Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin

Disusun oleh:

Kelompok 36

Jeremy Deandito	/ 18222112
Nathaniel Liady	/ 18222114
Gabriel Marcellino	/ 18222115
Nicolas Jeremy	/ 18222135

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

1. Pendahuluan

Pembelajaran mesin telah menjadi salah satu cabang utama dari kecerdasan buatan yang banyak diaplikasikan dalam berbagai bidang, termasuk keamanan siber. Dengan kemampuan untuk mempelajari pola dari data dan membuat prediksi atau keputusan tanpa program eksplisit, pembelajaran mesin memberikan solusi efektif untuk menghadapi tantangan yang kompleks, seperti deteksi phishing.

Phishing adalah salah satu bentuk ancaman siber yang semakin berkembang pesat. Dalam upaya mengidentifikasi URL phishing, penggunaan algoritma pembelajaran mesin seperti *K-Nearest Neighbors* (KNN) dan Gaussian Naive-Bayes menjadi pendekatan yang menjanjikan. KNN dikenal sebagai algoritma berbasis instance yang sederhana namun efektif, sedangkan Gaussian Naive-Bayes menawarkan kecepatan dan efisiensi dalam menangani data dengan distribusi probabilitas tertentu.

Algoritma KNN dan Gaussian Naive-Bayes akan diimplementasikan secara *from scratch* menggunakan bahasa Python dan menggunakan library Scikit-learn. Implementasi dilakukan dengan memanfaatkan pustaka matematika seperti NumPy untuk memastikan bahwa algoritma dapat dibangun dari dasar dengan pemahaman mendalam. Dataset yang digunakan adalah PhiUSIIL Phishing URL Dataset, yang berisi deskripsi URL, fitur-fitur terkait, serta label untuk membedakan antara URL yang sah (*legitimate*) dan URL phishing. Dengan dataset ini, fitur-fitur yang diekstraksi dari *source code* dan URL dapat menjadi bahan pelatihan bagi model pembelajaran mesin yang diimplementasikan.

1.1 Penjelasan KNN

K-Nearest Neighbors (KNN) adalah algoritma pembelajaran mesin berbasis instance yang digunakan untuk tugas klasifikasi dan regresi. KNN bekerja dengan mencari K neighbors terdekat dari data baru berdasarkan jarak tertentu, seperti jarak

Euclidean, Manhattan, atau lainnya. Data baru kemudian diklasifikasikan ke dalam kelas mayoritas dari neighbors terdekatnya (pada kasus klasifikasi) atau dihitung rata-ratanya (pada regresi). KNN mudah diimplementasikan dan tidak memerlukan proses pelatihan, tetapi menjadi kurang efisien untuk dataset besar karena memerlukan perhitungan jarak setiap kali data baru diberikan.

1.2 Penjelasan Naive Bayes

Naive Bayes adalah algoritma machine learning berbasis probabilistik yang didasarkan pada Teorema Bayes dengan asumsi bahwa setiap fitur bersifat independen satu sama lain (asumsi "naive"). Algoritma ini sangat cepat dalam pelatihan dan pengujian, sehingga cocok untuk dataset berukuran besar, terutama dalam klasifikasi teks seperti pengelompokan email spam atau analisis sentimen. Meskipun asumsi independensinya sering tidak realistis, Naive Bayes tetap memberikan hasil yang cukup baik dalam banyak aplikasi praktis, terutama ketika dataset memiliki struktur yang sederhana.

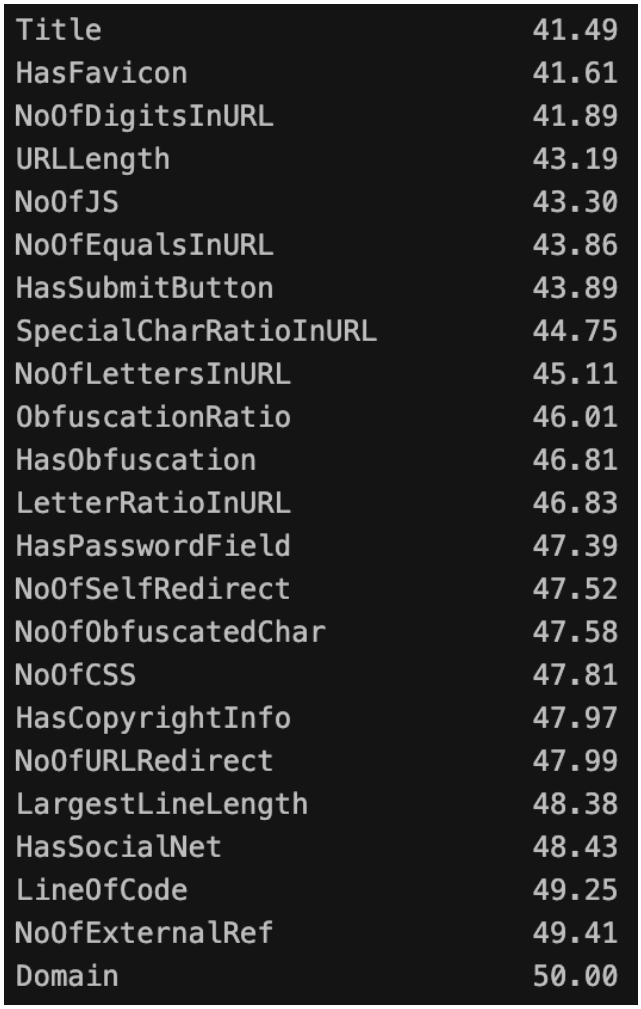
2. Implementasi & Pembahasan

2.1 Tahap *Cleaning & Preprocessing*

Setelah mendapati pengetahuan yang didapati pada tahap EDA sebelumnya, langkah selanjutnya adalah melakukan *cleaning* dan *preprocesessing* dari dataset link phising ini. Proses ini dilakukan untuk meningkatkan akurasi model yang dibuat. Variabel *features* akan sangat memengaruhi model dalam melakukan prediksi. Semakin baik *features,* maka akan semakin baik juga model untuk memberikan hasil yang maksimal.

Langkah pertama yang dilakukan adalah mengetahui jumlah data yang hilang untuk masing-masing *features.* Berikut merupakan presentasi *missing values* pada dataset.

URL	15.50
IsDomainIP	30.01
IsResponsive	30.30
NoOfEmptyRef	30.40
Pay	30.75
NoOfPopup	30.88
HasHiddenFields	31.19
NoOfSubDomain	31.38
NoOfQMarkInURL	31.41
HasTitle	31.75
TLD	32.33
NoOfAmpersandInURL	32.33
DomainLength	32.99
Robots	33.28
NoOfOtherSpecialCharsInURL	33.92
TLDLength	34.00
CharContinuationRate	34.22
NoOfSelfRef	34.28
IsHTTPS	35.16
NoOfiFrame	35.57
DomainTitleMatchScore	35.61
Crypto	35.75
NoOfImage	35.95
URLCharProb	37.09
URLTitleMatchScore	37.19
TLDLegitimateProb	37.66
DigitRatioInURL	38.11
HasDescription	38.92
Bank	39.17
HasExternalFormSubmit	39.59



Gambar 2.1.1 Presentasi *missing values features*

Berikut merupakan langkah-langkah yang dilakukan untuk membersihkan *missing values* pada *features:*

1. ***Handling* URL, Domain, TLD**

   Kolom URL, Domain, dan TLD merupakan kolom terpenting pada features di sini. Hal ini karena ketiga kolom tersebut bisa membantu mengekstrasi kolom-kolom lainnya. Karena itu, ketiga kolom ini akan di-*handle* terlebih dahulu. Untuk URL yang memiliki *missing values,* akan diisi dengan kolom Domain karena Domain merupakan kolom ektrasi Domain dari URL. Jika ternyata kolom

Domain kosong, URL akan diisi dengan URL untuk menjaga fungsionalitas dari train.csv dan test.csv (terdapat constraint jumlah row test.csv harus tetap sama).

2. **Ekstrasi dari URL, Domain, dan TLD**

   Ada kolom-kolom yang bisa didapatkan informasinya dari URL, Domain, dan TLD. Berikut merupakan kolom-kolom yang mendapatkan informasi melalui kolom URL, Domain, dan TLD:

   1. **DomainLength**

      Kolom ini mengukur panjang dari nama domain utama dalam URL.

      Proses Ekstraksi: Menggunakan fungsi domain_length(x) yang mengukur jumlah karakter dalam nama domain utama dari kolom URL atau Domain.

   2. **URLLength**

      Kolom ini mengukur panjang keseluruhan dari URL.

      Proses Ekstraksi: Fungsi url_length(x) mengukur jumlah karakter dalam seluruh URL yang ada di kolom URL.

   3. **IsDomainIP**

      Kolom ini menunjukkan apakah domain dalam URL adalah sebuah alamat IP (misalnya, 192.168.1.1) atau bukan.

      Proses Ekstraksi: Fungsi is_domain_ip(x) memeriksa apakah bagian domain dari URL (yang bisa diekstraksi dari kolom URL) merupakan alamat IP.

   4. **IsHTTPS**

      Kolom ini menentukan apakah URL menggunakan protokol HTTPS.

      Proses Ekstraksi: Fungsi isHttps(x) memeriksa apakah URL pada kolom URL menggunakan protokol HTTPS, yakni dimulai dengan https://.

   5. **NoOfSubDomain**

      Kolom ini menghitung jumlah subdomain dalam URL.

      Proses Ekstraksi: Fungsi count_subdomains(x) menghitung berapa banyak subdomain yang ada di dalam URL yang terdapat pada kolom URL. Subdomain adalah bagian dari URL yang berada di depan nama domain utama, seperti subdomain.example.com.

6. **CharContinuationRate**

   Kolom ini mengukur tingkat kelanjutan karakter dalam URL, misalnya seberapa banyak karakter berturut-turut yang sama dalam URL.

   Proses Ekstraksi: Fungsi char_continuation_rate menghitung seberapa sering karakter yang sama berulang secara berturut-turut di dalam URL yang ada di kolom URL.

7. **TLDLength**

   Kolom ini mengukur panjang dari Top-Level Domain (TLD) dalam URL.

   Proses Ekstraksi: Fungsi tld_length(x) mengukur panjang dari bagian TLD dalam URL yang diambil dari kolom TLD. Misalnya, untuk URL https://example.com, TLD-nya adalah .com.

8. **NoOfLettersInURL**

   Kolom ini menghitung jumlah huruf dalam URL.

   Proses Ekstraksi: Fungsi count_letters(x) menghitung jumlah huruf (A-Z, a-z) dalam kolom URL, mengecualikan angka dan karakter khusus.

9. **LetterRatioInURL**

   Kolom ini mengukur rasio huruf terhadap total karakter dalam URL.

   Proses Ekstraksi: Fungsi calc_letter_ratio(x) menghitung rasio antara jumlah huruf dengan panjang total URL, yang dapat dihitung dari kolom URL.

10. **NoOfDigitsInURL**

    Kolom ini menghitung jumlah angka dalam URL.

    Proses Ekstraksi: Fungsi count_digits(x) menghitung jumlah digit (0-9) yang ada dalam kolom URL.

11. **DigitRatioInURL**

    Kolom ini mengukur rasio angka terhadap total karakter dalam URL.

    Proses Ekstraksi: Fungsi calc_digit_ratio(x) menghitung rasio antara jumlah digit dengan panjang total URL, yang bisa diekstraksi dari kolom URL.

12. **NoOfEqualsInURL**

    Kolom ini menghitung jumlah tanda = dalam URL.

Proses Ekstraksi: Fungsi count_equals(x) menghitung jumlah tanda = yang ada dalam URL pada kolom URL, yang sering digunakan dalam query string (misalnya, key=value).

13. **NoOfQMarkInURL**

    Kolom ini menghitung jumlah tanda ? dalam URL.

    Proses Ekstraksi: Fungsi count_qmark(x) menghitung jumlah tanda ? yang muncul dalam kolom URL, yang menandakan awal dari query string dalam URL.

14. **NoOfAmpersandInURL**

    Kolom ini menghitung jumlah tanda & dalam URL.

    Proses Ekstraksi: Fungsi count_ampersand(x) menghitung jumlah tanda & dalam URL, yang digunakan untuk memisahkan parameter dalam query string.

15. **NoOfOtherSpecialCharsInURL**

    Kolom ini menghitung jumlah karakter khusus lainnya dalam URL selain tanda =, ?, dan &.

    Proses Ekstraksi: Fungsi count_special_chars(x) menghitung jumlah karakter khusus (seperti #, %, @, dll.) yang ada dalam URL pada kolom URL.

16. **SpecialCharRatioInURL**

    Kolom ini mengukur rasio karakter khusus terhadap total karakter dalam URL.

    Proses Ekstraksi: Fungsi calc_spacial_char_ratio(x) menghitung rasio antara jumlah karakter khusus dengan panjang total URL, yang dapat dihitung berdasarkan kolom URL.

3. Feature Engineering

Berikut merupakan kolom-kolom yang mengalami *feature engineering*.

1. **HasTitle**

   Penjelasan: Kolom ini menunjukkan apakah sebuah halaman web memiliki judul (Title) atau tidak.

   Proses Ekstraksi: Fungsi hastitle(title) mengembalikan nilai 1 jika

halaman memiliki judul yang valid (kolom Title tidak kosong), dan 0 jika halaman tidak memiliki judul (kolom Title kosong atau NaN).

2. **WebComplexity**

   Penjelasan: Kolom ini mengukur kompleksitas dari sebuah halaman web berdasarkan jumlah elemen-elemen seperti gambar, CSS, dan JavaScript.

   Proses Ekstraksi: Fungsi web_complexity(df) menjumlahkan jumlah gambar (NoOfImage), CSS (NoOfCSS), dan JavaScript (NoOfJS) di halaman web. Setiap nilai yang kosong diisi dengan 0, kemudian nilai totalnya dihitung.

3. **RefLinksCount**

   Penjelasan: Kolom ini menghitung jumlah total referensi/link yang ada pada halaman web, termasuk link internal (self-reference), link kosong, dan link eksternal.

   Proses Ekstraksi: Fungsi ref_links_count(df) menghitung total link berdasarkan tiga kategori: NoOfSelfRef (link internal), NoOfEmptyRef (link kosong), dan NoOfExternalRef (link eksternal). Setiap kolom yang kosong diisi dengan 0, kemudian jumlah totalnya dihitung.

4. **LinkMatchScore**

   Penjelasan: Kolom ini mengukur kecocokan antara judul domain dan judul URL halaman web.

   Proses Ekstraksi: Fungsi link_match_score(df) menghitung skor kecocokan berdasarkan dua parameter: DomainTitleMatchScore dan URLTitleMatchScore. Jika nilai kolom-kolom tersebut kosong, nilai diisi dengan 0. Skor rata-rata dihitung antara keduanya untuk menghasilkan nilai akhir.

5. **HasFinanceTransaction**

   Penjelasan: Kolom ini menunjukkan apakah halaman web memiliki transaksi keuangan (misalnya, transaksi bank, pembayaran, atau transaksi crypto).

   Proses Ekstraksi: Fungsi financial_transaction(df) memeriksa tiga kolom yang berhubungan dengan transaksi keuangan: Bank, Pay, dan Crypto.

Jika nilai dalam kolom tersebut kosong, diisi dengan 0. Kemudian, fungsi ini mengembalikan nilai maksimum dari ketiga kolom tersebut untuk menunjukkan apakah ada transaksi keuangan yang terjadi di halaman tersebut (nilai 1 jika ada transaksi).

4. **Features Manipulation**

Berikut merupakan kolom-kolom yang mengalami *features manipulation*.

1. **TLDMajorityLegit**
2. **Penjelasan**: Kolom ini menunjukkan apakah Top-Level Domain (TLD) dalam URL termasuk dalam daftar TLD yang sah atau tidak.

   Proses Ekstraksi: Fungsi tld_legit(tld, tld_legit) akan memeriksa apakah TLD yang diambil dari URL (kolom TLD) termasuk dalam daftar TLD yang sah (legit_tlds). Jika TLD valid, kolom ini akan bernilai 1; jika tidak valid, kolom ini akan bernilai 0. TLD yang sah harus terdapat dalam daftar legit_tlds, yang sebelumnya sudah didefinisikan.

3. **HasObfuscation**

   Penjelasan: Kolom ini menunjukkan apakah URL mengandung teknik obfuscation atau tidak. Obfuscation adalah metode yang digunakan untuk menyembunyikan tujuan URL dengan cara-cara tertentu, seperti menggunakan karakter yang dienkode atau struktur URL yang kompleks.

   Proses Ekstraksi: Fungsi has_obfuscation(url, legit_tld) memeriksa beberapa faktor untuk menentukan apakah sebuah URL menggunakan obfuscation. Beberapa pemeriksaan yang dilakukan dalam fungsi ini antara lain:

   - Apakah URL terdekripsi dengan urllib.parse.unquote() (menandakan ada encoding yang digunakan).
   - Apakah ada perubahan setelah normalisasi menggunakan unidecode(), yang menunjukkan penggunaan karakter non-standar.
   - Apakah panjang URL lebih dari 100 karakter.
   - Apakah URL mengandung string panjang yang terdiri dari karakter hexadecimal (misalnya hash MD5 atau SHA1).

- Apakah jumlah subdomain dalam URL lebih dari tiga (menandakan struktur URL yang mungkin lebih kompleks).
- Apakah TLD dalam URL valid berdasarkan daftar legit_tlds (list TLD yang mayoritas merupakan *legit)*.

Jika salah satu kondisi tersebut terpenuhi, maka URL dianggap menggunakan obfuscation dan kolom ini akan bernilai 1, sebaliknya 0 jika tidak ada obfuscation.

5. **Missing Values lainnya**

Untuk kolom-kolom lainnya, akan dilakukan imputer biasa. Numerikal akan diisi dengan rata-rata dan kategorikal akan diisi dengan *most_frequent.*

2.2 Implementasi KNN

2.2.1 Scratch

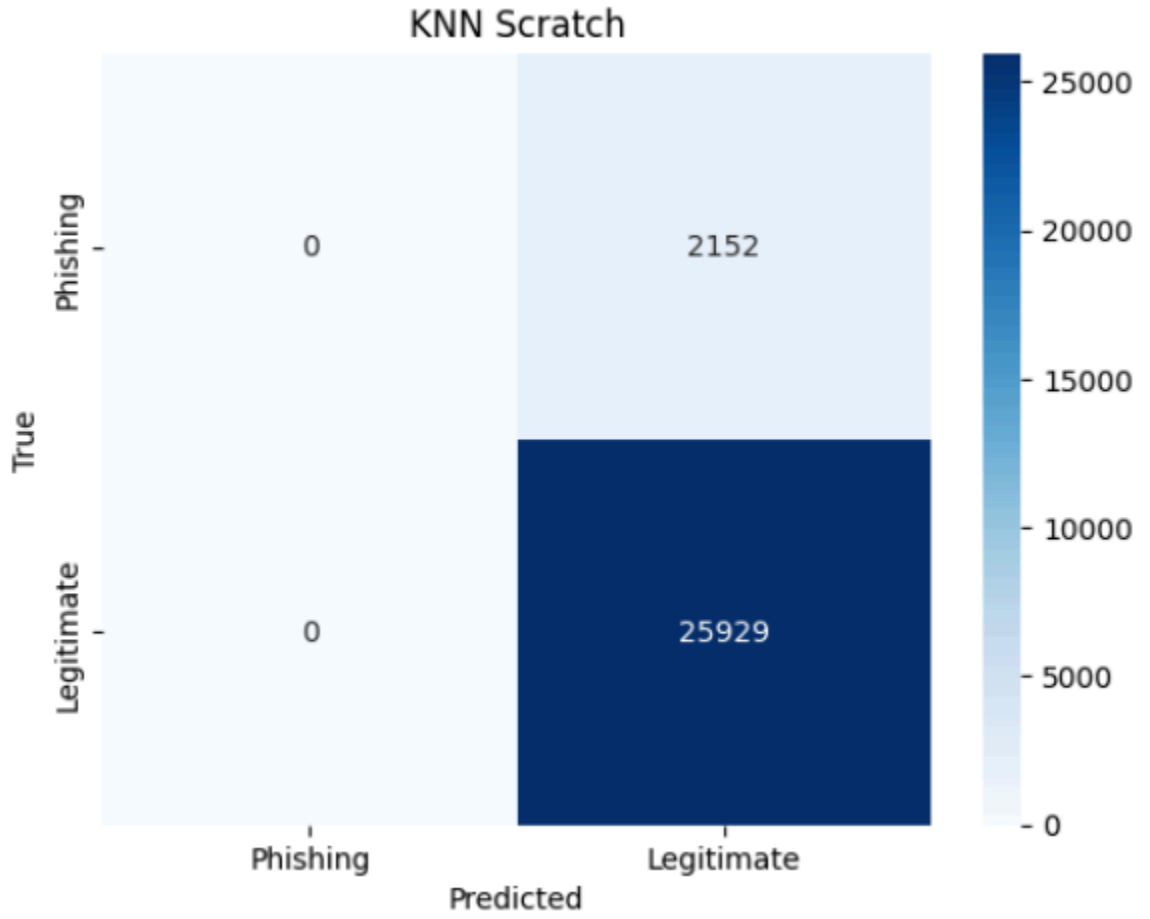
Pada Inisialisasi parameter “k” menentukan jumlah tetangga yang dipertimbangkan untuk voting mayoritas. Model menyimpan data latih dan memanfaatkan struktur data KDTree untuk mempercepat pencarian tetangga. Model menghitung jarak antara data baru dan data latih, menentukan tetangga terdekat, dan melakukan voting untuk menentukan kelas. Setelah itu dihitung akurasi dan klasifikasi skor. Dari cara ini didapatkan hasil:

```
Akurasi: 0.9234
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2152
           1       0.92      1.00      0.96     25929

    accuracy                           0.92     28081
   macro avg       0.46      0.50      0.48     28081
weighted avg       0.85      0.92      0.89     28081
```

Gambar 2.2.1.1 Hasil KNN Scratch



Gambar 2.2.1.2 Confusion matrix KNN Scratch

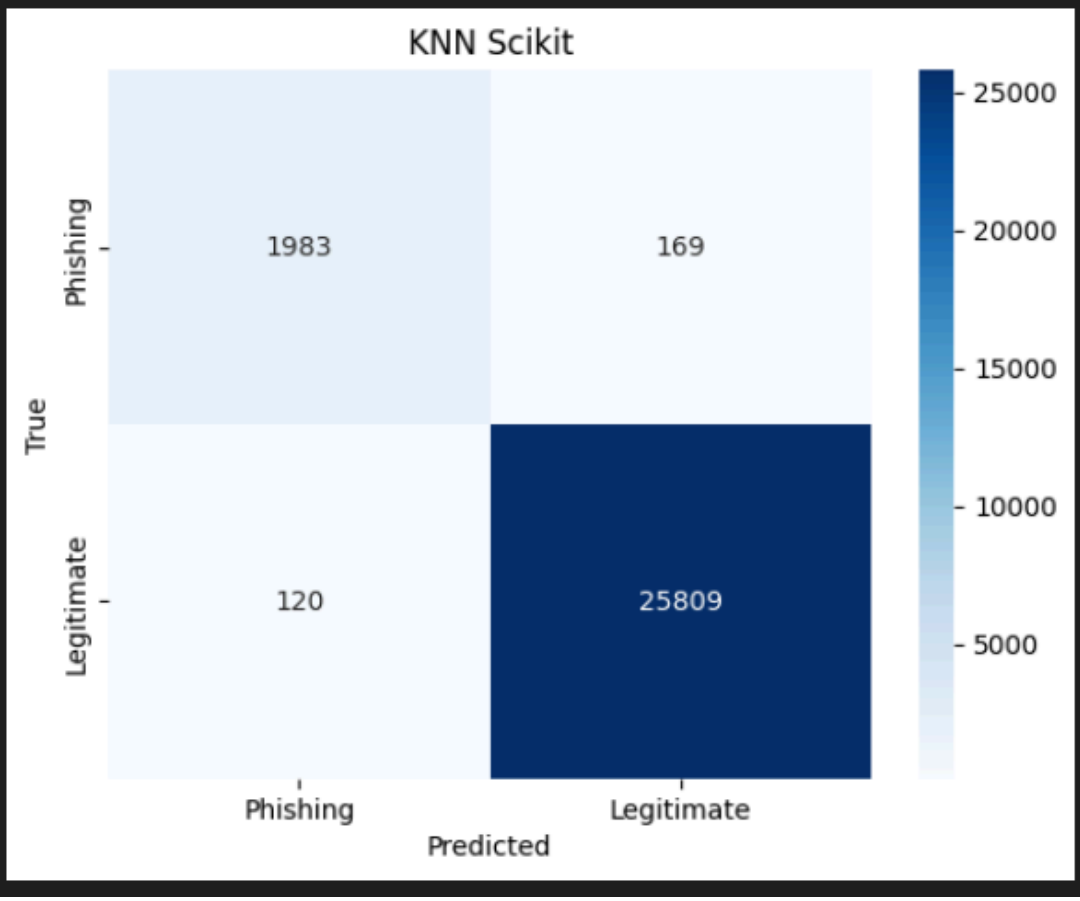
2.2.2 Scikit-learn

Pada implementasi KNN menggunakan library scikit-learn dilakukan import *KNeighborsClassifier* dan diinisiasi dengan jumlah tetangga yang ditentukan(5). Setelah itu model KNN dilatih dengan data yang sudah diskalakan dan dilabelkan. Dilakukan prediksi data uji untuk menghasilkan prediksi label target. Setelah itu dilakukan penghitungan akurasi model dengan membandingkan prediksi dengan label sebenarnya. Didapat:

```
Akurasi: 0.9897
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93      2152
           1       0.99      1.00      0.99     25929

    accuracy                           0.99     28081
   macro avg       0.97      0.96      0.96     28081
weighted avg       0.99      0.99      0.99     28081
```



Gambar 2.2.2.1 Hasil Scikit-learn KNN

2.3 Implementasi Naive-Bayes

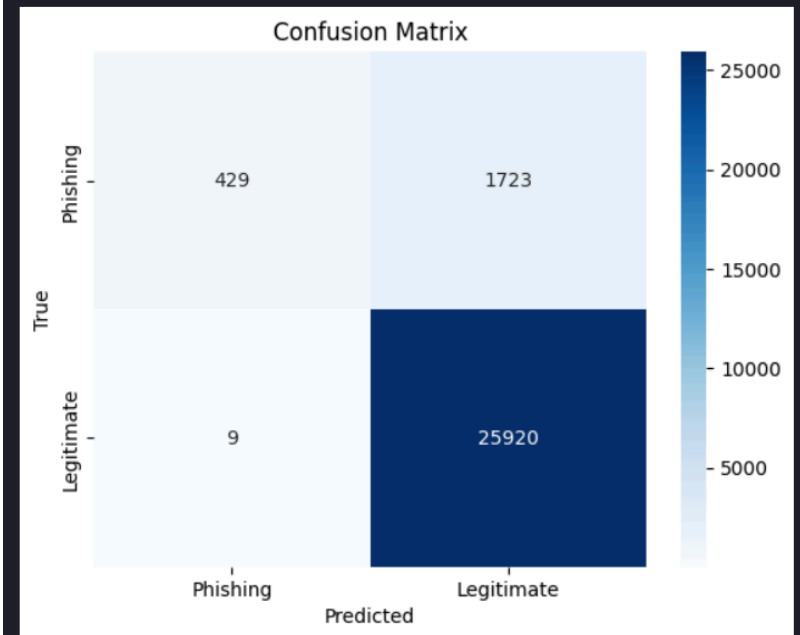
2.3.1 Scratch

Pada implementasi Gaussian Naive-Bayes secara *scratch,* Model memiliki metode *fit* yang berguna untuk melatih model dengan data pelatihan. Dalam proses pelatihan, model menghitung jumlah baris dan fitur dari X, serta mengidentifikasi kelas unik dalam y. Untuk setiap kelas unik, model menghitung rata-rata dan varians dari setiap fitur, serta probabilitas awal (prior) berdasarkan proporsi data yang termasuk dalam kelas tersebut. Setelah dilatih, model dapat melakukan prediksi dengan menghitung probabilitas posterior untuk setiap kelas dengan menjumlahkan logaritma dari prior dan fungsi densitas probabilitas (PDF) untuk setiap fitur. Kelas dengan probabilitas posterior tertinggi dipilih sebagai prediksi. Fungsi densitas probabilitas dihitung dengan memanfaatkan rata-rata dan varians yang telah dihitung selama pelatihan. Dengan asumsi bahwa fitur-fitur bersifat independen dan mengikuti distribusi Gaussian, model ini dapat melakukan klasifikasi dengan efisien. Dari implementasi ini, didapatkan hasil:

```
Akurasi: 0.9383
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.20      0.33      2152
           1       0.94      1.00      0.97     25929

    accuracy                           0.94     28081
   macro avg       0.96      0.60      0.65     28081
weighted avg       0.94      0.94      0.92     28081
```



Gambar 2.3.1.1 Hasil Naive-Bayes Scratch

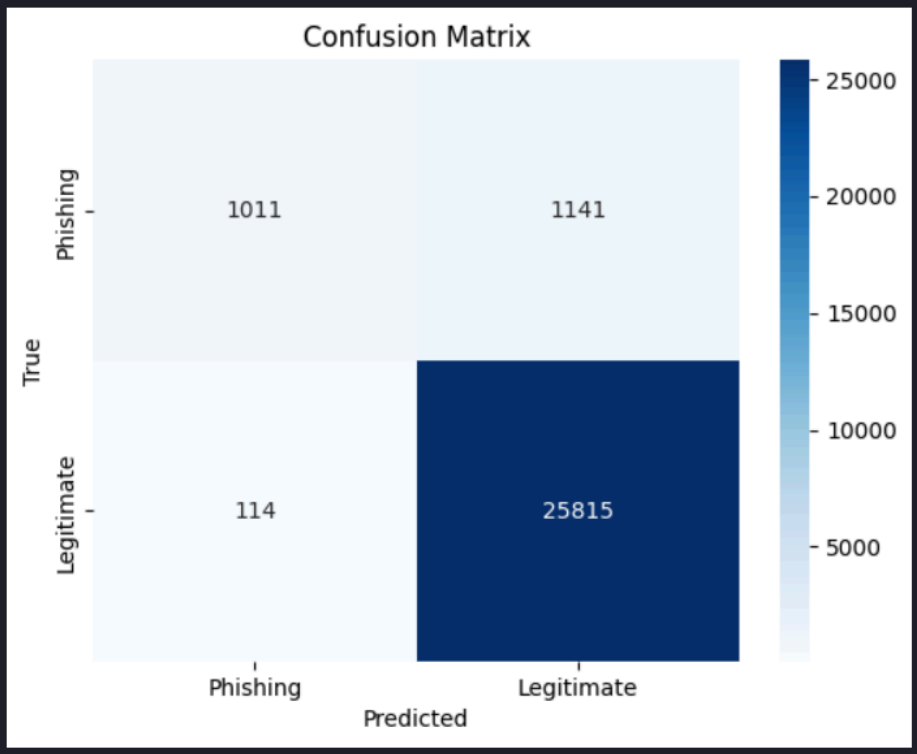
2.3.2 Scikit-learn

Pada implementasi Naive-Bayes menggunakan library scikit-learn dilakukan import salah satu algoritma Naive-Bayes yaitu Gaussian Naive-Bayes (GaussianNB). Setelah itu, model GNB dilatih dengan data yang sudah diskalakan dan dilabelkan. Dilakukan prediksi data uji untuk menghasilkan prediksi label target. Setelah itu dilakukan penghitungan akurasi model dengan membandingkan prediksi dengan label sebenarnya. Didapat:

```
Akurasi: 0.9553
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.47      0.62      2152
           1       0.96      1.00      0.98     25929

    accuracy                           0.96     28081
   macro avg       0.93      0.73      0.80     28081
weighted avg       0.95      0.96      0.95     28081
```



Gambar 2.3.2.1 Hasil Naive-Bayes Scikit-learn

3. Perbandingan

Berikut merupakan variabel yang digunakan sebagai evaluator:

1. Precision:
2. Recall:
3. F1-Score:

Berikut merupakan perbandingan hasil tiap model.

Tabel 3.1 Perbandingan evaluasi tiap model

Model	Akurasi	Precision		Recall		F1-Score	
		0	1	0	1	0	1
KNN Scratch	0.9899	0.95	0.99	0.92	1.00	0.93	0.99
KNN Sci-kit	0.9897	0.94	0.99	0.92	1.00	0.93	0.99
Naive-Bayes Scratch	0.9383	0.98	0.94	0.20	1.00	0.33	0.97
Naive-Bayes Sci-kit	0.9595	0.92	0.96	0.52	1.00	0.66	0.98

Dari hasil didapatkan bahwa akurasi dari Sci-kit lebih tinggi daripada Scratch pada kedua cara KNN dan GNB. ci-kit Learn juga secara otomatis menangani masalah pada data, seperti skala, missing values, dan outliers, serta memanfaatkan pustaka seperti NumPy untuk operasi matematis presisi tinggi. Selain itu, nilai parameter defaultnya telah diuji untuk menghasilkan performa optimal, sementara implementasi manual sering kali lebih sederhana dan rentan terhadap kesalahan akibat kurangnya

optimasi dan handling data. Akurasi tertinggi didapatkan dari KNN Scratch. Jika dilihat dari perbandingan antara KNN dan Naive-Bayes, KNN lebih unggul dibandingkan Naive Bayes hal ini karena KNN lebih fleksibel dalam mengklasifikasikan data dengan pola kompleks tanpa asumsi distribusi tertentu. Sebaliknya, Naive Bayes bergantung pada asumsi bahwa fitur bersifat independen, yang mungkin tidak sesuai dengan dataset phishing URL ini. Scikit-learn mengungguli implementasi Scratch karena menggunakan algoritma yang sudah diuji dan dikembangkan dalam waktu yang lama.

Log Act

Tabel Log Act

Nama	NIM	Kontribusi
Jeremy Deandito	18222112	Naive Bayes Scikit-Learn
Nathaniel Liady	18222114	Naive Bayes Scratch
Gabriel Marcellino	18222115	Data Cleaning, Preprocessing dan KNN Scratch
Nicolas Jeremy	18222135	KNN Scikit-Learn

Referensi

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21-27. https://doi.org/10.1109/TIT.1967.1053964

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning, 29*(2), 103-130. https://doi.org/10.1023/A:1007413511361

Scikit-learn developers. (n.d.). *Nearest Neighbors*. Scikit-learn. Diakses 22 Desember, 2024, from https://scikit-learn.org/1.5/modules/neighbors.html

Scikit-learn developers. (n.d.). *Naive Bayes*. Scikit-learn. Diakses 22 Desember, 2024, from https://scikit-learn.org/1.5/modules/naive_bayes.html