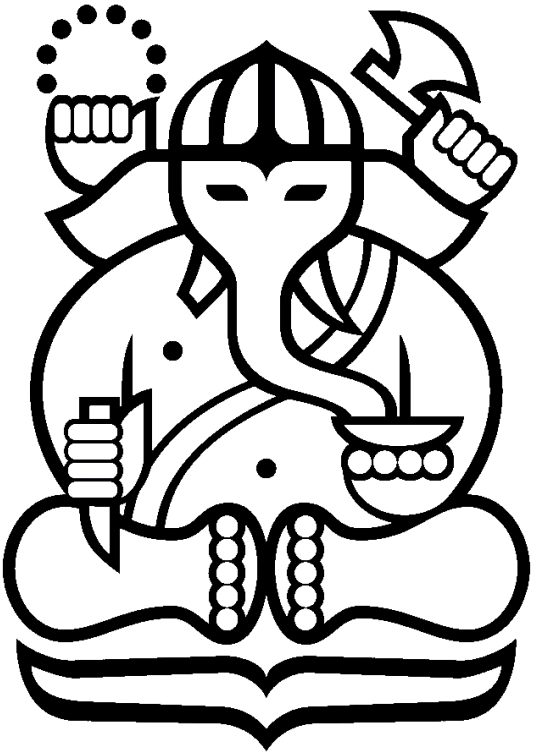
TUGAS BESAR 2 IF3070 DASAR INTELEGENSI ARTIFISIAL
IMPLEMENTASI ALGORITMA PEMBELAJARAN MESIN



DISUSUN OLEH KELOMPOK 12 :

ANTHONY BRYANT GOUW	18222033
CHRISTOPHER RICHARD CHANDRA	18222057
RICHIE LEONARDO	18222071
WISYENDRA LUNARMALAM	18222095

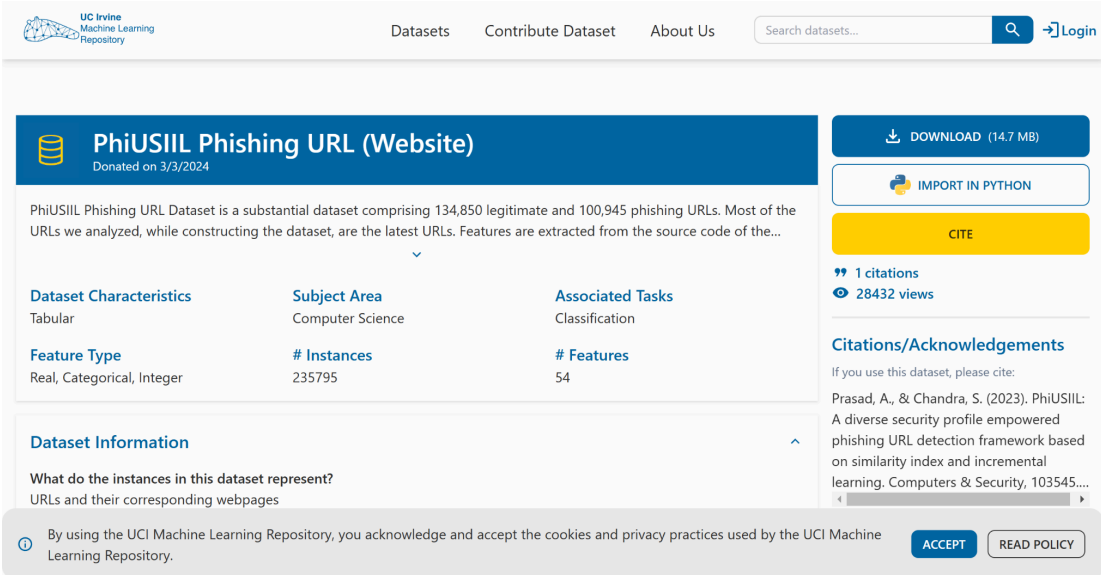
PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

I. PENDAHULUAN

Pada Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial, implementasi algoritma pembelajaran mesin (Machine Learning), KKN dan Gaussian Naive-Bayes, dilakukan pada dataset PhiUSIIL Phishing URL Dataset. PhiUSIIL Phishing URL Dataset merupakan sebuah dataset dengan ukuran cukup besar yang terdiri dari 134.850 URL asli (Legitimate) dan 100.945 URL palsu (Phishing). Analisa yang dilakukan pada PhiUSIIL Phishing URL merupakan URL terbaru yang mana sebagian besar features diekstraksi dari source code webpage dan URL. Dataset yang digunakan tim untuk implementasi algoritma Machine Learning dapat diakses pada link Kaggle berikut https://www.kaggle.com/competitions/tugas-besar-2-if-3070/ .



II. IMPLEMENTASI KNN

Algoritma KNN atau K-Nearest Neighbor merupakan salah satu jenis algoritma supervised learning yang bersifat non-parametric (tidak membuat asumsi terkait distribusi data yang mendasari atau tidak ada estimasi parameter tetap yang ditemukan di dalam model parametris) dan lazy learner (minim atau tidak adanya fase training model). Algoritma ini secara umum memiliki tujuan utama untuk menemukan persamaan antara data lama dan baru dan memasukkan hasil data baru ke dalam kategori yang paling cocok atau mirip yang telah ada sebelumnya. Adapun hasil implementasi algoritma K-Nearest Neighbor bekerja dengan alur sebagai berikut:

(1) Pemilihan nilai banyaknya K-Neighbor.

(2) Perhitungan jarak dari jumlah K-Neighbor (beberapa metode perhitungan yang dapat digunakan mencakup Euclidean Distance, Hamming Distance, atau Manhattan Distance).

(3) Pengambilan K-Neighbor yang paling dekat berdasarkan hasil perhitungan jarak.

(4) Perhitungan jumlah titik data untuk setiap kategori, antar K-Neighbor yang telah diambil pada langkah sebelumnya.

(5) Penetapan titik data baru ke dalam kategori atau kelas yang jumlah tetangganya paling banyak atau maksimum.

Model from Scratch

```
class KNN:
    """
    K-Nearest Neighbors (KNN) classification algorithm

    Parameters:
    -----------
    n_neighbors : int, optional (default=5)
        Number of neighbors to use in the majority vote.

    Methods:
    --------
    fit(X_train, y_train):
        Stores the values of X_train and y_train.

    predict(X):
        Predicts the class labels for each example in X.

    """
    def __init__(self, k=3, algorithm = "Euclidean", r = 1):
        self.k = k
```

```
        self.algorithm = algorithm
        self.r = r

    def fit(self, X, y):
        """
        Only stores the values of X_train and y_train.

        Parameters:
        -----------
               X_train  :  numpy.ndarray,  shape  (n_samples,
n_features)
            The training dataset.

        y_train : numpy.ndarray, shape (n_samples,)
            The target labels.
        """
        self.X_train = X
        self.y_train = y.astype(int)

    def euclidean_distance(self, x1, x2):
        """
          Calculate the Euclidean distance between two data
points.

        Parameters:
        -----------
        x1 : numpy.ndarray, shape (n_features,)
            A data point in the dataset.

        x2 : numpy.ndarray, shape (n_features,)
            A data point in the dataset.

        Returns:
        --------
        distance : float
            The Euclidean distance between x1 and x2.
        """
        return np.linalg.norm(x1-x2, axis=1)

    def manhattan_distance(self, x1, x2):
        """
          Calculate the Manhattan distance between two data
```

```
points.

        Parameters:
        -----------
        x1 : numpy.ndarray, shape (n_features,)
            A data point in the dataset.

        x2 : numpy.ndarray, shape (n_features,)
            A data point in the dataset.

        Returns:
        --------
        distance : float
            The Manhattan distance between x1 and x2.
        """
        return np.linalg.norm(x1-x2, ord=1, axis=1)

    def minowski_distance(self, x1, x2, r):
        """
           Calculate the Minowski distance between two data
points.

        Parameters:
        -----------
        x1 : numpy.ndarray, shape (n_features,)
            A data point in the dataset.

        x2 : numpy.ndarray, shape (n_features,)
            A data point in the dataset.

        Returns:
        --------
        distance : float
            The Minowski distance between x1 and x2.
        """
        return np.power(np.sum(np.abs(x1 - x2) ** r), 1 / r)

    def _predict(self, x):
        """
          Predicts the class label for a single row of data
only.
        Parameters:
```

```
        -----------
        x : numpy.ndarray, shape (n_features,)
            A data point in the test dataset.
        Returns:
        --------
        most_closest_label : int
            The predicted class label for x.
        """
        if(algorithm.lower() == "minowski"):
            distances = minowski_distance(self.X_train, x, r)
        else if(algorithm.lower() == "euclidean"):
            distances = euclidean_distance(self.X_train, x)
        else :
            distances = manhattan_distance(self.X_train, x)

        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = self.y_train[k_indices]
                                        most_closest_label    =
np.bincount(k_nearest_labels).argmax()
        return most_closest_label

    def predict(self, X):
        """
          Using Parralel predicts the class labels for each
example in X.

        Parameters:
        -----------
        X : numpy.ndarray, shape (n_samples, n_features)
            The test dataset.

        Returns:
        --------
        predictions : numpy.ndarray, shape (n_samples,)
            The predicted class labels for each example in X.
        """
                                                    return
np.array(Parallel(n_jobs=-1)(delayed(self._predict)(x)  for  x
in X))  # Use all CPU cores

def compute_accuracy(y_true, y_pred):
    """
```

```
    Computes the accuracy of a classification model.

    Parameters:
     y_true (numpy array): A numpy array of true labels for
each data point.
     y_pred (numpy array): A numpy array of predicted labels
for each data point.

    Returns:
       float: The accuracy of the model, expressed as a
percentage.
    """
    y_true = y_true.flatten()
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)

X_train                                                    =
processed_df[used_feature].astype('float64').to_numpy()
y_train = processed_df['label'].astype('float64').to_numpy()

model = KNN()
model.fit(X_train, y_train)

X_test                                                     =
val_processed_df[used_feature].astype('float64').to_numpy()
y_test = val_data['label'].astype('float64').to_numpy()

predictions = model.predict(X_test)

predictions

accuracy = compute_accuracy(y_test, predictions)
print(f" our model got accuracy score of : {accuracy}")
```

Model from SKLearn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```
knn = KNeighborsClassifier(n_neighbors=5)
# Fit the model
X_train_knn = processed_df[used_feature].astype('float64')
y_train_knn = train_data['label'].astype('float64')
X_test_knn = val_processed_df[used_feature].astype('float64')
y_test_knn = val_data['label'].astype('float64')

knn.fit(X_train_knn, y_train_knn)

# Make predictions
y_pred = knn.predict(X_test_knn)
# Evaluate accuracy
print(f'Accuracy: {accuracy_score(y_test_knn, y_pred)}')
```

III. IMPLEMENTASI NAIVE-BAYES

Algoritma Naive-Bayes atau biasa disebut Naive-Bayes Classifier merupakan salah satu jenis algoritma supervised learning probabilistik yang memiliki tujuan pada penyelesaian proses klasifikasi, salah satunya seperti klasifikasi teks. Algoritma ini dibuat berlandaskan pada Teorema Bayes. Secara sederhana, teorema ini akan melakukan prediksi probabilitas di masa depan berdasarkan pengalaman atau probabilitas kejadian sebelumnya, yang disertai dengan bukti baru. Berdasarkan jenis data yang akan digunakan, terdapat 3 tipe algoritma Naive-Bayes yang dapat digunakan mencakup Gaussian Naive-Bayes, Multinomial Naive-Bayes, dan Bernoulli Naive-Bayes. Adapun hasil implementasi algoritma Naive-Bayes (Gaussian Naive-Bayes) bekerja dengan alur sebagai berikut :

(1) Perhitungan frekuensi setiap atribut yang ada pada setiap kelas dan frekuensi setiap kelas.
(2) Perhitungan probabilitas dari atribut yang ada pada kelas tertentu.
(3) Perhitungan probabilitas dari setiap kelas yang ada.
(4) Hasil perhitungan setiap probabilitas yang membentuk model probabilitas sebagai bagian dari hasil pembelajaran. Model ini yang digunakan untuk memprediksi kelas atau label data baru di masa depan.
(5) Proses klasifikasi dimulai dengan perhitungan perkalian setiap probabilitas atribut pada data yang ditanyakan untuk setiap kelas yang mungkin.
(6) Hasil perkalian tersebut dikalikan dengan probabilitas dari setiap kelas.
(7) Pencarian kelas yang memiliki tingkat probabilitas tertinggi atau maksimum.

Model from Scratch

```
class GaussianNaiveBayes:
    """
    Gaussian Naive Bayes classification algorithm

    Methods:
    --------
    fit(X_train, y_train):
        Preparing data for the model based on X_train.

    pdf(x, mean, std):
        Calculate the probability density of the feature according to
the Gaussian distribution.

    predict(X):
        Predicts the class labels for each example in X.

    """
    def fit(self, X, y):
        """
        Preparing data for the model based on X_train.
```

```
        Parameters:
        -----------
        X_train : numpy.ndarray, shape (n_samples, n_features)
            The training dataset.

        y_train : numpy.ndarray, shape (n_samples,)
            The target labels.
        """
        classes, cls_counts = np.unique(y, return_counts=True)
        n_classes = len(classes)
        self.priors = cls_counts / len(y)

        self.X_cls_mean = np.array([np.mean(X[y == c], axis=0) for c
in range(n_classes)])
        self.X_stds = np.array([np.std(X[y == c], axis=0) for c in
range(n_classes)])

    def pdf(self, x, mean, std):
        """
        Calculate the probability density of the feature according to
the Gaussian distribution.

        Parameters:
        -----------
        X_train : numpy.ndarray, shape (n_samples, n_features)
            The training dataset.

        mean : numpy.ndarray, shape (n_columns,)
            The mean of each columns.

        std : numpy.ndarray, shape (n_columns,)
            The standard deviation of each columns.

        Returns:
        --------
        pdf : float
            The probability density.
        """
        return (1 / (np.sqrt(2 * np.pi) * std)) * np.exp(-0.5 * ((x -
mean) / std) ** 2)

    def save_model(self, filename):
        """
        Saves the trained KNN model to a file.

        Parameters:
        -----------
        filename : str
```

```
            The name of the file to save the model.
        """
        with open(filename, 'wb') as file:
            pickle.dump(self, file)
        print(f"Model saved to {filename}")

    @staticmethod
    def load_model(filename):
        """
        Loads a trained KNN model from a file.

        Parameters:
        -----------
        filename : str
            The name of the file from which to load the model.

        Returns:
        --------
        model : KNN
            The loaded KNN model.
        """
        with open(filename, 'rb') as file:
            model = pickle.load(file)
        print(f"Model loaded from {filename}")

    def predict(self, X):
        """
        Predicts the class labels for each example in X.

        Parameters:
        -----------
        X : numpy.ndarray, shape (n_samples, n_features)
            The test dataset.

        Returns:
        --------
        predictions : numpy.ndarray, shape (n_samples,)
            The predicted class labels for each example in X.
        """

        pdfs = np.array([self.pdf(x, self.X_cls_mean, self.X_stds)
for x in X])
        posteriors = self.priors * np.prod(pdfs, axis=2)   # shorten
Bayes formula

        return np.argmax(posteriors, axis=1)
```

Model from Sklearn

```
model_NB = GaussianNB()  # Use GaussianNB for continuous data
model_NB.fit(X_train_knn, y_train_knn)  # Train the model
y_pred_NB = model_NB.predict(X_test_knn)
```

IV. PROSES CLEANING DAN PREPROCESSING

A. Handling Missing Data (Recovery)

Semua feature pada dataset yang memiliki missing value >30% membuat modelling pada data menjadi tidak akurat. Untuk itu, perlu dilakukan cleaning seperti menangani data yang hilang atau null. Dari hasil analisis pada tugas sebelumnya, didapat *insight* bahwa banyak feature yang saling terhubung atau berkorelasi satu sama lain. Oleh karena itu, akan dicoba imputation dengan domain knowledge yang kami miliki kemudian menghapus data null yang tidak dapat direcovery.

Berikut adalah implementasi yang dilakukan untuk beberapa feature pada dataset :

- Domain

```
def recover_domain(row):
    if pd.isna(row['Domain']) and not
pd.isna(row['URL']):
        return urlparse(row['URL']).netloc
    else :
        return row['Domain']
```

- Domain Length

```
def recover_domain_length(row):
    if(not pd.isna(row['Domain']) ):
        return len(row['Domain'])
    return row['DomainLength']
```

- IsHTTPS

```
def recover_ishttps(row):
    if pd.isna(row['IsHTTPS']) and not
pd.isna(row['URL']):
        parsed_url = urlparse(row['URL'])
        return parsed_url.scheme == "https"
    else:
        return row['IsHTTPS']
```

- URL

```
def recover_url(row):
```

```
    if not pd.isna(row['URL']):
        return row['URL']
    if not pd.isna(row['Domain']) or
pd.isna(row['TLD']):
        return None
    protocol = "https://" if row['IsHTTPS'] == 1 else
"http://"
    return f"{protocol}{row['Domain']}.{row['TLD']}"
```

- HasTitle

```
def recover_url(row):
    if not pd.isna(row['URL']):
        return row['URL']
    if not pd.isna(row['Domain']) or
pd.isna(row['TLD']):
        return None
    protocol = "https://" if row['IsHTTPS'] == 1 else
"http://"
    return f"{protocol}{row['Domain']}.{row['TLD']}"
```

- DegitRationInURL

```
def recover_degit_ratio(row):
    if pd.isna(row['DegitRatioInURL']):
        if pd.notna(row['Domain']):
            # Calculate digit ratio for 'Domain'
            return count_degit_ratio(row['Domain'])
        else:
            # Return NaN if both DegitRatioInURL and
Domain are NaN
            return np.nan
    else:
        return np.float64(row['DegitRatioInURL'])
```

Dari hasil recovery tersebut, data null yang terdapat pada feature tersebut dapat berkurang.

B. Custom Transformer (Imputation Lanjutan)

Selain proses recovery yang telah dilakukan pada bagian A, dilakukan penanganan kembali terhadap missing values (data yang hilang) sisanya dengan pendekatan transformer berbasis scikit-learn dengan metode mean & most frequent data. Transformer ini didesain sedemikian rupa sehingga dapat digunakan pada pipeline

sklearn untuk pre-processing data. Mean & most frequent data merupakan metode yang digunakan karena hasil tes yang jauh lebih baik dibandingkan metode yang dicoba lainnya yaitu KNN (K-Nearest Neighbors).

```
# Create a pipeline
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import FunctionTransformer

columns_categorical = ['HasTitle', 'HasSocialNet',
'IsHTTPS','HasCopyrightInfo', 'HasDescription', 'IsResponsive']
columns_numerical = ['DegitRatioInURL',
'DomainLength','SpacialCharRatioInURL','DomainTitleMatchScore',
'CharContinuationRate', 'NoOfOtherSpecialCharsInURL']

feature_recovery = FeatureRecovery(
    column_func = [('Domain',recover_domain), ('DomainLength',
recover_domain_length),
     ('IsHTTPS',recover_ishttps),
('URL',recover_url),('DegitRatioInURL',recover_degit_ratio),
      ('SpacialCharRatioInURL',recover_specialCharRatio)]
)

# Create the custom imputer instance
custom_imputer_num = RandomizedSimpleImputer(strategy='median')
custom_imputer_cat = RandomizedSimpleImputer(strategy='most_frequent')

# Define the column transformer with RandomImputer for specific
columns
column_transformer = ColumnTransformer(
    transformers=[
        ('random_imputer', RandomImputer(),columns_categorical ),  #
Apply random imputer to these columns
        ('default_imputer', SimpleImputer(strategy='mean'),
columns_numerical)  # Apply default mean imputer to other column
    ],
     remainder='passthrough'
)

# Create the pipeline with ColumnTransformer
pipeline = Pipeline(steps=[
    ('feature_recovery', feature_recovery),
    ('transformer',column_transformer )
```

```
])
```

C. Data Preprocessing

Setelah melakukan data cleaning, tahap selanjutnya adalah mempersiapkan data yang ingin digunakan untuk model *machine learning*. Untuk step ini dilakukan *feature selection* yaitu memilih feature yang ingin digunakan dalam pembuatan model. Dikarenakan adanya beberapa data yang berkorelasi tinggi dan *feature* yang sangat banyak (54 features) akan membuat hasil *training* menjadi jelek.

Dari semua feature kami memilih 8 *features* yang akan digunakan untuk membuat model *machine learning*, yaitu :

- **HasTitle :** Untuk mengetahui apakah satu domain memiliki title atau tidak;
- **HasSocialNet :** Mengetahui apakah suatu website memiliki link yang terhubung pada aplikasi social;
- **IsHTTPS :** Mengetahui apakah suatu website memiliki protokol HTTPS atau tidak;
- **HasCopyrightInfo :** Mengetahui apakah suatu website memiliki suatu informasi terkait *Copyright;*
- **DegitRatioInURL :** Merupakan ratio atau perbandingan dari digit dan karakter keseluruhan pada suatu URL;
- **Domain Length :** Merupakan panjang domain yang digunakan suatu URL;
- **SpecialCharRatioInURL :** Ratio atau perbandingan dari banyak character spesial pada suatu URL (#, @, &, =, ?, dsb);

Semua feature tersebut telah dibersihkan dari data null dan siap untuk digunakan pada pemodelan.

Dilakukan juga *feature engineering* berupa penambahan feature ‘Name’ pada features yang akan digunakan. Berikut adalah implementasi dari penambahan *feature* ‘Name’ :

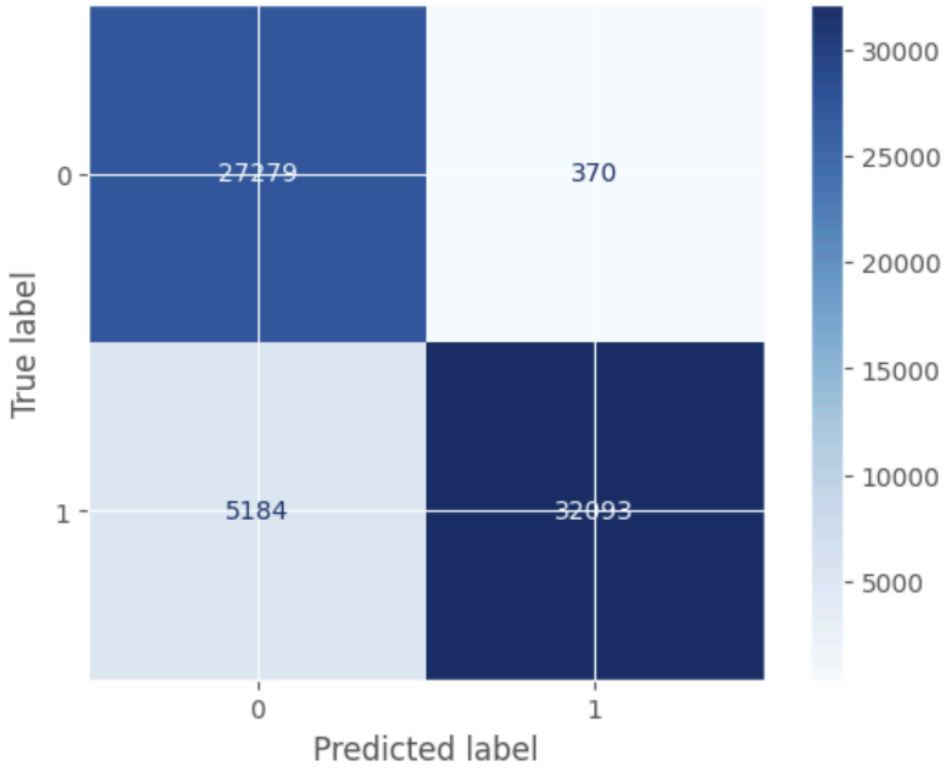
```
def generate_website_name(row):
    extracted = tldextract.extract(row['Domain'])
    return extracted.domain

train_cleaned['Name'] = train_cleaned.apply(generate_website_name,
axis = 1)
```

V. PERBANDINGAN HASIL PREDIKSI DAN PEROLEH

Setelah membuat model, hasil dari model yang dibuat akan dibandingkan dengan hasil yang didapat dengan menggunakan library KNN yang disediakan oleh Scikit-Learn salah satu library paling populer untuk melakukan modelling machine learning. Berikut adalah hasil yang didapat :

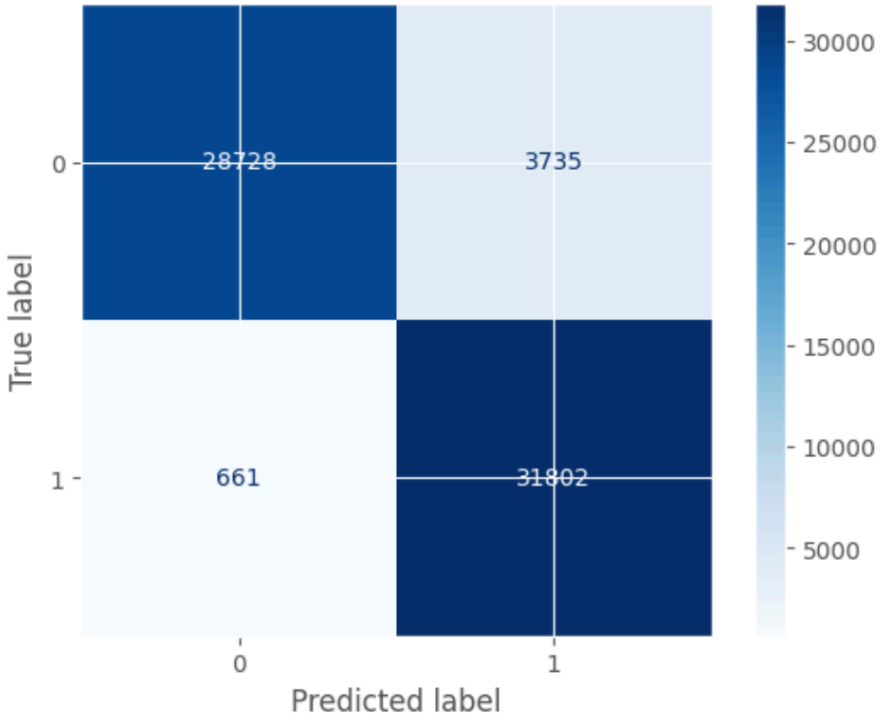
A.KNN

Hasil modelling yang kami buat mendapatkan hasil akurasi sebesar 0.91, sedangkan hasil dari library mendapatkan akurasi sebesar 0.94. Hal tersebut menunjukkan bahwa model yang dibuat library lebih bagus dibanding model yang dibuat. Namun, hasil dari model yang dibuat cukup baik dalam memprediksi URL phising dengan hasil dari confusion matrix dengan 49% data adalah true positive dan true negative adalah 42%. Dari hasil KNN yang dibuat terdapat false positive yaitu sebesar 0.5% dan false negative sebesar 7%. Hal tersebut mungkin dapat disebabkan karena adanya bias pada data train.



B. Gaussian Naive Bayes

Hasil modelling yang kami buat mendapatkan hasil akurasi sebesar 0.91446. Hasil dari model yang dibuat cukup baik dalam memprediksi URL phishing dengan hasil dari confusion matrix dengan 49% data adalah true positive dan true negative sebesar 44%. Dari hasil Gaussian Naive-Bayes yang dibuat terdapat false positive sebesar 6% dan false negative sebesar 1%.



VI. LOG ACTIVITY

Nama Anggota	NIM Anggota	Kontribusi
Anthony Bryant Gouw	18222033	• Pembuatan kode untuk Cleaning dan Pre-processing • Pembuatan Dokumen
Christopher Richard Chandra	18222057	• Pembuatan kode untuk implementasi model KNN dan Naive-Bayes • Pembuatan Dokumen
Richie Leonardo	18222071	• Pembuatan kode untuk Cleaning dan Pre-processing • Pembuatan Dokumen
Wisyendra Lunarmalam	18222095	• Pembuatan kode untuk implementasi model KNN dan Naive-Bayes • Pembuatan Dokumen

VII. REFERENSI

Algoritma Naive Bayes: Pengertian, Kegunaan Serta Teknik Meningkatkan Performanya. (n.d.). BINUS UNIVERSITY. Retrieved December 22, 2024, from https://binus.ac.id/bandung/2019/12/algoritma-naive-bayes/

Apa itu Pengklasifikasi Naïve Bayes? (n.d.). IBM. Retrieved December 22, 2024, from https://www.ibm.com/id-id/topics/naive-bayes

Mengenal K-nearest Neighbor dan Pengaplikasiannya. (2022, September 22). Blog - Algoritma Data Science School. Retrieved December 22, 2024, from https://blog.algorit.ma/k-nearest-neighbor/

Mengenal Naive Bayes Sebagai Salah Satu Algoritma Data Science. (2022, May 23). DQLab. Retrieved December 22, 2024, from https://dqlab.id/mengenal-naive-bayes-sebagai-salah-satu-algoritma-data-science

What is the k-nearest neighbors algorithm? (n.d.). IBM. Retrieved December 22, 2024, from https://www.ibm.com/think/topics/knn

Widrow, B., Hoff, T., & Jain, S. (2024, July 15). *K-Nearest Neighbor(KNN) Algorithm*. GeeksforGeeks. Retrieved December 22, 2024, from https://www.geeksforgeeks.org/k-nearest-neighbours/