

**PENGUJIAN METODE EXTREME GRADIENT BOOSTING
UNTUK DETEKSI PENIPUAN KARTU KREDIT**

TUGAS AKHIR

Nicholas Anthony Suhartono
1118049



PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2021

PENGUJIAN METODE EXTREME GRADIENT BOOSTING UNTUK DETEKSI PENIPUAN KARTU KREDIT

TUGAS AKHIR

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

**Nicholas Anthony Suhartono
1118049**



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

HALAMAN PERNYATAAN ORISINALITAS

**Saya menyatakan bahwa Tugas Akhir yang saya susun ini
adalah hasil karya saya sendiri.**

**Semua sumber yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.**

**Saya bersedia menerima sanksi pencabutan gelar akademik
apabila di kemudian hari Tugas Akhir ini terbukti plagiat.**

Bandung, Tanggal Bulan Tahun

A handwritten signature in black ink, appearing to read 'Nicholas Anthony Suhartono', written in a cursive style.

**Nicholas Anthony Suhartono
1118049**

HALAMAN PENGESAHAN TUGAS AKHIR

Tugas Akhir dengan judul:

**PENGUJIAN METODE EXTREME GRADIENT BOOSTING UNTUK
DETEKSI PENIPUAN KARTU KREDIT**

yang disusun oleh:

Nicholas Anthony Suhartono

1118049

telah berhasil dipertahankan di hadapan Dewan Penguji Sidang Tugas Akhir yang
dilaksanakan pada:

Hari / tanggal : Hari, Tanggal Bulan Tahun

Waktu : Jam (24-HOUR FORMAT, contoh 16.00 WIB) WIB

Menyetujui

Pembimbing Utama:

Pembimbing Pendamping:

Nama Dosen

NIK

Nama Dosen

NIK

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Institut Teknologi Harapan Bangsa, saya yang bertanda tangan di bawah ini:

Nama : Nicholas Anthony Suhartono

NIM : 1118049

Program Studi : Informatika

demikian pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Institut Teknologi Harapan Bangsa **Hak Bebas Royalti Noneksklusif** (*Non-exclusive Royalty Free Rights*) atas karya ilmiah saya yang berjudul:

PENGUJIAN METODE EXTREME GRADIENT BOOSTING UNTUK
DETEKSI PENIPUAN KARTU KREDIT

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Institut Teknologi Harapan Bangsa berhak menyimpan, mengalihmediakan, mengelola dalam pangkalan data, dan memublikasikan karya ilmiah saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Bandung, Tanggal Bulan Tahun

Yang menyatakan



Nicholas Anthony Suhartono

ABSTRAK

Nama : Nama Pengarang
Program Studi : Informatika
Judul : Judul Tugas Akhir dalam Bahasa Indonesia

Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et, ex est mazim libris quaeque. Habeo brute vel id, inani volumus adolescens et mei, solet mediocrem te sit. At sonet dolore atomorum sit, tistique sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tistique voluptua. Ei utroque electram gubergren per. Laudem nonumes an vis, cum veniam eligendi liberavisse eu. Etiam graecis id mel. An quo rebum iracundia definitionem. At quo congrue graeco explicari. Cu eos wisi legimus patrioque. Cum iisque offendit ei. Ei eruditi lobortis pericula sea, te graeco salutatus sed, ne integre insolens mei. Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet inductum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei. Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et.

Kata kunci: Sonet, dolore, atomorum, tistique, sapientem.

KATA PENGANTAR

Lorem ipsum dolor sit amet, quidam dicunt blandit duo in. Cu sed dictas vidisse admodum, at qualisque scripserit est, est case salutandi ea. No quot ornatus probatus nec, movet quodsi forensibus pri ad. His esse wisi vocent et, ex est mazim libris quaeque. Habeo brute vel id, inani volumus adolescens et mei, solet mediocrem te sit. At sonet dolore atomorum sit, tibiue sapientem contentiones no vix, dolore iriure ex vix. Vim commune appetere dissentiet ne, aperiri patrioque similique sed eu, nam facilisis neglegentur ex. Qui ut tibiue voluptua. Ei utroque electram gubergren per. Laudem nonumes an vis, cum veniam eligendi liberavisse eu. Etiam graecis id mel. An quo rebum iracundia definitionem. At quo congue graeco explicari. Cu eos wisi legimus patrioque. Cum iisque offendit ei. Ei eruditi lobortis pericula sea, te graeco salutatus sed, ne integre insolens mei. Mea tale aliquam minimum te. Eu mel putant virtute, essent inermis nominavi mea no. Laoreet indoctum sea te. Te scripta fabulas duo, pro doming recusabo voluptaria at. Cu sed numquam inciderint, ei minim altera disputando cum, te nec graeco maiorum convenire. Cu mel putent rationibus dissentiet. Per vidisse scaevola oportere ei, qui solet molestie eu. Hinc diceret nominati per at, nec dico denique laboramus et. Legere regione his at, aequae decore in mei.

Bandung, Tanggal Bulan Tahun

Hormat penulis,



Nama Pengarang

DAFTAR ISI

ABSTRAK	iv
ABSTRACT	v
KATA PENGANTAR	v
DAFTAR ISI	vi
DAFTAR TABEL	ix
DAFTAR GAMBAR	x
DAFTAR ALGORITMA	xii
DAFTAR LAMPIRAN	xiii
BAB 1 PENDAHULUAN	1-1
1.1 Latar Belakang	1-1
1.2 Rumusan Masalah	1-3
1.3 Tujuan Penelitian	1-3
1.4 Batasan Masalah	1-4
1.5 Kontribusi Penelitian	1-4
1.6 Metodologi Penelitian	1-4
1.7 Sistematika Pembahasan	1-5
BAB 2 LANDASAN TEORI	2-1
2.1 Tinjauan Pustaka	2-1
2.1.1 <i>Decision Tree</i>	2-1
2.1.2 <i>Ensemble Learning</i>	2-2
2.1.3 <i>Random Forest</i>	2-3
2.1.4 <i>Gradient Boost</i>	2-5
2.1.4.1 <i>XGBoost</i>	2-9
2.1.4.2 <i>Loss Function</i>	2-13
2.1.4.3 <i>Hyperparameter</i> dalam XGBoost	2-14
2.1.5 Optimisasi pada XGBoost	2-15
2.1.5.1 <i>Algoritme Basic Exact Greedy</i>	2-15
2.1.5.2 <i>Algoritme Approximate split-finding</i>	2-15

2.1.5.3	Sparsity-aware Split Finding	2-16
2.1.5.4	<i>Parallel Learning</i>	2-17
2.1.5.5	Cache-aware Access	2-18
2.1.5.6	<i>Blocks for Out-of-core Computation</i>	2-18
2.1.6	<i>Resampling</i>	2-18
2.1.7	<i>SMOTE</i>	2-19
2.1.8	<i>Hyperparameter Tuning</i>	2-22
2.1.9	<i>K-Fold Cross-Validation</i>	2-23
2.1.10	<i>Confusion Matrix</i>	2-24
2.1.11	Pustaka Python	2-26
2.1.11.1	Pandas	2-26
2.1.11.2	Numpy	2-27
2.1.11.3	Scikit-Learn	2-29
2.1.11.4	Imbalanced-Learn	2-29
2.1.11.5	Matplotlib	2-30
2.1.11.6	Seaborn	2-30
2.2	Tinjauan Studi	2-31
2.2.1	<i>State Of The Art</i>	2-31
2.3	Tinjauan Objek	2-33
2.3.1	Kartu Kredit (<i>Credit Card</i>)	2-33
2.3.2	<i>Dataset</i> Transaksi Kartu Kredit	2-33
BAB 3	ANALISIS DAN PERANCANGAN SISTEM	3-1
3.1	Analisis Masalah	3-1
3.2	Kerangka Pemikiran	3-1
3.2.1	Penjelasan Indikator	3-2
3.3	Analisis Urutan Proses Global	3-3
3.4	Analisis Data Sampling	3-5
3.5	<i>Data Preprocessing</i>	3-5
3.6	<i>Data splitting</i>	3-6
3.7	<i>Oversampling</i> dengan SMOTE	3-6
3.8	Analisis kasus	3-6
3.8.1	Pembentukan <i>Tree</i> Pertama	3-7
3.8.2	Pembentukan <i>Tree</i> Kedua	3-17
3.8.3	Melakukan prediksi	3-24
BAB 4	IMPLEMENTASI DAN PENGUJIAN	4-1
4.1	Lingkungan Implementasi	4-1

4.1.1	Spesifikasi Perangkat Keras	4-1
4.1.2	Spesifikasi Perangkat Lunak	4-1
4.2	Implementasi Perangkat Lunak	4-1
4.2.1	Implementasi <i>Class</i>	4-1
4.2.1.1	<i>Class</i> Nama_Class_1	4-1
4.2.1.2	<i>Class</i> Nama_Class_2	4-1
4.2.2	Implementasi Numquam	4-1
4.3	Implementasi Nama_Implementasi	4-1
4.4	Implementasi Aplikasi	4-1
4.5	Pengujian	4-1
4.5.1	Pengujian Nama_Pengujian_1	4-1
4.5.2	Pengujian Nama_Pengujian_2	4-1
BAB 5	KESIMPULAN DAN SARAN	5-1
5.1	Kesimpulan	5-1
5.2	Saran	5-1
Daftar Referensi		i
BAB A	LAMPIRAN A	A-1
	ASJDBAKJSDBKA	A-1
BAB B	DATASET HASIL KUISIONER 2	B-3

DAFTAR TABEL

2.1	Daftar masukan SMOTE [23]2-21
2.2	Daftar parameter SMOTE [23]2-21
2.3	Daftar keluaran SMOTE [23]2-21
2.4	Daftar Metode yang Digunakan2-26
2.5	Daftar Metode yang Digunakan2-27
2.6	Daftar Metode yang Digunakan2-29
2.7	Daftar Metode yang Digunakan2-29
2.8	Daftar Metode yang Digunakan2-30
2.9	Daftar Metode yang Digunakan2-30
2.10	Tinjauan Studi2-31
A-1	<i>Lorem ipsum</i>	A-1
A-1	<i>Lorem ipsum</i>	A-2
B-1	<i>Lorem ipsum</i>	B-3
B-1	<i>Lorem ipsum</i>	B-4

DAFTAR GAMBAR

2.1	Ilustrasi <i>Decision Tree</i> untuk menentukan apakah kustomer akan merespons terhadap <i>email marketing</i> . <i>Root node</i> digambarkan dengan bentuk segitiga. <i>Internal node</i> digambarkan dengan bentuk persegi panjang. <i>Leaf node</i> digambarkan dengan bentuk lingkaran [10].	2-2
2.2	Alur proses pelatihan pada <i>bagging</i> dan <i>boosting</i> . <i>Bagging</i> membangun <i>weak learner</i> secara independen dan prediksi didapat dari hasil <i>voting</i> setiap <i>learner</i> . <i>Boosting</i> membangun <i>learner</i> berdasarkan <i>learner</i> sebelumnya. Prediksi didapat dari nilai bobot rata-rata setiap <i>learner</i>	2-3
2.3	Ilustrasi <i>Random Forest</i> ketika melakukan prediksi.	2-5
2.4	Algoritme <i>Gradient Boost</i> [15]	2-6
2.5	Ilustrasi Algoritme Gradient Boosting.	2-6
2.6	Algoritme XGBoost [18]	2-9
2.7	Algoritme <i>exact greedy</i> [17]	2-15
2.8	Algoritme <i>approximate split-finding</i> [17]	2-16
2.9	Struktur <i>tree</i> dengan <i>default direction</i> . Sampel akan diklasifikasikan ke dalam <i>default direction</i> ketika fitur yang dibutuhkan untuk <i>split</i> hilang. [17]	2-16
2.10	Algoritme <i>approximate split-finding</i> [17]	2-17
2.11	Ilustrasi struktur blok untuk <i>parallel learning</i> . Setiap kolom dalam satu blok di urutkan berdasarkan nilainya. <i>Linear scan</i> dilakukan untuk mendapatkan <i>split</i>	2-18
2.12	Ilustrasi dataset yang <i>imbalance</i>	2-19
2.13	Ilustrasi <i>resampling</i> [22]	2-19
2.14	Ilustrasi <i>oversampling</i> menggunakan SMOTE	2-20
2.15	Perbandingan proporsi jumlah data pada suatu kelas menggunakan SMOTE [23]	2-20
2.16	<i>k-fold cross-validation</i> dengan <i>fold</i> sebanyak 5 [14]	2-24
2.17	<i>Confusion Matrix</i> [11] pada kasus klasifikasi <i>email spam</i> sebagai kelas positif dan bukan <i>spam</i> sebagai kelas negatif.	2-25
2.18	Kelas yang tidak seimbang pada <i>dataset</i> yang akan digunakan. Kelas 1 merupakan kelas <i>fraud</i> atau transaksi penipuan dan kelas 0 merupakan kelas <i>non-fraud</i> atau transaksi bukan penipuan.	2-34

3.1	Kerangka Pemikiran	3-2
3.2	<i>Flowchart</i> Urutan Proses Global	3-3
3.3	Isi <i>Dataset</i> Transaksi Pembayaran Kartu Kredit	3-5
3.4	<i>Heatmap Correlation Matrix</i>	3-6
3.5	<i>Dataset</i> setelah dilakukan <i>oversampling</i> dengan SMOTE. Data dengan label kelas 0 dan label kelas 1 memiliki jumlah yang seimbang.	3-6
3.6	<i>Dataset</i> untuk analisis kasus	3-7
3.7	Inisialisasi nilai prediksi dan <i>probability</i> untuk setiap data latih . . .	3-7
3.8	Menghitung nilai <i>gradient</i> dan <i>hessian</i> untuk setiap data latih. . . .	3-8
3.9	Data pada fitur v1 yang telah diurutkan untuk mempermudah perhitungan. Warna merah menunjukkan data yang akan digunakan untuk perhitungan <i>gradient left</i> dan <i>hessian left</i> fitur v1 untuk data indeks ke 2 dengan kriteria ≤ -12.803688754721 . Warna biru menunjukkan data yang akan digunakan untuk perhitungan <i>gradient right</i> dan <i>hessian right</i> fitur v1 untuk data indeks ke 2 dengan kriteria > -12.803688754721	3-9
3.10	Hasil perhitungan <i>gradient left</i> (gl), <i>gradient right</i> (gr), <i>hessian left</i> (hl), <i>hessian right</i> (hr), <i>gain</i> pada fitur v1 yang telah diurutkan untuk setiap data latih pada untuk pembentukan <i>tree</i> pertama. . . .	3-12
3.11	Hasil perhitungan <i>gradient left</i> (gl), <i>gradient right</i> (gr), <i>hessian left</i> (hl), <i>hessian right</i> (hr), <i>gain</i> pada fitur v2 yang telah diurutkan untuk setiap data latih untuk pembentukan <i>tree</i> pertama.	3-14
3.12	Visualisasi <i>tree</i> pertama setelah melakukan <i>split</i> berdasarkan fitur v1 dengan kondisi $v1 \leq -25.82598215$	3-14
3.13	Visualisasi tabel pada <i>leaf node</i> kiri berdasarkan gambar 3.12 . . .	3-15
3.14	Visualisasi tabel pada <i>leaf node</i> kanan berdasarkan gambar 3.12 . .	3-15
3.15	Visualisasi penelusuran <i>tree</i> untuk mendapatkan nilai bobot yang sesuai untuk setiap data latih. Terlihat bahwa data indeks ke 0 masuk ke dalam anggota <i>leaf node</i> dengan fitur v1 ≤ -25.82598215 , dengan nilai bobot sebesar 0.51.	3-16
3.16	Hasil prediksi dan <i>probability</i> baru untuk setiap data latih	3-17
3.17	Nilai <i>gradient</i> dan <i>hessian</i> untuk <i>tree</i> kedua	3-17
3.18	Nilai <i>gradient left</i> , <i>gradient right</i> , <i>hessian left</i> , <i>hessian right</i> , <i>gain</i> fitur v1 yang telah diurutkan untuk pembentukan <i>tree</i> kedua. . . .	3-20

3.19	Hasil perhitungan <i>gradient left</i> (gl), <i>gradient right</i> (gr), <i>hessian left</i> (hl), <i>hessian right</i> (hr), dan <i>gain</i> pada fitur v2 yang telah diurutkan untuk setiap data latih untuk pembentukan <i>tree</i> kedua.	3-22
3.20	Visualisasi <i>tree</i> kedua setelah melakukan split berdasarkan fitur v1 dengan kondisi $v_1 \leq -25.82598215$	3-22
3.21	Nilai bobot untuk data indeks ke 0 adalah 0.51 pada <i>tree</i> kedua . . .	3-23
3.22	Hasil prediksi akhir	3-24

DAFTAR ALGORITMA

LAMPIRAN A	A-1
LAMPIRAN B	B-3

DAFTAR LAMPIRAN

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 sebesar 24,26 milyar dollar Amerika Serikat [1]. Kejadian ini tentunya merugikan kedua belah pihak, baik dari sisi perusahaan kartu kredit yang dapat kehilangan kepercayaan, maupun dari pihak pemilik kartu kredit yang mengalami kerugian finansial. Kerugian yang ditimbulkan akibat penipuan kartu kredit di Amerika jumlahnya sangat besar [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 [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 untuk 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 masukan 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* pertama, 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 *Artificial 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 menunjukkan 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. Berapa nilai *hyperparameter* terbaik pada algoritme klasifikasi XGBoost?
2. Berapa hasil kinerja algoritme XGBoost dan SMOTE sebagai teknik *oversampling*?

1.3 Tujuan Penelitian

Tujuan yang ingin dicapai dalam penelitian ini adalah menguji algoritme klasifikasi XGBoost dengan menggunakan *hyperparameter 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*.
3. Mengevaluasi total waktu pemrosesan algoritme klasifikasi XGBoost.

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*

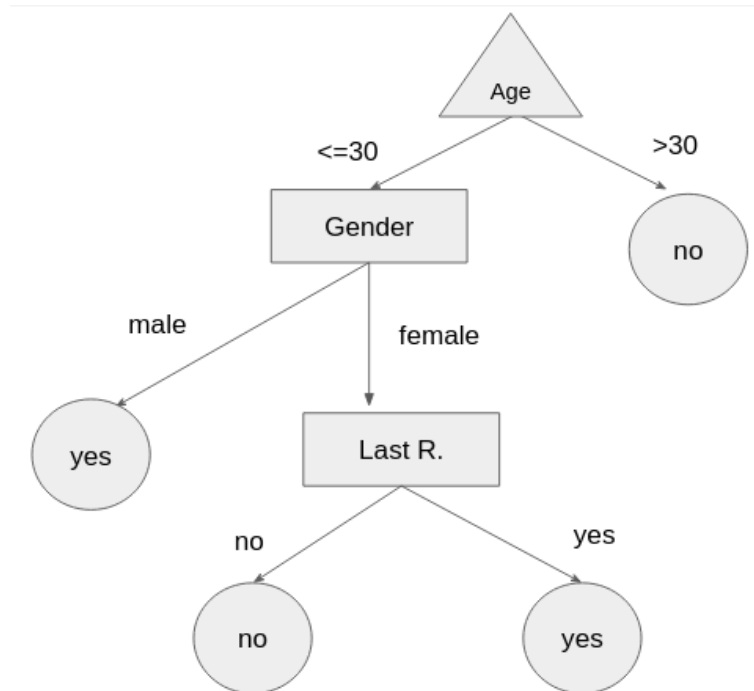
Decision Tree (DT) merupakan algoritme *machine learning* yang mampu melakukan klasifikasi dan regresi. Algoritme ini merupakan komponen fundamental untuk algoritme *Random Forest* [8], *Gradient Boost*, dan *XGBoost* [9]. DT terdiri dari simpul (*node*) yang membentuk sebuah pohon. Simpul akar (*root node*) adalah *node* yang tidak memiliki cabang (*edge*) yang masuk ke dalam, semua *Node* selain itu memiliki cabang keluar. *Node* yang memiliki cabang keluar disebut sebagai simpul dalam *internal node*. *Node* yang tidak memiliki cabang keluar disebut sebagai simpul daun *leaf node* [10]. Setiap *internal node* merepresentasikan atribut atau fitur tertentu. Sedangkan setiap *leaf node* berisi nilai dari atribut atau fitur *internal node* [10].

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*.

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

Keterangan :

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

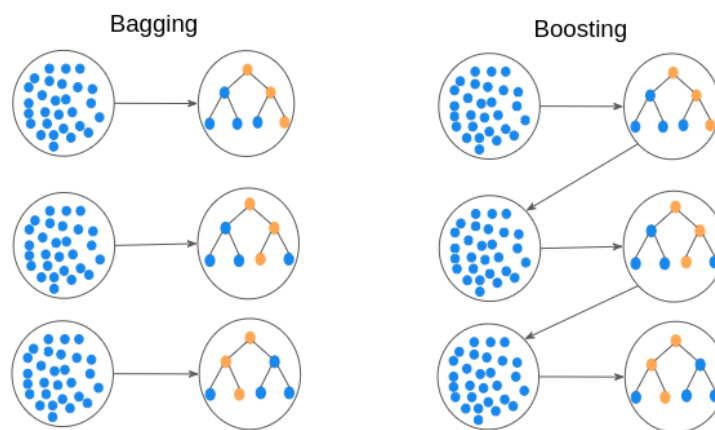


Gambar 2.1 Ilustrasi *Decision Tree* untuk menentukan apakah kustomer akan merespons terhadap *email marketing*. *Root node* digambarkan dengan bentuk segitiga. *Internal node* digambarkan dengan bentuk persegi panjang. *Leaf node* digambarkan dengan bentuk lingkaran [10].

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 membuat 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 [11]. 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* [11]. Proses pelatihan dalam *bagging* dapat berjalan secara bersamaan karena setiap *learner* yang dibangun secara independen. Dalam *bagging* hasil prediksi diperoleh dari hasil *voting* dari semua *weak learner* yang dibentuk. Salah satu contoh algoritme *machine learning* yang menggunakan *bagging* adalah *Random Forest*.



Gambar 2.2 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 [12]. *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 agregasi 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* terhadap 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.

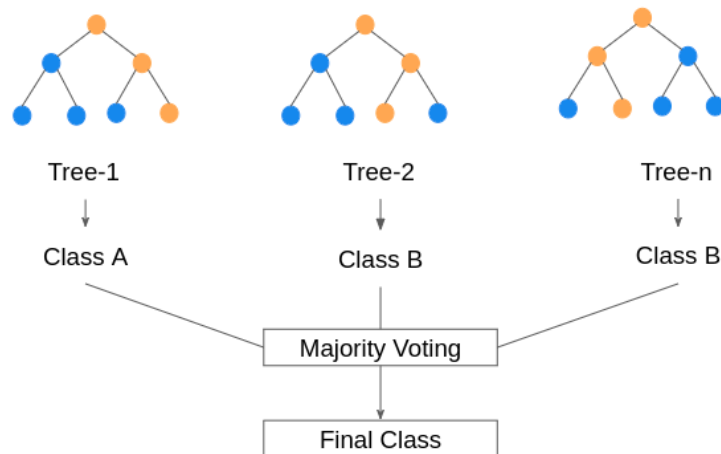
Algoritme *Random Forest* [13]:

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 bootstrap* 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.

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

Keterangan :

y	: hasil prediksi <i>Random Forest</i>
B	: jumlah tree
x	: sampel yang akan diprediksi
$f_b(x)$: hasil prediksi <i>tree</i> 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* [14]:

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.

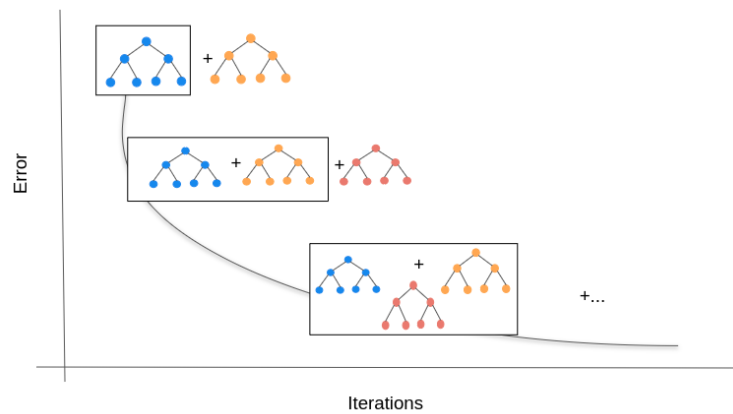
2.1.4 Gradient Boost

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 [12]. *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* [15]



Gambar 2.5 Ilustrasi Algoritme Gradient Boosting.

Algoritme Gradient Boost [15]:

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.

$$f_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^N \Psi(y_i, \gamma) \quad (2.3)$$

Keterangan :

$f_0(x)$: Nilai prediksi awal.

$\Psi(y_i, \gamma)$: *Loss function*.

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

Keterangan :

\hat{y}_{im} : Nilai residual.

N : Jumlah sampel data.

i : residual ke i .

m : *tree* ke m .

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

Keterangan :

R_{lm} : Nilai residual *terminal node*.

$$\gamma_{lm} = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{lm}} \Psi(y_i, F_{m-1}(x_i) + \gamma) \quad (2.6)$$

Keterangan :

γ_{lm} : Nilai gamma pada *leaf node l tree m*.

$$F_m(x) = F_{m-1}(x) + v \cdot \gamma_{lm} 1(x \in R_{lm}) \quad (2.7)$$

Keterangan :
 $F_m(x)$: Nilai prediksi baru untuk sampel data x.
 v *: *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* [14]:

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 [16]. Konsistensi dan hasil yang bagus dari *gradient boost*, membuat Tianqi Chen dari Universitas Washington untuk menciptakan algoritme dan system yang dinamakan *XGBoost* [16]. 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 [17].

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 $k = 1, 2, \dots, M$ **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 $\hat{b}(x) = \sum_{j=1}^T w I_j$;

 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 [18]

BAB 2 LANDASAN TEORI

Keterangan :

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

$$Gradient = p - y_i \quad (2.8)$$

Keterangan :

y_i	: Nilai label pada ke i .
p	: Nilai <i>probability</i> .

$$Hessian = p(1 - p) \quad (2.9)$$

Keterangan :

p	: Nilai <i>probability</i> .
-----	------------------------------

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 $\min_{child_w eight}$, 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.

$$f_0(x) = 0.5 \quad (2.10)$$

Keterangan :

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

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

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.

λ : Konstanta regularisasi.

γ : Konstanta pengurang nilai *gain*.

$$w_j = -\frac{G_j}{H_j + \lambda} \quad (2.12)$$

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*.

λ : konstanta regularisasi.

$$\hat{b}(x) = \eta \sum_{j=1}^T w_j I \quad (2.13)$$

Keterangan :

η : *Learning rate*.

T : Jumlah *leaf node*.

w : Nilai *weight*.

I : Set input pada masukkan data x

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

Keterangan :

$f_k(x)$: Nilai prediksi *tree k*.

$\hat{b}(x)$: Nilai weight yang didapat dengan menelusuri *tree*.

η : Learning rate.

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

Keterangan :

$f(x)$: Nilai *output* pada algoritme XGBoost.

M : Jumlah *tree* yang telah ditentukan.

$f_k(x)$: Nilai *output* setiap *tree k*.

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

Keterangan :

$f(x)$: Nilai prediksi akhir.

$f_0(x)$: Nilai prediksi awal.

M : Jumlah *tree*.

η : *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. Pada kasus deteksi transaksi kartu kredit, *loss function* yang digunakan adalah fungsi *sigmoid* karena masalah tersebut merupakan klasifikasi biner yang hanya memprediksi 2 kelas *output*, yaitu apakah transaksi merupakan penipuan (*fraud*) atau bukan penipuan (*non-fraud*).

$$Sigmoid = \frac{1}{1 + e^{-t}} \quad (2.17)$$

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. η : *Learning rate* merupakan besaran nilai yang digunakan untuk menyusutkan nilai bobot untuk mencegah *overfitting*. Dalam XGBoost, *learning rate* dilambangkan sebagai η sesuai persamaan 2.13.
2. λ : 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 [19] [12]. Regularisasi L2 atau yang biasa disebut *L2 norm* atau Ridge menangani *overfitting* dengan membuat nilai bobot menjadi lebih kecil. XGBoost menggunakan regularisasi L2 lambda (λ) sesuai persamaan 2.11 dan 2.12.
3. γ : Gamma merupakan nilai minimum pengurangan *loss* yang digunakan untuk membuat partisi *node* pada *tree*. Nilai γ 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 [20].

2.1.5 Optimisasi pada 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* [17]. *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  $sorted(I, \text{by } \mathbf{x}_{jk})$  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* [17]

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 [20].

```

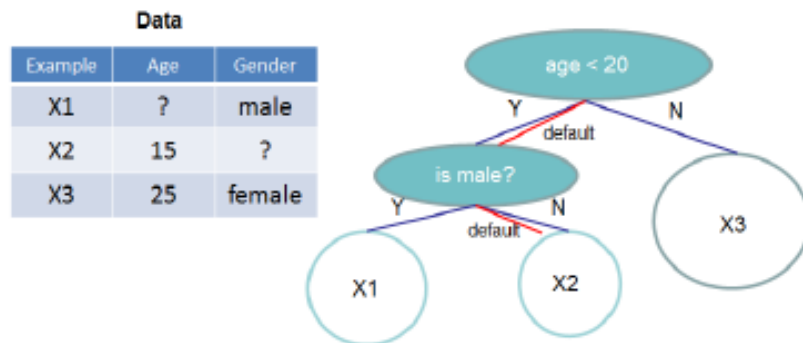
for  $k = 1$  to  $m$  do
  | Propose  $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$  by percentiles on feature  $k$ .
  | Proposal can be done per tree (global), or per split(local).
end
for  $k = 1$  to  $m$  do
  |  $G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g_j$ 
  |  $H_{kv}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

```

Gambar 2.8 Algoritme *approximate split-finding* [17]

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.



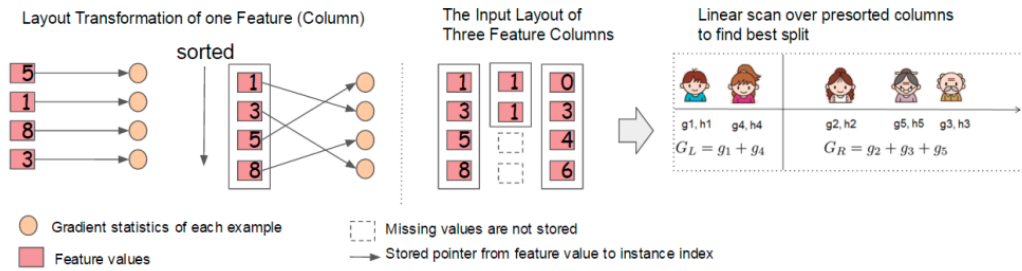
Gambar 2.9 Struktur *tree* dengan *default direction*. Sampel akan diklasifikasikan ke dalam *default direction* ketika fitur yang dibutuhkan untuk *split* hilang. [17]

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* [17]

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 [17].



Gambar 2.11 Ilustrasi struktur blok untuk *parallel learning*. Setiap kolom dalam satu blok di urutkan berdasarkan nilainya. *Linear scan* dilakukan untuk mendapatkan *split*.

2.1.5.5 Cache-aware Access

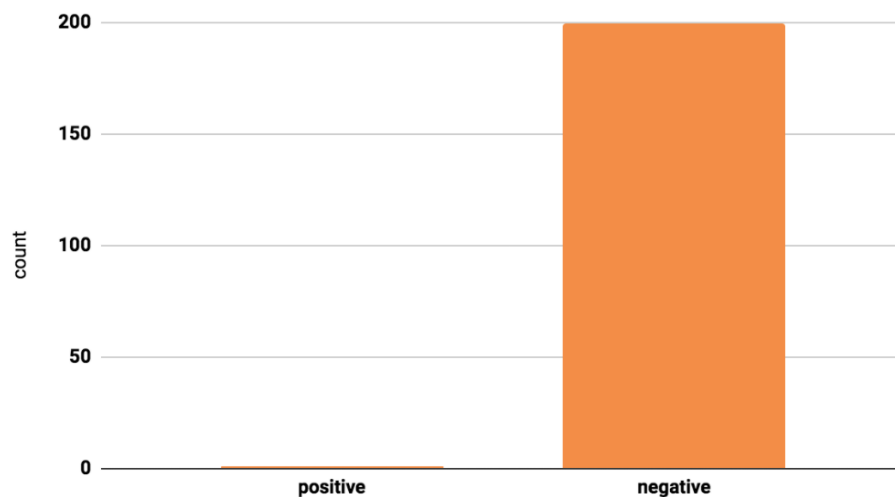
XGboost mengalokasikan *internal buffer* pada setiap *thread* dalam komputer, lalu menyimpan statistik hasil perhitungan gradient termasuk nilai dan arahnya untuk setiap *split node*, dan melakukan akumulasi secara *mini-batch* [16]. Proses ini mengurangi waktu untuk melakukan operasi *read/write* [17] 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 [17], sehingga mengurangi waktu yang digunakan untuk membaca data dari *disk* [16].

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 memiliki 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 memiliki 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* [21]. Terdapat beberapa teknik dalam melakukan *resampling*, yaitu:

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



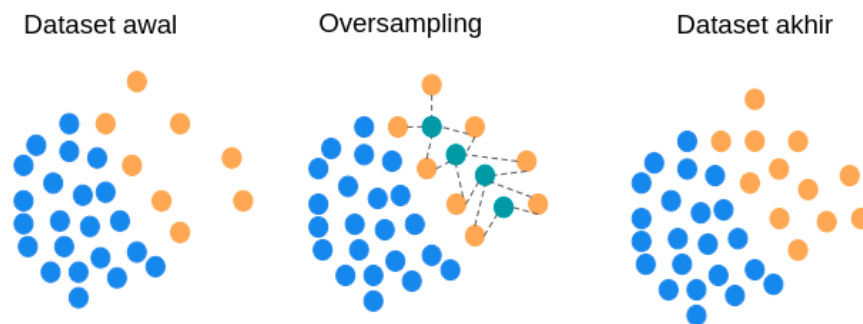
Gambar 2.13 Ilustrasi *resampling* [22]

2.1.7 SMOTE

Synthetic Minority Oversampling Technique (SMOTE), merupakan salah satu teknik *statistikoversampling* 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.

SMOTE menerima seluruh *dataset* sebagai masukan, tetapi meningkatkan persentase jumlah data dari data kelas minoritas. Sebagai contoh, bila *dataset* masukan 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 persentase jumlah data sebelumnya, SMOTE harus menerima parameter persentase 200.



Gambar 2.14 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 2.15 Perbandingan proporsi jumlah data pada suatu kelas menggunakan *SMOTE* [23]

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 [23]. Berikut merupakan daftar masukan, *parameter* dan keluaran SMOTE .

Tabel 2.1 Daftar masukan SMOTE [23]

Nama	<i>Tipe data</i>	Deskripsi
Sampel	Data tabel	Sampel data yang akan digunakan

Tabel 2.2 Daftar parameter SMOTE [23]

Nama	Range	Tipe Data	Default	Deskripsi
Presentase SMOTE	≥ 0	Integer	100	<i>dataset oversampling</i> Jumlah dalam kelipatan 100
Jumlah tetangga terdekat	≥ 1	Integer	1	Jumlah tetangga terdekat untuk menentukan fitur untuk data sampel terbaru yang akan dibuat oleh SMOTE.
<i>Random seed</i>	any	Integer	0	<i>Seed</i> untuk melakukan generasi angka acak.

Tabel 2.3 Daftar keluaran SMOTE [23]

Nama	<i>Tipe data</i>	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 $(presentase\ SMOTE \div 100 \times T)$, dengan T merupakan jumlah sampel kelas minoritas.
-------	------------	---

2.1.8 *Hyperparameter Tuning*

Hyperparameter tuning adalah proses eksperimental dengan tujuan mencari kombinasi nilai *hyperparameter* terbaik untuk algoritme pembelajaran yang digunakan, umumnya proses ini dilakukan oleh *data analyst*. *Hyperparameter* adalah konfigurasi eksternal yang dilakukan pada parameter model, dimana nilai parameter dimasukkan secara manual dan digunakan dalam proses trial-error untuk menemukan nilai terbaik untuk parameter. Ada beberapa strategi yang dapat diterapkan dalam proses *hyperparameter tuning*, yaitu *grid search*, *random search*, dan *Bayesian hyperparameter optimization* [11].

Grid search merupakan strategi *hyperparameter tuning* yang paling mudah untuk diterapkan. *Grid search* mencoba semua kombinasi dari *hyperparameter* kemudian menggunakan data latih untuk melatih model dan menilai performa model menggunakan data validasi untuk setiap kombinasi *hyperparameter*. Sebagai contoh, apabila terdapat *hyperparameter* $\lambda = [0.01, 0.1]$ dan $\epsilon = [0.5, 0.6]$, maka *grid search* akan mencoba semua kombinasi λ dan ϵ : $[0.01, 0.5]$, $[0.01, 0.6]$, $[0.1, 0.5]$, dan $[0.1, 0.6]$ untuk melatih dan menilai performa model. Kelebihan dari *grid search* adalah mencoba setiap kombinasi dari *hyperparameter* sehingga 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 [11].

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 [24].

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 [21].

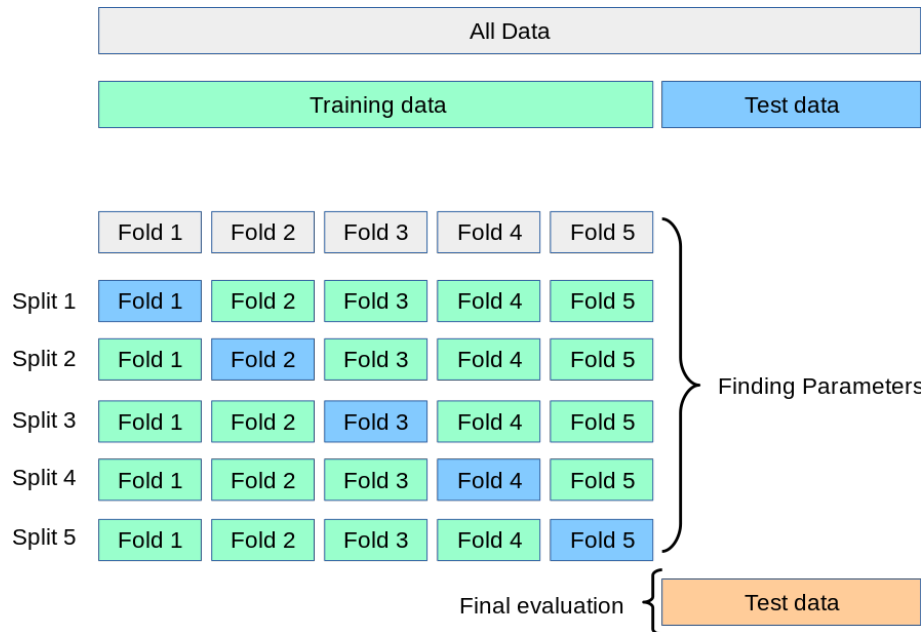
Berbeda dengan *grid search*, *random search* tidak membuat kombinasi nilai *hyperparameter*, melainkan menyediakan distribusi statistik untuk setiap *hyperparameter*, dimana nilai-nilainya dipilih secara acak dan menetapkan seberapa banyak kombinasi yang ingin diuji [6]. Salah satu kelebihan *random search* dibandingkan *grid search* adalah dari segi waktu pemrosesannya yang lebih cepat, karena pemilihan *hyperparameter*-nya yang acak. Selain itu, *random search* juga mampu mencapai akurasi yang lebih tinggi dari *grid search*, dengan jumlah kasus uji yang cukup. Namun jika dibandingkan dengan metode *Bayesian*, *grid search* dan *random search* masih kurang mampu mencapai hasil yang maksimal karena metodenya yang kurang intuitif dan acak.

Lebih berbeda lagi dengan *grid search* dan *random search*, *Bayesian hyperparameter optimization* menggunakan hasil evaluasi sebelumnya untuk memilih nilai berikutnya yang akan diuji, sehingga dapat mengurangi waktu pemrosesan dengan memilih nilai-nilai *hyperparameter* yang dianggap baik pada evaluasi sebelumnya [11].

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* [14]. 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 2.16 *k-fold cross-validation* dengan *fold* sebanyak 5 [14]

Berikut adalah beberapa langkah melakukan *Cross-validation* [11]:

1. Menentukan nilai dari *hyperparameter* yang akan diuji.
2. Memisahkan data latih menjadi beberapa *subset* data dengan ukuran yang sama. Setiap *subset* data disebut *fold*. Umumnya, jumlah *fold* adalah 5.
3. Memisahkan sebagian *fold* sebagai data uji, dan satukan *fold* sisanya sebagai data latih.
4. Mengulangi langkah ketiga hingga semua *fold* telah digunakan sebagai data uji (total proses yang dijalankan adalah sebanyak jumlah *fold*).
5. Untuk mendapatkan hasil akhir evaluasi model yang digunakan, hitung rata-rata hasil evaluasi dari hasil seluruh proses yang dijalankan.

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.

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

Gambar 2.17 *Confusion Matrix* [11] pada kasus klasifikasi *email spam* sebagai kelas positif dan bukan *spam* sebagai kelas negatif.

Berikut adalah penjelasan dari gambar 2.17:

1. *True Positive* (TP) adalah keadaan ketika diprediksi suatu kelas tergolong ke dalam kelas positif (*yes*) dan memang benar kelas itu tergolong ke dalam kelas positif. Pada gambar 2.17, terdapat 23 sampel data yang diprediksi dengan benar sebagai spam.
2. *True Negative* (TN) adalah keadaan ketika diprediksi suatu kelas tergolong ke dalam kelas negatif (*no*) dan memang benar kelas itu tergolong ke dalam kelas negatif. Pada gambar 2.17 terdapat 556 sampel data yang diprediksi dengan benar sebagai *email* bukan *spam*.
3. *False Positive* (FP) adalah keadaan ketika diprediksi suatu kelas tergolong ke dalam kelas positif, namun ternyata kelas tersebut tergolong ke dalam kelas negatif. Pada gambar 2.17 terdapat 12 sampel data yang salah diprediksi sebagai *email spam*.
4. *False Negative* (FN) adalah keadaan ketika diprediksi suatu kelas tergolong ke dalam kelas negatif, namun ternyata kelas tersebut tergolong ke dalam kelas positif. Pada gambar 2.17 terdapat 1 sampel data yang salah diprediksi sebagai *email* bukan *spam*.

Untuk menghitung TP rate yang dihasilkan, atau yang biasa disebut precision, dapat digunakan persamaan 2.18.

$$Precision = \frac{TP}{TP + FP} \quad (2.18)$$

Sementara untuk menghitung sensitivitas atau yang biasa disebut recall dapat menggunakan persamaan 2.19.

$$Recall = \frac{TP}{TP + FN} \quad (2.19)$$

Untuk menghitung akurasi dari metode yang digunakan, dapat menggunakan persamaan 2.20.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.20)$$

Untuk menghitung F-measure atau F1-score dari metode yang digunakan, dapat menggunakan persamaan 2.21

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (2.21)$$

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.

Tabel 2.4 – Daftar Metode yang Digunakan (lanjutan)

No	Metode	Masukan	Luaran	Keterangan
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: <i>int</i>	DataFrame	Menghapus kolom atau label tertentu dari DataFrame.
7	uplicated	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: <i>array</i>	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 array masukan.

Tabel 2.5 – Daftar Metode yang Digunakan (lanjutan)

No	Metode	Masukan	Luaran	Keterangan
3	<code>random_permutation</code>	x: int atau array	ndarray	Melakukan permutasi secara acak pada int atau array.
4	<code>sum</code>	a: array	ndarray	Menghitung penjumlahan nilai elemen pada array.
5	<code>quantile</code>	a: array, q: array <i>float</i> , axis: int atau tuple, interpolation: string	series atau DataFrame	Menghitung nilai kuantil data pada sumbu tertentu .
6	<code>array</code>	object: array	ndarray	Membuat array dengan tipe data ndarray.
7	<code>full</code>	shape: int atau tuple, fill _{value} : int	ndarray	Membuat array baru dengan dimensi dan nilai sesuai masukan.
8	<code>exp</code>	x: array	ndarray	Menghitung nilai eksponensial.
9	<code>log</code>	x: array	ndarray	Menghitung nilai logaritma natural.
10	<code>where</code>	condition: array atau boolean, x: array, y: array	ndarray	Mendapatkan element dari masukan x dan y berdasarkan kondisi tertentu.
11	<code>concatenate</code>	a1, a2, ...: array, axis: int	ndarray	Menggabungkan array berdasarkan sumbu tertentu.
12	<code>zeros</code>	shape: int atau tuple	ndarray	Fungsi untuk mendapatkan array baru yang berisi nilai 0.
13	<code>arange</code>	start: int, step: int, stop: int	ndarray	Fungsi untuk mendapatkan array baru dengna nilai sesuai masukan <i>start</i> dan <i>stop</i> dan memiliki interval antar nilai sebesar masukan <i>step</i> .

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	<code>train_test_split</code>	arrays: array, test_size : float, int, random_state : int	array	Memisahkan <i>dataset</i> menjadi data uji dan data latih.
2	QuantileTransformer	output_distribution: string	Objek scaler	Melakukan transformasi data menjadi distribusi normal.
3	<code>confusion_matrix</code>	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.
4	<code>accuracy_score</code>	y_{true} : array, y_{pred} : array	float	Fungsi yang digunakan untuk menghitung akurasi dari model klasifikasi <i>machine learning</i> 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 <i>oversampling</i> dengan metode <i>SMOTE</i> .

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	figsize: tuple	Figure	Membuat figur baru atau mengaktifkan 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 antar kolom menggunakan <i>heatmap</i> .

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 2.10 diberikan penjelasan mengenai perbandingan dari beberapa penelitian terkait deteksi transaksi penipuan kartu kredit:

Tabel 2.10 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 2. K-Nearest Neighbors (KNN) 3. Decision Tree 4. XGBoost 5. Random Forest 6. SMOTE sebagai metode <i>oversampling</i>	Penelitian menggunakan 89% data latih dan 20% data dan menyeimbangkan kelas yang <i>imbalance</i> menggunakan SMOTE. Akurasi dari KNN, <i>Decision Tree</i> , <i>Logistic Regression</i> , <i>Random Forest</i> , dan XGBoost sangat baik mencapai 99%. Jika dilihat berdasarkan evaluasi F1 score, XGBoost meraih hasil tertinggi sebesar 84%. Berdasarkan evaluasi <i>precision</i> , Random Forest meraih hasil tertinggi sebesar 90%. Berdasarkan evaluasi <i>recall</i> , XGBoost meraih hasil tertinggi sebesar 84%. Untuk masalah transaksi kartu kredit sangatlah penting, sehingga XGBoost merupakan model terbaik pada penelitian ini.

Tabel 2.10 – Tinjauan Studi (lanjutan)

Jurnal	Metode	Hasil Penelitian
Tao Ma et al. "Multiclassification Prediction of Clay Sensitivity Using Extreme Gradient Boosting Based on Imbalanced Dataset" in Journal of Applied Sciences, 2022	<ol style="list-style-type: none"> 1. XGBoost 2. Artificial Neural Network (ANN) 3. Naive Bayes (NB) 4. SMOTE 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 <i>oversampling</i> . Metode <i>cross-validation</i> juga diterapkan dalam proses pelatihan. Evaluasi dilakukan terhadap model XGBoost, ANN, NB, dan XGBoost tanpa SMOTE. Hasil terbaik diperoleh oleh XGBoost dengan SMOTE dengan <i>precision</i> sebesar 73%, <i>recall</i> 72%, dan <i>f1-score</i> 72% [25].
Altyeb Altaher Taha, Sharaf Jameel Malebary. "An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine". In: IEEE Access 8 (2020), pp. 25579–25587.	<ol style="list-style-type: none"> 1. Bayesian-based hyperparameter optimization 2. 5-fold Cross Validation 3. LightGBM 	Algoritma LightGBM yang sudah dioptimasi menghasilkan akurasi yang terbaik dari 2 dataset yang diuji dibandingkan dengan algoritma <i>machine learning</i> lainnya dalam mendeteksi penipuan kartu kredit. Pada dataset pertama, LightGBM memperoleh akurasi 98,40%, <i>recall</i> 40,59%, <i>precision</i> 97,34% dan <i>f1-score</i> 56,95%. Pada dataset kedua, LightGBM memperoleh akurasi sebesar 98,35%, <i>recall</i> 28,33%, <i>precision</i> 91,72%, dan <i>f1-score</i> 43,27%. Berdasarkan hasil penelitian ini, LightGBM masih memiliki kekurangan dimana nilai <i>recall</i> yang rendah. <i>Hyperparameter</i> yang di- <i>tuning</i> dalam penelitian tersebut adalah <i>num_leaves</i> , <i>max_depth</i> , dan <i>learning_rate</i> .

2.3 Tinjauan Objek

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

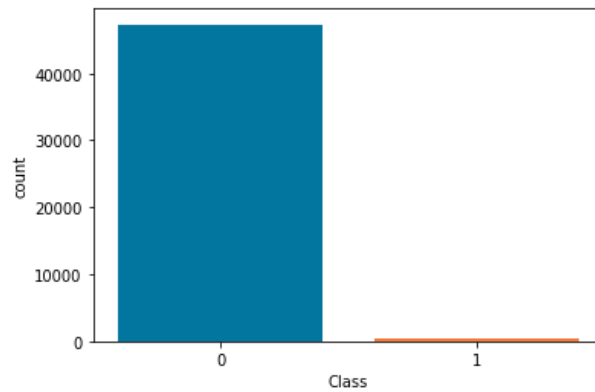
2.3.1 Kartu Kredit (*Credit Card*)

Dalam KBBI, kartu kredit adalah kartu kecil yang dikeluarkan oleh bank yang menjamin pemegangnya untuk dapat berbelanja tanpa membayar kontan dan pengeluaran belanja itu akan diperhitungkan dalam rekening pemilik kartu di bank tersebut. Kartu kredit juga dapat digunakan dalam transaksi pembayaran digital.

2.3.2 *Dataset* Transaksi Kartu Kredit

Dataset transaksi kartu kredit diperoleh dari pemegang kartu kredit di Eropa yang berisi transaksi selama 2 menggunakan kartu kredit selama 2 hari pada bulan September tahun 2013 yang dilakukan oleh pemegang kartu 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 *imbalanced* 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 proses PCA, nama fitur disamarkan dengan tujuan 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 2.18 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 2.18 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

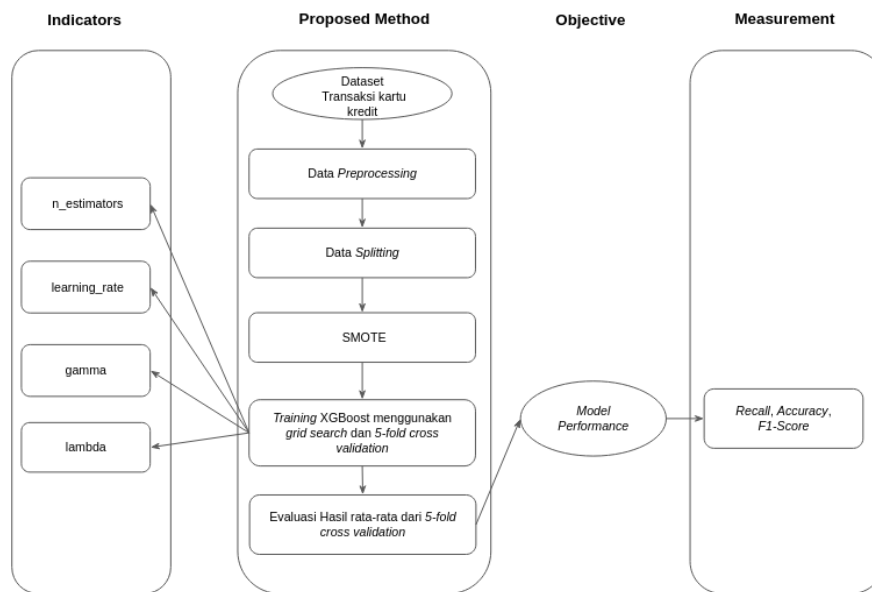
Pada Bab 1 telah dijelaskan masalah yang dihadapi terkait deteksi transaksi kartu kredit. Kemudian dijelaskan juga bahwa metode yang akurat dalam mendeteksi penipuan kartu kredit dibutuhkan untuk mencegah terjadinya kerugian yang besar akibat penipuan tersebut. Pada penelitian ini, metode klasifikasi *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, memilih fitur yang akan digunakan serta melakukan *scaling* pada data.
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* (η), *gamma* (γ), dan *lambda* (λ).

3.2.1 Penjelasan Indikator

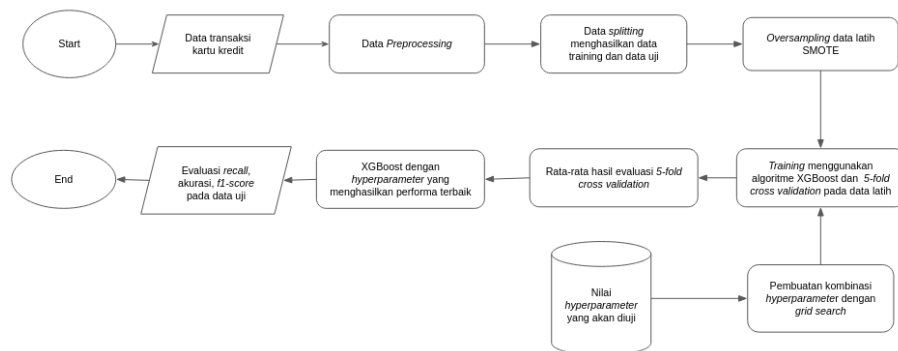
Pada bagian ini akan dijelaskan mengenai indikator *hyperparameter* yang digunakan pada XGBoost. Berikut ini merupakan penjelasan indikator yang akan digunakan pada XGBoost:

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

2. *Learning rate* (η): 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.
3. *gamma* (γ): Merupakan konstanta pengurang pada perhitungan yang mengurangi nilai *gain*. Apabila nilai *gain* < 0 maka *branch* tidak akan dikembangkan.
4. *lambda* (λ): Merupakan konstanta regularisasi pada XGBoost.

3.3 Analisis Urutan Proses Global

Dalam sistem deteksi transaksi penipuan kartu kredit terdapat beberapa 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 Urutan 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 untuk standarisasi data.
3. Tahap data *preprocessing* menghilangkan *missing value* pada data dan menghilangkan data yang duplikat, serta melakukan scaling pada data.

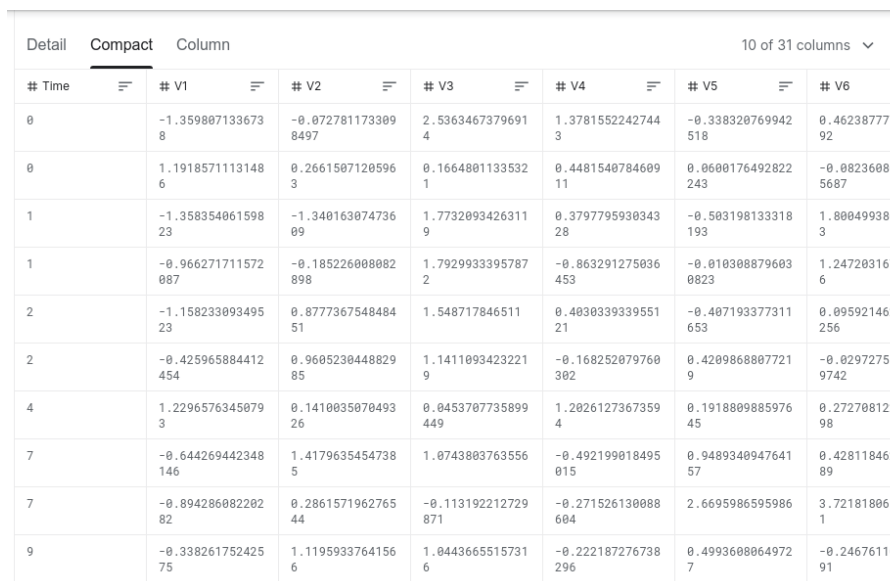
4. Data *splitting* membagi data menjadi data latih dan data uji. Data uji akan digunakan sebagai data *hold out* yang berarti data uji akan digunakan sebagai evaluasi terakhir dan tidak akan digunakan dalam proses training dan *hyperparameter* tuning seperti pada gambar 2.16.
5. Menerapkan teknik *oversampling* 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.

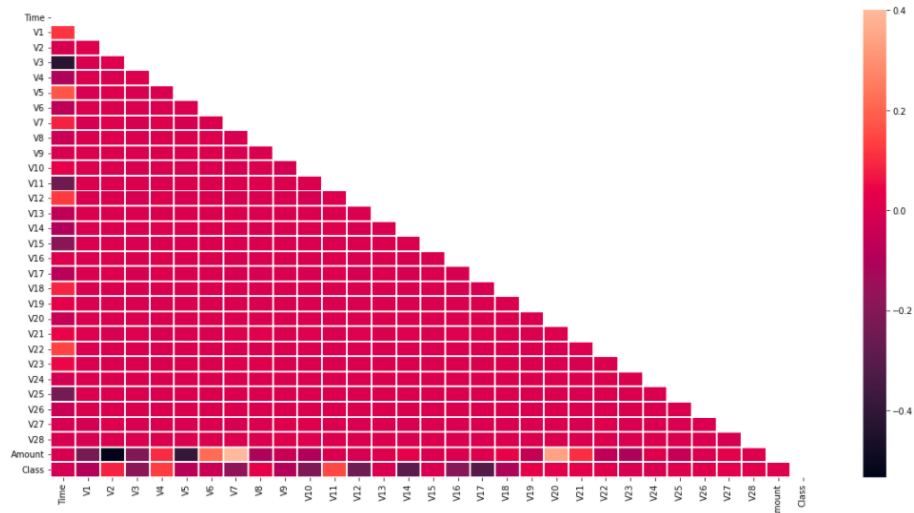


# Time	# V1	# V2	# V3	# V4	# V5	# V6
0	-1.3598871336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.46238777792
0	1.19185711131486	0.26615071205963	0.16648811335321	0.448154078460911	0.0600176492822243	-0.082360805687
1	-1.35835406159823	-1.34016307473609	1.77320934263119	0.379779593034328	-0.503198133318193	1.8004993803
1	-0.966271711572087	-0.18522608082898	1.79299333957872	-0.863291275036453	-0.0103088796030823	1.2472031676
2	-1.15823309349523	0.877736754848451	1.548717846511	0.403033933955121	-0.407193377311653	0.095921462256
2	-0.425965884412454	0.960523044882985	1.14110934232219	-0.168252079760302	0.42098688077219	-0.029727519742
4	1.22965763450793	0.141003507049326	0.0453707735899449	1.20261273673594	0.191880988597645	0.27270812298
7	-0.644269442348146	1.41796354547385	1.0743803763556	-0.492199018495015	0.948934094764157	0.42811846289
7	-0.89428608220282	0.286157196276544	-0.113192212729871	-0.271526130088604	2.66959865959867	3.7218180611
9	-0.33826175242575	1.11959337641566	1.04436655157316	-0.222187276738296	0.49936080649727	-0.2467611691

Gambar 3.3 Isi 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 *feature selection*, berdasarkan penelitian [6], semua fitur dalam dataset digunakan berdasarkan matriks korelasi. Oleh karena itu, dalam penelitian ini, semua fitur dalam *dataset* akan digunakan.



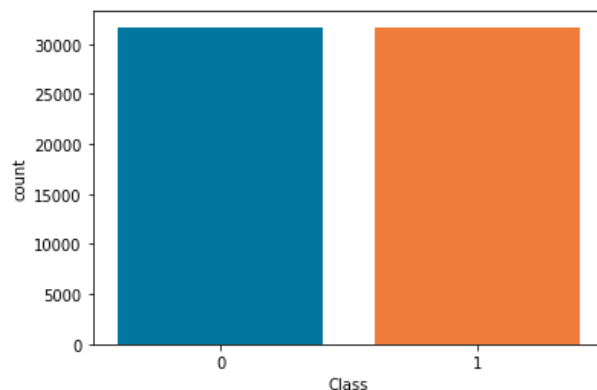
Gambar 3.4 Heatmap Correlation Matrix

3.6 Data splitting

Pada tahap ini akan dilakukan pembagian *dataset* untuk proses pelatihan dan proses pengujian. *Dataset* akan dibagi menjadi 33% data uji dan sisanya merupakan data latih.

3.7 Oversampling dengan SMOTE

Berdasarkan gambar 2.18, terlihat bahwa *dataset* sangat *imbalanced* dengan rasio kelas *fraud* sebesar 1% dari *dataset*. Pada tahap ini dilakukan *oversampling* untuk menyelesaikan masalah *dataset* yang *imbalanced* menggunakan SMOTE. Setelah dilakukan *oversampling*, dapat dilihat pada gambar 3.5 bahwa proporsi data dengan label *fraud* dan *non-fraud* sudah seimbang.



Gambar 3.5 *Dataset* setelah dilakukan *oversampling* dengan SMOTE. Data dengan label kelas 0 dan label kelas 1 memiliki jumlah yang seimbang.

3.8 Analisis kasus

Pada bagian ini dilakukan analisis tahapan proses dengan melakukan perhitungan manual. Untuk analisis kasus, *hyperparameter* regularisasi yang akan

digunakan adalah λ sebesar 1, kemudian nilai γ sebesar 0 dan *min_child_weight* sebesar 0 untuk mencegah adanya pemberhentian pada saat membangun *tree*. Sedangkan *hyperparameter* untuk η (*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' [20], 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	-30.55238004	16.71338924	1

Gambar 3.6 *Dataset* untuk analisis kasus

3.8.1 Pembentukan *Tree* Pertama

Pada contoh kasus, data dengan index 0-3 akan digunakan sebagai data latih, sedangkan data dengan index 4 akan digunakan sebagai data yang akan diuji. Langkah pertama adalah membuat prediksi inisial. XGboost menggunakan nilai 0.5 sebagai prediksi awal [20] 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 2.17.

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

Gambar 3.7 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	Initial Prediction	Gradient	Hessian
0	-30.55238004	16.71338924	1	0	0.50	-0.50	0.25
1	-25.82598215	19.16723901	1	0	0.50	-0.50	0.25
2	-12.80368875	9.666883426	0	0	0.50	0.50	0.25
3	-9.587359122	8.688022993	0	0	0.50	0.50	0.25

Gambar 3.8 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 menentukan *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* \leq -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.

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

Gambar 3.9 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 \leq -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.

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.

$$gl_0 = -0.38$$

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

$$gl_1 = -0.38 - 0.38 = -0.76$$

$$gr_1 = 0.62 + 0.62 = 1.24$$

$$gl_3 = -0.38 - 0.38 + 0.62 + 0.62 = 0.48$$

$$gr_3 = 0$$

$$hl_0 = 0.24$$

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

$$hl_1 = 0.24 + 0.24 = 0.48$$

$$hr_1 = 0.24 + 0.24 = 0.48$$

$$hl_3 = 0.24 + 0.24 + 0.24 + 0.24 = 0.96$$

$$hr_3 = 0$$

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

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

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

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

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

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

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

Gambar 3.10 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.

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

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

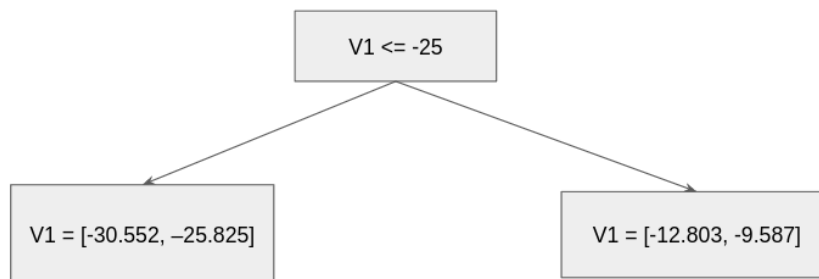
$$gain_2 = \frac{1}{2} \left[\frac{1.24^2}{0.48+1} + \frac{-0.76^2}{0.48+1} - \frac{(1.24+-0.76)^2}{0.48+0.48+1} \right] - 0 = 0.66$$

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

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

Gambar 3.11 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 $v1 \leq -25.82598215$, sedangkan pada fitur v2 diperoleh nilai *gain* tertinggi sebesar 0.66 pada kondisi ketika $v2 \leq 9.666883426$. Oleh karena fitur v1 merupakan fitur pertama pada *dataset*, maka fitur v1 akan menjadi *split* pertama.



Gambar 3.12 Visualisasi *tree* pertama setelah melakukan *split* berdasarkan fitur v1 dengan kondisi $v1 \leq -25.82598215$.

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

Gambar 3.13 Visualisasi tabel pada *leaf node* kiri berdasarkan gambar 3.12

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

Gambar 3.14 Visualisasi tabel pada *leaf node* kanan berdasarkan gambar 3.12

Setelah melakukan *split* berdasarkan fitur v1, algoritme harus perlu melakukan *split* kembali hingga mencapai kedalaman tertentu, atau hingga mendapatkan node yang *pure*. Berdasarkan gambar 3.13, *leaf node* kiri telah berisikan semua data latih dengan kelas 1, sedangkan pada gambar 3.14, *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 $v1 \leq -25.82598215$ sesuai dengan gambar 3.13 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 gambar 3.14 pada *tree* pertama.

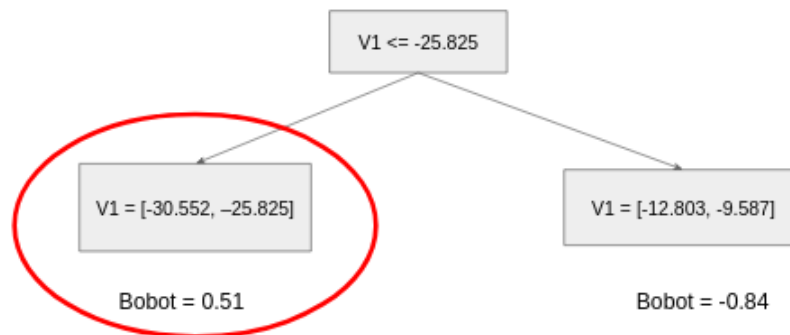
$$G = 0.62 + 0.62 = 1.24$$

$$H = 0.24 + 0.24 = 0.48$$

$$\lambda = 1$$

$$w = -\frac{1.24}{0.48+1} = -0.84$$

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 3.12. 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 3.15 Visualisasi penelusuran *tree* untuk mendapatkan nilai bobot yang sesuai untuk setiap data latih. Terlihat bahwa data indeks ke 0 masuk ke dalam anggota *leaf node* dengan fitur $v1 \leq -25.82598215$, dengan nilai bobot sebesar 0.51.

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.

$$NewPrediction_0 = 0.5 + 0.4 \times 0.51 = 0.71$$

$$NewPrediction_1 = 0.5 + 0.4 \times 0.51 = 0.71$$

$$NewPrediction_2 = 0.5 + 0.4 \times -0.84 = 0.16$$

$$NewPrediction_3 = 0.5 + 0.4 \times -0.84 = 0.16$$

$$probability_0 = \frac{1}{1+e^{-0.71}} = 0.67$$

$$probability_1 = \frac{1}{1+e^{-0.71}} = 0.67$$

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

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

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

Gambar 3.16 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.

v1	v2	Class	New Prediction	Probability	New Gradient	New Hessian
-30.55238004	16.71338924	1	0.7054054054	0.67	-0.33	0.22
-25.82598215	19.16723901	1	0.7054054054	0.67	-0.33	0.22
-12.80368875	9.666883426	0	0.1648648649	0.54	0.54	0.25
-9.587359122	8.688022993	0	0.1648648649	0.54	0.54	0.25

Gambar 3.17 Nilai *gradient* dan *hessian* untuk *tree* kedua .

$$\text{Gradient}_0 = 0.67 - 1 = -0.33$$

$$\text{Gradient}_1 = 0.67 - 1 = -0.33$$

$$\text{Gradient}_2 = 0.54 - 0 = 0.54$$

$$\text{Gradient}_3 = 0.54 - 0 = 0.54$$

$$\text{Hessian}_0 = 0.67(1 - 0.67) = 0.22$$

$$\text{Hessian}_1 = 0.67(1 - 0.67) = 0.22$$

$$\text{Hessian}_2 = 0.54(1 - 0.54) = 0.22$$

$$\text{Hessian}_3 = 0.54(1 - 0.54) = 0.22$$

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.

$$gl_0 = -0.33$$

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

$$gl_1 = -0.33 - 0.33 = -0.66$$

$$gr_1 = 0.54 + 0.54 = 1.08$$

$$gl_2 = -0.33 - 0.33 + 0.54 = -0.12$$

$$gr_2 = 0.54$$

$$gl_3 = -0.33 - 0.33 + 0.54 + 0.54 = 0.42$$

$$gr_3 = 0$$

$$hl_0 = 0.22$$

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

$$hl_1 = 0.22 + 0.22 = 0.44$$

$$hr_1 = 0.25 + 0.25 = 0.50$$

$$hl_2 = 0.22 + 0.22 + 0.25 = 0.69$$

$$hr_2 = 0.25$$

$$hl_3 = 0.22 + 0.22 + 0.25 + 0.25 = 0.94$$

$$hr_3 = 0$$

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

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

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

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

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

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

Gambar 3.18 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* keda.

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

$$gr_0 = -0.33$$

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

$$gr_1 = 0$$

$$gl_2 = 0.54 + 0.54 = 1.08$$

$$gr_2 = -0.33 - 0.33 = -0.66$$

$$gl_3 = 0.54$$

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

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

$$hr_0 = 0.22$$

$$hl_1 = 0.25 + 0.25 + 0.22 + 0.22 = 0.94$$

$$hr_1 = 0$$

$$hl_2 = 0.25 + 0.25 = 0.50$$

$$hr_2 = 0.22 + 0.22 = 0.44$$

$$hl_3 = 0.25$$

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

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

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

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

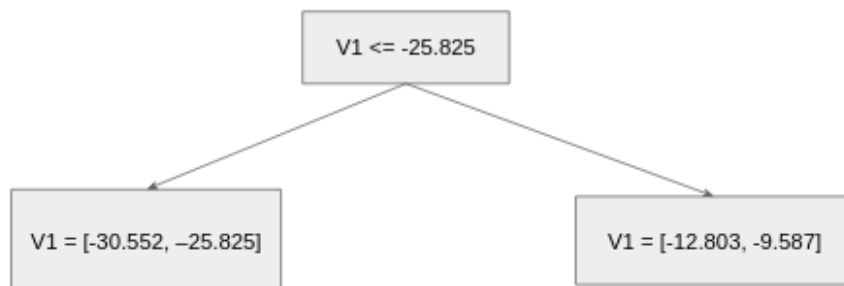
$$gain_2 = \frac{1}{2} \left[\frac{1.08^2}{0.50+1} + \frac{-0.66^2}{0.44+1} - \frac{(1.08+-0.66)^2}{0.50+0.44+1} \right] - 0 = 0.50$$

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

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

Gambar 3.19 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 $v1 \leq -25.825982149082$ 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 3.20 Visualisasi *tree* kedua setelah melakukan split berdasarkan fitur v1 dengan kondisi $v1 \leq -25.82598215$.

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

Perhitungan nilai bobot pada *leaf node* dengan fitur $v1 \leq -25.82598215$ sesuai dengan gambar 3.14 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 gambar 3.13 pada *tree* kedua.

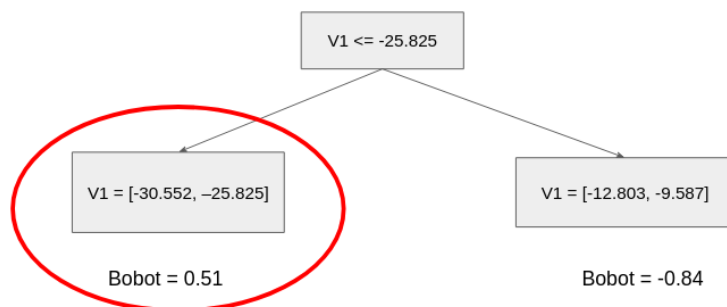
$$G = 0.54 + 0.54 = 1.08$$

$$H = 0.25 + 0.25 = 0.50$$

$$\lambda = 1$$

$$w = -\frac{1.08}{0.50+1} = -0.72$$

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 3.21 Nilai bobot untuk data indeks ke 0 adalah 0.51 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.

$$NewestPrediction_0 = 0.71 + 0.4 \times 0.46 = 0.89$$

$$NewestPrediction_1 = 0.71 + 0.4 \times 0.46 = 0.89$$

$$NewestPrediction_2 = 0.71 + 0.4 \times -0.72 = -0.12$$

$$NewestPrediction_3 = 0.71 + 0.4 \times -0.72 = -0.12$$

$$Newestprobability_0 = \frac{1}{1+e^{-0.89}} = 0.71$$

$$Newestprobability_1 = \frac{1}{1+e^{-0.89}} = 0.71$$

$$Newestprobability_2 = \frac{1}{1+e^{-(-0.12)}} = 0.47$$

$$Newestprobability_3 = \frac{1}{1+e^{-(-0.12)}} = 0.47$$

Index	v1	v2	v3	Class	Initial Prediction	New prediction	Newest Prediction	Final Probability
0	-30.55238004	16.71338924	-31.10368482	1	0.5	0.71	0.89	0.71
1	-25.82598215	19.16723901	-25.39022931	1	0.5	0.71	0.89	0.71
2	-12.80368875	9.666883426	-8.687080578	0	0.5	0.16	-0.12	0.47
3	-9.587359122	8.688022993	-2.877731892	0	0.5	0.16	-0.12	0.47

Gambar 3.22 Hasil prediksi akhir

3.8.3 Melakukan prediksi

Data yang akan digunakan untuk melakukan prediksi adalah data indeks ke 4 berdasarkan gambar 3.6. Prediksi dapat dihitung menggunakan persamaan 2.16.

Menghitung prediksi dan *probability* untuk data indeks ke 4.

$$Prediction = 0.5 + 0.4 \times 0.51 + 0.4 \times 0.45 = 0.89$$

$$Probability = \frac{1}{1 + e^{-0.89}} = 0.70$$

Karena hasil *probability* lebih besar dari 0.5, maka data indeks ke 4 diprediksi merupakan kelas 1. Pada data uji, label '*Class*' bernilai 1, sehingga hasil prediksi benar. Nilai akurasi, *recall*, dan *f1-score* atau *f-measure* mencapai 100%, karena data yang diuji hanya 1 buah. Berikut merupakan perhitungan nilai akurasi, *recall* dan *f1-score* atau *f-measure*.

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

$$Accuracy = \frac{1+0}{1+0+0+0}$$

$$Accuracy = \frac{1}{1} \times 100\%$$

$$Accuracy = 100\%$$

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

$$Recall = \frac{1}{1+0}$$

$$Recall = \frac{1}{1} \times 100\%$$

$$Recall = 100\%$$

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

$$F - measure = 2 \times \frac{1 \times 1}{1+1}$$

$$F - measure = 2 \times \frac{1}{2}$$

$$F - measure = 1 \times 100\%$$

$$F - measure = 100\%$$

BAB 4 IMPLEMENTASI DAN PENGUJIAN

- 4.1 Lingkungan Implementasi**
 - 4.1.1 Spesifikasi Perangkat Keras**
 - 4.1.2 Spesifikasi Perangkat Lunak**
- 4.2 Implementasi Perangkat Lunak**
 - 4.2.1 Implementasi *Class***
 - 4.2.1.1 *Class* Nama_Class_1**
 - 4.2.1.2 *Class* Nama_Class_2**
 - 4.2.2 Implementasi Numquam**
- 4.3 Implementasi Nama Implementasi**
- 4.4 Implementasi Aplikasi**
- 4.5 Pengujian**
 - 4.5.1 Pengujian Nama_Pengujian_1**
 - 4.5.2 Pengujian Nama_Pengujian_2**

BAB 5 KESIMPULAN DAN SARAN

5.1 Kesimpulan

5.2 Saran

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. [Online]. Available : <https://www.oreilly.com>, [Accessed: 28 January 2022].
- [9] *Xgboost*. [Online]. Available : <https://www.nvidia.com/en-us/glossary/data-science/xgboost/>, [Accessed: 28 January 2022].
- [10] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, 2005, ISBN: 9780387254654. [Online].

- Available : <https://link.springer.com/book/10.1007/b107408>, [Accessed: 28 January 2022].
- [11] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019, ISBN: 9781999579517. [Online]. Available : <http://themlbook.com/>, [Accessed: 10 October 2021].
 - [12] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016, ISBN: 9781449369415. [Online]. Available : <https://www.oreilly.com>, [Accessed: 28 January 2022].
 - [13] 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.
 - [14] 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].
 - [15] 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].
 - [16] C. Wade and K. Glynn, *Hands-On Gradient Boosting with XGBoost and scikit-learn*. 2020, ISBN: 9781839218354. [Online]. Available : <https://www.oreilly.com>, [Accessed: 31 January 2022].
 - [17] 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].
 - [18] 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].
 - [19] *Fighting overfitting with l1 or l2 regularization: Which one is better?* [Online]. Available : <https://neptune.ai>, [Accessed: 12 February 2022].
 - [20] *Xgboost parameters*. [Online]. Available : <https://xgboost.readthedocs.io/en/stable/parameter.html>, [Accessed: 30 January 2022].

- [21] M. Swamynathan, *Mastering Machine Learning with Python in Six Steps*. 2017, ISBN: 9781484228661. [Online]. Available : <https://link.springer.com/book/10.1007/978-1-4842-2866-1>, [Accessed: 10 October 2021].
- [22] *How to fix an unbalanced dataset*. [Online]. Available : <https://www.kdnuggets.com>, [Accessed: 13 February 2022].
- [23] *Smote*. [Online]. Available : <https://docs.microsoft.com/en-us/previous-versions/azure/machine-learning/studio-module-reference/smote#expected-input>, [Accessed: 29 January 2022].
- [24] T. Agrawal, *Hyperparameter Optimization in Machine Learning*. 2021, ISBN: 9781484265796. [Online]. Available : <https://link.springer.com>, [Accessed: 10 October 2021].
- [25] 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>.

LAMPIRAN A LAMPIRAN A

ASJDBAKJSDBKA

Tabel A-1 *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
1	Ei utroque electram	0	0	10	Laudem
2	Ei utroque electram	3	0	7	Laudem
3	Ei utroque electram	2	0	8	Laudem
4	Ei utroque electram	0	3	7	Laudem
5	Ei utroque electram	10	0	0	Laudem
6	Ei utroque electram	0	0	10	Laudem
7	Ei utroque electram	3	0	7	Laudem
8	Ei utroque electram	2	0	8	Laudem
9	Ei utroque electram	0	3	7	Laudem
10	Ei utroque electram	10	0	0	Laudem
11	Ei utroque electram	0	0	10	Laudem
12	Ei utroque electram	3	0	7	Laudem
13	Ei utroque electram	2	0	8	Laudem
14	Ei utroque electram	0	3	7	Laudem
15	Ei utroque electram	10	0	0	Laudem
16	Ei utroque electram	0	0	10	Laudem
17	Ei utroque electram	3	0	7	Laudem
18	Ei utroque electram	2	0	8	Laudem
19	Ei utroque electram	0	3	7	Laudem
20	Ei utroque electram	10	0	0	Laudem
21	Ei utroque electram	0	0	10	Laudem
22	Ei utroque electram	3	0	7	Laudem
23	Ei utroque electram	2	0	8	Laudem
24	Ei utroque electram	0	3	7	Laudem
25	Ei utroque electram	10	0	0	Laudem

Tabel A-1 *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
26	Ei utroque electram	0	0	10	Laudem
27	Ei utroque electram	3	0	7	Laudem
28	Ei utroque electram	2	0	8	Laudem
29	Ei utroque electram	0	3	7	Laudem
30	Ei utroque electram	10	0	0	Laudem
31	Ei utroque electram	0	0	10	Laudem
32	Ei utroque electram	3	0	7	Laudem
33	Ei utroque electram	2	0	8	Laudem
34	Ei utroque electram	0	3	7	Laudem
35	Ei utroque electram	10	0	0	Laudem
36	Ei utroque electram	0	0	10	Laudem
37	Ei utroque electram	3	0	7	Laudem
38	Ei utroque electram	2	0	8	Laudem
39	Ei utroque electram	0	3	7	Laudem
40	Ei utroque electram	10	0	0	Laudem

LAMPIRAN B DATASET HASIL KUISIONER 2

Tabel B-1 *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
1	Ei utroque electram	0	0	10	Laudem
2	Ei utroque electram	3	0	7	Laudem
3	Ei utroque electram	2	0	8	Laudem
4	Ei utroque electram	0	3	7	Laudem
5	Ei utroque electram	10	0	0	Laudem
6	Ei utroque electram	0	0	10	Laudem
7	Ei utroque electram	3	0	7	Laudem
8	Ei utroque electram	2	0	8	Laudem
9	Ei utroque electram	0	3	7	Laudem
10	Ei utroque electram	10	0	0	Laudem
11	Ei utroque electram	0	0	10	Laudem
12	Ei utroque electram	3	0	7	Laudem
13	Ei utroque electram	2	0	8	Laudem
14	Ei utroque electram	0	3	7	Laudem
15	Ei utroque electram	10	0	0	Laudem
16	Ei utroque electram	0	0	10	Laudem
17	Ei utroque electram	3	0	7	Laudem
18	Ei utroque electram	2	0	8	Laudem
19	Ei utroque electram	0	3	7	Laudem
20	Ei utroque electram	10	0	0	Laudem
21	Ei utroque electram	0	0	10	Laudem
22	Ei utroque electram	3	0	7	Laudem
23	Ei utroque electram	2	0	8	Laudem
24	Ei utroque electram	0	3	7	Laudem
25	Ei utroque electram	10	0	0	Laudem
26	Ei utroque electram	0	0	10	Laudem

Tabel B-1 *Lorem ipsum*

No	<i>Dolor sit amet</i>	At sonet	Vim commune	At quo congue	Cum iisque
27	Ei utroque electram	3	0	7	Laudem
28	Ei utroque electram	2	0	8	Laudem
29	Ei utroque electram	0	3	7	Laudem
30	Ei utroque electram	10	0	0	Laudem
31	Ei utroque electram	0	0	10	Laudem
32	Ei utroque electram	3	0	7	Laudem
33	Ei utroque electram	2	0	8	Laudem
34	Ei utroque electram	0	3	7	Laudem
35	Ei utroque electram	10	0	0	Laudem
36	Ei utroque electram	0	0	10	Laudem
37	Ei utroque electram	3	0	7	Laudem
38	Ei utroque electram	2	0	8	Laudem
39	Ei utroque electram	0	3	7	Laudem
40	Ei utroque electram	10	0	0	Laudem