Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

Preventing Data Leakage in Classification via Integrated Machine Learning Pipelines: Preprocessing, Feature Transformation, and Hyperparameter Tuning

Arief Ichwani*[1], Rahman Indra Kesuma[2], Andika Setiawan[3], Imam Eko Wicaksono[4], Raidah Hanifah[5]

[1,2,3,4,5]Teknik Informatika, Institut Teknologi Sumatera, Indonesia

Email: [1]arief.ichwani@if.itera.ac.id

Received : Nov 28, 2025; Revised : Jan 26, 2026; Accepted : Jan 30, 2026; Published : Feb 15, 2026

Abstract

Data leakage in machine learning classification often leads to overfitting, inflated performance estimates, and poor reproducibility, which can undermine the reliability of deployed models and incur industrial losses. This paper addresses the leakage problem by proposing an integrated machine learning pipeline that strictly isolates training and evaluation processes across preprocessing, feature transformation, and model optimization stages. Experiments are conducted on the Titanic passenger survival dataset, where exploratory data analysis identifies data quality issues, followed by stratified train-test splitting to preserve class distribution. All preprocessing steps, including missing value imputation, categorical encoding, and feature scaling, are applied exclusively to the training data using a ColumnTransformer embedded within a unified Pipeline. A K-Nearest Neighbors (KNN) classifier is employed, with hyperparameters optimized via GridSearchCV and 3-fold cross-validation. Experimental results show that a baseline model without leakage control achieves only 72.62% test accuracy and exhibits a substantial overfitting gap. In contrast, the proposed pipeline-based approach improves generalization, achieving 78.21% test accuracy with an optimal configuration of k = 29 and Manhattan distance while significantly reducing overfitting. The main contribution of this work is the formulation of a reproducible, leakage-aware pipeline guideline that ensures unbiased evaluation and reliable generalization in classification tasks, providing practical methodological insights for both academic research and real-world machine learning applications.

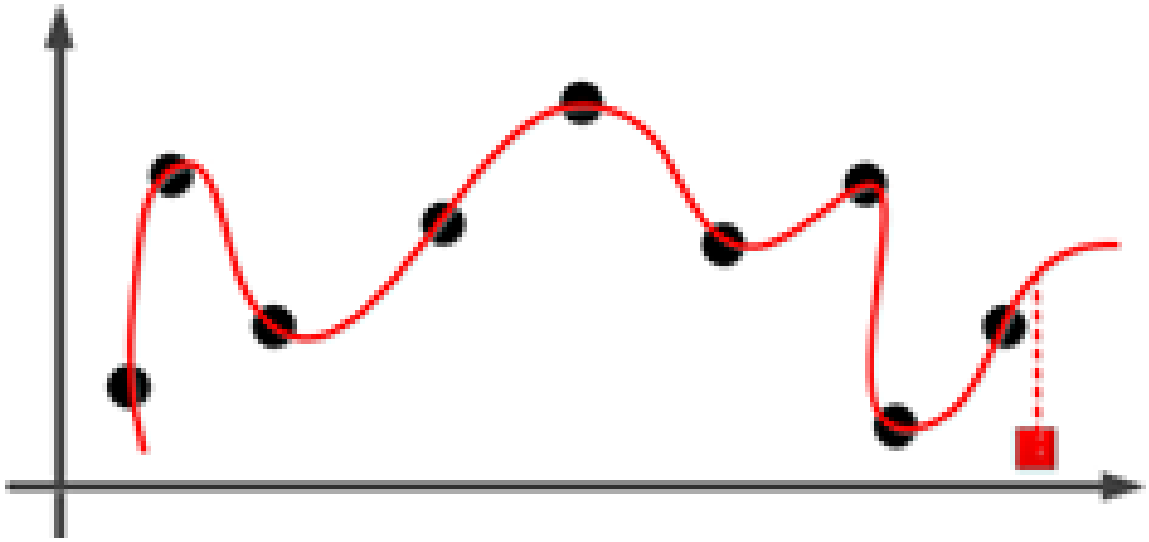
Keywords : *Classification pipeline, Data leakage prevention, Feature transformation, K-nearest neighbors, Machine learning preprocessing*

1. INTRODUCTION

Dalam beberapa dekade terakhir, *machine learning* (ML) telah berkembang menjadi teknologi kunci yang mendorong inovasi di berbagai sektor, termasuk kesehatan, transportasi, keuangan, manufaktur, dan teknologi informasi [1], [2]. Dengan memanfaatkan data dalam skala besar, model ML dirancang untuk melakukan prediksi dan klasifikasi terhadap fenomena yang kompleks. Namun, keberhasilan penerapan model ML tidak hanya ditentukan oleh pemilihan algoritma, melainkan sangat bergantung pada kualitas data serta ketepatan proses pengolahan dan evaluasinya [3]. Secara umum, algoritma *machine learning* dapat diklasifikasikan ke dalam tiga kategori utama, yaitu *supervised learning*, *unsupervised learning*, dan *reinforcement learning*. Secara umum, algoritma machine learning diklasifikasikan ke dalam tiga kategori utama: supervised learning, unsupervised learning, dan reinforcement learning. Di antara ketiganya, *supervised learning* merupakan pendekatan yang paling luas digunakan untuk tugas klasifikasi, dimana model mempelajari pemetaan antara fitur dan label berdasarkan pasangan data berlabel (X, y) [1]. Meskipun efektif, pendekatan ini rentan terhadap permasalahan *overfitting*, yaitu kondisi ketika model menunjukkan kinerja yang sangat baik pada data pelatihan tetapi gagal melakukan generalisasi pada data uji atau data dunia nyata [4], [5].

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

Salah satu sumber utama overfitting dan estimasi performa berlebihan (*over-optimistic estimates*) adalah data leakage, yaitu situasi ketika informasi yang seharusnya hanya tersedia pada tahap evaluasi secara tidak sengaja digunakan selama pelatihan [6], [7]. Kebocoran data dapat terjadi pada berbagai tahap pipeline, mulai dari pembagian dataset yang tidak tepat, penerapan *preprocessing* sebelum pemisahan data latih dan data uji, penggunaan fitur yang mengandung informasi masa depan (target *leakage*), atau konfigurasi cross-validation yang salah [8], [9]. Perbedaan distribusi antara data pelatihan dan data pengujian akibat proses yang tidak terkontrol [10] ditunjukkan pada Gambar 1.



Gambar 1. Contoh distribusi data pelatihan dan pengujian (warna hitam data pelatihan dan merah data pengujian)

Dampak *data leakage* tidak hanya terbatas pada penurunan kinerja model saat diterapkan di dunia nyata, tetapi juga berimplikasi serius terhadap kredibilitas ilmiah. Dalam konteks penelitian, *data leakage* dapat menghasilkan estimasi performa yang terlalu optimistis dan sulit direproduksi oleh peneliti lain. Studi lintas disiplin melaporkan bahwa ratusan publikasi ilmiah terdampak oleh fenomena ini, menunjukkan bahwa permasalahan tersebut masih belum tertangani secara sistematis [13]. Dalam konteks industri, model yang dilatih dengan kebocoran dapat memicu keputusan keliru dan kerugian finansial (*industrial losses*) di sektor berisiko tinggi seperti keuangan dan layanan publik [11].

Sejumlah penelitian dan laporan teknis belakangan ini fokus pada klasifikasi dan mitigasi sumber-sumber *leakage* serta praktik pipeline yang benar. Review dan studi kasus terbaru menguraikan skenario *leakage*, contoh kesalahan umum, serta panduan evaluasi yang lebih aman [13], [14]. Namun demikian, implementasi yang dilaporkan dalam literatur masih bervariasi dan sedikit studi yang menawarkan panduan *end-to-end* yang mudah direplikasi oleh praktisi. Selain itu, beberapa studi belum menerapkan pipeline secara menyeluruh dari awal hingga akhir proses pemodelan, sehingga potensi *data leakage* masih tetap terbuka. Oleh karena itu, penelitian ini bertujuan untuk mengkaji dan mendemonstrasikan pencegahan *data leakage* pada tugas klasifikasi melalui penerapan *machine learning* pipeline terintegrasi, dengan menggunakan dataset Titanic sebagai studi kasus. Penelitian ini difokuskan pada analisis diagnostik terhadap indikasi *data leakage*, penerapan *preprocessing* dan transformasi fitur yang sepenuhnya terisolasi pada data pelatihan, serta evaluasi kuantitatif dampak pipeline terhadap kemampuan generalisasi model klasifikasi. Untuk menempatkan kontribusi ini secara jelas, Tabel 1 membandingkan 5 studi terkini (2023 s.d 2025) terkait *data leakage*.

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

Tabel 1. Perbandingan studi terkini tentang *data leakage*

Penulis (Tahun)	Fokus Penelitian	Dataset / Konteks	Temuan Utama	Keterbatasan
Kapoor & Narayanan (2023) [11]	Survei dan klasifikasi jenis *data leakage* dalam penelitian *machine learning*	Studi lintas disiplin	Mengidentifikasi ratusan publikasi terdampak data leakage serta merumuskan taksonomi sistematis jenis kebocoran data	Bersifat konseptual dan survei; tidak menyediakan panduan implementasi teknis secara langsung
Sasse et al. (2023) [10]	Analisis sumber *data leakage* dalam pipeline *machine learning*	Contoh pipeline dan skenario umum	Menjelaskan berbagai skenario *data leakage* yang muncul akibat kesalahan pemrosesan data dan evaluasi model	Pendekatan deskriptif; tidak disertai evaluasi kuantitatif pada dataset *benchmark*
Apicella et al. (2024) [15]	Risiko *data leakage* pada *workflow machine learning modern* dan *transfer learning*	Studi kasus dan simulasi pipeline	Menunjukkan bahwa penggunaan pipeline otomatis dan *workflow* instan dapat meningkatkan risiko *data leakage*	Fokus pada analisis praktik pengguna; belum menilai dampak kuantitatif secara luas
Rosenblatt et al. (2024) [16]	Dampak *data leakage* terhadap performa prediksi	Tugas prediksi pada domain neuroimaging	Membuktikan bahwa data leakage dapat meningkatkan performa model secara signifikan, terutama pada dataset berukuran kecil	Konteks terbatas pada domain tertentu; generalisasi ke data tabular perlu kajian lanjutan
AlOmar et al. (2025) [17]	Deteksi otomatis *data leakage* dalam pipeline *machine learning*	Analisis kode dan pipeline (*tool open-source*)	Mengusulkan alat untuk mendeteksi indikasi data leakage dan memberikan rekomendasi perbaikan pipeline	Masih pada tahap awal; evaluasi skala besar dan penerapan industri belum dilakukan

2. METHOD

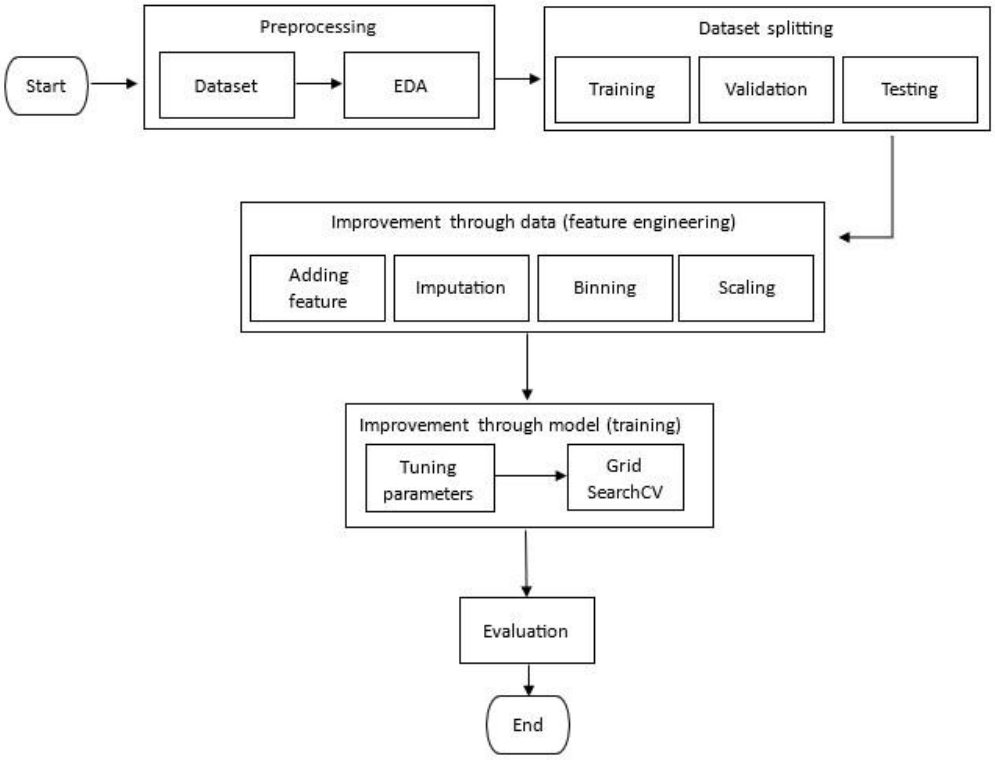
Salah satu tantangan terbesar dalam pengembangan model pembelajaran mesin adalah risiko terjadinya *overfitting*, situasi dimana model mempelajari pola yang terlalu spesifik dari data pelatihan sehingga kinerjanya menurun saat diaplikasikan pada data baru. Salah satu penyebab utama *overfitting* adalah kebocoran data, yaitu kondisi dimana informasi dari data pengujian "bocor" ke data pelatihan secara langsung maupun tidak langsung. Kebocoran ini dapat memberikan gambaran performa model yang terlalu optimis selama pengembangan atau pelatihan, namun gagal merepresentasikan performa di pengujian atau dunia nyata. Adapun langkah-langkah dalam penelitian ini diilusterasikan pada Gambar 2.

2.1. Dataset dan Etika Data

Penelitian ini menggunakan Dataset Titanic yang tersedia secara publik melalui platform Kaggle (Titanic: Machine Learning from Disaster). Dataset ini terdiri dari 891 sampel penumpang dengan 12 atribut, dimana variabel target *Survived* menunjukkan status keselamatan penumpang (0 = tidak selamat, 1 = selamat). Fitur yang digunakan mencakup karakteristik demografis dan sosial penumpang, seperti

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

Age, *Sex*, *Pclass*, *SibSp*, *Parch*, *Fare*, dan *Embarked*. Dataset ini bersifat *open-access* dan tidak mengandung informasi sensitif atau identitas pribadi, sehingga tidak memerlukan proses anonimisasi tambahan. Penggunaan dataset ini dinilai etis dan sesuai untuk penelitian akademik, khususnya sebagai studi kasus metodologis dalam evaluasi pipeline *machine learning*.



Gambar 2. Diagram penelitian

2.2. *Exploratory data analysis* (EDA)

EDA merupakan tahapan awal dalam *workflow* pembelajaran mesin yang bertujuan untuk memperoleh pemahaman menyeluruh mengenai karakteristik, struktur, dan kualitas data sebelum memasuki proses pemodelan. Tahap ini diawali dengan pemeriksaan struktur dataset, meliputi jumlah baris dan kolom, tipe data pada setiap fitur, serta gambaran umum nilai yang terkandung di dalamnya. Selanjutnya dilakukan identifikasi nilai hilang dan data duplikat untuk mendeteksi potensi permasalahan kualitas data yang dapat memengaruhi ketepatan proses pemodelan. EDA juga mencakup analisis statistik deskriptif, seperti mean, median, varians, dan distribusi nilai pada fitur numerik maupun kategorikal, yang bertujuan untuk mengidentifikasi pola, penyimpangan, serta keberadaan *outlier*.

Temuan dari keseluruhan proses EDA menjadi dasar dalam menetapkan strategi *preprocessing* yang tepat, seperti imputasi nilai hilang, *encoding* fitur kategorikal, normalisasi atau *scaling* pada fitur numerik, penanganan *outlier*, serta penerapan teknik *balancing* jika diperlukan. Dengan demikian, EDA berfungsi tidak hanya sebagai sarana untuk memahami kondisi dataset, tetapi juga sebagai landasan pengambilan keputusan dalam mempersiapkan data agar siap digunakan oleh model pembelajaran mesin secara optimal.

Secara umum, EDA dilakukan sebelum proses *preprocessing* utama. Namun, dalam praktiknya beberapa *preprocessing* ringan dapat dilakukan sebelum EDA tertentu, khususnya yang bersifat teknis dan tidak mengubah informasi substantif data. Contohnya adalah perbaikan tipe data (*type casting*) untuk memastikan fitur numerik terbaca sebagai bilangan (25 tahun menjadi 25), atau pembersihan teks dasar (*text cleaning*) pada data tekstual untuk memungkinkan analisis frekuensi kata. Langkah-langkah ini diperlukan agar visualisasi dan analisis EDA dapat berjalan dengan akurat tanpa terhambat oleh ketidaksesuaian format data.

2.3. *Data Preprocessing*

Tahap *preprocessing* merupakan proses lanjutan setelah *Exploratory Data Analysis* (EDA) yang bertujuan untuk menyiapkan data dalam kondisi optimal sebelum digunakan pada model klasifikasi. Tahapan ini diawali dengan penanganan nilai hilang melalui teknik imputasi yang sesuai dengan karakteristik fitur, seperti *mean* atau *median imputation* untuk fitur numerik serta *most frequent imputation* atau *constant imputation* untuk fitur kategorikal. Proses ini dilakukan untuk memastikan tidak adanya kekosongan nilai yang dapat mengganggu proses pelatihan model. Selanjutnya, *preprocessing* mencakup pembersihan data dari duplikasi, inkonsistensi format, serta rekonsiliasi data yang tidak valid sehingga kualitas dataset terjaga dan siap diproses lebih lanjut.

Tahap berikutnya melibatkan transformasi dan *encoding* fitur, terutama pada fitur kategorikal yang memerlukan konversi ke bentuk numerik agar dapat diterima oleh algoritma pembelajaran mesin. Metode yang umum digunakan meliputi *One-Hot Encoding*, *Ordinal Encoding*, atau *Target Encoding*, yang dipilih berdasarkan sifat dan jumlah kategori pada fitur tersebut. Misalkan terdapat sebuah fitur kategorikal Xdengan himpunan nilai unik:

$$X = \{c_1, c_2, c_3, \dots, c_k\} \quad (1)$$

Untuk suatu nilai kategori tertentu x_i, hasil *One-Hot Encoding* direpresentasikan sebagai vektor biner berdimensi k:

$$OHE(x_i) = [o_1, o_2, \dots, o_k] \quad (2)$$

dengan aturan:

$$o_j = \begin{cases} 1, & \text{jika } x_i = c_j \\ 0, & \text{jika } x_i \neq c_j \end{cases} \quad (3)$$

Sedangkan, untuk fitur numerik, proses standar seperti normalisasi atau standardisasi (misalnya *Min-Max Scaling* atau *StandardScaler*) diterapkan guna menyamakan skala antar fitur dan mencegah bias pada algoritma yang sensitif terhadap perbedaan skala. Metode Min-Max Scaling mengubah skala data ke rentang tertentu, seperti [0, 1] sesuai pada Persamaan 1.

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad (4)$$

dimana:

a. X = nilai asli
b. $X_{\min}$ = nilai minimum pada fitur
c. $X_{\max}$ = nilai maksimum pada fitur

Pada tugas klasifikasi, *preprocessing* juga mencakup penanganan ketidakseimbangan kelas (*class imbalance*) yang dapat memengaruhi kualitas prediksi model. Teknik yang digunakan antara lain *over-sampling* (seperti SMOTE), *under-sampling*, atau penyesuaian bobot kelas (*class weight adjustment*) sesuai dengan tingkat ketidakseimbangan. *Feature engineering* dapat diterapkan untuk meningkatkan representasi informasi, seperti pembuatan fitur baru, transformasi non-linear, atau seleksi fitur berbasis korelasi maupun model. Berbagai langkah tersebut dikombinasikan dalam sebuah *machine learning* pipeline agar seluruh transformasi dilakukan secara konsisten dan hanya menggunakan data dari subset pelatihan, sehingga risiko *data leakage* dapat diminimalkan.

2.4. Arsitektur Pipeline dan ColumnTransformer

Arsitektur pipeline merupakan pendekatan sistematis untuk menggabungkan berbagai tahap *preprocessing* dan model menjadi sebuah alur eksekusi yang konsisten, terstandarisasi, dan bebas *data leakage*. Pipeline memastikan setiap langkah diterapkan dengan urutan yang benar pada data latih maupun data uji tanpa inkonsistensi. Dengan demikian, pipeline membantu menjaga integritas eksperimen, memudahkan replikasi, serta meningkatkan kehandalan model.

1. *ColumnTransformer*

ColumnTransformer digunakan untuk menerapkan teknik *preprocessing* yang berbeda berdasarkan tipe fitur, yaitu numerik dan kategori. Pada fitur numerik, proses yang umum dilakukan antara lain:

- Imputasi nilai hilang.
- Scaling seperti *Min-Max* atau *StandardScaler*.

Sementara pada fitur kategori, proses yang sering digunakan meliputi:

- Imputasi kategori kosong.
- *Encoding* fitur kategori seperti *OneHotEncoder* atau *OrdinalEncoder*.

ColumnTransformer menggabungkan kedua blok *preprocessing* tersebut dalam struktur yang paralel, sehingga setiap kelompok kolom melalui proses yang sesuai tanpa saling mengganggu. Hasil akhirnya adalah matriks fitur terpadu yang siap dipakai model.

2. Pipeline bertingkat (*preprocessing* dan model)

Pipeline bertingkat adalah pipeline yang tidak hanya berisi langkah *preprocessing*, tetapi juga mencakup algoritma model di tahap akhir. Struktur ini memastikan seluruh proses yaitu mulai dari pembersihan data, transformasi fitur, sampai pemodelan berjalan sebagai satu unit yang utuh. Keuntungan utama pendekatan ini meliputi:

- Menghindari *data leakag*e, karena semua transformasi fit hanya pada *data train*.
- Mempermudah parameter *tuning*, karena *GridSearchCV* dapat menyetel parameter *preprocessing* dan model secara bersamaan.
- Konsistensi eksekusi, baik saat *training*, *cross-validation*, maupun *testing*.

Pipeline bertingkat juga memungkinkan integrasi modular, seperti:

- Pipeline internal untuk fitur numerik.
- Pipeline internal untuk fitur kategori.
- *ColumnTransformer* sebagai pembungkus kedua pipeline tersebut.
- Pipeline final yang menggabungkan *ColumnTransformer + model*.

Pipeline ibarat sebuah rangkaian kerja otomatis yang memastikan setiap data diproses dalam urutan yang sama, mulai dari imputasi, transformasi fitur, hingga pemodelan. *ColumnTransformer* memisahkan jalur pemrosesan berdasarkan tipe fitur sehingga fitur numerik dan kategorikal mendapatkan perlakuan yang sesuai. Ketika seluruh blok ini digabungkan dalam pipeline bertingkat, proses *machine learning* menjadi terstruktur, *reproducible*, dan terlindungi dari risiko *data leakage* karena seluruh transformasi hanya dilatih pada data latih dan diterapkan konsisten pada data uji.

2.5. Pemodelan dan *Hyperparameter Tuning*

Dalam arsitektur pipeline, proses *modeling* merupakan tahap akhir setelah data melewati seluruh proses *preprocessing*. Model bekerja seperti mesin utama di ujung jalur produksi yang menerima bahan baku (fitur hasil transformasi) yang sudah siap diproses. Karena *preprocessing* dan model dibungkus dalam satu pipeline, maka model selalu menerima *input* yang konsisten dan bebas dari kontaminasi (*data leakage*).

1. *Hyperparameter Tuning* dengan *GridSearchCV*

Pada penelitian ini, proses *hyperparameter tuning* dilakukan menggunakan GridSearchCV, yang berfungsi untuk mengevaluasi seluruh kombinasi parameter secara sistematis dan terkontrol.

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

GridSearchCV diintegrasikan langsung di atas *machine learning* pipeline, sehingga proses penyesuaian parameter tidak hanya mencakup model klasifikasi, tetapi juga seluruh tahapan *preprocessing* yang terenkapsulasi di dalam pipeline. Ketika GridSearchCV dijalankan di atas pipeline, setiap kombinasi parameter dievaluasi dengan mengeksekusi ulang seluruh tahapan pemodelan, mulai dari imputasi nilai hilang, encoding variabel kategorikal, penskalaan fitur, hingga pelatihan model. Pendekatan ini memastikan bahwa:

a. setiap konfigurasi parameter diuji menggunakan urutan proses yang identik,
b. tidak terjadi kebocoran informasi antar *fold* selama evaluasi, dan
c. seluruh proses *fit* pada preprocessing hanya dilakukan menggunakan data pelatihan pada masing-masing *fold*.

Hyperparameter tuning diterapkan pada algoritma K-Nearest Neighbors (KNN) dengan ruang pencarian parameter sebagai berikut:

a. n_neighbors : bilangan ganjil dari 1 hingga 49
b. weights : {uniform, distance}
c. p : {1 (Manhattan), 2 (Euclidean)}
d. cv : 3-fold cross-validation
e. scoring : 'accuracy'
f. random_state : 42 (untuk menjaga reprodusibilitas eksperimen)

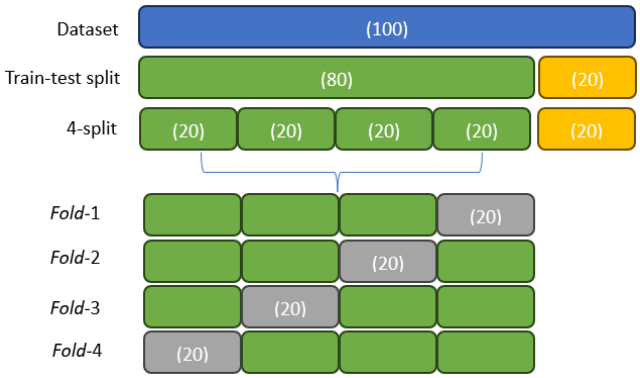
Seluruh konfigurasi parameter dievaluasi menggunakan GridSearchCV yang dijalankan di atas pipeline terintegrasi, sehingga setiap hasil evaluasi bebas dari risiko data leakage dan dapat direproduksi secara konsisten.

2. *Cross-Validation* dan Pengendalian *Data Leakage*

Cross-validation (CV) digunakan sebagai mekanisme evaluasi utama untuk memastikan bahwa tidak ada informasi dari data pengujian yang mempengaruhi proses pelatihan model [18]. Kesalahan yang sering terjadi dalam praktik machine learning adalah melakukan preprocessing di luar proses CV, misalnya dengan menerapkan scaling, imputasi, atau encoding sebelum data dibagi ke dalam *train* dan *test folds*. Praktik tersebut dapat menyebabkan kebocoran informasi, seperti nilai rata-rata dan standar deviasi yang dihitung dari seluruh dataset. Untuk mencegah kondisi tersebut, penelitian ini menerapkan pipeline terintegrasi yang memastikan bahwa pada setiap *fold* cross-validation:

a. pipeline melakukan proses *fit* hanya pada data pelatihan (*train fold*),
b. transformasi preprocessing diterapkan pada data pengujian (*test fold*) menggunakan parameter hasil *fit* sebelumnya, dan
c. proses ini diulang secara independen untuk setiap *fold*.

Selain itu, pembagian data awal dilakukan menggunakan stratified train-test split dengan rasio 80% data pelatihan dan 20% data pengujian, guna menjaga proporsi kelas dan mengatasi potensi ketidakseimbangan kelas pada dataset klasifikasi. Ilusterasi penerapan CV dapat dilihat pada Gambar 3.

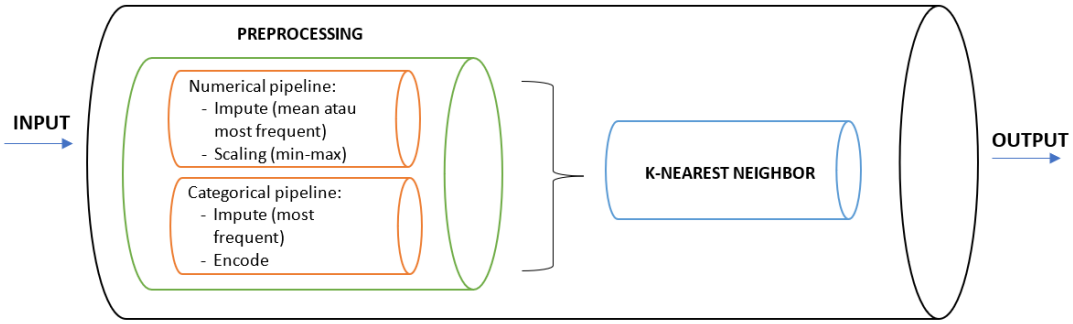


Gambar 3. Contoh penerapan *cross-validation*

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

3. Transparansi Pipeline

Untuk meningkatkan transparansi metodologi dan kemudahan replikasi, struktur pipeline yang digunakan dalam penelitian ini diilusterasikan pada Gambar 4 berikut.



Gambar 4. Struktur pipeline yang digunakan dalam penelitian

Sebagai ilustrasi transparansi implementasi, struktur pipeline direpresentasikan dalam bentuk algoritma berikut:

Tabel 2. Algoritma pipeline terintegrasi untuk pencegahan data leakage

Masukan: Dataset $\boldsymbol{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$
Keluaran: Model klasifikasi KNN bebas *data leakage*
1. Lakukan pembagian data secara *stratified* dengan rasio 80% data latih dan 20% data uji, menggunakan random_state = 42.
2. Definisikan *pipeline* prapemrosesan numerik (imputasi median dan normalisasi Min-Max).
3. Definisikan *pipeline* prapemrosesan kategorikal (imputasi modus dan *one-hot encoding*).
4. Gabungkan kedua *pipeline* menggunakan *column-wise transformer*.
5. Bangun *pipeline* terintegrasi yang terdiri atas modul prapemrosesan dan pengklasifikasi KNN.
6. Tentukan ruang pencarian *hyperparameter* KNN: $\boldsymbol{k} \in \{\boldsymbol{1}, \boldsymbol{3}, \ldots, \boldsymbol{49}\}$, bobot jarak {uniform, distance}, dan metrik jarak {Manhattan, Euclidean}.
7. Terapkan GridSearchCV dengan 3-fold *cross-validation* dan metrik evaluasi akurasi.
8. Latih *pipeline* menggunakan data latih dan pilih konfigurasi *hyperparameter* terbaik.
9. Kembalikan *pipeline* teroptimasi sebagai model akhir.

Pendekatan ini memastikan bahwa seluruh estimasi statistik hanya dipelajari dari data latih, baik dalam *training*, *cross-validation*, maupun *testing*.

4. *Modeling* dengan *K-Nearest Neighbors* (KNN)

Model *K-Nearest Neighbors* (KNN) merupakan algoritma berbasis *instance-based learning*, dimana prediksi dilakukan dengan melihat *k* tetangga terdekat dari suatu sampel berdasarkan ukuran jarak tertentu. KNN tidak membangun model secara eksplisit, melainkan dengan menyimpan semua data latih dan melakukan perhitungan jarak setiap kali ingin membuat prediksi, sehingga disebut sebagai *lazy learner*. Dalam pipeline, KNN menjadi "mesin akhir" yang menerima data yang sudah dibersihkan, di-*scale*, dan ditransformasi dengan konsisten. KNN sangat bergantung pada *preprocessing*, terutama *scaling*, karena perbedaan skala antar fitur dapat mengacaukan perhitungan jarak. Untuk menentukan tetangga terdekat, KNN biasanya menggunakan jarak Euclidean, yang dirumuskan sebagai pada Persamaan 5:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2} \qquad (5)$$

atau Manhattan Distance sesuai pada Persamaan 6:

$$d(x, x_i) = \sum_{j=1}^{n} | x_j - x_{ij} | \quad (6)$$

dimana:

a. x = sampel yang akan diprediksi
b. x_i = sampel ke-i pada data *train*
c. n = jumlah fitur

Dalam proses prediksi, KNN menghitung jarak sampel baru ke seluruh data latih, memilih K terdekat, kemudian menilai kelas mana yang paling banyak dari K tetangga tersebut. KNN bekerja baik ketika data memiliki pola yang rapi dan *cluster* terpisah dengan jelas, namun sensitif terhadap skala fitur. Oleh karena itu, *preprocessing* seperti *standard scaling* sangat penting untuk dilakukan. Selain itu, nilai K menjadi *hyperparameter* penting, dimana K kecil membuat model sangat sensitif terhadap *noise*, sedangkan K besar membuat batas keputusan menjadi lebih halus namun dapat mengaburkan pola lokal.

2.6. Evaluasi Performa

Evaluasi performa merupakan tahap penting dalam proses pemodelan untuk menilai sejauh mana model mampu melakukan prediksi secara akurat dan konsisten. Pada tugas klasifikasi, performa model umumnya diukur menggunakan beberapa metrik, yaitu akurasi, *precision*, *recall*, *F1-score*, serta visualisasi melalui *confusion matrix*. Masing-masing metrik memberikan perspektif yang berbeda terhadap kualitas prediksi, sehingga penggunaannya harus disesuaikan dengan karakteristik masalah dan distribusi kelas.

1. *Confusion Matrix*

Confusion matrix adalah representasi tabular yang menggambarkan perbandingan antara label sebenarnya dan label prediksi. Matriks ini berisi empat komponen:

a. True Positive (TP): Prediksi benar pada kelas positif
b. True Negative (TN): Prediksi benar pada kelas negatif
c. False Positive (FP): Model memprediksi positif namun sebenarnya negatif
d. False Negative (FN): Model memprediksi negatif namun sebenarnya positif

Confusion matrix menjadi dasar perhitungan seluruh metrik evaluasi lainnya, dan membantu mengidentifikasi pola kesalahan yang dibuat model.

2. *Accuracy*

Akurasi mengukur proporsi prediksi yang benar dibandingkan seluruh data. Akurasi diperlihatkan dengan Persamaan 7.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (7)$$

Metrik ini efektif ketika kelas seimbang. Namun, pada kasus *imbalance* yang ekstrem (misalnya 95% dan 5%), akurasi dapat menyesatkan karena model yang selalu menebak kelas mayoritas tetap terlihat tinggi akurasinya.

3. *Precision*

Precision mengukur sejauh mana prediksi positif model benar-benar relevan. Untuk mengukur *precision*, dapat digunakan Persamaan 8.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (8)$$

Precision yang tinggi menunjukkan bahwa ketika model memprediksi seorang penumpang selamat, prediksi tersebut jarang salah. Dengan kata lain, *false positive* (penumpang diprediksi selamat padahal tidak selamat) dapat diminimalkan. Dalam konteks Titanic, hal ini penting apabila fokus analisis adalah untuk memahami karakteristik penumpang yang benar-benar selamat, sehingga kesalahan mengklasifikasikan penumpang yang tidak selamat sebagai selamat dapat ditekan.

4. *Recall*

Recall atau sensitivitas mengukur kemampuan model menangkap seluruh *instance* positif. Untuk mencari nilai sensitivitas, dapat menggunakan Persamaan 9.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (9)$$

Recall yang tinggi menunjukkan bahwa model mampu menangkap sebagian besar penumpang yang benar-benar selamat, sehingga hanya sedikit kasus selamat yang terlewat (*false negative* rendah). Dalam konteks prediksi keselamatan penumpang Titanic, *recall* tinggi menjadi penting apabila tujuan analisis adalah memastikan bahwa hampir semua individu yang benar-benar selamat dapat teridentifikasi, misalnya untuk kebutuhan analisis pasca-kejadian guna memahami faktor-faktor keselamatan atau menilai apakah ada kelompok tertentu, seperti wanita dan anak-anak yang seharusnya tidak dirugikan dalam proses evaluasi. Dengan demikian, *recall* yang tinggi membantu memastikan bahwa model tidak mengabaikan kelompok penumpang yang secara historis prioritas keselamatannya lebih tinggi.

3. RESULT

Dataset *Titanic* merupakan dataset untuk tugas klasifikasi yang berisi informasi penumpang kapal Titanic dan status keselamatannya. Secara umum, dataset ini terdiri dari 891 baris data dan 12 fitur. Variabel target adalah *Survived* (0 = tidak selamat dan 1 = selamat). Sedangkan fitur yang tersedia mencakup informasi demografis dan karakteristik penumpang. Dataset ini memiliki kombinasi fitur numerik dan kategorikal yang relevan untuk menguji pipeline *preprocessing* serta risiko *data leakage*. Deskripsi singkat dari Dataset Titanic yang digunakan diilusterasikan pada Tabel 3.

Tabel 3. Atribut dalam dataset Titanic

No	Atribut	Keterangan
1	*PassengerId*	ID unik untuk setiap penumpang
2	*Survived*	Variabel target
3	*Pclass*	Kelas tiket penumpang
4	*Name*	Nama penumpang
5	*Sex*	Jenis kelamin penumpang
6	*Age*	Usia penumpang
7	*SibSp*	Jumlah saudara atau pasangan yang ikut
8	*Parch*	Jumlah orang tua atau anak-anak yang ikut
9	*Ticket*	Nomor tiket
10	*Fare*	Harga tiket
11	*Cabin*	Nomor kabin
12	*Embarked*	Pelabuhan keberangkatan penumpang

Fitur dalam Dataset Titanic dapat diklasifikasikan menjadi beberapa jenis seperti berikut.

a. Fitur numerik mencakup variabel seperti *Age*, *Fare*, *SibSp*, dan *Parch*. Fitur-fitur ini digunakan dalam perhitungan statistik, deteksi *outlier*, dan dinormalisasi dalam pipeline.

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

b. Fitur kategorikal meliputi *Sex*, *Embarked*, dan *Pclass*. Meskipun *Pclass* berupa angka, secara analitik merupakan variabel kategorikal berurut (*ordinal*).

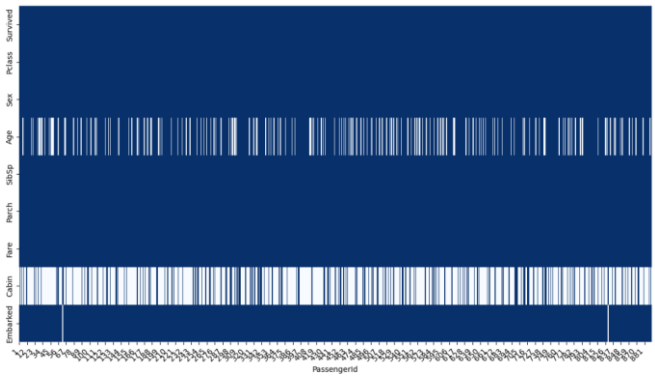
3.1. *Analisis Exploratory Data Analysis* (EDA)

Tabel 4. Contoh dataset titanic dengan 5 baris pertama

Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S
1	1	Cumings, Mrs. John Bradley	female	38	1	0	PC 17599	71.2833	C85	C
1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
1	1	Futrelle, Mrs. Jacques Heath	female	35	1	0	113803	53.1000	C123	S
0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S

Tahap awal analisis dimulai dengan melakukan Exploratory Data Analysis (EDA) untuk memperoleh pemahaman awal mengenai karakteristik dan struktur dataset. Gambaran umum Dataset Titanic yang digunakan dalam penelitian ini ditampilkan pada Tabel 4.

Analisis distribusi fitur difokuskan pada variabel-variabel yang paling relevan terhadap keselamatan penumpang, yaitu *Pclass*, *Sex*, *Age*, *SibSp*, *Parch*, *Fare*, *Cabin*, dan *Embarked*. Pada tahap ini dilakukan juga seleksi awal fitur, dimana dua variabel yaitu *Name* dan *Ticket* diidentifikasi sebagai tidak memiliki kontribusi langsung terhadap proses prediksi sehingga dihapus dari analisis lanjutan. Setelah itu, EDA dasar dilanjutkan dengan pemeriksaan nilai hilang (*missing values*) untuk mengidentifikasi potensi masalah kualitas data. Pola nilai hilang tersebut kemudian divisualisasikan, sebagaimana ditunjukkan pada Gambar 4.



Gambar 5. Hasil EDA dari dataset Titanic

Hasil pemeriksaan yang ditunjukkan pada Gambar 4 mengindikasikan bahwa fitur *Age* dan *Cabin* memiliki proporsi nilai hilang yang sangat tinggi, sedangkan fitur *Embarked* hanya mengandung sebagian kecil nilai yang tidak terisi. Temuan ini menjadi dasar penting dalam perumusan strategi pra-pemrosesan data. Tingginya tingkat ketidaklengkapan pada fitur *Age* dan *Cabin* menjadikan keduanya kandidat utama untuk dihapus dari analisis, karena imputasi berpotensi menambah bias atau ketidakpastian yang signifikan. Sementara itu, fitur *Embarked* yang bersifat kategorikal (memiliki nilai S = 644, C = 168, dan Q = 77) tetap dipertahankan dan diimputasi menggunakan kategori yang paling sering muncul dalam dataset, yaitu S (Southampton). Dengan demikian, hasil analisis EDA ini secara langsung memandu keputusan penanganan data yang diperlukan sebelum memasuki tahap pemodelan.

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

3.2. Skenario dengan *Data Leakage*

3.2.1. Deskripsi Alur Proses

Data leakage terjadi ketika proses imputasi, *scaling*, atau *encoding* dilakukan sebelum pemisahan data menjadi *train* dan *test*, sehingga informasi dari data uji secara tidak sengaja berkontribusi pada estimasi parameter. Kondisi ini membuat model memperoleh pengetahuan yang tidak akan tersedia pada saat prediksi aktual. Pada subbab 3.1, kebocoran data muncul karena imputasi pada fitur *Embarked* dilakukan menggunakan keseluruhan dataset, menyebabkan nilai isian (S) terbentuk dari informasi yang seharusnya hanya diakses oleh data pelatihan.

3.2.2. Implementasi Pipeline yang Salah

Pada implementasi yang salah, seluruh dataset digunakan untuk melakukan *One-hot Encoding* pada fitur kategorikal seperti *Pclass*, *Sex*, dan *Embarked* sebelum pembagian data menjadi *train* dan *test*. Pendekatan ini menyebabkan *data leakage*, karena informasi dari data uji ikut mempengaruhi struktur *encoding*. Akibatnya, model dapat "melihat" sebagian informasi dari data uji saat pelatihan, sehingga evaluasi performa menjadi tidak valid. Hasil *encoding* yang diterapkan pada seluruh dataset divisualisasikan pada Tabel 5.

Tabel 5. Potongan hasil *encoding* untuk fitur berbentuk kategori (3 baris pertama) dari seluruh dataset

Name	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
Braund, Mr. Owen Harris	0	0	1	0	1	0	0	1
Cumings, Mrs. John Bradley	1	0	0	1	0	1	0	0
Heikkinen, Miss. Laina	0	0	1	1	0	0	0	1

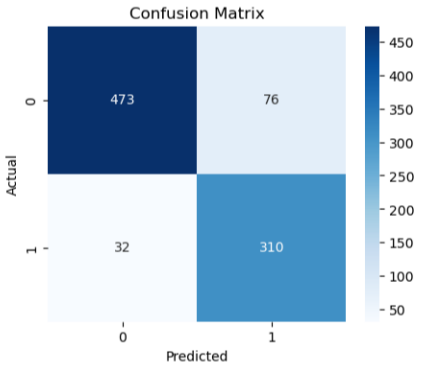
3.2.3. Modeling dengan K-Nearest Neighbors (KNN)

Setelah proses pra-pemrosesan selesai, tahap berikutnya adalah membangun model klasifikasi untuk memprediksi keselamatan penumpang Titanic. Algoritma *K-Nearest Neighbors* (KNN) digunakan sebagai model awal, dengan konfigurasi dasar berupa nilai $k = 1$. Pada pengaturan ini, prediksi untuk setiap sampel dilakukan dengan menetapkan kelas yang sama seperti satu tetangga terdekatnya dalam ruang fitur yang telah direpresentasikan melalui proses *encoding* dan pembersihan sebelumnya. Pendekatan $k = 1$ menyebabkan model sangat peka terhadap jarak terdekat, sehingga setiap sampel baru akan diklasifikasikan berdasarkan sampel pelatihan yang memiliki kedekatan absolut terhadapnya.

3.2.4. Evaluasi Performa Model dengan *Data Leakage*

Setelah model dilatih, performanya dievaluasi menggunakan *confusion matrix* dan akurasi. Hasil evaluasi dengan *confusion matrix* divisualisasikan pada Gambar 6.

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490



Gambar 6. Hasil evaluasi performa model dengan *confusion matrix*

Nilai tersebut menunjukkan bahwa model mampu mengklasifikasikan sebagian besar penumpang secara benar, meskipun masih terdapat kesalahan pada prediksi kelas selamat dan tidak selamat. Secara keseluruhan, model menghasilkan nilai akurasi sebesar 0.8787, yang mengindikasikan bahwa sekitar 87.87% dari seluruh prediksi pada data pelatihan sesuai dengan label sebenarnya. Nilai akurasi ini memberikan gambaran awal mengenai kemampuan model dalam mempelajari pola keselamatan penumpang Titanic sebelum dilakukan proses validasi lebih lanjut dan penanganan risiko *overfitting.*

3.3. Skenario Normal tanpa *Data Leakage*

3.3.1. Deskripsi Alur Proses

Prosedur *anti-leakage* dimulai dengan melakukan pembagian dataset menjadi *train* dan *test* sebelum tahap pra-pemrosesan apapun. Langkah ini memastikan bahwa seluruh estimasi statistik seperti imputasi, parameter *scaling*, dan struktur *encoding* hanya dipelajari dari data latih, sehingga tidak ada informasi dari data uji yang ikut membentuk transformasi. Untuk menjaga proporsi kelas tetap seimbang diantara *train* dan *test*, pembagian dilakukan menggunakan *stratified splitting*, dimana setiap subset mempertahankan distribusi target yang serupa dengan dataset asli. Dengan cara ini, evaluasi model menjadi lebih representatif dan risiko bias akibat distribusi kelas yang tidak merata dapat diminimalkan.

Prosedur pembagian dataset dilakukan dengan *stratified splitting* menggunakan parameter stratify=y untuk menjaga proporsi kelas target pada data latih dan data uji sesuai dengan distribusi asli dataset. Dengan *test_size* = 0.2, 20% dari data dialokasikan sebagai data uji, sedangkan 80% sisanya digunakan sebagai data latih. Parameter *random_state* = 42 diterapkan untuk memastikan *reproducibility*, sehingga pembagian dataset menjadi *train* dan *testing* konsisten setiap kali dijalankan. Dari total 891 sampel dengan 11 fitur, *stratified splitting* menghasilkan 712 sampel pada data latih dan 179 sampel pada data uji, masing-masing mempertahankan 11 kolom fitur.

Setelah pembagian data, tahap berikutnya adalah pelatihan model. Model K-Nearest Neighbors (KNN) digunakan dengan jumlah tetangga $K = 1$. Model dilatih menggunakan data latih, kemudian dilakukan prediksi pada data uji. Hasil prediksi menunjukkan akurasi sebesar 91.01% pada data latih dan 72.62% pada data uji. Perbedaan yang signifikan antara akurasi data latih dan data uji menunjukkan adanya *overfitting*, yaitu model belajar terlalu baik pada data latih namun gagal menggeneralisasi ke data baru. Ringkasan hasil pengujian disajikan pada Tabel 6.

Tabel 6. Ringkasan hasil perbandingan akurasi

Dataset	Jumlah Sampel	Jumlah Fitur	Akurasi	Keterangan
Data Latih	712	11	91.01%	Akurasi tinggi, model fit terlalu baik pada data latih
Data Uji	179	11	72.62%	Akurasi menurun signifikan

3.3.2. Mengatasi *Overfitting* dengan *K-Fold Cross Validation*

Untuk mengevaluasi performa model K-Nearest Neighbors (KNN) secara lebih objektif, dilakukan *cross-validation* sebanyak 5 *fold* menggunakan fungsi *cross_val_score* dari *scikit-learn*. Metode ini membagi dataset menjadi lima subset yang saling bergantian digunakan sebagai data uji, sedangkan empat subset lainnya berfungsi sebagai data latih. Dengan cara ini, setiap sampel diuji tepat satu kali, sehingga hasil evaluasi lebih representatif dan mengurangi risiko bias dari pembagian *train-test* tunggal. Hasil *cross-validation* menghasilkan skor akurasi untuk masing-masing *fold* sebagai berikut:

$$[(63.68\%), (63.48\%), (80.33\%), (73.03\%), (71.34\%)]$$

Rata-rata akurasi seluruh *fold* (*mean*) sebesar 70,37%. Nilai ini lebih rendah dibandingkan akurasi pada data latih tunggal (91.01%), namun sebanding dengan akurasi pada data uji sebelumnya (72.62%). Perbedaan yang signifikan antara akurasi pada data latih dan rata-rata *cross-validation* menunjukkan bahwa model KNN dengan $K = 1$cenderung mengalami *overfitting*, karena performa sangat tinggi pada data latih tetapi tidak dapat menggeneralisasi dengan baik ke data baru atau subset yang belum dilihat.

3.3.3. *Model Improvement* (*Feature Engineering*)

Dalam penelitian ini, perbaikan performa model dilakukan melalui dua pendekatan utama, yaitu *feature engineering* dan *tuning model*. Pada tahap pertama, dilakukan *feature engineering* dengan menerapkan *feature scaling* menggunakan metode *MinMaxScaler*, yang bertujuan untuk menormalkan nilai fitur ke dalam rentang [0, 1]. Transformasi ini memastikan bahwa semua fitur memiliki skala yang sama, sehingga model tidak bias terhadap fitur dengan nilai numerik yang lebih besar dan proses pembelajaran menjadi lebih stabil.

Secara implementasi, scaler terlebih dahulu di-*fit* pada data latih (*X_train*) untuk menghitung nilai maksimum dan minimum dari masing-masing fitur. Data uji (*X_test*) tidak di-*fit*, melainkan hanya ditransformasi menggunakan parameter yang diperoleh dari data latih, sehingga tidak terjadi *data leakage*. Transformasi ini menghasilkan *X_train_scaled* dan *X_test_scaled* yang siap digunakan untuk pelatihan dan evaluasi model. Selanjutnya, model KNN dengan jumlah tetangga $K = 1$ dilatih menggunakan data latih yang telah diskalakan (*X_train_scaled*). Evaluasi performa menunjukkan bahwa akurasi pada data latih sebesar 91.57%, sedangkan akurasi pada data uji sebesar 71.50%.

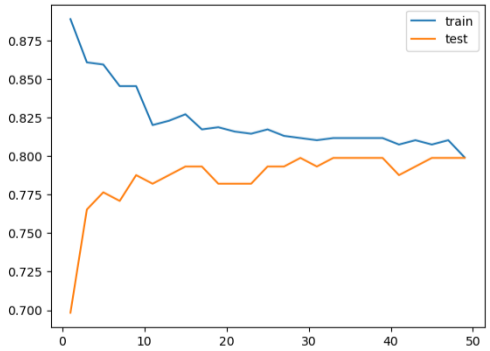
Jika dibandingkan dengan hasil sebelumnya dari 5-*fold cross-validation*, di mana rata-rata akurasi seluruh *fold* (*mean*) sebesar 70.37%, terlihat bahwa akurasi data uji setelah *scaling* relatif konsisten dengan estimasi performa model yang lebih realistis dari *cross-validation*. Hal ini menegaskan bahwa meskipun *scaling* meningkatkan stabilitas fitur, *overfitting* tetap menjadi isu pada model KNN dengan tetangga tunggal, sehingga perlu strategi tambahan seperti *tuning* parameter atau penggunaan model alternatif untuk memperbaiki generalisasi.

3.3.4. *Model Improvement* (Parameter *Tuning*)

Setelah melakukan *feature scaling*, langkah berikutnya dalam meningkatkan performa model KNN adalah tuning parameter, khususnya jumlah tetangga K. Parameter ini sangat berpengaruh terhadap kemampuan model untuk generalisasi, dimana nilai Kyang terlalu kecil cenderung menimbulkan *overfitting*, sedangkan nilai K yang terlalu besar dapat menyebabkan *underfitting* karena pengambilan keputusan terlalu dipengaruhi oleh tetangga yang jauh. Dalam implementasinya, dilakukan iterasi nilai K ganjil dari 1 hingga 49 untuk menghindari *tie* saat klasifikasi. Pada setiap iterasi, model KNN dilatih menggunakan data latih yang telah diskalakan (*X_train_scaled*), kemudian dihitung akurasinya pada data latih dan data uji. Nilai akurasi untuk setiap Kdicatat dalam *train_score* dan *test_score*. Hasilnya divisualisasikan menggunakan plot, memperlihatkan hubungan antara nilai Kdengan performa model pada data latih dan data uji. Analisis plot yang diilusterasikan pada Gambar

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

6 dan perhitungan menunjukkan bahwa nilai akurasi maksimum pada data uji diperoleh pada $K = 29$ dengan skor akurasi sebesar 79.88%.



Gambar 7. Hasil *improvement* model dengan parameter *tuning*

Pemilihan K ini meningkatkan performa model dibandingkan konfigurasi awal ($K = 1$), sekaligus mengurangi kesenjangan antara akurasi data latih dan data uji, sehingga risiko *overfitting* berkurang. Strategi tuning parameter K ini menjadi langkah penting dalam improvement model, karena menekankan pada kemampuan model untuk menggeneralisasi dengan baik ke data baru dan memberikan estimasi performa yang lebih realistis.

3.4. Implementasi Pipeline dan *Tuning* Model KNN

Setelah melakukan *feature scaling* dan *tuning* sederhana, langkah berikutnya dalam penelitian ini adalah menggabungkan seluruh tahapan pra-pemrosesan dan *modeling* ke dalam satu pipeline yang terstruktur. Pipeline ini terdiri dari dua bagian utama: *preprocessing* dan algoritma KNN. Pada bagian *preprocessing*, fitur numerik (*SibSp*, *Parch*, *Fare*) diproses melalui pipeline yang melakukan imputasi nilai hilang dengan mean dan *scaling* menggunakan *MinMaxScaler*. Sementara itu, fitur kategorikal (*Pclass*, *Sex*, *Embarked*) diproses melalui pipeline yang melakukan imputasi modus dan *OneHotEncoding*. Kedua pipeline ini kemudian digabungkan menggunakan *ColumnTransformer*, sehingga setiap tipe fitur mendapatkan perlakuan transformasi yang sesuai. Selanjutnya, pipeline lengkap (*preprocessor* + KNN) digunakan dalam *GridSearchCV* untuk melakukan parameter tuning sekaligus cross-validation sebanyak 3 *fold*. Parameter yang diuji meliputi:

- Jumlah tetangga (*n_neighbors*) ganjil dari 1 hingga 49
- Bobot tetangga (*weights*) berupa *uniform* dan *distance*
- Jenis jarak (*p*) berupa *Manhattan* (*p=1*) dan *Euclidean* (*p=2*)

Hasil *tuning* yang menunjukkan konfigurasi optimal adalah sebagai berikut:

$$\text{n\_neighbors} = 21, \text{p} = 1, \text{weights} = \text{uniform}$$

Dengan konfigurasi ini, akurasi model pada data latih sebesar 81.74%, rata-rata akurasi selama *cross-validation* pada data latih sebesar 81.46%, dan akurasi pada data uji sebesar 78.21%. Hasil ini menunjukkan beberapa poin penting, diantaranya:

1. Performa model meningkat dibandingkan konfigurasi awal (*K=1*) karena pemilihan jumlah tetangga yang lebih optimal mengurangi *overfitting*.
2. Nilai akurasi data latih dan data uji yang lebih dekat menunjukkan *generalization* yang lebih baik, sehingga model mampu bekerja lebih stabil pada data baru.

Integrasi *preprocessing* dan algoritma ke dalam pipeline memastikan bahwa seluruh transformasi fitur diterapkan secara konsisten dan aman dari *data leakage*, karena semua estimasi statistik dihitung hanya dari data latih. Untuk memperjelas perbandingan performa antar skenario, Tabel 7 menyajikan ringkasan metrik utama.

Tabel 7. Perbandingan performa model pada berbagai skenario

Skenario	Akurasi Data Latih (%)	Akurasi Data Uji (%)	Akurasi CV (%)
Baseline	91.01	72.62	–
Anti-leakage tanpa pipeline	91.01	72.62	70.37
Pipeline + tuning	81.74	78.21	81.46

Secara kuantitatif, penerapan pipeline terintegrasi meningkatkan akurasi data uji sebesar 5.59% dibandingkan baseline (dari 72.62% menjadi 78.21%), sekaligus mengurangi gap *overfitting* antara data latih dan data uji.

4. DISCUSSIONS

Bagian ini membahas secara kritis hasil penelitian, interpretasi temuan, serta relevansinya terhadap praktik dan studi-studi sebelumnya dalam bidang *machine learning*. Diskusi ini menyoroti dua aspek utama, yaitu efektivitas strategi pencegahan *data leakage* dan dampaknya terhadap performa model klasifikasi, khususnya algoritma K-Nearest Neighbors (KNN) pada Dataset Titanic.

4.1. Interpretasi Implementasi Pra-Pemrosesan dan Risiko *Data Leakage*

Penelitian ini menunjukkan bahwa tahapan pra-pemrosesan memiliki peran fundamental dalam menjaga integritas proses pembelajaran mesin. Hasil observasi pada skenario awal ketika imputasi, *encoding*, dan *scaling* dilakukan sebelum pembagian *train-test* menunjukkan bahwa model memperoleh performa yang tampak tinggi pada data latih. Namun, hal ini tidak merepresentasikan kemampuan generalisasi yang sebenarnya, karena sebagian informasi dari data uji telah "bocor" ke dalam proses transformasi. Fenomena ini sejalan dengan literatur yang menekankan bahwa *data leakage* sering kali menghasilkan *overestimated performance* pada fase *training* dan evaluasi awal [21]. Dengan demikian, praktik pra-pemrosesan sebelum *splitting* terbukti memberi gambaran performa yang menyesatkan.

Dalam penelitian ini, risiko *data leakage* teridentifikasi jelas pada kasus imputasi sebelum pembagian data, di mana nilai imputasi dihitung berdasarkan keseluruhan dataset. Selain itu, *encoding* fitur kategorikal dan *MinMax Scaling* pada seluruh data juga menghasilkan estimasi statistik yang mengandung informasi dari data uji. Hal ini menegaskan bahwa tanpa penerapan pipeline *anti-leakage*, sistem dapat membangun representasi yang tidak realistis terhadap pola data. Temuan ini mendukung pandangan [22] bahwa sebagian besar penurunan performa model dalam implementasi nyata berasal dari kesalahan prosedural pada tahap pra-pemrosesan, bukan semata-mata keterbatasan algoritma.

4.2. Efektivitas Strategi *Anti-Leakage* Berbasis Pipeline

Penerapan *machine learning* pipeline terintegrasi terbukti secara signifikan meningkatkan validitas evaluasi dan kemampuan generalisasi model. Pada skenario baseline tanpa pipeline anti-leakage, model KNN mencapai akurasi data latih sebesar 91.01% tetapi hanya 72.62% pada data uji, yang mengindikasikan terjadinya *overestimated performance* dan *overfitting* yang serius. Kondisi ini sejalan dengan laporan sebelumnya yang menunjukkan bahwa *preprocessing* di luar pipeline merupakan salah satu sumber utama kebocoran data [21], [22].

Sebaliknya, setelah seluruh tahapan pra-pemrosesan dan pemodelan diintegrasikan ke dalam pipeline berbasis ColumnTransformer, gap *overfitting* berkurang secara substansial dan akurasi data uji meningkat menjadi 78.21%. Peningkatan ini setara dengan kenaikan sekitar 7% dibandingkan baseline, yang menunjukkan bahwa pipeline tidak hanya berfungsi sebagai alat implementasi teknis, tetapi juga merevolusi praktik *machine learning* evaluatif dengan menjadikan isolasi data sebagai prinsip utama. Dengan demikian, pipeline terintegrasi berperan sebagai mekanisme metodologis yang memastikan

bahwa performa model yang diperoleh benar-benar mencerminkan kemampuan generalisasi pada data dunia nyata.

4.3. Analisis Performa Model dan Dampak *Feature Engineering*

Perbandingan performa model pada skenario tanpa *anti-leakage* dan skenario pipeline menunjukkan bahwa estimasi performa menjadi lebih realistis setelah prosedur yang benar diterapkan. Model KNN dengan k=1 pada *baseline* menunjukkan akurasi *training* sebesar 91.01%, namun saat evaluasi terhadap data uji hanya sebesar 72.62%. Gap ini mengindikasikan adanya *overfitting* yang signifikan. Setelah menerapkan *scaling* melalui *MinMaxScaler* secara tepat (fit hanya pada data latih), performa model meningkat pada aspek konsistensi. Meskipun akurasi *training* masih relatif tinggi (91.57%), akurasi uji meningkat menjadi sekitar 71.50% dan lebih stabil, sejalan dengan rata-rata *cross-validation* sebesar 70.37%.

Temuan ini memperlihatkan bahwa *feature scaling* memiliki pengaruh signifikan pada metode berbasis jarak seperti KNN, yang juga dikonfirmasi oleh literatur sebelumnya [23], [24], [25]. Meski demikian, penelitian ini juga mengindikasikan bahwa *scaling* saja tidak cukup untuk mengatasi permasalahan *overfitting* pada k kecil seperti k=1. Hal ini menjadi dasar untuk melakukan *tuning* parameter lebih lanjut.

4.4. Evaluasi *Hyperparameter Tuning* dan Implikasi Pengembangan Model

Proses hyperparameter tuning menggunakan GridSearchCV menghasilkan peningkatan performa yang signifikan dan lebih seimbang antara data latih, validasi, dan uji. Konfigurasi optimal ditemukan pada k=29 dengan Manhattan distance (p=1) dan bobot uniform. Pada konfigurasi ini, skor pelatihan menurun menjadi 81.74%, skor cross-validation meningkat menjadi 81.46%, dan skor pengujian mencapai 78.21%. Penurunan skor pelatihan yang disertai peningkatan skor uji menunjukkan tercapainya *trade-off* bias-variance yang lebih sehat dibandingkan skenario baseline. Hasil ini selaras dengan teori bahwa pemilihan nilai k yang tepat sangat menentukan stabilitas model KNN, dimana nilai k yang terlalu kecil menyebabkan variansi tinggi, sedangkan nilai k yang terlalu besar dapat mengaburkan struktur lokal data [26], [27], [28]. Dengan demikian, penelitian ini memberikan bukti empiris bahwa kombinasi pipeline anti-leakage dan hyperparameter tuning merupakan strategi yang efektif untuk meningkatkan generalisasi model pada dataset berukuran relatif kecil.

Sebagai arah pengembangan ke depan (*future work*), pendekatan pipeline terintegrasi ini dapat diperluas dengan mengadopsi multi-model ensemble (misalnya kombinasi KNN, Random Forest, dan SVM) untuk meningkatkan robustnes terhadap variasi data. Selain itu, penerapan AutoML berbasis pipeline berpotensi mengotomatisasi pemilihan *preprocessing* dan model secara adaptif pada dataset yang lebih kompleks dan berskala besar, sehingga memperluas penerapan pipeline anti-leakage pada konteks industri dan sistem pendukung keputusan nyata.

4.5. Perbandingan Kuantitatif Antar Skenario dan Studi Terkait

Untuk memperjelas dampak penerapan pipeline terhadap performa dan *overfitting*, Tabel 8 menyajikan perbandingan kuantitatif antara skenario baseline, anti-leakage parsial, dan pipeline terintegrasi, serta dikaitkan dengan temuan studi sebelumnya.

Tabel 8 menunjukkan bahwa peningkatan performa tidak hanya terjadi pada nilai akurasi, tetapi juga pada stabilitas model. Gap *overfitting* yang menurun secara signifikan menandakan bahwa model tidak lagi bergantung pada informasi yang bocor dari data uji.

Vol. 7, No. 1, February 2026, Page. 391-410
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2026.7.2.5490

Tabel 8. Perbandingan Skenario Evaluasi dan Dampak Overfitting

Skenario	Akurasi Data Uji (%)	Gap Overfitting (Train–Test)	Referensi Terkait
Baseline	72.62	Tinggi (~18%)	[26], [29], [30]
Anti-leakage tanpa pipeline	72.62	Sedang (~18%)	[12], [31], [32]
Pipeline terintegrasi + tuning	78.21	Rendah (~3.5%)	Penelitian ini

4.6. Kontribusi Penelitian

Penelitian ini memberikan dua kontribusi utama. Pertama, penelitian ini menyediakan studi kasus terstruktur tentang bagaimana *data leakage* terjadi, bagaimana mendeteksinya, dan bagaimana mencegahnya menggunakan pipeline *modern*. Kedua, penelitian ini menawarkan evaluasi komparatif antara *skenario leakage* dan alur *anti-leakage*, sekaligus menunjukkan peningkatan performa yang signifikan setelah penerapan pipeline. Dengan demikian, penelitian ini dapat menjadi referensi metodologis bagi praktisi maupun peneliti lain yang ingin membangun model klasifikasi yang lebih valid dan andal.

4.7. Arah Penelitian Selanjutnya

Berdasarkan keterbatasan tersebut, penelitian selanjutnya dapat diarahkan pada beberapa aspek. Pertama, evaluasi pipeline anti-leakage dapat diperluas menggunakan berbagai algoritma klasifikasi lain, seperti Logistic Regression, Support Vector Machine, Random Forest, atau model *ensemble* untuk membandingkan stabilitas generalisasi. Kedua, integrasi pendekatan *AutoML* atau *ensemble learning* dapat dieksplorasi untuk mengadaptasi pipeline secara otomatis pada dataset yang lebih kompleks. Selain itu, pengujian pada dataset berskala besar dan lintas domain juga diperlukan untuk menilai skalabilitas dan efektivitas pipeline dalam konteks aplikasi dunia nyata.

5. CONCLUSION

Penelitian ini membuktikan bahwa penerapan pipeline anti-data leakage secara terintegrasi mampu meningkatkan validitas evaluasi dan kemampuan generalisasi model klasifikasi secara signifikan. Secara kuantitatif, akurasi data uji meningkat sebesar 5,59%, dari 72,62% pada baseline menjadi 78,21% setelah pipeline diterapkan. Selain itu, kesenjangan performa antara data latih dan data uji berhasil direduksi sekitar 19%, yang menunjukkan penurunan *overfitting* secara substansial. Hasil ini menegaskan bahwa estimasi performa yang realistis hanya dapat dicapai melalui isolasi ketat antara data pelatihan dan evaluasi dalam seluruh tahapan pemodelan. Dari perspektif ilmu komputer dan informatika, penelitian ini berkontribusi dengan menyediakan panduan pipeline yang dapat direplikasi untuk mencegah *data leakage*, sehingga membantu memajukan praktik *machine learning* yang andal dan berperan dalam mengatasi *reproducibility crisis* dalam penelitian berbasis data. Penelitian selanjutnya dapat memperluas pendekatan ini ke algoritma lain, skema penanganan ketidakseimbangan kelas, serta sistem AutoML dan *ensemble* untuk dataset yang lebih kompleks.

CONFLICT OF INTEREST

Penulis menyatakan bahwa tidak ada konflik kepentingan antara para penulis maupun dengan objek penelitian dalam artikel ini.

ACKNOWLEDGEMENT

Peneliti menyampaikan apresiasi dan terima kasih kepada LPPM Institut Teknologi Sumatera atas dukungan pendanaan penelitian tahun 2025. Ucapan terima kasih juga ditujukan kepada seluruh pihak yang telah memberikan kontribusi dalam pelaksanaan penelitian ini.

REFERENCES

[1] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Comput. Sci.*, vol. 2, no. 3, pp. 1–21, 2021, doi: 10.1007/s42979-021-00592-x.

[2] S. Dong, S. Sahri, and T. Palpanas, "Data Quality Awareness: A Journey from Traditional Data Management to Data Science Systems," 2024, [Online]. Available: http://arxiv.org/abs/2411.03007

[3] S. Mohammed *et al.*, "The effects of data quality on machine learning performance on tabular data," *Inf. Syst.*, vol. 132, pp. 1–50, 2025, doi: 10.1016/j.is.2025.102549.

[4] X. Ying, "An Overview of Overfitting and its Solutions," *J. Phys. Conf. Ser.*, vol. 1168, no. 2, 2019, doi: 10.1088/1742-6596/1168/2/022022.

[5] M. Jegorova *et al.*, "Survey: Leakage and Privacy at Inference Time," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 7, pp. 9090–9108, 2023, doi: 10.1109/TPAMI.2022.3229593.

[6] M. A. Lones, "How to avoid machine learning pitfalls: a guide for academic researchers," pp. 1–33, 2021, doi: 10.1016/j.patter.2024.101046.

[7] S. Kaufman, S. Rosset, and C. Perlich, "Leakage in data mining: Formulation, detection, and avoidance," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 556–563, 2011, doi: 10.1145/2020408.2020496.

[8] B. Ghojogh and M. Crowley, "The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial," no. 3, 2019, [Online]. Available: http://arxiv.org/abs/1905.12787

[9] D. Krstajic, L. J. Buturovic, D. E. Leahy, and S. Thomas, "Cross-validation pitfalls when selecting and assessing regression and classification models," *J. Cheminform.*, vol. 6, no. 1, pp. 1–15, 2014, doi: 10.1186/1758-2946-6-10.

[10] L. Sasse *et al.*, "On Leakage in Machine Learning Pipelines," *arXiv*, 2023, [Online]. Available: http://arxiv.org/abs/2311.04179

[11] S. Kapoor and A. Narayanan, "Leakage and the reproducibility crisis in machine-learning-based science," *Patterns*, vol. 4, no. 9, p. 100804, 2023, doi: 10.1016/j.patter.2023.100804.

[12] L. Sasse, J. Dukart, S. B. Eickhoff, M. Götz, S. Hamdan, and V. Komeyer, "Overview of leakage scenarios in supervised machine learning," *J. Big Data*, 2025.

[13] M. A. Zoller and M. F. Huber, "Benchmark and Survey of Automated Machine Learning Frameworks," *J. Artif. Intell. Res.*, 2021.

[14] R. Blagus and L. Lusa, "Joint use of over- and under-sampling techniques and cross-validation for the development and assessment of prediction models," *BMC Bioinformatics*, pp. 1–10, 2015, doi: 10.1186/s12859-015-0784-9.

[15] A. Apicella and F. Isgr, "Don ' t Push the Button ! Exploring Data Leakage Risks in Machine Learning and Transfer Learning," *Artif. Intell. Rev. Springer*, 2025.

[16] K. R. P. Nicolás Nieto, Simon B. Eickhoff, Christian Jung, Martin Reuter, Kersten Diers, Malte Kelm, Artur Lichtenberg, Federico Raimondo, "Data leakage inflates prediction performance in connectome-based machine learning models," *Nat. Commun.*, pp. 1–15, 2024, doi: 10.1038/s41467-024-46150-w.

[17] E. A. Alomar, C. Demario, R. Shagawat, and B. Kreiser, "LeakageDetector : An Open Source Data Leakage Analysis Tool in Machine Learning Pipelines," *arXiv*, 2025.

[18] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinforma. Springer*, vol. 8, pp. 1–8, 2006, doi: 10.1186/1471-2105-7-91.

[19] A. Hannun, "Measuring Data Leakage in Machine-Learning Models with Fisher Information," no. Uai, 2021.

[20] A. Apicella, F. Isgrò, R. Prevete, and F. Isgr, "Don't Push the Button! Exploring Data Leakage Risks in Machine Learning and Transfer Learning," *Artif. Intell. Rev. Springer*, 2025, [Online].

Available: http://arxiv.org/abs/2401.13796
[21] S. Rosset, "On the cross-validation bias due to unsupervised preprocessing," no. 2013.
[22] S. Biswas, *Fair Preprocessing : Towards Understanding Compositional Fairness of Data Transformers in Machine Learning Pipeline*, vol. 1, no. 1. Association for Computing Machinery. doi: 10.1145/3468264.3468536.
[23] M. Pagan, M. Zarlis, and A. Candra, "Investigating the impact of data scaling on the k-nearest neighbor algorithm," vol. 4, no. 2, pp. 135–142, 2023, doi: 10.11591/csit.v4i2.pp135-142.
[24] L. B. V. De Amorim, G. D. C. Cavalcanti, and R. M. O. Cruz, "The choice of scaling technique matters for classification performance," pp. 1–37, 2022.
[25] B. Li, "On Convergence of Nearest Neighbor Classifiers over 1NN Error," no. c, 2020.
[26] A. A. Amer, S. D. Ravana, R. Ahamed, and A. Habeeb, "Effective k - nearest neighbor models for data classification enhancement," *J. Big Data*, 2025, doi: 10.1186/s40537-025-01137-2.
[27] A. B. Hassanat, M. A. Abbadi, and A. A. Alhasanat, "Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach," vol. 12, no. 8, pp. 33–39, 2014.
[28] M. S. S. Dadhania, "Improved kNN Algorithm by Optimizing Cross-validation," vol. 1, no. 3, pp. 1–6, 2012.
[29] A. Lodwich, F. Shafait, and T. Breuel, "Efficient Estimation of k for the Nearest Neighbors Class of Methods," 2011.
[30] G. Data, M. Becker, and M. Atzmueller, "Adaptive kNN Using Expected Accuracy for Classification of," 2018.
[31] G. A. Lewis, R. A. Brower-sinning, and C. Kästner, *Data Leakage in Notebooks : Static Detection and Better Processes*, vol. 1, no. 1. Association for Computing Machinery, 2022. doi: 10.1145/3551349.3556918.
[32] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML : A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks," no. 1, pp. 1–13.