tag{2.15}$$

Regularisasi pada Gradient Boosting umumnya dilakukan melalui shrinkage $F_m(x) = F_{m-1}(x) + \nu\rho_m h_m(x)$, dengan ν bernilai kecil sekitar 0.1, serta subsampling data pelatihan pada setiap iterasi. Algoritma 4[33] menunjukkan prosedur lengkap Gradient Boosting.

Algorithm 4 Gradient Boosting

Input: training set $S = (x_i, y_i)_{i=1}^{N}$,
differentiable loss function $L(y, F(x))$
number of iterations T
Algorithm:
1) Initialize model with a constant value:
$F_0 = argmin_\alpha \sum_{i=1}^{N} L(y_i, \alpha)$
2) **for** $m = 1, ..., M$:
(i) Calculate the false residuals:
$r_{mi} = [\frac{\partial L(y_i, F(x))}{\partial F(x)}]_{F(x)=F_{m-1}(x)}$ for $i = 1, ..., n$
(ii) Train a base learner using the training set:
$D = x_i, r_{mi}{}_{i=1}^{N}$
(iii) Obtain ρ_m by performing line search optimization:
$(\rho_m h_m(x)) = argmin_{\rho,h} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i))$
(iv) Update the model:
$F_m(x) = F_{m-1}(x) + \rho_m h_m(x)$
end for
Output: Return the final model $F_m(x)$

Gradient Boosting unggul dalam kemampuannya mempelajari pola kompleks melalui penyesuaian residual dari model sebelumnya. Meski demikian, algoritma ini rentan terhadap overfitting pada data yang mengandung noise jika tidak diregularisasi dengan tepat [46, 47]. Penggunaan optimal algoritma ini adalah pada dataset berukuran kecil hingga menengah [48].

2.8 Light Gradient Boosting Machine (LightGBM)

LightGBM merupakan pengembangan signifikan dalam pohon keputusan gradient boosting (GBDT), khususnya dalam mengatasi tantangan komputasi pada pemrosesan dataset skala besar [49]. Sebagai metode pembelajaran ensemble, LightGBM membangun kesuksesan gradient boosting sambil memperkenalkan optimasi baru untuk meningkatkan kecepatan dan efisiensi [33]. Algoritma ini memperkenalkan dua inovasi utama: Gradient-based One-Side Sampling (GOSS) dan Exclusive Feature Bundling (EFB), yang secara bersamaan mengurangi waktu pelatihan secara signifikan sambil mempertahankan akurasi model.

2.8.1 *Histogram Based Algorithm*

Dasar dari LightGBM adalah pendekatan berbasis histogram untuk menemukan titik split optimal dalam pohon keputusan. Algoritma ini meningkatkan pendekatan pre-sorted tradisional dengan mendiskritisasi fitur kontinu menjadi bin, menghasilkan pengurangan konsumsi memori dan kompleksitas komputasi. Algoritma histogram dasar dapat dilihat pada Algoritma 5.

Algorithm 5 Histogram-based Algorithm

Require: Training data I, max depth d, feature dimension m
Ensure: Trained decision tree

```
nodeSet ← 0                                   ▷ tree nodes in current level
rowSet ← 0,1,2,...                            ▷ data indices in tree nodes
for i = 1 to d do
    for node in nodeSet do
        usedRows ← rowSet[node]
        for k = 1 to m do
            H ← new Histogram()
            for j in usedRows do
                bin ← I.f[k][j].bin
                H[bin].y ← H[bin].y + I.y[j]
                H[bin].n ← H[bin].n + 1
            end for
            Find best split on histogram H
        end for
        Update rowSet and nodeSet based on best splits
    end for
end for
```

2.8.2 *Gradient-based One-Side Sampling* (GOSS)

GOSS memperkenalkan pendekatan baru untuk sampling data yang mempertahankan akurasi model sambil mengurangi biaya komputasi. Berbeda dengan metode sampling acak tradisional yang digunakan dalam teknik ensemble lainnya [33], GOSS secara selektif mempertahankan instance berdasarkan nilai gradiennya. Algoritma GOSS dapat dilihat pada Algoritma 6.

Algorithm 6 Gradient-based One-Side Sampling

Require: Training data I, iterations d, sampling ratio a for large gradient data, sampling ratio b for small gradient data

Ensure: Trained model

```
models ←
fact ← (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
    newModel ← L(I[usedSet], -g[usedSet], w[usedSet])
    models.append(newModel)
end for
```

Estimasi information gain dalam GOSS dihitung sebagai:

$$\tilde{V}j(d) = \frac{1}{n}\left(\frac{(\sum x_i \in A_l g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n_r^j(d)}\right) \quad (2.16)$$

dimana:

1. A_l dan A_r adalah instance dengan gradien besar di node kiri dan kanan
2. B_l dan B_r adalah instance dengan gradien kecil di node kiri dan kanan
3. $n_l^j(d)$ dan $n_r^j(d)$ adalah jumlah instance di node kiri dan kanan
4. a dan b adalah rasio sampling untuk instance gradien besar dan kecil

2.8.3 *Exclusive Feature Bundling* (EFB)

EFB menangani tantangan data sparse berdimensi tinggi dengan menggabungkan fitur yang saling eksklusif. Prosesnya melibatkan dua algoritma utama:

Algorithm 7 Greedy Bundling

Require: Features F, max conflict count K
Ensure: Feature bundles

```
Construct graph G
searchOrder ← G.sortByDegree()
bundles ←
bundlesConflict ←
for i in searchOrder do
    needNew ← True
    for j = 1 to len(bundles) do
        cnt ← ConflictCnt(bundles[j], F[i])
        if cnt + bundlesConflict[i] ≤ K then
            bundles[j].add(F[i])
            needNew ← False
            break
        end if
    end for
    if needNew then
        Add F[i] as new bundle to bundles
    end if
end for
```

Setelah identifikasi bundle, fitur-fitur digabungkan menggunakan Algoritma 8.

Algorithm 8 Merge Exclusive Features

Require: numData, bundle of exclusive features F

Ensure: Merged feature bundle

```
binRanges ← 0
totalBin ← 0
for f in F do
    totalBin += f.numBin
    binRanges.append(totalBin)
end for
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] ≠ 0 then
            newBin[i] ← F[j].bin[i] + binRanges[j]
        end if
    end for
end for
```

LightGBM mengintegrasikan optimasi ini ke dalam kerangka gradient boosting, yang telah terbukti sangat efektif dalam aplikasi pembelajaran ensemble [33]. Fungsi objektif keseluruhan mencakup term regularisasi untuk mencegah overfitting:

$$L_M(F(x_i)) = \sum_{i=1}^{n} L(y_i, F(x_i)) + \sum_{m=1}^{M} \Omega(h_m) \tag{2.17}$$

dimana $\Omega(h_m)$ adalah term regularisasi:

$$\Omega(h) = \gamma T + \frac{1}{2}\lambda |\omega|^2 \tag{2.18}$$

LightGBM telah menunjukkan peningkatan kinerja yang signifikan dibandingkan implementasi GBDT tradisional, mencapai kecepatan pelatihan hingga 20 kali lebih cepat sambil mempertahankan akurasi yang sebanding [49]. Seperti yang dicatat dalam survei pembelajaran ensemble terbaru [33], efisiensi dan efektivitas LightGBM telah menjadikannya pilihan populer untuk aplikasi pembelajaran mesin skala besar, terutama ketika sumber daya komputasi terbatas atau ketika berhadapan dengan data sparse berdimensi tinggi.

LightGBM merepresentasikan kemajuan signifikan dalam algoritma gradient boosting melalui pendekatan inovatifnya dalam menangani data skala besar. Kombinasi algoritma berbasis histogram, GOSS, dan EFB memberikan solusi efisien untuk tantangan pembelajaran mesin modern sambil mempertahankan akurasi model. Optimasi ini membuat LightGBM sangat berharga untuk aplikasi yang melibatkan dataset besar dengan ruang fitur sparse berdimensi tinggi.