# Penerapan Extreme Gradient Boosting (XGBoost) dengan SMOTE untuk Deteksi Penipuan Kartu Kredit

**TUGAS AKHIR**

**Nicholas Anthony Suhartono**
**1118049**

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
*Veritas vos liberabit*

**PROGRAM STUDI INFORMATIKA**
**INSTITUT TEKNOLOGI HARAPAN BANGSA**
**BANDUNG**
**2021**

# Penerapan Extreme Gradient Boosting (XGBoost) dengan SMOTE untuk Deteksi Penipuan Kartu Kredit

**TUGAS AKHIR**

**Diajukan sebagai salah satu syarat untuk memperoleh**
**gelar sarjana dalam bidang Informatika**

**Nicholas Anthony Suhartono**
**1118049**

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
*Veritas vos liberabit*

**PROGRAM STUDI INFORMATIKA**
**INSTITUT TEKNOLOGI HARAPAN BANGSA**
**BANDUNG**
**2021**

# DAFTAR ISI

# DAFTAR TABEL

# DAFTAR GAMBAR

# BAB 1 PENDAHULUAN

## 1.1 Latar Belakang

Kartu kredit merupakan salah satu metode pembayaran yang umum di dunia finansial maupun bisnis. Penipuan kartu kredit adalah kejadian dimana penipu menggunakan identitas kartu kredit milik orang lain untuk melakukan transaksi. Jumlah penipuan transaksi kartu kredit meningkat setiap tahunnya, khususnya di Amerika Serikat. Pada tahun 2014, terjadi 55.553 penipuan kartu kredit, sedangkan pada tahun 2018, jumlahnya meningkat menjadi 157.688 kasus penipuan [1]. Jumlah kerugian yang diakibatkan oleh transaksi kartu kredit di seluruh dunia pada tahun 2018 sangat besar, yaitu sebesar 24,26 miliar dollar Amerika Serikat [1]. Penipuan kartu kredit dapat terjadi ketika penipu menggunakan informasi palsu untuk melakukan transaksi kartu kredit, dan pihak penerbit kartu kredit menerima transaksi tersebut atau ketika kartu kredit diterbitkan dengan benar namun transaksi melibatkan aktifitas yang bersifat curang atau penipuan. Deteksi penipuan kartu kredit memiliki tantangan tersendiri, karena dari banyaknya transaksi kartu kredit yang terjadi, hanya beberapa transaksi yang dapat dikategorikan kedalam penipuan. Oleh karena itu, diperlukan sebuah sistem yang sensitif untuk mendeteksi transaksi penipuan kartu kredit [2], [3], [4], [5], [6].

Penelitian [2] bertujuan untuk mendeteksi anomali atau *outlier* dengan menggunakan algoritme *Local Outlier Factor* dan *Isolation Forest*. Kedua algoritme ini mampu mengukur nilai anomali untuks setiap sample data. Dataset merupakan data transaksi kartu kredit berjumlah 285 ribu data. Hasil penelitian dengan menggunakan 10% dari dataset, *Isolation Forest* memperoleh *precision*, *recall*, dan *f1-score* tertinggi untuk mendeteksi kelas positif *(fraud)* dengan nilai 28%, 29%, dan 28%. Sedangkan jika keseluruhan dataset digunakan, *Isolation Forest* memperoleh *precision*, *recall*, dan *f1-score* tertinggi dengan nilai 33%, 33%, dan 33%. Perlu diperhatikan bahwa *recall* seharusnya menjadi ukuran dalam deteksi penipuan kartu kredit karena *recall* menekan terjadinya klasifikasi *false negative*, yang berarti bahwa transaksi diklasifikasikan sebagai transaksi bukan penipuan, tetapi sebenarnya merupakan transaksi penipuan.

Penelitian [3] bertujuan untuk mendeteksi penipuan dalam transaksi kartu kredit dengan menggunakan pendekatan *Hybrid Classification* yaitu klasifikasi dengan menggunakan metode *K-nearest neighbors* (KNN) dan *Naive Bayes*. Algoritme KNN akan melakukan klasifikasi transaksi kartu kredit. Hasil klasifikasi

dari KNN akan menjadi masukkan bagi algoritme *Naive Bayes*, sehingga hasil sesungguhnya merupakan keluaran dari algoritme *Naive Bayes*. Dataset pada penelitian ini diambil dari *UCI Machine Learning Repository*. Dataset berisi 284,807 transaksi, dengan 492 (0.172%) diantaranya merupakan transaksi penipuan. Hasil pengujian menunjukkan bahwa *Hybrid Classification* memperoleh nilai *recall* sebesar 36%, akurasi sebesar 100% dan *precision* sebesar 100%.

Penelitian [4] bertujuan untuk mendeteksi penipuan dalam transaksi kartu kredit dengan menggunakan *Light Gradient Boosting Machine* (LightGBM) dan *Bayesian-based hyperparameter optimization* yang digunakan sebagai metode *Hyperparameter tuning* dan menggunakan 2 buah dataset. Dataset pertama 284 ribu data transaksi kartu kredit dengan 492 merupakan transaksi penipuan. Sedangkan dataset kedua diambil dari *University of California Data Mining Contest* yang berisi data transaksi *E-commerce* berjumlah 94.683 data transaksi dengan 2,094 diantaranya merupakan transaksi penipuan. Data tersebut diambil dari 73.729 kartu kredit selama 98 hari. *Cross validation* dengan jumlah *5-fold* diterapkan agar memperoleh hasil pengukuran yang lebih akurat. Pada dataset kedua, LightGBM memperoleh akurasi 98,40%, *recall* 40,59%, *precision* 97,34%, dan *f1-score* 56,95%. Pada dataset pertama, LightGBM memperoleh akurasi 98,35%, *recall* 28,33%, *precision* 91,72%, dan *f1-score* 43,27%.

Penelitian [5] bertujuan untuk melakukan pengujian algoritme klasifikasi *machine learning* dengan metode klasifikasi *imbalance*. Algoritme klasifikasi yang digunakan adalah C5.0, *Support Vector Machine* (SVM), *Artificial Neural Network* (ANN), *Naive Bayes* (NB), *Bayesian Belief Network* (BBN), *Logistic Regression* (LR), *K-Nearest Neighbor* (KNN), dan *Artifiical Immune System* (AIS). Sedangkan untuk menangani dataset yang *imbalance*, digunakan metode *Random Oversampling* (RO), *One-Class Classification* (OCC) yang terdiri dari *One-Class Classification SVM* (OCC SVM) dan *Auto Associative Neural Network* (AANN), dan *Cost-sensitive models* (CS). Dataset penelitian ini berisi 10 juta data transaksi kartu kredit. Evaluasi pengukuran yang digunakan pada penelitian ini adalah akurasi, *precision*, *sensitivity* (*recall)* dan *area under the precision-recall curve* (AUPRC). Hasil terbaik pada pengujian penelitian ini menunjukakn bahwa akurasi tertinggi diperoleh oleh C5.0, ANN dan SVM sebesar 96% sedangkan *recall* tertinggi diperoleh RO C5.0 dengan nilai 66% dan AUPRC tertinggi diperoleh oleh SVM dengan nilai 63%.

Penelitian [6] menguji *Synthetic Minority Oversampling Technique* (SMOTE) sebagai metode *oversampling* dengan menggunakan algoritme

klasifikasi *K-Nearest Neighbors* (KNN), *Decision Tree*, *Logistic Regression*, Random Forest, dan *Extreme gradient boosting* (XGBoost). Dataset transaksi penipuan kartu kredit berjumlah 285 ribu data. Hasil penelitian menunjukkan bahwa akurasi tertinggi diperoleh oleh semua algoritme dengan nilai 99%. Algoritme *Random Forest* memperoleh *precision* tertinggi dengan nilai 90%. XGBoost memperoleh *recall* dan *f1-score* tertinggi dengan nilai 79% dan 84%. Pada masalah transaksi kartu kredit, evaluasi yang paling tepat digunakan adalah *recall*. Oleh karena itu, XGBoost akan digunakan dalam penelitian ini karena memiliki nilai *recall* tertinggi.

Penelitian ini akan berfokus kepada pengujian *hyperparameter Extreme gradient boosting* (XGBoost). Selain itu, metode *k-fold cross validation* akan diterapkan dalam pengujian dengan tujuan untuk melihat performa model untuk memprediksi data yang tidak ada dalam proses *training*. Metode *Synthetic Minority Oversampling Technique* (SMOTE) untuk menangani dataset yang *imbalance*.

## 1.2 Rumusan Masalah

Berdasarkan latar belakang di atas penulis merumuskan masalah sebagai berikut:

1. Bagaimana konfigurasi *hyperparameter* terbaik pada algoritme klasifikasi XGBoost?
2. Berapa nilai akurasi, *recall* dan *f-measure* terbaik algoritme XGBoost dan SMOTE?

## 1.3 Tujuan Penelitian

Tujuan yang ingin dicapai dalam penelitian ini adalah menguji algoritme klasifikasi XGBoost dengan menggunakan *hyperparmeter tuning*.

1. Mencari nilai *hyperparameter* terbaik XGBoost untuk klasifikasi transaksi penipuan kartu kredit.
2. Mengetahui hasil kinerja model klasifikasi *machine learning* XGBoost dengan *k-fold cross-validation* dan SMOTE sebagai metode *oversampling*.

## 1.4 Batasan Masalah

Dalam penelitian ini, peneliti akan membatasi masalah yang akan diteliti, antara lain:

1. Sistem deteksi transaksi penipuan kartu kredit bekerja secara *offline* sehingga tidak dapat digunakan pada kasus *real-time*.

2. Masukan berupa dataset transaksi kartu kredit yang diambil dari ULB Machine Learning Group yang dapat diakses melalui situs *kaggle* [7].

## 1.5 Kontribusi Penelitian

Dengan dilakukannya penelitian ini, diharapkan dapat mengetahui *hyperparameter* terbaik pada XGBoost dengan menggunakan SMOTE sebagai teknik *oversampling*.

## 1.6 Metodologi Penelitian

Metode penelitian yang dilakukan dalam penelitian ini adalah sebagai berikut:

1. Studi Literatur
   Penulisan ini dimulai dengan studi kepustakaan yaitu mengumpulkan bahan-bahan referensi baik dari jurnal, paper, dan buku mengenai deteksi transaksi penipuan kartu kredit.
2. Data *Sampling*
   Data *sampling* yang akan digunakan berupa data transaksi pembayaran menggunakan kartu kredit dan akan diambil dari beberapa penyedia data terbuka di internet.
3. Analisis Masalah
   Pada tahap ini dilakukan analisis permasalahan yang ada, batasan yang dimiliki dan kebutuhan yang diperlukan.
4. Perancangan dan Implementasi *Algoritme*
   Pada tahap ini dilakukan pendefinisian beberapa aturan dalam teknik klasifikasi data transaksi, serta perancangan pada algoritma yang akan dipakai untuk menyelesaikan masalah berdasarkan metode yang telah dipilih.
5. Pengujian
   Pada tahap ini dilakukan pengujian terhadap aplikasi yang telah dibangun.
6. Dokumentasi
   Pada tahap ini dilakukan pendokumentasian hasil analisis dan implementasi secara tertulis dalam bentuk laporan skripsi.

## 1.7 Sistematika Pembahasan

Pada penelitian ini peneliti menyusun berdasarkan sistematika penulisan sebagai berikut:

**Bab I Pendahuluan**

Pendahuluan yang berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, kontribusi penelitian, serta metode penelitian.

**Bab II Landasan Teori**

Landasan teori yang berisi penjelasan dasar teori yang mendukung penelitian ini.

**Bab III Analisis dan Perancangan**

Analisis dan perancangan yang berisi analisis berupa algoritma yang digunakan.

**Bab IV Implementasi dan Pengujian**

Implementasi dan pengujian yang berisi implementasi pengujian dengan berbagai data *testing* beserta hasilnya.

**Bab V Kesimpulan dan Saran**

Penutup yang berisi kesimpulan dari penelitian dan saran untuk penelitian lebih lanjut di masa mendatang.

# BAB 2 LANDASAN TEORI

## 2.1 Tinjauan Pustaka

Penelitian ini menggunakan beberapa teori terkait yang diperlukan dalam pengerjaan yang dilakukan. Penjelasan mengenai teori-teori tersebut akan dijelaskan sebagai berikut.

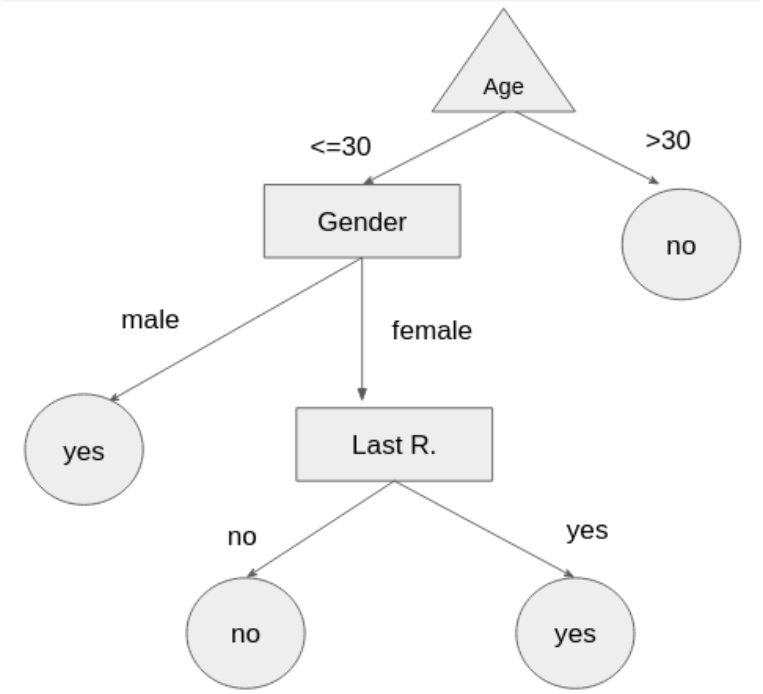
### 2.1.1 *Decision Tree*

*Decision Tree* (DT) merupakan algoritme *machine learning* yang mampu melakukan klasifikasi dan regresi. Algoritme ini merupakan komponen fundamental untuk algoritme *Random Forest* [8], *Gradient Boost*, dan *XGBoost*. DT terdiri dari simpul (*node*) yang membentuk sebuah pohon (*tree*). Simpul akar (*root node*) adalah *node* yang tidak memiliki cabang (*edge*) yang masuk ke dalam. *Node* yang memiliki cabang keluar disebut sebagai simpul dalam *internal node*. *Node* yang tidak memiliki cabang keluar disebut sebagai simpul daun *leaf node* [9]. Setiap *internal node* merepresentasikan atribut atau fitur tertentu. Sedangkan setiap *leaf node* berisi nilai dari atribute atau fitur *internal node* [9].

Untuk menentukan fitur yang akan dijadikan *root node* ataupun *internal node* dapat ditentukan dengan menghitung rumus *gini impurity* untuk mengukur ketidakmurnian sebuah *node impurity*. Sebuah *node* disebut murni (*impure*) apabila nilai gini sama dengan 0. Persamaan 2.1 merupakan rumus untuk menghitung nilai *gini impurity* .

$$G_i = 1 - \sum_{k=1}^{n} {P_{i,k}}^2 \tag{2.1}$$

| | |
|---|---|
| Keterangan | : |
| $P_{i,k}$ | : merupakan rasio dari kelas $k$ pada data latih pada *node* ke $i$ |



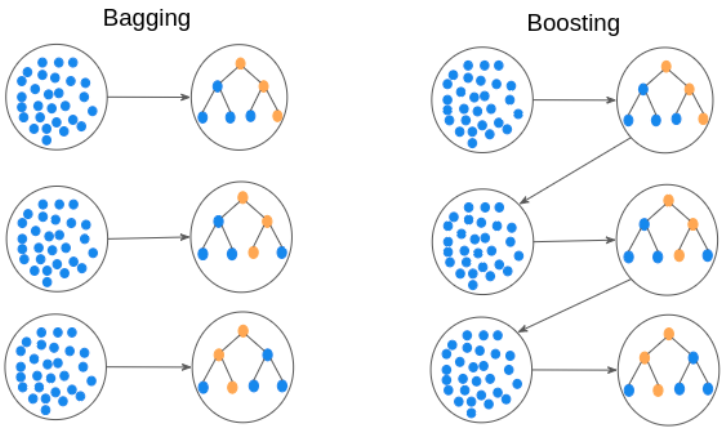

**Gambar 2.1** Ilustrasi *Decision Tree*

Gambar 2.1 merupakan ilustarasi *Decision Tree* untuk menentukan apakah kustomer akan merespons terhadap *email marketing*. *Root node* digambarkan dengan bentuk segitiga. *Internal node* digambarkan dengan bentuk persegi panjang. *Leaf node* digambarkan dengan bentuk lingkaran [9].

### 2.1.2 *Ensemble Learning*

*Ensemble learning* merupakan salah satu paradigma dalam pembelajaran *machine learning* yang menggabungkan prediksi dari berbagai model yang menghasilkan akurasi yang rendah *weak learner* untuk mendapatkan akurasi yang tinggi. *Weak learner* tidak mampu mempelajari model yang kompleks, tetapi cepat dalam proses *training* dan membuat prediksi. Salah satu contoh model *weak learner* yang biasanya digunakan adalah *Decision Tree* yang hanya melakukan beberapa *split* saja. Kedalaman dari *tree* tersebut tidak sangat dalam, sehingga tidak sangat akurat dalam mebuat prediksi. *Ensemble learning* akan memanfaatkan *tree* dengan jumlah yang banyak dan yang tidak identikal serta memiliki akurasi setidaknya lebih baik daripada menebak *random guessing* untuk memperoleh akurasi yang lebih baik. Terdapat 2 metode dasar dalam *ensemble learning* yaitu *boosting* dan *bagging*. *Boosting* merupakan proses dimana secara iteratif membuat beberapa model menggunakan *weak learner* dengan memanfaatkan data latih. Setiap model akan berbeda dengan model lainnya karena setiap model baru akan berusaha untuk memperbaiki *error* dari model sebelumnya [10]. Model *ensemble*

terakhir merupakan kombinasi dari berbagai model yang telah dibuat secara iteratif sebelumnya. Proses pelatihan dalam *boosting* berjalan secara sekuensial karena setiap *learner* dibangun berdasarkan *learner* sebelumnya. Dalam *boosting* hasil prediksi diperoleh dari *weak learner* yang memiliki nilai bobot masing-masing. *Weak learner* yang memiliki hasil prediksi yang baik memiliki bobot yang lebih besar. Contoh algoritme yang menggunakan *boosting* adalah XGBoost dan Gradient Boosting. *Bagging* merupakan proses untuk membuat data baru menggunakan data latih yang memiliki sedikit perbedaan, kemudian akan diterapkan *weak learner* untuk setiap data baru untuk mendapatkan *weak model* [10]. Proses pelatihan dalam *bagging* dapat berjalan secara bersamaan karena setiap *learner* yang dibangun secara independen. Dalam *bagging* hasil prediksi diperoleh dari dari hasil *voting* dari semua *weak learner* yang dibentuk. Salah satu contoh algoritme *machine learning* yang menggunakan *bagging* adalah *Random Forest*.




**Gambar 2.2** Ilustrasi alur proses pelatihan pada *bagging* dan *boosting*.

Gambar 2.2 merupakan ilustrasi alur proses pelatihan pada *bagging* dan *boosting*. *Bagging* membangun *weak learner* secara independen dan prediksi didapat dari hasil *voting* setiap *learner*. *Boosting* membangun *learner* berdasarkan *learner* sebelumnya. Prediksi didapat dari nilai bobot rata-rata setiap *learner*.

### 2.1.3 *Random Forest*

Salah satu kekurangan dari *Decision Tree* adalah cenderung untuk *overfit* , yang berarti *Decision Tree* memiliki performa yang baik pada data latih namun kurang baik dengan data uji [11]. *Random forest* mampu mengatasi masalah *overfit* pada *Decision Tree*. *Random Forest* memanfaatkan algoritme *Decision Tree* untuk membangun banyak *tree* untuk mengurangi *overfit* dan membuat prediksi dengan mengambil aggregasi dari *tree* yang telah dibuat. *Random Forest* memanfaatkan

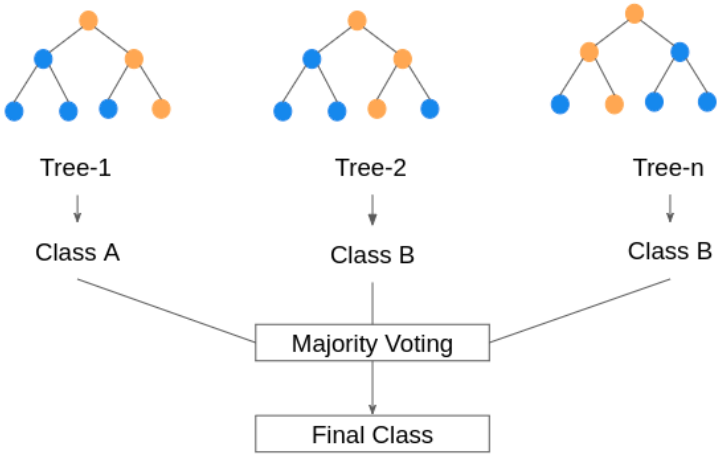
prinsip *bagging* atau *bootstrap aggregating* dalam *ensemble learning*. *Random Forest* melakukan proses pelatihan dengan mengambil sebagian data dari dataset secara acak sebagai data latih dan proses ini dapat dilakukan berulang kali (*bootstrap dataset*). Proses prediksi dilakukan dengan melakukan *voting* terdapat seluruh *tree* yang telah dibuat. Hasil prediksi merupakan jumlah terbanyak dari *voting* yang telah dilakukan. Proses ini dikenal dengan nama *majority voting*, yaitu mengambil hasil prediksi dari *voting* terbanyak.

Algoritme *Random Forest* [12]:

1. Untuk sejumlah *tree* yang ingin dibangun dalam *Random Forest* lakukan iterasi sebagai berikut:
   (a) Membuat dataset *bootstrap* dengan jumlah data N dari dataset yang digunakan.
   (b) Gunakan dataset boostrap sebelumnya untuk dijadikan sebagai data latih dan mulailah proses pelatihan.
   (c) Memilih *root node* dengan menghitung nilai *gini* untuk setiap fitur yang terpilih lalu lakukan langkah berikut untuk secara rekursif.
   (d) Memilih beberapa fitur secara acak.
   (e) Temukan *binary split* terbaik dengan menghitung nilai *gini*.
   (f) Fitur dengan nilai *gini* terendah pada iterasi pertama akan menjadi *root node*. Lalu lakukan *split* kembali. Proses *splitting* akan berhenti jika nilai *gini* sebesar 0 atau sudah mencapai jumlah maksimum kedalaman *tree* yang diinginkan.
2. Lakukan prediksi menggunakan persamaan 2.2.

$$y = \frac{1}{B}\sum_{b=1}^{B} f_b(x) \tag{2.2}$$

| | |
|---|---|
| Keterangan | : |
| $y$ | : hasil prediksi *Random Forest* |
| $B$ | : jumlah tree |
| $x$ | : sampel yang akan diprediksi |
| $f_b(x)$ | : hasil prediksi *tree* pada sampel $x$ |




**Gambar 2.3** Ilustrasi *Random Forest* ketika melakukan prediksi.

Pada gambar 2.3 terlihat bahwa terdapat *n tree* yang dibangun oleh algoritme *Random Forest*. Setiap *tree* akan membuat prediksi. Terlihat bahwa *tree* pertama mengeluarkan prediksi kelas A, sedangkan *tree* kedua mengeluarkan prediksi kelas B. Proses prediksi ini dilakukan oleh setiap *tree* yang dibangun. Setelah melakukan prediksi, *Random Forest* akan melakukan *voting* terhadap setiap hasil prediksi yang dihasilkan oleh setiap *tree*. Hasil prediksi dengan jumlah *voting* terbanyak merupakan hasil akhir dari prediksi yang dihasilkan oleh algoritme *Random Forest*.

*Random Forest* memiliki beberapa *hyperparameter*. Berikut adalah *hyperparameter* dari model *Random Forest* [13]:

1. *n_estimators*: Jumlah *tree* yang akan dibentuk oleh *Random Forest*.
2. *criterion*: Fungsi untuk menentukan kualitas dari sebuah split.
3. *max_depth*: Nilai kedalaman maximum *tree*.
4. *min_sample_split*: Jumlah sampel minimum untuk melakukan *split*.
5. *min_sample_leaf*: Jumlah sampel minimum yang diperlukan pada *leaf node*.

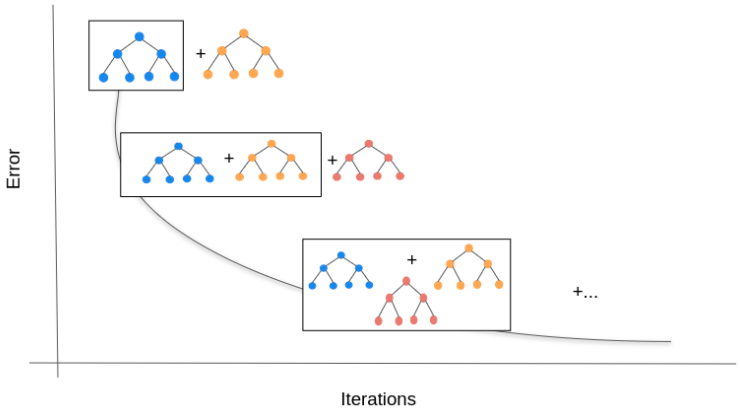
6. *max_features*: Jumlah fitur yang akan digunakan ketika mencari *split* terbaik.

### 2.1.4 *Gradient Boost*

*Gradient Boost* merupakan salah satu metode *ensemble learning* yang mengkombinasikan banyak *Decision Tree* untuk membuat model baru yang lebih baik performanya. *Gradient Boost* bekerja dengan cara membangun *tree* secara serial, dimana setiap tree yang dibangun memperbaiki kesalahan dari *tree* sebelumnya [11]. *Gradient Boost* biasanya menggunakan *weak learner* dengan ketinggian 1 - 5 *node* yang membuat model ini efisien dalam penggunaan memori dan cepat dalam melakukan prediksi namun dalam jumlah yang banyak untuk meningkatkan performa. Kelemahan dari *Gradient Boost* adalah waktu pelatihan yang lama.

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2. For $m = 1$ to $M$ do:
3. $\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4. $\mathbf{a}_m = \arg\min_{\mathbf{a}, \beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5. $\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6. $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$

**Gambar 2.4** Algoritme *Gradient Boost* [14]




**Gambar 2.5** Ilustrasi Algoritme Gradient Boosting.

Algoritme Gradient Boost [14]:

1. Melakukan inisialisasi prediksi yang meminimalkan *loss function* menggunakan persamaan 2.3
2. Untuk setiap *m tree* dalam sejumlah *M tree*, lakukan:
   (a) Menghitung residual untuk setiap sampel data $x_i$ menggunakan persamaan 2.4

(b) Membuat *decision tree* dengan *terminal node* atau *leaf node* sebanyak L sesuai persamaan 2.5.

(c) Menghitung nilai output atau *gamma* pada setiap leaf node *l* tree ke *m* sesuai persamaan 2.6.

(d) Memperbaharui nilai prediksi $f_m(x)$ sesuai persamaan 2.7.

$$f_0(x) = argmin_\gamma \sum_{i=1}^{N} \psi(y_i, \gamma) \tag{2.3}$$

| | |
|---|---|
| Keterangan | : |
| $f_0(x)$ | : Nilai prediksi awal. |
| $\psi(y_i, \gamma)$ | : *Loss function*. |

$$\hat{y}_{im} = -[\frac{\partial \psi(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}, i = 1, N \tag{2.4}$$

| | |
|---|---|
| Keterangan | : |
| $\hat{y}_{im}$ | : Nilai residual. |
| $N$ | : Jumlah sampel data. |
| $i$ | : residual ke $i$. |
| $m$ | : *tree* ke $m$. |

$$\{R_{lm}\}_1^L = L - \text{terminal node } tree(\{\hat{y}_{im}, x_i\}_1^N) \tag{2.5}$$

| | |
|---|---|
| Keterangan | : |
| $R_{lm}$ | : Nilai residual *terminal* node. |

$$\gamma_{lm} = argmin_{\gamma} \sum_{x_i \varepsilon R_{lm}} \psi(y_i, F_{m-1}(x_i) + \gamma) \tag{2.6}$$

| Keterangan | : |
|---|---|
| $\gamma_{lm}$ | : Nilai gamma pada *leaf node l tree m*. |

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_{lm} 1(x \varepsilon R_{lm}) \tag{2.7}$$

| Keterangan | : |
|---|---|
| $F_m(x)$ | : Nilai prediksi baru untuk sampel data x. |
| $\nu$ *: *Learning rate*. | |

Berdasarkan gambar 2.5, terlihat bahwa pada iterasi pertama, terdapat *base learner tree* dan sebuah *tree* baru yang terbentuk. Kemudian model *gradient boost* diperbaharui menggunakan kedua tree yang terbentuk pada iterasi pertama, dan membangun tree berikutnya pada iterasi selanjutnya. Proses ini berlanjut hingga jumlah tree telah memenuhi yang diinginkan. Setiap *tree* yang dibangun akan berkontribusi untuk memperbaiki model untuk meminimalkan *error* yang terjadi.

*Gradient boost* memiliki beberapa *hyperparameter* yang dapat diatur. Berikut merupakan beberapa *hyperparameter* dari *Gradient boost* [13]:

1. *n_estimators*: Jumlah *tree* yang akan di bangun oleh *gradient boost*.
2. *criterion*: Fungsi yang digunakan untuk mengukur *split* terbaik.
3. *learning_rate*: Nilai yang digunakan untuk menyusutkan kontribusi dari setiap *tree*.
4. *loss*: Fungsi *loss* yang digunakan oleh algoritme *gradient boost*.
5. *subsample*: Jumlah sampel yang digunakan untuk proses pelatihan.
6. *min_samples_split*: Jumlah minimum sampel yang diperlukan untuk melakukan *split*.

7. *min_samples_leaf*: Jumlah sampel minimum yang diperlukan pada *leaf node*.
8. *max_depth*: Nilai maksimum kedalaman untuk setiap tree yang akan dibuat.
9. *max_features*: Jumlah fitur yang akan dipertimbangkan ketika melakukan proses *split*.

#### 2.1.4.1 *XGBoost*

*Decision tree* memiliki kelemahan dimana model terlalu akurat sehingga gagal untuk memprediksi data baru, sedangkan *ensemble learning* yang menggabungkan *decision tree* menggunakan *bagging* dan *boosting* terbukti lebih efektif [15]. Konsistensi dan hasil yang bagus dari *gradient boost*, membuat Tianqi Chen dari Universitas Washington untuk menciptakan algoritme dan system yang dinamakan *XGBoost* [15]. XGBoost tersedia dalam bentuk pustaka *python* yang dirilis pada tanggal 27 Maret 2014. Semula, XGBoost berawal dari proyek penelitian. Namun pada tahun 2015 XGBoost menjadi solusi paling dominan untuk menyelesaikan masalah klasifikasi dan regresi. Pada kompetisi yang diselenggarakan oleh *Kaggle* pada tahun 2015 yang terdapat 29 tantangan, 17 solusi diantaranya menggunakan XGBoost. Begitu juga dengan kompetisi yang diselenggarakan oleh KDDCup pada tahun 2015, XGBoost digunakan oleh setiap tim dalam peringkat 10 besar [16].

XGBoost diciptakan menggunakan prinsip *gradient boost* yang menggabungkan *weak learner* dengan *strong learner*. *Gradient-boosted tree* pada umumnya dibangun secara sekuensial, sedikit demi sedikit memperbaiki hasil prediksi di iterasi selanjutnya, tetapi XGBoost mampu membangun *tree* secara paralel. XGBoost memiliki performa prediksi lebih tinggi dengan mengendalikan kompleksitas model dan mengurangi *overfitting* melalui regularisasi (*regularization*).

**Data:** Dataset and hyperparameters
Initialize $f_0(x)$;
**for** $k = 1, 2, \ldots, M$ **do**
  Calculate $g_k = \frac{\partial L(y,f)}{\partial f}$;
  Calculate $h_k = \frac{\partial^2 L(y,f)}{\partial f^2}$;
  Determine the structure by choosing splits with maximized gain
  $\mathbf{A} = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G^2}{H}\right]$;
  Determine the leaf weights $w^* = -\frac{G}{H}$;
  Determine the base learner $\hat{b}(x) = \sum_{j=1}^{T} wI$;
  Add trees $f_k(x) = f_{k-1}(x) + \hat{b}(x)$;
**end**
**Result:** $f(x) = \sum_{k=0}^{M} f_k(x)$

**Gambar 2.6** Algoritme XGBoost [17]

| | |
|---|---|
| Keterangan | : |
| $x$ | : Fitur pada dataset. |
| $y$ | : Label yang akan diprediksi pada dataset. |
| $f_0(x)$ | : Prediksi awal model yang menerima fitur $x$. |
| $M$ | : Banyaknya *tree* yang akan dibentuk. |
| $L(y,f)$ | : *Loss function* yang akan digunakan pada model. |
| $g_k$ | : Nilai *gradient* pada *tree k* sesuai persamaan *gradient* 2.8. |
| $h_k$ | : Nilai *hessian* pada *tree k* sesuai persamaan *hessian* 2.9. |
| $A$ | : Nilai *gain* untuk menentukan *split*. |
| $G_L$ | : Nilai *gradient* pada *node* kiri. |
| $H_L$ | : Nilai *hessian* pada *node* kiri. |
| $G_R$ | : Nilai *gradient* pada *node* kanan. |
| $H_R$ | : Nilai *hessian* pada *node* kanan. |
| $T$ | : Jumlah *leaf node*. |
| $I$ | : Kumpulan index dari input $x$. |
| $w^*$ | : Nilai *output* pada *leaf node*. |
| $\hat{b}$ | : Hasil prediksi tree*tree*. |

$$Gradient = p - y_i \tag{2.8}$$

| | |
|---|---|
| Keterangan | : |
| $y_i$ | : Nilai label pada ke $i$. |
| $p$ | : Nilai *probability*. |

$$Hessian = p(1-p) \tag{2.9}$$

Keterangan :
$p$ : Nilai *probability*.

Berdasarkan gambar 2.6, algoritme XGBoost dapat dijelaskan sebagai berikut.

1. Algoritme XGBoost dimulai dengan membuat inisialisasi prediksi dengan nilai 0.5 sesuai persamaan 2.10.
2. Untuk setiap *M tree* yang akan dibentuk
   (a) Pada masing-masing fitur yang telah diurutkan, tentukan *binary split* terbaik dengan menghitung nilai *gradient* menggunakan persamaan 2.8 dan *hessian* menggunakan persamaan 2.9.
   (b) Tentukan nilai *gain* untuk masing-masing *leaf node* yang akan di *split*. Nilai *gain* dapat dihitung menggunakan persamaan 2.11. Apabila nilai *gain* lebih kecil 0, maka berhenti membangun *branch* tersebut. Apabila nilai penjumlahan *hessian* lebih kecil daripada nilai *min_child_weight*, maka berhenti membangun *branch* tersebut (*prunning*).
   (c) Menghitung nilai bobot pada setiap *leaf node* pada *tree k* menggunakan persamaan 2.12.
   (d) Menghitung nilai base learner sesuai persamaan 2.13.
   (e) Memperbaharui nilai prediksi sebelumnya yaitu $f_{k-1}(x)$ menggunakan persamaan 2.14
3. Model XGBoost didapat sesuai persamaan 2.15
4. Untuk melakukan prediksi, menggunakan persamaan 2.16.

$$f_0(x) = 0.5 \tag{2.10}$$

Keterangan :
$f_0(x)$ : Nilai prediksi awal model yang menerima fitur $x$.

$$Gain = \frac{1}{2}[\frac{G_L{}^2}{H_L+\lambda} + \frac{G_R{}^2}{H_R+\lambda} - \frac{(G_L+G_R{}^2)}{H_L+H_R+\lambda}] - \gamma \tag{2.11}$$

Keterangan :
$G_L$ : Nilai *gradient leaf node* kiri.
$G_R$ : Nilai *gradient leaf node* kanan.
$H_L$ : Nilai *hessian leaf node* kiri.
$H_R$ : Nilai *hessian leaf node* kanan.
$\lambda$ : Konstanta regularisasi.
$\gamma$ : Konstanta pengurang nilai *gain*.

$$w_j = -\frac{G_j}{H_j+\lambda} \tag{2.12}$$

Keterangan :
$w_j$ : Nilai bobot pada *leaf node j*.
$G_j$ : Nilai *gradient* pada *leaf node j*.
$H_j$ : Nilai *hessian* pada *leaf node j*.
$\lambda$ : konstanta regularisasi.

$$\hat{b}(x) = \sum_{j=1}^{T} wI \tag{2.13}$$

Keterangan :
$T$ : Jumlah *leaf node*.
$w$ : Nilai *weight*.
$I$ : Set input pada masukkan data x

$$f_k(x) = f_k - 1(x) + \eta\hat{b}(x) \tag{2.14}$$

| Keterangan | : |
|---|---|
| $f_k(x)$ | : Nilai prediksi *tree k*. |
| $\hat{b}(x)$ | : Nilai weight yang didapat dengan menelusuri tree. |
| $\eta$ | : Learning rate. |

$$f(x) = \sum_{k=1}^{M} f_k(x) \tag{2.15}$$

| Keterangan | : |
|---|---|
| $f(x)$ | : Nilai *output* pada algoritme XGBoost. |
| $M$ | : Jumlah *tree* yang telah ditentukan. |
| $f_k(x)$ | : Nilai *output* setiap *tree k*. |

$$f(x) = f_0(x) + \sum_{k=1}^{M} \eta f_k(x) \tag{2.16}$$

| Keterangan | : |
|---|---|
| $f(x)$ | : Nilai prediksi akhir. |
| $f_0(x)$ | : Nilai prediksi awal. |
| $M$ | : Jumlah *tree*. |
| $\eta$ | : *Learning rate*. |
| $f_k(x)$ | : Nilai *output* pada *tree* ke *k* |

#### 2.1.4.2 *Loss Function*

*Loss function* merupakan fungsi yang mengukur seberapa jauh *output* atau nilai yang dihasilkan dari nilai yang ekspektasi. Persamaan 2.17 merupakan *loss function* yang digunakan XGBoost untuk melakukan klasifikasi. Fungsi *sigmoid* akan digunakan untuk konversi nilai prediksi XGBoost kedalam rentang 0 hingga

1 karena masalah deteksi transaksi penipuan kartu kredit tersebut merupakan klasifikasi biner yang hanya memprediksi 2 kelas *output*, yaitu apakah transaksi merupakan penipuan (*fraud*) atau bukan penipuan (*non-fraud*).

$$L(y_i, p_i) = -[y_i log(p_i) + (1 - y_i) log(1 - p_i)] \tag{2.17}$$

| Keterangan | : |
|---|---|
| $y_i$ | : Nilai sesungguhnya. |
| $p_i$ | : Nilai prediksi. |

$$Sigmoid = \frac{1}{1 + e^{-t}} \tag{2.18}$$

| Keterangan | : |
|---|---|
| $t$ | : Nilai prediksi. |

#### 2.1.4.3 *Hyperparameter* dalam XGBoost

XGBoost memiliki *hyperparameter* yang dapat diatur (*tuning*). Berikut adalah beberapa *hyperparameter* dalam XGBoost.

1. $\eta$: *Learning rate* merupakan besaran nilai yang digunakan untuk menyusutkan nilai bobot untuk mencegah *overfitting*. Dalam XGBoost, *learning rate* dilambangkan sebagai $\eta$ sesuai persamaan 2.13.

2. $\lambda$: Lambda merupakan konstanta yang berfungsi sebagai penalti untuk mengurangi kompleksitas model sehingga model tidak mengalami *overfitting*. Terdapat 2 macam regularisasi, yang pertama adalah regularisasi L1 yang biasa disebut *L1 norm* atau Lasso, yang menangani *overfitting* dengan mengeleminiasi fitur yang tidak penting dengan mengatur nilai bobot pada fitur menjadi 0 [18] [11]. Regularisasi L2 atau yang biasa disebut *L2 norm* atau Ridge menangani *overfitting* dengan membuat nilai bobot menjadi lebih kecil. XGBoost menggunakan regularisasi L2 lambda ($\lambda$) sesuai persamaan 2.11 dan 2.12.

3. $\gamma$: Gamma merupakan nilai minimum pengurangan *loss* yang digunakan untuk membuat partisi *node* pada *tree*. Nilai $\gamma$ dapat ditemukan pada persamaan 2.11.
4. *n_estimators*: Jumlah *tree* yang akan dibuat dalam *XGBoost*.
5. *max_depth*: Tinggi maximum dari *tree* yang akan dibuat dalam XGBoost.
6. *min_child_weight*: Jumlah minimum nilai *hessian* yang dibutuhkan oleh *leaf node*. Apabila nilai *hessian* lebih kecil daripada nilai *min_child_weight*, maka algoritme akan berhenti melakukan partisi pada *leaf node*.
7. *subsample*: Nilai yang menentukan bagaimana data observasi akan dipilih sebagai data latih.
8. *colsample*: Nilai yang menentukan bagaimana kolom akan dipilih sebagai data latih.
9. *base_score*: Nilai prediksi awal XGBoost. Nilai *default base_score* XGBoost adalah 0.5 [19].

### 2.1.5 Optimisasi pada *Library* XGBoost

Sistem XGBoost dirancang untuk efisiensi dan skalabilitas yang melakukan *boosting* secara paralel sehingga memiliki waktu eksekusi yang lebih cepat dibandingkan algoritma *ensemble learning* lainnya. XGBoost memiliki akurasi yang tinggi, sehingga XGBoost memiliki reputasi yang cukup baik dengan memenangkan berbagai macam kompetisi *machine learning* [16]. *Design feature* dibawah ini menunjukkan bagaimana XGBoost memiliki waktu yang lebih cepat dibandingkan algoritme *ensemble learning* lainnya.

#### 2.1.5.1 Algoritme *Basic Exact Greedy*

Salah satu masalah dalam pembelajaran berbasis *tree* adalah menemukan *split* terbaik. Algoritme pencarian *split* mencari berbagai macam kombinasi *split* yang memungkinkan untuk semua fitur. Algoritme ini disebut *exact greedy*. Algoritme ini memiliki *cost computation* yang tinggi untuk mencari berbagai macam split yang memungkinkan dengan melakukan iterasi setiap sampel, khususnya apabila fitur memiliki nilai kontinu. Untuk melakukan proses ini secara efisien, algoritme harus mengurutkan nilai pada fitur terlebih dahulu dan mengunjungi setiap data yang telah diurutkan untuk mengakumulasi nilai gradient. Algoritme ini membutuhkan waktu pemrosesan yang lama dan memori yang besar apabila data yang diproses jumlahnya sangat banyak.

**Input**: $I$, instance set of current node
**Input**: $d$, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
**for** $k = 1$ ***to*** $m$ **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  **for** $j$ *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  **end**
**end**
**Output**: Split with max score

**Gambar 2.7** Algoritme *exact greedy* [16]

### 2.1.5.2 Algoritme *Approximate split-finding*

Berbeda dengan algoritme *exact* yang melakukan iterasi untuk setiap sampel data, algoritme *approximate split-finding* ini menggunakan persentil, yaitu persentase data yang digunakan untuk mendapatkan kandidat yang akan dilakukan *split*. Secara *default*, XGBoost menggunakan *exact greedy algorithm* untuk dataset yang kecil, dan menggunakan algoritme *approximate split-finding* untuk dataset yang besar [19].

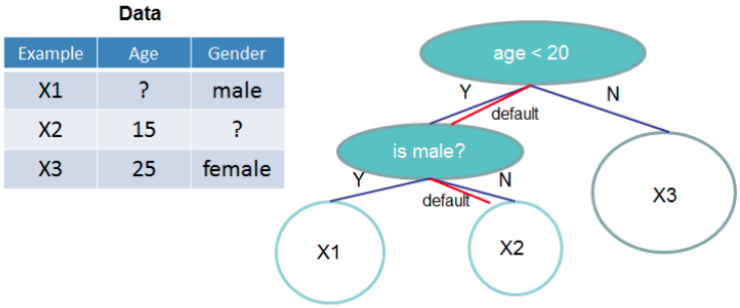
**for** $k = 1$ ***to*** $m$ **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature $k$.
  Proposal can be done per tree (global), or per split(local).
**end**
**for** $k = 1$ ***to*** $m$ **do**
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
**end**
Follow same step as in previous section to find max score only among proposed splits.

**Gambar 2.8** Algoritme *approximate split-finding* [16]

### 2.1.5.3 Sparsity-aware Split Finding

*Sparsity-aware split finding* merupakan salah satu optimisasi dari XGBoost untuk menangani data yang hilang (*missing*). Dataset yang memiliki banyak data yang hilang disebut *sparse*. Ada beberapa hal yang menyebabkan dataset *sparse*, diantaranya karena banyaknya data yang hilang, atau karena hasil *feature engineering* seperti *one-hot encoding*. XGBoost menyediakan arah bawaan *default direction* ketika menelusuri *tree* untuk melakukan prediksi. Gambar 2.9 merupakan ilustrasi dari struktur *tree* dengan *default direction*. Sampel akan

diklasifikasikan ke dalam *default direction* ketika fitur yang dibutuhkan untuk *split* memiliki nilai yang *missing*. [16]




**Gambar 2.9** Struktur *tree* dengan *default direction* [16].

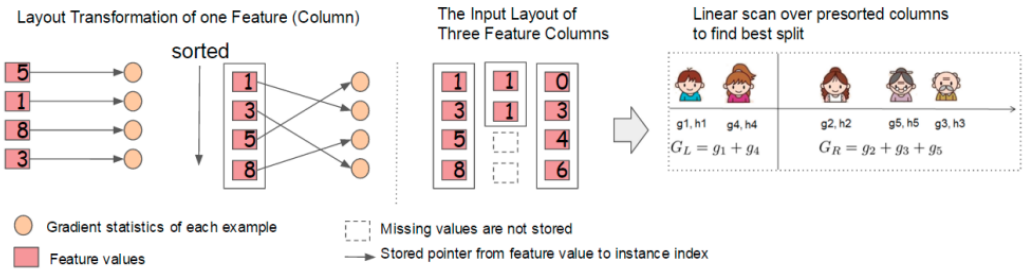
**Input**: $I$, instance set of current node
**Input**: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
**Input**: $d$, feature dimension
*Also applies to the approximate setting, only collect statistics of non-missing entries into buckets*
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i, H \leftarrow \sum_{i \in I} h_i$
**for** $k = 1$ ***to*** $m$ **do**
  *// enumerate missing value goto right*
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  **for** $j$ *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  **end**
  *// enumerate missing value goto left*
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  **for** $j$ *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  **end**
**end**
**Output**: Split and default directions with max gain

**Gambar 2.10** Algoritme *approximate split-finding* [16]

#### 2.1.5.4 *Parallel Learning*

Salah satu hal yang paling memakan waktu dalam proses pelatihan adalah mengurutkan Data. Untuk mengurangi waktu pemrosesan ini, XGBoost menyimpan data di dalam *memory unit* yang disebut sebagai blok (*block*). Data dalam setiap blok disimpan dalam kolom yang sudah di kompresi dengan format *compressed sparse column* (CSC). Pada algoritme *exact greedy*, seluruh dataset

disimpan dalam 1 blok, lalu melakukan *split* dengan melakukan iterasi (*linear scan*) data yang sudah di sort untuk menemukan *split* terbaik. Sedangkan pada algoritme *approximate split-finding*, setiap blok yang berbeda bisa didistribusikan pada mesin komputer yang berbeda, dimana setiap blok merupakan bagian(*subset*) dari dataset. Menggunakan data yang telah diurutkan, pencarian kuantil hanya memerlukan iterasi pada kolom yang telah terurut. Untuk mendapatkan hasil statistik pada setiap kolom, bisa dilakukan secara parallel, sehingga algoritme *split-finding* secara parallel diterapkan dalam XGBoost [16]. Gambar 2.11 merupakan Ilustrasi struktur blok untuk *parallel learning*. Setiap kolom dalam satu blok di urutkan berdasarkan nilainya. *Linear scan* dilakukan untuk mendapatkan *split*.




**Gambar 2.11** Ilustrasi struktur blok untuk *parallel learning* [16].

### 2.1.5.5 Cache-aware Access

XGboost mengalokasikan *internal buffer* pada setiap *thread* dalam komputer, lalu menyimpan statistik hasil perhitungan gradient termasuk nilai dan arahnya untuk setiap *split node*, dan melakukan akumulasi secara *mini-batch* [15]. Proses ini mengurangi waktu untuk melakukan operasi *read/write* [16] khususnya ketika dataset yang digunakan sangat besar dan menghindari *cache miss*, yaitu suatu kondisi dimana data tidak ada dalam *cache*.

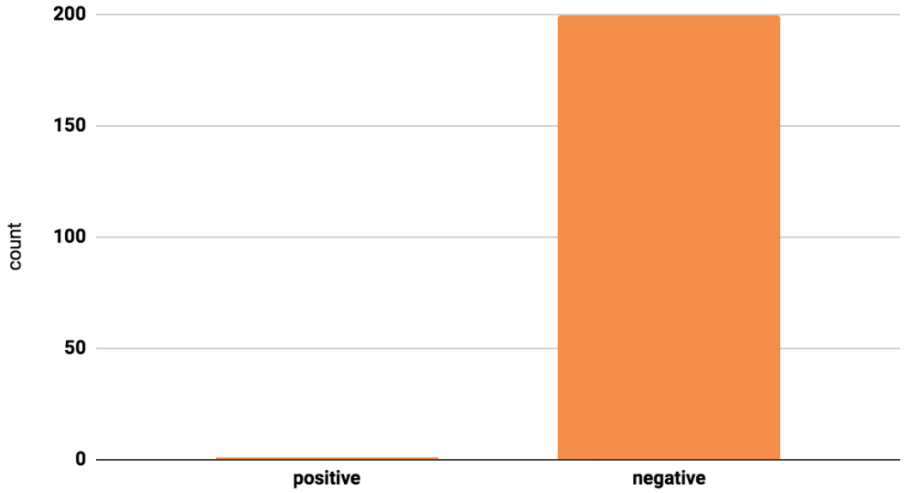
### 2.1.5.6 *Blocks for Out-of-core Computation*

XGBoost membagi data menjadi beberapa blok dan menyimpan setiap blok dalam *disk*. Namun hal ini tidaklah cukup, karena membaca *disk* meningkatkan waktu komputasi. XGBoost menggunakan kompresi blok (*Block Compression*) untuk melakukan kompresi berdasarkan kolom. *Thread* dalam komputer akan melakukan proses dekompresi untuk dimasukkan ke dalam *memory*. XGBoost juga menggunakan teknik yang dinamakan *block sharding*, dimana data akan dipecah ke dalam berbagai *disk* yang berbeda. Suatu *thread* pada komputer akan dialokasikan untuk mengambil data tersebut dan akan dimasukkan ke dalam *buffer* dalam *memory*. *Thread* yang bertanggung jawab untuk proses pelatihan akan mengambil data dari setiap *buffer*. Proses ini meningkatkan

banyaknya data yang bisa dibaca ketika beberapa *disk* tersedia [16], sehingga mengurangi waktu yang digunakan untuk membaca data dari *disk* [15].

### 2.1.6 *Resampling*

Data *Imbalance* merupakan istilah bagi dataset yang memiliki distribusi kelas label yang tidak seimbang, dimana salah satu label memiliki jumlah observasi yang sangat sedikit dibandingkan dengan label lainnya yang memilki jumlah yang sangat besar seperti pada gambar 2.12.




**Gambar 2.12** Ilustrasi dataset yang *imbalance*

Berdasarkan gambar 2.12 terlihat bahwa kelas dengan label positif memilki jumlah observasi yang sangat sedikit dibandingkan dengan kelas negatif. Untuk menyeimbangkan masing-masing label dapat menggunakan teknik *resampling*. Resampling merupakan salah satu metode untuk menyelesaikan masalah dataset yang *imbalance* [20]. Terdapat beberapa teknik dalam melakukan *resampling*, yaitu:

- *Undersampling*, yaitu mengurangi data kelas yang paling dominan atau kelas mayoritas.
- *Oversampling* yaitu meningkatkan jumlah data kelas minoritas. Salah satu teknik yang paling populer untuk melakukan oversampling adalah *Synthetic Minority Oversampling Technique* (SMOTE).


**Gambar 2.13** Ilustrasi *resampling*

### 2.1.7 *SMOTE*

*Synthetic Minority Oversampling Technique* (SMOTE), merupakan salah satu teknik *statistik*oversampling untuk meningkatkan jumlah data pada dataset. SMOTE membuat data baru dari data kelas minoritas sebagai masukan sementara data kelas mayoritas jumlahnya tidak berubah. Data baru yang dihasilkan SMOTE bukan hanya merupakan duplikat dari data minoritas, tetapi algoritme SMOTE mengambil sampel dari ruang fitur untuk setiap target kelas terhadap tetangga terdekatnya, dan membuat sampel baru yang menggabungkan fitur dari kelas target dengan fitur tetangganya [21].

SMOTE menerima seluruh dataset sebagai masukkan, tetapi meningkatkan persentase jumlah data dari data kelas minoritas. Sebagai contoh, bila dataset masukkan sangat tidak seimbang *imbalance* dengan presentasi 1% data memiliki kelas dengan nilai target A dan 99% data memiliki kelas dengan nilai target B. Untuk meningkatkan data kelas minoritas dua kali lipat dari presentase jumlah data sebelumnya, SMOTE harus menerima parameter presentase 200 [21].

```
function SMOTE(T, N, k)
Input: T; N; k                    ▷ #minority class examples, Amount of oversampling, #NNs
Output: (N/100) * T synthetic minority class samples
Variables: Sample[][]: array for original minority class samples;
newindex: keeps a count of number of synthetic samples generated, initialized to 0;
Synthetic[][]: array for synthetic samples
    if N < 100 then
        Randomize the T minority class samples
        T = (N/100)*T
        N = 100
    end if
    N = (int)N/100   ▷ The amount of SMOTE is assumed to be in integral multiples of 100.
    for i = 1 to T do
        Compute KNN for i, and save the indices in the nnarray
        POPULATE(N, i, nnarray)
    end for
end function
```

**Gambar 2.14** Algoritme SMOTE

Algoritme SMOTE:

1. Masukkan berupa nilai $T$ yaitu jumlah sampel kelas minoritas, $N$ jumlah *oversampling* dan $k$ yang merupakan jumlah tetangga terdekat, *array Sample* yang berisi data kelas minoritas orisinil. Deklarasikan *array synthetic* untuk menampung data sintetis atau buatan.
2. Jika nilai $N < 100$, maka hitung nilai $T$ sesuai persamaan 2.19, dan inisialisasi nilai $N$ sebesar 100.
3. Konversi nilai $N$ menjadi persentase.
4. Untuk setiap sample minoritas sebanyak $T$, lakukan perulangan berikut:
   (a) Menghitung jarak $k$ tetangga terdekat.
   (b) Memanggil fungsi *populate*.
5. Keluaran dari algoritme SMOTE adalah sample data buatan untuk kelas minoritas sebanyak $(N/100) * T$.

```
function POPULATE(N, i, nnarray)
Input: N; i; nnarray                ▷ #instances to create, original sample index, array of NNs
Output: N new synthetic samples in Synthetic array
    while N ≠ 0 do
        nn = random(1,k)
        for attr = 1 to numattrs do                    ▷ numattrs = Number of attributes
            Compute: dif = Sample[nnarray[nn]][attr] − Sample[i][attr]
            Compute: gap = random(0, 1)
            Synthetic[newindex][attr] = Sample[i][attr] + gap · dif
        end for
        newindex++
        N−−
    end while
end function
```

**Gambar 2.15** Algoritme *populate* pada SMOTE

Algoritme fungsi *populate*:

1. Fungsi *populate* memiliki masukan yaitu nilai $N$ yang merupakan jumlah data sintetis yang akan dibuat, $i$ yang merupakan indeks sampel data orisinil dan *nnarray* yang merupakan *array* dari tetangga terdekat.
2. Selama $N \neq 0$, lakukan:
   (a) Memiliki angka secara acak antara 1 hingga $k$ dan simpan dalam variabel *nn*, untuk memilih salah satu tetangga dari $k$ tetangga terdekat dari sampel data $i$.

(b) Untuk setiap atribut, lakukan:

i. Menghitung nilai variabel *dif* sesuai persamaan 2.21 pada tetangga *nn*.

ii. Menghitung nilai variabel *gap* sesuai persamaan 2.20.

iii. Membuat data sintetis data baru sesuai persamaan 2.22 dan menyimpannya ke dalam *array Syntethic*.

(c) Mengurangi nilai N sejumlah 1.

3. Keluaran dari fungsi ini adalah *N* data sintetis dalam *array Syntethic*.

$$T = \frac{N}{100} \times T \tag{2.19}$$

| Keterangan | : |
|---|---|
| $T$ | : Jumlah sampel kelas minoritas. |

$$gap = random(0,1) \tag{2.20}$$

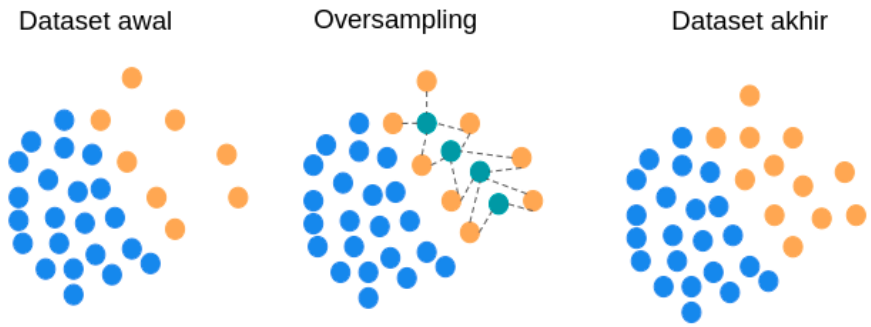
| Keterangan | : |
|---|---|
| *gap* | : Nilai acak antara 0 hingga 1. |

$$dif = f_{21} - f_{11}, f_{22} - f_{12} \tag{2.21}$$

| Keterangan | : |
|---|---|
| $f_{11}$ | : Titik sampel data orisinil sumbu X. |
| $f_{12}$ | : Titik sampel data orisinil sumbu Y. |
| $f_{21}$ | : Titik sampel data tetangga terdekat sumbu X. |
| $f_{22}$ | : Titik sampel data tetangga terdekat sumbu Y. |

$$(f_1', f_2') = (f_{11}, f_{12}) + gap \times dif \tag{2.22}$$

Keterangan :
$f_1'$ : Titik sampel data sintetis sumbu X.
$f_2'$ : Titik sampel data sintetis sumbu Y.
$f_{11}$ : Titik sampel data orisinil sumbu X.
$f_{12}$ : Titik sampel data orisinil sumbu Y.




**Gambar 2.16** Ilustrasi *oversampling* menggunakan SMOTE

| | Class 0 | Class 1 | total |
|---|---|---|---|
| Original dataset | 570 | 178 | 748 |
| (equivalent to **SMOTE percentage = 0**) | 76% | 24% | |
| **SMOTE percentage = 100** | 570 | 356 | 926 |
| | 62% | 38% | |
| **SMOTE percentage = 200** | 570 | 534 | 1104 |
| | 52% | 48% | |
| **SMOTE percentage = 300** | 570 | 712 | 1282 |
| | 44% | 56% | |

**Gambar 2.17** Perbandingan proporsi jumlah data pada suatu kelas menggunakan *SMOTE* [21]

Parameter persentase pada SMOTE bergantung pada setiap dataset. Meningkatkan jumlah kelas minoritas menggunakan SMOTE tidak menjamin akan menghasilkan akurasi model yang bagus. Oleh karena itu, diperlukan percobaan untuk menguji persentase, fitur yang berbeda, dan jumlah tetangga terdekat dan bagaimana pengaruhnya terhadap model yang dibangun [21]. Berikut merupakan

daftar masukan, *parameter* dan keluaran SMOTE .

**Tabel 2.1** Daftar masukkan SMOTE [21]

| **Nama** | ***Tipe data*** | **Deskripsi** |
|---|---|---|
| Sampel | Data tabel | Sampel data yang akan digunakan |

**Tabel 2.2** Daftar parameter SMOTE [21]

| **Nama** | **Range** | **Tipe Data** | **Default** | **Deskripsi** |
|---|---|---|---|---|
| Presentase SMOTE | $\geq 0$ | Integer | 100 | Jumlah *oversampling* dalam kelipatan 100 |
| Jumlah tetangga terdekat | $\geq 1$ | Integer | 1 | Jumlah tetangga terdekat untuk menentukan fitur untuk data sampel terbaru yang akan dibuat oleh SMOTE. |
| *Random seed* | any | Integer | 0 | *Seed* untuk melakukan generasi angka acak. |

**Tabel 2.3** Daftar keluaran SMOTE [21]

| **Nama** | ***Tipe data*** | **Deskripsi** |
|---|---|---|
| Tabel | Data tabel | Data tabel yang berisi sampel data orisinil dan sampel data baru yang dibuat oleh SMOTE. Jumlah sampel data terbaru tersebut adalah *(presentase SMOTE ÷ 100 x T)*, dengan *T* merupakan jumlah sampel kelas minoritas. |

### 2.1.8 *Hyperparameter Tuning*

*Hyperparameter tuning* adalah proses pencarian kombinasi nilai *hyperparameter* model yang mampu menghasilkan performa yang baik. Ada beberapa metode yang digunakan dalam proses *hyperparameter tuning*, yaitu *grid search*, *random search*, dan *Bayesian hyperparameter optimization* [10].

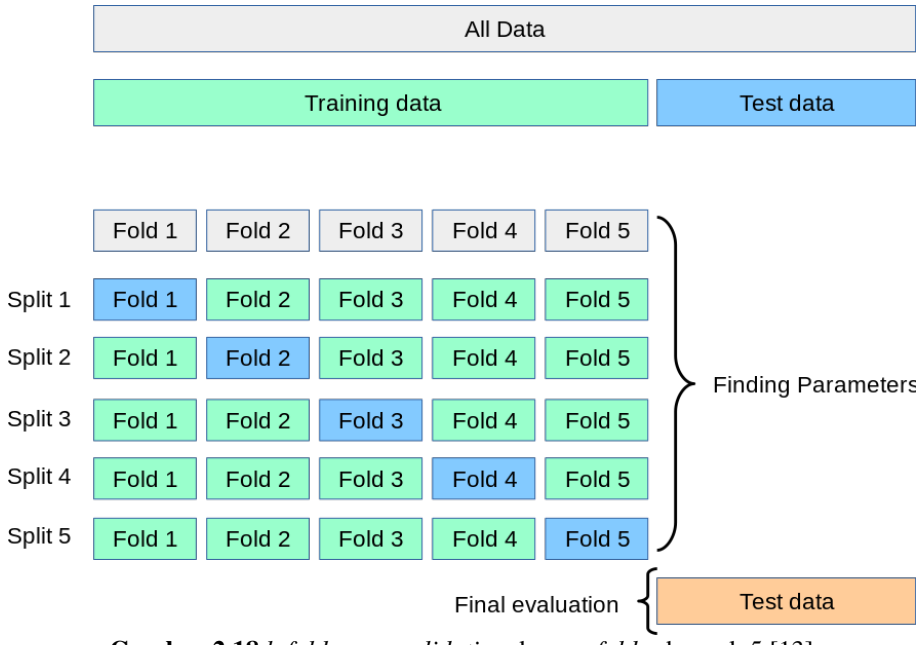
*Grid search* merupakan metode yang paling mudah untuk diterapkan. *Grid search* yaitu mencoba seluruh kombinasi dari *hyperparameter* kemudian menggunakan*data latih* untuk melatih model dan menilai performa model menggunakan data validasi. Sebagai contoh, apabila terdapat *hyperparameter* $\lambda$ = [0.01, 0.1] dan $\varepsilon$ = [0.5, 0.6], maka *grid search* akan mencoba semua kombinasi $\lambda$ dan $\varepsilon$: [0.01, 0.5], [0.01, 0.6], [0.1, 0.5], dan [0.1, 0.6] untuk melatih dan menilai performa model. Kelebihan dari *grid search* adalah mencoba setiap kombinasi dari *hyperparameter* sehingga tentu akan menemukan kombinasi *hyperparameter* yang menghasilkan model dengan hasil evaluasi terbaik. Salah satu kelemahan dari *grid search* adalah waktu pemrosesannya yang lama, khususnya ketika jumlah data yang digunakan sangat banyak [10].

*Random search* merupakan salah satu strategi dalam *hyperparameter tuning* yang membutuhkan distribusi statistik untuk setiap *hyperparameter* dan jumlah kombinasi yang akan dicoba. Kelebihan dari *random search* adalah mengurangi waktu pemrosesan *hyperparameter tuning* karena tidak mencoba setiap kombinasi dari *hyperparameter* dengan syarat bahwa jumlah percobaan yang dilakukan nilainya tidak besar [22].

*Bayesian hyperparameter optimization* merupakan merupakan salah satu strategi dalam *hyperparameter tuning* yang berusaha untuk meminimalisir waktu pemrosesan dengan menggunakan hasil evaluasi sebelumnya yang memiliki hasil evaluasi yang bagus untuk memilih nilai *hyperparameter* selanjutnya untuk dievaluasi. Kelebihan dari *bayesian* adalah waktu pemrosesannya yang lebih singkat, kekurangannya adalah *computational cost* yang tinggi [20].

### 2.1.9 *K-Fold Cross-Validation*

Dalam *machine-learning*, ketika melatih dan menguji model menggunakan data yang sama, model mampu memprediksi dengan baik pada tahap pengujian, tetapi memiliki performa buruk ketika memprediksi data baru. Hal ini disebut juga dengan *overfitting*. Untuk mengatasi ini, dataset bisa dibagi menjadi 2 bagian, data latih dan data uji. Namun, pada saat mencari *hyperparameter* terbaik, ada resiko model akan *overfitting* pada data uji ketika melakukan *hyperparameter tuning* [13]. Jika membagi dataset menjadi 3 bagian dengan menggunakan data validasi, data latih yang digunakan untuk melatih model akan semakin sedikit. Salah satu solusi untuk mengatasi masalah ini adalah menggunakan *k-fold cross validation*. *K-fold cross-validation* atau *cross-validation* adalah salah satu metode validasi model dalam *machine learning*. *Cross-validation* digunakan apabila ingin menggunakan lebih banyak data dalam dataset untuk melatih model, dalam hal ini, melakukan *data splitting* menjadi data latih (*train set*) dan data uji (*test set*), sehingga tidak lagi diperlukan data validasi (*validation set*). *Cross-validation* kemudian digunakan pada *training set* atau data latih untuk mensimulasikan *validation set*.




**Gambar 2.18** *k-fold cross-validation* dengan *fold* sebanyak 5 [13]

Berikut adalah beberapa langkah melakukan *Cross-validation* [10]:

1. Memisahkan data latih menjadi beberapa beberapa *split* data dengan jumlah *split* sama dengan jumlah fold. Umumnya, jumlah *fold* adalah 5.
2. Lalu tentukan fold yang akan digunakan sebagai data uji dan fold lainnya sebagai data latih.
3. Mengulangi langkah sebelumnya pada setiap *split* hingga semua *fold* telah digunakan sebagai data uji.
4. Hitung rata-rata hasil evaluasi dengan menggunakan data uji dari seluruh *split*.

### 2.1.10 *Confusion Matrix*

*Confusion matrix* adalah tabel yang menyimpulkan apakah model klasifikasi telah berhasil mengklasifikasikan data ke beberapa kelas. Setiap baris dalam tabel *confusion matrix* merupakan kelas yang sebenarnya, sedangkan setiap kolom merupakan kelas yang diprediksi. Gambar 2.19 merupakan ilustrasi *Confusion Matrix* pada kasus klasifikasi *email spam* sebagai kelas positif dan bukan *spam* sebagai kelas negatif.

| | spam (predicted) | not_spam (predicted) |
|---|---|---|
| spam (actual) | 23 (TP) | 1 (FN) |
| not_spam (actual) | 12 (FP) | 556 (TN) |

**Gambar 2.19** Ilustrasi *Confusion Matrix* [10].

Berikut adalah penjelasan dari gambar 2.19:

1. *True Positive* (TP) adalah keadaan ketika suatu data diprediksi sebagai kelas positif dan memang benar kelas itu tergolong ke dalam kelas positif. Pada gambar 2.19, terdapat 24 sampel data yang diprediksi dengan benar sebagai spam.
2. *True Negative* (TN) adalah keadaan ketika suatu data sebagai kelas negatif dan memang benar kelas itu tergolong ke dalam negatif. Pada gambar 2.19 terdapat 556 sampel data yang diprediksi dengan benar sebagai *email* bukan *spam*.
3. *False Positive* (FP) adalah keadaan ketika suatu data diprediksi sebagai kelas positif, namun ternyata kelas tersebut tergolong ke dalam kelas negatif. Pada

gambar 2.19 terdapat 12 sampel data yang salah diprediksi sebagai email *spam*.

4. *False Negative* (FN) adalah keadaan ketika suatu data diprediksi sebagai kelas negatif, namun ternyata kelas tersebut tergolong ke dalam kelas positif. Pada gambar 2.19 terdapat 1 sampel data yang salah diprediksi sebagai *email* bukan *spam*.

Untuk menghitung TP *rate* atau *Precision*, dapat digunakan persamaan 2.23.

$$Precision = \frac{TP}{TP + FP} \tag{2.23}$$

Sementara untuk menghitung sensitivitas atau *recall* dapat menggunakan persamaan 2.24.

$$Recall = \frac{TP}{TP + FN} \tag{2.24}$$

Untuk menghitung akurasi, dapat menggunakan persamaan 2.25.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.25}$$

Untuk menghitung F-measure atau F1-score atau F-score, dapat menggunakan persamaan 2.26 atau 2.27.

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.26}$$

$$F - measure = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2.27}$$

### 2.1.11 Pustaka Python

Pada bagian ini akan dijelaskan mengenai pustaka atau *library* yang digunakan dalam penelitian.

#### 2.1.11.1 Pandas

Pandas merupakan pustaka yang tersedia pada bahasa pemrograman python yang berfungsi untuk melakukan analisis, pengolahan dan manipulasi data.

**Tabel 2.4** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | read_csv | file_path: string | DataFrame | Membaca data dengan format csv dan merepresentasikannya dalam bentuk DataFrame. |
| 2 | DataFrame | data: array | DataFrame | Membuat DataFrame berdasarkan data masukan dengan bentuk 2 dimensi atau tabel yang berisikan baris dan kolom. |
| 3 | head | n: int | DataFrame | Melihat 5 data teratas. |
| 4 | describe | percentiles: array | DataFrame | Memberikan informasi statistik mengenai DataFrame. |
| 5 | isna | - | DataFrame | Menampilkan apakah setiap kolom memiliki data yang missing. |
| 6 | drop_duplicates | subset: array, inplace: boolean | DataFrame | Merupakan fungsi untuk menghilangkan data duplikat dalam DataFrame. |
| 6 | drop | label: string, array, axis: *int* | DataFrame | Menghapus kolom atau label tertentu dari DataFrame. |
| 7 | duplicated | subset: string, array | Series | Menampilkan apakah terdapat data duplikat pada DataFrame |
| 8 | sum | axis: int | Series atau DataFrame | Digunakan untuk menjumlahkan nilai pada sumbu tertentu dalam DataFrame. |
| 9 | corr | method | DataFrame | Menghitung korelasi antar kolom. |

#### 2.1.11.2 Numpy

Numpy merupakan pustaka dalam pemrograman python yang digunakan untuk melakukan operasi dan perhitungan matematika pada array.

**Tabel 2.5** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | triu | m: *array* | array | Menghasilkan array 2 dimensi nilai dibawah diagonal utama adalah 0. |
| 2 | ones_like | a: array, dtype: data type | ndarray | Menghasilkan array baru yang dimensi dan jenis datanya sama seperti aray masukan. |
| 3 | random_permutation | x: int atau array | ndarray | Melakukan permutasi secara acak pada int atau array. |
| 4 | sum | a: array | ndarray | Menghitung penjumlahan nilai elemen pada array. |
| 5 | quantile | a: array, q: array *float*, axis: int atau tuple, interpolation: string | series atau DataFrame | Menghitung nilai kuantil data pada sumbu tertentu . |
| 6 | array | object: array | ndarray | Membuat array dengan tipe data ndarray. |
| 7 | full | shape: int atau tuple, $fill_value$ : *int* | ndarray | Membuat array baru dengan dimensi dan nilai sesuai masukan. |
| 8 | exp | x: array | ndarray | Menghitung nilai eksponensial. |
| 9 | log | x: array | ndarray | Menghitung nilai logaritma natural. |
| 10 | where | condition: array atau boolean, x: array, y: array | ndarray | Mendapatkan element dari masukan $x$ dan $y$ berdasarkan kondisi tertentu. |

**Tabel 2.5** – Daftar Metode yang Digunakan (lanjutan)

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 11 | concatenate | a1, a2, ...: array, axis: int | ndarray | Menggabungkan array berdasarkan sumbu tertentu. |
| 12 | zeros | shape: int atau tuple | ndarray | Fungsi untuk mendapatkan array baru yang berisi nilai 0. |
| 13 | arange | start: int, step: int, stop: int | ndarray | Fungsi untuk mendapatkan array baru dengan nilai sesuai masukan *start* dan *stop* dan memiliki inverval antar nilai sebesar masukan *step*. |

#### 2.1.11.3 Scikit-Learn

Scikit-Learn merupakan pustaka *open source* yang menyediakan beragam algoritme *supervised* dan *unsupervised* untuk*machine learning*.

**Tabel 2.6** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | train_test_split | arrays: array, test_size: float, int, random$_{s}tate$ : *int* | array | Memisahkan dataset menjadi data uji dan data latih. |
| 2 | confusion_matrix | y_true: array, y_pred: array | confusion matrix | Mendapatkan confusion matrix untuk evaluasi klasifikasi. Pada klasifikasi biner akan mengeluarkan perhitungan TN, FN, TP, dan FP. |
| 3 | accuracy_score | y_true: array, y_pred: array | float | Fungsi yang digunakan untuk menghitung akurasi dari model klasifikasi *machine learning* yang digunakan. |

#### 2.1.11.4 Imbalanced-Learn

Imbalanced-Learn merupakan pustaka *open-source* yang digunakan untuk menangani klasifikasi dengan kelas yang *imbalance*.

**Tabel 2.7** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | SMOTE | sampling_strategy: float, random_state: int, k_neighbors: int | dataset | Merupakan kelas yang digunakan untuk melakukan *oversampling* dengan metode *SMOTE*. |

#### 2.1.11.5 Matplotlib

Matplotlib merupakan pustaka yang digunakan untuk membuat visualisasi data pada Python.

**Tabel 2.8** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | figure | figszie: tuple | Figure | Membuat figur baru ataumengaktifkan figur yang sudah ada sebelumnya. |
| 2 | show | - | Figure | Menampilkan plot atau visualisasi dari figure yang telah didefinisikan sebelumnya. |

#### 2.1.11.6 Seaborn

Seaborn merupakan pustaka visualisasi data berbasis pustaka Matplotlib yang digunakan untuk mempermudah visualisasi data.

**Tabel 2.9** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | heatmap | data: array 2 dimensi, cmap: string, mask: bool array atau DataFrame, annot: bool, fmt: str, center: float, linewidths: float | matplotlib Axes | Fungsi yang digunakan untuk membuat visualisasi korelasi antarkolom menggunakan *heatmap*. |

#### 2.1.11.7 XGboost

XGBoost merupakan pustaka yang digunakan untuk membuat model XGboost.

**Tabel 2.10** Daftar Metode yang Digunakan

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1. | *¡nit* | objective: string, eta: float, n_estimators: int, max_depth: int, gamma: float, reg_lambda: float, base_score: float, nthread: int | object | Inisialisasi objek kelas XGboost untuk melakukan klasifikasi. |
| 2 | fit | X: array, y:array | - | Melakukan pelatihan model XGboost dengan menggunakan X sebagai variabel independen dan y sebagai variabel dependen. |
| 3 | predict | X: array | array | Melakukan prediksi menggunakan model XGboost dengan variabel X sebagai masukkan dan keluaran berupa hasil prediksi. |

## 2.2 Tinjauan Studi

Pada bagian ini akan dijelaskan mengenai perbandingan dari berbagai penelitian terkait metode deteksi transaksi penipuan kartu kredit.

### 2.2.1 *State Of The Art*

Pada Tabel 2.11 diberikan penjelasan mengenai perbandingan dari beberapa penelitian terkait deteksi transaksi penipuan kartu kredit:

**Tabel 2.11** Tinjauan Studi

| Jurnal | Metode | Hasil Penelitian |
|---|---|---|
| Sanmati Marabad, "Credit Card Fraud Detection using Machine Learning" in Asian Journal of Convergence in Technology, 2021 | 1. Logistic Regression<br>2. K-Nearest Neighbors (KNN)<br>3. Decision Tree<br>4. XGBoost<br>5. Random Forest<br>6. SMOTE sebagai metode *oversampling* | Penelitian menggunakan 89% data latih dan 20% data dan menyeimbangkan kelas yang *imbalance* menggunakan SMOTE. Akurasi dari KNN, *Decision Tree*, *Logistic Regression*, *Random Forest*, dan XGBoost sangat baik mencapai 99%. Jika dilihat berdasarkan evaluasi F1 score, XGBoost meraih hasil tertinggi sebesar 84%. Berdasarkan evaluasi *precision*, Random Forest meraih hasil tertinggi sebesar 90%. Berdasarkan evaluasi *recall*, XGBoost meraih hasil tertinggi sebesar 79%. Untuk masalah transaksi kartu kredit sangatlah penting, sehingga XGBoost merupakan model terbaik pada penelitian ini. |

**Tabel 2.11** – Tinjauan Studi (lanjutan)

| Jurnal | Metode | Hasil Penelitian |
|---|---|---|
| T. Ma, L. Wu, S. Zhu and H. Zhu, "Multiclassification prediction of clay sensitivity using extreme gradient boosting based on imbalanced dataset", Applied Sciences, vol. 12, no. 3, 2022. | 1. XGBoost<br>2. Artificial Neural Network (ANN)<br>3. Naive Bayes (NB)<br>4. SMOTE<br>5. *5-fold cross-validation* | Menerapkan algoritme XGBoost, ANN, dan NB untuk memprediksi sensitifitas tanah liat. Dataset yang digunakan merupakan dataset yang imbalance, sehingga diterapkan metode SMOTE sebagai *oversampling*. Metode *cross-validation* juga diterapkan dalam proses pelatihan. Evaluasi dilakukan terhadap model XGBoost, ANN, NB, dan XGBoost tanpa SMOTE. Hasil terbaik diperoleh oleh XGBoost dengam SMOTE dengan *precision* sebesar 73%, *recall* 72%, dan *f1-score* 72% [23]. |
| A. A. Taha and S. J. Malebary, "An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine", IEEE Access, vol. 8, pp. 25 579–25 587, 2020. | 1. *Bayesian-based hyperparameter optimization*<br>2. *5-fold Cross Validation*<br>3. LightGBM | Algoritma LightGBM yang sudah dioptimasi menghasilkan akurasi yang terbaik dari 2 dataset yang diuji dibandingkan dengan algoritma *machine learning* lainnya dalam mendeteksi penipuan kartu kredit. Pada dataset pertama, LightGBM memperoleh akurasi 98,40%, *recall* 40,59%, *precision* 97,34% dan *f1-score* 56,95%. Pada dataset kedua, LightGBM memperoleh akurasi sebesar 98,35%, recall 28,33%, *precision* 91,72%, dan *f1-score* 43,27%. Berdasarkan hasil penelitian ini, LightGBM masih memiliki kekurangan dimana nilai *recall* yang rendah. *Hyperparameter* yang di-*tuning* dalam penelitian tersebut adalah *num_leaves*, *max_depth*, dan *learning_rate*. |

## 2.3 Tinjauan Objek

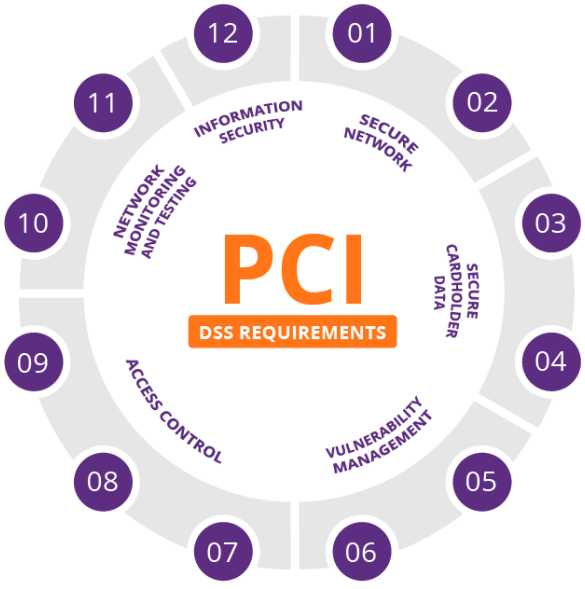
Pada bagian ini akan diulas mengenai objek-objek yang terkait dengan deteksi penipuan transaksi kartu kredit.

### 2.3.1 Kartu Kredit

Kartu kredit adalah alat pembayaran yang diterbitkan oleh bank yang memungkinkan pemegang kartu untuk meminjam uang dan membayar barang dan jasa kepada *merchant* yang menerima metode pembayaran kartu kredit. Pemegang kartu kredit memiliki kewajiban untuk membayar uang yang dipinjam dengan bunga tertentu dalam jangka waktu tertentu [24]. Kartu kredit juga dapat digunakan dalam transaksi pembayaran digital.

### 2.3.2 Standar Keamanan Data dalam Industri Kartu Pembayaran

*Payment Card Industry Data Security Standard* (PCI DSS) merupakan standar keamanan global yang dibuat untuk membantu institusi finansial memproses transaksi pembayaran untuk mengurangi resiko penipuan. Standar PCI ini dibentuk oleh institusi finansial seperti American Express, Discover Financial Services, JCB, MasterCard dan Visa Inc [25]. PCI DSS diadopsi oleh perusahaan kartu untuk memproses, menyimpan dan mentransmisi data pemegang kartu atau data autentikasi yang sensitif dan berlaku untuk semua pihak yang terlibat. PCI DSS terdiri dari beberapa langkah yang mencerminkan praktik terbaik dalam keamanan.




**Gambar 2.20** PCI DSS

Berikut merupakan tujuan dan standar keamanan PCI DSS:

1. Membuat dan memelihara jaringan dan sistem yang aman
   (a) Instalasi dan pemeliharan konfigurasi *firewall* untuk melindungi data pemegang kartu.
   (b) Jangan menggunakan password bawaan dari vendor sebagai password system.
2. Melindungi data pemegang kartu.
   (a) Data pemegang kartu harus dilindungi.
   (b) Melakukan enkripsi pada saat melakukan transmisi data pemegang kartu pada jaringan publik.
3. Memelihara program manajemen kerentanan
   (a) Melindungi semua sistem terhadap *malware* dan secara periodik melakukan pembaharuan perangkat lunak *antivirus* atau program.
   (b) Mengembangkan dan memelihara sistem dan aplikasi yang aman.
4. Mengimplementasikan kontrol akses yang kuat.
   (a) Membatasi akses data pemegang kartu berdasarkan kebutuhan bisnis.
   (b) Identifikasi dan autentikasi pada komponen sistem.
   (c) Membatasi akses secara fisik terhadap data pemegang kartu.
5. Secara periodik melakukan pemanatauan dan pengujian jaringan.
   (a) Akses terhadap data pemegang kartu dan jaringan harus dipantau.
   (b) Secara periodik melakukan pengujian terhadap keamanan sistem.
6. Memelihara informasi kebijakan keamanan.
   (a) Mempertahankan kebijakan yang membahas keamanan informasi.

#### 2.3.2.1 Transaksi Pembayaran Kartu Kredit

Transaksi pembayaran menggunakan kartu kredit melibatkan beberapa pihak, diantaranya sebagai berikut:

1. Kustomer: Pihak yang melakukan pembelian barang atau jasa menggunakan kartu kredit dari *merchant* atau pedagang.
2. *Merchant* atau pedagang: Pihak penjual atau penyedia jasa.
3. *Acquirer*: Bank yang digunakan oleh *merchant*.
4. *Issuer*: Bank yang menyediakan kartu kredit kepada *user*.

Berikut merupakan proses transaksi pembayaran menggunakan kartu kredit [26]:

1. Kustomer melakukan pembelian barang atau jasa dari *merchant* dengan menggunakan kartu kredit sebagai metode pembayaran. Data-data kartu akan masuk ke dalam sistem pembayaran *point-of-sale* (POS) *merchant*
2. Sistem *point-of-sales* yang ada pada *merchant* akan mengirimkan data-data kartu kepada *acquirer*.
3. *Acquirer* akan mengirimkan data-data kartu ke dalam jaringan perusahaan kartu kredit, seperti Visa atau MasterCard yang akan meneruskannya kepada *issuer*.
4. Issuer akan melaukan otorisasi terhadap data-data kartu. Pada tahap ini, perusahaan kartu kredit dapat menggunakan perangkat lunak atau model klasifikasi untuk mendeteksi transaksi penipuan kartu kredit.
5. *Issuer* akan memberikan otorisasi terhadap transaksi yang dilakukan. Dengan otorisasi tersebut, *issuer* bersedia membayar transaksi kustomer yang bersangkutan.
6. Jaringan persuahaan kartu kredit akan meneruskan otorisasi kepada *acquirer*.
7. *Acquirer* akan meneruskan otorisasi kepada *merchant*.
8. *Merchant* akan menyelesaikan transaksi dan mengeluarkan rincian pembayaran kepada kustomer.


**Gambar 2.21** Ilustrasi proses transaksi pembayaran menggunakan kartu kredit pada jaringan *Mastercard*.

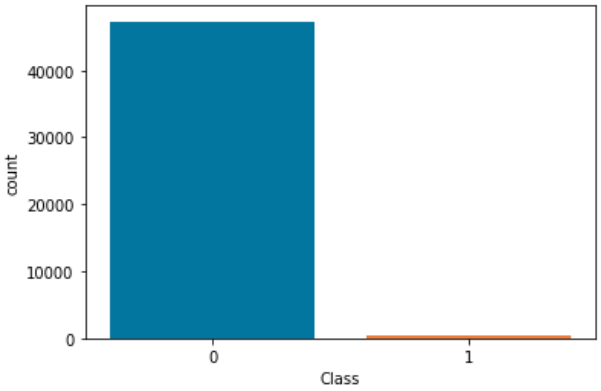
#### 2.3.2.2 Penipuan Kartu Kredit

Penipuan kartu kredit merupakan salah satu bentuk kejahatan penipuan identitas yang terjadi karena adanya pencurian identitas. Pencurian identitas merupakan kejahatan ketika seseorang menggunakan informasi dan identitas milik orang lain seperti nama, tangga lahir, mutasi rekening untuk mengakses data seseorang atau membuka akun baru dengan nama orang lain [1]. Kejahatan ini memungkinkan seseorang menggunakan kartu kredit orang lain atau menggunakan akun dengan kartu kredit orang lain untuk membuat transaksi [1]. Hal ini dapat terjadi apabila kartu kredit dicuri oleh orang lain, informasi mengenai kartu kredit seperti nomor keamanan atau *password* diketahui oleh orang lain, atau informasi kartu kredit yang telah tersebar secara online hasil dari kebocoran data atau peretasan situs sehingga memungkinkan orang lain untuk mendapatkannya [1].

### 2.3.3 Dataset Transaksi Kartu Kredit

Dataset transaksi kartu kredit diperoleh dari pemegang kartu kredit di Eropa yang menggunakan kartu kredit selama 2 hari pada bulan September tahun 2013 yang dilakukan oleh pemegang katu kredit di Eropa. Data yang akan dipakai dari dataset sejumlah 47773 dimana 47300 diantaranya merupakan transaksi *non-fraud* dan 473 diantaranya merupakan transaksi *fraud*, sehingga dataset ini sangat *imbalance* dengan rasio sekitar 1% transaksi penipuan dari dataset yang akan dipakai. Dataset ini terdiri dari 31 fitur. Berikut merupakan penjelasan dari masing-masing fitur:

1. *Time*: Jumlah waktu dalam detik yang berlalu antara setiap transaksi dengan transaksi pertama yang ada dalam dataset.
2. V1-V28 (28 fitur): Fitur yang dihasilkan dari transformasi PCA dan nama fitur yang disamarkan. Hal ini dilakukan oleh penerbit dataset dengan tujuan untuk melindungi data identitas asli yang bersifat privasi dan sensitif.
3. *Amount*: Nominal transaksi yang terjadi.
4. *Class*: Label yang menandakan transaksi pada data baris tersebut termasuk ke dalam penipuan atau bukan. Jika *Class* bernilai 1 maka transaksi tersebut termasuk kedalam transaksi penipuan. Jika *Class* bernilai 0, maka transaksi tersebut bukan merupakan transaksi penipuan.




**Gambar 2.22** Kelas yang tidak seimbang pada dataset yang akan digunakan. Kelas 1 merupakan kelas *fraud* atau transaksi penipuan dan kelas 0 merupakan kelas *non-fraud* atau transaksi bukan penipuan.

Pada gambar 2.22 dapat dilihat bahwa kelas *fraud* (1) memiliki jumlah yang sangat sedikit jika dibandingkan kelas *non-fraud* (0). Oleh karena itu dataset perlu dilakukan *oversampling* untuk meningkatkan proporsi jumlah data kelas *fraud* menggunakan SMOTE.

# BAB 3 ANALISIS DAN PERANCANGAN SISTEM

Bab ini memaparkan analisis masalah yang diatasi berserta pendekatan dan alur kerja dari perangkat lunak yang dikembangkan, mengimplementasikan metode yang digunakan dan hasil yang akan ditampilkan.

## 3.1 Analisis Masalah

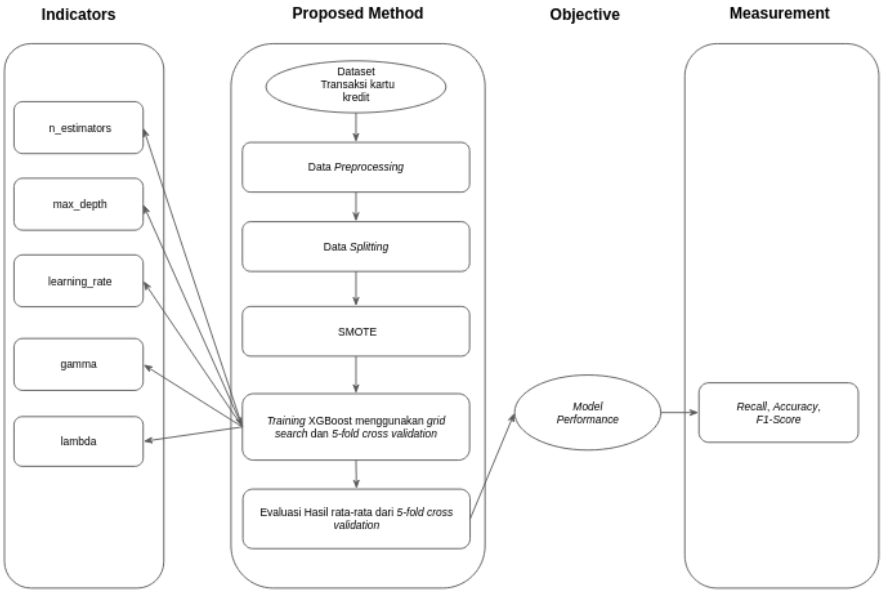
Pada Bab 1 telah dijelaskan masalah yang dihadapi terkait deteksi transaksi kartu kredit. Kemudian dijelaskan juga bahwa metode yang akurat dalam mendeteksi penipuan kartu kredit dibutuhkan untuk mencegah terjadinya kerugian yang besar akibat penipuan tersebut. Pada penelitian ini, metode klasifikasi *machine learning* XGBoost digunakan karena memiliki hasil *recall* tertinggi pada pengujian [6]. *Hyperparameter tuning* akan digunakan untuk membantuk memaksimalkan akurasi XGBoost dengan memilih nilai *hyperparameter* dengan hasil evaluasi terbaik. Metode *grid search* digunakan dalam penelitian karena implementasinya yang paling mudah dibandingkan dengan dua metode lainnya, serta karena keterbatasan sumber daya yang digunakan. Agar penelitian ini lebih akurat dalam menguji keseluruhan dataset, maka akan digunakan *k-fold cross-validation* dengan jumlah fold sebanyak 5.

Pada penelitian yang dilakukan oleh penulis bertujuan untuk mencari nilai *hyperparameter* terbaik XGBoost untuk mendeteksi transaksi penipuan kartu kredit. *Hyperparameter* terbaik dengan hasil terbaik akan dievaluasi kembali pada pengujian selanjutnya dengan menggunakan teknik *oversampling* SMOTE.

Masukkan pada penelitian ini adalah dataset transaksi kartu kredit. Keluaran atau hasil dari sistem deteksi penipuan kartu kredit yang dirancang adalah berupa nilai *hyperparameter* terbaik, akurasi, *precision*, *recall*, *F-measure* pada data latih sebagai data *hold out* pada proses evaluasi terakhir untuk membuktikan seberapa baik kinerja XGBoost dalam mendeteksi transaksi penipuan kartu kredit.

## 3.2 Kerangka Pemikiran

Berikut ini adalah kerangka pemikiran dari metode yang diusulkan untuk melakukan deteksi transaksi penipuan kartu kredit.




**Gambar 3.1** Kerangka Pemikiran

Pada gambar 3.1 adalah kerangka pemikiran yang telah disusun dalam bentuk diagram:

1. Data *Preprocessing*: Menghilangkan *missing values* dan data yang duplikat dan memilih fitur yang akan digunakan.
2. Data *Splitting*: Membagi data menjadi data latih dan data uji.
3. SMOTE: Menerapkan metode *SMOTE* untuk meningkatkan proporsi jumlah data transaksi *fraud*.
4. *Training*: Melatih model XGBoost menggunakan *grid search* dan *5-fold cross validation* menggunakan data latih.
5. Evaluasi: Mengambil rata-rata hasil pengukuran dari proses *cross validation* pada setiap fold.
6. *Indicators*: Merupakan *hyperparameter* pada model XGBoost. *Hyperparameter* yang akan diuji adalah *n_estimators*, *learning_rate* ($\eta$), gamma ($\gamma$), dan lambda ($\lambda$).

### 3.2.1 Penjelasan Indikator

Pada bagian ini akan dijelaskan mengenai indikator *hyperparameter* yang digunakan pada XGBoost. Berikut ini merupakan penjelasan indikator yang akan digunakan pada XGBoost:

1. *n_estimators*: Merupakan banyaknya *tree* yang akan dibentuk oleh XGBoost.

2. *max_depth*: Merupakan kedalaman maksimum *tree* yang dapat dibentuk oleh XGBoost.

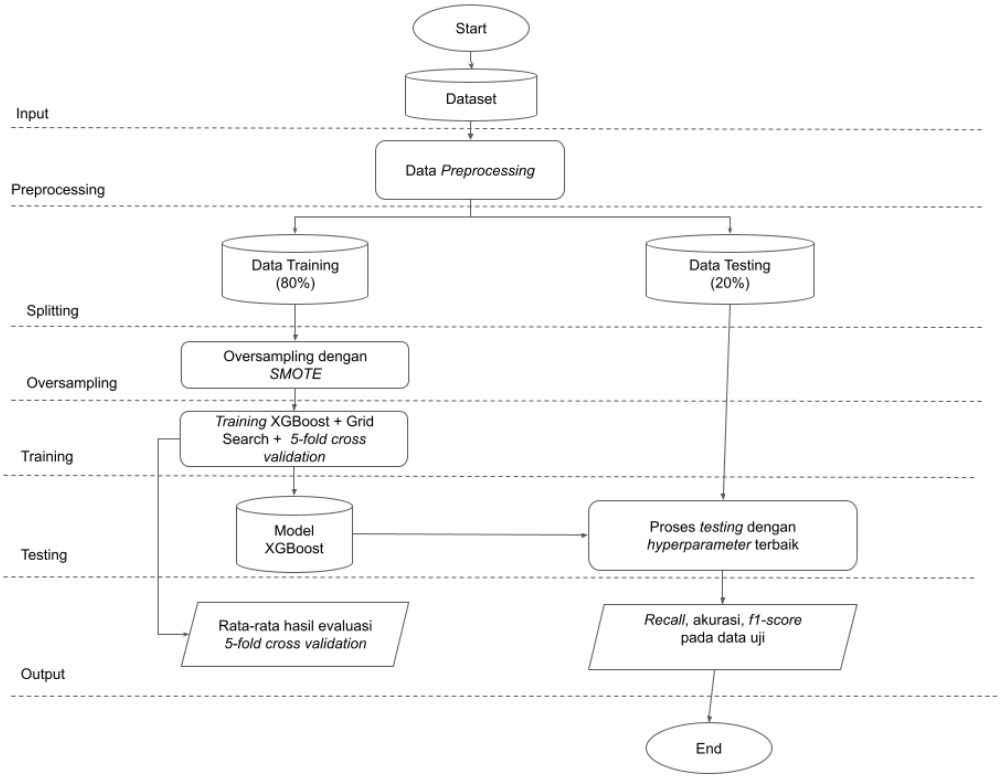
3. *Learning rate* ($\eta$): Merupakan besaran nilai yang harus diambil untuk meminimalkan *loss function*. Pada XGBoost, Nilai ini akan digunakan sebagai nilai *scaling* pada nilai *output* setiap *tree*. *Learning rate* yang terlalu besar mengakibatkan model akan *overshoot*, jika nilainya terlalu kecil maka proses pelatihan model akan semakin lama.

4. *gamma* ($\gamma$): Merupakan konstanta pengurang pada perhitungan yang mengurangi nilai *gain*. Apabila nilai gain $< 0$ maka *branch* tidak akan dikembangkan.

5. *lambda* ($\lambda$): Merupakan konstanta regularisasi pada XGBoost.

## 3.3 Analisis Urutan Proses Global

Dalam sistem deteksi transaksi penipuan kartu kredit terdapat bebrapa proses, yang berisi data *preprocessing*, data *splitting*, penerapan teknik *oversampling* menggunakan SMOTE pada data latih, proses *training* XGBoost menggunakan *5-fold cross validation* dan *grid search*, evaluasi rata-rata hasil metrik *5-fold cross-validation*), memilih *hyperparameter* terbaik untuk selanjutnya dilakukan proses pengujian pada data uji, dan evaluasi menggunakan *confusion matrix*.




**Gambar 3.2** *Flowchart* Proses Global

Berikut adalah uraian proses global yang dilakukan dalam penelitian ini yang terlihat pada Gambar 3.2:

1. Masukan berupa file CSV berisi data transaksi kartu kredit [7].
2. *File* CSV akan disimpan kedalam bentuk DataFrame, yang kemudian akan dilakukan *data preprocessing* untuk standarisasi data.
3. Tahap data *preprocessing* menghilangkan *missing value* pada data dan menghilangkan data yang duplikat.
4. Data *splitting* membagi data menjadi data latih dan data uji. Data uji akan digunakan sebagai data *hold out* yang berarti data uji akan digunakan sebagai evaluasi terakhir dan tidak akan digunakan dalam proses training dan *hyperparameter* tuning seperti pada gambar 2.18.
5. Menerapkan teknik *oversampling* SMOTE pada data latih untuk memperbanyak data dengan label *fraud* (*'Class'* = 1).
6. Membuat kombinasi *hyperparameter* yang akan diuji menggunakan metode *grid search*.

7. Proses *training* pada XGBoost akan menggunakan *5-fold cross validation* dengan menggunakan kombinasi *hyperparameter* hasil dari proses sebelumnya. Berikut merupakan proses *cross-validation*:

   (a) Membuat model XGBoost menggunakan *hyperparameter* dari kombinasi *hyperparameter* yang telah dibuat sebelumnya.

   (b) Memisahkan data latih menjadi data latih untuk 4 fold lainnya dan data uji untuk proses cross-validation. Data uji disini berbeda dengan data uji yang bersifat sebagai *hold out*. Data uji akan menjadi 1 *fold* dan 4 *fold* lainnya merupakan data latih. Setiap iterasi *cross-validation*, fold yang digunakan pada data uji harus berbeda sehingga semua fold pernah menjadi data uji.

   (c) Melakukan proses pelatihan menggunakan data latih dan *hyperparameter* yang sudah di *tuning*.

   (d) Melakukan testing menggunakan data uji menggunakan model yang didapat dari proses sebelumnya dari proses sebelumnya menggunakan .

   (e) Mengukur performa model menggunakan metrik *recall, accuracy, f1-score*.

8. Mencatat hasil evaluasi dan menghitung rata-rata setiap metrik pengukuran dari proses *5-fold cross validation*.

9. Memilih *hyperparameter* terbaik berdasarkan hasil rata-rata evaluasi *5-fold cross-validation*.

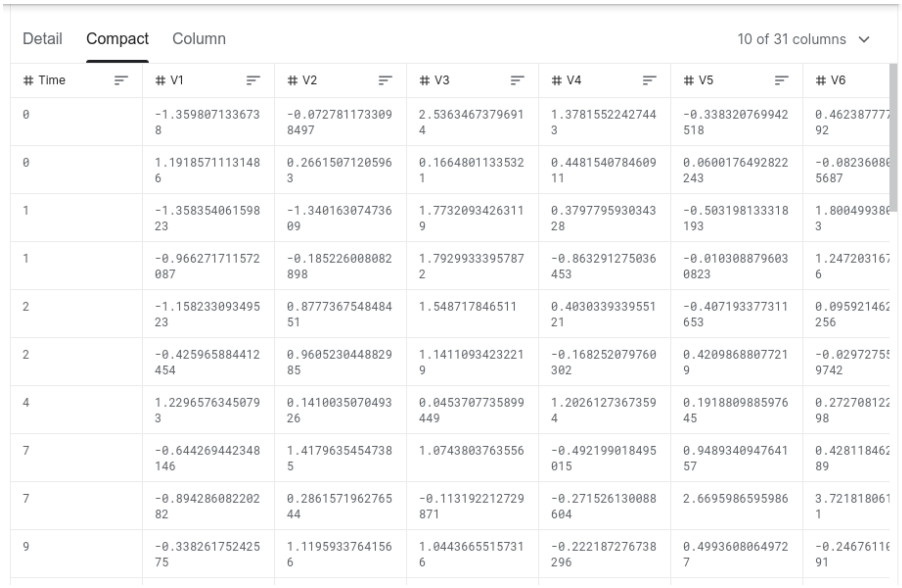
10. Membentuk model XGboost dengan *hyperparameter* terbaik berdasarkan rata-rata evaluasi *5-fold cross-validation* untuk dilakukan proses pengujian menggunakan data uji yang telah ditetapkan sebagai data *hold out* untuk evaluasi akhir, dengan hasil pengukuran menggunakan *confusion matrix* untuk mengetahui *precision, recall, accuracy, dan f1-score*.

11. Keluaran berupa XGBoost dengan *hyperparameter* yang memiliki rata-rata evaluasi *5-fold cross-validation* terbaik dan waktu pemrosesan terbaik serta hasil pengujian berupa *confusion matrix* pada data uji sebagai data *hold out*.

## 3.4 Analisis Data Sampling

Dalam penelitian ini, dataset berupa 1 file dengan format Comma Separated Value (CSV). Fitur yang digunakan dalam penelitian adalah fitur Time,

V1-V28, Amount, dan Class karena seluruh fitur tersebut memuat data penting mengenai transaksi yang terjadi. Data yang akan dipakai dari dataset sejumlah 47773 dimana 47300 diantaranya merupakan transaksi *non-fraud* dan 473 diantaranya merupakan transaksi *fraud*, sehingga dataset ini sangat *imbalance* dengan rasio sekitar 1% transaksi penipuan dari dataset yang akan dipakai.

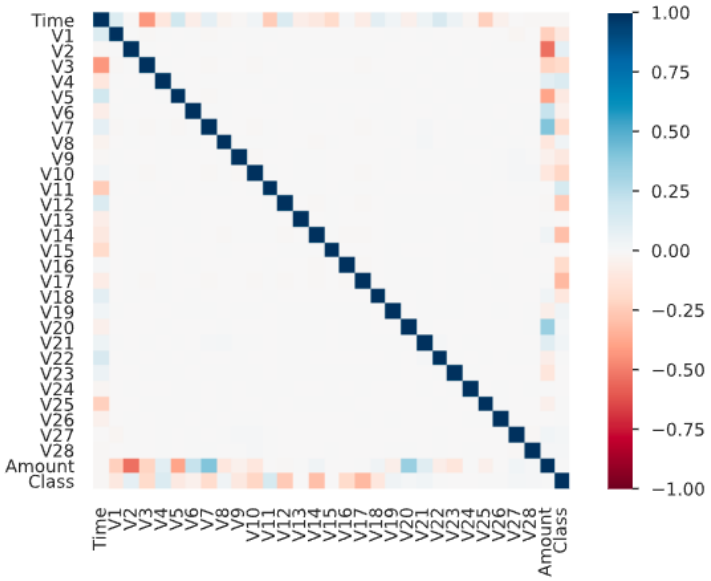

Detail Compact Column 10 of 31 columns

| # Time | # V1 | # V2 | # V3 | # V4 | # V5 | # V6 |
|---|---|---|---|---|---|---|
| 0 | -1.3598071336738 | -0.072781173 3098497 | 2.536346737 96914 | 1.378155224 27443 | -0.338320769 942518 | 0.462387777 92 |
| 0 | 1.191857111 31486 | 0.266150712 05963 | 0.166480113 35321 | 0.448154078 460911 | 0.060017649 2822243 | -0.082360 85687 |
| 1 | -1.358354061 59823 | -1.340163074 73609 | 1.773209342 63119 | 0.379779593 034328 | -0.503198133 318193 | 1.800499 383 |
| 1 | -0.966271711 572087 | -0.185226008 082898 | 1.792993339 578572 | -0.863291275 036453 | -0.010308879 603 0823 | 1.247203 167 6 |
| 2 | -1.158233093 49523 | 0.877736754 848451 | 1.548717846 511 | 0.403033933 955121 | -0.407193377 311653 | 0.095921 462 256 |
| 2 | -0.425965884 412454 | 0.960523044 882985 | 1.141109342 322 19 | -0.168252079 760302 | 0.420986880 77219 | -0.029727 59742 |
| 4 | 1.229657634 50793 | 0.141003507 049326 | 0.045370773 5899449 | 1.202612736 73594 | 0.191880988 597645 | 0.272708 12298 |
| 7 | -0.644269442 348146 | 1.417963545 473 85 | 1.074380376 3556 | -0.492199018 495015 | 0.948934094 764157 | 0.428118 4689 |
| 7 | -0.894286082 20282 | 0.286157196 276544 | -0.113192212 729871 | -0.271526130 088604 | 2.669598659 5986 | 3.721818 061 1 |
| 9 | -0.338261752 42575 | 1.119593376 415 66 | 1.044366551 573 16 | -0.222187276 738296 | 0.499360806 4972 7 | -0.246761 691 |

**Gambar 3.3** Dataset transaksi pembayaran kartu kredit

## 3.5 Data *Preprocessing*

Tahap ini dilakukan untuk membersihkan data dari *missing values* dan *duplicate values* yang dapat merusak model yang akan dibangun. Setelah dilakukan pemeriksaan, tidak ada satupun *missing values* dalam dataset. Tahap selanjutnya adalah menghapus data yang duplikat. Terdapat 1081 duplikat dalam dataset yang perlu dihapus. Pada tahap ini juga dilakukan visualisasi matrix korelasi untuk melihat hubungan antar variabel sesuai gambar 3.4 dan gambar 3.5.




**Gambar 3.4** *Pearson Heatmap Correlation Matrix*

Matriks korelasi *pearson* mengukur korelasi linear antar 2 variabel. Gambar 3.4 menunjukkan bahwa terdapat beberapa variabel yang memiliki korelasi yang tinggi baik korelasi positif yang ditandai dengan *heatmap* berwarna biru ataupun korelasi negatif yang ditandai dengan *heatmap* berwarna merah apabila diukur menggunakan matriks korelasi pearson, namun ada banyak variabel yang tidak memiliki korelasi yang ditandai dengan *heatmap* berwarna putih. Berdasarkan gambar 3.4 dapat disimpulkan sebagai berikut:

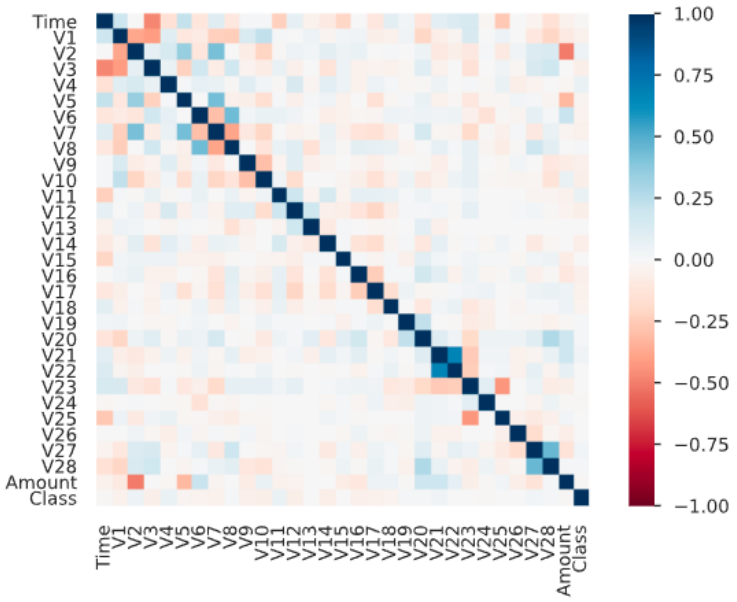
1. Variabel *Amount* memiliki korelasi negatif yang tinggi dengan variabel V2, dan memiliki korelasi positif yang tinggi dengan variabel V7, 20.

2. Variabel V3 memiliki korelasi negatif yang tinggi dengan variabel *Time* dan memiliki dengan variabel *Amount* dan *Class*

3. Variabel *Time* memiliki beberapa korelasi negatif dengan V11 dan V25 dan korelasi negatif yang tinggi dengan V3. Variabel ini juga memiliki sedikit korelasi positif dengan variabel V5, V11 dan V22.

4. Variabel *Class* memiliki korelasi negatif yang cukup tinggi dengan variabel V12, V14, dan V17. Variabel ini juga memiliki korelasi positif dengan variabel V11 meskipun nilainya cukup rendah.

5. Banyak variabel yang tidak memiliki korelasi antar satu dengan yang lainnya yang ditandai dengan *heatmap* berwarna putih. Hal ini dapat terjadi apabila korelasi antar variabel tersebut tidak memiliki hubungan yang linear.




**Gambar 3.5** *Spearman Heatmap Correlation Matrix*

Matriks korelasi *spearman* mengukur korelasi non-linear antar 2 variabel. Gambar 3.5 menunjukkan bahwa terdapat beberapa variabel yang memiliki korelasi yang tinggi baik korelasi positif ataupun korelasi negatif apabila diukur menggunakan matriks korelasi *spearman*. Ada beberapa variabel yang tidak memiliki korelasi namun jumlahnya jauh lebih sedikit apabila dibandingkan dengan matriks korelasi *pearson* sesuai gambar 3.4. Berdasarkan gambar 3.5 dapat disimpulkan sebagai berikut.

1. Variabel *Time* memiliki korelasi negatif dengan variabel V11, V15,V25 dan korelasi negatif yang cukup tinggi dengan variabel V3. Variabel ini juga memiliki korelasi positif dengan variabel V1, V5, V21, V22, dan V23.

2. Variabel V1 memiliki korelasi positif dengan variabel *time* dan variabel V10 serta memiliki korelasi negatif yang cukup tinggi dengan variabel V2 dan V3.

3. Variabel V2 memiliki korelasi negatif dengan cukup tinggi dengan variabel V1 dan *Amount* serta memiliki korelasi positif yang cukup tinggi dengan variabel V5 dan V7.

4. Variabel V5 memiliki korelasi positif yang cukup tinggi dengan variabel V2 dan V7 dan korelasi negatif yang cukup tinggi dengan variabel *Amount*.

5. Terdapat beberapa variabel lainnya yang memiliki korelasi positif yang cukup tinggi diantaranya V6-V8, V21-V22, dan V27-V28.

6. Beberapa variabel yang memiliki korelasi positif dan negatif yang rendah.

Matriks korelasi *spearman* pada gambar 3.5 mampu menangkap korelasi antar variabel lebih baik daripada matriks korelasi *pearson* pada gambar 3.4 karena jumlah variabel yang tidak memiliki korelasi lebih sedikit jika dibandingkan dengan matriks korelasi *spearman*, meskipun matriks korelasi pearson juga mampu menangkap beberapa korelasi antarvariabel yang ditandai dengan warna yang lebih tebal dibandingkan matriks korelasi spearman. Dapat disimpulkan bahwa terdapat lebih banyak korelasi non-linear antar variabel dalam dataset yang dipakai.

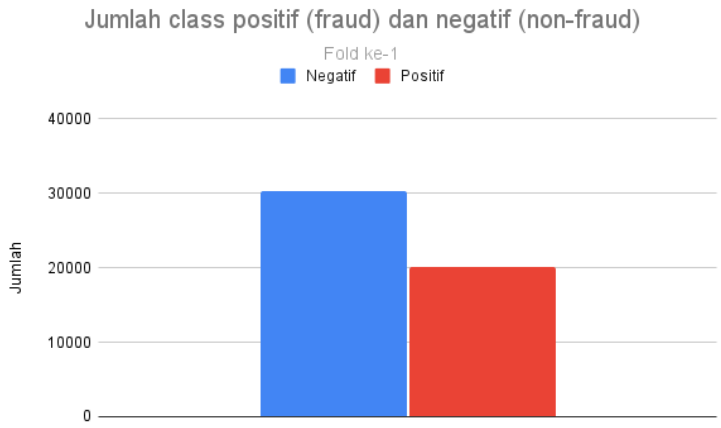
Berdasarkan penelitian [6], semua fitur dalam dataset digunakan. Oleh karena itu, dalam penelitian ini, semua fitur dalam dataset akan digunakan.

## 3.6 Data *splitting*

Pada tahap ini akan dilakukan pembagian dataset untuk proses pelatihan dan proses pengujian. Dataset akan dibagi menjadi 20% data uji dan 80% data latih. Data latih terdiri dari 38.218 data dengan 37.840 diantaranya merupakan data transaksi bukan penipuan dan 378 diantaranya merupakan data transaksi penipuan. Data uji terdiri dari sebanyak 9.555 data dengan 9.460 diantaranya merupakan data transaksi bukan penipuan dan 95 diantaranya merupakan data transaksi penipuan.

## 3.7 *Oversampling* dengan SMOTE

Berdasarkan gambar 2.22, terlihat bahwa dataset sangat *imbalance* dengan rasio kelas *fraud* sebesar 1% dari dataset. Pada tahap ini dilakukan *oversampling* pada data latih untuk menyelesaikan masalah dataset yang *imbalance* menggunakan SMOTE. Proses Oversampling menggunakan SMOTE ini akan digunakan dalam *5-fold cross-validation* ketika membagi data latih menjadi 5 fold dengan 4 fold akan menjadi data latih dan 1 fold akan menjadi data uji dan ketika melakukan pelatihan dengan kombinasi *hyperparameter* terbaik sebelum melakukan pengujian menggunakan data uji. Oversampling SMOTE akan dilakukan pada data pada kelas positif yang terdapat dalam data latih untuk meningkatkan jumlah data tersebut. Gambar 3.6 menunjukkan perbandingan data kelas positif yang berjumlah 20.181 dan kelas negatif yang berjumlah 30.272 pada fold pertama dalam data latih.




**Gambar 3.6** Data latih setelah dilakukan *oversampling* dengan SMOTE pada fold ke 1.

## 3.8 Analisis kasus

Pada bagian ini dilakukan analisis tahapan proses dengan melakukan perhitungan manual. Untuk analisis kasus, *hyperparameter* regularisasi yang akan digunakan adalah $\lambda$ sebesar 1, kemudian nilai $\gamma$ sebesar 0 dan *min_child_weight* sebesar 0 untuk mencegah adanya pemberhentian pada saat membangun *tree*. Sedangkan *hyperparameter* untuk $\eta$ (*learning rate*) senilai 0.4 dan jumlah *tree* yang akan digunakan (*n_estimators*) senilai 2 akan digunakan. Jumlah fitur yang akan digunakan pada analisis kasus sebanyak 2, yaitu fitur V1 dan V2. Karena nilai *hyperparameter* untuk jumlah fitur yang akan digunakan (*max_features*) adalah 'auto' [19], maka jumlah fitur yang akan digunakan untuk proses pelatihan adalah sebanyak 2 ('auto' merupakan jumlah fitur maksimal yang akan digunakan, jumlah fitur dalam dataset analisis kasus adalah 2).

Berikut merupakan contoh dataset yang akan digunakan dalam analisis kasus seperti yang terlihat pada gambar di bawah ini.

| Index | v1 | v2 | Class |
|---|---|---|---|
| 0 | -30.55238004 | 16.71338924 | 1 |
| 1 | -25.82598215 | 19.16723901 | 1 |
| 2 | -12.80368875 | 9.666883426 | 0 |
| 3 | -9.587359122 | 8.688022993 | 0 |
| 4 | -28.70922925 | 22.05772899 | 1 |
| 5 | -10.81352565 | 8.932731526 | 0 |
| 6 | -2.312226542 | 1.951992011 | 1 |

**Tabel 3.1** Dataset untuk analisis kasus.

### 3.8.1 Pembentukan *Tree* Pertama

Pada contoh kasus, data dengan index 0, 1, 2, dan 3 akan digunakan sebagai data latih, sedangkan data dengan index 4, 5 dan 6 akan digunakan sebagai data yang akan diuji. Langkah pertama adalah membuat prediksi inisial. XGboost menggunakan nilai 0.5 sebagai prediksi awal [19] sebagai inisialisasi nilai $f_0(x)$ algoritme XGBoost pada gambar 2.6. Kemudian konversi nilai prediksi awal ke dalam *probability* menggunakan fungsi sigmoid sesuai persamaan 2.18.

| Index | v1 | v2 | Class | Initial Prediction | Probability |
|---|---|---|---|---|---|
| 0 | -30.55238004 | 16.71338924 | 1 | 0.5 | 0.62 |
| 1 | -25.82598215 | 19.16723901 | 1 | 0.5 | 0.62 |
| 2 | -12.80368875 | 9.666883426 | 0 | 0.5 | 0.62 |
| 3 | -9.587359122 | 8.688022993 | 0 | 0.5 | 0.62 |

**Tabel 3.2** Inisialisasi nilai prediksi dan *probability* untuk setiap data latih

Perhitungan nilai *probability* untuk setiap data latih.

$probability = \frac{1}{1+e^{-0.5}} = 0.62$

Langkah selanjutnya adalah menghitung nilai *gradient* dan *hessian* pada setiap data latih. Nilai *gradient* dapat dihitung menggunakan persamaan 2.8, sedangkan nilai *hessian* dapat dihitung menggunakan persamaan 2.9.

| Index | v1 | v2 | Class | Initial Prediction | Probability | Gradient | Hessian |
|---|---|---|---|---|---|---|---|
| 0 | -30.55238004 | 16.71338924 | 1 | 0.5 | 0.62 | -0.38 | 0.24 |
| 1 | -25.82598215 | 19.16723901 | 1 | 0.5 | 0.62 | -0.38 | 0.24 |
| 2 | -12.80368875 | 9.666883426 | 0 | 0.5 | 0.62 | 0.62 | 0.24 |
| 3 | -9.587359122 | 8.688022993 | 0 | 0.5 | 0.62 | 0.62 | 0.24 |

**Tabel 3.3** Menghitung nilai *gradient* dan *hessian* untuk setiap data latih.

$$Gradient_0 = 0.62 - 1 = -0.38$$
$$Gradient_1 = 0.62 - 1 = -0.38$$
$$Gradient_2 = 0.62 - 0 = 0.62$$
$$Gradient_3 = 0.62 - 0 = 0.62$$

$$Hessian_0 = 0.62(1 - 0.62) = 0.24$$
$$Hessian_1 = 0.62(1 - 0.62) = 0.24$$
$$Hessian_2 = 0.62(1 - 0.62) = 0.24$$
$$Hessian_3 = 0.62(1 - 0.62) = 0.24$$

Setelah menghitung *gradient* dan *hessian* pada masing-masing data latih, selanjutnya algoritme akan membentuk sejumlah *tree* yang digunakan untuk melakukan prediksi. Untuk membentuk *tree* pertama, algoritme menentukan *split* terbaik fitur v1 dan v2. Sebelum menenentukan split, algoritme mengurutkan nilai pada masing-masing fitur untuk mempermudah perhitungan. Algoritme menentukan *split* terbaik berdasarkan nilai *gain* terbesar.

Nilai *gain* dihitung berdasarkan masing-masing fitur. Untuk menghitung nilai *gain*, diperlukan juga nilai *gradient left*, *gradient right*, *hessian left*, *hessian right* pada masing-masing fitur yang telah diurutkan sesuai persamaan 2.11. *Gradient left* menghitung nilai *gradient* pada data index tertentu terhadap data lain yang nilai fiturnya lebih kecil sama dengan dengan dari data tersebut. Sedangkan *gradient right* menghitung nilai *gradient* data index tertentu dengan data lain yang nilai fiturnya lebih besar dari data tersebut. *Hessian left* menghitung nilai *hessian* pada data index tertentu terhadap data lain yang nilai fiturnya lebih kecil sama dengan dengan dari data tersebut. Sedangkan *Hessian right* menghitung nilai *hessian* data index tertentu dengan data lain yang nilai fiturnya lebih besar dari data tersebut.

Sebagai contoh, nilai fitur v1 pada data latih index 2 adalah -12.803688754721. Untuk menghitung nilai *gl, gr, hl, hr*, dan *gain* fitur v1, perlu dicari data lain yang nilai fitur v1 $<=$ -12.803688754721. Dalam hal ini, data latih index ke 0 dan 1 memenuhi kriteria tersebut dengan nilai fitur v1 sebesar -30.552380043581 dan -25.825982149082. Kemudian lakukan penjumlahan *gradient* untuk mendapatkan *gradient left* fitur v1 pada data latih index ke 2.

Untuk menghitung *gradient right*, maka gunakan kriteria lebih besar daripada nilai fitur, sehingga untuk menghitung *gradient right* pada fitur v1 untuk data indeks ke 2, perlu dicari data lain yang nilai fitur v1 $>$ 9.66688342609514. Terdapat data indeks ke 0 dan ke 1 yang memenuhi kriteria ini dengan nilai fitur v2 sebesar 16.7133892350242 dan 19.1672390103062, lalu jumlahkan *gradient*-nya. Lakukan kriteria yang sama untuk mencari *hessian left* dan *hessian right*, namun dengan menjumlahkan nilai *hessian*-nya. Gambar 3.4 menunjukkan data pada fitur v1 yang telah diurutkan untuk mempermudah perhitungan. Warna merah menunjukkan data yang akan digunakan untuk perhitungan *gradient left* dan *hessian left* fitur v1 untuk data indeks ke 2 dengan kriteria $<=$ -12.803688754721. Warna biru menunjukkan data yang akan digunakan untuk perhitungan *gradient right* dan *hessian right* fitur v1 untuk data indeks ke 2 dengan kriteria $>$ -12.803688754721.

| Index | v1 | v2 | Class | Initial Prediction | Probability | Gradient | Hessian |
|---|---|---|---|---|---|---|---|
| 0 | -30.55238004 | 16.71338924 | 1 | 0.5 | 0.62 | -0.38 | 0.24 |
| 1 | -25.82598215 | 19.16723901 | 1 | 0.5 | 0.62 | -0.38 | 0.24 |
| 2 | -12.80368875 | 9.666883426 | 0 | 0.5 | 0.62 | 0.62 | 0.24 |
| 3 | -9.587359122 | 8.688022993 | 0 | 0.5 | 0.62 | 0.62 | 0.24 |

**Tabel 3.4** Data pada fitur v1 yang telah diurutkan.

Perhitungan *gradient left, gradient right, hessian left* dan *hessian right* fitur v1 pada data latih index 2 pada *tree* pertama.

$gl_2 = -0.38 + -0.38 + 0.62 = -0.14$
$gr_2 = 0.62$
$hl_2 = 0.24 + 0.24 + 0.24 = 0.72$
$hr_2 = 0.24$

Kemudian menghitung nilai *gradient left (gl), gradient right (gr), hessian left (hl)* dan *hessian right (hr)* fitur v1 untuk data latih sisanya, yaitu data indeks 0, 1, dan 3.

Perhitungan *gl, gr, hl* dan *hr* fitur v1 untuk data indeks 0, 1, dan 3 pada *tree* pertama.

$gl_0 = -0.38$
$gr_0 = -0.38 + 0.62 + 0.62 = 0.86$

$\mathrm{gl}_1 = -0.38 - 0.38 = -0.76$
$gr_1 = 0.62 + 0.62 = 1.24$

$\mathrm{gl}_3 = -0.38 - 0.3.8 + 0.62 + 0.62 = 0.48$
$gr_3 = 0$

$\mathrm{hl}_0 = 0.24$
$hr_0 = 0.24 + 0.24 + 0.24 = 0.72$

$\mathrm{hl}_1 = 0.24 + 0.24 = 0.48$
$hr_1 = 0.24 + 0.24 = 0.48$

$\mathrm{hl}_3 = 0.24 + 0.24 + 0.24 + 0.24 = 0.96$
$hr_3 = 0$

Selanjutnya, menghitung gain fitur v1 untuk setiap data latih sesuai persamaan 2.11 dengan nilai $\lambda$ sebesar 1.

Perhitungan *gain* sesuai persamaan 2.11 untuk fitur v1 pada setiap data latih pada *tree* pertama.

$$gain_0 = \frac{1}{2}[\frac{-0.38^2}{0.24+1} + \frac{0.86^2}{0.72+1} - \frac{(-0.38+0.86)^2}{0.24+0.72+1}] - 0 = 0.21$$

$$gain_1 = \frac{1}{2}[\frac{-0.76^2}{0.48+1} + \frac{1.24^2}{0.48+1} - \frac{(-0.76+1.24)^2}{0.48+0.48+1}] - 0 = 0.66$$

$$gain_2 = \frac{1}{2}[\frac{-0.14^2}{0.72+1} + \frac{0.62^2}{0.24+1} - \frac{(-0.14+0.62)^2}{0.72+0.24+1}] - 0 = 0.10$$

$$gain_3 = \frac{1}{2}[\frac{0.48^2}{0.96+1} + \frac{0^2}{0+1} - \frac{(0.48+0)^2}{0.96+0+1}] - 0 = 0$$

| Index | v1 | gl | hl | gr | hr | gain |
|---|---|---|---|---|---|---|
| 0 | -30.55238004 | -0.38 | 0.24 | 0.86 | 0.72 | 0.21 |
| 1 | -25.82598215 | -0.76 | 0.48 | 1.24 | 0.48 | 0.66 |
| 2 | -12.80368875 | -0.14 | 0.72 | 0.62 | 0.24 | 0.10 |
| 3 | -9.587359122 | 0.48 | 0.96 | 0.00 | 0.00 | 0.00 |

**Tabel 3.5** Hasil perhitungan *gradient*, *hessian*, dan *gain* pada *tree* pertama fitur v1.

Tabel 3.5 menunjukkan hasil perhitungan *gradient left* (gl), *gradient right* (gr), *hessian left* (hl), *hessian right* (hr), *gain* pada fitur v1 yang telah diurutkan untuk setiap data latih pada untuk pembentukan *tree* pertama.

Lakukan perhitungan *gradient left, gradient right, hessian left, hessian right* dan nilai *gain* untuk fitur v2 pada setiap data latih.

Perhitungan *gl, gr, hl, hr* fitur v2 pada *tree* pertama.

$gl_0 = 0.62 + 0.62 - 0.38 = 0.86$
$gr_0 = -0.38$

$gl_1 = 0.62 + 0.62 - 0.38 - 0.38 = 0.48$
$gr_1 = 0$

$gl_2 = 0.62 + 0.62 = 1.24$
$gr_2 = -0.38 - 0.38 = -0.76$

$gl_3 = 0.62$
$gr_3 = 0.62 - 0.38 - 0.38 = -0.14$

$hl_0 = 0.24 + 0.24 + 0.24 = 0.72$
$hr_0 = 0.24$

$hl_1 = 0.24 + 0.24 + 0.24 + 0.24 = 0.96$
$hr_1 = 0$

$hl_2 = 0.24 + 0.24 = 0.48$
$hr_2 = 0.24 + 0.24 = 0.48$

$hl_3 = 0.24$
$hr_3 = 0.24 + 0.24 + 0.24 = 0.72$

Perhitungan *gain* untuk fitur v2 pada setiap data latih pada *tree* pertama.

$$gain_0 = \frac{1}{2}[\frac{0.86^2}{0.72+1} + \frac{-0.38^2}{0.24+1} - \frac{(0.86+-0.38)^2}{0.72+0.24+1}] - 0 = 0.21$$

$$gain_1 = \frac{1}{2}[\frac{0.48^2}{0.96+1} + \frac{0^2}{0+1} - \frac{(0.48+0)^2}{0.96+0+1}] - 0 = 0$$

$$gain_2 = \frac{1}{2}[\frac{1.24^2}{0.48+1} + \frac{-0.76^2}{0.48+1} - \frac{(1.24+-0.76)^2}{0.48+0.48+1}] - 0 = 0.66$$

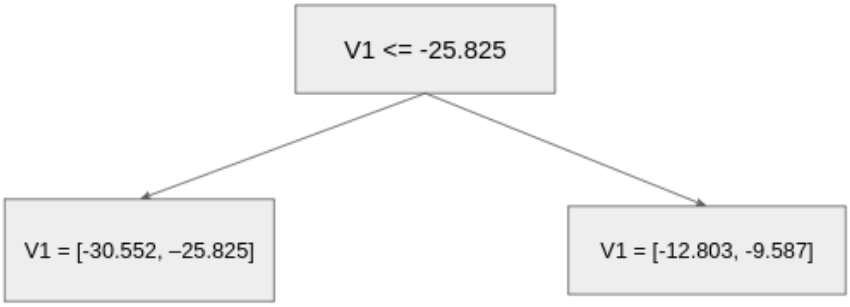
$$gain_3 = \frac{1}{2}[\frac{0.62^2}{0.24+1} + \frac{-0.14^2}{0.72+1} - \frac{(0.62+-0.14)^2}{0.24+0.72+1}] - 0 = 0.10$$

| Index | v2 | gl | hl | gr | hr | gain |
|---|---|---|---|---|---|---|
| 3 | 8.688022993 | 0.62 | 0.24 | -0.14 | 0.72 | 0.10 |
| 2 | 9.666883426 | 1.24 | 0.48 | -0.76 | 0.48 | 0.66 |
| 0 | 16.71338924 | 0.86 | 0.72 | -0.38 | 0.24 | 0.21 |
| 1 | 19.16723901 | 0.48 | 0.96 | 0.00 | 0.00 | 0.00 |

**Tabel 3.6** Hasil perhitungan *gradient*, *hessian*,dan *gain tree* pertama fitur v2.

Tabel 3.6 menunjukkan hasil perhitungan *gradient left* (gl), *gradient right* (gr), *hessian left* (hl), *hessian right* (hr), *gain* pada fitur v2 yang telah diurutkan untuk setiap data latih untuk pembentukan *tree* pertama.

Pada fitur v1 diperoleh nilai *gain* terbesar sebesar 0.66 pada kondisi ketika v1 <= -25.82598215, sedangkan pada fitur v2 diperoleh nilai *gain* tertinggi sebesar 0.66 pada kondisi ketika v2 <= 9.666883426. Oleh karena fitur v1 merupakan fitur pertama pada dataset, maka fitur v1 akan menjadi *split* pertama. Gambar 3.7 merupakan visualisasi *tree* pertama setelah melakukan *split* berdasarkan fitur v1 dengan kondisi v1 <= -25.82598215.




**Gambar 3.7** Visualisasi *tree* pertama setelah melakukan *split*.

| v1 | v2 | Class | Initial Prediction | Probability | gradient | hessian |
|---|---|---|---|---|---|---|
| -30.55238004 | 16.71338924 | 1 | 0.5 | 0.62 | -0.38 | 0.24 |
| -25.82598215 | 19.16723901 | 1 | 0.5 | 0.62 | -0.38 | 0.24 |

**Tabel 3.7** Tabel pada *leaf node* kiri berdasarkan gambar 3.7

| v1 | v2 | Class | Initial Prediction | Probability | gradient | hessian |
|---|---|---|---|---|---|---|
| -12.80368875 | 9.666883426 | 0 | 0.5 | 0.62 | 0.62 | 0.24 |
| -9.587359122 | 8.688022993 | 0 | 0.5 | 0.62 | 0.62 | 0.24 |

**Tabel 3.8** Tabel pada *leaf node* kanan berdasarkan gambar 3.7

Setelah melakukan *split* berdasarkan fitur v1, algoritme harus perlu melakukan *split* kembali hinga mencapai kedalaman tertentu, atau hingga mendapatkan node yang *pure*. Berdasarkan tabel 3.7, *leaf node* kiri telah berisikan semua data latih dengan kelas 1, sedangkan pada tabel 3.8 , *right node* kanan juga telah berisikan semua data latih dengan kelas 0, artinya *leaf node* kiri dan kanan merupakan *leaf node* yang *pure*. *Leaf node* tersebut mampu membedakan setiap nilai pada label kelas (*Class*), sehingga tidak perlu melakukan *split* lagi. Setiap *leaf node* memiliki nilai bobot tersendiri, yang bisa dihitung menggunakan persamaan 2.12.

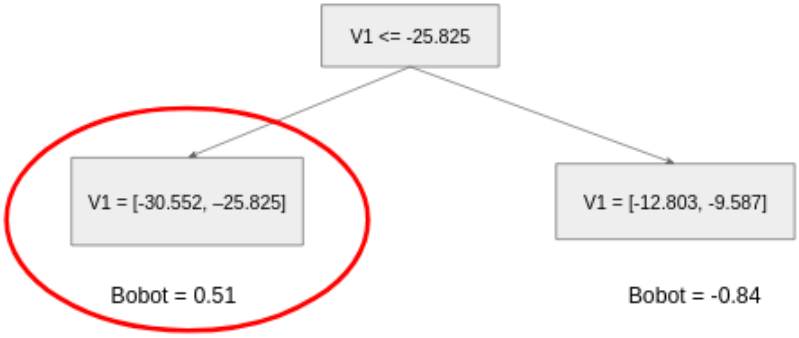
Perhitungan nilai bobot pada *leaf node* dengan fitur v1 <= -25.82598215 sesuai dengan tabel 3.7 pada *tree* pertama.

$G = -0.38 - 0.38 = -0.76$
$H = 0.24 + 0.24 = 0.48$
$\lambda = 1$
$w = -\frac{0.76}{0.48+1} = 0.51$

Perhitungan nilai bobot pada *leaf node* dengan fitur v1 > -25.82598215 sesuai dengan tabel 3.8 pada *tree* pertama.

$G = 0.62 + 0.62 = 1.24$
$H = 0.24 + 0.24 = 0.48$
$\lambda = 1$
$w = -\frac{1.24}{0.48+1} = -0.84$

Tahap selanjutnya adalah membuat prediksi baru menggunakan nilai bobot yang telah dihitung sebelumnya. Misalnya untuk membuat prediksi pada data indeks ke 0 dengan nilai v1 = -30.552380043581, maka perlu dilakukan penelusuran *tree* sesuai gambar 3.7. Terlihat bahwa fitur v1 data indeks ke 0 memiliki nilai yang lebih kecil daripada v1 = -25.82598215, sehingga data indeks ke 0 merupakan anggota dari *leaf node* kiri, sehingga nilai bobotnya adalah 0.51. Gambar 3.8 merupakan visualisasi penelusuran *tree* pertama untuk mendapatkan nilai bobot yang sesuai untuk setiap data latih. Terlihat bahwa data indeks ke 0 masuk ke dalam anggota *leaf node* dengan fitur v1 <= -25.82598215, dengan nilai bobot sebesar 0.51.




**Gambar 3.8** Visualisasi penelusuran *tree* pertama.

Selanjutnya adalah memperbaharui nilai prediksi awal dengan nilai bobot dari *leaf node* yang dikalikan dengan *learning rate* sesuai persamaan 2.14.

Memperbaharui nilai prediksi awal untuk setiap data latih dan mengkonversi menjadi *probability* untuk setiap data latih.

$$NewPrediction_0 = 0.5 + 0.4 \times 0.51 = 0.70$$
$$NewPrediction_1 = 0.5 + 0.4 \times 0.51 = 0.70$$
$$NewPrediction_2 = 0.5 + 0.4 \times -0.84 = 0.16$$
$$NewPrediction_3 = 0.5 + 0.4 \times -0.84 = 0.16$$

$$probability_0 = \frac{1}{1+e^{-0.70}} = 0.67$$

$$probability_1 = \frac{1}{1+e^{-0.70}} = 0.67$$

$$probability_2 = \frac{1}{1+e^{-0.16}} = 0.54$$

$$probability_3 = \frac{1}{1+e^{-0.16}} = 0.54$$

| Indeks | v1 | v2 | Class | Initial Prediction | Probability | Gradient | Hessian | New Prediction | New Probability |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -30.55238004 | 16.71338924 | 1 | 0.5 | 0.62 | -0.38 | 0.24 | 0.70 | 0.67 |
| 1 | -25.82598215 | 19.16723901 | 1 | 0.5 | 0.62 | -0.38 | 0.24 | 0.70 | 0.67 |
| 2 | -12.80368875 | 9.666883426 | 0 | 0.5 | 0.62 | 0.62 | 0.24 | 0.16 | 0.54 |
| 3 | -9.587359122 | 8.688022993 | 0 | 0.5 | 0.62 | 0.62 | 0.24 | 0.16 | 0.54 |

**Tabel 3.9** Hasil prediksi dan *probability* baru untuk setiap data latih .

### 3.8.2 Pembentukan *Tree* Kedua

Hitung nilai *gradient* dan *hessian* untuk *tree* kedua dengan nilai *probability* baru yang telah didapat. Lalu urutkan data berdasarkan setiap fitur untuk menentukan *split* terbaik.

| Indeks | v1 | v2 | Class | New Prediction | Probability | New Gradient | New Hessian |
|---|---|---|---|---|---|---|---|
| 0 | -30.55238004 | 16.71338924 | 1 | 0.70 | 0.67 | -0.33 | 0.22 |
| 1 | -25.82598215 | 19.16723901 | 1 | 0.70 | 0.67 | -0.33 | 0.22 |
| 2 | -12.80368875 | 9.666883426 | 0 | 0.16 | 0.54 | 0.54 | 0.25 |
| 3 | -9.587359122 | 8.688022993 | 0 | 0.16 | 0.54 | 0.54 | 0.25 |

**Tabel 3.10** Nilai *gradient* dan *hessian* untuk *tree* kedua .

$$Gradient_0 = 0.67 - 1 = -0.33$$
$$Gradient_1 = 0.67 - 1 = -0.33$$
$$Gradient_2 = 0.54 - 0 = 0.54$$
$$Gradient_3 = 0.54 - 0 = 0.54$$

$$Hessian_0 = 0.67(1 - 0.67) = 0.22$$
$$Hessian_1 = 0.67(1 - 0.67) = 0.22$$
$$Hessian_2 = 0.54(1 - 0.54) = 0.25$$
$$Hessian_3 = 0.54(1 - 0.54) = 0.25$$

Kemudian menghitung *gradient left*, *gradient right*, *hessian left*, dan *hessian right* untuk tree kedua untuk setiap data latih.

Perhitungan *gl, gr, hl, hr* fitur v1 untuk setiap data latih pada *tree* kedua.

$gl_0 = -0.33$
$gr_0 = -0.33 + 0.54 + 0.54 = 0.75$

$\mathrm{gl}_1 = -0.33 - 0.33 = -0.66$
$gr_1 = 0.54 + 0.54 = 1.08$

$\mathrm{gl}_2 = -0.33 - 0.33 + 0.54 = -0.12$
$gr_2 = 0.54$

$\mathrm{gl}_3 = -0.33 - 0.33 + 0.54 + 0.54 = 0.42$
$gr_3 = 0$

$\mathrm{hl}_0 = 0.22$
$hr_0 = 0.22 + 0.25 + 0.25 = 0.72$

$\mathrm{hl}_1 = 0.22 + 0.22 = 0.44$
$hr_1 = 0.25 + 0.25 = 0.50$

$\mathrm{hl}_2 = 0.22 + 0.22 + 0.25 = 0.69$
$hr_2 = 0.25$

$\mathrm{hl}_3 = 0.22 + 0.22 + 0.25 + 0.25 = 0.94$
$hr_3 = 0$

Perhitungan *gain* fitur v1 untuk setiap data latih pada *tree* kedua.

$$gain_0 = \frac{1}{2}[\frac{-0.33^2}{0.22+1} + \frac{0.75^2}{0.72+1} - \frac{(-0.33+0.75)^2}{0.22+0.72+1}] - 0 = 0.16$$

$$gain_1 = \frac{1}{2}[\frac{-0.66^2}{0.44+1} + \frac{1.08^2}{0.50+1} - \frac{(-0.66+1.24)^2}{0.44+0.50+1}] - 0 = 0.5$$

$$gain_2 = \frac{1}{2}[\frac{-0.12^2}{0.69+1} + \frac{0.54^2}{0.25+1} - \frac{(-0.12+0.54)^2}{0.69+0.25+1}] - 0 = 0.08$$

$$gain_3 = \frac{1}{2}[\frac{0.42^2}{0.94+1} + \frac{0^2}{0+1} - \frac{(0.42+0)^2}{0.94+0+1}] - 0 = 0$$

| Index | v1 | gl | hl | gr | hr | gain |
|---|---|---|---|---|---|---|
| 0 | -30.55238004 | -0.33 | 0.22 | 0.75 | 0.72 | 0.16 |
| 1 | -25.82598215 | -0.66 | 0.44 | 1.08 | 0.50 | 0.50 |
| 2 | -12.80368875 | -0.12 | 0.69 | 0.54 | 0.25 | 0.08 |
| 3 | -9.587359122 | 0.42 | 0.94 | 0.00 | 0.00 | 0 |

**Tabel 3.11** Nilai *gradient*, *hessian*, dan *gain tree* kedua fitur v1.

Tabel 3.11 menunjukkan nilai *gradient left, gradient right, hessian left, hessian right*, *gain* fitur v1 yang telah diurutkan untuk pembentukan *tree* kedua.

Kemudian menghitung nilai *gradient left, gradient right, hessian left, hessian right* dan *gain* fitur v2 untuk *tree* kedua.

Perhitungan *gl, gr, hl, hr* dan *gain* fitur v2 pada *tree* kedua.

$gl_0 = 0.54 + 0.54 - 0.33 = 0.75$
$gr_0 = -0.33$

$gl_1 = 0.54 + 0.54 - 0.33 - 0.33 = 0.42$
$gr_1 = 0$

$gl_2 = 0.54 + 0.54 = 1.08$
$gr_2 = -0.33 - 0.33 = -0.66$

$gl_3 = 0.54$
$gr_3 = 0.54 - 0.33 - 0.33 = -0.12$

$hl_0 = 0.25 + 0.25 + 0.22 = 0.72$
$hr_0 = 0.22$

$hl_1 = 0.25 + 0.25 + 0.22 + 0.22 = 0.94$
$hr_1 = 0$

$hl_2 = 0.25 + 0.25 = 0.50$
$hr_2 = 0.22 + 0.22 = 0.44$

$hl_3 = 0.25$
$hr_3 = 0.25 + 0.22 + 0.22 = 0.69$

Perhitungan *gain* fitur v2 pada *tree* kedua.

$$gain_0 = \frac{1}{2}[\frac{0.75^2}{0.72+1} + \frac{-0.33^2}{0.22+1} - \frac{(0.75+-0.33)^2}{0.72+0.22+1}] - 0 = 0.16$$

$$gain_1 = \frac{1}{2}[\frac{0.42^2}{0.94+1} + \frac{0^2}{0+1} - \frac{(0.42+0)^2}{0.94+0+1}] - 0 = 0$$

$$gain_2 = \frac{1}{2}[\frac{1.08^2}{0.50+1} + \frac{-0.66^2}{0.44+1} - \frac{(1.08+-0.66)^2}{0.50+0.44+1}] - 0 = 0.50$$

$$gain_3 = \frac{1}{2}[\frac{0.54^2}{0.25+1} + \frac{-0.12^2}{0.69+1} - \frac{(0.54+-0.12)^2}{0.25+0.69+1}] - 0 = 0.08$$

| Index | v2 | gl | hl | gr | hr | gain |
|---|---|---|---|---|---|---|
| 3 | 8.688022993 | 0.54 | 0.25 | -0.12 | 0.69 | 0.08 |
| 2 | 9.666883426 | 1.08 | 0.50 | -0.66 | 0.44 | 0.50 |
| 0 | 16.71338924 | 0.75 | 0.72 | -0.33 | 0.22 | 0.16 |
| 1 | 19.16723901 | 0.42 | 0.94 | 0.00 | 0.00 | 0.00 |

**Tabel 3.12** Hasil perhitungan *gradient*, *hessian*, dan *gain tree* kedua fitur v2.

Tabel 3.12 menunjukkan hasil perhitungan *gradient left* (gl), *gradient right* (gr), *hessian left* (hl), *hessian right* (hr), dan *gain* pada fitur v2 yang telah diurutkan untuk setiap data latih untuk pembentukan *tree* kedua.

Berdasarkan hasil perhitungan nilai *gain* pada fitur v1 dan v2 untuk pembentukan tree kedua, fitur v1 memiliki nilai *gain* tertinggi sebesar 0.50 ketika nilai v1 <= -25.825982149082 dan fitur v2 memiliki nilai *gain* tertinggi sebesar 0.50 ketika fitur v2 memiliki nilai <= 9.66688342609514. Karena fitur v1 merupakan fitur paling pertama pada dataset, maka fitur v1 <= -25.825982149082 akan digunakan sebagai *split* untuk membuat *tree* kedua. Gambar 3.9 merupakan visualisasi *tree* kedua setelah melakukan split berdasarkan fitur v1 dengan kondisi v1 <= -25.82598215.


**Gambar 3.9** Visualisasi *tree* kedua setelah melakukan split.

Selanjutnya adalah menghitung nilai bobot untuk setiap *leaf node* sesuai persamaan 2.12.

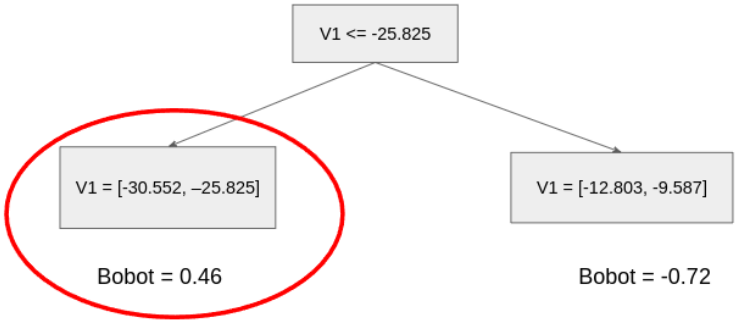
Perhitungan nilai bobot pada *leaf node* dengan fitur v1 <= -25.82598215 sesuai dengan tabel 3.8 pada *tree* kedua.

$G = -0.33 - 0.33 = -0.66$
$H = 0.22 + 0.22 = 0.44$
$\lambda = 1$
$w = -\frac{-0.66}{0.44+1} = 0.46$

Perhitungan nilai bobot pada *leaf node* dengan fitur v1 > -25.82598215 sesuai dengan tabel 3.7 pada *tree* kedua.

$G = 0.54 + 0.54 = 1.08$
$H = 0.25 + 0.25 = 0.50$
$\lambda = 1$
$w = -\frac{1.08}{0.50+1} = -0.72$

Langkah selanjutnya adalah memperbaharui nilai prediksi untuk setiap data latih berdasarkan *tree* yang telah dibuat sebelumnya. Sebagai contoh, untuk mendapatkan prediksi terbaru data indeks ke 0, dapat dilakukan dengan melakukan penelusuran *tree* kedua untuk mendapatkan nilai bobot.




**Gambar 3.10** Visualisasi penelusuran *tree* kedua.

Gambar 3.10 menunjukkan bahwa nilai bobot untuk data indeks ke 0 adalah 0.46 pada *tree* kedua.

Selanjutnya adalah memperbaharui nilai prediksi dengan nilai bobot dari *leaf node* yang dikalikan dengan *learning rate* sesuai persamaan 2.14.

Memperbaharui nilai prediksi sebelumnya untuk setiap data latih dan mengkonversi menjadi *probability* untuk setiap data latih sesuai persamaan.

$$NewestPrediction_0 = 0.71 + 0.4 \times 0.46 = 0.89$$
$$NewestPrediction_1 = 0.71 + 0.4 \times 0.46 = 0.89$$
$$NewestPrediction_2 = 0.71 + 0.4 \times -0.72 = -0.12$$
$$NewestPrediction_3 = 0.71 + 0.4 \times -0.72 = -0.12$$

$$Newest\,probability_0 = \frac{1}{1+e^{-0.89}} = 0.71$$

$$Newest\,probability_1 = \frac{1}{1+e^{-0.89}} = 0.71$$

$$Newest\,probability_2 = \frac{1}{1+e^{-(-0.12)}} = 0.47$$

$$Newest\,probability_3 = \frac{1}{1+e^{-(-0.12)}} = 0.47$$

### 3.8.3 Melakukan prediksi

Data yang akan digunakan untuk melakukan prediksi adalah data indeks ke 4, 5, dan 6 berdasarkan tabel 3.1. Prediksi dapat dihitung menggunakan persamaan 2.16.

| Indeks | v1 | v2 | Class | Initial Prediction | New prediction | Newest Prediction | Final Probability |
|---|---|---|---|---|---|---|---|
| 1 | -30.55238004 | 16.71338924 | 1 | 0.5 | 0.71 | 0.89 | 0.71 |
| 2 | -25.82598215 | 19.16723901 | 1 | 0.5 | 0.71 | 0.89 | 0.71 |
| 3 | -12.80368875 | 9.666883426 | 0 | 0.5 | 0.16 | -0.12 | 0.47 |
| 4 | -9.587359122 | 8.688022993 | 0 | 0.5 | 0.16 | -0.12 | 0.47 |

**Tabel 3.13** Hasil prediksi akhir

Menghitung prediksi dan *probability* untuk data indeks ke 4, 5, dan 6.

$Prediction_4 = 0.5 + 0.4 \times 0.51 + 0.4 \times 0.46 = 0.89$

$Probability_4 = \frac{1}{1+e^{-0.89}} = 0.71$

$Prediction_5 = 0.5 + 0.4 \times -0.84 + 0.4 \times -0.72 = -0.124$

$Probability_5 = \frac{1}{1+e^{-(-0.124)}} = 0.47$

$Prediction_6 = 0.5 + 0.4 \times -0.84 + 0.4 \times -0.72 = -0.124$

$Probability_6 = \frac{1}{1+e^{-(-0.124)}} = 0.47$

Karena hasil *probability* lebih besar dari 0.5, maka data indeks ke 4 diprediksi merupakan kelas 1 dan memang kelas sesungguhnya adalah kelas 1 (*true positive*). Data indeks ke 5 diprediksi sebagai kelas 0 dan memang kelas sesungguhnya adalah kelas 0 (*true negative*). Data indeks ke 6 diprediksi sebagai kelas 0, tetapi kelas sesungguhnya adalah kelas 1 (*false negative*). Berikut merupakan perhitungan nilai akurasi, *recall* dan *f1-score* atau *f-measure*.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$
$$Accuracy = \frac{1+1}{1+1+0+1}$$
$$Accuracy = \frac{2}{3} \times 100\%$$
$$Accuracy = 67\%$$

$$Recall = \frac{TP}{TP+FN}$$
$$Recall = \frac{1}{1+1}$$
$$Recall = \frac{1}{2} \times 100\%$$
$$Recall = 50\%$$

$$F-measure = \frac{2\times TP}{2\times TP+FP+FN}$$
$$F-measure = \frac{2\times 1}{2\times 1+0+1}$$
$$F-measure = \frac{2}{3}$$
$$F-measure = \frac{2}{3} \times 100\%$$
$$F-measure = 67\%$$

# BAB 4 IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan menjelaskan mengenai proses implementasi dan pengujian terhadap sistem yang telah dibangun berdasarkan penjelasan pada bab sebelumnya.

## 4.1 Lingkungan Implementasi

Pada lingkungan implementasi, akan dijelaskan mengenai perangkat-perangkat yang digunakan selama proses pembangunan sistem baik dari perangkat keras maupun perangkat lunak yang digunakan.

### 4.1.1 Spesifikasi Perangkat Keras

Spesifikasi dari perangkat keras yang digunakan dalam pembangunan sistem deteksi transaksi penipuan kartu kredit adalah:

1. *Processor* Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz.
2. Solid State Drive (SSD) kapasitas 250GB.
3. Hard Disk (HDD) kapasitas 1TB.
4. RAM dengan kapasitas 16GB.
5. VGA NVIDIA GeForce MX130.

### 4.1.2 Spesifikasi Perangkat Lunak

Spesifikasi dari perangkat lunak yang digunakan dalam pembangunan sistem deteksi transaksi penipuan kartu kredit adalah:

1. Sistem Operasi Pop! OS 20.04 LTS.
2. IDE: Google Colaboratory Pro.
3. Python 3.7.13.

## 4.2 Implementasi Perangkat Lunak

Pada bab ini akan dijelaskan mengenai *class* dan *method* yang digunakan pada sistem klasifikasi transaksi penipuan kartu kredit.

### 4.2.1 *Class* Preprocessing

*Class* Preprocessing merupakan kelas yang digunakan untuk melakukan *preprocessing* pada *dataset*.

**Tabel 4.1** Daftar Metode *Class* Preprocessing

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | __init__ | df: dataframe | - | Fungsi yang digunakan untuk melakukan inisialisasi untuk menyimpan *dataframe* yang akan dilakukan *preprocessing*. |
| 2 | print_duplicate | - | - | Fungsi untuk melakukan terhadap data duplikat. |
| 3 | drop_duplicate | - | - | Fungsi untuk menghapus data duplikat. |
| 4 | get_is_na | - | - | Fungsi untuk mendapatkan data dengan *missing value*. |
| 5 | plot_corr | method: string | - | Fungsi untuk melakukan plot *correlation heatmap*. |
| 6 | similarity_score | mask: array | int | Menghitung pembagian antara nilai radient dengan nilai hessian dan lambda. |
| 7 | split$_data$ | X: array,y: array,test$_size$ : $float$ random$_state$ : $int$ | array, array, array, array | Fungsi untuk melakukan memisahkan *dataset* menjadi data latih dan data uji. |

### 4.2.2 *Class* Node

*Class* Node merupakan kelas yang digunakan untuk membuat *node* untuk dalam *tree* yang akan dibuat.

**Tabel 4.2** Daftar Metode *Class Node*

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | __init__ | x: array, y: array, grad: float, hess: float, depth: int, gamma: float, min_child_weight: float, colsample: int | - | Fungsi yang digunakan untuk melakukan inisialisasi pembuatan *node*. |
| 2 | split_node | - | - | Fungsi untuk melakukan *splitting*. |
| 3 | find_split | - | - | Fungsi untuk menemukan split terbaik. |
| 4 | not_valid_split | mask: array | boolean | Fungsi untuk menentukan validitas dari sebuah split. |
| 5 | is_leaf | - | - | Fungsi untuk menentukan apakah *node* merupakan *leaf node*. |
| 6 | similarity_score | mask: array | int | Menghitung pembagian antara nilai radient dengan nilai hessian dan lambda. |
| 7 | predict | - | array | Fungsi yang akan memanggil fungsi predict_single_val. |
| 8 | predict_single_val | - | float | Fungsi rekursif yang akan melakukan penelusuran *tree* untuk mendapatkan bobot *leaf node*. |

### 4.2.3 *Class* XGBTree

*Class* XGBTree merupakan kelas yang digunakan untuk membuat *tree* dalam algoritme XGBoost.

**Tabel 4.3** Daftar Metode *Class* XGBTree

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | __init__ | x: array, y: array, grad: float, hess: float, depth: int, gamma: float, min_child_weight: float, colsample: int | - | Fungsi untuk melakukan inisialisasi *node decision tree*. |
| 2 | predict | x: array | array | Fungsi pembungkus yang memanggil fungsi *predict* yang terdapat dalam *class node* melalui *root node*. |

### 4.2.4 *Class* XGBClassifierBase

*Class* XGBClassifierBase merupakan kelas yang digunakan untuk melakukan pelatihan algoritme klasifikasi XGBoost.

**Tabel 4.4** Daftar Metode *Class* XGBClassifierBase

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | __init__ | eta: float, n_estimators: int, max_depth: int, gamma : float, min_child_weight: float, lambda: float, subsample:int | - | Fungsi untuk melakukan inisialisasi setiap variabel masukkan dan inisialisasi *list* untuk menyimpan setiap*tree* yang akan dibangun. |
| 2 | fit | x: array, y: array | - | Fungsi yang membentuk sejumlah *tree* dan melatih setiap *tree* yang dibangun. |
| 3 | predict | x: array, prob: boolean | array | Melakukan prediksi dan mengkonversi ke dalam probabilitas. |
| 4 | sigmoid | x: array | float | Fungsi untuk menghitung nilai *sigmoid*. |

**Tabel 4.4** – Daftar Metode *Class* XGBClassifierBase (lanjutan)

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 5 | grad | y: array, a: float | float | Fungsi untuk menghitung nilai *gradient*. |
| 6 | hess | y: array, a: float | array | Fungsi untuk menghitung nilai *hessian*. |

### 4.2.5 *Class* XGBClassifier

*Class* XGBClassifier merupakan kelas yang digunakan untuk membangun algoritme XGBoost untuk melakukan klasifikasi.

**Tabel 4.5** Daftar Metode *Class* XGBClassifier

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | __init__ | n_classes: int, eta: float, n_estimators: int, max_depth: int, gamma : float, min_child_weight: float, lambda: float, subsample:int | - | Fungsi untuk melakukan inisialisasi setiap variabel masukkan. |
| 2 | fit | x: array, y: array | - | Fungsi yang memanggil fungsi *fit* pada kelas XGBClassifierBase. |
| 3 | predict | x: array | array | Fungsi pembungkus yang akan memanggil fungsi *predict* pada kelas XGBClassifierBase. |

### 4.2.6 *Class* GridSearch

**Tabel 4.6** Daftar Metode *Class* GridSearch

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | get_hyper_comb | - | DataFrame | Fungsi yang menghasilkan kombinasi antar *hyperparameter*. |

### 4.2.7 CrossValidation

**Tabel 4.7** Daftar Metode *Class* CrossValidation

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | __init__ | fold: int | - | Fungsi untuk inisialisasi nilai *fold*. |
| 2 | get_split_data_idx | df | DataFrame | Fungsi untuk membagi DataFrame menjadi data latih dan data uji untuk setiap *fold*. |
| 3 | save_to_csv | csv_path: string, test_num: int, n_estimators: int, learning_rate: float, gamma: float, lamb: float, fold : int, recall: float, accuracy: float, f1_score: float | - | Fungsi untuk menulis hasil ke dalam file csv. |
| 4. | get_cross_val_score | cv_split: array, xgb: XGBClassifier, test_num: int, csv_file_path: string | - | Fungsi untuk mendapatkan hasil *cross-validation score* dan akan memanggil fungsi save_to_csv. |

### 4.2.8 *Class* Evaluation

**Tabel 4.8** Daftar Metode *Class* Evaluation

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 1 | evaluate | y_true: array, y_pred: array | array | Fungsi pembungkus yang memanggil metode evaluasi yang akan diukur. |
| 2 | get_accuracy | tn: int, fp: int, fn: int, tp: int | float | Fungsi yang menghitung akurasi. |
| 3 | get_recall | tn: int, fn: int | float | Fungsi yang menghitung nilai *recall*. |

**Tabel 4.8** – Daftar Metode *Class Node* (lanjutan)

| No | Metode | Masukan | Luaran | Keterangan |
|---|---|---|---|---|
| 4 | get_f1_score | tn: int, fp: int, fn: int, tp: int | float | Fungsi yang menghitung nilai *f1_score* atau *f_measure*. |

## 4.3 Pengujian

Pada penelitian ini, pengujian yang dilakukan adalah menguji hyperparameter pada algoritme XGBoost untuk mendeteksi transaksi penipuan kartu kredit. Hasil yang diharapkan pada penelitian ini adalah nilai *recall* yang tinggi.

Pada pengujian dalam penelitian ini akan melibatkan beberapa kombinasi *hyperparameter*. Pengujian akan melibatkan proses pelatihan menggunakan metode *grid search* dan *cross-validation* pada data latih. Setiap *fold* dalan *cross-validation* akan mengeluarkan hasil evaluasi. Evaluasi yang akan digunakan untuk mengukur performa model adalah *recall* dan *f1-score*. *Recall* digunakan untuk mengukur tingkat kesalahan dalam mengklasifikasikan transaksi *fraud* dan *non-fraud*. Semakin tinggi persentase nilai *recall* maka kesalahan mengklasifikasikan *fraud* sebagai *non-fraud* akan lebih sedikit jumlahnya, begitu pula sebaliknya. *F1-score* merupakan nilai rata-rata antara *precision* dan *recall*. *Precision* digunakan untuk mengukur tingkat kesalahan dalam mengklasifikasikan *non-fraud* sebagai *fraud*. Hasil pengujian dengan *hyperparameter* yang dinilai terbaik yaitu memiliki nilai *recall* yang tinggi karena pada kasus deteksi transaksi penipuan kartu kredit diperlukan jumlah *false negative* yang sangat rendah. Kombinasi *hyperparameter* yang menghasilkan nilai *recall* tertinggi akan diuji pada data uji.

### 4.3.1 Skenario Pengujian XGBoost

Pengujian dilakukan dengan mengklasifikasikan data uji dari transaksi penipuan kartu kredit yang sudah ditentukan dalam proses *5-fold cross-validation* menggunakan model XGBoost. Hasil dari pengujian adalah hasil rata-rata *accuracy*, *recall*, dan *f1-score* yang dihasilkan dari proses *5-fold cross-validation*.

Dalam pembentukan kombinasi *hyperparameter* XGBoost, terdapat 5 *hyperparameter* yang diuji, yaitu *n_estimators*, *max_depth eta*, *gamma*, dan *lambda*. Total kombinasi yang terbentuk adalah 72 kombinasi kasus pengujian.

| XGBoost | | | | | |
|---|---|---|---|---|---|
| | n_estimators | max_depth | eta (learning rate) | gamma | lambda |
| | 50 | 5 | 0.1 | 0 | 0.5 |
| | 100 | 10 | 0.3 | 0.5 | 1 |
| | 200 | 15 | | | |
| Total | 3 | 3 | 2 | 2 | 2 |
| Total Pengujian | = 3 x 3 x 2 x 2 x 2 | | | | |
| | 72 | | | | |

**Tabel 4.9** Total kombinasi kasus pengujian XGBoost.

Berikut merupakan daftar dari setiap kombinasi pengujian yang akan dilakukan.

**Tabel 4.10** Daftar kombinasi kasus pengujian XGBoost

| **No** | **n_estimators** | **max_depth** | **eta** | **lambda** | **gamma** |
|---|---|---|---|---|---|
| 1 | 50 | 5 | 0.1 | 0 | 0.5 |
| 2 | 50 | 5 | 0.1 | 0 | 1 |
| 3 | 50 | 5 | 0.1 | 0.5 | 0.5 |
| 4 | 50 | 5 | 0.1 | 0.5 | 1 |
| 5 | 50 | 5 | 0.3 | 0 | 0.5 |
| 6 | 50 | 5 | 0.3 | 0 | 1 |
| 7 | 50 | 5 | 0.3 | 0.5 | 0.5 |
| 8 | 50 | 5 | 0.3 | 0.5 | 1 |
| 9 | 50 | 10 | 0.1 | 0 | 0.5 |
| 10 | 50 | 10 | 0.1 | 0 | 1 |
| 11 | 50 | 10 | 0.1 | 0.5 | 0.5 |
| 12 | 50 | 10 | 0.1 | 0.5 | 1 |
| 13 | 50 | 10 | 0.3 | 0 | 0.5 |
| 14 | 50 | 10 | 0.3 | 0 | 1 |
| 15 | 50 | 10 | 0.3 | 0.5 | 0.5 |
| 16 | 50 | 10 | 0.3 | 0.5 | 1 |
| 17 | 50 | 15 | 0.1 | 0 | 0.5 |
| 18 | 50 | 15 | 0.1 | 0 | 1 |

**Tabel 4.10** – Daftar kombinasi kasus pengujian XGBoost (lanjutan)

| No | n_estimators | max_depth | eta | lambda | gamma |
|---|---|---|---|---|---|
| 19 | 50 | 15 | 0.1 | 0.5 | 0.5 |
| 20 | 50 | 15 | 0.1 | 0.5 | 1 |
| 21 | 50 | 15 | 0.3 | 0 | 0.5 |
| 22 | 50 | 15 | 0.3 | 0 | 1 |
| 23 | 50 | 15 | 0.3 | 0.5 | 0.5 |
| 24 | 50 | 15 | 0.3 | 0.5 | 1 |
| 25 | 100 | 5 | 0.1 | 0 | 0.5 |
| 26 | 100 | 5 | 0.1 | 0 | 1 |
| 27 | 100 | 5 | 0.1 | 0.5 | 0.5 |
| 28 | 100 | 5 | 0.1 | 0.5 | 1 |
| 29 | 100 | 5 | 0.3 | 0 | 0.5 |
| 30 | 100 | 5 | 0.3 | 0 | 1 |
| 31 | 100 | 5 | 0.3 | 0.5 | 0.5 |
| 32 | 100 | 5 | 0.3 | 0.5 | 1 |
| 33 | 100 | 10 | 0.1 | 0 | 0.5 |
| 34 | 100 | 10 | 0.1 | 0 | 1 |
| 35 | 100 | 10 | 0.1 | 0.5 | 0.5 |
| 36 | 100 | 10 | 0.1 | 0.5 | 1 |
| 37 | 100 | 10 | 0.3 | 0 | 0.5 |
| 38 | 100 | 10 | 0.3 | 0 | 1 |
| 39 | 100 | 10 | 0.3 | 0.5 | 0.5 |
| 40 | 100 | 10 | 0.3 | 0.5 | 1 |
| 41 | 100 | 15 | 0.1 | 0 | 0.5 |
| 42 | 100 | 15 | 0.1 | 0 | 1 |
| 43 | 100 | 15 | 0.1 | 0.5 | 0.5 |
| 44 | 100 | 15 | 0.1 | 0.5 | 1 |
| 45 | 100 | 15 | 0.3 | 0 | 0.5 |
| 46 | 100 | 15 | 0.3 | 0 | 1 |
| 47 | 100 | 15 | 0.3 | 0.5 | 0.5 |

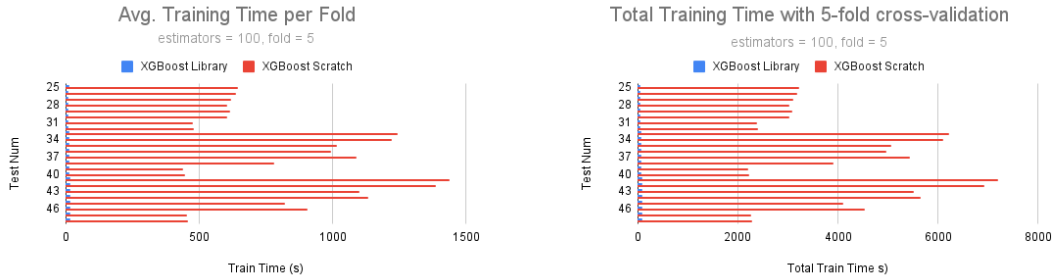
**Tabel 4.10** – Daftar kombinasi kasus pengujian XGBoost (lanjutan)

| No | n_estimators | max_depth | eta | lambda | gamma |
|---|---|---|---|---|---|
| 48 | 100 | 15 | 0.3 | 0.5 | 1 |
| 49 | 200 | 5 | 0.1 | 0 | 0.5 |
| 50 | 200 | 5 | 0.1 | 0 | 1 |
| 51 | 200 | 5 | 0.1 | 0.5 | 0.5 |
| 52 | 200 | 5 | 0.1 | 0.5 | 1 |
| 53 | 200 | 5 | 0.3 | 0 | 0.5 |
| 54 | 200 | 5 | 0.3 | 0 | 1 |
| 55 | 200 | 5 | 0.3 | 0.5 | 0.5 |
| 56 | 200 | 5 | 0.3 | 0.5 | 1 |
| 57 | 200 | 10 | 0.1 | 0 | 0.5 |
| 58 | 200 | 10 | 0.1 | 0 | 1 |
| 59 | 200 | 10 | 0.1 | 0.5 | 0.5 |
| 60 | 200 | 10 | 0.1 | 0.5 | 1 |
| 61 | 200 | 10 | 0.3 | 0 | 0.5 |
| 62 | 200 | 10 | 0.3 | 0 | 1 |
| 63 | 200 | 10 | 0.3 | 0.5 | 0.5 |
| 64 | 200 | 10 | 0.3 | 0.5 | 1 |
| 65 | 200 | 15 | 0.1 | 0 | 0.5 |
| 66 | 200 | 15 | 0.1 | 0 | 1 |
| 67 | 200 | 15 | 0.1 | 0.5 | 0.5 |
| 68 | 200 | 15 | 0.1 | 0.5 | 1 |
| 69 | 200 | 15 | 0.3 | 0 | 0.5 |
| 70 | 200 | 15 | 0.3 | 0 | 1 |
| 71 | 200 | 15 | 0.3 | 0.5 | 0.5 |
| 72 | 200 | 15 | 0.3 | 0.5 | 1 |

### 4.3.2 Pengujian XGBoost yang Dibangun Dari *Scratch*

Pada tahap ini, algoritme XGBoost diimplementasikan dan dibangun dari awal(*scratch*) untuk melakukan pelatihan terhadap kombinasi hyperparameter dengan *estimators* sebesar 100, *max_depth* sebesar 5, 10, dan 15, *eta* sebesar 0.1 dan 0.3, gamma dengan nilai 0 dan 0.5 dan *lambda* dengan nilai 0.5 dan 1. Total

kombinasi yang diuji dengan menggunakan algoritme XGBoost *scratch* adalah sebanyak 24 kombinasi. Proses pelatihan ini memakan waktu yang cukup lama seperti terlihat pada gambar 4.1.




**(a)** Rata-rata waktu pelatihan setiap fold

**(b)** Total waktu pelatihan menggunakan *5-fold cross-validation*

**Gambar 4.1** Perbandingan waktu pelatihan

Gambar 4.1a menunjukkan proses pelatihan XGBoost yang dibangun dari awal. Terlihat bahwa waktu pelatihan minimum hampir menyentuh angka 500 detik, sedangkan waktu pelatihan maksimum yang terjadi hampir menyentuh angka 1500 detik hanya untuk 1 *fold* dari *5-fold cross-validation*. Hal ini tentunya akan sangat lama mengingat fold yang digunakan adalah sebesar 5. Gambar 4.1b menunjukkan waktu pelatihan yang dibutuhkan pada XGBoost *scratch* dengan menggunakan *5-fold cross-validation*. Pada gambar tersebut terlihat bahwa waktu pelatihan minimum yang terjadi sekitar 2000 detik, sedangkan waktu pelatihan maksimum yang terjadi adalah sekitar 7000 detik atau hampir 2 jam untuk 1 kombinasi pengujian *hyperparameter* dari 24 kombinasi pengujian *hyperparameter*.

Waktu yang telah diperhitungkan tersebut tentunya akan terus bertambah seiring bertambahnya jumlah *estimators* atau jumlah *tree* yang akan dibangun dan kedalaman *tree*. Selain itu, tentunya *eta* atau *learning_rate* juga akan mempengaruhi lama waktu pelatihan. Sehingga, lamanya waktu pelatihan menjadi hambatan dalam penelitian ini. Besaran *memory* yang dipakai dalam menjalankan algoritme XGBoost yang diimplementasikan dari awal juga menjadi hambatan, karena ketika menjalankan algoritme XGBoost dengan *estimators* yang lebih besar dari 100, Google Colab Pro yang menjadi IDE dalam penelitian ini mengalami *crash* karena *memory* yang dipakai terlalu besar. Hal ini juga dapat disebabkan karena data latih yang digunakan terlalu besar, selain itu juga terdapat proses *oversampling* SMOTE untuk memperbanyak jumlah kelas minoritas sehingga menambah jumlah data latih dalam proses pelatihan *5-fold cross validation* untuk

setiap fold. Dapat disimpulkan bahwa semakin besar data latih serta nilai *estimators* dan *max_depth* yang digunakan akan meningkatkan jumlah penggunaan memory dan lama waktu pelatihan. Selain itu semakin kecil nilai *eta* yang digunakan juga dapat meningkatkan lama waktu pelatihan.

Setelah itu, algoritme XGBoost yang telah diimplementasikan dalam pustaka *python* [16] digunakan untuk melakukan pengujian kombinasi *hyperparameter*. Gambar 4.1a menunjukkan grafik rata-rata waktu pelatihan XGBoost yang lebih cepat dibandingkan algoritme XGBoost yang diimplementasikan secara *scratch* untuk setiap fold. Gambar 4.1b juga menunjukkan bahwa XGBoost dengan pustaka python jauh cepat untuk total pelatihan yang dilakukan. Hal ini disebabkan karena pustaka XGBoost mampu menggunakan proses pelatihan dengan menggunakan *multi-core*, yaitu memanfaatkan semua *core* untuk melakukan proses pelatihan. Tentunya proses komputasi dengan *multi-core* akan membuat proses pelatihan jauh lebih cepat. Hal lainnya tentunya dikarenakan pustaka XGBoost telah menggunakan beberapa optimisasi untuk menangani jumlah data pelatihan yang sangat besar seperti yang telah dijelaskan pada bab 2, diantaranya adalah *cache-aware access* yang memanfaatkan *cache* dan *blocks for out-of-core computation* yang memanfaatkan teknik *block sharding* dengan menyimpan data ke dalam *disk* dan memanfaatkan *thread* yang dikhususkan untuk mengambil data tersebut. Hal ini tentunya akan membuat data latih tidak perlu disimpan semuanya ke dalam *memory* sehingga mencegah penggunaan memory yang terlalu besar yang mampu menyebabkan IDE Google Colab Pro mengalai *crash*. Beberapa hal tersebut membuat penelitian ini akan dilakukan dengan menggunakan pustaka XGBoost.

### 4.3.3 Pengujian XGBoost menggunakan *5-fold Cross-Validation* dan *Grid Search*

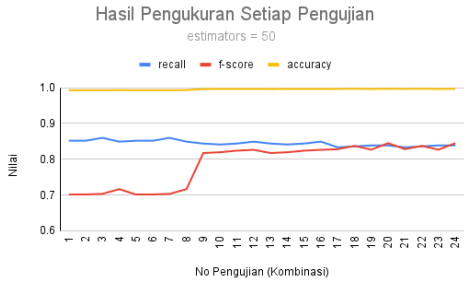
#### 4.3.3.1 Pengujian Estimators 50

Pengujian ini akan menguji model XGBoost dengan *estimators* atau jumlah *tree* yang akan dibentuk sebesar 50. Pengujian ini akan dikombinasikan dengan *max_depth* atau kedalaman tree dengan nilai 5, 10, dan 15, kemudian *eta* dengan nilai 0.1 dan 0.3, lalu *gamma* dengan nilai 0 dan 0.5 dan *lambda* dengan nilai 0.5 dan 1. Hasil pengujian yaitu *recall*, *accuracy* dan *f-score* diambil berdasarkan rata-rata proses *5-fold cross-validation*. Tabel 4.11 menunjukkan hasil pengujian XGBoost dengan *estimators* sebesar 50.

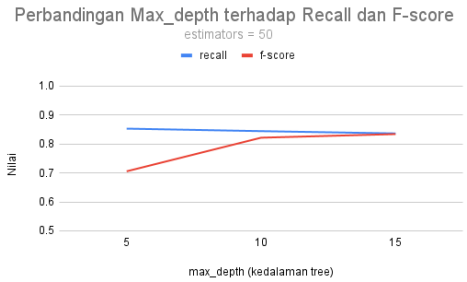
**Tabel 4.11** Hasil Pengujian XGBoost dengan Estimators 50

| max_depth | eta | gamma | lambda | recall | accuracy | f-score |
|---|---|---|---|---|---|---|
| 5 | 0.1 | 0 | 0.5 | 0.8518 | 0.9928 | 0.7015 |
| | | | 1 | 0.8518 | 0.9928 | 0.7017 |
| | | 0.5 | 0.5 | **0.8598** | 0.9928 | 0.7034 |
| | | | 1 | 0.8492 | 0.9933 | 0.7165 |
| | 0.3 | 0 | 0.5 | 0.8518 | 0.9928 | 0.7015 |
| | | | 1 | 0.8518 | 0.9928 | 0.7017 |
| | | 0.5 | 0.5 | **0.8598** | 0.9928 | 0.7034 |
| | | | 1 | 0.8492 | 0.9933 | 0.7165 |
| 10 | 0.1 | 0 | 0.5 | 0.8439 | 0.9963 | 0.8175 |
| | | | 1 | 0.8413 | 0.9963 | 0.8196 |
| | | 0.5 | 0.5 | 0.8439 | 0.9964 | 0.8241 |
| | | | 1 | 0.8492 | 0.9964 | 0.8264 |
| | 0.3 | 0 | 0.5 | 0.8439 | 0.9963 | 0.8175 |
| | | | 1 | 0.8413 | 0.9963 | 0.8196 |
| | | 0.5 | 0.5 | 0.8439 | 0.9964 | 0.8241 |
| | | | 1 | 0.8492 | 0.9964 | 0.8264 |
| 15 | 0.1 | 0 | 0.5 | 0.8333 | 0.9966 | 0.8281 |
| | | | 1 | 0.8360 | 0.9968 | 0.8375 |
| | | 0.5 | 0.5 | 0.8387 | 0.9965 | 0.8269 |
| | | | 1 | 0.8386 | **0.9969** | **0.8450** |
| | 0.3 | 0 | 0.5 | 0.8333 | 0.9966 | 0.8281 |
| | | | 1 | 0.8360 | 0.9968 | 0.8375 |
| | | 0.5 | 0.5 | 0.8387 | 0.9965 | 0.8269 |
| | | | 1 | 0.8386 | **0.9969** | **0.8450** |

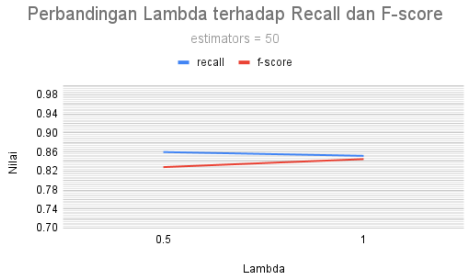
Berdasarkan gambar 4.2a terlihat bahwa grafik akurasi selalu berada diatas 0.9 dan hampir mendekati angka 1.0. Kemudian grafik *recall* cenderung menurun seiring bertambahnya nomor kombinasi pengujian. Grafik *f-score* menunjukkan bahwa terdapat kenaikan drastis antara pengujian kombinasi nomor 8 dan kombinasi nomor 9 sesuai pada tabel 4.10. Selain itu, grafik *f-score* juga cenderung naik seiring bertambahnya nomor kombinasi pengujian. Pada gambar 4.2b, terlihat bahwa nilai *recall* cenderung menurun seiring bertambanya nilai *max_depth*. Grafik *f-score* juga mengalami peningkatan antara nilai *max_depth* 5 dan 10 dan cenderung naik seiring bertambahnya nilai *max_depth*. Pada gambar



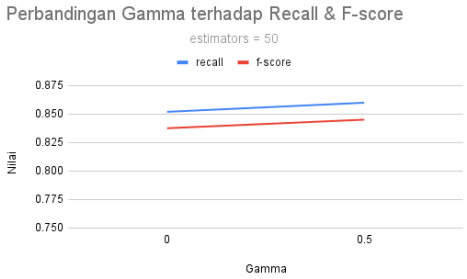

**(a)** Hasil pengukuran setiap kombinasi pengujian pada *estimators* 50




**(b)** Perbandingan *max_depth* terhadap *recall* dan *f-score*




**(c)** Perbandingan *lambda* terhadap *recall* dan *F-score*

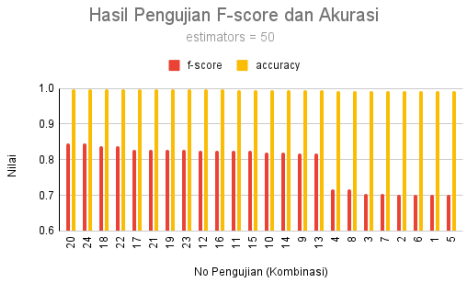



**(d)** Perbandingan *gamma* terhadap *recall* dan *f-score*

**Gambar 4.2** Hasil Pengujian pada Estimator 50

4.2c terlihat bahwa nilai *recall* cenderung menurun apabila nilai *lambda* semakin bertambah, sedangkan grafik *f-score* cenderung naik. Pada gambar 4.2d terlihat bahwa grafik *recall* dan *f-score* keduanya cenderung naik apabila nilai *gamma* semakin bertambah.


**(a)** Hasil pengujian dengan menggunakan metrik pengukuran *recall*




**(b)** Hasil pengujian dengan menggunakan metrik pengukuran *f-score* dan akurasi

**Gambar 4.3** Hasil pengujian yang telah diurutkan dari nilai tertinggi ke nilai terendah pada *estimators* 50

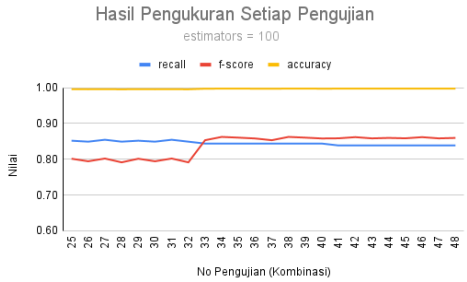
Gambar 4.3a menunjukkan bahwa nilai *recall* tertinggi diperoleh oleh kombinasi nomor 3 dan 7, yaitu dengan nilai sebesar 85,98%. Gambar 4.3b menunjukkan bahwa nilai *f-score* dan akurasi tertinggi diperoleh oleh kombinasi nomor 20 dan 24 dengan nilai *f-score* tertinggi sebesar 84,50% dan nilai akurasi tertinggi sebesar 99.69%. Nomor kombinasi dapat dilihat berdasarkan tabel 4.10.

#### 4.3.3.2 Pengujian Estimators 100

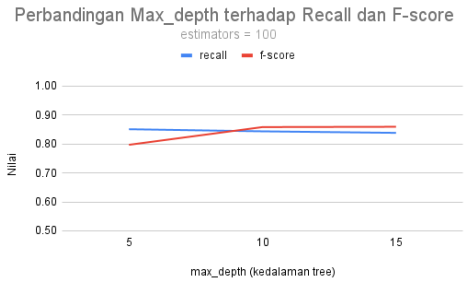
Pengujian ini akan menguji model XGBoost dengan *estimators* atau jumlah *tree* yang akan dibentuk sebesar 100. Pengujian ini akan dikombinasikan dengan *max_depth* atau kedalaman tree dengan nilai 5, 10, dan 15, kemudian *eta* dengan nilai 0.1 dan 0.3, lalu *gamma* dengan nilai 0 dan 0.5 dan *lambda* dengan nilai 0.5 dan 1. Hasil pengujian yaitu *recall*, *accuracy* dan *f-score* diambil berdasarkan rata-rata proses *5-fold cross-validation*. Tabel 4.12 menunjukkan hasil pengujian XGBoost dengan *estimators* sebesar 100.

**Tabel 4.12** Hasil Pengujian XGBoost dengan Estimators 100

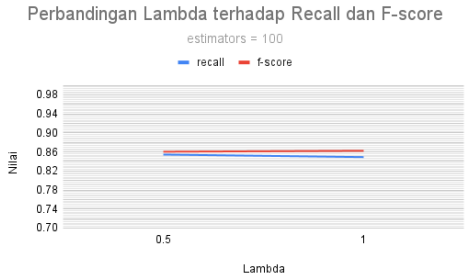
| max_depth | eta | gamma | lambda | recall | accuracy | f-score |
|---|---|---|---|---|---|---|
| 5 | 0.1 | 0 | 0.5 | 0.8519 | 0.9958 | 0.8015 |
| | | | 1 | 0.8492 | 0.9956 | 0.7943 |
| | | 0.5 | 0.5 | **0.8545** | 0.9958 | 0.8021 |
| | | | 1 | 0.8492 | 0.9956 | 0.7914 |
| | 0.3 | 0 | 0.5 | 0.8519 | 0.9958 | 0.8015 |
| | | | 1 | 0.8492 | 0.9956 | 0.7943 |
| | | 0.5 | 0.5 | **0.8545** | 0.9958 | 0.8021 |
| | | | 1 | 0.8492 | 0.9956 | 0.7914 |
| 10 | 0.1 | 0 | 0.5 | 0.8439 | 0.9971 | 0.8533 |
| | | | 1 | 0.8439 | **0.9973** | **0.8625** |
| | | 0.5 | 0.5 | 0.8439 | **0.9973** | 0.8604 |
| | | | 1 | 0.8439 | 0.9972 | 0.8581 |
| | 0.3 | 0 | 0.5 | 0.8439 | 0.9971 | 0.8533 |
| | | | 1 | 0.8439 | **0.9973** | **0.8625** |
| | | 0.5 | 0.5 | 0.8439 | **0.9973** | 0.8604 |
| | | | 1 | 0.8439 | 0.9972 | 0.8581 |
| 15 | 0.1 | 0 | 0.5 | 0.8386 | **0.9973** | 0.8586 |
| | | | 1 | 0.8386 | **0.9973** | 0.8618 |
| | | 0.5 | 0.5 | 0.8386 | **0.9973** | 0.8584 |
| | | | 1 | 0.8386 | **0.9973** | 0.8597 |
| | 0.3 | 0 | 0.5 | 0.8386 | **0.9973** | 0.8586 |
| | | | 1 | 0.8386 | **0.9973** | 0.8618 |
| | | 0.5 | 0.5 | 0.8386 | **0.9973** | 0.8584 |
| | | | 1 | 0.8386 | **0.9973** | 0.8597 |



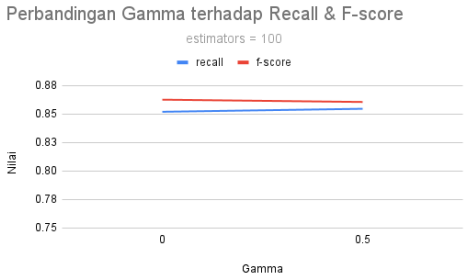

**(a)** Hasil pengukuran setiap kombinasi pengujian pada *estimators* 100




**(b)** Perbandingan *max_depth* terhadap *recall* dan *F-score*




**(c)** Perbandingan *lambda* terhadap *recall* dan *F-score*




**(d)** Perbandingan *gamma* terhadap *recall* dan *F-score*

**Gambar 4.4** Hasil Pengujian pada Estimator 100

Berdasarkan gambar 4.4a terlihat bahwa grafik akurasi berbentuk seperti garis lurus yang statis dan mendekati angka 1.00 seiring bertambahnya nomor kombinasi. Hal ini berarti akurasi pada pengujian XGBoost dengan *estimators* 100 hampir mendekati 100%. Kemudian grafik *recall* terlihat cenderung mengalami penurunan seiring bertambahnya nomor kombinasi walaupun tingkat penurunannya tidak begitu besar. Berdasarkan grafik *f-score*, terlihat bahwa nilai *f-score* cenderung bergerak stabil dari pengujian dengan nomor kombinasi 23 hingga pengujian dengan nomor kombinasi 31, dan mengalami peningkatan yang signifikan dari pengujian dengan nomor kombinasi 32 ke pengujian dengan nomor kombinasi 33. Setelah itu grafik *f-score* cenderung bergerak stabil meskipun terjadi sedikit kenaikan dan sedikit penurunan.

Berdasarkan gambar 4.4b terlihat bahwa grafik *recall* bergerak cenderung menurun seiring bertambahnya nomor kombinasi dengan nilai *recall* tertinggi diperoleh ketika *max_depth* bernilai 5 dan semakin menurun dengan nilai *recall* minimum diperoleh dengan *max_depth* bernilai 15. Berdasarkan grafik *f-score*, terlihat bahwa terjadi peningkatan yang signifikan antara nilai *max_depth* 5 dengan nilai *max_depth* 10, lalu setelah itu grafik *f-score* cenderung bergerak stabil.

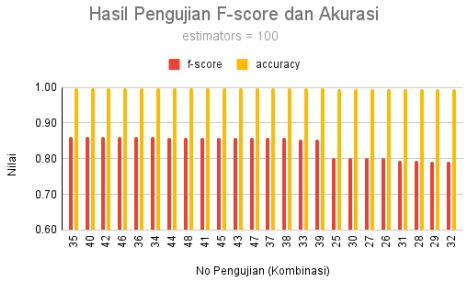
Berdasarkan gambar 4.4c terlihat bahwa nilai *recall* cenderung menurun

semakin bertambahnya nilai *lambda*, dengan nilai *recall* maximum diperoleh ketika nilai lambda bernilai 5 dan nilai minimum *recall* diperoleh dengan nilai *lambda* 1. Sedangkan nilai *F-score* bergerak sebaliknya, yaitu nilai *f-score* cenderung bergerak naik seiring bertambahnya nilai lambda, dan nilai *f-score* minimum diperoleh ketika *lambda* bernilai 0.5 dan nilai maksimum *f-score* diperoleh ketika *lambda* bernilai 1.

Berdasarkan gambar 4.4d terlihat bahwa nilai *recall* cenderung bergerak naik seiring bertambahnya nilai *gamma*, dengna nilai *recall* minimum diperoleh ketika *gamma* bernilai 0 dan nilai *recall* maksimum diperoleh ketika *gamma* bernilai 0.5. Sedangkan berdasarkan grafik *f-score* terlihat sebaliknya, di mana nilai *f-score* cenderung bergerak menurun dengan nilai maksimum *f-score* diperoleh ketika *gamma* bernilai 0 dan nilai minimum *f-score* diperoleh ketika *gamma* bernilai 0.5.

Selanjutnya dilakukan plot untuk mengetahui nilai kombinasi berapa yang mendapatkan nilai *recall*, *f-score* dan akurasi tertinggi.


**(a)** Hasil pengujian dengan menggunakan metrik pengukuran *recall*




**(b)** Hasil pengujian dengan menggunakan metrik pengukuran *f-score* dan akurasi

**Gambar 4.5** Hasil pengujian yang telah diurutkan dari nilai tertinggi ke nilai terendah pada *estimators* 100

Berdasarkan gambar 4.5a terlihat bahwa nilai *recall* tertinggi diperoleh oleh kombinasi pengujian nomor 27 dan 31 dengan nilai sebesar 0.8545 atau sebesar 85.45%. Berdasarkan gambar 4.5b terlihat bahwa nilai *f-score* tertinggi diperoleh oleh kombinasi pengujian dengan nomor 35 dan 40 dengan nilai *f-score* sebesar 0.8439 atau sebesar 84.39%. Sedangkan untuk akurasi, terdapat beberapa nomor pengujian yang memperoleh akurasi tertinggi dengan nilai akurasi tertinggi diperoleh sebesar 0.9973 atau sebesar 99.73% yang diperoleh oleh kombinasi pengujian dengan nomor 34, 35, 38, 39, 41, 42, 43, 44, 45, 46, 47, dan 48. Nomor kombinasi dengan *hyperparameter* yang diuji dapat dilihat berdasarkan tabel 4.10.

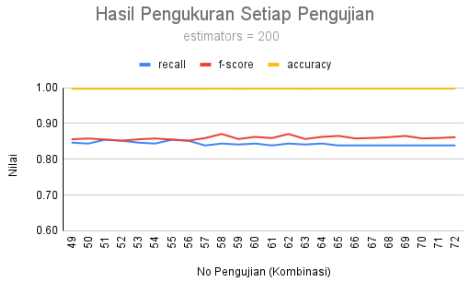
#### 4.3.3.3 Pengujian Estimators 200

Pengujian ini akan menguji model XGBoost dengan *estimators* atau jumlah *tree* yang akan dibentuk sebesar 200. Pengujian ini akan dikombinasikan dengan *max_depth* atau kedalaman tree dengan nilai 5, 10, dan 15, kemudian *eta* dengan nilai 0.1 dan 0.3, lalu *gamma* dengan nilai 0 dan 0.5 dan *lambda* dengan nilai 0.5 dan 1. Hasil pengujian yaitu *recall*, *accuracy* dan *f-score* diambil berdasarkan rata-rata proses *5-fold cross-validation*. Tabel 4.13 menunjukkan hasil pengujian XGBoost dengan *estimators* sebesar 200.

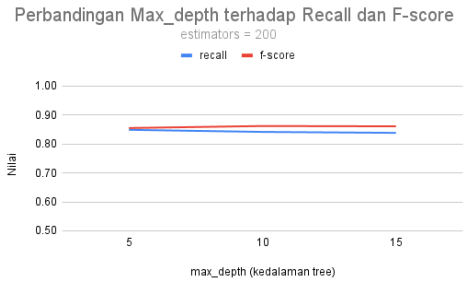
**Tabel 4.13** Hasil Pengujian XGBoost dengan Estimators 200

| max_depth | eta | gamma | lambda | recall | accuracy | f-score |
|---|---|---|---|---|---|---|
| 5 | 0.10 | 0 | 0.5 | 0.8465 | 0.9972 | 0.8558 |
| | | | 1 | 0.8439 | 0.9972 | 0.8580 |
| | | 0.5 | 0.5 | **0.8544** | 0.9971 | 0.8553 |
| | | | 1 | 0.8518 | 0.9971 | 0.8520 |
| | 0.30 | 0 | 0.5 | 0.8465 | 0.9972 | 0.8558 |
| | | | 1 | 0.8439 | 0.9972 | 0.8580 |
| | | 0.5 | 0.5 | **0.8544** | 0.9971 | 0.8553 |
| | | | 1 | 0.8518 | 0.9971 | 0.8520 |
| 10 | 0.10 | 0 | 0.5 | 0.8386 | 0.9973 | 0.8593 |
| | | | 1 | 0.8439 | **0.9975** | **0.8705** |
| | | 0.5 | 0.5 | 0.8413 | 0.9972 | 0.8567 |
| | | | 1 | 0.8439 | 0.9973 | 0.8625 |
| | 0.30 | 0 | 0.5 | 0.8386 | 0.9973 | 0.8593 |
| | | | 1 | 0.8439 | **0.9975** | **0.8705** |
| | | 0.5 | 0.5 | 0.8413 | 0.9972 | 0.8567 |
| | | | 1 | 0.8439 | 0.9973 | 0.8625 |
| 15 | 0.10 | 0 | 0.5 | 0.8386 | 0.9974 | 0.8653 |
| | | | 1 | 0.8386 | 0.9973 | 0.8582 |
| | | 0.5 | 0.5 | 0.8386 | 0.9973 | 0.8595 |
| | | | 1 | 0.8386 | 0.9973 | 0.8618 |
| | 0.30 | 0 | 0.5 | 0.8386 | 0.9974 | 0.8653 |
| | | | 1 | 0.8386 | 0.9973 | 0.8582 |
| | | 0.5 | 0.5 | 0.8386 | 0.9973 | 0.8595 |
| | | | 1 | 0.8386 | 0.9973 | 0.8618 |

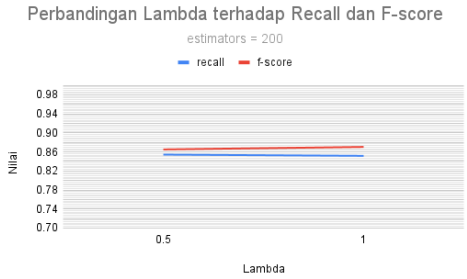
Berdasarkan gambar 4.6a terlihat bahwa grafik akurasi bergerak stabil dan



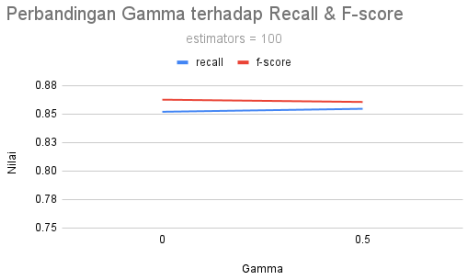

**(a)** Hasil pengukuran setiap kombinasi pengujian pada *estimators* 200




**(b)** Perbandingan *max_depth* terhadap *recall* dan *f-score*




**(c)** Perbandingan lambda terhadap *recall* dan *f-score*




**(d)** Perbandingan *gamma* terhadap *recall* dan *f-score*

**Gambar 4.6** Hasil Pengujian pada Estimator 200

statis pada angka mendektai 1.00 sehingga akurasi mendekati 100%. Pada grafik *recall*, terlihat bahwa nilai recall bergerak cenderung turun seiring bertambahnya nomor kombinasi pengujian. Pada grafik *f-score* terlihat bahwa terjadi kenaikan signifikan namun tidak terlalu besar dibandingkan pengujian lainnya yang terjadi antara kombinasi pengujian nomor 56 hingga 58, lalu kemudian terjadi kenaikan dan penurunan, tetapi setelah itu grafik *f-score* cenderung bergerak stabil.

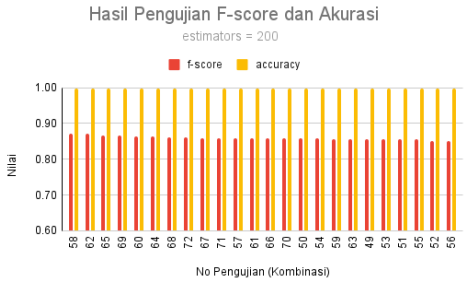
Gambar 4.6b menunjukkan perbandingan grafik *recall* dan *f-score* dibandingkan dengan pengujian *max_depth*. Terlihat bahwa grafik *recall* cenderung bergerak turun seiring bertambahnya nilai *max_depth*. Nilai *recall* maksimum diperoleh ketika nilai *max_depth* sebesar 5 dan nilai minimum *recall* diperoleh ketika nilai *max_depth* sebesar 15. Grafik *f-score* bergerak sebaliknya dimana grafik cenderung bergerak naik seiring bertambahnya nilai *max_depth*. Nilai *f-score* minimum diperoleh ketika nilai *max_depth* sebesar 5 dan nilai *f-score* maksimum diperoleh ketika nilai *max_depth* sebesar 15.

Gambar 4.6c menunjukkan perbandingan antara nilai *recall* dan *f-score* dengan pengujian nilai *lambda*. Pada gambar tersebut, terlihat bahwa nilai *recall* cenderung bergerak turun seiring bertambahnya nilai *lambda*. Nilai maksimum *recall* diperoleh ketika nilai *lambda* sebesar 0.5 dan nilai minimum *recall* diperoleh ketika nilai *lambda* sebesar 1. Sedangkan pada grafik *f-score* bergerak

sebaliknya dimana grafik cenderung bergerak naik seiring bertambahnya nilai *lambda*. Terlihat bahwa nilai minimum *f-score* diperoleh ketika nilai *lambda* sebesar 0.5 dan nilai maksimum *f-score* diperoleh ketika nilai *lambda* sebeasr 1.

Gambar 4.6d menunjukkan perbandingan antara nilai *recall* dan *f-score* dengan pengujian nilai *gamma*. Gambar tersebut menunjukkan bahwa grafik *recall* cenderung bergerak naik seiring bertambahnya nilai *gamma*. Nilai minimum *recall* diperoleh ketika nilai *gamma* sebesar 0 dan nilai maksimum *recall* diperoleh ketika nilai *gamma* sebesar 0.5. Sedangkan grafik *f-score* bergerak sebaliknya dimana grafik tersebut cenderung bergerak turun ketika nilai *gamma* bertambah. Nilai *f-score* maksimum diperoleh ketika nilai *gamma* sebesar 0 dan nilai minimum *f-score* diperoleh ketika nilai *gamma* sebesar 0.5.


**(a)** Hasil pengujian dengan menggunakan metrik pengukuran *recall*




**(b)** Hasil pengujian dengan menggunakan metrik pengukuran *f-score* dan akurasi

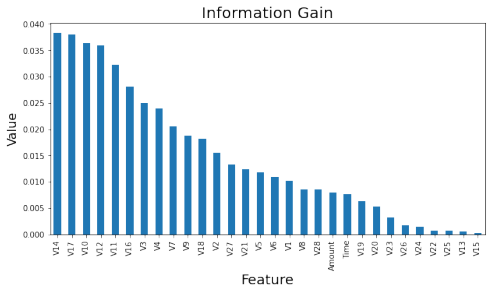
**Gambar 4.7** Hasil pengujian yang telah diurutkan dari nilai tertinggi ke nilai terendah pada *estimators* 200

Gambar 4.7a menunjukkan hasil pengujian yang mengukur nilai *recall* yang telah diurutkan. Berdasarkan gambar tersebut, terlihat bahwa nilai *recall* tertinggi diperoleh oleh kombinasi pengujian nomor 51 dan 52 dengan nilai sebesar 0.8544 atau sebesar 85.44%. Sedangkan nilai *f-score* dan akurasi tertinggi diperoleh oleh kombinasi pengujian nomor 58 dan 62 dengan nilai *f-score* sebesar 0.8705 atau sebesar 87.05% dan nilai akurasi sebesar 0.9975 atau sebesar 99.75%.

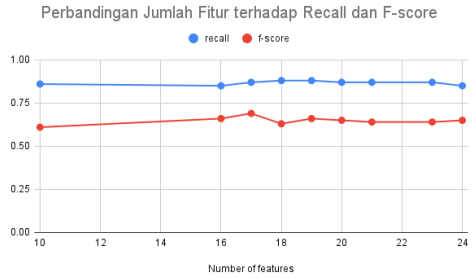
#### 4.3.3.4 Pengujian XGBoost dengan Seleksi Fitur

Pengujian ini dilakukan untuk mengetahui apakah seleksi fitur mampu meningkatkan performa XGBoost, maka dilakukan pengujian XGBoost dengan seleksi fitur menggunakan *information gain*. Dengan menggunakan *information gain*, tentunya fitur-fitur yang kurang begitu berpengaruh terhadap model bisa dihilangkan. Dengan mengeliminasi fitur-fitur yang kurang begitu berpengaruh, tentunya akan mengurangi kompleksitas model dan beban serta waktu komputasi dalam pelatihan model sehingga diharapkan mampu meningkatkan kemampuan

model dalam mengklasifikasikan transaksi penipuan [27]. Pengujian dengan seleksi fitur menggunakan *information gain* dilakukan untuk mengetahui fitur-fitur mana saja yang paling mempengaruhi model serta untuk menguji berapa banyak fitur yang perlu digunakan untuk mendapatkan *recall* tertinggi.




**(a)** Nilai *information gain* untuk setiap fitur yang telah diurutkan dari yang tertinggi hingga terendah.




**(b)** Hasil pengujian untuk menentukan jumlah fitur yang digunakan.

**Gambar 4.8** Hasil Pengujian dengan Seleksi Fitur

Gambar 4.8a menunjukkan nilai *information gain* pada setiap fitur. Dapat terlihat bahwa fitur yang memperoleh nilai *information gain* yang tinggi diantaranya adalah fitur V14, V17, V10, V12, V11, dan V16 dan nilai information gain terendah diperoleh diantaranya diperoleh oleh fitur V24, V22, V25, V13, dan V15. Seperti yang telah dibahas pada bab 3, pada gambar 3.4, terlihat bahwa fitur V14, V17, V10, V12, V11, dan V16 memang memiliki korelasi yang kuat terhadap variabel *class*. Sedangkan fitur V24, V22, V25, V13, dan V15 juga terlihat hampir tidak memiliki korelasi yang kecil atau bahkan tidak memiliki korelasi sama sekali terhadap variabel *class*.

Selanjutnya dilakukan pengujian untuk memilih berapa banyak fitur yang harus digunakan agar memaksimalkan nilai *recall*. Salah satu kombinasi pengujian *hyperparameter* akan dipilih dalam pengujian sebagai pembanding hasil evaluasi ini yaitu kombinasi *hyperparameter* nomor 7. *Hyperparameter* pada kombinasi tersebut dapat dilihat berdasarkan tabel 4.10.

Berdasarkan gambar 4.8b terlihat bahwa dengan menggunakan 18 dan 19 fitur dengan *information gain* tertinggi, keduanya memperoleh nilai recall tertinggi sebesar 0.8842 atau 88.42% dan nilai *f-score* masing-masing sebesar 0.6339 atau sebesar 63.39% dan 0.6614 atau sebesar 66.14%. Menggunakan 19 fitur yang memperoleh *information gain* tertinggi lebih baik daripada menggunakan 18 fitur karena meskipun nilai *recall* yang diperoleh sama, namun nilai *f-score* dengan menggunakan 19 fitur lebih tinggi. Dengan menggunakan 19 fitur, fitur-fitur yang

namanya tidak disamarkan seperti *time* dan *amount* ternyata tidak begitu mempengaruhi peforma model.

Jika dibandingkan pengujian dengan menggunakan seleksi fitur dan pengujian sebelumnya, hasil pengujian menggunakan seleksi fitur yang terbaik memperoleh nilai *recall* sebesar 88.42% dan *f-score* sebesar 66.14%. Pada hasil pengujian pada kombinasi *hyperparameter* npomor 7 memperoleh nilai *recall* sebesar 85.25% dan *f-score* sebesar 65.06%. Dalam hal ini terdapat perbedaan nilai *recall* tertinggi sekitar 0.0316 atau sekitar 3% dan perbedaan nilai *f-score* sebesar 0.0108 atau hanya sekitar 1%. Berdasarkan hasil diatas dapat disimpulkan bahwa menggunakan seleksi fitur dengan *informatin gain* tidak berpengaruh banyak dalam meningkatkan nilai *recall* ataupun *f-score* dalam pengujian ini.

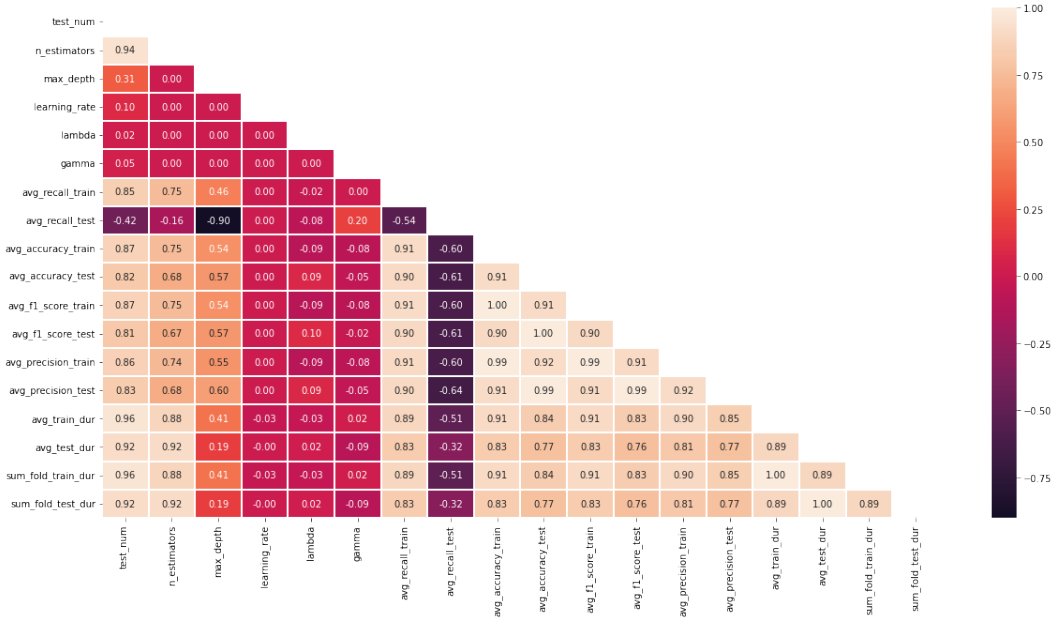
### 4.3.4 Pembahasan Pengujian

**Tabel 4.14** Kombinasi hyperparameter yang mengasilkan nilai *recall*, *f-score*, dan akurasi tertinggi

| No | n_estimators | max_depth | eta | lambda | gamma | recall | accuracy | f-score |
|---|---|---|---|---|---|---|---|---|
| 3 | 50 | 5 | 0.1 | 0.5 | 0.5 | **0.8598** | 0.9928 | 0.7034 |
| 7 | 50 | 5 | 0.3 | 0.5 | 0.5 | **0.8598** | 0.9928 | 0.7034 |
| 58 | 200 | 10 | 0.1 | 1 | 0 | 0.8439 | **0.9975** | **0.8705** |
| 62 | 200 | 10 | 0.3 | 1 | 0 | 0.8439 | **0.9975** | **0.8705** |

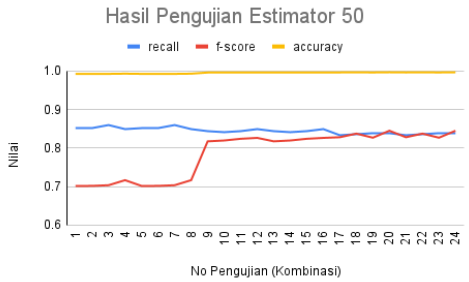
Berdasarkan tabel 4.14 ada beberapa kombinasi pengujian yang memperoleh nilai tertinggi pada pengukuran berdasarkan *recall*, *f-score* dan akurasi yang diambil dari keseluruhan pengujian dari *estimators* 50, 100, dan 200. Nilai *recall* terbaik diperoleh oleh kombinasi pengujian nomor 3 dan nomor 7 dengan nilai recall tertinggi sebesar 0.8598 atau sebesar 85.98%. Hal yang membedakan antara kombinasi pengujian nomor 3 dan nomor 7 adalah besaran *eta* yang digunakan. Kombinasi pengujian nomor 3 menggunakan *eta* sebesar 0.1 sedangkan kombinasi pengujian nomor 7 menggunakan *eta* sebesar 0.3. Nilai *f-score* dan akurasi terbaik diperoleh oleh kombinasi pengujian nomor 58 dan kombinasi pengujian nomor 62 dengan nilai *f-score* sebesar 0.8705 atau sebesar 87.05% dan nilai *akurasi* sebesar 0.9975 atau sebesar 99.75%. Hal yang membedakan kombinasi pengujian nomor 58 dan 62 adalah nilai *eta* yang digunakan. Kombinasi pengujian nomor 58 menggunakan *eta* sebesar 0.1 sedangkan kombinasi pengujian nomor 62 menggunakan *eta* 0.3. Berdasarkan tabel ini, dapat disimpulkan juga bahwa *eta* dengan nilai 0.1 dan *eta* 0.3 mampu

memperoleh performa terbaiknya dari pengukuran *recall*, *f-score*, dan akurasi.

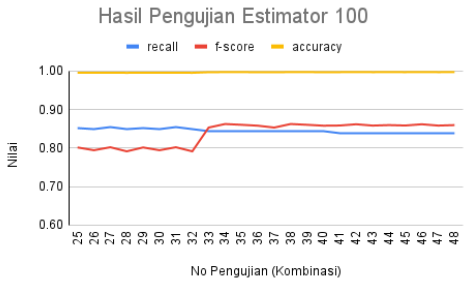



**Gambar 4.9** Matriks Korelasi Pengukuran Pengujian dengan *Hyperparameter* yang diuji.

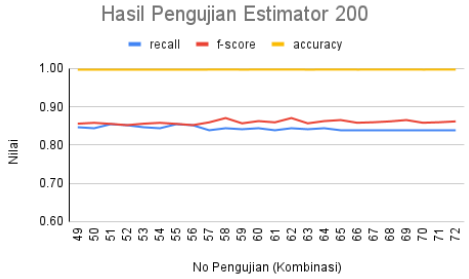
Gambar 4.9 merupakan antar hasil pengukuran pengujian dengan *hyperparameter* yang diuji. Berdasarkgan gambar tersebut, terlihat bahwa nilai *recall* (*avg_recall_test*)memiliki korelasi yang negatif dengan *hyperparameter max_depth* dan *n_estimators*, dan korelasi positif dengan *hyperparameter gamma*. *Lambda* memiliki korelasi yang sedikit negatif dengan *recall* tetapi memiliki korelasi positif dengan *f-score*. Dapat dilihat juga bahwa ketika nilai *recall* memiliki korelasi negatif, maka *f-score* (*avg_f1_score_test*) memiliki korelasi yang positif, hal ini dapat disebabkan juga karena *precision* (*avg_precision_test*) juga memiliki korelasi positif sehingga mempengaruhi perhitungan *f-score*.



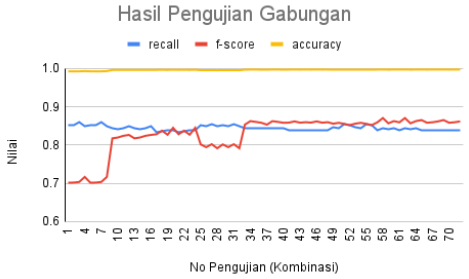

**(a)** Hasil pengukuran setiap kombinasi pengujian pada *estimators* 50




**(b)** Hasil pengukuran setiap kombinasi pengujian pada *estimators* 100




**(c)** Hasil pengukuran setiap kombinasi pengujian pada *estimators* 200

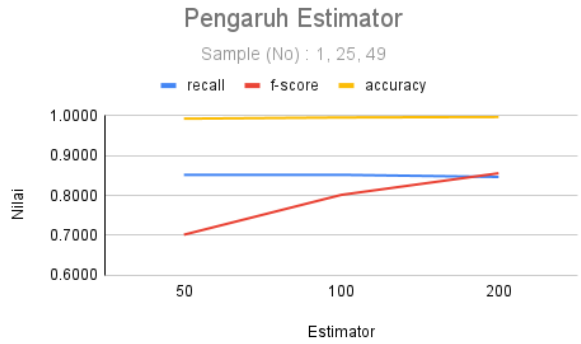



**(d)** Hasil pengukuran gabungan *estimator*

**Gambar 4.10** Hasil Pengujian setiap *estimator*

Gambar 4.10d merupakan hasil pengujian gabungan seluruh *estimator* yang diuji. Berdasarkan gambar tersebut, terlihat bahwa nilai akurasi mendekati nilai 1.0 atau hampir 100%. Hal ini sesuai ekspektasi, mengingat data latih dan data uji mengalami *class imbalance* meskipun pada data latih sudah dilakukan *oversampling* menggunakan SMOTE, tetapi proses *oversampling* tidak menyeimbangkan data latih tetapi hanya meningkatkan jumlah kelas minoritas. Selain itu yang perlu diperhatikan adalah terdapat adanya kecenderungan nilai *recall* menurun seiring bertambahnya nomor pengujian serta pola kenaikan grafik *f-score* yang sama pada gambar 4.10a, 4.10b, dan 4.10c. Pola penurunan grafik *recall* seiring bertambahnya nomor pengujian sebagian besar dipengaruhi oleh *n_estimator* yang diuji, karena nomor pengujian 1-24 menguji *n_estimator* 50, nomor 25-48 menguji *n_estimators* 100, serta nomor 49-72 menguji *n_estimator* 200. Selain itu nilai *max_depth* yang lebih kecil pada setiap masing-masing pengujian juga memperngaruhi grafik recall tersebut. Hal ini juga dapat dibuktikan melalui gambar 4.9 dimana *n_estimator* dan *max_depth* mempengaruhi grafik recall. Kenaikan grafik *f-score* seiring bertambahnya nomor pengujian juga sebagian besar dipengaruhi *hyperparameter* Pola kenaikan pada grafik *f-score* siring bertambahnya nomor pengujian juga dipengaruhi oleh *n_estimators*, tetapi pola kenaikan pada *f-score* disebabkan oleh *hyperparameter max_depth*. Dapat

dilihat bahwa pola kenaikan *f-score* secara signifikan pada gambar 4.10a terjadi pada pengujian antara nomor 8 dan 9, pada gambar 4.10b terjadi pada nomor pengujian antara nomor 32 dan 33, serta pada gambar 4.10c terjadi pada nomor pengujian antara 56 dan 57 dimana pada masing-masing nomor sebelumnya menguji *max_depth* 5 dan nomor setelahnya menguji *max_depth* 10.




**Gambar 4.11** Hasil pengukuran *n_estimators* pada kombinasi pengujian pada nilai *hyperparameter* yang sama terkecuali *n_estimators*

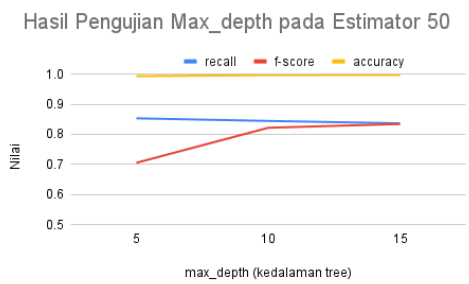
Gambar 4.11 menunjukkan hasil pengujian pada kombinasi pengujian nomor 1, 25 dan 49. Kombinasi tersebut dipilih sebagai visualisasi karena pada kombinasi tersebut, semua nilai *hyperparameter* yang diuji sama kecuali *hyperparameter n_estimators*. Bila dilihat berdasarkan jumlah *estimator* pada masing-masing kombinasi pengujian terbaik, terlihat bahwa *n_estimator* dengan nilai 50 memiliki nilai *recall* lebih tinggi daripada *n_estimator* dengan nilai 200. Tetapi kombinasi pengujian dengan *n_estimator* dengan nilai 200 memiliki nilai akurasi dan *f-score* yang lebih tinggi, meskipun perbedaan akurasi tidak terlalu signifikan. Dapat disimpulkan bahwa semakin kecil nilai *n_estimator*, maka nilai *recall* semakin besar, namun nilai *f-score* semakin kecil. Hal ini dapat terjadi karena terdapat *trade-off* antara nilai *recall* dan nilai *precision*, dimana jika nilai *recall* semakin tinggi, maka nilai *precision semakin rendah*, dan nilai *precision* akan mempengaruhi perhitungan *f-score*. Selain itu, hal lain yang dapat menyebabkan *recall* semakin turun seiring bertambahnya nilai *n_estimator* adalah adanya kemungkinan model mengalami *overfit* dikarenakan jumlah *tree* yang dibangun semakin banyak sehingga *model* kurang mampu melakukan generalisasi terhadap data baru. Pengukuran menggunakan akurasi tidak menunjukkan perubahan yang signifikan antara *n_estimator* 50 dan 200 sehingga *n_estimator* memiliki pengaruh yang kecil dalam hal akurasi. Hal ini dapat dilihat pada gambar 4.10 yang memperlihatkan pengujian *estimator* 50, 100, dan 200 terhadap *recall*, *f-score*, dan akurasi.

Berdasarkan gambar 4.10d terlihat bahwa grafik *recall* memang cenderung

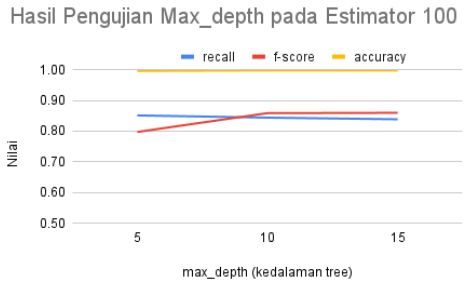
turun seiring bertambahnya nomor pengujian, dimana pengujian dengan kombinasi nomor 1 hingga 24 menguji *estimator* 50, kombinasi nomor 24 hingga 48 menguji *estimator* 100 dan kombinasi 49 hingga 72 menguji *estimator* 200. Sehingga dapat disimpulkan bahwa grafik *recall* cenderung seiring bertambahnya *estimator*. Pada grafik *recall* cenderung naik seiring bertambahnya kombinasi, dimana kenaikan signifikan terjadi antara kombinasi nomor 7 menuju 10, dan kombinasi nomor 31 menuju 34, serta terdapat kenaikan kecil antara kombinasi nomor 55 dan 58. Sehingga dapat disimpulkan bahwa semakin banyak nilai *estimators* akan mengakibatkan model cenderung mengalami *overfit* yang kurang bagus performanya terhadap data uji, khususnya untuk mendeteksi transaksi penipuan yang mementingkan pengukuran *recall*.

Berdasarkan tabel 4.14 juga terlihat bahwa nilai *max_depth* yang dengan nilai 5 memiliki nilai *recall* yang lebih tinggi daripada nilai *max_depth* dengan nilai 10. Hal ini juga bisa diakibatkan oleh *overfitting* yaitu kondisi ketika model sangat baik dalam memprediksi data latih dan menyimpan informasi data latih dengan baik namun kurang baik dalam memprediksi data yang belum terlihat sebelumnya karena model belum menangkap pola yang lebih umum.

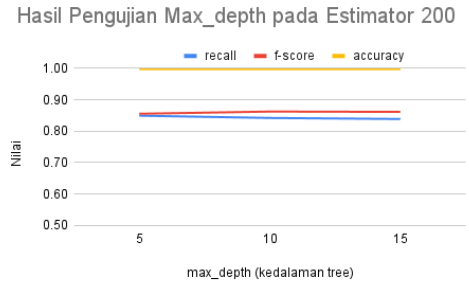
Gambar 4.12 menunjukkan pengujian *hyperparameter max_depth* 5, 10, dan 15 pada masing-masing *estimator* 50, 100, dan 200. Pada gambar ini, terlihat bahwa akurasi selalu bergerak statis mendekati angka 1.0 atau 100% baik pada *estimator* 50, 100, dan 200. Sehingga dapat disimpulkan bahwa *max_depth* kurang mempengaruhi evaluasi berdasarkan akurasi.



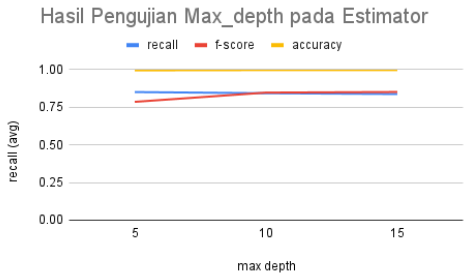

**(a)** Hasil pengujian *max_depth* pada *estimator* 50




**(b)** Hasil pengujian *max_depth* pada *estimator* 100




**(c)** Hasil pengujian *max_depth* pada *estimator* 200

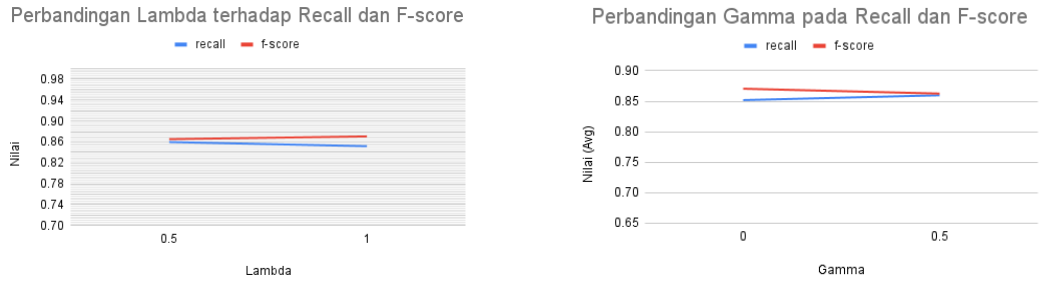



**(d)** Hasil pengujian *max_depth* gabungan

**Gambar 4.12** Hasil Pengujian *max_depth*

Pada gambar 4.12a dan gambar 4.12b terlihat jelas bahwa terdapat kenaikan signifikan pada grafik *f-score* antara *max_depth* 5 dan 10, tetapi pada gambar 4.12c kenaikan *f-score* tidak terlalu signifikan. Hal ini tentu tidak hanya diperngaruhi oleh *max_depth* saja, tetapi ada *hyperparameter* lain yang memperngaruhi hal tersebut seperti *estimator* atau jumlah *tree* yang akan dibangun. Secara keseluruhan, terlihat pada gambar 4.12 terlihat bahwa *f-score* akan naik seiring bertambahnya *max_depth* khususnya pada *max_depth* 5 hingga 10. Pada gambar 4.12a, 4.12b, dan gambar 4.12c terlihat bahwa nilai *recall* cenderung bergerak menurun di setiap pengujian *estimator* 50, 100, dan 200. Secara keseluruhan, grafik *recall* memang bergerak menurun seiring bertambahnya nilai *max_depth*. Hal ini disebabkan apabila *tree* yang dibangun memiliki kedalaman yang dalam atau bahkan sangat dalam, dapat membuat *tree* mampu menyimpan seluruh informasi pada data latih sehingga mampu memprediksi dengan baik pada data latih, namum *tree* tersebut tidak menyimpan informasi secara umum yang mengakibatkan *tree* gagal dalam memprediksi data uji atau data baru, khususnya data transaksi *fraud* sebagai kelas minoritas. Sehingga dapat disimpulkan bahwa semakin besar nilai *max_depth* yang digunakan, maka semakin besar kemungkinan *tree* mengalami *overfit*, yang dapat menyebabkan performa kurang baik pada data uji.

Berdasarkan tabel 4.14 terlihat bahwa nilai *lambda* sebesar 0.5 memperoleh nilai *recall* tertinggi dibandingkan dengan lamdba sebesar 1. Selain itu nilai *gamma* sebesar 0 memperoleh nilai *f-score* lebih tinggi daripada nilai *gamma* sebesar 0.5. Gambar 4.13 menunjukkan hasil pengujian terhadap *lambda* dan *gamma*. Dapat terlihat bahwa semakin besar nilai *gamma* maka nilai *recall* semakin meningkat, namun nilai *f-score* semakin menurun. Sedangkan semakin besar nilai *lambda*, maka nilai *recall* semakin menurun, sedangkan nilai *f-score* semakin besar. Hal ini dapat terjadi karena kedua *hyperparameter* tersebut merupakan konstanta regularisasi. Regularisisasi berperan untuk mencegah *overfitting* pada model. Diperlukan *hyperparameter tuning* yang tepat untuk mendapatkan nilai regularisasi yang mampu mencegah *overfitting*, Jika nilai regularisasi terlalu kecil, maka akan membuat model tetap mengalami *overfitting*, tetapi jika nilai regularisasi terlalu besar, maka dapat membuat model mengalami *underfitting*. Memperbesar nilai *gamma* dari 0 menjadi 0.5 mengurangi *overfitting* bila diukur menggunakan *recall* karena gamma berperan untuk mengurangi nilai *gain*, sehingga apabila nilai *gain* merupakan nilai negatif, maka algoritme XGBoost akan berhenti membangun *tree*. Sedangkan nilai *lambda* sebesar 1 mengalami sedikit *underfitting* bila dibandingkan dengan nilai lambda 0.5 jika diukur menggunakan *recall*.




**(a)** Hasil pengujian *lambda* **(b)** Hasil pengujian *gamma*

**Gambar 4.13** Hasil Pengujian *lambda* dan *gamma*

### 4.3.5 Pengujian XGBoost pada Data Uji

Tahap selanjutnya adalah memilih *hyperparameter* terbaik yang untuk dilakukan pengujian menggunakan data uji. *Hyperparameter* tersebut dipilih berdasarkan hasil pengujian *5-fold cross-validation* yang telah dilakukan pada tahap sebelumnya. Tabel 4.14 merupakan *hyperparameter* yang mengasilkan nilai *recall*, *f-score*, dan akurasi tertinggi berdasarkan pengujian yang dilakukan pada tahap sebelumnya.

Berdasarkan tabel 4.14, kombinasi pengujian nomor 7 dan 62 akan dipilih

sebagai kombinasi *hyperparameter* yang akan dilakukan pengujian menggunakan data uji yang belum terpakai sebelumnya. Hal ini dikarenakan kedua kombinasi ini menggunakan *eta* tertinggi yaitu sebesar 0.3 yang berarti proses pelatihan akan berlangsung lebih cepat dibandingkan proses pelatihan menggunakan *eta* 0.1. Proses pelatihan akan menggunakan data *training* tanpa melakukan *5-fold cross-validation*. Selanjutnya akan dilakukan pengujian menggunakan data uji.

**Tabel 4.15** Hasil Pengujian XGBoost pada Data Uji

| No | n_estimators | max_depth | eta | lambda | gamma | recall | accuracy | f-score |
|---|---|---|---|---|---|---|---|---|
| 7 | 50 | 5 | 0.3 | 0.5 | 0.5 | **0.8526** | 0.9909 | 0.6506 |
| 62 | 200 | 10 | 0.3 | 1 | 0 | 0.8421 | **0.9977** | **0.8791** |

Tabel 4.15 yang menunjukkan hasil pengujian XGBoost dengan *hyperparameter* yang mengasilkan nilai *recall*, *f-score*, dan akurasi tertinggi pada data uji. Berdasarkan tabel tersebut, nilai *recall* tertinggi diperoleh sebesar 0.8526 atau sebesar 85.25% pada kombinasi *hyperparameter* nomor 6 dan nilai *recall* terendah diperoleh sebesar 0.8421 atau 84.21% oleh kombinasi *hyperparameter* nomor 62. Kombinasi *hyperparameter* nomor 62 memperoleh nilai akurasi tertinggi sebesar 0.9977 atau sebesar 99.77% dan nilai terendah diperoleh oleh kombinasi *hyperparameter* nomor 6 sebesar 0.9909 atau sebesar 99.09%. Nilai *f-score* tertinggi diperoleh sebesar 0.8791 atau sebesar 87.91% oleh kombinasi *hyperparameter* nomor 62 dan nilai terendahnya diperoleh sebesar 0.6505 atau 65.06% oleh kombinasi *hyperparameter* nomor 6.

Berdasarkan tabel 4.15 dapat disimpulkan bahwa kombinasi *hyperparameter* nomor 6 merupakan kombinasi *hyperparameter* terbaik karena menghasilkan nilai *recall* tertinggi. Hal ini dikarenakan dalam kasus transaksi penipuan kartu kredit, nilai *recall* yang tinggi menunjukkan bahwa model mampu menekan *false negative* yaitu transaksi yang diprediksi sebagai *non fraud* namun sebenarnya transaksi tersebut merupakan transaksi *fraud*. Sehingga, tingginya nilai *recall* sangat penting dalam mendeteksi transaksi penipuan kartu kredit.

Jika melihat pengujian dengan menggunakan seleksi fitur, hasil dengan nilai *recall* tertinggi diperoleh menggunakan 19 fitur yang memperoleh nilai *information gain* tertinggi yang dapat dilihat berdasarkan gambar 4.8b. Berdasarkan pengujian tersebut, nilai *recall* tidak meningkat begitu signifikan, begitu juga dengan nilai *f-score*. Selain itu, fitur-fitur yang tidak disamarkan tidak

memiliki terlalu berpengaruh terhadpa performa model, seperti fitur *amount* dan *time* berdasarkan gambar 4.8.

Berdasarkan penelitian [28], dalam melakukan deteksi transaksi penipuan kartu kredit, terdapat beberapa fitur utama yang berisikan informasi mengenai suatu transaksi yang dilakukan. Berdasarkan beberapa penelitian yang diamati, fitur-fitur utama yang digunakan hampir sama. Hal ini dikarenakan data-data yang dikumpulkan selama melakukan transaksi kartu kredit mengikuti standar pelaporan internasional. Gambar 4.14 menunjukkan beberapa fitur yang dikumpulkan dalam sebuah transaksi kartu kredit berdasarkan standar pelaporan internasional.

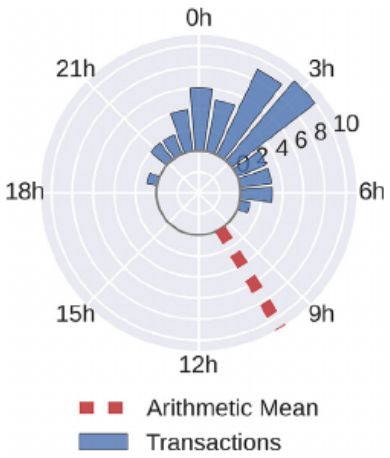
| Attribute name | Description |
|---|---|
| Transaction ID | Transaction identification number |
| Time | Date and time of the transaction |
| Account number | Identification number of the customer |
| Card number | Identification of the credit card |
| Transaction type | ie. Internet, ATM, POS, ... |
| Entry mode | ie. Chip and pin, magnetic stripe, ... |
| Amount | Amount of the transaction in Euros |
| Merchant code | Identification of the merchant type |
| Merchant group | Merchant group identification |
| Country | Country of trx |
| Country 2 | Country of residence |
| Type of card | ie. Visa debit, Mastercard, American Express... |
| Gender | Gender of the card holder |
| Age | Card holder age |
| Bank | Issuer bank of the card |

**Gambar 4.14** Beberapa fitur utama yang dikumpulkan dalam transaksi kartu kredit [28].

Dalam gambar 4.14 terlihat ada 2 buah fitur yang sama dengan fitur pada *dataset* yang digunakan dalam penelitian ini, yaitu *time* dan *amount*. Fitur *time* berisikan tanggal dan waktu terjadinya transaksi sedangkan fitur *amount* menunjukkan nominal dari transaksi yang dilakukan dalam mata uang *euro*. Penelitian tersebut menjelaskan bahwa jika menggunakan informasi yang berkaitan dengan satu transaksi saja tidaklah cukup dalam mendeteksi penipuan kartu kredit terlebih jika jika hanya menggunakan fitur-fitur yang ada pada gambar 4.14.

Minimnya *information gain* yang diperoleh oleh fitur *time* dan *amount* didukung didukung oleh penelitian [28] yang menjelskan bahwa dalam melakukan deteksi penipuan kartu kredit biasanya dilakukan beberapa teknik *feature engineering* untuk menganalisa kebiasan konsumen dalam melakukan belanja menggunakan kartu kredit. Tentunya hal tersebut tidak bisa dilakukan dalam penelitian ini karena nama fitur yang disamarkan. Penelitian tersebut juga menyebutkan ada metode yang biasa dilakukan seperti analisa kebiasaan belanja konsumen melibatkan informasi transaksi kartu kredit sebelumnya. Metode ini

kurang baik karena kurang mampu menangkap informasi mengenai kebiasaan konsumen, selain itu juga menimbulkan pertanyaan baru mengenai informasi mengenai transaksi sebelumnya yang harus diambil. Metode lainnya adalah menggunakan fungsi agregasi yaitu mengelompokkan beberapa data seperti pengelompokkan berdasarkan kartu,jenis transaksi, *merchant*, dan menghitung jumlah transaksi yang terjadi berdasarkan akumulasi *amount*. Metode agregasi fitur ini juga memiliki kelemahan, yaitu seperti berapa banyak transaksi yang perlu dilakukan agregasi atau apabila melakukan analisa berdasarkan fitur *time*, maka jika menghitung rata-rata aritmatika waktu transaksi untuk mengetahui kapan biasanya konsumen melakukan transaksi juga kurang tepat, seperti yang terlihat pada gambar 4.15.




**Gambar 4.15** Contoh perhitungan rata-rata waktu terjadinya transaksi kartu kredit yang kurang tepat [28].

Berdasrkan gambar 4.15, terlihat bahwa transaksi kartu kredit sebenarnya terjadi dalam rentang waktu 18:00 hingga pukul 06:00, tetapi jika dihitung menggunakan rata-rata waktu yang terjadi, hasilnya menyatakan bahwa rata-rata transaksi yang terjadi sekitar pukul 09:00 - 10:00 yang tentunya kurang tepat.

Sehingga dapat disimpulkan bahwa rendahnya *ninformation gain* pada fitur *time* dan *amount* karena minimnya *feature engineering* yang dilakukan. Metode seperti analisa perilaku konsumen dalam berbelanja dengan mencatat informasi transaksi kartu kredit sebelumnya atau melakukan agregasi transaksi hanya menimbulkan pertanyaan baru seperti berapa transaksi yang harus dicatat sebelumnya atau berapa transaksi yang perlu dilakukan agregasi, dan tentunya juga tidak bisa dilakukan karena hampir semua nama-nama fitur pada dataset ini yang telah disamarkan.

# BAB 5 KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan berdasarkan penelitian, pelatihan, dan pengujian yang dilakukan oleh peneliti. Selain itu terdapat juga saran yang dapat digunakan atau dipertimbangkan pada saat melakukan penelitian di masa mendatang.

## 5.1 Kesimpulan

Kesimpulan dari XGBoost dengan SMOTE untuk Deteksi Penipuan Kartu Kredit melalui *5-fold cross-validation* dan pengujian pada data uji adalah sebagai berikut:

1. *Recall* tertinggi pada data uji diperoleh oleh kombinasi nomor 7 sesuai pada tabel 4.15 dengan nilai *recall* tertinggi sebesar 0.8526 atau 85.26%. Akurasi dan *f-score* tertinggi diperoleh oleh kombinasi nomor 62 dengan nilai masing-masing 0.9977 atau 99.77% dan 0.8791 atau 87.91%.

2. Berdasarkan pengujian menggunakan seleksi fitur yang dilakukan, seleksi fitur menggunakan *information gain* hanya meningkatkan nilai *recall* sebesar 3% dan nilai *f-score* sebesar 1% dengan menggunakan 19 fitur dengan nilai information gain tertinggi. Seleksi fitur yang dilakukan tidak berpengaruh banyak untuk meningkatkan nilai *recall* ataupun *f-score*.

3. Dalam kasus deteksi penipuan kartu kredit yang menekankan sensitivitas (*recall*) untuk deteksi transaksi penipuan (*fraud*) dengan rasio jumlah data yang tentunya lebih rendah dari kelas *non-fraud*, maka kombinasi nomor 7 merupakan *hyperparameter* terbaik karena memperoleh nilai *recall* tertinggi.

4. Berdasarkan hasil pengujian yang dilakukan, semakin besar nilai *n_estimator* dan *max_depth* akan menurunkan nilai *recall* tetapi meningkatkan nilai *f-score* karena adanya *trade off* antara nilai *recall* dan *precision* yang memengaruhi perhitungan *f-score*. Selain itu tingginya nilai *n_estimator* dan *max_depth* akan mengakibatkan model cenderung mengalami *overfit*. Semakin besar nilai *gamma* meingkatkan nilai *recall* serta semkain besar nilai *lambda* menurunkan nilai *recall* Hal ini karena *gamma* dan *lambda* berperan untuk mencegah adanya *overfitting*, tentunya apabila nilainya terlalu besar maka akan menyebabkan model mengalami *underfitting*, tetapi jika nilainya terlalu kecil maka model tetap akan mengalami *overfitting*.

## 5.2 Saran

Saran dari penulis untuk pengembangan deteksi transaksi penipuan kartu kredit dengan di masa mendatang adalah:

1. Melakukan pengujian terhadap pengaruh *hyperparameter* yang terdapat dalam teknik *oversampling* SMOTE seperti rasio kelas yang akan digunakan dan jumlah tetangga yang perlu dipertimbangkan untuk melakukan deteksi penipuan kartu kredit.
2. Menambah jumlah pengujian terhadap *hyperparameter lambda* dan *gamma*, karena penulis hanya menggunakan 2 nilai untuk pengujian terhadap *hyperparameter lambda* dan *gamma*.

# DAFTAR REFERENSI


[1] *Credit card fraud statistics*. [Online]. Available : https://shiftprocessing.com/credit-card-fraud-statistics, [Accessed: 5 December 2021].

[2] S. P. Maniraj, A. Saini, S. Ahmed and S. D. Sarkar, "Credit card fraud detection using machine learning and data science", *International Journal of Engineering Research  Technology (IJERT)*, vol. 8, no. 9, 2019. DOI: http://dx.doi.org/10.17577/IJERTV8IS090031, [Accessed: 27 January 2022].

[3] I. Kaur and M. Kalra, "Ensemble classification method for credit card fraud detection", *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 8, no. 3, pp. 25 579–25 587, 2019. DOI: 10.35940/ijrte.C4213.098319, [Accessed: 27 January 2022].

[4] A. A. Taha and S. J. Malebary, "An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine", *IEEE Access*, vol. 8, pp. 25 579–25 587, 2020. DOI: 10.1109/ACCESS.2020.2971354, [Accessed: 10 October 2021].

[5] S. Makki, Z. Assaghir, Y. Taher, R. Haque, M.-S. Hacid and H. Zeineddine, "An experimental study with imbalanced classification approaches for credit card fraud detection", *IEEE Access*, vol. 7, pp. 93 010–93 022, 2019. DOI: 10.1109/ACCESS.2019.2927266, [Accessed: 10 October 2021].

[6] S. Marabad, "Credit card fraud detection using machine learning", *Asian Journal of Convergence in Technology*, vol. 7, no. 2, 2021, ISSN: 2350-1146. DOI: https://doi.org/10.33130/AJCT.2021v07i02.023, [Accessed: 27 January 2022].

[7] *Credit card fraud detection*. [Online]. Available : https://www.kaggle.com/mlg-ulb/creditcardfraud, [Accessed: 10 October 2021].

[8] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, Inc., 2019, ISBN: 9781492032649.

[9] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, 2005, ISBN: 9780387254654.

[10] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019, ISBN: 9781999579517.

[11] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016, ISBN: 9781449369415.

[12] A. Cutler, D. Cutler and J. Stevens, "Random forests", in. Jan. 2011, vol. 45, pp. 157–176, ISBN: 978-1-4419-9325-0. DOI: 10.1007/978-1-4419-9326-7_5.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available : https://scikit-learn.org/, [Accessed: 10 October 2021].

[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.", *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. DOI: 10 . 1214 / aos / 1013203451. [Online]. Available : https://doi.org/10.1214/aos/1013203451, [Accessed: 29 January 2022].

[15] C. Wade and K. Glynn, *Hands-On Gradient Boosting with XGBoost and scikit-learn*. 2020, ISBN: 9781839218354.

[16] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", *CoRR*, vol. abs/1603.02754, 2016. arXiv: 1603.02754. [Online]. Available : http://arxiv.org/abs/1603.02754, [Accessed: 10 October 2021].

[17] D.-K. Choi, "Data-driven materials modeling with xgboost algorithm and statistical inference analysis for prediction of fatigue strength of steels", *International Journal of Precision Engineering and Manufacturing*, vol. 20, pp. 129–138, 2019. DOI: 10.1007/s12541-019-00048-6. [Online]. Available : https://doi.org/10.1007/s12541-019-00048-6, [Accessed: 01 February 2022].

[18] *Fighting overfitting with l1 or l2 regularization: Which one is better?* [Online]. Available : https://neptune.ai, [Accessed: 12 February 2022].

[19] *Xgboost documentation*. [Online]. Available : https://xgboost.readthedocs.io, [Accessed: 30 January 2022].

[20] M. Swamynathan, *Mastering Machine Learning with Python in Six Steps*. 2017, ISBN: 9781484228661.

[21] *Smote*. [Online]. Available : https://docs.microsoft.com/en-us/previous-versions/azure/machine-learning/studio-module-reference/smote#expected-input, [Accessed: 29 January 2022].

[22] T. Agrawal, *Hyperparameter Optimization in Machine Learning*. 2021, ISBN: 9781484265796.

[23] T. Ma, L. Wu, S. Zhu and H. Zhu, "Multiclassification prediction of clay sensitivity using extreme gradient boosting based on imbalanced dataset",

*Applied Sciences*, vol. 12, no. 3, 2022, ISSN: 2076-3417. DOI: 10.3390/app12031143. [Online]. Available : https://www.mdpi.com.

[24] *Credit card*. [Online]. Available : https://www.investopedia.com/terms/c/creditcard.asp, [Accessed: 21 February 2021].

[25] *Pci dss quick reference guide: Understanding the payment card industry data security standard version 3.2.1*. [Online]. Available : https://www.pcisecuritystandards.org/, [Accessed: 03 March 2022].

[26] *How the payment process works*. [Online]. Available : https://sea.mastercard.com/en-region-sea/business/merchants/start-accepting/payment-process.html, [Accessed: 21 February 2021].

[27] D. K. K. Kajal, "Credit card fraud detection using imbalance resampling method with feature selection", *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, Jun. 2021. DOI: https://doi.org/10.30534/ijatcse/2021/811032021.

[28] A. Correa Bahnsen, D. Aouada, A. Stojanovic and B. Ottersten, "Feature engineering strategies for credit card fraud detection", *Expert Systems with Applications*, vol. 51, Jan. 2016. DOI: 10.1016/j.eswa.2015.12.030.