TUGAS BESAR KELOMPOK PML

Tugas Explorasi AutoML Vehicle

disusun oleh:

Bellamy Bintang Pratama	1301210266
Hadziq Umar Azka Mirza	1301210516
Maritza Amalia Dwiputri	1301213399
Muhammad Faqih Abdussalam	1301213056

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

BANDUNG

2023

DAFTAR ISI

1. PENDAHULUAN

1.1. Latar Belakang

Dalam era digital saat ini, penggunaan machine learning telah menjadi salah satu pilar utama dalam mengolah dan menganalisis data. Machine learning memungkinkan pengambilan keputusan yang lebih cepat dan akurat dengan cara memanfaatkan data historis untuk memprediksi kejadian di masa depan. Namun, proses pengembangan model machine learning yang efektif dan efisien seringkali membutuhkan waktu dan usaha yang tidak sedikit. Oleh karena itu, Automated Machine Learning (AutoML) hadir sebagai solusi untuk menyederhanakan proses ini.

AutoML adalah pendekatan untuk mengotomatisasi tugas-tugas machine learning, termasuk pemilihan algoritma, penyesuaian hyperparameter, dan validasi model. Dengan AutoML, para peneliti dan praktisi dapat lebih fokus pada pemahaman masalah dan data, sementara proses teknis dioptimalkan secara otomatis. Dalam tugas ini, kami akan menerapkan dua alat AutoML, yaitu TPOT dan NiaPy, untuk menyelesaikan sebuah task pada dataset yang berbeda. Kami memilih dataset terkait kendaraan untuk dianalisis dan diprediksi menggunakan alat-alat tersebut.

1.2. Tujuan

Dalam tugas ini, tujuan utama kami adalah menerapkan AutoML untuk menyelesaikan sebuah task pada dataset kendaraan. Kami juga bertujuan untuk membandingkan performa dan langkah-langkah dari dua alat AutoML, yaitu TPOT dan NiaPy, serta mengevaluasi hasil model yang dihasilkan oleh kedua alat tersebut dan memberikan analisis mendalam terhadap hasil evaluasi tersebut. Selain itu, kami juga akan memberikan kesimpulan dan saran untuk peningkatan penggunaan AutoML di masa depan berdasarkan pengalaman dan hasil yang diperoleh selama eksperimen.

1.3. Ruang Lingkup

Ruang lingkup tugas kami mencakup beberapa aspek penting. Pertama, setiap kelompok harus memilih dataset yang berbeda untuk memastikan variasi analisis. Selanjutnya, kami akan menerapkan teknik eksplorasi dan persiapan data pada dataset kendaraan yang kami pilih. Kami juga akan mengimplementasikan dan melakukan eksperimen menggunakan TPOT dan NiaPy sebagai alat AutoML. Evaluasi model akan dilakukan menggunakan metode evaluasi yang sesuai, dan kami akan melakukan perbandingan antara kedua alat AutoML dalam hal langkah dan hasil yang dicapai. Akhirnya, kami

akan menyusun laporan akhir yang mencakup kesimpulan dari semua proses yang dijalankan, lesson learned, dan saran untuk improvement ke depan.

1.4. Formulasi Masalah

Formulasi masalah dalam tugas ini akan mengikuti empat tahap From Problem to ML Solution. Kami akan mulai dengan mendefinisikan masalah yang ingin kami selesaikan, seperti prediksi harga kendaraan berdasarkan fitur-fitur tertentu. Selanjutnya, kami akan melakukan persiapan data dengan melakukan eksplorasi, pembersihan data, transformasi, dan splitting data menjadi training dan testing set. Kami akan menerapkan dua alat AutoML, yaitu TPOT dan NiaPy, untuk membangun model prediktif. Evaluasi terhadap model yang dihasilkan akan dilakukan menggunakan matrik evaluasi yang relevan, seperti akurasi dan confusion matrix, dengan memberikan analisis mendalam terhadap hasil evaluasi dan melakukan perbandingan performa kedua alat AutoML.

2. EKSPLORASI DAN PERSIAPAN DATA

2.1. Eksplorasi Data

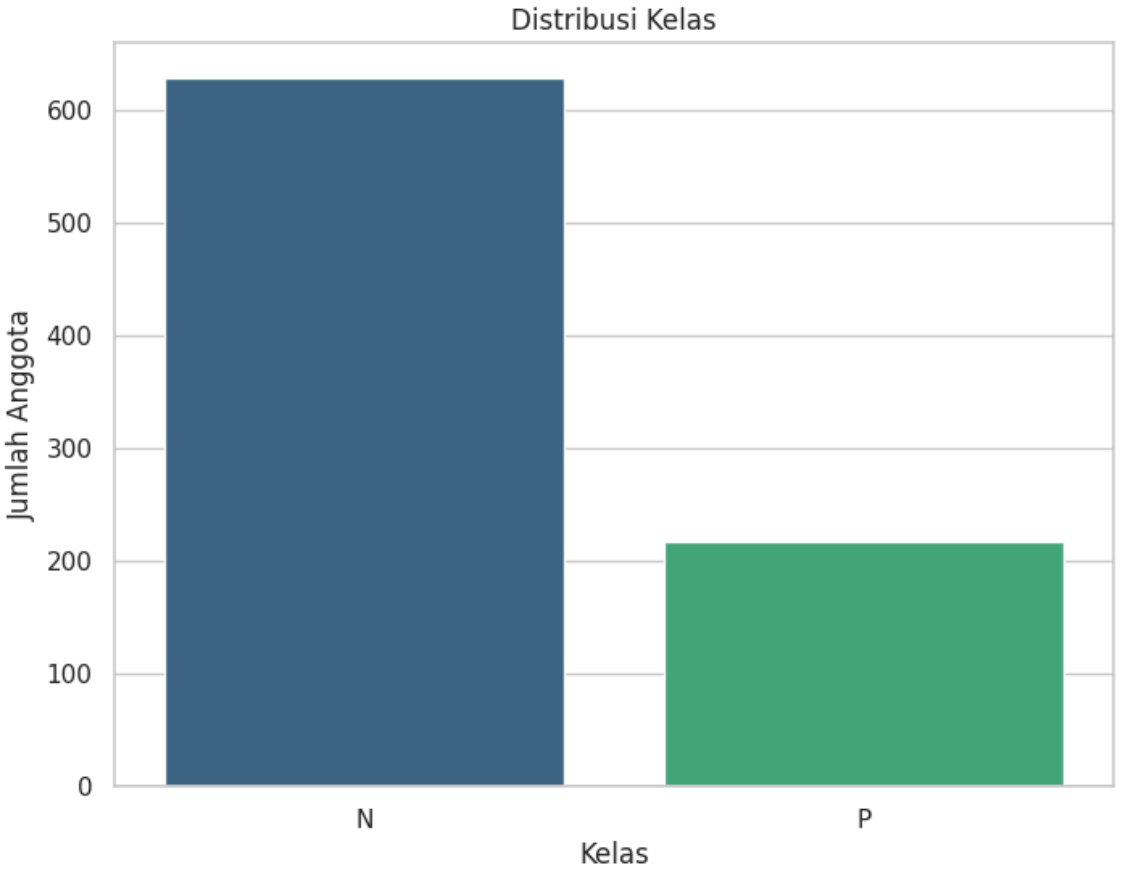
2.1.1. Statistik Deskriptif

	COMPACTNESS	CIRCULARITY	DISTANCE CIRCULARITY	RADIUS RATIO	PR.AXIS ASPECT RATIO	MAX.LENGTH ASPECT RATIO	SCATTER RATIO	ELONGATEDNESS	PR.AXIS RECTANGULARITY	MAX.LENGTH RECTANGULARITY	SCALED VARIANCE_MAJOR	SCALED VARIANCE_MINOR	SCALED RADIUS OF GYRATION	SKEWNESS ABOUT_MAJOR	SKEWNESS ABOUT_MINOR	KURTOSIS ABOUT_MAJOR	KURTOSIS ABOUT_MINOR	HOLLOWS RATIO
count	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.000000	846.00000	846.000000	846.000000	846.000000	846.000000	846.000000
mean	93.678487	44.861702	82.088652	168.940898	61.693853	8.567376	168.839243	40.933806	20.582742	147.998818	188.625296	439.911348	174.70331	72.462175	6.377069	12.599291	188.932624	195.632388
std	8.234474	6.169866	15.771533	33.472183	7.888251	4.601217	33.244978	7.811560	2.592138	14.515652	31.394837	176.692614	32.54649	7.486974	4.918353	8.931240	6.163949	7.438797
min	73.000000	33.000000	40.000000	104.000000	47.000000	2.000000	112.000000	26.000000	17.000000	118.000000	130.000000	184.000000	109.00000	59.000000	0.000000	0.000000	176.000000	181.000000
25%	87.000000	40.000000	70.000000	141.000000	57.000000	7.000000	146.250000	33.000000	19.000000	137.000000	167.000000	318.250000	149.00000	67.000000	2.000000	5.000000	184.000000	190.250000
50%	93.000000	44.000000	80.000000	167.000000	61.000000	8.000000	157.000000	43.000000	20.000000	146.000000	178.500000	364.000000	173.00000	71.500000	6.000000	11.000000	188.000000	197.000000
75%	100.000000	49.000000	98.000000	195.000000	65.000000	10.000000	198.000000	46.000000	23.000000	159.000000	217.000000	587.000000	198.00000	75.000000	9.000000	19.000000	193.000000	201.000000
max	119.000000	59.000000	112.000000	333.000000	138.000000	55.000000	265.000000	61.000000	29.000000	188.000000	320.000000	1018.000000	268.00000	135.000000	22.000000	41.000000	206.000000	211.000000

Gambar 1 Statistik deskriptif dataset

Gambar 1 menunjukkan statistik deskriptif dari dataset yang didapatkan dengan memanggil perintah “df.describe()”. Statistik berisikan count, mean, std, min, 25%, 50%, 75%, dan max dari masing-masing fitur. Count adalah jumlah elemen masing-masing fitur pada dataset. Mean adalah rata-rata nilai seluruh elemen pada setiap fitur di dataset. Std adalah standar deviasi seluruh elemen di masing-masing fitur. Min dan max adalah nilai minimum dan maksimum pada masing-masing fitur. Terakhir 25%, 50%, dan 75% masing-masing menunjukkan kuartil I, II, dan III dari masing-masing fitur. Statistik deskriptif sangat penting dilakukan terutama dalam melihat bagaimana fitur ini berbeda satu sama lain dan bagaimana perilaku nilai yang ada pada masing-masing fitur.

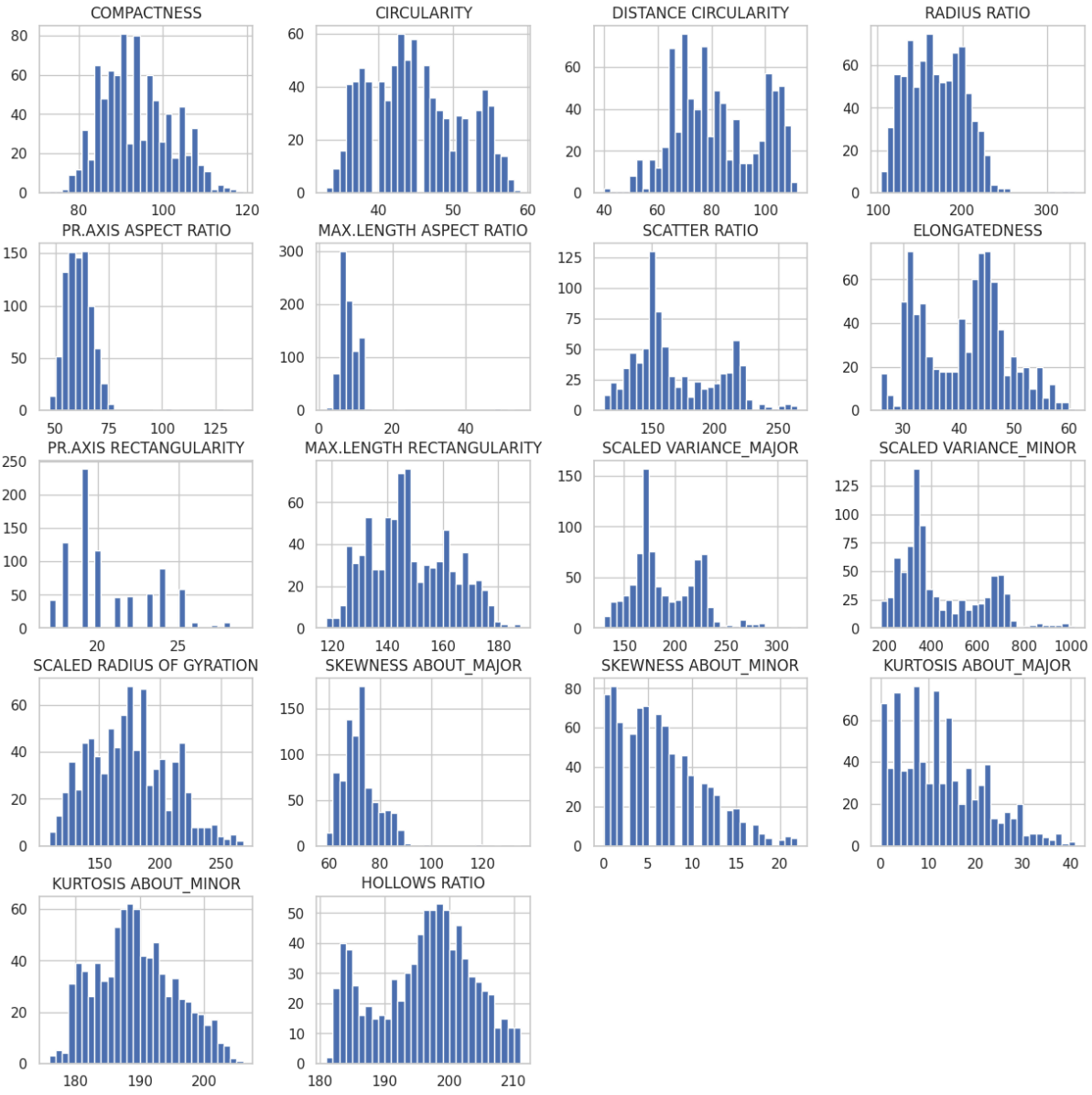
2.1.2. Visualisasi Data



Gambar 2. Distribusi Kelas

Gambar 2 menampilkan sebuah grafik batang yang menggambarkan distribusi dataset kendaraan dalam dua kelas biner, yaitu kelas "N" dan kelas "P". Sumbu horizontal (sumbu X) menunjukkan kategori kelas, sementara sumbu vertikal (sumbu Y) menunjukkan jumlah anggota di masing-masing kelas.

Pada grafik tersebut, terlihat bahwa jumlah kendaraan dalam kelas "N" adalah 628, yang diwakili oleh batang berwarna biru. Sebaliknya, jumlah kendaraan dalam kelas "P" adalah 218, diwakili oleh batang berwarna hijau. Distribusi ini menunjukkan bahwa kelas "N" memiliki lebih banyak kendaraan dibandingkan dengan kelas "P". Hal ini menunjukkan adanya perbedaan signifikan dalam distribusi jumlah kendaraan di kedua kelas tersebut. Judul grafik ini adalah "Distribusi Kelas", yang menjelaskan bahwa grafik tersebut menggambarkan perbandingan jumlah kendaraan dalam dua kelas yang berbeda dalam dataset kendaraan.

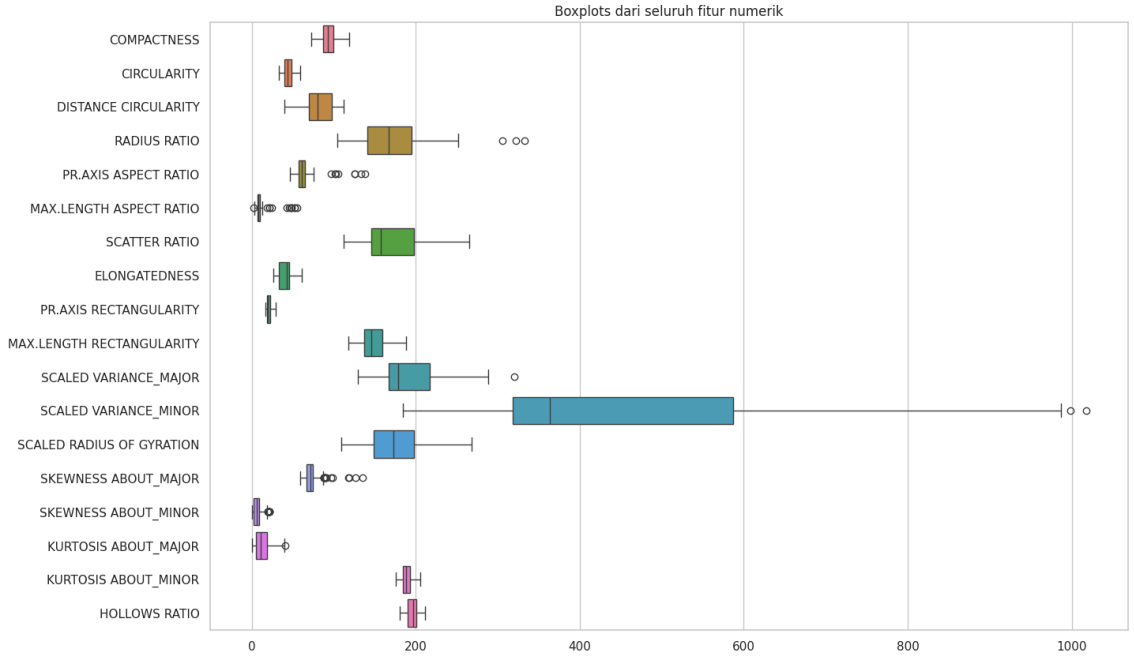


Gambar 3. Histogram Seluruh Fitur Numerik

Histogram-histogram diatas memberikan gambaran distribusi frekuensi fitur-fitur numerik dalam dataset kendaraan. Beberapa fitur, seperti COMPACTNESS dan CIRCULARITY, menunjukkan puncak frekuensi pada rentang nilai tertentu, sementara fitur lain memiliki distribusi yang lebih tersebar. Misalnya, fitur RADIUS RATIO memiliki distribusi dengan puncak di sekitar nilai 150-200. Di sisi lain, fitur-fitur seperti PR.AXIS ASPECT RATIO dan MAX.LENGTH ASPECT RATIO menampilkan distribusi yang lebih merata dengan beberapa puncak di berbagai interval.

Distribusi kompleks terlihat pada fitur SCALED VARIANCE_MAJOR dengan beberapa puncak di berbagai interval. Informasi ini penting untuk mengidentifikasi pola

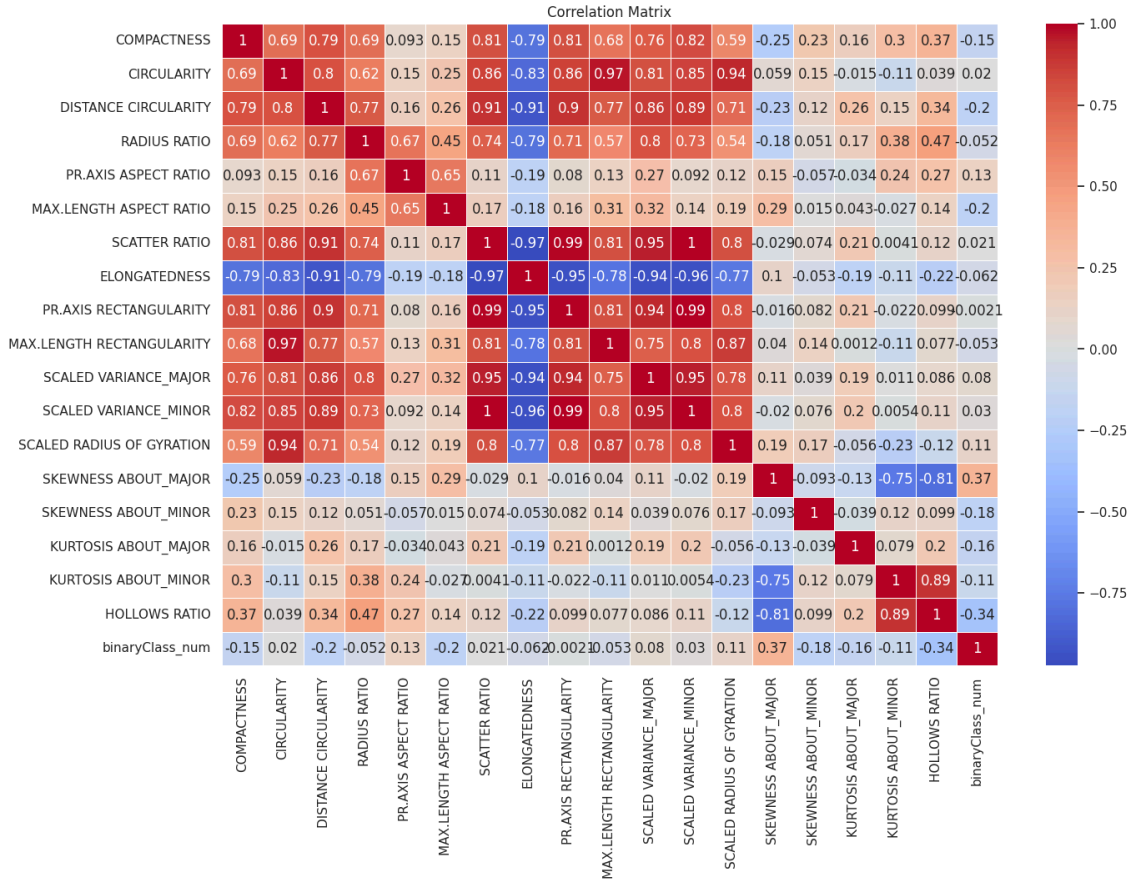
atau tren dalam data kendaraan dan mengevaluasi relevansi fitur-fitur ini dalam analisis lebih lanjut atau pemodelan.



Gambar 4. Boxplot Seluruh Fitur Numerik

Boxplots memberikan gambaran yang komprehensif tentang distribusi statistik dari berbagai fitur numerik dalam dataset kendaraan. Beberapa fitur menunjukkan distribusi yang relatif simetris seperti COMPACTNESS dan CIRCULARITY, sementara yang lain menunjukkan variasi yang signifikan dengan adanya outlier seperti RADIUS RATIO dan PR.AXIS ASPECT RATIO. Perbedaan dalam distribusi ini mencerminkan karakteristik unik dari setiap fitur numerik dan dapat memberikan wawasan yang berharga dalam analisis lanjutan atau pemodelan.

Dengan menggunakan boxplots, dapat diamati bahwa beberapa fitur memiliki rentang nilai yang luas dengan adanya outlier di kedua ujung, sementara yang lain memiliki distribusi yang lebih terkonsentrasi dengan outlier yang jarang. Informasi ini penting dalam mengidentifikasi pola atau tren dalam data kendaraan, serta dalam mengevaluasi apakah fitur-fitur tersebut relevan dalam memprediksi atribut tertentu dari kendaraan. Dengan demikian, analisis menggunakan boxplots dapat menjadi langkah awal yang penting dalam menggali lebih dalam dataset kendaraan ini.



Gambar 5. Correlation Matrix

Matriks korelasi hasil analisis dari dataset kendaraan menggambarkan hubungan antara berbagai fitur, dengan warna merah menunjukkan korelasi positif yang kuat dan warna biru menunjukkan korelasi negatif yang kuat. Temuan menarik meliputi korelasi positif yang sangat tinggi antara COMPACTNESS dengan MAX.LENGTH RECTANGULARITY, serta antara SCALED VARIANCE_MAJOR dengan SCALED RADIUS OF GYRATION. Di sisi lain, terdapat korelasi negatif yang kuat antara SKEWNESS ABOUT_MAJOR dengan SCALED VARIANCE_MAJOR, dan SKEWNESS ABOUT_MINOR dengan SCALED VARIANCE_MINOR. Temuan ini memberikan wawasan penting dalam memahami interaksi antar fitur dalam dataset, yang dapat digunakan untuk analisis lebih lanjut seperti pengurangan dimensi data atau fokus pada fitur-fitur yang memiliki korelasi rendah untuk mendapatkan informasi yang lebih unik.

Matriks korelasi juga memberikan petunjuk bagi strategi analisis lebih lanjut. Misalnya, pengurangan dimensi dengan menghilangkan fitur-fitur yang redundan berdasarkan korelasi tinggi, atau fokus pada fitur-fitur dengan korelasi rendah untuk mendapatkan wawasan yang lebih mendalam. Dengan demikian, matriks korelasi ini

bukan hanya alat visualisasi yang informatif, tetapi juga berguna dalam menentukan pendekatan analisis yang tepat untuk memahami struktur data pada dataset kendaraan.

2.2. Persiapan Data

2.2.1. Pembersihan Data

Outliers Removal

```
def remove_outliers(df, columns):
    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df

# Daftar fitur numerik untuk proses penghapusan outliers
numerical_features = df.columns[:-2]

# Menghapus outliers dari dataset
print(f"Shape of dataset before removing outliers: {df.shape}")
df_cleaned = remove_outliers(df, numerical_features)
print(f"Shape of dataset after removing outliers: {df_cleaned.shape}")

```

```
Shape of dataset before removing outliers: (846, 20)
Shape of dataset after removing outliers: (814, 20)
```

Kode tersebut mendefinisikan fungsi remove_outliers yang menghapus outliers dari kolom-kolom numerik dalam dataframe df berdasarkan Interquartile Range (IQR). Fungsi ini iterasi melalui setiap kolom yang diberikan, menghitung Q1 (kuartil pertama) dan Q3 (kuartil ketiga), menentukan IQR sebagai selisih antara Q3 dan Q1, lalu menetapkan batas bawah dan batas atas sebagai Q1 - 1.5*IQR dan Q3 + 1.5*IQR. Data yang berada di luar rentang ini dianggap sebagai outliers dan dihapus. Setelah mendefinisikan daftar fitur numerik (numerical_features), kode ini mencetak ukuran dataset sebelum dan sesudah penghapusan outliers untuk menunjukkan pengaruh dari proses tersebut.

2.2.2. Transformasi Data

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Transformasi data yang dilakukan adalah normalisasi menggunakan MinMaxScaler. Normalisasi ini dilakukan setelah data dipisah menjadi train dan test, tujuannya agar menghindari kebocoran informasi dari testing set ke training set. MinMaxScaler sendiri adalah adalah metode normalisasi yang mengubah nilai-nilai fitur ke dalam rentang [0, 1] atau ke rentang yang ditentukan oleh parameter feature_range. Proses ini dilakukan dengan menghitung nilai minimum dan maksimum dari fitur-fitur pada training set, kemudian menerapkan skala ini untuk mengubah nilai-nilai fitur pada training dan testing set.

Dengan melakukan normalisasi, fitur-fitur dalam dataset memiliki skala yang seragam, yang dapat meningkatkan performa algoritma machine learning tertentu yang sensitif terhadap skala data, seperti k-nearest neighbors atau gradient descent. Normalisasi juga membantu dalam menghindari dominasi fitur dengan skala nilai yang lebih besar terhadap fitur dengan skala nilai yang lebih kecil. Secara keseluruhan, penggunaan MinMaxScaler memastikan bahwa model yang dilatih memiliki performa yang konsisten dan dapat diandalkan saat diuji pada data yang belum pernah dilihat sebelumnya.

2.3. Pembagian Data

```
X = df.drop(columns=['binaryClass', 'binaryClass_num'])
y = df['binaryClass_num']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Kode tersebut melakukan pemisahan data menjadi fitur (X) dan label (y) dari dataset `df`. Kolom 'binaryClass' dan 'binaryClass_num' dihapus dari fitur, dan kolom 'binaryClass_num' digunakan sebagai label. Kemudian, data dibagi menjadi data training dan testing dengan menggunakan fungsi train_test_split, dengan 80% data digunakan untuk training dan 20% untuk testing. Parameter random_state=42 memastikan bahwa pemisahan data dilakukan secara konsisten setiap kali kode dijalankan.

3. PEMODELAN

3.1. Langkah-Langkah Detail Penggunaan TPOT

3.1.1. Import Library

```
from tpot import TPOTClassifier
```

Library yang dibutuhkan untuk penggunaan TPOT adalah TPOTClassifier. Library ini dibutuhkan untuk melakukan auto optimasi di dataset dengan tugas klasifikasi.

3.1.2. Penentuan parameter

Parameter default dari TPOT Classifier berdasarkan github.com/EpistasisLab/tpot adalah sebagai berikut.

```
def __init__(
    self,
    generations=100,
    population_size=100,
    offspring_size=None,
    mutation_rate=0.9,
    crossover_rate=0.1,
    scoring=None,
    cv=5,
    subsample=1.0,
    n_jobs=1,
    max_time_mins=None,
    max_eval_time_mins=5,
    random_state=None,
    config_dict=None,
    template=None,
    warm_start=False,
    memory=None,
    use_dask=False,
    periodic_checkpoint_folder=None,
    early_stop=None,
    verbosity=0,
    disable_update_check=False,
    log_file=None,
):
```

Pada project ini, parameter dari TPOT Classifier yang diubah adalah verbosity=2, generations=5, population_size=50, random_state=42.

- Verbosity digunakan untuk mengatur informasi apa saja yang akan ditampilkan di output ketika TPOT dijalankan. Angka 2 (dari skala 0 sampai 3) dipilih agar model menampilkan informasi yang detail namun tidak terlalu banyak yang dapat mendistraksi user.
- Generation adalah parameter yang berperan sebagai iterasi dari TPOT. Nilai default generation adalah 100, namun untuk mempersingkat waktu, generations hanya akan di set sebesar 50.
- Population size adalah jumlah individu yang ada pada tiap generasi. Nilai default dari population size adalah 100. Walaupun angka besar pada populasi

dapat membantu variansi model namun angka yang besar juga akan menambah komputasi, sehingga population size hanya akan di set 50.

- Random state adalah parameter yang berfungsi untuk mengatur seed untuk generator angka acak yang digunakan oleh TPOT. Ini memastikan hasil yang dapat direproduksi dengan membuat proses acak dalam optimasi (misalnya, pembuatan populasi awal, seleksi, dan mutasi) menjadi deterministik. Angka 42 dipilih karena merupakan pilihan populer di komunitas data science.

3.1.3. Training

```
tpot = TPOTClassifier(verbosity=2, generations=5, population_size=50, random_state=42)
start_time_tpot = time.time()
tpot.fit(X_train_scaled, y_train)
end_time_tpot = time.time()
```

Training dimulai dengan memanggil TPOTClassifier dengan parameter yang telah ditentukan sebelumnya, yaitu verbosity=2, generations=5, population_size=50, dan random_state=42. Kemudian, waktu pelatihan diukur dengan mencatat waktu mulai sebelum proses training dimulai menggunakan time.time(). Selanjutnya, model TPOT dilatih (fit) menggunakan data training yang telah dinormalisasi (X_train_scaled) dan label (y_train). Setelah proses training selesai selesai, waktu akhir training dicatat untuk menghitung durasi keseluruhan training.

3.2. Langkah-Langkah Detail Penggunaan NiaPy

3.2.1. Import Library

```
from niapy.task import Task
from niapy.problems import Problem
from niapy.algorithms.basic import ParticleSwarmOptimization
```

Library yang dibutuhkan untuk penggunaan NiaPy adalah Task, Problem, dan ParticleSwarmOptimization. Library ini dibutuhkan untuk melakukan optimasi hyperparameter menggunakan algoritma Particle Swarm Optimization. Task adalah salah satu komponen utama NiaPy untuk mendefinisikan setiap tugas. Lalu, Problem adalah komponen yang berguna untuk mendefinisikan masalah optimasi yang mendetail.

3.2.2. Penentuan parameter model

```
n_estimators = int(x[0])
max_depth = int(x[1])
min_samples_split = int(x[2])

model = make_pipeline(
    MinMaxScaler(),
    RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split, random_state=42)
)
```

Parameter dari Particle Swarm Optimization (PSO) digunakan untuk mengoptimalkan hyperparameter model RandomForestClassifier.

- Hyperparameter terbaik untuk n_estimators, max_depth, dan min_samples_split diekstrak dari hasil optimasi best dan dikonversi ke tipe integer.
- MinMaxScaler merupakan ormalisasi data dengan mengubah nilai fitur ke dalam rentang [0, 1]. Ini memastikan semua fitur memiliki skala yang sama.
- RandomForestClassifier dibangun dengan hyperparameter terbaik yang telah dioptimalkan. Parameter n_estimators, max_depth, dan min_samples_split diatur sesuai hasil optimasi, dan random_state=42 memastikan hasil yang dapat direproduksi.
- Fit melatih model pada data training yang telah dinormalisasi (X_train_scaled) dan label (y_train). Model RandomForestClassifier dilatih dengan menggunakan hyperparameter terbaik yang ditemukan oleh PSO, yang diharapkan meningkatkan kinerja model pada data yang tidak terlihat sebelumnya.

3.2.3. Training

```
class HyperparameterOptimization(Problem):
    def __init__(self):
        super().__init__(dimension=3, lower=[10, 1, 2], upper=[200, 50, 100], dtype=int)

    def _evaluate(self, x):
        n_estimators = int(x[0])
        max_depth = int(x[1])
        min_samples_split = int(x[2])

        model = make_pipeline(
            MinMaxScaler(),
            RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split, random_state=42)
        )

        scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='accuracy')

        return 1.0 - np.mean(scores)

start_time_niapy = time.time()
task = Task(problem=HyperparameterOptimization(), max_iters=50)
algo = ParticleSwarmOptimization()
best = algo.run(task)
end_time_niapy = time.time()
```

Training dimulai dengan mencatat waktu mulai menggunakan time.time() dan menyimpan hasilnya dalam variabel start_time_niapy. Kemudian, sebuah Task dibuat untuk mengoptimalkan hyperparameter dengan mendefinisikan masalah (HyperparameterOptimization()) dan menentukan jumlah iterasi maksimum (max_iters=50). Selanjutnya, algoritma Particle Swarm Optimization (PSO) diinisialisasi dengan memanggil ParticleSwarmOptimization(). Proses training dilakukan dengan menjalankan algoritma PSO pada tugas yang telah didefinisikan (task). Setelah training selesai, waktu akhir training dicatat menggunakan time.time() dan disimpan dalam variabel end_time_niapy untuk menghitung durasi keseluruhan training.

4. EVALUASI DAN PERBANDINGAN

4.1. Metode Evaluasi

Metriks evaluasi yang dipakai adalah Accuracy beserta Confusion Matrix dan juga waktu eksekusi.

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

Accuracy (akurasi) adalah metrik evaluasi yang mengukur persentase prediksi yang benar dari keseluruhan prediksi. Ini dihitung dengan membagi jumlah prediksi benar (baik positif maupun negatif) dengan total jumlah prediksi.

Confusion Matrix adalah metrik evaluasi yang menunjukkan jumlah prediksi benar dan salah untuk masing-masing kelas. Matriks ini terdiri dari empat komponen utama:

- True Positives (TP): Jumlah data dengan kelas positif yang diklasifikasikan dengan benar sebagai positif.
- True Negatives (TN): Jumlah data dengan kelas negatif yang diklasifikasikan dengan benar sebagai negatif.
- False Positives (FP): Jumlah data dengan kelas negatif yang salah diklasifikasikan sebagai positif..
- False Negatives (FN): Jumlah data dengan kelas positif yang salah diklasifikasikan sebagai negatif.

4.2. Hasil Evaluasi

4.2.1. Evaluasi Model dengan TPOT

```
y_pred_tpot = tpot.predict(X_test_scaled)
accuracy_tpot = accuracy_score(y_test, y_pred_tpot)
conf_matrix_tpot = confusion_matrix(y_test, y_pred_tpot)


print(f"TPOT Time: {end_time_tpot - start_time_tpot}")
print(f"TPOT Accuracy: {accuracy_tpot}")
print(f"TPOT Confusion Matrix:\n{conf_matrix_tpot}")
```

```
TPOT Time: 316.06198716163635
TPOT Accuracy: 1.0
TPOT Confusion Matrix:
[[118   0]
 [  0  52]]
```

4.2.2. Evaluasi Model dengan NiaPy

```
[ ]  y_pred_niapy = best_model.predict(X_test_scaled)
     accuracy_niapy = accuracy_score(y_test, y_pred_niapy)
     conf_matrix_niapy = confusion_matrix(y_test, y_pred_niapy)

     print(f"NiaPy Time: {end_time_niapy - start_time_niapy}")
     print(f"NiaPy Accuracy: {accuracy_niapy}")
     print(f"NiaPy Confusion Matrix:\n{conf_matrix_niapy}")
```

```
NiaPy Time: 1032.003620147705
NiaPy Accuracy: 0.9882352941176471
NiaPy Confusion Matrix:
[[118   0]
 [  2  50]]
```

4.3. Analisis Hasil Evaluasi

Hasil evaluasi menunjukkan bahwa TPOT dan NiaPy, dua alat AutoML yang dibandingkan, memiliki perbedaan yang signifikan dalam hal waktu eksekusi, akurasi, dan kinerja berdasarkan matriks kebingungan. TPOT menyelesaikan tugas dalam 316 detik, jauh lebih cepat dibandingkan NiaPy yang memerlukan 1032 detik. Dari segi akurasi, TPOT mencapai nilai sempurna 1.0, sementara NiaPy sedikit lebih rendah dengan 0.988. Matriks kebingungan TPOT menunjukkan kinerja sempurna tanpa kesalahan klasifikasi (true positives: 118, true negatives: 52, false positives: 0, false negatives: 0), sedangkan NiaPy memiliki dua kesalahan klasifikasi negatif sebagai positif (false negatives) meskipun tetap tidak memiliki false positives (true positives: 118, true negatives: 50, false positives: 0, false negatives: 2). Secara keseluruhan, TPOT lebih unggul baik dalam hal akurasi maupun efisiensi waktu, menjadikannya pilihan yang lebih baik untuk tugas ini, meskipun NiaPy tetap menunjukkan performa yang sangat baik dengan akurasi tinggi dan kesalahan minimal.

4.4. Perbandingan Langkah

TPOT dan NiaPy memiliki perbedaan pendekatan pada pemakaiannya. TPOT bekerja secara otomatis sedangkan NiaPy membutuhkan pengaturan yang lebih kompleks.

Pada TPOT kita hanya akan memanggil library classifier lalu menentukan parameter dan TPOT akan dengan sendirinya memilih model sekaligus melakukan hyperparameter tuning pada model. Langkah utamanya adalah:

- Pemanggilan Library: Memanggil library TPOT.
- Penentuan Parameter: Tentukan beberapa parameter dasar seperti generations, population_size, dan random_state.

- Otomatisasi Pemilihan Model dan Tuning: TPOT secara otomatis akan memilih model yang terbaik dan melakukan tuning hyperparameter.

Pada NiaPy, pemilihan model dan hyperparameter tuning yang dilakukan harus sendiri.

- Pemanggilan Library: Memanggil library NiaPy.
- Pemilihan Model: Tentukan model yang ingin digunakan secara manual.
- Pengaturan Tuning Hyperparameter: Lakukan tuning hyperparameter sendiri, sesuai dengan kebutuhan.

4.5. Perbandingan Hasil

Dari segi waktu eksekusi, TPOT lebih unggul dari NiaPy. Waktu eksekusi TPOT adalah sekitar 316.06 detik. Meskipun relatif cepat, ini mungkin karena TPOT secara otomatis mencari model dan tuning hyperparameter secara paralel. Sedangkan Waktu eksekusi NiaPy lebih lama, yaitu sekitar 1032.00 detik. Hal ini mungkin disebabkan oleh kompleksitas pengaturan manual dan iteratif yang dilakukan oleh pengguna dalam memilih model dan tuning hyperparameter.

Dari segi akurasi, TPOT juga lebih unggul dari NiaPy. TPOT mencapai akurasi sebesar 100%, yang menunjukkan bahwa model yang dihasilkan sangat baik dalam melakukan klasifikasi. Namun, akurasi sempurna ini juga mungkin menunjukkan adanya overfitting, terutama jika model diuji pada data yang belum pernah dilihat sebelumnya. Sedangkan NiaPy mencapai akurasi sebesar 98.82%, yang juga sangat tinggi dan mendekati akurasi TPOT. Meskipun sedikit lebih rendah dari TPOT, akurasi yang tinggi ini menunjukkan bahwa model yang dihasilkan oleh NiaPy juga efektif dalam melakukan klasifikasi.

Confusion matrix TPOT menunjukkan bahwa model tidak membuat kesalahan dalam klasifikasi. Semua sampel dari kedua kelas (positif dan negatif) diklasifikasikan dengan benar, dengan 118 true positives (TP) dan 52 true negatives (TN). Confusion matrix NiaPy juga menunjukkan performa yang sangat baik dalam klasifikasi. Namun, terdapat sedikit kesalahan di mana 2 sampel positif salah diklasifikasikan sebagai negatif (false negatives, FN). Ini bisa menandakan bahwa model dari NiaPy mungkin sedikit lebih konservatif dalam mengklasifikasikan sampel positif.

5. KESIMPULAN

5.1. Kesimpulan Proses yang Dijalankan

TPOT dan NiaPy adalah algoritma yang digunakan dalam tugas optimasi model. Terdapat perbedaan dalam pendekatan dan hasil evaluasi dari kedua algoritma ini. Pendekatan pada TPOT merupakan pendekatan otomatis yang meminimalkan campur tangan *programmer*, dengan mencari dan mengoptimalkan pipeline pembelajaran mesin secara otomatis. Sebaliknya, NiaPy memerlukan lebih banyak pengaturan manual yang mengharuskan *programmer* untuk secara langsung memilih model dan menyesuaikan hyperparameter.

Dalam evaluasi, TPOT menunjukkan keunggulan dalam waktu eksekusi yang lebih singkat, hanya sekitar 316 detik, dibandingkan dengan NiaPy yang memerlukan waktu sekitar 1032 detik. Selain itu, TPOT juga mencapai akurasi sempurna, yaitu 100%, menandakan kemampuan yang sangat baik dalam melakukan klasifikasi. Namun, NiaPy juga menghasilkan performa yang bagus, dengan akurasi sebesar 98.82%, sedikit lebih rendah dari TPOT. Meskipun NiaPy membutuhkan lebih banyak waktu dan pengaturan dari *programmer*, algoritma ini tetap memiliki performa yang bagus, terutama bagi *progammer* yang menginginkan fleksibilitas dan kontrol lebih besar dalam proses optimasi.

Oleh karena itu, dapat disimpulkan bahwa TPOT menawarkan solusi otomatis yang lebih cepat dan efisien, hasil yang didapatkan juga lebih cepat, dan sangat sedikit melibatkan *programmer* dalam pemilihan model maupun parameter. Di lain sisi, NiaPy juga tetap merupakan pilihan yang bagus untuk *programmer* yang lebih memilih fleksibilitas dan kontrol yang lebih besar dalam proses optimasi model. Dalam mengevaluasi dan memilih alat yang sesuai, penting untuk mempertimbangkan kebutuhan dan preferensi pengguna, serta faktor-faktor seperti efisiensi waktu, akurasi, dan tingkat kontrol yang diinginkan dalam proses optimasi.

5.2. Lesson Learned

Dari perbandingan antara TPOT dan NiaPy untuk optimasi model, ada beberapa pelajaran yang dapat dipetik:

- Peran Otomatisasi terhadap Waktu dan Akurasi

  TPOT menunjukkan bahwa otomatisasi dalam proses optimasi model dapat menghasilkan solusi yang efisien dan akurat dalam waktu yang relatif singkat. Hal ini mengurangi beban programmer dalam menentukan model dan tuning hyperparameter secara manual.

- Peran Pengaturan Manual terhadap Fleksibilitas

  Meskipun NiaPy membutuhkan lebih banyak pengaturan manual, pendekatan ini memberikan fleksibilitas yang lebih besar bagi programmer dalam menyesuaikan proses optimasi sesuai dengan kebutuhan dan preferensi mereka.

- Penyesuaian dengan Kebutuhan

  Penting untuk memilih alat yang paling sesuai dengan tujuan dan situasi spesifik. Apakah itu untuk mendapatkan solusi cepat atau untuk memiliki kontrol yang lebih besar atas proses optimasi, pemilihan alat harus selaras dengan kebutuhan dan preferensi programmer.

5.3. Saran untuk Improvement

Untuk meningkatkan pengalaman pengguna dan kinerja alat-alat seperti TPOT dan NiaPy, beberapa saran untuk perbaikan dapat dipertimbangkan:

- Pengaturan Parameter

  Mengatur parameter model optimasi dengan lebih variatif dan eksploratif dapat membantu dalam mendapatkan hasil yang lebih beragam dan mungkin lebih optimal. Hal ini termasuk menyesuaikan parameter seperti jumlah generasi, ukuran populasi, tingkat mutasi, dan parameter lainnya yang dapat mempengaruhi hasil akhir. Dengan mencoba berbagai kombinasi parameter, model optimasi dapat menemukan solusi yang lebih baik dan lebih sesuai dengan kebutuhan spesifik dataset dan masalah yang sedang dihadapi.

- Penambahan Fitur

  Menambahkan fitur-fitur yang dibutuhkan atau diinginkan dapat membuat alat lebih lengkap dan berguna. Ini bisa mencakup integrasi dengan alat-alat lain, kemampuan untuk menangani jenis data yang lebih beragam, atau fitur-fitur analisis tambahan.

- Penanganan Data yang Lebih Baik

  Fokus pada kemampuan alat untuk menangani data yang besar dan kompleks dengan lebih baik dapat menjadi poin peningkatan yang signifikan, mengingat pentingnya data dalam proses pembelajaran mesin.

LAMPIRAN

Link video presentasi : youtube

Link code : colab