

Dokumen Tugas Besar 2 IF3070
Implementasi Algoritma Pembelajaran Mesin



Disusun oleh:
Kelompok 06

Dzulfaqor Ali Dipangegara / 18222017

Billy Samuel Setiawan / 18222039

Dama Dhananjaya Daliman / 18222047

Steven Adrian Corne / 18222101

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

Daftar Isi

Daftar Isi.....	2
Daftar Gambar.....	3
Daftar Tabel.....	4
BAB I	
DESKRIPSI DATASET.....	5
BAB II	
IMPLEMENTASI ALGORITMA PEMBELAJARAN MESIN.....	6
2.1. Penjelasan KNN.....	6
2.2. Penjelasan Naive Bayes.....	7
BAB III	
PENJELASAN CLEANING DAN PREPROCESSING.....	8
3.1. Cleaning.....	8
3.1.1. Handling Missing Values.....	8
3.1.2. Handling Outliers.....	13
3.1.3. Removing Duplicates.....	15
3.1.4. Feature Engineering.....	16
3.2. Preprocessing.....	17
3.2.1. Scaling.....	17
3.2.2. Encoding.....	22
3.2.3. Handle Imbalance.....	24
BAB IV	
PERBANDINGAN DENGAN PUSTAKA.....	25
3.3. Perbandingan KNN.....	25
3.4. Perbandingan Naive Bayes.....	25
REFERENSI.....	27
LAMPIRAN.....	28
PEMBAGIAN TUGAS.....	29

Daftar Gambar

Gambar 3.1.1.1 Handling Missing Value Bagian 1.....	10
Gambar 3.1.1.2 Handling Missing Value Bagian 2.....	10
Gambar 3.1.1.3 Kode Multiple Imputation.....	13
Gambar 3.1.2.1 Boxplot Outlier.....	15
Gambar 3.1.2.1 Kode Handling Outlier.....	15
Gambar 3.1.3.1 Kode Removing Duplicates.....	16
Gambar 3.1.4.1 Kode Feature Engineering.....	16
Gambar 3.2.1.1 Kode Scaling Feature.....	17
Gambar 3.2.1.2 Boxplot Before Scaling.....	18
Gambar 3.2.1.3 Boxplot Min-Max Scaling (Train).....	19
Gambar 3.2.1.4 Boxplot Min-Max Scaling (Val).....	19
Gambar 3.2.1.5 Boxplot Standard Scaling (Train).....	20
Gambar 3.2.1.6 Boxplot Standard Scaling (Val).....	20
Gambar 3.2.1.7 Boxplot Robust Scaling (Train).....	21
Gambar 3.2.1.8 Boxplot Robust Scaling (Val).....	21
Gambar 3.2.2.1 Kode Encoding Train.....	22
Gambar 3.2.2.2 Kode Encoding Val.....	22
Gambar 3.2.2.1 Tipe Data Train Setelah Encoding.....	23
Gambar 3.2.2.2 Tipe Data Val Setelah Encoding.....	23
Gambar 3.2.3.1 Handle Imbalance.....	24
Gambar 3.3.1 Perbandingan KNN dengan Lib.....	25
Gambar 3.3.2 Perbandingan GNB dengan Lib.....	26

Daftar Tabel

Tabel 3.1.1.1 Kondisi Awal Missing Values.....	8
Tabel 3.1.1.2 Kondisi Missing Value Setelah Implementasi Kode.....	10
Tabel 3.1.1.3 Kondisi AKhir Missing Value Setelah Drop Fitur.....	12

BAB I

DESKRIPSI DATASET

[PhiUSIIL Phishing URL Dataset](#) adalah dataset yang terdiri dari deskripsi suatu URL dan juga fitur-fitur yang terkait dengan URL tersebut beserta label URL legitimate dan URL phishing. Label 1 merupakan label untuk URL legitimate, sementara label 0 merupakan label untuk URL phishing. Sebagian besar URL yang dianalisis saat membangun dataset ini adalah URL terbaru. Fitur-fitur diekstraksi dari source code halaman web dan URL. Dataset dapat diakses pada tautan [berikut](#).

BAB II

IMPLEMENTASI ALGORITMA PEMBELAJARAN MESIN

2.1. Penjelasan KNN

K-Nearest Neighbor (KNN) adalah algoritma pembelajaran mesin yang termasuk dalam kategori *supervised learning*. Dalam supervised learning, algoritma belajar dari data yang sudah memiliki label atau kelas yang diketahui. Tujuannya adalah agar algoritma dapat mempelajari pola atau hubungan antara fitur-fitur data dengan label atau kelas yang bersesuaian, sehingga dapat melakukan prediksi terhadap data baru yang belum pernah dilihat sebelumnya.

Aspek penting yang menjadikan KNN berbeda dengan algoritma *supervised learning* yang lain yaitu proses pembelajaran yang dilakukan. KNN tidak melakukan pembangunan model atau ekstraksi pola secara eksplisit dari data pelatihan. Sebaliknya, KNN hanya menyimpan seluruh data pelatihan beserta label atau kelas yang terkait dalam memori (*instance-based classifier*). Hal ini menjadikan KNN dikenal dengan *lazy learner*, karena proses pembelajaran baru dilakukan apabila ada data baru yang memerlukan klasifikasi.

Prinsip kerja algoritma KNN dalam melakukan klasifikasi pada data baru dapat dijelaskan sebagai berikut:

1. Menghitung jarak antara data baru dengan setiap data pada data pelatihan. Pengukuran jarak umumnya dilakukan menggunakan metode seperti Euclidean, Manhattan, atau Minkowski bergantung pada karakteristik data yang digunakan.
2. Menentukan sejumlah K data pelatihan yang memiliki jarak terdekat dengan data baru. Nilai K merupakan parameter yang ditentukan sebelum proses klasifikasi dilakukan dan mewakili jumlah tetangga terdekat yang akan dipertimbangkan dalam penentuan kelas.
3. Melakukan voting atau pemilihan kelas berdasarkan kelas mayoritas dari K tetangga terdekat. Kelas yang paling banyak muncul di antara K tetangga terdekat akan menjadi prediksi atau label untuk data baru tersebut.

Algoritma ini memiliki keunggulan berupa kesederhanaan implementasi dan fleksibilitas dalam menangani berbagai jenis data. Namun, kelemahannya meliputi kebutuhan komputasi tinggi saat memprediksi data baru karena seluruh data pelatihan perlu dipertimbangkan, serta sensitivitas terhadap fitur yang tidak relevan jika semua fitur diberi bobot yang sama. Dengan mempertimbangkan kelebihan dan kekurangannya, KNN cocok untuk masalah dengan jumlah fitur relevan yang terdefinisi baik dan data pelatihan yang tidak terlalu besar.

2.2. Penjelasan Naive Bayes

Naive Bayes adalah salah satu metode klasifikasi yang berbasis probabilitas dalam supervised learning. Algoritma ini bekerja dengan menghitung peluang suatu data masuk

ke dalam salah satu kelas target berdasarkan atribut-atributnya. Model ini mengasumsikan bahwa setiap atribut bersifat independen secara kondisional terhadap atribut lainnya (*naive assumption*), yang membuat perhitungannya menjadi sederhana dan efisien. Meskipun asumsi ini seringkali tidak sepenuhnya benar dalam dunia nyata, Naive Bayes tetap memberikan hasil yang baik dalam banyak aplikasi praktis.

Dalam fase pelatihan, Naive Bayes membangun sebuah model probabilitas berdasarkan data pelatihan yang diberikan. Proses pembelajaran ini melibatkan beberapa langkah:

1. Menghitung frekuensi kemunculan setiap nilai atribut untuk setiap kelas.
2. Menghitung frekuensi kemunculan setiap kelas dalam data observasi.
3. Menentukan probabilitas setiap nilai atribut dalam suatu kelas.
4. Menentukan probabilitas setiap kelas.

Probabilitas setiap nilai atribut dalam suatu kelas dihitung dengan membagi frekuensi kemunculan nilai atribut tersebut dalam kelas yang bersangkutan dengan frekuensi total kelas tersebut. Sedangkan probabilitas setiap kelas dihitung dengan membagi frekuensi kemunculan kelas dalam data observasi dengan jumlah total data observasi. Ketika ada data baru yang akan diklasifikasikan, Naive Bayes akan menghitung probabilitas setiap kelas dengan mengalikan probabilitas setiap nilai atribut dalam data baru untuk kelas tersebut. Lalu, menghitung probabilitas akhir dengan mengalikan hasil probabilitas tadi dengan probabilitas awal kelas tersebut. Terakhir, Naive Bayes memilih kelas dengan probabilitas tertinggi sebagai kelas prediksi untuk data baru.

Algoritma ini memiliki keunggulan dalam efisiensi komputasi dan performa yang baik pada data dengan atribut independen. Namun, kelemahannya adalah sensitivitas terhadap asumsi independensi, sehingga kurang cocok untuk data dengan atribut yang sangat saling berkorelasi. Naive Bayes banyak digunakan pada aplikasi seperti klasifikasi teks, deteksi spam, dan diagnosis medis.

BAB III

PENJELASAN CLEANING DAN PREPROCESSING

3.1. Cleaning

3.1.1. Handling Missing Values

Tahap pertama cleaning kami melakukan penanganan terhadap nilai-nilai kosong (*null*) dengan mengisi fitur-fitur dengan *domain knowledge* hasil dari membaca paper terkait yang dicantumkan di metadata. Didapatkan dari paper dan metadata bahwa beberapa fitur bisa diinferensi dari fitur lain. Salah satu contoh fitur yang bisa mengisi nilai dari fitur-fitur di dataset adalah fitur URL. Fitur-fitur seperti domain, TLD, LetterRatioInURL, HasObfuscation, dan masih banyak lagi, bisa diinferensi dari URL. Contohnya, domain bisa diinferensi dari URL dengan mengambil bagian setelah “https://” dan LetterRatioInURL bisa didapatkan dengan menghitung jumlah karakter alfabetis (A-Z, a-z) dibagi dengan panjangnya URL.

Pada awalnya jumlah nilai kosong di setiap fitur adalah sebagai berikut:

Tabel 3.1.1.1 Kondisi Awal Missing Values

Nama Fitur	Banyak Nilai Kosong
label	0
IsDomainIP	33735
IsResponsive	33993
NoOfEmptyRef	34151
Pay	34472
URL	34695
NoOfPopup	34702
HasHiddenFields	34921
NoOfSubDomain	35075
NoOfQMarkInURL	35307
HasTitle	35562
TLD	36307
NoOfAmpersandInURL	36320
DomainLength	37078
Robots	37504
NoOfOtherSpecialCharsInURL	38045
TLDLength	38157
CharContinuationRate	38358
NoOfSelfRef	38466

Nama Fitur	Banyak Nilai Kosong
IsHTTPS	39547
NoOfiFrame	39893
DomainTitleMatchScore	40037
Crypto	40184
NoOfImage	40264
URLCharProb	41653
URLTitleMatchScore	41815
TLDLegitimateProb	42296
DegitRatioInURL	42944
HasDescription	43654
Bank	43989
HasExternalFormSubmit	44467
Title	46694
HasFavicon	46735
NoOfDegitsInURL	47019
URLLength	48518
NoOfJS	48664
NoOfEqualsInURL	49215
HasSubmitButton	49236
SpacialCharRatioInURL	50203
NoOfLettersInURL	50672
ObfuscationRatio	51481
HasObfuscation	52499
LetterRatioInURL	52669
HasPasswordField	53109
NoOfSelfRedirect	53422
NoOfObfuscatedChar	53443
NoOfCSS	53646
HasCopyrightInfo	53879
NoOfURLRedirect	53925
HasSocialNet	54363
LargestLineLength	54374
LineOfCode	55320
NoOfExternalRef	55549
Domain	56122

Kemudian, dengan memanfaatkan *domain knowledge* yang didapatkan, maka kami mengimplementasikan kode untuk mengisi fitur-fitur dengan nilai kosong. Salah satunya adalah untuk fitur “domain”, kami melakukannya dengan kode berikut:

```
# Kita mulai dari mengisi domain, karena dia yang paling banyak kosong
def ambil_domain(url):
    if isinstance(url, str):
        return url.split('/')[1].split('/')[0] if '/' in url else url.split('/')[0]
    return None # Return None buat instance yg URL nya juga kosong

df_to_clean.loc[df_to_clean['Domain'].isnull(), 'Domain'] = df_to_clean.loc[df_to_clean['Domain'].isnull()
    'URL'].apply(lambda x: ambil_domain(x)
    if pd.notnull(x) else None)
# pake loc biar ga overwrite nilai yang asalnya ga null
```

Gambar 3.1.1.1 Handling Missing Value Bagian 1

Dari sini kami juga bisa mendapatkan “DomainLength”, dengan menghitung panjang dari “Domain”, implementasinya seperti ini:

```
#DomainLength
df_to_clean.loc[df_to_clean['DomainLength'].isnull(), 'DomainLength'] = df_to_clean.loc[df_to_clean['DomainLength'].isnull()
    'Domain'].apply(lambda x: len(x) if pd.notnull(x) else None)

df_to_clean['DomainLength'].isnull().sum()
```

Gambar 3.1.1.2 Handling Missing Value Bagian 2

Kolom-kolom lainnya juga diisi dengan cara serupa, menggunakan *domain knowledge* lengkapnya bisa dicek di notebook. Hingga hasil akhirnya setelah metode ini adalah seperti ini:

Tabel 3.1.1.2 Kondisi Missing Value Setelah Implementasi Kode

Nama Fitur	Banyaknya Nilai Kosong
HasObfuscation	0
label	0
TLDLength	3793
NoOfSubDomain	5417
DomainLength	5767
IsDomainIP	10400
NoOfQMarkInURL	10884
TLD	11192
NoOfAmpersandInURL	11314
TLDLegitimateProb	11440
CharContinuationRate	11795
IsHTTPS	12184

URLCharProb	12880
DegitRatioInURL	13286
NoOfDegitsInURL	14576
HasTitle	14630
URLLength	14973
NoOfEqualsInURL	15157
SpacialCharRatioInURL	15403
NoOfLettersInURL	15645
ObfuscationRatio	16016
LetterRatioInURL	16345
NoOfObfuscatedChar	16460
Domain	17344
DomainTitleMatchScore	23946
URLTitleMatchScore	24922
IsResponsive	33993
NoOfEmptyRef	34151
Pay	34472
URL	34695
NoOfPopup	34702
HasHiddenFields	34921
Robots	37504
NoOfOtherSpecialCharsInURL	38045
NoOfSelfRef	38466
NoOfiFrame	39893
Crypto	40184
NoOfImage	40264
HasDescription	43654
Bank	43989
HasExternalFormSubmit	44467
Title	46694
HasFavicon	46735
NoOfJS	48664
HasSubmitButton	49236
HasPasswordField	53109
NoOfSelfRedirect	53422
NoOfCSS	53646
HasCopyrightInfo	53879

NoOfURLRedirect	53925
HasSocialNet	54363
LargestLineLength	54374
LineOfCode	55320
NoOfExternalRef	55549

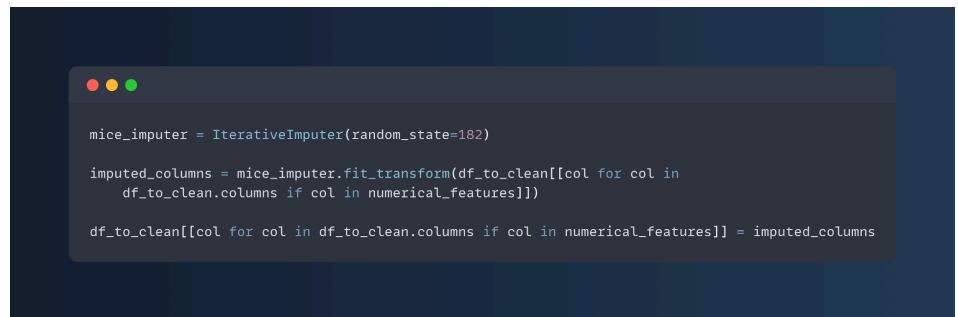
Terlihat fitur-fitur yang masih banyak mengandung nilai kosong adalah fitur-fitur yang tidak bisa diinferensi dari kolom lain seperti, “NoOfExternalRef”, “LineOfCode”, “LargestLineLength”, “HasSocialNet”, dan sebagainya. Fitur-fitur ini harus dicek dengan memeriksa HTML atau mengunjungi webpage pada URL tersebut. Maka dari itu, kami memutuskan untuk menghapus (*drop*) semua fitur yang nilai kosongnya di atas median banyaknya nilai kosong. Pada kasus ini, median banyaknya nilai kosong adalah 34.072, sehingga semua fitur dengan nilai kosong lebih banyak dari 34.072 akan dihapus. Setelah proses penghapusan ini, maka yang tersisa adalah fitur-fitur ini:

Tabel 3.1.1.3 Kondisi Akhir Missing Value Setelah Drop Fitur

Nama Fitur	Banyaknya Nilai Kosong
HasObfuscation	0
label	0
TLDLength	3793
NoOfSubDomain	5417
DomainLength	5767
IsDomainIP	10400
NoOfQMarkInURL	10884
TLD	11192
NoOfAmpersandInURL	11314
TLDLegitimateProb	11440
CharContinuationRate	11795
IsHTTPS	12184
URLCharProb	12880
DegitRatioInURL	13286
NoOfDegitsInURL	14576
HasTitle	14630
URLLength	14973
NoOfEqualsInURL	15157
SpacialCharRatioInURL	15403
NoOfLettersInURL	15645

ObfuscationRatio	16016
LetterRatioInURL	16345
NoOfObfuscatedChar	16460
Domain	17344
DomainTitleMatchScore	23946
URLTitleMatchScore	24922
IsResponsive	33993

Dari sisa fitur dengan nilai kosong yang tidak terlalu banyak (sekitar 20%), maka kami memutuskan untuk mengimplementasikan *multiple imputation* dengan memanfaatkan *IterativeImputer* dari pustaka *fancyimpute*. *IterativeImputer* hanya diterapkan pada fitur numerik, dengan kode seperti ini,



```

mice_imputer = IterativeImputer(random_state=182)

imputed_columns = mice_imputer.fit_transform(df_to_clean[[col for col in
df_to_clean.columns if col in numerical_features]])

df_to_clean[[col for col in df_to_clean.columns if col in numerical_features]] = imputed_columns

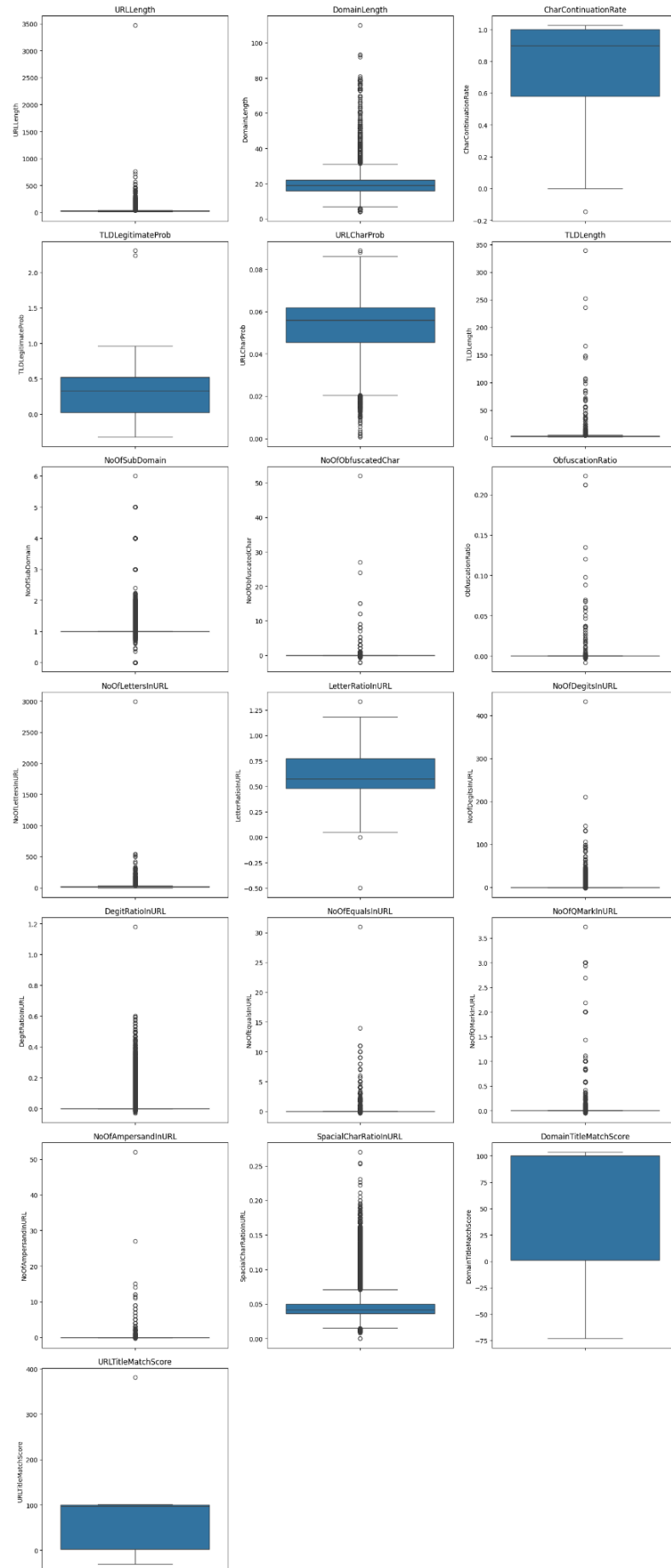
```

Gambar 3.1.1.3 Kode Multiple Imputation

Untuk fitur kategorikal, kami melakukan *mode imputing* dengan pengelompokkan. Kategori pengelompokkan pertama didasarkan pada fitur kategorikal yang sudah terisi penuh, yaitu “HasObfuscation”, diterapkan untuk pengelompokkan pada “IsDomainIP”, karena merupakan fitur kategorikal dengan banyak nilai kosong terendah setelah “HasObfuscation”. Kemudian setelah “IsDomainIP” terisi penuh, maka “IsDomainIP” digunakan untuk mengisi kolom kategorikal dengan banyak nilai kosong terendah selanjutnya “TLD”, dan begitu seterusnya untuk setiap fitur kategorikal yang kosong. Tetapi fitur kategorikal yang dijadikan dasar pengelompokkan hanya didasarkan pada 3 fitur kategorikal pertama: “HasObfuscation”, “IsDomainIP”, dan “TLD”. Setelah implementasi tahap ini, ternyata masih ada sangat sedikit nilai kosong. Nilai kosong ini langsung kami hapus instance-nya, karena paling banyak hanya 200 instance. Penghapusan instance yang mengandung nilai kosong ini adalah tahap terakhir dari *Handling Missing Values*.

3.1.2. Handling Outliers

Untuk bagian penanganan pencilan, di sini kami mencoba menganalisa melalui boxplot dengan hasil seperti ini,



Gambar 3.1.2.1 Boxplot Outlier

Terlihat sangat banyak pencilan jika dihitung berdasarkan IQR, tetapi setelah kami coba olah dengan Log Transform dan Sqrt Transform, ternyata distribusinya tetap tidak cukup “normal” atau “baik, bahkan menurut kami distribusi pencilan pada awalnya lebih tepat. Maka kami memutuskan lebih baik dihapus pencilannya berdasarkan nilai yang benar-benar pengecualian dan tidak masuk akal secara logis, contohnya menghapus nilai URLLength yang lebih dari 500 karena terlihat distribusinya mulai jarang-jarang (*sparse*) setelah melewati ambang batas tersebut. Lebih lanjut, bila dihapus nilai URLLength yang pengecualian, maka distribusi pencilan pada kolom lain yang terkait URL juga seharusnya menyesuaikan. Fitur-fitur yang menjadi tolak ukur modifikasi antara lain: “URLLength”, “TLDDLength”, “DomainLength”, “URLCharProb”, “ObfuscationRatio”, dan “NoOfAmpersandInURL”, dengan beberapa potongan kode yang kami terapkan untuk penghapusan adalah sebagai berikut,



```
# Drop rows with URLLength ≥ 500
nonnull_df = nonnull_df.loc[nonnull_df["URLLength"] < 500].reset_index(drop=True)

# Drop rows with TLDDLength ≥ 100
nonnull_df = nonnull_df.loc[nonnull_df["TLDDLength"] < 100].reset_index(drop=True)

# Drop rows with DomainLength ≥ 80
nonnull_df = nonnull_df.loc[nonnull_df["DomainLength"] < 80].reset_index(drop=True)

# Drop rows with URLCharProb ≥ 0.08
nonnull_df = nonnull_df.loc[nonnull_df["URLCharProb"] < 0.08].reset_index(drop=True)

# Drop rows with ObfuscationRatio ≥ 0.10
nonnull_df = nonnull_df.loc[nonnull_df["ObfuscationRatio"] < 0.10].reset_index(drop=True)

# Drop rows with NoOfAmpersandInURL ≥ 15
nonnull_df = nonnull_df.loc[nonnull_df["NoOfAmpersandInURL"] < 15].reset_index(drop=True)
```

Gambar 3.1.2.1 Kode Handling Outlier

3.1.3. Removing Duplicates

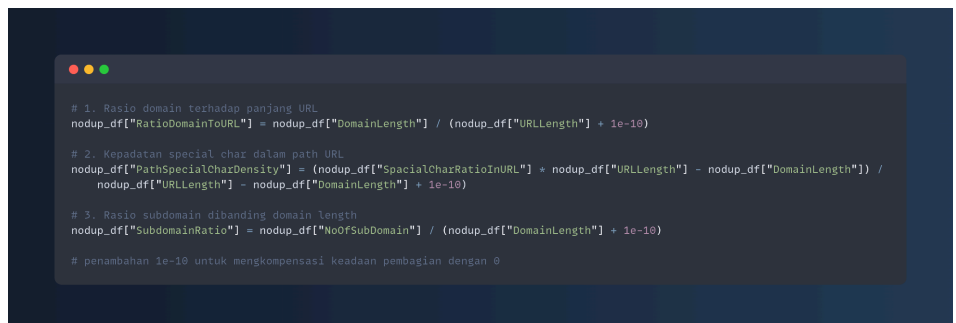
Setelah melakukan penanganan terhadap nilai kosong dan pencilan, didapatkan ada 397 data (*instance*) yang duplikat. Sehingga pada bagian penghapusan data (*instance*) yang duplikat, yang dilakukan hanya penghapusan dengan kode yang cukup jelas, seperti ini



Gambar 3.1.3.1 Kode Removing Duplicates

3.1.4. Feature Engineering

Untuk bagian rekayasa fitur, kami melakukan dua proses yaitu: pembuatan fitur-fitur baru dan seleksi fitur. Pertama akan dibahas pembuatan fitur baru, kami membuat 3 fitur baru yang masuk akal berdasarkan fitur-fitur yang tersisa setelah penghapusan di penanganan nilai kosong. 3 fitur baru yang kami buat adalah: “RatioDomainToURL”, “PathSpecialCharDensity”, dan “SubdomainRatio”.



Gambar 3.1.4.1 Kode Feature Engineering

Tahap kedua, seleksi fitur, kami melakukan seleksi dengan dua pendekatan, korelasi dan *Recursive Feature Elimination* (RFE). Proses seleksi berdasarkan korelasi adalah:

1. Mencari fitur-fitur yang memiliki kolinearitas tinggi
2. Melihat dari setiap pasangan dengan kolinearitas tinggi yang mana yang lebih tinggi korelasinya dengan label
3. Mengeliminasi fitur yang korelasinya dengan label lebih rendah

Sedangkan proses seleksi dengan RFE, secara garis besar algoritmanya adalah RFE akan mulai dengan semua fitur kemudian mengeliminasi fitur di setiap iterasi sampai fitur yang tersisah sejumlah n jumlah fitur yang diinginkan.

Dari kedua proses ini, kami mengambil fitur-fitur yang beririsan dari hasil kedua proses. Fitur-fitur yang beririsan akhirnya sejumlah 26 fitur (termasuk label).

3.2. Preprocessing

Data preprocessing adalah langkah yang lebih luas yang mencakup pembersihan data dan transformasi tambahan untuk membuat data siap untuk algoritma machine learning. Proses ini sangat penting karena data yang tidak terproses dengan baik dapat mengakibatkan model yang buruk.

3.2.1. Scaling

Pada tahap ini, dilakukan scaling pada fitur-fitur numerikal menggunakan metode seperti Min-Max Scaling, Standardization, dan Robust Scaling. Hal ini dilakukan untuk memastikan bahwa semua fitur memiliki skala yang sama atau mirip, yang sangat penting untuk algoritma yang sensitif terhadap skala, seperti KNN. Tanpa scaling, fitur dengan rentang nilai yang lebih besar dapat mendominasi perhitungan jarak, yang dapat mengarah pada hasil yang tidak akurat. Berikut potongan kodenya:

```
# Initialize scalers
min_max_scaler = MinMaxScaler()
standard_scaler = StandardScaler()
robust_scaler = RobustScaler()

# Apply scaling to the training set
X_train_scaled_minmax = min_max_scaler.fit_transform(clean_train[final_numerical_features])
X_train_scaled_standard = standard_scaler.fit_transform(clean_train[final_numerical_features])
X_train_scaled_robust = robust_scaler.fit_transform(clean_train[final_numerical_features])

# Apply scaling to the validation set
X_val_scaled_minmax = min_max_scaler.transform(clean_val_df[final_numerical_features])
X_val_scaled_standard = standard_scaler.transform(clean_val_df[final_numerical_features])
X_val_scaled_robust = robust_scaler.transform(clean_val_df[final_numerical_features])

# Convert scaled arrays back to DataFrame for better readability
X_train_scaled_minmax_df = pd.concat([pd.DataFrame(X_train_scaled_minmax, columns=final_numerical_features, index=clean_train.index),
                                      clean_train[final_categorical_features],
                                      clean_train[["label"]]], axis=1)
X_train_scaled_standard_df = pd.concat([pd.DataFrame(X_train_scaled_standard, columns=final_numerical_features, index=clean_train.index),
                                      clean_train[final_categorical_features],
                                      clean_train[["label"]]], axis=1)
X_train_scaled_robust_df = pd.concat([pd.DataFrame(X_train_scaled_robust, columns=final_numerical_features, index=clean_train.index),
                                      clean_train[final_categorical_features],
                                      clean_train[["label"]]], axis=1)

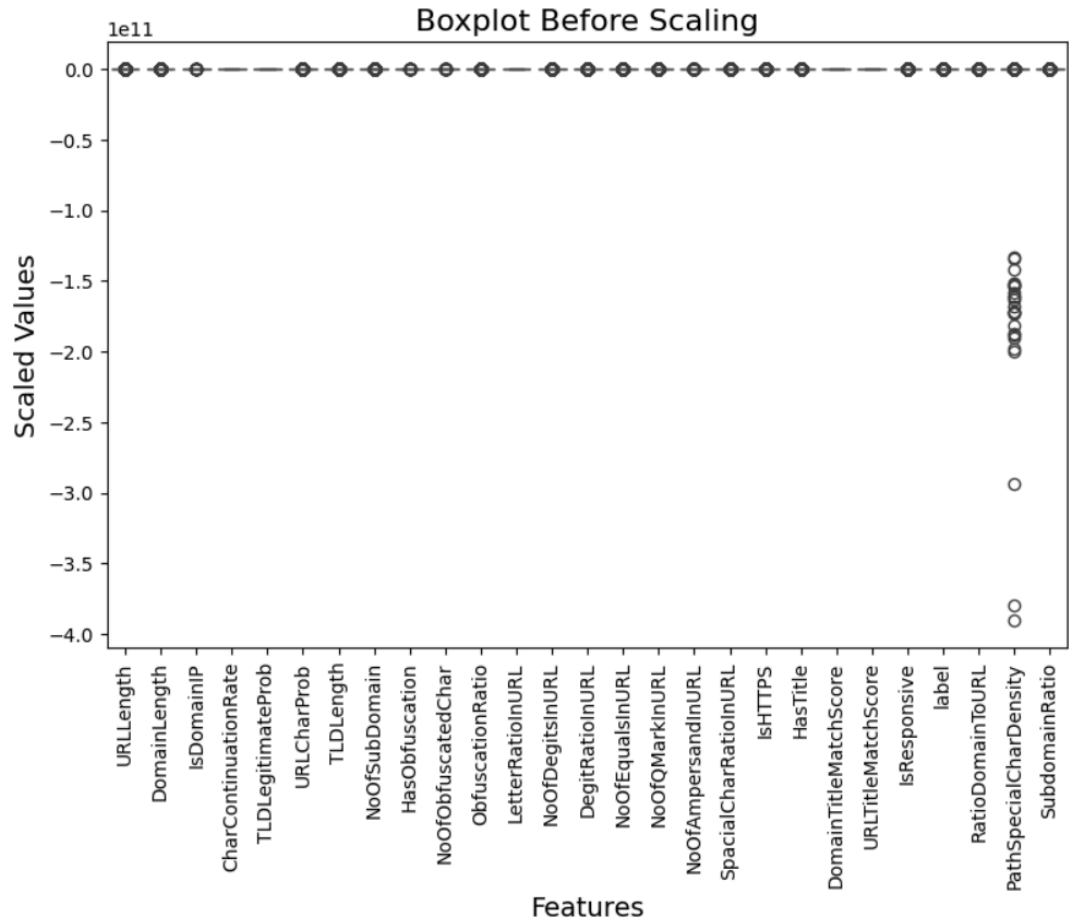
X_val_scaled_minmax_df = pd.concat([pd.DataFrame(X_val_scaled_minmax, columns=final_numerical_features, index=clean_val_df.index),
                                      clean_val_df[final_categorical_features],
                                      clean_val_df[["label"]]], axis=1)
X_val_scaled_standard_df = pd.concat([pd.DataFrame(X_val_scaled_standard, columns=final_numerical_features, index=clean_val_df.index),
                                      clean_val_df[final_categorical_features],
                                      clean_val_df[["label"]]], axis=1)
X_val_scaled_robust_df = pd.concat([pd.DataFrame(X_val_scaled_robust, columns=final_numerical_features, index=clean_val_df.index),
                                      clean_val_df[final_categorical_features],
                                      clean_val_df[["label"]]], axis=1)
```

Gambar 3.2.1.1 Kode Scaling Feature

Kode ini melakukan proses scaling pada data numerik dalam dataset pelatihan dan validasi untuk mempersiapkannya bagi analisis atau model pembelajaran mesin. Pertama, tiga metode Min-Max Scaling, Standard Scaling, dan Robust Scaling diinisialisasi untuk mengubah data. Data pelatihan yang telah dibersihkan kemudian diubah menggunakan ketiga metode ini, sementara data validasi hanya ditransformasi dengan pengaturan yang telah dipelajari dari data pelatihan. Hasil scaling dikonversi kembali ke dalam format DataFrame agar lebih

mudah dibaca, dengan menggabungkan data yang telah diubah dengan fitur kategorikal dan label. Terakhir, beberapa baris pertama dari data yang telah diubah ditampilkan untuk memverifikasi hasilnya, memastikan bahwa data siap digunakan dalam model.

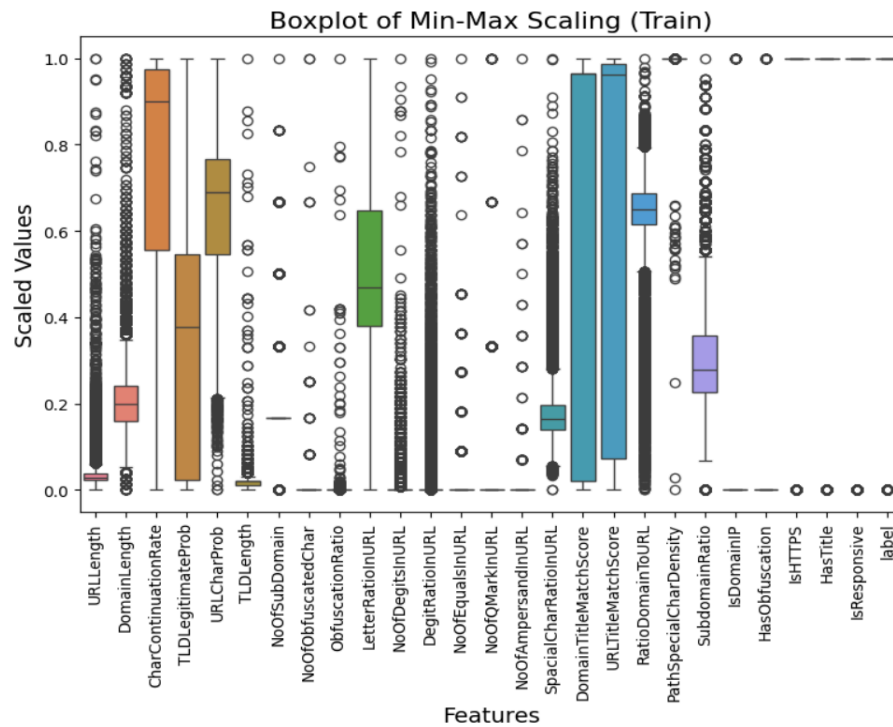
Berikut merupakan visualisasi sebelum Scaling:



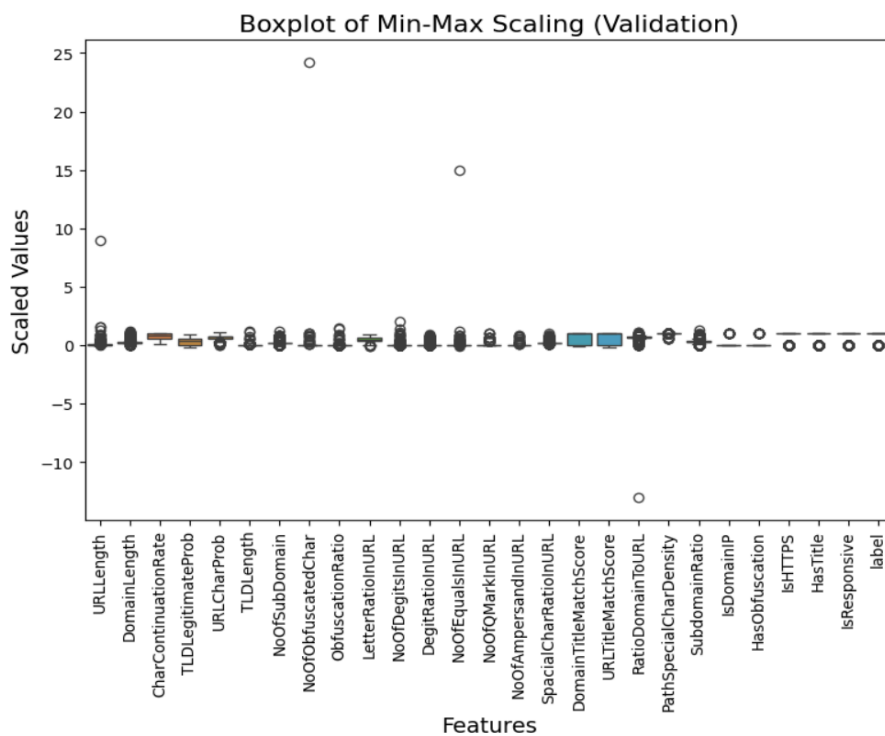
Gambar 3.2.1.2 Boxplot Before Scaling

Setelah dilakukan scaling pada data Train dan Val, hasilnya seperti berikut:

Min-Max Scaling:



Gambar 3.2.1.3 Boxplot Min-Max Scaling (Train)

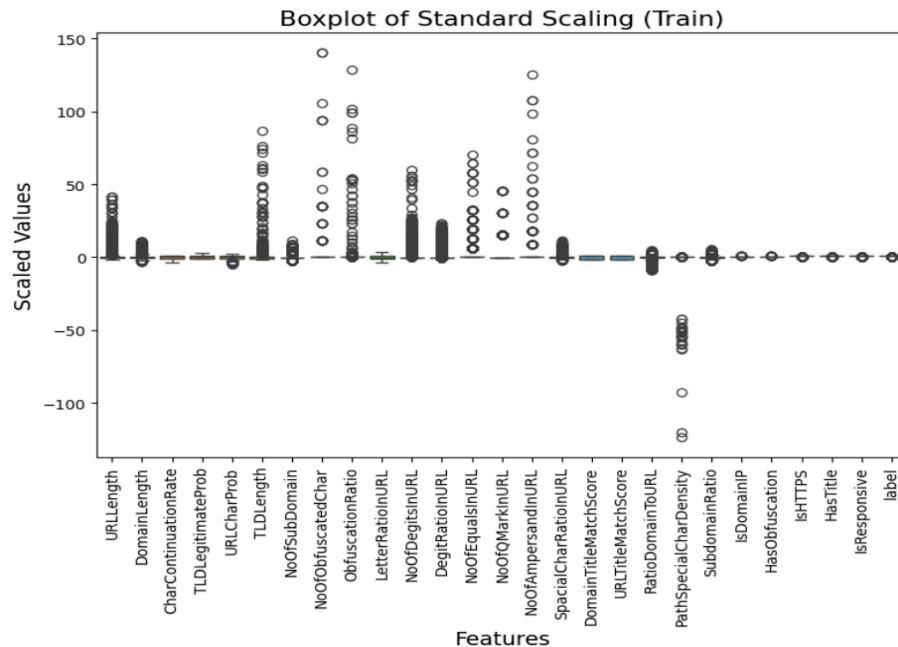


Gambar 3.2.1.4 Boxplot Min-Max Scaling (Val)

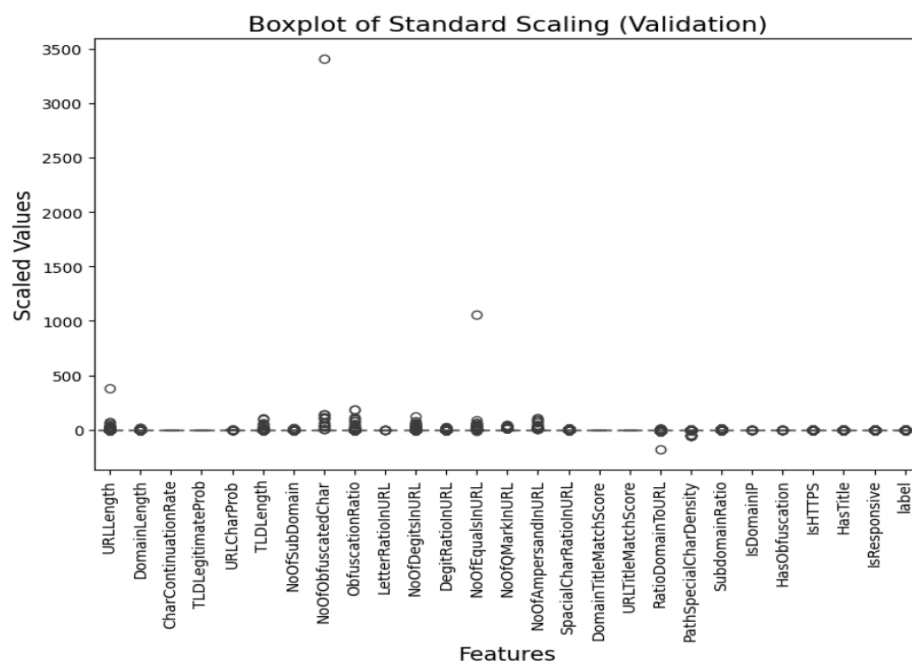
Pada Min-Max Scaling, data training menunjukkan rentang nilai terskala antara 0 dan 1, dengan beberapa fitur seperti DomainTitleMatchScore dan

RatioDomainToURL memiliki distribusi yang lebih merata, meski masih terdapat banyak outlier. Data validation untuk Min-Max Scaling menunjukkan distribusi yang lebih terkontrol namun masih memiliki beberapa outlier ekstrem yang mencapai nilai 25.

Standard Scaling:



Gambar 3.2.1.5 Boxplot Standard Scaling (Train)

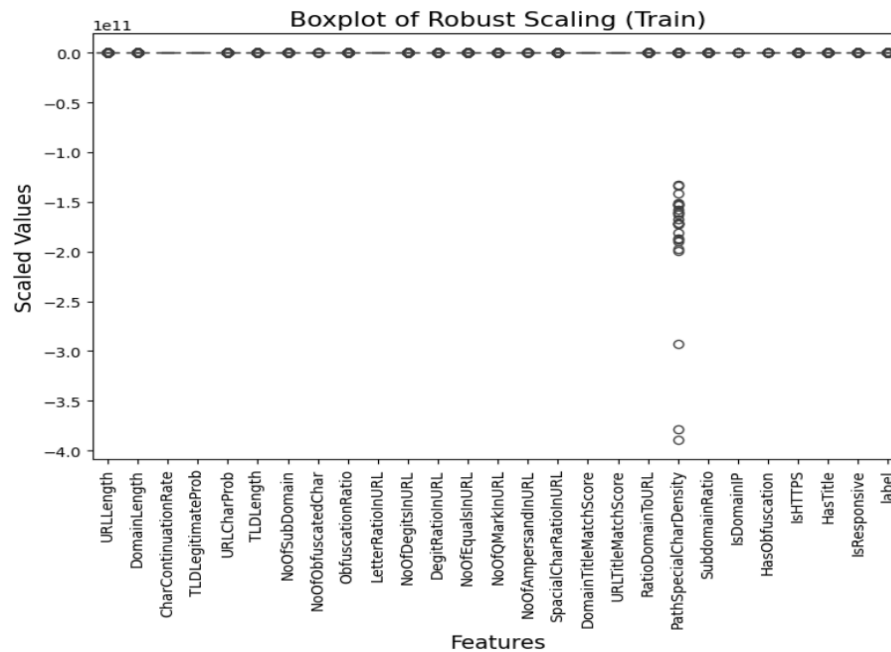


Gambar 3.2.1.6 Boxplot Standard Scaling (Val)

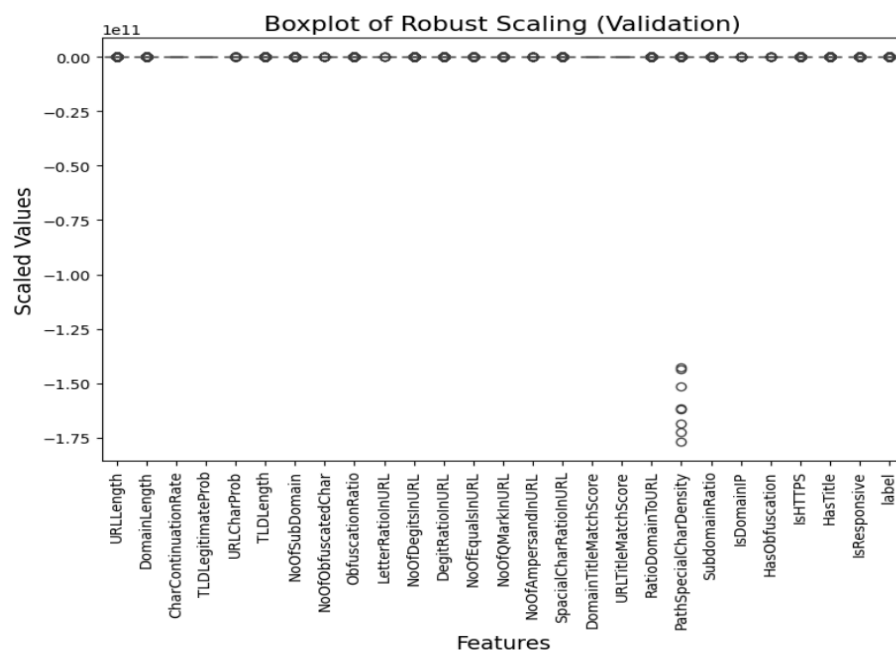
Standard Scaling menghasilkan distribusi dengan mean 0 dan standar deviasi 1 pada data training, namun menunjukkan outlier yang sangat ekstrem

mencapai nilai +150 dan -100, terutama pada fitur PathSpecialCharRatio. Pada data validation, skala nilai bahkan mencapai 3500 untuk beberapa outlier, menunjukkan perbedaan distribusi yang signifikan antara data training dan validation, meski mayoritas data tetap terkonsentrasi di sekitar nilai 0.

Robust Scaling:



Gambar 3.2.1.7 Boxplot Robust Scaling (Train)



Gambar 3.2.1.8 Boxplot Robust Scaling (Val)

Robust Scaling memberikan hasil yang paling konsisten antara data training dan validation, dengan distribusi yang sangat compact untuk mayoritas

fitur dan skala nilai dalam order 10^{11} . Metode ini menunjukkan outlier yang lebih terkontrol dibandingkan metode scaling lainnya, dengan hampir semua fitur memiliki median di sekitar 0.

Dari ketiga metode, Min-Max Scaling memberikan hasil yang mudah diinterpretasi namun sensitif terhadap outlier, Standard Scaling menunjukkan variasi yang lebih besar antara training dan validation, sementara Robust Scaling memberikan hasil paling stabil meski dengan skala nilai yang sangat besar. Pemilihan metode scaling sebaiknya disesuaikan dengan karakteristik data dan kebutuhan model yang akan digunakan.

3.2.2. Encoding

Pada tahap ini, dilakukan encoding pada fitur kategorikal pada data Train dan Val agar menjadi numerikal. Hal ini dilakukan untuk menghindari masalah yang muncul dari One-Hot Encoding yang dapat menghasilkan terlalu banyak kolom, yang dapat membuat model menjadi lebih kompleks dan sulit untuk dikelola. Kami melakukannya dengan kode berikut:

```
encoder = TargetEncoder()

# Fit and transform the training data
X_train_encoded = encoder.fit_transform(X_train_clean[final_categorical_features], y_train_clean)

# Convert the numpy array to DataFrame with proper column names
X_train_encoded_df = pd.DataFrame(
    X_train_encoded,
    columns=final_categorical_features,
    index=X_train_clean.index
)

# Combine the target-encoded categorical features with numerical features
X_train_combined = pd.concat([
    X_train_clean[final_numerical_features], # Keep original numerical features
    X_train_encoded_df # Add the target-encoded categorical features
], axis=1)
```

Gambar 3.2.2.1 Kode Encoding Train

```
X_val_encoded = encoder.transform(X_val_clean[final_categorical_features])

# Convert the numpy array to DataFrame with proper column names
X_val_encoded_df = pd.DataFrame(
    X_val_encoded,
    columns=final_categorical_features,
    index=X_val_clean.index
)

# Combine the target-encoded categorical features with numerical features
X_val_combined = pd.concat([
    X_val_clean[final_numerical_features], # Keep original numerical features
    X_val_encoded_df # Add the target-encoded categorical features
], axis=1)
```

Gambar 3.2.2.2 Kode Encoding Val

Kode ini melakukan proses target encoding pada fitur kategorikal dalam dataset pelatihan untuk mempersiapkannya bagi model pembelajaran mesin.

Pertama, sebuah objek TargetEncoder diinisialisasi untuk menggantikan kategori dalam fitur dengan rata-rata target (label) yang sesuai. Kemudian, data pelatihan yang telah dibersihkan diubah menggunakan encoder ini, di mana fitur kategorikal diubah menjadi representasi numerik berdasarkan nilai target. Hasil encoding, yang berupa array NumPy, kemudian dikonversi menjadi DataFrame dengan nama kolom yang sesuai dan indeks yang sama dengan data asli. Terakhir, DataFrame yang berisi fitur numerikal asli dan fitur kategorikal yang telah di-encode digabungkan menjadi satu DataFrame baru, sehingga semua fitur siap digunakan dalam analisis atau pelatihan model.

Hasil data setelah dilakukan encoding seperti berikut:

```
<class 'pandas.core.frame.DataFrame'>
Index: 100469 entries, 0 to 100864
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                -
0   URLLength                            100469 non-null  int64
1   DomainLength                        100469 non-null  int64
2   CharContinuationRate                100469 non-null  float64
3   TLDLegitimateProb                   100469 non-null  float64
4   URLCharProb                         100469 non-null  float64
5   TLDLength                           100469 non-null  int64
6   NoOfSubDomain                       100469 non-null  int64
7   NoOfObfuscatedChar                  100469 non-null  int64
8   ObfuscationRatio                    100469 non-null  float64
9   LetterRatioInURL                    100469 non-null  float64
10  NoOfDigitsInURL                     100469 non-null  int64
11  DigitRatioInURL                     100469 non-null  float64
12  NoOfEqualsInURL                     100469 non-null  int64
13  NoOfQMarkInURL                      100469 non-null  int64
14  NoOfAmpersandInURL                  100469 non-null  int64
15  SpacialCharRatioInURL                100469 non-null  float64
16  DomainTitleMatchScore                100469 non-null  float64
17  URLTitleMatchScore                   100469 non-null  float64
18  RatioDomainToURL                     100469 non-null  float64
19  PathSpecialCharDensity                100469 non-null  float64
20  SubdomainRatio                       100469 non-null  float64
21  IsDomainIP                           100469 non-null  float64
22  TLD                                   100469 non-null  float64
23  HasObfuscation                       100469 non-null  float64
24  IsHTTPS                              100469 non-null  float64
25  HasTitle                             100469 non-null  float64
26  IsResponsive                         100469 non-null  float64
dtypes: float64(18), int64(9)
```

Gambar 3.2.2.1 Tipe Data Train Setelah Encoding

```
Index: 28081 entries, 110142 to 126907
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                -
0   URLLength                            28081 non-null  int64
1   DomainLength                        28081 non-null  int64
2   CharContinuationRate                28081 non-null  float64
3   TLDLegitimateProb                   28081 non-null  float64
4   URLCharProb                         28081 non-null  float64
5   TLDLength                           28081 non-null  int64
6   NoOfSubDomain                       28081 non-null  int64
7   NoOfObfuscatedChar                  28081 non-null  int64
8   ObfuscationRatio                    28081 non-null  float64
9   LetterRatioInURL                    28081 non-null  float64
10  NoOfDigitsInURL                     28081 non-null  int64
11  DigitRatioInURL                     28081 non-null  float64
12  NoOfEqualsInURL                     28081 non-null  int64
13  NoOfQMarkInURL                      28081 non-null  int64
14  NoOfAmpersandInURL                  28081 non-null  int64
15  SpacialCharRatioInURL                28081 non-null  float64
16  DomainTitleMatchScore                28081 non-null  float64
17  URLTitleMatchScore                   28081 non-null  float64
18  RatioDomainToURL                     28081 non-null  float64
19  PathSpecialCharDensity                28081 non-null  float64
20  SubdomainRatio                       28081 non-null  float64
21  IsDomainIP                           28081 non-null  float64
22  TLD                                   28081 non-null  float64
23  HasObfuscation                       28081 non-null  float64
24  IsHTTPS                              28081 non-null  float64
25  HasTitle                             28081 non-null  float64
26  IsResponsive                         28081 non-null  float64
dtypes: float64(18), int64(9)
```

Gambar 3.2.2.2 Tipe Data Val Setelah Encoding

Dapat dilihat dari hasilnya, tipe data semua fitur telah menjadi numerik (int64 dan float64). Hal ini menandakan encoding berhasil.

3.2.3. Handle Imbalance

Dataset yang tidak seimbang dapat menyebabkan model bias terhadap kelas mayoritas. Dalam file ini, meskipun labelnya sangat tidak seimbang, kami memutuskan untuk tidak melakukan sampling. Sebagai gantinya, fokus diberikan pada penyesuaian bobot dalam algoritma prediksi untuk menangani ketidakseimbangan ini.

```
y_train_clean.value_counts()

label
1    93221
0     7248
Name: count, dtype: int64
```

Gambar 3.2.3.1 Handle Imbalance

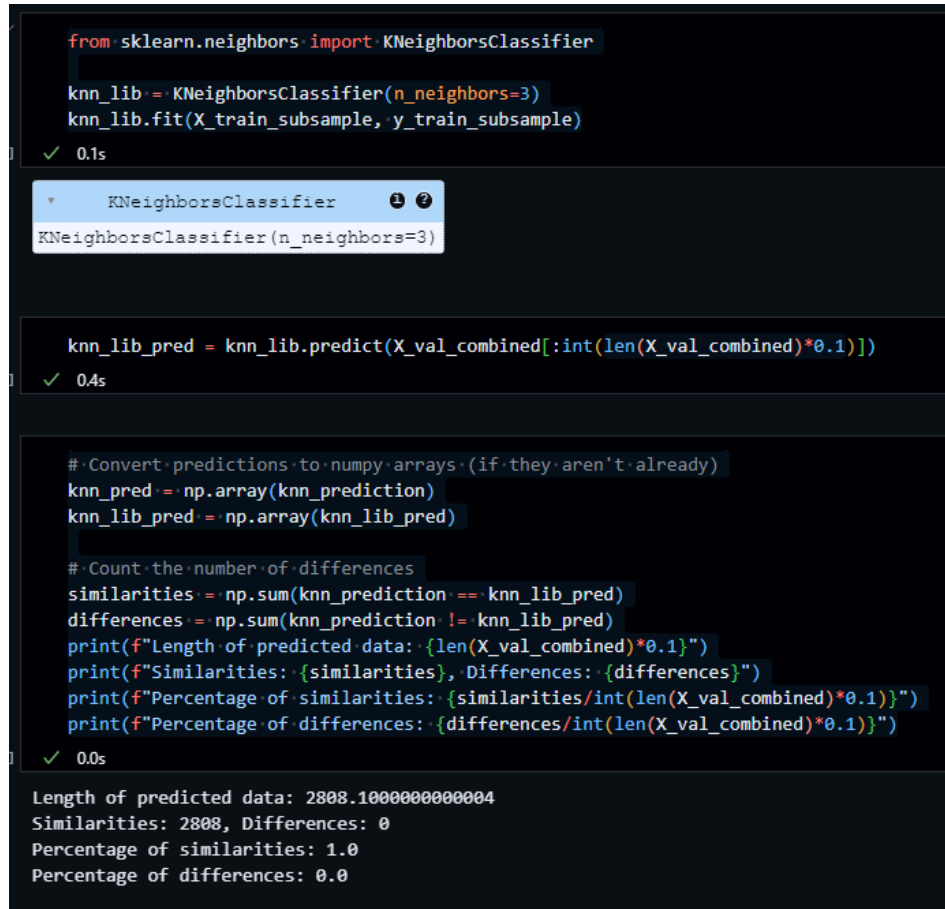
Meskipun tidak ada perubahan langsung pada dataset, pemahaman tentang distribusi kelas membantu dalam merancang model yang lebih baik. Dengan tidak melakukan sampling, model tetap dapat belajar dari data yang ada tanpa kehilangan informasi penting.

BAB IV

PERBANDINGAN DENGAN PUSTAKA

3.3. Perbandingan KNN

Pada train set dan validation set, KNN yang kami implementasikan sendiri (*from scratch*) memiliki hasil prediksi yang persis sama dengan KNN dari pustaka sklearn. Hal ini menunjukkan bahwa algoritma KNN yang kami implementasikan sudah optimal. Bisa dilihat dari hasil di tangkapan layar di bawah ini,



```
from sklearn.neighbors import KNeighborsClassifier

knn_lib = KNeighborsClassifier(n_neighbors=3)
knn_lib.fit(X_train_subsample, y_train_subsample)
✓ 0.1s

KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)

knn_lib_pred = knn_lib.predict(X_val_combined[:int(len(X_val_combined)*0.1)])
✓ 0.4s

# Convert predictions to numpy arrays (if they aren't already)
knn_pred = np.array(knn_prediction)
knn_lib_pred = np.array(knn_lib_pred)

# Count the number of differences
similarities = np.sum(knn_prediction == knn_lib_pred)
differences = np.sum(knn_prediction != knn_lib_pred)
print(f"Length of predicted data: {len(X_val_combined)*0.1}")
print(f"Similarities: {similarities}, Differences: {differences}")
print(f"Percentage of similarities: {similarities/int(len(X_val_combined)*0.1)}")
print(f"Percentage of differences: {differences/int(len(X_val_combined)*0.1)}")
✓ 0.0s

Length of predicted data: 2808.1000000000004
Similarities: 2808, Differences: 0
Percentage of similarities: 1.0
Percentage of differences: 0.0
```

Gambar 3.3.1 Perbandingan KNN dengan Lib

3.4. Perbandingan Naive Bayes

Pada train set dan validation set, Gaussian NB yang kami implementasikan prediksinya 92.16% sama dengan yang dari pustaka sklearn, dengan 2199 prediksi yang berbeda. Algoritma Naive Bayes yang kami implementasikan sudah baik, namun masih dapat dilakukan *minor improvement*. Bisa dilihat dari hasil di tangkapan layar di bawah ini,

```
from sklearn.naive_bayes import GaussianNB

gnb_lib = GaussianNB()

gnb_lib.fit(X_train_combined, y_train_clean)
✓ 0.1s

GaussianNB()

gnb_lib_pred = gnb_lib.predict(X_val_combined)
✓ 0.0s

# Convert predictions to numpy arrays (if they aren't already)
gnb_pred = np.array(gnb_prediction)
gnb_lib_pred = np.array(gnb_lib_pred)

# Count the number of differences
similarities = np.sum(gnb_prediction == gnb_lib_pred)
differences = np.sum(gnb_prediction != gnb_lib_pred)
print(f"Length of predicted data: {len(X_val_combined)}")
print(f"Similarities: {similarities}, Differences: {differences}")
print(f"Percentage of similarities: {similarities/int(len(X_val_combined))}")
print(f"Percentage of differences: {differences/int(len(X_val_combined))}")
✓ 0.0s

Length of predicted data: 28081
Similarities: 25882, Differences: 2199
Percentage of similarities: 0.9216908229763897
Percentage of differences: 0.07830917702361027
```

Gambar 3.3.2 Perbandingan GNB dengan Lib

REFERENSI

Prasad, A., & Chandra, S. (2024). PhiUSIIL Phishing URL (Website) [Dataset]. UCI Machine Learning Repository. <https://doi.org/10.1016/j.cose.2023.103545>

ScienceDirect. (2023). Article on phishing URL dataset.
<https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub>

Scikit-learn. (2023). Neighbors module.
<https://scikit-learn.org/1.5/modules/neighbors.html>

Scikit-learn. (2023). Naive Bayes module.
https://scikit-learn.org/1.5/modules/naive_bayes.html

LAMPIRAN

Link github : <https://github.com/dzulfaqorali196/Tubes-2-DAI>

PEMBAGIAN TUGAS

Nama	NIM	Tugas
Dzulfaqor Ali D.	18222017	Notebook membuat Model KNN yang from the scratch
		Notebook Save and Load KNN model
		Membuat repo github, foldering, dan membuat readme
Billy Samuel S.	18222039	Notebook bagian Data Preprocessing
		Formatting laporan
		Laporan bagian implementasi algoritma
Dama D. Daliman	18222047	Notebook bagian data cleaning & error analysis
		Laporan bagian Data Cleaning, Perbandingan
		Notebook Split Training Set and Validation Set
		Notebook Modeling and Validation
Steven Adrian Corne	18222101	Notebook bagian Data Preprocessing
		Notebook bagian Save and Load Gaussian Naive Bayes Model
		Laporan bagian Data Preprocessing dan formating