LAPORAN AKHIR
JARINGAN SYARAF TIRUAN

IMPLEMENTASI BPNN DALAM KASUS KLASIFIKASI BINER *DEFAULT CREDIT CARD*

Disusun Oleh:
Qolbu Salim NIM. 23031030020

Dosen Pengampu:
Nur Insani, Ph.D.

PROGRAM STUDI STATISTIKA
DEPARTEMEN PENDIDIKAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI YOGYAKARTA
2025

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Risiko kredit adalah salah satu aspek paling krusial di industri keuangan modern, terutama bagi lembaga pemberi kredit seperti bank dan penyedia kartu kredit. Salah satu risiko utama yang harus dikendalikan adalah gagal bayar oleh nasabah. Kemampuan untuk mengidentifikasi secara tepat nasabah yang berpotensi mengalami gagal bayar memungkinkan pihak kreditur, seperti bank dan penyedia kartu kredit, untuk melakukan langkah mitigasi yang lebih efektif, seperti penyesuaian limit kredit, penetapan bunga yang sesuai dengan profil risiko, hingga penyusunan strategi pengelolaan portofolio yang lebih aman. Deteksi dini ini tidak hanya mengurangi potensi kerugian finansial, tetapi juga berperan penting dalam menjaga kesehatan sistem perbankan secara keseluruhan.

Salah satu dataset yang banyak digunakan dalam penelitian mengenai prediksi gagal bayar adalah *Default Payment Next Month* dari UCI Machine Learning Repository. Dataset ini berisi data 30.000 pelanggan kartu kredit di Taiwan dengan 23 variabel prediktor yang meliputi informasi demografis, status pembayaran, serta perilaku finansial nasabah. Variabel target pada dataset ini bersifat biner, yaitu menunjukkan apakah seorang nasabah mengalami gagal bayar pada bulan berikutnya atau tidak. Dataset ini menjadi benchmark penting dalam studi credit scoring dan prediksi default karena kompleksitas serta representasinya terhadap kondisi nyata dalam industri keuangan.

Dalam upaya meningkatkan akurasi deteksi dini gagal bayar, metode berbasis machine learning telah banyak diterapkan. Di antara berbagai pendekatan yang tersedia, Jaringan Saraf Tiruan (JST) menunjukkan kemampuan unggul dalam menangkap hubungan non-linear yang kompleks antar variabel *input*. Tidak seperti model linier tradisional, JST mampu mempelajari pola tersembunyi dalam data yang sering kali tidak dapat dijelaskan oleh model statistik konvensional. Dengan adanya mekanisme *feedforward* dan *backpropagation*, JST dapat secara iteratif menyesuaikan bobot dan bias untuk meminimalkan kesalahan prediksi, sehingga menghasilkan model yang lebih adaptif dan akurat.

Selain itu, perkembangan metode modern seperti *batch normalization* dan *dropout* semakin memperkuat kinerja JST dengan mengatasi masalah *overfitting* serta mempercepat konvergensi model. Penerapan teknik ini sangat relevan untuk dataset seperti *Default Payment Next Month* yang memiliki distribusi kelas tidak seimbang dan dimensi fitur yang tinggi. Oleh karena itu, kombinasi antara arsitektur JST yang dalam (*deep neural network*) dan strategi regularisasi yang tepat dapat meningkatkan efektivitas deteksi dini risiko kredit secara signifikan.

Dengan latar belakang tersebut, penelitian ini difokuskan untuk membangun dan menganalisis model BPNN yang mampu melakukan klasifikasi biner *default* dan *non-default* pada dataset *Default Payment Next Month* secara optimal. Penelitian ini diharapkan dapat memberikan kontribusi terhadap pengembangan sistem deteksi dini risiko kredit berbasis kecerdasan buatan, yang tidak hanya meningkatkan ketepatan prediksi, tetapi juga memperkuat landasan ilmiah bagi penerapan teknologi *deep learning* dalam bidang analisis keuangan.

1.2 Tujuan

1. Pengembangan dan evaluasi model BPNN untuk memprediksi *default payment next month*.
2. Mengaplikasikan jaringan syaraf tiruan pada bidang keuangan.
3. Menghasilkan sebuah model prediktif yang andal sebagai alat bantu pengambilan keputusan untuk mitigasi risiko kredit bagi lembaga keuangan.

1.3 Metodologi dan Deskripsi Dataset

1.3.1 Dataset

Dataset yang digunakan pada penelitian ini adalah data sekunder *Default of Credit Card Clients* yang bersumber dari University of California Irvine Machine Learning Repository. Dataset ini berisi kasus keterlambatan pembayaran kartu kredit pelanggan di Taiwan. Variabel dependen pada dataset ini bersifat biner, dengan ketarangan 1 melambangkan pembayaran *default* dan 0 melambangkan pembayaran yang tidak *default* atau terlambat. Data ini terdiri dari 24 kolom dan 30.000 baris. Kolom pada data ini terdiri dari 23 variabel prediktor dan 1 variabel respon.

1.3.2 *Preprocessing*

1) Penanganan *Missing Value*
2) Penanganan *Outlier*
3) *Feature Selection*
4) Pembagian Data
5) *Class Weighting*

1.3.3 Metode Analisis

Backpropagation Neural Network (BPNN) Adalah salah satu model *deep learning* yang meniru kemampuan dari jaringan syaraf biologis (Li, 2024). Pertama kali dikenalkan pada tahun 1980, model ini dengan cepat menjadi fokus utama dalam berbagai penelitian karena kemampuannya dalam mempelajari data dan adaptabilitas yang tinggi. Model jaringan syaraf tiruan ini terdiri dari *input layer*, *hidden layer*, *outpur layer*, dan optimalisasi bobot menggunakan algoritma *backpropagation*.

Algoritma *Backpropagation* (*Backward Propagation of Errors*) merupakan salah satu metode yang paling banyak digunakan dalam proses pelatihan JST. Tujuan utamanya adalah menyesuaikan bobot dan bias pada setiap

neuron sehingga jaringan mampu mempelajari pola dari data pelatihan serta menghasilkan prediksi yang akurat terhadap data baru yang belum pernah dikenali sebelumnya. Secara umum, proses pelatihan jaringan saraf dengan algoritma *backpropagation* melibatkan dua tahap utama:

1) *Feedforward* (Propagasi Maju)

   Pada tahap ini, data masukan (*input data*) dimasukkan ke jaringan dan dipropagasikan ke lapisan-lapisan berikutnya hingga menghasilkan keluaran (*output*). Setiap neuron melakukan perhitungan menggunakan bobot dan bias yang ada, lalu hasil akhirnya dibandingkan dengan target sebenarnya untuk menghitung galat (*error*).

2) *Backward Propagation of Errors* (Propagasi Balik Kesalahan)

   Setelah galat dihitung, algoritma melakukan proses mundur dari lapisan *output* ke lapisan *input* untuk menyesuaikan bobot dan bias. Penyesuaian ini dilakukan menggunakan turunan (gradien) dari fungsi aktivasi terhadap bobot, berdasarkan prinsip *Gradient Descent*. Tujuannya adalah meminimalkan fungsi kesalahan (*loss function*) secara iteratif hingga model mencapai konvergensi.

Diperlukan penjabaran lebih lanjut mengenai tahapan pelatihan yang dilakukan oleh jaringan saraf tiruan tersebut. Algoritma *backpropagation* berfungsi untuk mengoptimalkan bobot dan bias melalui proses pembelajaran yang bersifat iteratif, dengan tujuan meminimalkan nilai galat antara keluaran prediksi dan nilai target. Proses ini dilakukan secara sistematis dengan mempropagasikan sinyal dari lapisan *input* menuju lapisan *output*, kemudian menghitung dan menyebarkan kembali kesalahan ke arah berlawanan untuk memperbarui parameter jaringan. Dengan demikian, jaringan saraf dapat secara bertahap menyesuaikan parameter internalnya agar mampu menghasilkan prediksi yang semakin akurat. Tahapan pelatihan menggunakan algoritma *backpropagation* secara lengkap dapat dijelaskan sebagai berikut.

- Step 0: Inisialisasi Bobot dan Bias

  Pada tahap awal, jaringan saraf tiruan diinisialisasi dengan memberikan nilai awal secara acak pada bobot dan bias. Nilai-nilai ini umumnya berada di sekitar nol (misalnya antara -1 hingga 1) agar proses pembelajaran dapat berlangsung secara optimal. Pemilihan nilai acak ini penting untuk mencegah setiap neuron mempelajari pola yang sama, sehingga jaringan dapat melakukan pembelajaran secara efektif.

- Step 1: Pemeriksaan Kondisi Berhenti (*Stopping Condition*)

  Sebelum proses pelatihan dimulai, ditetapkan kriteria penghentian (*stopping condition*) sebagai acuan kapan pelatihan harus dihentikan. Kriteria ini dapat berupa jumlah iterasi maksimum (*epochs*) yang telah tercapai, atau ketika perbedaan nilai galat (*error*) antara dua iterasi berturut-turut berada di bawah ambang batas tertentu. Pemeriksaan ini bertujuan untuk memastikan bahwa

model tidak berlatih secara berlebihan (*overfitting*) dan proses pembelajaran berhenti pada waktu yang tepat.

- Step 2-9: Proses Pelatihan Jaringan

  Langkah 2 hingga 9 merupakan proses iteratif yang dilakukan untuk setiap data pelatihan hingga kondisi berhenti terpenuhi. Proses ini mencakup tahap propagasi maju, perhitungan kesalahan, propagasi balik, serta pembaruan bobot dan bias.

- Step 3: Propagasi Maju (*Feedforward*)

  Pada tahap ini, data masukan dialirkan melalui seluruh lapisan jaringan, dimulai dari lapisan input menuju lapisan tersembunyi (*hidden layer*) hingga lapisan *output*. Setiap neuron menghitung nilai keluarannya berdasarkan kombinasi linier dari masukan yang telah dikalikan bobot, kemudian diproses melalui fungsi aktivasi yang telah ditentukan. Hasil akhir dari proses ini adalah nilai keluaran (*output*) jaringan yang akan dibandingkan dengan target sebenarnya.

- Step 4: Perhitungan *Output* pada Lapisan Tersembunyi (*Hidden Layer Output*)

  Nilai keluaran dari setiap neuron pada lapisan tersembunyi dihitung dengan menjumlahkan seluruh masukan yang telah diberi bobot, kemudian menerapkannya ke dalam fungsi aktivasi. Fungsi aktivasi ini berperan penting dalam memperkenalkan sifat non-linear ke dalam model, sehingga jaringan mampu mempelajari hubungan yang kompleks antara variabel input dan output.

- Step 5: Perhitungan *Output* pada Lapisan Keluaran (*Output Layer Output*)

  Tahap ini dilakukan dengan cara serupa seperti pada lapisan tersembunyi. Nilai keluaran akhir jaringan dihitung berdasarkan masukan dari lapisan tersembunyi yang telah dikalikan dengan bobot menuju lapisan output. Hasil perhitungan ini menjadi prediksi jaringan terhadap data pelatihan.

- Step 6: Propagasi Balik Kesalahan (*Backpropagation of Error*)

  Setelah memperoleh keluaran jaringan, kesalahan atau galat dihitung sebagai selisih antara nilai target dan nilai prediksi. Nilai kesalahan ini kemudian digunakan untuk menghitung gradien atau turunan dari fungsi kesalahan terhadap bobot di lapisan *output*. Tujuannya adalah untuk mengetahui arah perubahan bobot yang dapat meminimalkan kesalahan.

- Step 7: Propagasi Balik ke Lapisan Tersembunyi

  Kesalahan yang telah dihitung pada lapisan *output* kemudian dipropagasikan ke lapisan tersembunyi. Pada tahap ini, setiap neuron di lapisan tersembunyi menghitung kontribusinya terhadap kesalahan total jaringan. Nilai gradien pada lapisan tersembunyi digunakan untuk menentukan besarnya pembaruan bobot antara lapisan tersembunyi dan lapisan *input*.

- Step 8: Pembaruan Bobot dan Bias

Bobot dan bias diperbarui berdasarkan gradien kesalahan yang telah dihitung menggunakan metode *Gradient Descent* atau variasinya (misalnya *Stochastic Gradient Descent*, *Adam Optimizer*). Pembaruan ini dilakukan dengan menyesuaikan nilai bobot dan bias ke arah yang dapat meminimalkan fungsi kesalahan. Secara umum, rumus pembaruan bobot dituliskan sebagai berikut:

$$w_{baru} = w_{lama} - \alpha \frac{\partial E}{\partial w}$$

Dimana:
α adalah *learning rate*;
$\frac{\partial E}{\partial w}$ adalah turunan fungsi kesalahan terhadap bobot.

- Step 9: Pemeriksaan Kembali Kondisi Berhenti

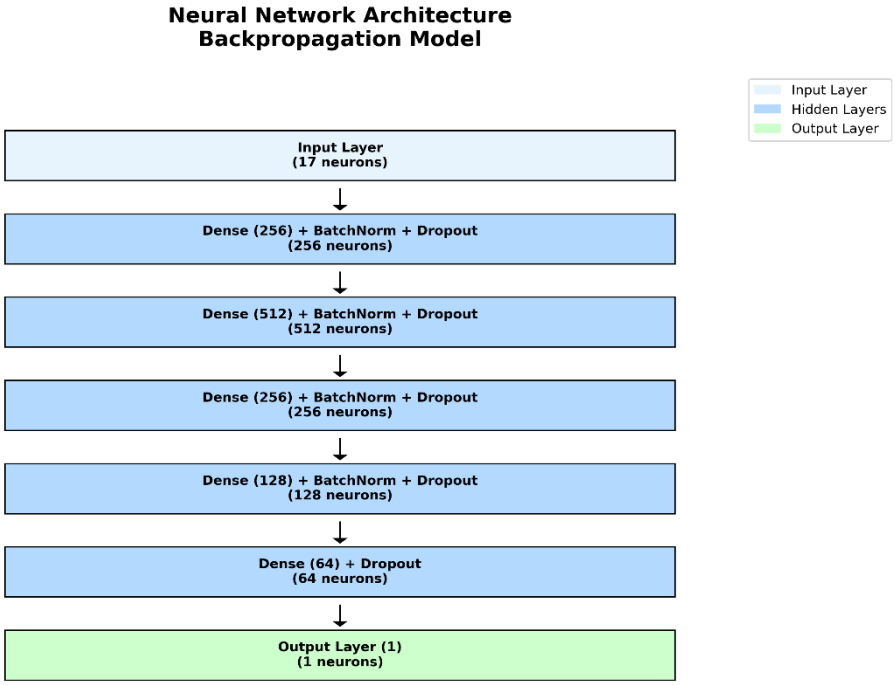
  Setelah satu siklus pelatihan selesai, sistem memeriksa kembali apakah kondisi berhenti telah terpenuhi. Jika belum, proses pelatihan diulang kembali mulai dari langkah *feedforward* (Step 3). Namun, apabila kondisi berhenti telah tercapai, maka proses pelatihan dihentikan.

BAB 2
KAJIAN TEORI

2.1 Jenis dan Arsitektur Model

Arsitektur model BPNN yang digunakan dalam penelitian ini terdiri atas beberapa lapisan yang tersusun secara berurutan dan saling terhubung (*Fully Connected*). Model dibangun menggunakan pendekatan Sequential dari Keras, yang memungkinkan pembentukan jaringan saraf secara berlapis dari *input* hingga *output*. Secara umum, struktur jaringan terdiri atas satu *input layer* dengan 256 neuron yang menerima fitur hasil *preprocessing*, diikuti oleh empat *hidden layer* dengan jumlah neuron berturut-turut 512, 256, 128, dan 64. Setiap lapisan tersembunyi menggunakan fungsi aktivasi ReLU (*Rectified Linear Unit*) untuk menangani hubungan non-linear dalam data. Untuk meningkatkan stabilitas pembelajaran dan mencegah *overfitting*, diterapkan *Batch Normalization* setelah setiap lapisan serta *Dropout* dengan tingkat 0,3 sebagai mekanisme regularisasi. Bagian *output layer* menggunakan satu neuron dengan fungsi aktivasi sigmoid yang sesuai untuk permasalahan klasifikasi biner, yaitu memprediksi kemungkinan *default* atau *non-default*. Struktur keseluruhan model BPNN yang diimplementasikan dapat dilihat pada Gambar 1,

Gambar 1. Arsitektur Model



2.2 Parameter Model

2.2.1 *Activation Function*

Pada penelitian ini digunakan 2 jenis *activation function*, yakni ReLU dan sigmoid. Pada hidden layer digunakan fungsi aktivasi ReLU karena kemampuannya untuk mengatasi masalah gradien yang menghilang (*vanishing gradient problem*). Tidak seperti *activation function* lain, turunan ReLU adalah 1 untuk semua *input* positif. Hal ini memungkinkan gradien untuk mengalir lebih baik melalui lapisan-lapisan yang dalam pada jaringan saraf, sehingga proses pelatihan menjadi lebih cepat dan efektif.

$$ReLU(x) = \max(0, x)$$

Sedangkan pada output layer digunakan *activation function* sigmoid karena masalah yang diselesaikan adalah klasifikasi biner (gagal bayar atau tidak). Fungsi Sigmoid memetakan output apa pun ke rentang antara 0 dan 1. Output ini dapat diinterpretasikan secara langsung sebagai probabilitas. Sebagai contoh, output 0.8 dapat diartikan sebagai probabilitas 80% bahwa seorang nasabah akan mengalami gagal bayar.

$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

2.2.2 *Regularization*

Regularisasi adalah kunci untuk membuat model yang dapat bergeneralisasi dengan baik pada data baru. Model ini menggunakan dua teknik utama, yakni *Batch Normalization* dan *Dropout*. Dimana Batch normalization akan menstabilkan dan mempercepat proses pelatihan dengan menormalisasi *output*

dari lapisan sebelumnya. Ini juga memberikan efek regularisasi ringan yang membantu model menjadi lebih *robust*. Sedangkan Dropout secara acak menonaktifkan sebagian neuron selama pelatihan. Teknik ini memaksa jaringan untuk belajar fitur yang lebih kuat dan tidak terlalu bergantung pada neuron tunggal, sehingga sangat efektif dalam mencegah *overfitting*.

2.2.3 *Optmizers*

Pada model BPNN ini menggunakan Adam (*Adaptive Moment Estimation*) sebagai *optimizer*. Adam adalah algoritma optimisasi yang sangat populer dan sering menjadi pilihan utama karena menggabungkan keunggulan dari dua algoritma lain, yakni AdaGrad dan RMSProp. Secara ilmiah, Adam unggul karena laju pembelajaran adaptif. Adam menghitung laju pembelajaran (*learning rate*) yang berbeda dan adaptif untuk setiap parameter (bobot) dalam model. Serta Adam menggunakan momentum (reta-rata bergerak dari gradien) untuk mempercepat konvergensi dan membantu melewati titik *saddle points* atau minimum lokal. Kombinasi ini membuat Adam sangat efisien, membutuhkan lebih sedikit penyetelan manual pada learning rate, dan sering kali mencapai hasil yang baik dengan cepat.

2.2.4 *Training*

Proses *training* (pelatihan) pada dasarnya adalah saat model belajar dari data. Proses ini tidak terjadi sekaligus, melainkan secara *iteratif* (berulang-ulang) untuk secara bertahap menyesuaikan bobot internalnya agar dapat membuat prediksi yang akurat. Jumlah iterasi pada pelatihan ini ditentukan oleh *epochs* (siklus belajar penuh). *Epochs* adalah *hyperparameter* yang menentukan berapa kali keseluruhan dataset pelatihan akan "dilihat" oleh model selama proses belajar. Satu epoch berarti setiap sampel dalam data latih telah melewati proses *forward propagation* (membuat prediksi) dan *backward propagation* (memperbarui bobot) satu kali. Pada model ini digunakan parameter epochs=200, yang berarti model akan mengulang proses belajar pada keseluruhan data latih sebanyak 200 kali.

Parameter *training* selanjutnya yakni *batch size*. Parameter ini digunakan untuk menentukan jumlah sampel data yang diproses oleh model sebelum bobotnya diperbarui. Dataset pelatihan yang besar tidak dimasukkan ke dalam model sekaligus karena keterbatasan memori. Sebaliknya, data tersebut dipecah menjadi beberapa "*batch*" atau kelompok kecil. Pada model ini digunakan batch_size=32, yang berarti model akan mengambil 32 sampel data dari *dataset* pelatihan, memprosesnya, menghitung *error*, dan memperbarui bobotnya. Setelah itu, model akan mengambil 32 sampel data berikutnya, dan begitu seterusnya hingga seluruh data dalam *dataset* selesai diproses.

2.2.5 *Loss Function*

Loss function untuk mengukur kesalahan pada model ini adalah *Weighted Binary Cross-Entropy*. Versi yang diberi bobot (*weighted*) digunakan untuk menangani masalah ketidakseimbangan kelas dalam dataset, di mana jumlah nasabah yang gagal bayar jauh lebih sedikit daripada yang tidak.

2.2.6 *Callbacks*

Callbacks berlaku sebagai "asisten" cerdas yang memantau dan mengambil tindakan pada berbagai tahap selama pelatihan. Pada model ini digunakan 3 *callbacks*.

1) *EarlyStopping*

   Callback EarlyStopping berfungsi sebagai mekanisme pengaman untuk mencegah *overfitting* dan meningkatkan efisiensi dengan menghentikan proses pelatihan secara otomatis. Ia bekerja dengan terus memantau metrik performa kunci, yang dalam hal ini diatur melalui monitor='val_accuracy' untuk mengawasi akurasi pada data validasi. Parameter patience=20 memberikan "kesabaran" pada model, di mana pelatihan akan dihentikan jika akurasi validasi tidak menunjukkan peningkatan (mode='max') selama 20 *epoch* berturut-turut. Jika pelatihan dihentikan, restore_best_weights=True memastikan bahwa bobot model yang dikembalikan adalah bobot dari epoch dengan performa terbaik, bukan dari *epoch* terakhir. Terakhir, verbose=1 akan memberikan notifikasi di layar ketika mekanisme penghentian dini ini diaktifkan.

2) *ModelCheckpoint*

   ModelCheckpoint bertugas untuk menyimpan versi terbaik dari model ke sebuah file selama proses pelatihan. Dengan menggunakan monitor='val_accuracy' dan mode='max', *callback* ini akan selalu mengevaluasi akurasi pada data validasi di setiap akhir *epoch*. Parameter save_best_only=True adalah instruksi krusial yang memastikan bahwa file best_model.h5 hanya akan ditimpa ketika model pada *epoch* saat ini memiliki akurasi validasi yang lebih tinggi daripada versi yang tersimpan sebelumnya. Dengan verbose=1, *output* akan memberikan pemberitahuan setiap kali model berhasil disimpan, sehingga kita dapat melacak kemajuan dan memastikan bahwa salinan model yang paling optimal selalu tersedia.

3) *ReduceLROnPlateau*

   Callback ini berfungsi sebagai pengatur *learning rate* yang dinamis. *ReduceLROnPlateau* akan memantau metrik val_loss, yaitu nilai *loss* pada data validasi dan akan mengambil tindakan jika tidak ada perbaikan. Jika val_loss tidak menurun selama patience=10 *epoch*, *callback* ini akan mengurangi *learning rate* dengan mengalikannya dengan factor=0.3. Proses ini membantu model untuk "melangkah" lebih hati-hati saat mendekati titik optimal. Pengurangan ini akan terus terjadi hingga *learning rate* mencapai batas bawah yang ditentukan oleh min_lr=1e-7. Setelah laju pembelajaran dikurangi, cooldown=5 akan memberlakukan jeda selama 5 epoch sebelum *callback* kembali aktif memantau, memberikan waktu bagi model untuk beradaptasi. Notifikasi akan ditampilkan saat laju pembelajaran disesuaikan berkat pengaturan verbose=1.

2.3 Justifikasi Teknik

2.3.1 Justifikasi Model

Pemilihan JST sebagai model utama didasarkan pada kemampuannya untuk menangani masalah yang kompleks dan non-linear, seperti prediksi gagal bayar kartu kredit. Terdapat tiga poin mengapa digunakan JST sebagai model.

1) Kemampuan Menangkap Pola Kompleks

   Data keuangan seringkali memiliki hubungan yang rumit dan tidak dapat dipisahkan secara linear. JST dengan beberapa lapisan tersembunyi (*hidden layers*) sangat baik dalam mempelajari interaksi non-linear antara berbagai fitur (seperti limit kredit, riwayat pembayaran, dan jumlah tagihan) untuk membuat prediksi yang lebih akurat dibandingkan model linear tradisional.

2) Fleksibilitas Arsitektur

   JST memungkinkan fleksibilitas tinggi dalam merancang arsitektur model. Seperti yang terlihat di notebook, kita dapat dengan mudah menambahkan lapisan, mengubah jumlah neuron, dan menerapkan teknik regularisasi (*Dropout* dan *Batch Normalization*) untuk menyesuaikan kompleksitas model dengan data dan mencegah *overfitting*.

3) Algoritma *Backpropagation*

   Backpropagation merupakan algoritma fundamental untuk melatih JST. Algoritma ini secara efisien menghitung gradien dari *loss function* dan memperbarui bobot di seluruh jaringan, memungkinkan model untuk belajar dari kesalahannya secara iteratif dan bertahap meningkatkan akurasinya. Ini adalah standar emas untuk melatih *deep neural networks*.

2.3.2 Justifikasi *Preprocessing*

Preprocessing adalah langkah krusial untuk membersihkan dan menyiapkan data agar model dapat belajar secara efektif. Setiap teknik yang dipilih dalam notebook ini memiliki tujuan spesifik untuk mengatasi masalah umum dalam dataset dunia nyata.

1) Penanganan Nilai Tidak Valid dan Imputasi

   Dataset memiliki nilai pada kolom pendidikan ('X3') dan status perkawinan ('X4') yang tidak sesuai dengan deskripsi. Langkah pertama adalah mengidentifikasi dan mengubah nilai-nilai anomali ini menjadi NaN (kosong). Teknik SimpleImputer dengan strategi most_frequent (modus) kemudian digunakan untuk mengisi nilai-nilai kosong ini. Pilihan ini masuk akal karena pendidikan dan status perkawinan adalah data kategorikal, di mana mengisi dengan nilai yang paling sering muncul adalah pendekatan yang aman dan logis.

2) Penskalaan Fitur dengan RobustScaler

   EDA melalui boxplot menunjukkan adanya pencilan (*outliers*) pada beberapa fitur numerik. Outliers dapat secara signifikan memengaruhi performa model JST. RobustScaler dipilih karena tahan terhadap outliers.

Tidak seperti StandardScaler yang menggunakan rata-rata dan deviasi standar, RobustScaler menggunakan median dan rentang interkuartil (IQR) untuk menskalakan data. Ini memastikan bahwa nilai-nilai ekstrim tidak mendominasi proses pelatihan.

3) Seleksi Fitur dengan SelectKBest

Tidak semua fitur dalam dataset mungkin relevan untuk memprediksi gagal bayar. Menggunakan fitur yang tidak relevan dapat menambah noise dan kompleksitas yang tidak perlu pada model. Teknik SelectKBest dengan f_classif (ANOVA F-test) digunakan untuk memilih fitur-fitur yang memiliki hubungan statistik paling signifikan dengan variabel target. Ini membantu menyederhanakan model, mengurangi waktu pelatihan, dan berpotensi meningkatkan performa dengan fokus pada prediktor yang paling kuat.

4) Penanganan Kelas Tidak Seimbang

Untuk menangani masalah ketidakseimbangan kelas yang signifikan dalam dataset, di mana jumlah sampel untuk kelas mayoritas ('*No Default'*) jauh lebih besar daripada kelas minoritas ('*Default'*), penelitian ini menerapkan metode *cost-sensitive learning* melalui pembobotan kelas (*class weighting*). Pendekatan ini dipilih karena kemampuannya untuk menyeimbangkan pengaruh setiap kelas tanpa mengubah distribusi data asli melalui *oversampling* atau *undersampling*. Dengan menggunakan fungsi compute_class_weight dengan parameter 'balanced', bobot untuk setiap kelas dihitung secara otomatis berbanding terbalik dengan frekuensinya. Akibatnya, kelas minoritas ('*Default*') diberikan bobot yang lebih tinggi, sementara kelas mayoritas (*'No Default*') menerima bobot yang lebih rendah. Bobot ini kemudian diintegrasikan ke dalam *loss function* selama proses pelatihan, sehingga setiap kesalahan klasifikasi pada sampel kelas minoritas akan dikenakan pinalti yang lebih besar. Hal ini secara efektif memaksa model untuk memberikan perhatian lebih pada kelas minoritas yang lebih sulit dipelajari, mendorongnya untuk membangun batas keputusan yang lebih adil dan meningkatkan kemampuannya dalam mengidentifikasi kasus gagal bayar yang krusial.

BAB 3
ISI PROGRAM DAN LANGKAH PENGERJAAN

3.1 Struktur dan Dokumentasi

Kode program untuk penelitian ini dikembangkan dengan mengikuti prinsip-prinsip kode yang bersih, terstruktur, dan terdokumentasi. Setiap blok kode diberikan komentar yang relevan untuk menjelaskan fungsionalitas, alur logika, dan tujuan dari setiap tahapan, mulai dari pra-pemrosesan data hingga evaluasi model. Hal ini bertujuan untuk memastikan keterbacaan (*readability*), kemudahan pemeliharaan

(*maintainability*), dan memfasilitasi proses reproduksi hasil penelitian (*reproducibility*).

Untuk mendukung transparansi dan kolaborasi, keseluruhan *source code* beserta aset terkait disimpan dan dikelola menggunakan sistem kontrol versi Git dan dihosting secara publik pada sebuah repositori GitHub. Dokumentasi lengkap mengenai cara menjalankan program, dependensi *library* yang dibutuhkan, serta panduan langkah demi langkah untuk mengeksekusi *notebook* dari awal hingga akhir telah disediakan dalam file README.md pada repositori tersebut. Repositori proyek dapat diakses melalui tautan berikut: https://github.com/qlbusalim/Binary-Classification-Using-NN.

3.2 Alur Logika Implementasi

3.2.1 *Data Preparation*

Tahap awal pada penelitian ini adalah *data preparation* atau persiapan data. Pada tahap ini dilakukan pengecekan pada data secara secara visual dengan tabel. Dijumpai beberapa variabel yang memiliki data diluar indeks yang sudah ditentukan, yakni pada variabel X3 dan X4. Pada X3 data memiliki 4 kategori dari 1-4, sedangkan pada data ditemukan nilai 0, 5, dan 6. Sedangkan pada variabel X4 terdiri dari 3 kategori, yakni 1-3, sedangkan pada data ditemukan nilai 0. Nilai yang tidak sesuai dengan indeks ini diganti menjadi *missing value*. Dihasilkan 345 baris data NaN pada variabel X3 dan 54 baris data NaN pada variabel X4.

3.2.2 EDA

Dilakukan *Exploratory Data Analysis* (EDA) dimana data dieksplor untuk mencari temuan terkait data yang akan berguna pada saat modelling. Pada tahap EDA ini dilakukan pengecekan kualitas data dengan melihat jumlah *missing value*, data duplikat, dan juga kesesuaian nilai pada data. Selain mengecek kualitas data, dilakukan juga visualisasi data untuk melihat temuan menarik baik pada masing-masing variabel maupun antar variabel. Visualisasi ini meliputi histogram untuk melihat sebaran data numerik, *bar plot* untuk melihat sebaran data kategorik, *boxplot* untuk mengecek *outlier* pada data numerik, dan *correlation plot* untuk melihat hubungan antar data berdasarkan nilai korelasi.

3.2.3 *Preprocessing*

Setelah data dieksplor pada EDA, selanjutnya temuan pada data akan diolah untuk lebih sesuai dengan model. Pada proses ini dilakukan serangkaian langkah *preprocessing* untuk meningkatkan kualitas data dan performa model. Pertama, imputasi *missing value* dilakukan pada variabel X3 dan X4 menggunakan SimpleImputer dengan strategi modus (*most frequent*) untuk mengisi nilai-nilai yang sebelumnya diidentifikasi sebagai anomali. Kedua, untuk mengatasi adanya *outlier*, seluruh fitur numerik diskala menggunakan RobustScaler yang tahan terhadap nilai-nilai ekstrim. Ketiga, dilakukan seleksi fitur menggunakan SelectKBest dengan metode ANOVA F-test (f_classif) untuk

memilih fitur-fitur yang paling signifikan secara statistik terhadap variabel target, sehingga model dapat fokus pada prediktor yang paling kuat. Terakhir, untuk menangani masalah ketidakseimbangan kelas, diterapkan metode *cost-sensitive learning* dengan memberikan pembobotan kelas (*class weighting*), di mana kelas minoritas (gagal bayar) diberi bobot pinalti yang lebih tinggi selama pelatihan untuk memastikan model memberikan perhatian yang seimbang pada kedua kelas.

3.2.4 *Modelling*

Setelah data diolah agar lebih sesuai untuk dipelajari oleh model, dilakukan proses *modelling*. Pada proses ini, disusun satu model BPNN (*Backpropagation Neural Network*) menggunakan arsitektur sekuensial. Model ini terdiri dari satu *input layer*, beberapa *hidden layers* dengan fungsi aktivasi ReLU, dan satu *output layer* dengan fungsi aktivasi Sigmoid yang cocok untuk masalah klasifikasi biner. Untuk mencegah *overfitting* dan menstabilkan pelatihan, arsitektur ini juga dilengkapi dengan lapisan BatchNormalization dan Dropout. Model kemudian dikompilasi dengan *optimizer* Adam, *loss function* Binary Cross-Entropy yang telah diberi bobot sesuai dengan distribusi kelas, dan metrik evaluasi utama berupa akurasi. Proses pelatihan dikontrol secara cermat menggunakan serangkaian *callbacks*, termasuk EarlyStopping untuk menghentikan pelatihan saat tidak ada kemajuan, ModelCheckpoint untuk menyimpan versi model terbaik, dan ReduceLROnPlateau untuk menyesuaikan laju pembelajaran secara dinamis.

3.2.5 Evaluasi

Setelah model dilatih, ketepatan prediksi model akan dites menggunakan data validasi yang berasal dari pembagian data pada saat *preprocessing*. Pengukuran ketepatan dilakukan untuk melihat performa klasifikasi yang telah dilakukan (Heryadi, dkk. 2020). Dalam mengukur ketepatan klasifikasi, perlu diketahui jumlah data pada setiap kelas aktual yang terdiri dari TP (*True Positive*) yaitu label positif yang tepat diklasifikasi kedalam kelas positif, TN (*True Negative*) yaitu label yang tepat diklasifikasi dalam kelas negatif, FP (*False Positive*) adalah label negatif yang terklasifikasi kedalam kelas positif, dan FN (*False Negative*) yaitu label positif yang terklasifikasi kedalam kelas negatif (Chawla, dkk. 2002). Pada penelitian ini menggunakan F1-Score.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)}$$

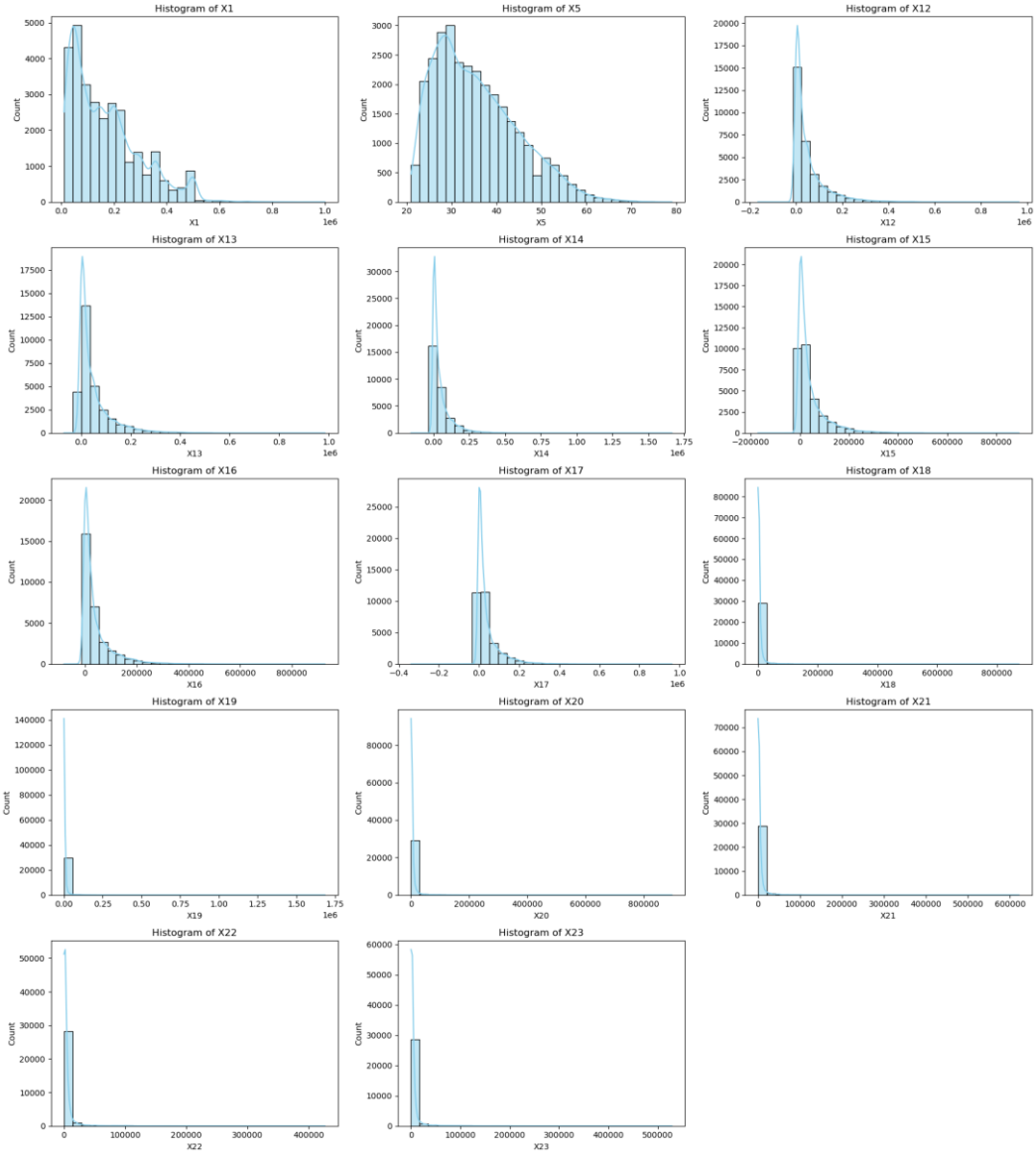
$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

BAB 4
HASIL DAN PEMBAHASAN

4.1 Visualisasi dan Output

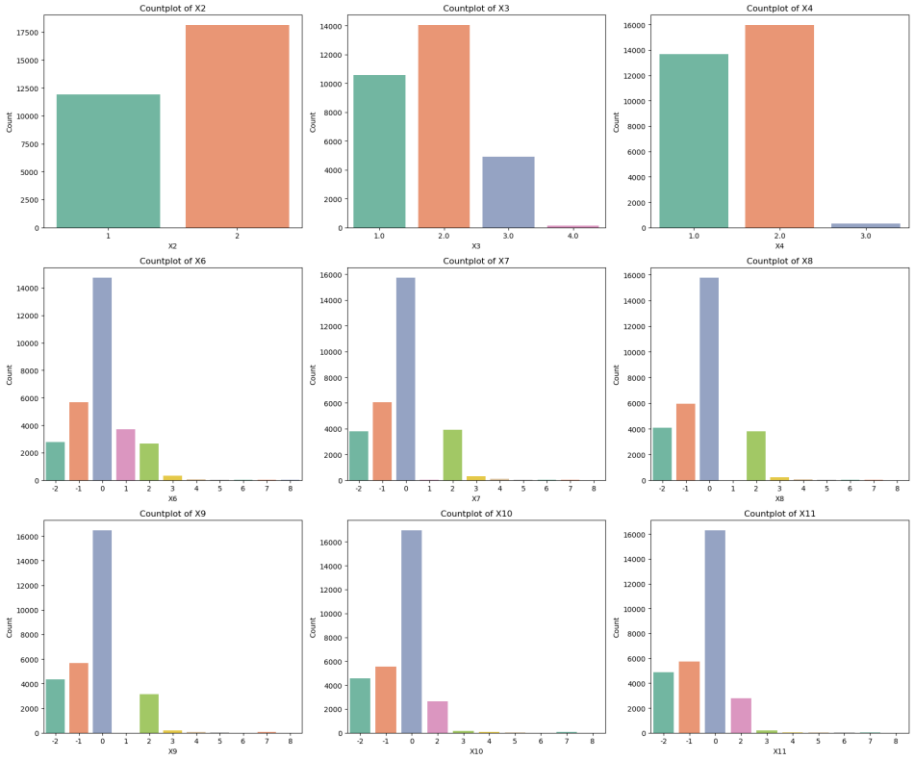
4.1.1 Distribusi Variabel Numerik

Gambar 2. Plot Distribusi Variabel Numerik



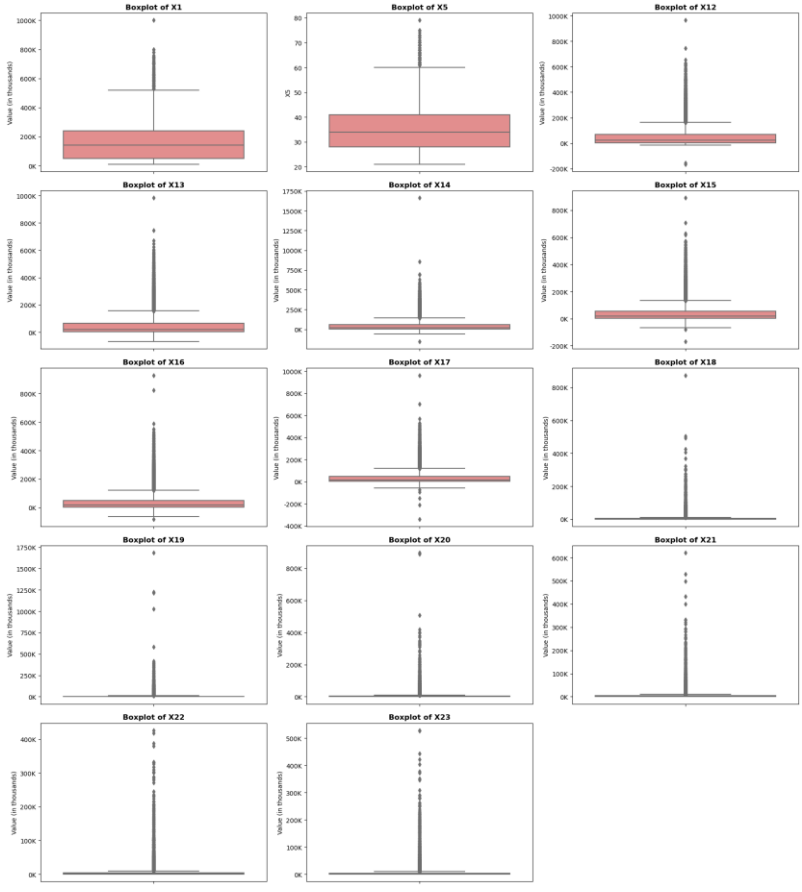
4.1.2 Distribusi Variabel Kategorik

Gambar 3. Plot Distribusi Variabel Kategorik



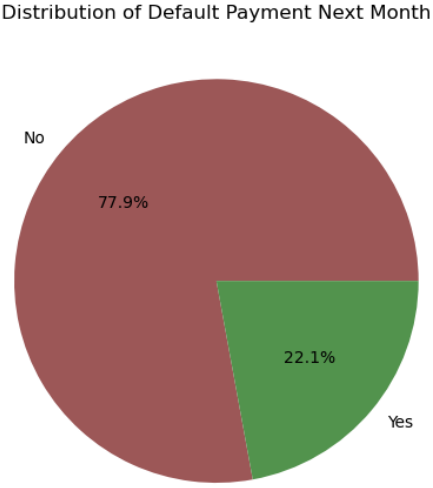
4.1.3 Boxplot

Gambar 4. Boxplot



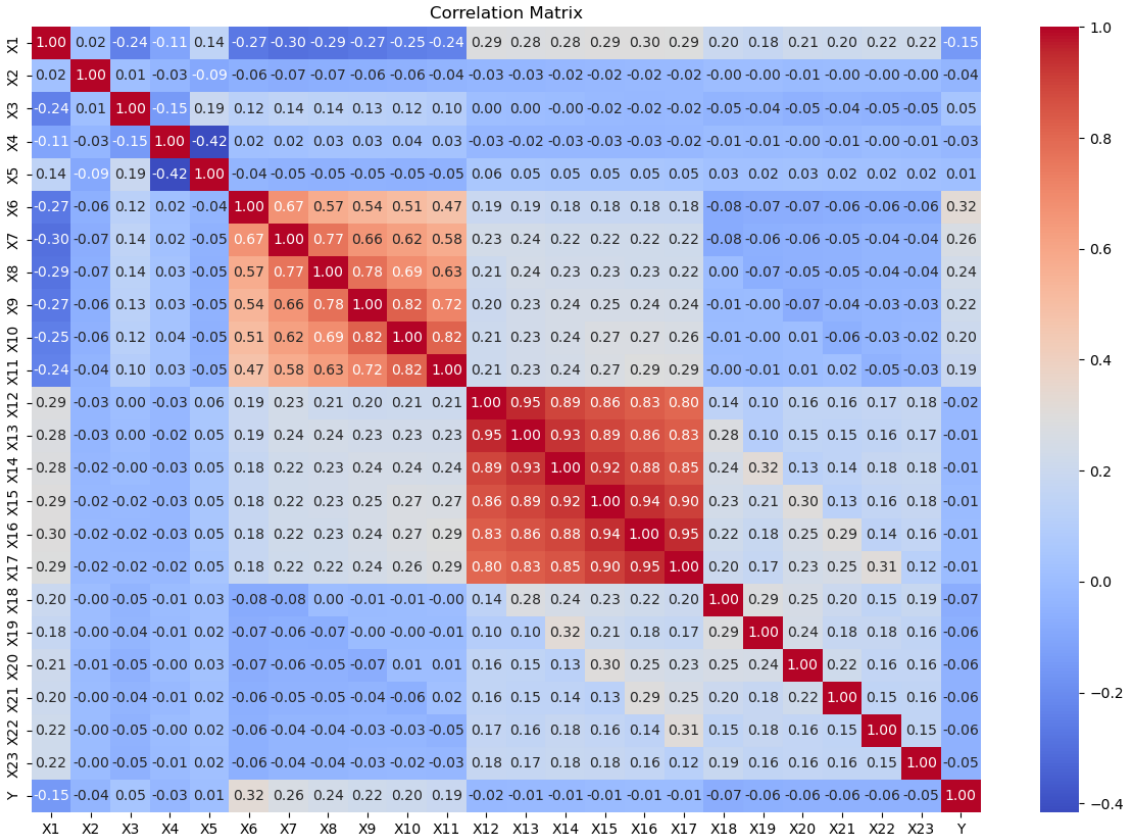
4.1.4 Distribusi Variabel Respon

Gambar 5. Plot Distribusi Variabel Resp



4.1.5 Korelasi

Gambar 6. Plot Korelasi



4.2 Interpretasi Hasil

Tabel 1. Metric Score

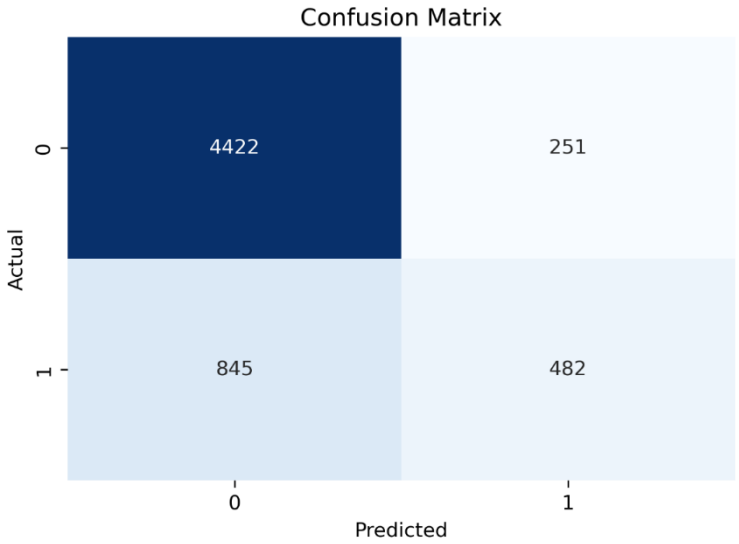
Accuracy (%)	Precision (%)	Recall (%)	F1-Score
81,73	65,76	36,32	46,80

Berdasarkan hasil evaluasi pada data uji, model mencapai akurasi keseluruhan sebesar 81,73%. Nilai ini menunjukkan bahwa model mampu memprediksi dengan benar sekitar 8 dari 10 nasabah secara keseluruhan. Namun, untuk memahami performa model secara lebih mendalam, terutama dalam konteks masalah bisnis prediksi gagal bayar, analisis metrik *precision*, *recall*, dan *confusion matrix* menjadi sangat penting.

Precision model untuk kelas 'Default' adalah 65,76%. Ini berarti dari semua nasabah yang diprediksi akan gagal bayar, sekitar 66% di antaranya memang benar-benar gagal bayar. Di sisi lain, Recall menunjukkan nilai yang lebih rendah, yaitu 36,32%. Angka ini merupakan metrik yang krusial, karena ia mengindikasikan bahwa model hanya berhasil mengidentifikasi sekitar 36% dari seluruh nasabah yang seharusnya gagal bayar. Sebagian besar (sekitar 64%) kasus gagal bayar yang sebenarnya masih terlewatkan oleh model dan keliru diklasifikasikan sebagai tidak gagal bayar (False Negative).

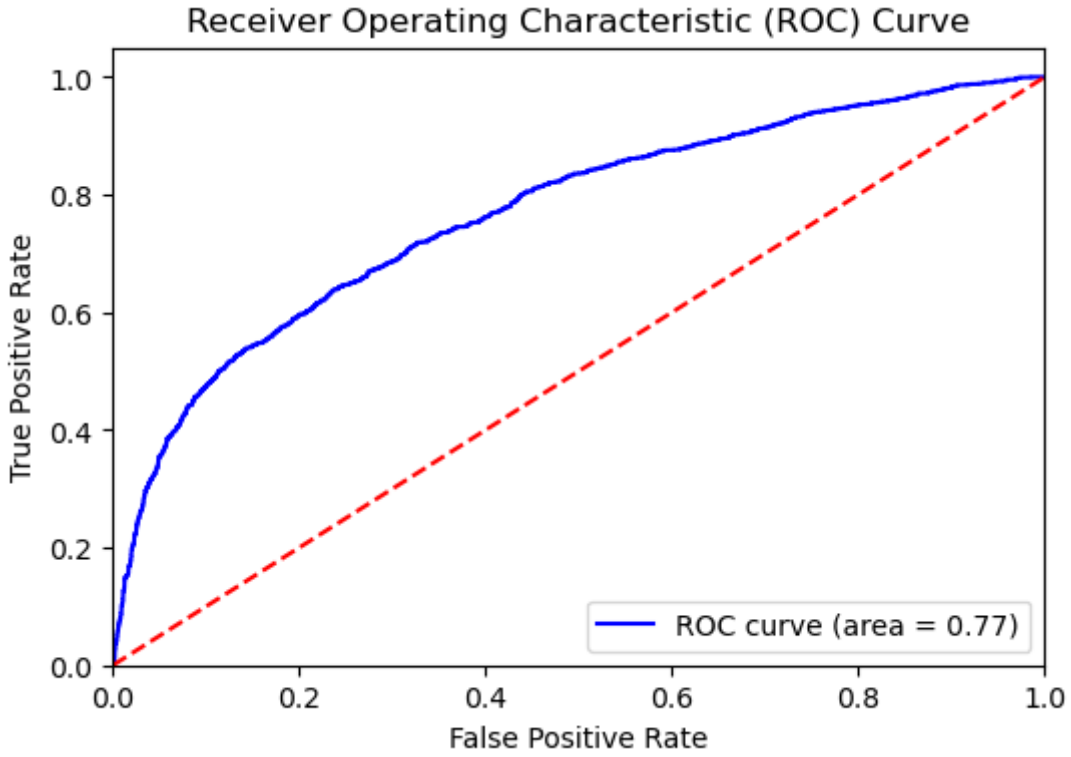
F1-Score sebesar 46,80% merupakan rata-rata harmonik dari precision dan recall. Nilai F1-Score yang moderat ini menyoroti adanya trade-off antara kemampuan model untuk tidak salah melabeli nasabah sebagai 'Default' (precision) dan kemampuannya untuk menangkap semua kasus 'Default' yang ada (recall). Rendahnya nilai recall menjadi tantangan utama, yang menunjukkan bahwa meskipun model cukup baik dalam memastikan prediksinya benar ketika menebak 'Default', ia masih terlalu konservatif dan melewatkan banyak kasus yang seharusnya diidentifikasi.

Gambar 7. Confusion Matrix



Gambar 8. Train and Validation Loss

4.3 Validasi Model



BAB 5
KESIMPULAN DAN SARAN

5.1 Ringkasan Temuan

Eksperimen ini berhasil membangun model]JST untuk memprediksi gagal bayar kartu kredit dengan performa yang menjanjikan, namun juga menyoroti beberapa tantangan penting. Serangkaian langkah pra-pemrosesan data, mulai dari penanganan nilai anomali, penskalaan dengan RobustScaler untuk mengatasi outlier, hingga seleksi fitur, terbukti krusial dalam menyiapkan data yang berkualitas untuk pemodelan. Teknik pembobotan kelas (class weighting) juga berhasil memaksa model untuk tidak mengabaikan kelas minoritas yang tidak seimbang.

Hasil akhir menunjukkan bahwa model memiliki kemampuan diskriminatif yang baik, yang dikonfirmasi oleh akurasi keseluruhan sebesar 81,73% dan nilai AUC 0.77. Ini mengindikasikan bahwa model secara fundamental mampu membedakan antara nasabah berisiko tinggi dan rendah, jauh lebih baik daripada tebakan acak.

Namun, refleksi utama dari eksperimen ini terletak pada adanya *trade-off* yang signifikan antara precision dan recall. Meskipun model memiliki *precision* yang cukup baik (65,76%), yang berarti prediksinya terhadap kasus 'Default' relatif dapat diandalkan, tantangan terbesarnya adalah nilai recall yang rendah (36,32%). Angka ini menunjukkan bahwa model masih melewatkan sebagian besar (sekitar 64%) dari total nasabah yang sebenarnya gagal bayar. Dari perspektif bisnis, ini adalah risiko yang signifikan karena banyak nasabah berisiko tidak teridentifikasi.

Sebagai kesimpulan, eksperimen ini sukses menghasilkan model dasar yang valid, namun untuk implementasi di dunia nyata, perlu dilakukan iterasi lebih lanjut. Fokus utama untuk pengembangan selanjutnya adalah meningkatkan recall, bahkan jika harus sedikit mengorbankan precision. Hal ini dapat dieksplorasi melalui penyesuaian threshold klasifikasi, mencoba teknik oversampling yang lebih agresif seperti ADASYN, atau menyempurnakan arsitektur model agar lebih sensitif terhadap kelas minoritas.Refleksi hasil eksperimen

5.2 Saran dan Pengembangan

5.2.1 Pengembangan Model

1) Optimasi Threshold Klasifikasi

   Model saat ini menggunakan threshold standar 0.5 untuk menentukan klasifikasi. Mengingat rendahnya nilai recall, disarankan untuk melakukan tuning pada threshold. Dengan menurunkan threshold (misalnya ke 0.4 atau 0.35), model akan menjadi lebih sensitif dalam memprediksi kelas 'Default', yang berpotensi meningkatkan recall secara signifikan, meskipun mungkin akan sedikit menurunkan precision. Analisis kurva Precision-Recall dapat membantu menemukan titik threshold yang optimal sesuai dengan kebutuhan bisnis.

2) Eksplorasi Teknik Oversampling Lanjutan

   Selain pembobotan kelas, disarankan untuk mengimplementasikan dan membandingkan teknik oversampling canggih seperti ADASYN atau Borderline-SMOTE. Teknik-teknik ini menciptakan sampel sintetis secara lebih cerdas dengan berfokus pada area batas keputusan yang sulit, yang berpotensi membantu model mengenali kasus gagal bayar dengan lebih baik.

3) Rekayasa Fitur (*Feature Engineering*)

   Untuk memberikan informasi yang lebih kaya kepada model, dapat dilakukan pembuatan fitur-fitur baru dari data yang ada. Contohnya, membuat fitur rasio seperti rasio utang terhadap limit (BILL_AMTX / LIMIT_BAL) atau rasio pembayaran terhadap tagihan (PAY_AMTX / BILL_AMTX) dapat memberikan sinyal prediktif yang lebih kuat mengenai perilaku keuangan nasabah.

5.2.2 Studi Lanjutan

1) Eksperimen dengan Model Alternatif

   Disarankan untuk membandingkan performa model JST dengan algoritma ensemble modern yang dikenal andal pada data tabular, seperti XGBoost, LightGBM, atau CatBoost. Model-model ini seringkali memiliki performa yang sangat kompetitif dan menyediakan mekanisme bawaan untuk menangani data yang tidak seimbang.

2) Analisis Berbasis Biaya (*Cost-Sensitive Analysis*)

Penelitian di masa depan dapat mengadopsi pendekatan yang lebih berorientasi bisnis dengan mengintegrasikan matriks biaya. Dalam skenario nyata, biaya dari False Negative (tidak mendeteksi nasabah gagal bayar) jauh lebih tinggi daripada biaya False Positive (salah menandai nasabah baik). Membangun model yang bertujuan untuk meminimalkan total biaya finansial, bukan hanya metrik klasifikasi standar, akan menghasilkan solusi yang lebih aplikatif.

3) Analisis Temporal Perilaku Pembayaran

Fitur riwayat pembayaran dan tagihan memiliki dimensi waktu. Studi lanjutan dapat melakukan analisis sekuensial atau time-series pada enam bulan terakhir data pembayaran untuk menangkap tren dan pola perilaku nasabah dari waktu ke waktu, yang mungkin menjadi prediktor yang lebih kuat daripada hanya melihat data sebagai satu titik waktu statis.

DAFTAR PUSTAKA

N. V. Chawla, K. W. Bowyer, L. O. Hall, dan W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, hlm. 321–357, Jun 2002, doi: 10.1613/jair.953.

Li, Mingfeng. (2024). Comprehensive Review of Backpropagation Neural Networks. Academic Journal of Science and Technology. 9. 150-154. 10.54097/51y16r47.

Y. Heryadi dan T. Wahyono, Machine Learning: Konsep dan Implementasi. Yogyakarta: Gava Media Yogyakarta, 2020.

Yeh, I. (2009). Default of Credit Card Clients [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C55S3H.

Yeh, I., & Lien, C. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Syst. Appl., 36*, 2473-2480.

LAMPIRAN

Notebook Lengkap: https://github.com/qlbusalim/Binary-Classification-Using-NN.