PENDEKATAN ALGORITMA GENETIKA UNTUK OPTIMASI HYPERPARAMETER PADA MODEL RBF DALAM DETEKSI PENIPUAN KARTU KREDIT

Reyner Ongkowijoyo

181221042

Program S-1 Matematika, Fakultas Sains dan Teknologi
Universitas Airlangga

Jl. Mulyorejo, Kampus C,
Surabaya 60115

Abstract: Deteksi penipuan kartu kredit telah menjadi tantangan krusial di sektor perbankan, terutama dalam konteks Revolusi Industri Keempat dan meningkatnya ancaman keamanan siber. Penelitian ini mengeksplorasi penerapan Algoritma Genetika (GA) untuk optimasi hiperparameter pada jaringan saraf Radial Basis Function (RBF) dalam menangani masalah ini. Model RBF, yang dikenal mampu menangani tugas klasifikasi yang kompleks, ditingkatkan melalui GA untuk menentukan konfigurasi parameter optimal seperti jumlah centroid dan nilai spread. Menggunakan dataset publik yang tersedia di Kaggle, penelitian ini menggunakan dataset seimbang yang telah di-downsample untuk mengklasifikasikan transaksi penipuan dan transaksi sah. GA mengoptimalkan hiperparameter secara iteratif melalui proses seperti mutasi dan seleksi, dengan panduan skor fitness berdasarkan performa F1. Hasil penelitian menunjukkan bahwa akurasi pengujian tertinggi sebesar 94,8% dicapai pada iterasi keenam, dengan konfigurasi optimal berupa 395 centroid. Temuan ini menunjukkan kemampuan GA untuk memperbaiki model RBF, mencapai keseimbangan antara generalisasi dan kompleksitas sambil menghindari overfitting. Penelitian ini menyimpulkan bahwa GA merupakan metode yang menjanjikan untuk tuning hiperparameter pada jaringan saraf tiruan, dengan syarat sumber daya komputasi yang memadai tersedia.

Keywords: algoritma genetika, radial basis function, tuning hiperparameter, klasifikasi penipuan kartu kredit, jaringan saraf tiruan

KATA PENGANTAR

Penulis menyampaikan rasa syukur kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas makalah yang berjudul "PENDEKATAN ALGORITMA GENETIKA UNTUK OPTIMASI HYPERPARAMETER PADA MODEL RBF DALAM DETEKSI PENIPUAN KARTU KREDIT" ini tepat pada waktunya. Adapun tujuan dari penulisan makalah ini adalah untuk memenuhi tugas individu mata kuliah Simulasi. Selain itu, makalah ini juga bertujuan untuk menambah wawasan dan informasi terbaru mengenai penerapan algoritma genetika bagi para pembaca dan juga bagi penulis.

Kami mengucapkan terima kasih kepada bapak Dr. Ahmadin, S.Si., M.Si. dan bapak Drs. Edi Winarko, M.Cs selaku pembimbing mata kuliah Simulasi yang telah memberikan tugas makalah ini sehingga kami dapat menambah pengetahuan dan wawasan sesuai dengan bidang yang kami tekuni. Kami menyadari, makalah yang kami tulis ini masih jauh dari kata sempurna. Oleh karena itu, kritik dan saran yang membangun akan kami nantikan demi kesempurnaan makalah ini.

Surabaya, 16 Desember 2024

Penulis

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di era digitalisasi dan Revolusi Industri Keempat, transformasi digital di sektor perbankan telah menjadi sangat penting untuk mengatasi perubahan dalam ekonomi global [1]. Industri perbankan harus memanfaatkan transformasi digital untuk meningkatkan daya saing dan tetap relevan di pasar. Studi terbaru [2] juga menunjukkan bahwa sangat penting untuk mengelola risiko dan memastikan keamanan transaksi di tengah meningkatnya prevalensi kejahatan dunia maya, seperti penipuan kartu kredit. Untuk mengatasi tantangan ini, teknologi seperti pembelajaran mesin menawarkan solusi efektif untuk mendeteksi pola dan memprediksi aktivitas penipuan dengan cepat dan akurat.

Deteksi penipuan kartu kredit menggunakan metode pembelajaran mesin memungkinkan identifikasi dan penemuan pola dari kumpulan data transaksi yang diberikan. Radial Basis Function (RBF), sejenis jaringan saraf tiruan, sangat cocok untuk menangani masalah klasifikasi [3]. Namun, salah satu tantangan utama dalam mengimplementasikan model RBF terletak pada pemilihan parameter optimal seperti penentuan centroid, spread, learning rate, dan jumlah node.

Untuk mengatasi tantangan ini, Algoritma Genetika (GA), yang terinspirasi oleh proses evolusi biologis, dapat digunakan sebagai metode optimasi hiperparameter [4]. Algoritma ini secara iteratif mengeksplorasi ruang hiperparameter dan memilih kromosom atau individu terbaik yang mewakili hiperparameter terkait dengan fungsi kebugaran, yang dalam hal ini adalah skor metrik berdasarkan dataset uji. Dengan demikian, Algoritma Genetika dapat membantu mengidentifikasi kumpulan hiperparameter optimal untuk meningkatkan skor metrik dalam klasifikasi deteksi penipuan kartu kredit menggunakan model RBF.

Penelitian ini berfokus pada membangun model jaringan saraf RBF untuk deteksi penipuan kartu kredit, mengoptimalkan hiperparameternya menggunakan Algoritma Genetika, dan mengevaluasi performanya setelah optimasi. Dengan mengintegrasikan metode-metode ini, studi ini bertujuan untuk berkontribusi pada pengembangan sistem deteksi penipuan yang lebih akurat dan mengeksplorasi potensi Algoritma Genetika dalam penyetelan hiperparameter.

1.2 Rumusan Masalah

1. Bagaimana menkonstruksikan model jaringan syaraf tiruan RBF untuk klasifikasi deteksi penipuan kartu kredit
2. Bagaimana mencari parameter optimal dalam model RBF menggunakan Algoritma Genetika
3. Bagaimana kinerja model RBF setelah dioptimasi menggunakan Algoritma Genetika dalam mendeteksi penipuan kartu kredit

1.3 Tujuan

1. Mengidentifikasi dan mengimplementasikan model RBF untuk klasifikasi deteksi penipuan kartu kredit.
2. Menerapkan Algoritma Genetika untuk mengoptimalkan hyperparameter pada model RBF dalam deteksi penipuan kartu kredit.
3. Mengevaluasi kinerja model RBF-GA dalam mendeteksi penipuan kartu kredit melalui analisis akurasi dan efisiensi.

1.4 Manfaat

1. Memberikan kontribusi dalam pengembangan metode deteksi penipuan kartu kredit yang lebih akurat melalui optimasi hyperparameter menggunakan Algoritma Genetika.
2. Meningkatkan efisiensi sistem deteksi penipuan di dunia finansial, sehingga dapat membantu mengurangi kerugian akibat penipuan kartu kredit.

1.5 Batasan

1. Model dan algoritma akan dikembangkan menggunakan Visual Studio Code dengan Python dalam lingkungan Jupyter Notebook.
2. Untuk menjaga originalitas pengembangan model dan algoritma, hanya library dasar seperti numpy, pandas, random, dan display yang akan digunakan. Tidak ada penggunaan library machine learning tambahan seperti scikit-learn atau tensorflow.
3. Algoritma Genetika akan digunakan secara spesifik untuk optimasi hyperparameter, tanpa kombinasi metode optimasi lain seperti Grid Search atau Random Search.

BAB II
TINJAUAN PUSTAKA

2.1 Genetic Algorithm

Berdasarkan [5], Algoritma Genetika (GA) didefinisikan sebagai pendekatan optimasi metaheuristik yang banyak digunakan, terinspirasi oleh teori evolusi, di mana individu dengan sifat kelangsungan hidup dan adaptasi yang superior lebih mungkin untuk berkembang dan mewariskan atribut mereka ke generasi mendatang. Melalui proses seleksi dari setiap individu atau setiap kromosom berdasarkan fungsi objektif yang terdefinisi dengan baik, generasi berikutnya mewarisi karakteristik ini, sering menghasilkan campuran individu yang lebih baik dan lebih lemah. Seiring waktu, individu yang paling adaptif diidentifikasi sebagai optimum global [6].

Dalam konteks optimasi hiperparameter (HPO), GA merepresentasikan setiap hiperparameter sebagai kromosom atau individu, dan nilai desimalnya sesuai dengan input aktual hiperparameter selama evaluasi. Operasi seperti crossover dan mutasi diterapkan pada gen-gen ini untuk menyempurnakan pencarian. Populasi mewakili rentang nilai hiperparameter yang mungkin diinisialisasi, sedangkan fungsi kebugaran mengevaluasi kinerja parameter-parameter ini [6].

Karena nilai parameter yang diinisialisasi secara acak sering kali tidak termasuk yang optimal, GA menggabungkan proses seleksi, crossover, dan mutasi untuk menyempurnakan populasi. Kromosom dengan nilai kebugaran lebih tinggi dipilih dengan probabilitas lebih besar, memastikan sifat superior diwariskan ke generasi berikutnya. Kromosom-kromosom ini menghasilkan keturunan baru yang mencerminkan sifat terkuat mereka. Melalui crossover, sebagian gen dipertukarkan antara kromosom, menghasilkan keragaman. Mutasi memperkenalkan keacakan dengan mengubah gen tertentu, mengurangi kemungkinan kehilangan solusi superior.

Prosedur utama GA adalah sebagai berikut [7]:

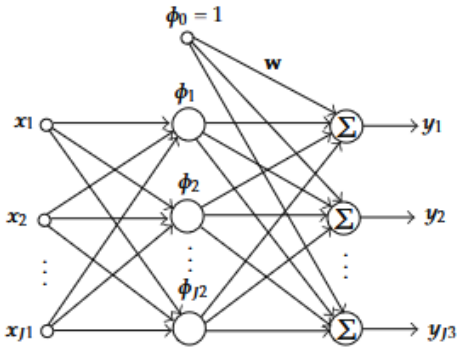
1. Inisialisasi populasi, kromosom, dan gen secara acak untuk mencakup ruang pencarian dan merepresentasikan hiperparameter dan nilainya.
2. Evaluasi kinerja setiap individu menggunakan fungsi kebugaran, yang mengukur fungsi objektif dari model pembelajaran mesin.

3. Terapkan seleksi, crossover, dan mutasi untuk menghasilkan populasi baru dengan konfigurasi hiperparameter yang disempurnakan.
4. Ulangi langkah 2 dan 3 hingga kondisi terminasi terpenuhi.
5. Keluarkan konfigurasi hiperparameter optimal.

Inisialisasi populasi adalah langkah kritis dalam GA karena memberikan estimasi awal nilai-nilai optimal. Meskipun nilai-nilai ini secara iteratif ditingkatkan selama proses optimasi, inisialisasi yang efektif dapat secara signifikan meningkatkan kecepatan konvergensi dan kinerja optimasi. Populasi yang diinisialisasi dengan baik harus mencakup individu yang dekat dengan optima global dan menghindari terlokalisasi ke area ruang pencarian yang tidak menjanjikan [7]. Inisialisasi acak, yang menghasilkan populasi awal dengan nilai acak dalam ruang pencarian, adalah teknik yang umum digunakan dalam GA [8]. Meskipun kesederhanaannya, operasi seleksi, crossover, dan mutasi yang kuat dari GA memastikan bahwa tetap efektif dalam menghindari kehilangan optima global.

2.2 Radial Basis Function

Radial Basis Function (RBF) merupakan jaringan saraf feedforward tiga lapis, yang terdiri dari lapisan masukan, lapisan tersembunyi, dan lapisan keluaran, seperti ditunjukkan pada **Gambar 1** [9].



Gambar 1 Jenis Arsitektur jaringan RBF. $\boldsymbol{\phi_0(\vec{x}) = 1}$ c berhubungan dengan bias pada lapisan keluaran, sedangkan $\boldsymbol{\phi_i(\vec{x})}$'s menunjukkan nonlinier pada node tersembunyi.

Setiap node di lapisan tersembunyi menerapkan RBF, dilambangkan dengan $\phi(\boldsymbol{r})$, sebagai fungsi aktivasi nonlinier. Lapisan tersembunyi menerapkan transformasi nonlinier dari input, sedangkan lapisan output berfungsi sebagai penggabung linier, mengubah nonlinieritas menjadi ruang baru. Biasanya, semua node RBF berbagi fungsi aktivasi yang sama. Nonlinieritas pada setiap node RBF direpresentasikan sebagai $\boldsymbol{\phi_i(\vec{x} - \vec{c}_i)}$, di mana $\boldsymbol{\vec{c}_i}$

menunjukkan pusat atau prototipe dari node $\boldsymbol{i}$-th, dan $\boldsymbol{\phi_i}(\vec{\boldsymbol{x}})$ merepresentasikan RBF. Selain itu, bias untuk neuron di lapisan output dapat dimasukkan melalui neuron ekstra di lapisan tersembunyi [9].

Salah satu fungsi contoh yang dapat digunakan sebagai RBF adalah fungsi Gaussian [10], yang didefinisikan sebagai:

$$\boldsymbol{\phi}(\boldsymbol{r}) = \boldsymbol{e}^{-r^2/2\sigma^2} \qquad (2.1)$$

di mana $r > 0$ menunjukkan nilai jarak dari titik data $\vec{\boldsymbol{x}}$ terhadap pusat atau centroid $\vec{\boldsymbol{c}}$, dan $\boldsymbol{\sigma}$ yang kadang-kadang disebut sebagai spread, digunakan untuk mengontrol kelancaran fungsi interpolasi. Selain lapisan tersembunyi, lapisan output dalam jaringan RBF adalah penggabung linier yang didefinisikan sebagai:

$$\boldsymbol{y_i}(\vec{\boldsymbol{x}}) = \sum_{i=1}^{J} \boldsymbol{w}_i \boldsymbol{\phi}_i(\vec{\boldsymbol{x}}) \qquad (2.2)$$

di mana $\boldsymbol{w}_i$ adalah bobot yang menghubungkan lapisan tersembunyi ke lapisan output, J adalah jumlah node RBF di lapisan tersembunyi, dan $\boldsymbol{y_i}(\vec{\boldsymbol{x}})$ adalah output dari node ke-$\boldsymbol{i}$ dari jaringan untuk input $\vec{\boldsymbol{x}}$ yang diberikan.

Salah satu metode pelatihan untuk optimasi bobot adalah regresi linier yang meminimalkan kesalahan output. Dalam praktiknya, metode pseudo-inverse atau least squares dapat digunakan untuk menghitung bobot ini:

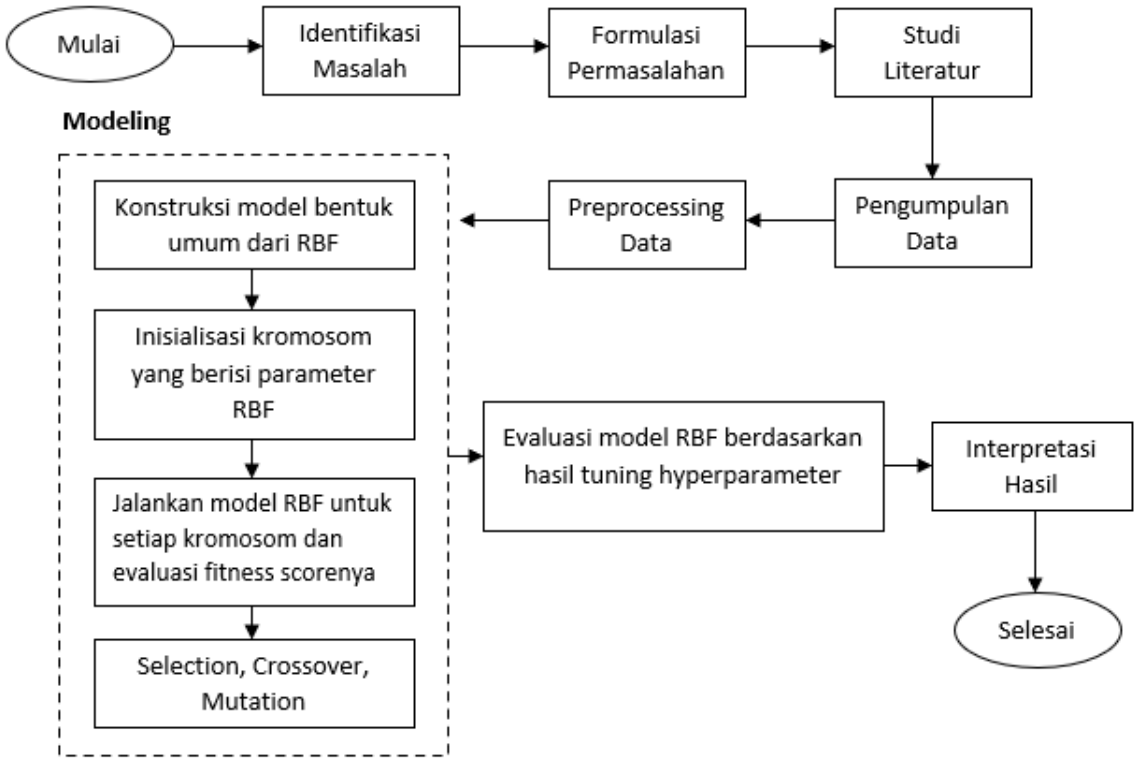
$$\boldsymbol{W} = \boldsymbol{\Phi}^{+}\mathbf{Y} \qquad (2.3)$$

di mana $\boldsymbol{\Phi}$ adalah matriks output lapisan tersembunyi (aktivasi Gaussian), $\mathbf{Y}$ adalah matriks output target, dan $\boldsymbol{\Phi}^{+}$ adalah pseudo-inverse dari matriks $\boldsymbol{\Phi}$ [9].

BAB III
METODOLOGI PENELITIAN

3.1 Alur Penelitian

Metodologi penelitian ini didasarkan pada pendekatan penelitian kuantitatif. Tujuannya adalah untuk mengklasifikasikan apakah kredit tersebut penipuan atau sah menggunakan model RBF dengan optimasi parameter dari Genetic Algorithm. Alur kerja penelitian diilustrasikan pada **Gambar 2**.



Gambar 2 Alur Penelitian

3.2 Pengambilan Dataset

Dataset yang digunakan dalam penelitian ini diambil secara sekunder dari platform Kaggle. Dataset yang digunakan merupakan bagian dari **"The Bank Account Fraud (BAF) Suite"**, yang dipublikasikan di NeurIPS 2022 [11]. Dataset ini terdiri dari enam dataset tabular sintetis terkait deteksi penipuan rekening bank. BAF menyediakan lingkungan pengujian yang realistis, dinamis, dan menjaga privasi untuk mengevaluasi metode machine learning, termasuk menangani masalah bias terkontrol serta pergeseran distribusi temporal.

Oleh karena itu, peneliti yakin bahwa dataset yang dipilih sesuai dengan rumusan masalah dan tujuan yang telah dijabarkan pada bab sebelumnya.

Karena keterbatasan komputasi, peneliti memilih subset data untuk diolah lebih lanjut dengan melakukan **downsampling** dari total sekitar 1 juta data menjadi 1.000 data yang telah diseimbangkan untuk keperluan klasifikasi biner, yaitu antara transaksi penipuan (fraud) dan transaksi sah (non-fraud). Selain itu, dari 32 fitur yang tersedia, telah diseleksi 8 fitur berdasarkan pengetahuan domain dan untuk menyederhanakan proses komputasi. Selanjutnya, data yang telah di-downsample akan dibagi menjadi data latih dan data validasi secara acak dengan proporsi 75% untuk data latih dan 25% untuk data validasi.

3.3 Pengolahan Dataset

Dataset yang telah diseleksi dengan proses downsampling dan pembagian train-validation sebagaimana dijelaskan sebelumnya akan melalui tahap preprocessing dan pembersihan data lebih lanjut. Berdasarkan metadata yang disediakan, kolom *current_address_months_count* mengandung nilai -1 sebagai penanda data yang hilang. Pada tahap ini, peneliti akan mengganti nilai -1 tersebut dengan nilai median kolom tersebut yang dihitung dari data pelatihan. Penggantian nilai missing dengan median ini bertujuan untuk menjaga distribusi data tetap representatif tanpa memengaruhi struktur nilai-nilai asli pada data validasi.

Selanjutnya, untuk kolom kategorikal seperti payment_type dan employment_status, dilakukan teknik ***target encoding*** guna menggantikan nilai kategori dengan nilai numerik yang mencerminkan hubungan rata-rata antara kategori dan target variabel. *Target encoding* dihitung dengan mengambil rata-rata target untuk setiap kategori dalam data pelatihan, sehingga setiap kategori memiliki nilai numerik yang merepresentasikan kemungkinan kejadiannya dalam kaitannya dengan target. Rumus perhitungan mean target untuk kategori adalah sebagai berikut:

$$Enc(k) = \frac{\sum_{i=1}^{N_k} y_i}{N_k},$$

dimana:

- N_k adalah jumlah total data dalam kategori k

- y_i adalah nilai target (dimana kasus ini, 1 untuk transaksi fraud dan 0 untuk non-fraud)
- $Enc(k)$ adalah nilai rata-rata target untuk kategori k

Setelah target encoding selesai, semua fitur numerik termasuk hasil encoding kategori akan dinormalisasi dengan ***Standard Scaling***. *Standard Scaling* bertujuan untuk mengubah distribusi fitur menjadi memiliki rata-rata 0 dan standar deviasi 1, agar skala nilai antar-fitur menjadi seragam dan meningkatkan stabilitas algoritma pada tahap pemodelan. Rumus Standard Scaling untuk setiap nilai ***x*** pada kolom fitur adalah sebagai berikut:

$$x' = \frac{x - \mu}{\sigma}$$

Dimana:

- X merupakan nilai dari fitur/kolom input
- μ merupakan rata-rata dari fitur/kolom input
- σ merupakan stadar deviasi dari fitur pada data pelatihan

2.4 Algoritma dan Pemodelan

Model yang digunakan dalam penelitian ini adalah model jaringan saraf dengan arsitektur Radial Basis Function (RBF) yang memiliki satu hidden layer sebagaimana umumnya. Pada hidden layer, node RBF berfungsi untuk mengukur kesamaan data input dengan suatu titik pusat (centroid) yang telah ditentukan, menggunakan fungsi kernel Gaussian. Model tersebut menggunakan metode **pseudoinverse** pada layer output untuk mendapatkan solusi optimal yang meminimalkan error pada data latih.

Untuk menentukan posisi centroid dan nilai spread (parameter skala Gaussian) pada hidden layer, digunakan algoritma genetika (GA) sebagai metode optimasi. GA dirancang untuk mengoptimalkan parameter-parameter ini dengan tahapan sebagai berikut:

1. **Inisialisasi Populasi Awal**: Populasi awal dibentuk sebagai sekumpulan kromosom yang merepresentasikan koordinat centroid dan nilai spread untuk setiap neuron pada hidden layer. Setiap kromosom mengandung parameter yang mendefinisikan konfigurasi centroid dan spread di dalam RBF. Pada penelitian ini ditentukan jumlah populasi awal sebesar 500**.**

2. **Evaluasi dan Seleksi**: Setiap kromosom dalam populasi dinilai berdasarkan performa model RBF dengan menghitung nilai **F1 score** sebagai fitness. Kromosom dengan nilai **F1** terbaik dipilih sebagai dasar untuk tahap seleksi berikutnya, di mana sejumlah kromosom terbaik sesuai jumlah parameter diteruskan ke tahap berikutnya. Pada penelitian ini, parameter seleksi yang digunakan sebanyak 50% dari populasi awal yaitu sebesar 250.
3. **Crossover**: Setelah seleksi, crossover dilakukan dengan memilih dua kromosom secara acak sebagai pasangan induk. Dari setiap pasangan induk ini, dihasilkan dua kromosom anak dengan pewarisan parameter secara parsial, yaitu 80% dari parameter kromosom induk pertama dan 20% dari kromosom kedua, dan sebaliknya. Proses ini diulangi hingga mencapai jumlah populasi awal, di mana kromosom induk tidak digunakan dalam generasi berikutnya dan hanya kromosom anak yang dilanjutkan ke tahap mutasi.
4. **Mutasi**: Pada tahap ini, setiap kromosom anak yang telah terbentuk melalui crossover dievaluasi untuk melihat apakah akan mengalami mutasi, berdasarkan ***exposure rate*** yang telah ditentukan. Jika suatu kromosom terpilih untuk mutasi, maka beberapa parameter utamanya akan berubah:
    - **Spread**: Nilai spread diperbarui dengan persamaan $Spread\ akhir = Spread\ awal \times (1 \pm RNG\ mutation\ rate)$. Dengan $RNG\ mutation\ rate$ adalah nilai acak yang diambil dari interval [-mutation rate, mutation rate].
    - **Koordinat Centroid**: Setiap kromosom memiliki peluang sebesar nilai mutation rate untuk mengalami perubahan koordinat centroid secara acak berdasarkan sampel dari dataset.
    - **Jumlah Centroid**: Nilai jumlah centroid juga dapat diperbarui menggunakan persamaan $Centroid\ akhir = Centroid\ awal \times (1 \pm RNG\ mutation\ rate)$ sehingga memungkinkan perubahan jumlah node RBF pada hidden layer. Apabila jumlah *centroid* akhir berbentuk decimal akan dilakukan *round-up* agar tetap berbentu bilangan bulat positif.

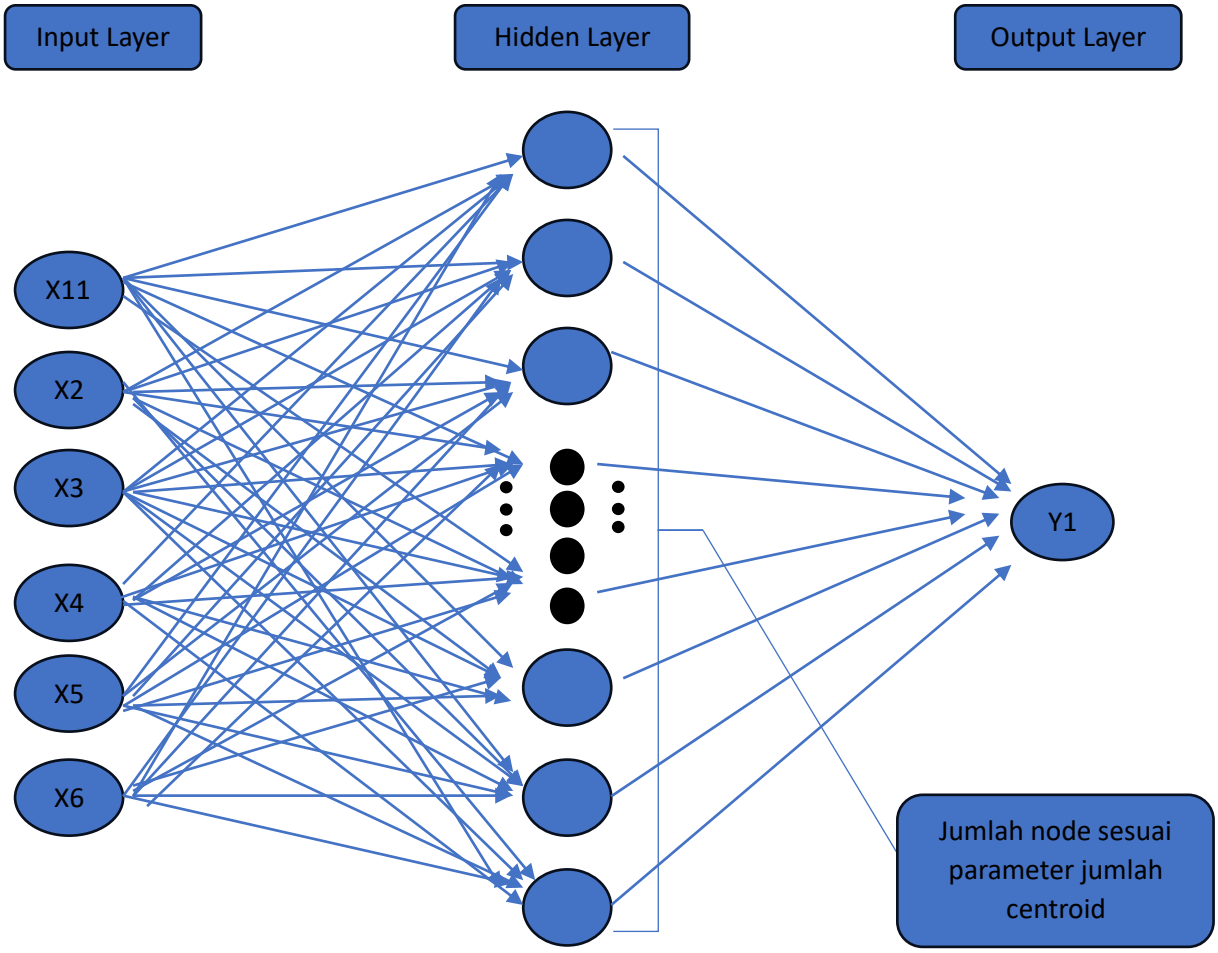
    Pada penelitian ini, digunakan *exposure rate* sebesar 0.25 dan *mutation rate* sebesar 0.4 untuk mengurangi risiko terjebak pada masalah *local optima* sekaligus menjaga diversifikasi agar proses optimasi tetap efektif dan tidak terlalu cepat mengalami konvergensi.

Setelah tahap mutasi selesai maka dapat proses **evaluasi, seleksi, crossover,** dan **mutasi** dalam diulangi sesuai dengan jumlah iterasi yang ditetapkan dengan harapan agar parameter pada setiap generasi memiliki performa lebih baik sampai sesuai dengan hasil yang diinginkan. Untuk alasan kesederhanaan dan menghemat daya komputasi, peneliti menjalankan algoritma genetika sebanyak 9 iterasi.

BAB IV
PEMBAHASAN

4.1 Hasil Konstruksi Model

Pada tahap ini, model jaringan saraf Radial Basis Function (RBF) telah berhasil dibangun dan disiapkan untuk dilatih menggunakan dataset yang telah diproses sebelumnya. Untuk lebih jelasnya model RBF dapat dilihat sebagai berikut:



Catatan:
X1 (Income)
X2 (Credit Risk Score)
X3 (Customer Age)
X4 (Payment Type)
X5 (Employment Status)
X6 (Current_Address_months count)
Untuk lebih jelasnya dapat dilihat pada [11]

4.2 Hasil Kinerja Model RBF dengan Optimasi Parameternya

Setelah model Radial Basis Function (RBF) dibangun, dilakukan penerapan algoritma genetika untuk mengoptimalkan parameter, terutama jumlah centroid yang optimal. Proses ini mengikuti metode dan parameter yang telah dijelaskan dalam Bab III. Hasil dari penerapan algoritma genetika dalam pencarian jumlah centroid terbaik dapat dilihat pada **Tabel 1** di bawah ini melalui program [12]..

Tabel 1 Hasil dari GA *hyperparameter tuning*

Iteration	Train dataset Accuracy	Test dataset Accuracy	Centroid Counts	Spread Value	Fitness Score Value
1	0.7693	0.812	43	1.3107	0.8000
2	0.8067	0.816	71	1.2878	0.8115
3	0.7987	0.828	142	1.4494	0.8170
4	0.8267	0.828	157	1.4135	0.8186
5	0.9013	0.876	213	1.2026	0.8703
6	0.9573	0.948	395	0.3992	0.9456
7	0.9547	0.928	361	0.9456	0.9244
8	0.9453	0.924	447	0.3568	0.9212
9	0.9533	0.920	341	0.6338	0.9153

Dari tabel tersebut, dapat dilihat bahwa nilai akurasi data uji tertinggi sebesar 94,8% dicapai pada iterasi ke-6 dengan jumlah centroid optimal sebesar 395. Jumlah centroid optimal sebesar 395 yang dicapai pada iterasi ke-6 mengindikasikan bahwa model RBF telah menemukan keseimbangan yang baik antara kemampuan menggeneralisasi dan kompleksitas model. Jumlah centroid yang terlalu banyak dapat menyebabkan overfitting. Peningkatan jumlah centroid pada setiap iterasi merupakan hasil dari interaksi antara proses mutasi dan seleksi dalam algoritma genetika. Mutasi menghasilkan variasi dalam jumlah centroid, sementara seleksi berdasarkan nilai fitness score menentukan individu mana yang akan bertahan dan diteruskan ke generasi berikutnya. Seiring berjalannya iterasi, algoritma genetika cenderung memilih individu dengan jumlah centroid yang optimal, yaitu jumlah yang cukup untuk menangkap pola dalam data tanpa menyebabkan overfitting.

BAB V
PENUTUP

5.1 Kesimpulan

Pendekatan algoritma genetika dalam optimasi parameter model Radial Basis Function (RBF) terbukti efektif dalam menemukan kombinasi parameter yang optimal, terutama terkait dengan jumlah centroid. Dari tabel hasil, didapatkan bahwa akurasi tertinggi pada data uji sebesar 94,8% dicapai pada iterasi ke-6 dengan jumlah centroid optimal sebesar 395, menunjukkan bahwa model RBF mampu mencapai keseimbangan antara generalisasi dan kompleksitas.

Penerapan algoritma genetika memungkinkan variasi pada jumlah centroid melalui proses mutasi, sedangkan seleksi berdasarkan fitness score membantu memilih individu dengan performa terbaik pada setiap generasi. Dalam proses iterasi, algoritma genetika cenderung menyempurnakan model dengan memilih individu yang memiliki akurasi tinggi namun tidak terlalu kompleks untuk menghindari overfitting. Oleh karena itu, algoritma genetika dapat dianggap sebagai metode yang menjanjikan untuk optimasi parameter atau hyperparameter tuning pada model jaringan saraf tiruan (JST), asalkan disertai dengan sumber daya komputasi yang memadai untuk menangani banyak iterasi dan kombinasi.

5.2 Saran

Penelitian ini masih memiliki keterbatasan, seperti ketergantungan pada pustaka tertentu serta penggunaan dataset sekunder yang dapat memengaruhi akurasi model jika diimplementasikan pada skala dunia nyata. Kami menyarankan agar penelitian selanjutnya menggunakan dataset yang lebih besar dan bervariasi untuk meningkatkan kemampuan generalisasi model. Selain itu, jumlah fitur yang digunakan juga sebaiknya ditambah agar model dapat mengidentifikasi pola yang lebih kompleks dan spesifik.

Di sisi pengembangan model, eksplorasi terhadap jenis model lain seperti Multi-Layer Perceptron (MLP), XGBoost, dan model lainnya juga patut dipertimbangkan. Setiap model ini memiliki parameter yang unik dan berpotensi memberikan performa yang lebih baik dalam kasus tertentu, sehingga optimasi model dengan pendekatan yang lebih variatif akan membuka peluang untuk mencapai hasil yang lebih akurat dan robust.

DAFTAR PUSTAKA

[1] Ardianto R, Ramdhani RF, Apriliana Dewi LO, Prabowo A, Saputri YW, Lestari AS, Hadi N. Transformasi Digital dan Antisipasi Perubahan Ekonomi Global dalam Dunia Perbankan. MARAS: Jurnal Penelitian Multidisiplin. 2024;2(1):80–88. doi:10.60126/maras.v2i1.114

[2] Mijwil MM, Aljanabi M, ChatGPT. Towards Artificial Intelligence-Based Cybersecurity: The Practices and ChatGPT Generated Ways to Combat Cybercrime. Iraqi Journal for Computer Science and Mathematics. 2023;4(1):65–70. doi:10.52866/ijcsm.2023.01.01.0019

[3] Shekhar S, Xiong H. Artificial Neural Network. In: Encyclopedia of GIS. Boston, MA: Springer US; 2008. p. 31–31. doi:10.1007/978-0-387-35973-1_72

[4] Raji ID, Bello-Salau H, Umoh IJ, Onumanyi AJ, Adegboye MA, Salawudeen AT. Simple Deterministic Selection-Based Genetic Algorithm for Hyperparameter Tuning of Machine Learning Models. Applied Sciences (Switzerland). 2022;12(3). doi:10.3390/app12031186

[5] Lessmann S, Stahlbock R, Crone SF. Optimizing Hyperparameters of Support Vector Machines by Genetic Algorithms. Optimizing Hyperparameters of Support Vector Machines by Genetic Algorithms. 2005. https://www.researchgate.net/publication/220835507

[6] Itano F, de Abreu de Sousa MA, Del-Moral-Hernandez E. Extending MLP ANN hyper-parameters Optimization by using Genetic Algorithm. In: 2018 International Joint Conference on Neural Networks (IJCNN). IEEE; 2018. p. 1–8. doi:10.1109/IJCNN.2018.8489520

[7] Kazimipour B, Li X, Qin AK. A review of population initialization techniques for evolutionary algorithms. In: 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE; 2014. p. 2585–2592. doi:10.1109/CEC.2014.6900618

[8] Rahnamayan S, Tizhoosh HR, Salama MMA. A novel population initialization method for accelerating evolutionary algorithms. Computers & Mathematics with Applications. 2007;53(10):1605–1614. doi:10.1016/j.camwa.2006.07.013

[9] Wu Y, Wang H, Zhang B, Du K-L. Using Radial Basis Function Networks for Function Approximation and Classification. ISRN Applied Mathematics. 2012;2012:1–34. doi:10.5402/2012/324194

[10] Poggio T, Girosi F. Networks for approximation and learning. Proceedings of the IEEE. 1990;78(9):1481–1497. doi:10.1109/5.58326

[11] Dataset. [accessed 2024 Dec 15]. https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022

[12] RBF & GA code. [accessed 2024 Dec 15]. https://drive.google.com/drive/folders/1g3W1_fLlangLpfu7n--0sp3u-5Da2y3C?usp=sharing

LAMPIRAN PROGAM

Progam dapat di unduh pada [12]

```
# Label Encoding

payment_type_map = {

    'AA': 0.353846,

    'AB': 0.485255,

    'AC': 0.614198,

    'AD': 0.472222

}

employment_status_map = {

    'CA': 0.526868,

    'CB': 0.333333,

    'CC': 0.728814,

    'CD': 0.266667,

    'CE': 0.333333,

    'CF': 0.181818

}


class DataPreprocessor:

    def __init__(self):

        self.payment_type_map = {}

        self.employment_status_map = {}

        self.means = {}

        self.std_devs = {}


    def fit(self, df):

        # Label Encoding

        self.payment_type_map = df['payment_type'].value_counts(normalize=True).to_dict()

        self.employment_status_map = df['employment_status'].value_counts(normalize=True).to_dict()


        # Hanya fitur numerik tertentu yang belum terscale akan di standardisasi

        for feature in ['credit_risk_score', 'customer_age', 'current_address_months_count']:

            self.means[feature] = df[feature].mean()

            self.std_devs[feature] = df[feature].std()


    def transform(self, df):

        preprocessed_df = df.copy()


        # Label Encoding

        preprocessed_df['payment_type'] = preprocessed_df['payment_type'].map(self.payment_type_map).astype(float)

        preprocessed_df['employment_status'] = preprocessed_df['employment_status'].map(self.employment_status_map).astype(float)


        # Impute Missing Value

preprocessed_df['current_address_months_count'] = preprocessed_df['current_address_months_count'].replace(-1, 71)  # 71 is the median


        # Standard Scaling

        for feature in ['credit_risk_score', 'customer_age', 'current_address_months_count']:

            preprocessed_df[feature] = (preprocessed_df[feature] - self.means[feature]) / self.std_devs[feature]

        return preprocessed_df


    def fit_transform(self, df):

        self.fit(df)

        return self.transform(df)


preprocessor = DataPreprocessor()

preprocessed_train_df = preprocessor.fit_transform(train_df)

preprocessed_test_df = preprocessor.transform(test_df)


X_train = preprocessed_train_df.drop(columns=["fraud_bool"]).values

y_train = preprocessed_train_df["fraud_bool"].values


X_test = preprocessed_test_df.drop(columns=["fraud_bool"]).values

y_test = preprocessed_test_df["fraud_bool"].values



class RBF:

    def __init__(self, spread, centroids, debug=False):

        self.spread = spread

        self.centroids = np.array(centroids)  # Ensure centroids are of proper shape

        self.debug = debug

        self.weights = None

        self.bias = 1


    def _basis_function(self, r):

        """Radial basis function: exp(-r^2 / (2 * spread^2))"""

        return np.exp(-r ** 2 / (2 * self.spread ** 2))


    def _format_matrix_html(self, matrix, title):

        """Format a matrix as an HTML table (for debugging purposes)."""

        table_html = f"<b>{title}:</b><br><table style='border:1px solid black;'>"

        for row in matrix:

            table_html += "<tr>"

            for val in row:

                table_html += f"<td style='border:1px solid black; padding:5px;'>{val:.4f}</td>"

            table_html += "</tr>"

        table_html += "</table><br>"

        return table_html


    def _calculate_outputs(self, X):

        """Calculate the outputs of the hidden layer (RBF layer)."""

        assert X.shape[1] == self.centroids.shape[1], f"Input dimensionality ({X.shape[1]}) must match centroid dimensionality ({self.centroids.shape[1]})"


        # Nilai r (jarak)

        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)

        if self.debug:

display(HTML(self._format_matrix_html(distances, "Distances")))


        # Apply
```

```
        outputs =
self._basis_function(distances)

        if self.debug:


display(HTML(self._format_matrix_html(ou
tputs, "RBF Outputs")))

        return outputs



    def fit(self, X, y):

        """Train the RBF network using the
pseudo-inverse."""

        rbf_outputs =
self._calculate_outputs(X)

        y = y.reshape(-1, 1)

        rbf_outputs = np.hstack((rbf_outputs,
np.ones((rbf_outputs.shape[0], 1))))  # Add
bias +1



        # Calculate weights using pseudo-
inverse

        self.weights =
np.linalg.pinv(rbf_outputs) @ y



        if self.debug:


display(HTML(self._format_matrix_html(se
lf.weights, "Calculated Weights (including
bias)")))



    def predict(self, X):

        """Predict using the trained RBF
network."""

        rbf_outputs =
self._calculate_outputs(X)

        rbf_outputs = np.hstack((rbf_outputs,
np.ones((rbf_outputs.shape[0], 1))))  # Add
bias

        return rbf_outputs @ self.weights



class GeneticAlgorithm:

    def __init__(self, population_size,
num_selection, mutation_rate, exposure_rate,
max_iterations, seed, train_dataset,
test_dataset, debug=True):

        self.population_size = population_size

        self.num_selection = num_selection

        self.mutation_rate = mutation_rate

        self.exposure_rate = exposure_rate

        self.max_iterations = max_iterations

        self.dataset_X_train,
self.dataset_y_train = train_dataset

        self.dataset_X_test, self.dataset_y_test
= test_dataset

        self.population = []

        np.random.seed(seed)

        self.debug = debug  # Enable debug
mode

        self.initialize_population()



    def calculate_fitness(self):

        fitness_values = []

        for centroids, spread in self.population:

            rbf = RBF(spread, centroids,
debug=False)

            rbf.fit(self.dataset_X_train,
self.dataset_y_train)

            predictions =
rbf.predict(self.dataset_X_test)



            predicted_classes = (predictions >=
0.5).astype(int)

            correct_predictions =
sum(predicted_classes.flatten() ==
self.dataset_y_test)

            accuracy = correct_predictions /
len(self.dataset_y_test)



            true_positive =
sum((predicted_classes.flatten() == 1) &
(self.dataset_y_test == 1))

            false_positive =
sum((predicted_classes.flatten() == 1) &
(self.dataset_y_test == 0))

            false_negative =
sum((predicted_classes.flatten() == 0) &
(self.dataset_y_test == 1))



            precision = true_positive /
(true_positive + false_positive) if
(true_positive + false_positive) > 0 else 0

            recall = true_positive / (true_positive
+ false_negative) if (true_positive +
false_negative) > 0 else 0



            f1_score = 2 * (precision * recall) /
(precision + recall) if precision + recall > 0
else 0

            fitness_values.append(f1_score)



        return fitness_values



    def initialize_population(self):

        dataset_X = self.dataset_X_train

        for _ in range(self.population_size):

            num_centroids =
np.random.randint(24, 90)

            centroids =
self._generate_unique_centroids(dataset_X,
num_centroids)

            spread = np.random.uniform(0.01, 2)

            self.population.append((centroids,
spread))



        if self.debug:

            print("Initial Population:")

            for i, (centroids, spread) in
enumerate(self.population):

                print(f"Individual {i + 1}: Number
of Centroids={len(centroids)},
Spread={spread:.4f}")



    def _generate_unique_centroids(self,
dataset_X, num_centroids):

        """Generate unique centroids by
ensuring no duplicates."""

        indices =
np.random.choice(dataset_X.shape[0],
num_centroids, replace=False)

        return dataset_X[indices]



    def select(self, fitness_values):

        total_fitness = sum(fitness_values)

        selection_probabilities = [f /
total_fitness for f in fitness_values]

        cumulative_probabilities =
np.cumsum(selection_probabilities)



        selected_population = []

        for _ in range(self.num_selection):

            random_value = np.random.rand()

            selected_index =
np.searchsorted(cumulative_probabilities,
random_value)


selected_population.append(self.population[
selected_index])



        if self.debug:

            print("Selected Population:")

            for i, (centroids, spread) in
enumerate(selected_population):

                print(f"Selected Individual {i + 1}:
Number of Centroids={len(centroids)},
Spread={spread:.4f}")
```

```
        return selected_population


    def crossover(self, parent1, parent2):
        x = np.random.uniform(0.2, 0.8)
        centroids1, spread1 = parent1
        centroids2, spread2 = parent2

        num_centroids1 = centroids1.shape[0]
        num_centroids2 = centroids2.shape[0]

        # Child 1
        num_child_centroids1_from_parent1 = round(x * num_centroids1)
        num_child_centroids1_from_parent2 = round((1 - x) * num_centroids2)

        selected_indices1 = np.random.choice(num_centroids1, num_child_centroids1_from_parent1, replace=False)
        selected_centroids1 = centroids1[selected_indices1]
        selected_indices2 = np.random.choice(num_centroids2, num_child_centroids1_from_parent2, replace=False)
        selected_centroids2 = centroids2[selected_indices2]
        child1_centroids = np.vstack((selected_centroids1, selected_centroids2))

        # Child 2
        num_child_centroids2_from_parent1 = round((1 - x) * num_centroids1)
        num_child_centroids2_from_parent2 = round(x * num_centroids2)

        selected_indices3 = np.random.choice(num_centroids1, num_child_centroids2_from_parent1, replace=False)
        selected_centroids3 = centroids1[selected_indices3]
        selected_indices4 = np.random.choice(num_centroids2, num_child_centroids2_from_parent2, replace=False)
        selected_centroids4 = centroids2[selected_indices4]
        child2_centroids = np.vstack((selected_centroids3, selected_centroids4))

        # IMPORTANT!! Ensure uniqueness in child centroids
        child1_centroids = self._remove_duplicates(child1_centroids)
        child2_centroids = self._remove_duplicates(child2_centroids)

        if self.debug:
            print(f"Crossover: x={x:.4f}")
            print(f"Parent 1: Number of Centroids={num_centroids1}")
            print(f"Parent 2: Number of Centroids={num_centroids2}")
            print(f"Child 1: Number of Centroids={len(child1_centroids)}")
            print(f"Child 2: Number of Centroids={len(child2_centroids)}")

        return (child1_centroids, spread1), (child2_centroids, spread2)


    def _remove_duplicates(self, centroids):
        """Remove duplicate centroids."""
        unique_centroids = np.unique(centroids, axis=0)
        return unique_centroids[:max(1, len(centroids))] # to avoid 0 centroids


    def mutate(self, chromosome):
        centroids, spread = chromosome
        if np.random.rand() < self.exposure_rate:
            # Original mutation on centroid values
            for i in range(len(centroids)):
                if np.random.rand() < self.mutation_rate:
                    mutation_factor = np.random.uniform(-self.mutation_rate, self.mutation_rate)
                    centroids[i] = centroids[i] * (1 + mutation_factor)

            # Update spread within bounds
            spread += np.random.uniform(-self.mutation_rate, self.mutation_rate)
            spread = np.clip(spread, 0.1, 3)

            # New mutation to modify the number of centroids
            if np.random.rand() < self.mutation_rate:  # Roll to decide if we change centroid count
                # Calculate the percentage change in the number of centroids
                change_percent = np.random.choice([2, 5]) * self.mutation_rate
                target_centroids_count = min(600, max(1, round(len(centroids) * (1 + change_percent))))  # Limit centroids to 500

                if target_centroids_count > len(centroids):
                    # Add new unique centroids if the target count is higher
                    additional_centroids = self._generate_unique_centroids(self.dataset_X_train, target_centroids_count - len(centroids))
                    centroids = np.vstack((centroids, additional_centroids))
                elif target_centroids_count < len(centroids):
                    # Remove random centroids if the target count is lower
                    keep_indices = np.random.choice(len(centroids), target_centroids_count, replace=False)
                    centroids = centroids[keep_indices]

            # Ensure centroids remain unique after mutation
            centroids = self._remove_duplicates(centroids)

            if self.debug:
                print(f"Mutating: New Number of Centroids={len(centroids)}, Spread={spread:.4f}")
        return centroids, spread


    def run(self):
        best_accuracy = 0
        best_parameters = None

        for iteration in range(self.max_iterations):
            fitness_values = self.calculate_fitness()
```

```
        print(f"Iteration {iteration + 1}, Best Fitness: {max(fitness_values)}")



        best_index = np.argmax(fitness_values)

        best_fitness = fitness_values[best_index]

        best_centroids, best_spread = self.population[best_index]



        rbf = RBF(best_spread, best_centroids, debug=False)

        rbf.fit(self.dataset_X_train, self.dataset_y_train)

        predictions_train = rbf.predict(self.dataset_X_train)

        predictions_test = rbf.predict(self.dataset_X_test)



        # Calculate metrics for training data

        predicted_classes_train = (predictions_train >= 0.5).astype(int)

        correct_predictions_train = sum(predicted_classes_train.flatten() == self.dataset_y_train)

        train_accuracy = correct_predictions_train / len(self.dataset_y_train)



        # Calculate metrics for test data

        predicted_classes_test = (predictions_test >= 0.5).astype(int)

        correct_predictions_test = sum(predicted_classes_test.flatten() == self.dataset_y_test)

        test_accuracy = correct_predictions_test / len(self.dataset_y_test)



        # Calculate F1 Score for test data

        true_positive = sum((predicted_classes_test.flatten() == 1) & (self.dataset_y_test == 1))

        false_positive = sum((predicted_classes_test.flatten() == 1) & (self.dataset_y_test == 0))

        false_negative = sum((predicted_classes_test.flatten() == 0) & (self.dataset_y_test == 1))



        precision = true_positive / (true_positive + false_positive) if (true_positive + false_positive) > 0 else 0

        recall = true_positive / (true_positive + false_negative) if (true_positive + false_negative) > 0 else 0

        f1_score = 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0



        print(f"Iteration {iteration + 1}:")

        print(f" - RBF Parameters: Number of Centroids={len(best_centroids)}, Spread={best_spread:.4f}")

        print(f" - Train Accuracy: {train_accuracy:.4f}, Test Accuracy: {test_accuracy:.4f}, F1 Score: {f1_score:.4f}")



        if test_accuracy > best_accuracy:

            best_accuracy = test_accuracy

            best_parameters = {

              "num_centroids": len(best_centroids),

                "spread": best_spread

            }



        selected_population = self.select(fitness_values)

        new_population = []



        while len(new_population) < self.population_size:

            parent1, parent2 = random.sample(selected_population, 2)

            child1, child2 = self.crossover(parent1, parent2)

            mutated_child1 = self.mutate(child1)

            mutated_child2 = self.mutate(child2)


new_population.extend([mutated_child1, mutated_child2])



        # Ensure the new population doesn't exceed population_size

        self.population = new_population[:self.population_size]



    print(f"Best Test Accuracy: {best_accuracy:.4f}")

    print(f"Best Parameters: Number of Centroids = {best_parameters['num_centroids']}, Spread = {best_parameters['spread']}")
```

```
# Parameters for the Genetic Algorithm

population_size = 500  # jumlah populasi awal

num_selection = 250    # Jumalah seleksi populasi, rekomendasi 50% populasi awal

mutation_rate = 0.4    # Probabilitas centroid mutasi

exposure_rate = 0.25   # Probabilitas chromosome terkena mutasi

max_iterations = 9     # Iterasi

seed = 2024          # Seed Random


ga = GeneticAlgorithm(

    population_size=population_size,

    num_selection=num_selection,

    mutation_rate=mutation_rate,

    exposure_rate=exposure_rate,

    max_iterations=max_iterations,

    seed=seed,

    train_dataset=(X_train, y_train),

    test_dataset=(X_test, y_test),

    debug=False

)


ga.run()
```