IF3070 Dasar Inteligensi Artifisial

Tugas Besar 2

Disusun oleh:

Jihan Aurelia	/ 18222001
Nasywaa Anggun Athiefah	/ 18222021
Ricky Wijaya	/ 18222043
Timotius Vivaldi Gunawan	/ 18222091

Dosen Pengampu :

Ir. Windy Gambetta, M.B.A.

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

DAFTAR ISI

BAB I
Pendahuluan

A. Pendahuluan

Seiring berkembangnya teknologi digital dan semakin meluasnya penggunaan internet, ancaman siber seperti phishing menjadi semakin signifikan. Phishing merupakan salah satu metode serangan siber yang bertujuan untuk menipu pengguna agar memberikan informasi sensitif, seperti kredensial akun atau data keuangan, melalui URL palsu yang menyerupai situs resmi. Ancaman tersebut terus meningkat dengan berkembangnya teknik-teknik baru yang semakin canggih dan sulit dideteksi.

Mendeteksi URL phishing adalah tantangan yang kompleks karena sifat serangan yang terus berkembang. Penyerang menggunakan berbagai teknik seperti mengganti karakter dalam URL dengan simbol yang serupa, menambahkan subdomain yang menyerupai nama domain asli, atau menggunakan domain tingkat atas (*top-level domain/TLD*) yang jarang dikenal. Teknik-teknik tersebut membuat serangan sulit dikenali, bahkan oleh pengguna yang berpengalaman.

Selain itu, terdapat tantangan dalam memproses dataset besar seperti PhiUSIIL. Dataset yang berisi cukup banyak URL membutuhkan pemrosesan yang efisien agar analisis dapat dilakukan secara tepat waktu.

Tugas berikut bertujuan untuk menerapkan algoritma **K-Nearest Neighbors (KNN)** dan **Gaussian Naive Bayes** untuk mendeteksi phishing URL. Tugas berikut menggunakan dataset **PhiUSIL Phishing URL Dataset** yang telah diberikan yang menyediakan informasi tentang karakteristik URL phishing dan legitimate.

Tugas berikut mencakup :

1. Membuat implementasi algoritma KNN dan Gaussian Naive Bayes dari awal (*from scratch*) untuk menganalisis dataset

2. Membandingkan hasil dari algoritma yang diimplementasikan secara manual dengan hasil *library* Scikit-Learn
3. Menyimpan dan memuat model untuk mendukung proses prediksi selanjutnya
4. Memahami perbedaan performa algoritma dalam mendeteksi URL phishing dan mendapatkan hasil analisisnya.

BAB II
Deskripsi Implementasi

A. K-Nearest Neighbor

1. Dasar Teori

K-Nearest Neighbor (KNN) adalah salah satu algoritma klasifikasi dalam *machine learning* yang menggunakan metode *supervised learning*. Model ini termasuk kategori *lazy learner* karena tidak membangun model eksplisit atau hipotesis selama pelatihan model. Sebaliknya, KNN menyimpan semua data *training* dan hanya melakukan proses klasifikasi saat ada data baru yang perlu diprediksi. Hal ini membuat KNN dikenal sebagai *instance-based classifier*.

Cara kerja KNN dimulai dengan menghitung jarak antara data baru dengan setiap instance dalam data *training*. Metode yang umum digunakan untuk menghitung jarak ini adalah *Euclidean* atau *Manhattan distance*. Setelah jarak dihitung, KNN memilih sejumlah tetangga terdekat (k) berdasarkan nilai jaraknya (jarak terkecil). Prediksi dilakukan dengan melihat mayoritas kelas dari tetangga-tetangga terdekat tersebut. Selanjutnya, program akan melabeli data input dengan mayoritas label yang sudah dipilih tadi.

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Gambar 1.1 Euclidean Distance

$$\sum_{i=1}^{k}\left|x_i - y_i\right|$$

Gambar 1.2 Manhattan Distance

$$\left(\sum_{i=1}^{k} \left(|x_i - y_i| \right)^q \right)^{1/q}$$

Gambar 1.3 Minkowski Distance

(Sumber: studymachinelearning.com)

Pada rumus di atas, X adalah data *training* dan y adalah data *testing*. Variabel k pada rumus di atas menggambarkan jumlah atribut atau kolom individu.

2. Implementasi algoritma tanpa *library*

Fungsi	Deskripsi
`__init__(self, k=5, n_jobs=1, metric='minkowski', p=2, weights='uniform')`	Inisialisasi model KNN dengan hyperparameter. **Args:** • k (int): Jumlah tetangga terdekat (default: 5). • metric (str): Metrik jarak yang digunakan ('manhattan', 'euclidean', atau 'minkowski', default: 'minkowski'). • p (int): Parameter Minkowski Distance jika metric='minkowski' (default: 2). • weights (str): Strategi pembobotan tetangga ('uniform' atau 'distance', default: 'uniform'). • n_jobs (int): Jumlah inti CPU yang digunakan untuk paralelisme (-1 untuk semua inti, default: 1). **Raises:**

	ValueError: Jika parameter tidak valid.
`get_params(self, deep=True)`	Mengembalikan hyperparameter model dalam bentuk dictionary yang digunakan untuk melakukan *hyperparameter tuning*. **Args:** ● deep (bool): Parameter opsional untuk kompatibilitas dengan Scikit-learn (default: True). **Returns:** ● dict: Dictionary yang berisi hyperparameter model.
`set_params(self, **params)`	Mengatur hyperparameter model dari dictionary yang digunakan untuk melakukan *hyperparameter tuning*. **Args:** ● **params: Hyperparameter model dalam bentuk keyword arguments. **Returns:** ● self: Mengembalikan instance model dengan hyperparameter yang diperbarui. **Raises:** ● ValueError: Jika parameter yang diberikan tidak valid.
`_compute_distances(self, test)`	Menghitung jarak antara satu data *training* dan seluruh data *testing*. **Args:**

	• test (array-like): Instance tunggal dari dataset *testing*. **Returns:** • distances (array): Array jarak antara instance *testing* dan semua instance di data *training*. **Raises:** • ValueError: Jika dimensi data *testing* tidak sesuai dengan dimensi data *training*.
`fit(self, X_train, y_train)`	Melatih model KNN dengan menyimpan data *training*. **Args:** • X_train (array-like): Data latih berupa array NumPy atau DataFrame Pandas dengan bentuk (n_samples, n_features). • y_train (array-like): Label untuk data latih dengan panjang n_samples. **Returns:** • self: Mengembalikan instance model yang sudah dilatih. **Raises:** • ValueError: Jika panjang X_train dan y_train tidak sama.
`predict(self, X_test)`	Memprediksi label untuk dataset *testing*. **Args:**

	• X_test (array-like): Data *uji* berupa array NumPy atau DataFrame Pandas dengan bentuk (n_samples, n_features). **Returns:** • np.ndarray: Array prediksi dengan panjang n_samples. **Raises:** • ValueError: Jika dimensi X_test tidak sesuai dengan dimensi data latih.
`predict_instance(row)`	Memprediksi label untuk satu data uji. **Args:** • row (array-like): Data uji berupa array dengan bentuk (n_features,). **Returns:** • any: Label yang diprediksi untuk data *testing* (tipe sesuai dengan y_train).
`save(self, path)`	Menyimpan model yang sudah dilatih ke file. **Args:** • path (str): Jalur file tempat model akan disimpan. **Returns: -**
`load(path)`	Memuat model dari file. **Args:**

	• path (str): Jalur file tempat model disimpan. **Returns:** • KNN: Instance model KNN yang dimuat dari file.

3. Langkah Implementasi

1) Melakukan inisialisasi model menggunakan fungsi __init__ denagn parameter sebagai berikut,
    - k: Jumlah tetangga terdekat yang akan digunakan untuk prediksi.
    - metric: Metode untuk menghitung jarak, seperti Euclidean, Manhattan, atau Minkowski.
    - weights: Strategi voting, yaitu:
        i) 'uniform': Semua tetangga memiliki bobot yang sama.
        ii) 'distance': Tetangga yang lebih dekat memiliki bobot lebih besar.
    - n_jobs: Jumlah inti CPU yang digunakan untuk paralelisme.
2) Melatih model dengan memberikan data latih X_train (fitur) dan y_train (label)
3) Menyimpan data latih dalam model untuk digunakan dalam prediksi.
4) Model menerima data uji X_test, yang berisi satu atau lebih sampel. Untuk setiap baris data di X_test, model memanggil fungsi predict_instance.
5) Melakukan prediksi untuk satu sampel dengan
    - menghitung jarak ke semua data latih menggunakan metric yang dipilih pada parameter.
    - memilih k tetangga terdekat berdasarkan jarak terkecil
    - Menggunakan *voting* untuk memilih prediksi untuk meningkatkan efisiensi model
        i) Jika weights='uniform', *voting* dilakukan berdasarkan mayoritas label dari tetangga terdekat.

ii) Jika weights='distance', Bobot dihitung sebagai kebalikan dari jarak dengan tetangga yang lebih dekat memiliki bobot lebih besar dalam voting.

6) Mempercepat hasil prediksi dengan menggunakan konsep paralelisme menggunakan *concurrent.futures*.
7) Menyimpan model menggunakan *pickle.*
8) Apabila model ingin digunakan tanpa dilatih ulang, maka bisa memanggil *load.*

B. Naive-Bayes

1. Dasar Teori

Naive Bayes adalah metode klasifikasi berbasis probabilitas yang didasarkan pada Teorema Bayes dengan asumsi bahwa setiap fitur saling independen (asumsi "naive"). Metode ini cocok untuk tugas klasifikasi dengan dataset berukuran besar dan sering digunakan karena kesederhanaan serta efisiensinya.

1.1 Teorema Bayes

Teorema Bayes digunakan untuk menghitung probabilitas posterior P(C∣X), yaitu probabilitas suatu kelas C diberikan data X. Rumusnya adalah:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Gambar 1.4 Teorema Bayes

- P(C∣X): Probabilitas posterior dari kelas C dengan bukti X.
- P(X∣C): Likelihood atau probabilitas data X diberikan kelas C.
- P(C): Probabilitas prior kelas C.
- P(X): Probabilitas data X.

1.2 Asumsi Naive Bayes

Naive Bayes mengasumsikan bahwa setiap fitur dalam data $X = \{x1, x2,..., Xn)$ adalah **independen secara kondisional**. Maka, likelihood P(X|C) dapat dihitung sebagai:

$$P(X|C) = \prod_{i=1}^{n} P(x_i|C)$$

Gambar 1.5 *Likelihood*

1.3 Rumus Naive Bayes

Untuk data kontinu, Gaussian Naive Bayes menggunakan **Distribusi Normal (Gaussian)** untuk menghitung likelihood P(xi|C). Rumus probabilitas kepadatan Gaussian adalah:

$$P(x_i|C) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

Gambar 1.6 Rumus Gaussian Naive Bayes

(Sumber: scikitlearn- documentation)

- μ : Mean (rata-rata) dari fitur Xi dalam kelas C.
- σ^2: Variansi dari fitur Xi dalam kelas C.

2. **Implementasi algoritma tanpa *library***

Fungsi	Deskripsi
`def __init__(self)`	Menginisialisasi model Gaussian Naive Bayes dengan atribut untuk menyimpan data pelatihan, probabilitas prior, rata-rata, dan variansi fitur.
`def fit(self, X, y)`	Melatih model Gaussian Naive Bayes dengan menghitung probabilitas prior, rata-rata, dan variansi fitur berdasarkan data pelatihan. **Args:** • **X** : Data fitur dalam bentuk array dua dimensi. • **y** : Label data dalam bentuk array satu dimensi **Returns:** • **self**: Mengembalikan instance model yang telah dilatih.
`def gaussian(self, x, mean, var)`	Menghitung nilai fungsi probabilitas kepadatan Gaussian untuk sebuah nilai tertentu berdasarkan mean dan variansi. **Args:** • **nilai:** Nilai fitur yang ingin dihitung probabilitasnya. • **mean:** Rata-rata dari fitur untuk kategori tertentu. • **varians:** Variansi dari fitur untuk kategori tertentu**.** **Returns :** • **float:** Nilai probabilitas Gaussian**.**
`def predict(self, X)`	Memprediksi label untuk data baru berdasarkan model Gaussian Naive Bayes yang telah dilatih.

	Args: • **X**: Data fitur baru dalam bentuk array dua dimensi. Returns: • **numpy.array**: Array satu dimensi berisi label prediksi untuk setiap sampel dalam data. ValueError: • Jika jumlah kolom dalam data baru tidak sesuai dengan jumlah fitur dalam data pelatihan.
`def score(self, X, y)`	Menghitung akurasi prediksi model dengan membandingkan label yang diprediksi dengan label sebenarnya. Args: • **X**: Data fitur dalam bentuk array dua dimensi. • **y**: Label data sebenarnya dalam bentuk array satu dimensi. Returns: • **float**: Akurasi model dalam bentuk nilai antara 0 dan 1.
`save(self, path)`	Menyimpan model yang sudah dilatih ke file. **Args:** • path (str): Jalur file tempat model akan disimpan. **Returns: -**

C. Langkah Implementasi

1) Melakukan inisialisasi model menggunakan fungsi **__init__** dengan parameter berikut:
    - Tidak ada parameter yang diberikan langsung ke fungsi ini.
    - Model akan menginisialisasi atribut berikut:
        - fitur (self.fitur): Untuk menyimpan data fitur dari data pelatihan.
        - label (self.label): Untuk menyimpan label dari data pelatihan.
        - kategori (self.kategori): Untuk menyimpan daftar unik kategori dari label pelatihan.
        - probabilitas_prior (self.probabilitas_prior): Menyimpan probabilitas prior untuk setiap kategori.
        - mean_fitur (self.mean_fitur): Menyimpan rata-rata fitur untuk setiap kategori.
        - varians_fitur (self.varians_fitur): Menyimpan variansi fitur untuk setiap kategori.
2) Melatih model dengan memberikan data pelatihan X_train (fitur) dan y_train (label):
   Fungsi **fit** akan:
    - Mengonversi data pelatihan ke array NumPy.
    - Menghitung probabilitas prior P(C) untuk setiap kategori.
    - Menghitung rata-rata μ dan variansi σ^2 untuk setiap fitur pada setiap kategori.
3) Menyimpan informasi pelatihan dalam model:
    - Data pelatihan (fitur, label, dan statistik fitur) disimpan dalam atribut model untuk digunakan selama prediksi.
4) Model menerima data uji X_test, yang berisi satu atau lebih sampel :
    - Fungsi **predict** digunakan untuk memprediksi label untuk setiap sampel dalam data uji.

- Untuk setiap sampel, model menghitung probabilitas posterior $P(C|X)$ untuk setiap kategori menggunakan:
    - Probabilitas prior $P(C)$.
    - **Likelihood $P(X|C)$**, yang dihitung menggunakan fungsi probabilitas kepadatan Gaussian (**gaussian**).

5) Melakukan prediksi untuk satu sampel dengan:
    - Menghitung nilai probabilitas posterior untuk semua kategori.
    - Memilih kategori dengan probabilitas posterior tertinggi sebagai prediksi.
6) Menggunakan voting berbasis probabilitas posterior:
    - Model akan memilih kategori berdasarkan nilai probabilitas tertinggi yang dihitung dari likelihood dan prior.
    - Model tidak menggunakan bobot voting seperti pada algoritma KNN, karena prediksi dilakukan secara langsung berdasarkan probabilitas.

BAB IV
Data Cleaning

A. *Data Cleaning*

1. Missing Values

Pengisian *missing values* dilakukan menggunakan ***domain specific strategies*** dari jurnal yang disediakan serta informasi *phising* dan *non-phising* lainnya. Kami menggunakan *pipeline* untuk memudahkan transformasi.

Fungsi	Deskripsi
`HandleLength(BaseEstimator, TransformerMixin)`	Menangani panjang URL dan domain untuk menambah informasi ke dataset. **Transformasi:** 1. Menghapus nilai kosong pada kolom 'URL' untuk data *training*, mengganti dengan string kosong untuk data *testing*. 2. Mengekstrak domain dari URL jika kolom 'Domain' kosong. 3. Mengekstrak TLD dari domain jika kolom 'TLD' kosong. 4. Menghitung panjang URL dan menyimpannya di kolom 'URLLength'. 5. Menghitung panjang domain dan menyimpannya di kolom 'DomainLength'. 6. Menghitung panjang TLD dan menyimpannya di kolom 'TLDLength'. **Alasan:** Berdasarkan jurnal yang disediakan dari dataset original

	(PHILUsil), panjang URL dan domain sering kali menjadi indikator penting untuk mendeteksi *phishing*.
`HandleIsHTTPS(BaseEstimator, TransformerMixin)`	Menambahkan fitur biner 'IsHTTPS' untuk menandai apakah URL menggunakan protokol HTTPS. **Transformasi:** 1. Menghapus nilai kosong pada kolom 'URL' untuk data *training*, atau mengganti dengan string kosong untuk data *testing*. 2. Membuat kolom 'IsHTTPS' yang bernilai 1 jika URL menggunakan HTTPS, dan 0 jika tidak. **Alasan:** Penggunaan protokol HTTPS sering kali menjadi tanda bahwa sebuah situs web lebih aman. Berdasarkan hasil EDA, IsHTTPS memiliki korelasi positif yang tinggi terhadap target label.
`HandleCharContinuationRate(BaseEstimator, TransformerMixin)`	Menghitung rasio karakter berurutan dalam URL untuk menambah fitur numerik baru. **Transformasi:** 1. Membagi URL menjadi urutan karakter berdasarkan huruf, angka, atau simbol. 2. Menghitung total panjang urutan karakter. 3. Menghasilkan fitur 'CharContinuationRate'. **Alasan:** URL phishing sering kali memiliki pola karakter yang tidak biasa atau kompleks
`HandleHasTitle(BaseEstimator,`	Menambahkan fitur biner 'HasTitle' untuk menandai apakah halaman memiliki judul.

`TransformerMixin)`	**Transformasi:** 1. Menandai 1 pada kolom 'HasTitle' jika 'Title' tidak kosong, atau 0 jika kosong. Alasan: Phishing URL sering kali tidak memiliki judul halaman yang sesuai atau bahkan tidak memiliki judul sama sekali. Fitur ini membantu model mendeteksi pola tersebut.
`HandleURLTitleMatchScore(BaseEstimator, TransformerMixin)`	Menghitung skor kecocokan antara URL dan judul halaman untuk menambah fitur numerik. **Transformasi:** 1. Membersihkan URL dengan menghapus protokol dan elemen awal yang tidak relevan. 2. Membandingkan kata-kata dalam judul dengan bagian URL yang relevan. 3. Menghitung skor kecocokan dengan skala 0-100 berdasarkan panjang kata yang cocok. Alasan: URL phishing sering kali memiliki perbedaan signifikan antara judul halaman dan URL. Skor kecocokan ini membantu mendeteksi anomali tersebut.
`FeatureImputer(BaseEstimator, TransformerMixin)`	Mengisi *missing values* pada dataset berdasarkan jenis kolom. **Transformasi:** 1. Mengisi kolom boolean dengan modus (nilai paling sering muncul).

	2. Mengisi kolom numerik menggunakan *Iterative Imputer* yang memprediksi nilai berdasarkan pola antar kolom. **Alasan:** Nilai kosong dalam dataset dapat mengganggu performa model, terutama jika fitur yang hilang penting. Digunakan algoritma ***Iterative Imputer*** untuk melakukan imputasi tersebut.
`FeatureDropper(BaseEstimator, TransformerMixin)`	Menghapus kolom tertentu dari dataset untuk mengurangi fitur yang tidak relevan. **Transformasi:** 1. Memeriksa apakah kolom yang akan dihapus ada dalam dataset. 2. Menghapus kolom tersebut jika ditemukan. **Alasan:** Kolom yang dihapus tidak relevan untuk prediksi atau dapat menyebabkan menurunnya performa model sehingga perlu untuk dihapus.

2. Outliers

Fungsi	Deskripsi
`handle_outliers(data, method="iterative", estimator=None)`	Menangani outlier pada data numerik dengan metode ***IQR (Interquartile Range)*** dan imputasi nilai yang hilang. **Transformasi:** 1. Menghitung batas bawah dan atas untuk mendeteksi outlier menggunakan metode IQR. 2. Menandai nilai sebagai outlier jika berada di bawah batas bawah atau di atas batas atas dengan menggantinya dengan NaN.

	3. Mengisi nilai NaN (outlier) dengan salah satu metode berikut: ● "iterative": Menggunakan ***Iterative Imputer*** dengan estimator yang diberikan. ● "median": Mengisi dengan ***nilai median*** dari kolom tersebut. ● Nilai NaN akan dihapus. **Alasan:** Outlier dapat mempengaruhi performa model prediksi dengan menimbulkan bias atau *noise*. Teknik yang digunakan merupakan iterative imputing karena jumlah outlier setiap kolom < 5000 sehingga dapat dilakukan imputing dengan neighbor terdekatnya dan tidak begitu mempengaruhi pada prediksi.

3. Duplicates

Fungsi	Deskripsi

`.drop_duplicates(inplace=True)`
 	
`.reset_index(drop=True, inplace=True)`
	Transformasi:
	1. Menghapus baris yang memiliki nilai duplikat di seluruh kolom dataset menggunakan `drop_duplicates`.
	2. Menampilkan jumlah baris sebelum dan setelah proses penghapusan duplikat untuk pelacakan perubahan.
	3. Mereset indeks dataset menggunakan `reset_index` agar urutan indeks konsisten setelah penghapusan duplikat.
	
	Alasan:
	- Duplikasi data dapat menyebabkan bias karena informasi yang sama dihitung lebih dari satu kali.
	- Mereset indeks memastikan dataset tetap memiliki indeks yang konsisten, terutama jika data diolah lebih lanjut.

B. *Data Preprocessing*

1. Feature Scaling

Feature scaling dilakukan berdasarkan hasil EDA yang menunjukan bahwa pada dataset kami ditemukan cukup banyak outlier dengan data yang tidak terdistribusi normal.

Fungsi	Deskripsi

`.drop_duplicates(inplace=True)`
 	
`.reset_index(drop=True, inplace=True)`
	Melakukan scaling pada fitur numerik menggunakan **RobustScaler** dan **log transformation** pada kolom 'URLTitleMatchScore'.
	
	Transformasi:
	1. Mengidentifikasi kolom numerik (int64 dan float64) yang tidak biner dan bukan 'URLTitleMatchScore'.
	2. Melakukan scaling pada kolom numerik menggunakan RobustScaler untuk mengurangi sensitivitas terhadap outlier.
	3. Menerapkan log transformation pada 'URLTitleMatchScore' untuk mengurangi skala distribusi yang sangat lebar.
	
	Alasan:
	1. RobustScaler digunakan karena ukuran dataset yang besar dan memiliki cukup banyak outliers.
	2. Log transformation diterapkan pada 'URLTitleMatchScore' untuk menangani distribusi data yang sangat *skewed* dan URLTitleMatchScore diharapkan memiliki distribusi yang normal.

2. Feature Encoding

Untuk seluruh data object sudah dilakukan cleaning sehingga tidak diperlukan Feature Encoding. Selain itu, menurut EDA pada Tugas Kecil 1,

seluruh data object memiliki kardinalitas yang tinggi yang dapat mempengaruhi fenomena *curse of dimensionality*.

3. Handling Imbalanced Dataset

Fungsi	Deskripsi

`def balance_classes(X_t, y_t):`
`    sm = SMOTE(random_state=42)`
`    X_t_res, y_t_res = sm.fit_resample(X_t, y_t)`
`    return X_t_res, y_t_res`
	Melakukan oversampling dengan tujuan untuk meningkatkan persebaran data yang lebih sedikit menjadi seimbang dengan data yang merupakan *majority*.
	
	Transformasi:
	1. Inisialisasi library imblearn untuk memanggil fungsi SMOTE dengan mengisi random state
	2. Melakukan fit resample pada X_train dan y_train
	
	Alasan :
	Untuk menyeimbangkan prediksi antar kelas supaya seimbang dan memiliki *resources* yang cukup untuk melakukan prediksi pada *unseen data*

`def balance_classes_undersample(X_t, y_t):`
`    undersampler = RandomUnderSampler(random_state=42)`
`    X_t_res, y_t_res = undersampler.fit_resample(X_t, y_t)`
`    return X_t_res, y_t_res`
	Melakukan undersampling dengan tujuan untuk mengurangi persebaran data yang lebih banyak menjadi seimbang dengan data yang merupakan minority.
	
	Transformasi:
	1. Inisialisasi library imblearn untuk memanggil fungsi RandomUnderSampler dengan mengisi random state
	2. Melakukan fit resample pada X_train dan y_train

	Alasan : Untuk menyeimbangkan prediksi antar kelas supaya seimbang dan memiliki *resources* yang cukup untuk melakukan prediksi pada *unseen data*

Setelah melakukan beberapa eksperimen, hasil balancing menggunakan SMOTE lebih memberikan representasi terhadap model sehingga dapat dilakukan dengan performa yang lebih baik.

BAB V

Feature Engineering

A. Feature Selection & Binning (Discretization)

Feature selection adalah proses memilih fitur-fitur (variabel) yang paling relevan dan informatif dari dataset untuk digunakan dalam model machine learning. Tujuan dari feature selection adalah meningkatkan performa model, mengurangi kompleksitas data, dan mencegah overfitting dengan menghapus fitur yang tidak relevan, redundan, atau tidak memberikan kontribusi signifikan terhadap prediksi. Tujuan feature selection :

a. Mengurangi dimensi data
b. Meningkatkan fitur yang relevan
c. Mengurangi waktu komputasi
d. Mempermudah implementasi model selanjutnya
e. Mengurangi kemungkinan overfitting

Binning atau discretization adalah proses membagi fitur numerik kontinu menjadi beberapa kategori atau interval diskret. Teknik tersebut digunakan untuk menyederhanakan data numerik dan mengurangi sensitivitas terhadap outlier sehingga membantu meningkatkan performa model. Tujuan binning :

a. Mengurangi dimensionality
b. Mengurangi outlier karena mengkategorikannya jadi beberapa bagian
c. Mempermudah analisis
d. Meningkatkan kinerja beberapa model

Fungsi	Deskripsi

`fit(self, X, y=None)`	Transformasi: ● Menyimpan nama-nama kolom dari data input ke dalam atribut self.fitted_cols Alasan: ● Fungsi tersebut digunakan untuk mempelajari struktur awal dataset sebelum dilakukan transformasi
`transform(self, X)`	Transformasi : ● Kolom IsCountryCodeTLD direncanakan untuk dibuat berdasarkan apakah nilai dalam kolom TLD termasuk dalam daftar kode negara (disertakan dalam komentar). Fungsi categorize_tld akan memberikan nilai 1 jika TLD adalah kode negara, dan 0 jika tidak ● Menambahkan kolom baru IsHighURLMatchScore dengan nilai 1 jika nilai dalam URLTitleMatchScore lebih besar dari 20, dan 0 jika sebaliknya Alasan : ● Menambahkan informasi tambahan untuk membantu model dalam membedakan karakteristik data, seperti identifikasi apakah sebuah URL memiliki skor kecocokan tinggi (IsHighURLMatchScore) atau apakah TLD merupakan kode negara (IsCountryCodeTLD)
`X['IsHighURLMatchScore'`	Binning dilakukan karena distribusi fitur ini memiliki

] = X['URLTitleMatchScore'].apply(lambda x: 1 if x > 20 else 0)	rentang yang lebar dengan outlier yang signifikan. Kategorisasi membantu menyederhanakan analisis, mengurangi sensitivitas model terhadap noise, serta memperjelas pola hubungan dengan target. Selain itu, binning dapat menangani distribusi yang skewed, sehingga menciptakan kategori nilai seperti rendah, sedang, dan tinggi untuk mendukung interpretasi dan prediksi model yang lebih baik. Alasan pemilihan feature : Pada saat melakukan analisis distribusi, terlihat bahwa distribusi antar skor sangat jauh yang dapat dilakukan kategorisasi skor yang kurang dari 20 dan lebih dari 20
def categorize_tld(tld): return 1 if tld.lower() in country_code_tlds else 0 X['IsCountryCodeTLD'] = X['TLD'].apply(categorize_tld)	**Alasan Pemilihan Feature:** Saat melakukan analisis data, ditemukan bahwa URL dengan **TLD negara resmi** memiliki pola penggunaan tertentu, terutama untuk domain yang digunakan oleh organisasi lokal atau individu dalam negara tertentu. Hal ini dapat membantu membedakan antara URL yang legitimate dan URL phishing, karena banyak URL phishing menggunakan TLD generik atau tidak resmi untuk mengelabui pengguna. Dengan menambahkan fitur ini, model dapat lebih efektif dalam mengidentifikasi pola-pola tersebut dan meningkatkan performa dalam mendeteksi URL phishing.
def categorize_char_continu	

ation_rate(rate): return 1 if rate > 0.9 else 0 X['CharContinuationRateCategory'] = X['CharContinuationRate'].apply(categorize_char_continuation_rate)	

def categorize_largest_line_length(length):
if length < 5000:
return 0
elif 5000 <= length <= 10000:
return 1
else:
return 2
 	
X['LargestLineLengthCategory'] = X['LargestLineLength'].apply(categorize_largest_line_length)
	Alasan pemilihan fitur :
	nilai **LargestLineLength** memiliki rentang yang sangat lebar. Untuk menangani hal ini, data dikelompokkan ke dalam tiga kategori berdasarkan nilai length: **<3000**, **3000–7000**, dan **>7000**. Kategorisasi ini juga menyederhanakan proses analisis dan memungkinkan model untuk fokus pada pola utama tanpa dipengaruhi oleh

BAB V

Perbandingan

A. Perbandingan KNN

- **Metrics evaluasi sklearn**

	precision	recall	f1-score	support
0.0	0.931937	0.985467	0.957955	1445.00000
1.0	0.998818	0.994172	0.996489	17844.00000
accuracy	0.993520	0.993520	0.993520	0.99352
macro avg	0.965377	0.989819	0.977222	19289.00000
weighted avg	0.993807	0.993520	0.993603	19289.00000

- **Metrics evaluasi scratch**

	precision	recall	f1-score	support
0.0	0.964983	0.991696	0.978157	1445.000000
1.0	0.999326	0.997086	0.998205	17844.000000
accuracy	0.996682	0.996682	0.996682	0.996682
macro avg	0.982155	0.994391	0.988181	19289.000000
weighted avg	0.996753	0.996682	0.996703	19289.000000

Kedua gambar tersebut menunjukan adanya perbedaan hasil evaluasi dari model KNN dengan menggunakan sklearn dan *from scratch.* Hasil tersebut menunjukkan akurasi yang sangat mirip dengan selisih yang cenderung kecil pada sebagian besar metrik evaluasi. Akurasi kedua model hampir setara dengan selisih hanya sekitar **0,3162%** (99,3520% untuk *library* dan 99,6682% untuk *scratch*). Pada macro average precision, model scratch unggul dengan selisih **1,6781%** dibandingkan library.

Secara keseluruhan, implementasi manual dari model KNN memberikan hasil yang sedikit lebih baik pada beberapa metrik, terutama **precision** dan **recall**. Hal ini menandakan bahwa kedua pendekatan menggunakan logika dan algoritma yang serupa

dengan library lebih unggul dari sisi efisiensi waktu komputasi, sementara implementasi scratch lebih cocok untuk pembelajaran algoritma secara mendalam.

Keunggulan performa implementasi KNN dari scratch dibandingkan library kemungkinan besar disebabkan oleh penyesuaian yang lebih spesifik terhadap dataset. Dalam implementasi manual, preprocessing data dapat disesuaikan lebih optimal. Selain itu, penanganan data minoritas atau tie-breaking dapat dirancang lebih relevan untuk dataset tertentu (perbedaan weight untuk label 0), sementara library cenderung menggunakan pendekatan yang lebih umum. Dengan fleksibilitas ini, implementasi manual dapat lebih unggul dalam menangkap pola data, meskipun memerlukan lebih banyak waktu dan upaya dibandingkan library yang mengutamakan efisiensi.

B. Perbandingan *Gaussian Naive-Bayes*

- **Metrics evaluasi sklearn**

	precision	recall	f1-score	support
0.0	0.965944	0.863668	0.911947	1445.000000
1.0	0.989054	0.997534	0.993276	17844.000000
accuracy	0.987506	0.987506	0.987506	0.987506
macro avg	0.977499	0.930601	0.952612	19289.000000
weighted avg	0.987323	0.987506	0.987183	19289.000000

- **Metrics evaluasi scratch**

	precision	recall	f1-score	support
0.0	0.939956	0.877509	0.907659	1445.000000
1.0	0.990134	0.995461	0.992790	17844.000000
accuracy	0.986625	0.986625	0.986625	0.986625
macro avg	0.965045	0.936485	0.950225	19289.000000
weighted avg	0.986375	0.986625	0.986413	19289.000000

Berdasarkan perhitungan selisih persentase antara hasil evaluasi model Gaussian Naive Bayes dari **scikit-learn** dan **implementasi *scratch***, terlihat bahwa perbedaan

performa sangat kecil. Untuk metrik **accuracy**, selisihnya hanya sekitar **0.0881%**, menunjukkan bahwa kedua model mampu memberikan hasil klasifikasi yang sangat konsisten. Pada metrik **macro average precision**, perbedaannya sedikit lebih besar, yaitu **1.2454%**, tetapi tetap dalam rentang yang dapat diterima. Untuk metrik **macro average recall** dan **f1-score**, selisihnya masing-masing hanya **0.5884%** dan **0.2387%**, yang juga menunjukkan kesamaan performa yang sangat baik.

Secara keseluruhan, hasil ini membuktikan bahwa **implementasi *scratch*** mampu menghasilkan performa yang hampir setara dengan model yang dihasilkan oleh **scikit-learn**, meskipun mungkin membutuhkan waktu komputasi lebih lama. Selisih persentase yang kecil ini menunjukkan bahwa pendekatan manual telah menggunakan rumus dan cara yang sama dengan **scikit-learn**.

BAB VI

Kesimpulan dan Saran

A. Kesimpulan

Algoritma K-Nearest Neighbors (KNN) dan Gaussian Naive Bayes dapat digunakan untuk mendeteksi URL phishing. Implementasi algoritma *from scratch* menunjukkan performa hampir setara dengan implementasi menggunakan library Scikit-Learn, meskipun terdapat sedikit perbedaan pada metrik evaluasi. Hal tersebut membuktikan bahwa algoritma manual dapat memberikan hasil yang baik apabila metode dan langkah-langkah implementasinya dilakukan dengan tepat. Preprocessing data, seperti penanganan missing values, outliers, dan balancing dataset, memainkan peran penting dalam meningkatkan akurasi model. Selain itu, teknik *feature engineering* seperti feature selection dan binning berhasil menyederhanakan data serta meningkatkan relevansi fitur terhadap target. Pada hasil perbandingan performa, model KNN menunjukkan selisih evaluasi sebesar 0.3162% antara implementasi manual dan Scikit-Learn, sementara Gaussian Naive Bayes memiliki selisih performa yang sangat kecil, dengan maksimum 1.2454% pada metrik precision. Meskipun implementasi

manual membutuhkan waktu komputasi yang lebih lama, hasil ini menunjukkan konsistensi algoritma dalam mendeteksi URL phishing.

B. Saran

Pengembangan model dapat dilakukan dengan mencoba algoritma machine learning lain, seperti Random Forest atau Support Vector Machine (SVM), untuk membandingkan performa dengan KNN dan Naive Bayes. Selain itu, penggunaan metode ensemble seperti bagging atau boosting dapat meningkatkan akurasi dan stabilitas prediksi. Dalam menggunakan device yang cukup baik untuk mempercepat proses komputasi terutama dikarenakan dataset yang digunakan besar, dan melakukan hyperparameter tuning yang lebih intensif. Dari sisi dataset, penambahan data dari sumber lain atau penggunaan teknik augmentation dapat meningkatkan generalisasi model dalam mendeteksi URL phishing yang lebih beragam. Model yang telah dihasilkan juga dapat diintegrasikan ke dalam sistem keamanan berbasis web untuk deteksi phishing secara otomatis. Penelitian lebih lanjut diperlukan untuk mengukur performa model dalam mendeteksi URL phishing secara real-time. Dengan langkah-langkah tersebut, model dapat lebih efisien dan efektif dalam menangani ancaman phishing di berbagai skenario aplikasi.

Pembagian Tugas

Nama Lengkap	NIM	Deskripsi Tugas
Jihan Aurelia	18222001	Implementasi KNN
		Laporan
Nasywaa Anggun Athiefah	18222021	Implementasi *Feature Engineering*
		Laporan
Ricky Wijaya	18222043	Implementasi Naive Bayes
		Laporan
Timotius Vivaldi	18222091	*Data Cleaning* dan *Pre-Processing*
		Laporan

REFERENSI

Brownlee, J. (2021, October 12). *Stochastic Hill Climbing in Python from Scratch - MachineLearningMastery.com*. Machine Learning Mastery. Retrieved October 2, 2024, from https://machinelearningmastery.com/stochastic-hill-climbing-in-python-from-scratch/

Jain, S. (2023, April 20). *Introduction to Hill Climbing | Artificial Intelligence*. GeeksforGeeks. Retrieved October 2, 2024, from https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

Putria, N. E. (2022). Comparison of Simple Hill Climbing Algorithm and Stepest Ascent Hill Climbing Algorithm in The Game Order of Numbers. *International Journal of Information System & Technology*, *6*(1), 33-40.

Samosir, S. A. (2019). IMPLEMENTASI METODE STEEPEST ASCENT HILL CLIMBING DALAM PENCARIAN RUTE TERDEKAT PROMOSI KAMPUS STMIK BUDI DARMA. *Jurnal Pelita Informatika*, *8*(2), 283-287.

Silvilestari. (2021). Steepest Ascent Hill Climbing Algorithm To Solve Cases In Puzzle Game 8. *International Journal of Information System & Technology*, *5*(4), 366-370.

Stochastic Hill Climbing With Random Restarts. (n.d.). Algorithm Afternoon. Retrieved October 2, 2024, from https://algorithmafternoon.com/stochastic/stochastic_hill_climbing_with_random_restarts/

Slide Kuliah IF3070 - Foundations of Artificial Intelligence.

KNeighborsClassifier — scikit-learn 1.5.2 documentation. (n.d.). Scikit-learn. Retrieved December 22, 2024, from https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsClassifier.html