Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin

Disusun oleh
Kelompok 32:

Matthew Lim (18222005)
Habib Akhmad Al Farisi (18222029)
Winata Tristan (18222061)
Ahmad Habibie Marjan (18222082)
Satria Wisnu Wibowo (18222087)

IF3070 Dasar Intelegensi Artifisial, Sistem dan Teknologi Informasi, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Jl. Ganesha No. 10, Bandung, Indonesia, 40132

Daftar Isi

Bab I
Pendahuluan

Pembelajaran mesin merupakan salah satu cabang dari kecerdasan buatan yang memungkinkan sistem untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit. Salah satu algoritma pembelajaran mesin, yaitu, Algoritma K-Nearest Neighbours (KNN) dan Algoritma Gaussian Naive-Bayes. Algoritma-algoritma pembelajaran mesin ini dapat diimplementasikan pada suatu dataset, misalnya PhiUSIIL Phishing URL Dataset. Algoritma-algoritma pembelajaran mesin ini pun dapat diimplementasikan *from scratch* dan menggunakan *scikit-learn*.

Algoritma KNN adalah algoritma yang mengklasifikasi data baru berdasarkan k data terdekat di sekitarnya, dengan cara menghitung jarak (biasanya Euclidean) ke semua data training dan mengambil kelas mayoritas dari k tetangga terdekat.

Algoritma Gaussian Naive-Bayes adalah algoritma klasifikasi yang menggunakan teorema Bayes dengan asumsi fitur berdistribusi normal dan saling independen. Menghitung probabilitas setiap kelas berdasarkan mean dan variance fitur-fiturnya.

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.

Pada tugas ini, akan diimplementasikan algoritma pembelajaran mesin K-Nearest Neighbours dan Gaussian Naive-Bayes pada suatu dataset, PhiUSIIL Phishing URL Dataset. Dengan rinciannya, implementasi kedua algoritma pembelajaran mesin tersebut *from scratch* dan menggunakan *scikit-learn*. Akan dilakukan data cleaning dan preprocessing. Terakhir, menunjukkan perbandingan hasil prediksi dari algoritma yang diimplementasikan dengan hasil yang didapatkan dengan menggunakan pustaka.

Bab II
Pembahasan

2.1 Implementasi Algoritma Pembelajaran Mesin *From Scratch*

Implementasi algoritma pembelajaran mesin *from scratch* merupakan suatu pendekatan membangun algoritma pembelajaran mesin dari nol dengan menulis semua komponen dan perhitungan matematis menggunakan bahasa pemrograman dasar, tanpa library khusus. Membutuhkan pemahaman matematika mendalam tapi memberikan kontrol penuh atas algoritma.

2.1.1 Implementasi Algoritma KNN from scratch

```
import numpy as np
from collections import Counter
import logging


class KNearestNeighbor:
   def __init__(self, verbose=True):
       self.best_k = None
       self.training_data = None
       self.training_labels = None
       self.verbose = verbose

   def fit(self, X_train, Y_train, max_k=None, num_of_k_values=10):
       self.training_data = np.array(X_train)
       self.training_labels = np.array(Y_train)
       n_splits = 5
       split_length = len(X_train) // n_splits
       if max_k is None:
           max_k = int(np.sqrt(len(self.training_data)))  # Changed to
use rows
       k_range = np.linspace(1, max_k, num_of_k_values, dtype=int)
       k_accuracies = {k: [] for k in k_range}

       if self.verbose:
           logging.info(f"Starting fit with max_k={max_k} and
num_of_k_values={num_of_k_values}")
           logging.info(f"Training data shape:
{self.training_data.shape}, Labels shape:
{self.training_labels.shape}")
```

```
        for fold in range(n_splits):
            val_start = fold * split_length
            val_end = (fold + 1) * split_length if fold != n_splits - 1
else len(X_train)
            X_val = self.training_data[val_start:val_end]
            Y_val = self.training_labels[val_start:val_end]
            X_fold_train = np.vstack((self.training_data[:val_start],
self.training_data[val_end:]))
            Y_fold_train = np.hstack((self.training_labels[:val_start],
self.training_labels[val_end:]))

            for k in k_range:
                predictions = self._predict_vectorized(X_val,
X_fold_train, Y_fold_train, k=k)
                accuracy = np.mean(predictions == Y_val)
                k_accuracies[k].append(accuracy)

                if self.verbose:
                    logging.info(f"Fold {fold + 1}/{n_splits}, k={k},
accuracy={accuracy:.4f}")

        avg_k_accuracies = {k: np.mean(acc) for k, acc in
k_accuracies.items()}
        best_k = max(avg_k_accuracies, key=avg_k_accuracies.get)
        self.best_k = (best_k, avg_k_accuracies[best_k])

        if self.verbose:
            logging.info(f"Best k: {self.best_k[0]} with average
training CV accuracy: {self.best_k[1]:.4f}")
        return self

    def _compute_distances(self, X_test, train_data):
        dists = np.sqrt(
            np.maximum(
                np.sum(X_test[:, np.newaxis, :] ** 2, axis=2) +
                np.sum(train_data[np.newaxis, :, :] ** 2, axis=2) -
                2 * np.dot(X_test, train_data.T),
                0
            )
        )
        if self.verbose:
            logging.info(f"Computed distances with shape:
{dists.shape}")
        return dists

    def _get_neighbours(self, dists, k):
```

```
        nearest_indices = np.argpartition(dists, k, axis=1)[:, :k]
        if self.verbose:
            logging.info(f"Identified {k} nearest neighbors for each
test point")
        return nearest_indices

    def _predict_vectorized(self, X_test, train_data=None,
train_labels=None, k=None):
        train_data = train_data if train_data is not None else
self.training_data
        train_labels = train_labels if train_labels is not None else
self.training_labels
        if self.best_k is not None and k is None:
            k = self.best_k[0]

        if self.verbose:
            logging.info(f"Predicting with k={k}, test data shape:
{X_test.shape}")

        dists = self._compute_distances(X_test, train_data)
        nearest_indices = self._get_neighbours(dists, k)
        nearest_labels = train_labels[nearest_indices]

        predictions = [Counter(neighbors).most_common(1)[0][0] for
neighbors in nearest_labels]
        if self.verbose:
            logging.info(f"Prediction complete for {len(X_test)} test
points")
        return np.array(predictions)

    def predict(self, X_test, train_data=None, train_labels=None,
k=None):
        return self._predict_vectorized(X_test, train_data,
train_labels, k)
```

Algoritma 2.1.1 Implementasi Algoritma KNN *from scratch*

Pada algoritma di atas, pendekatan utama adalah implementasi manual dari algoritma *k-Nearest Neighbors (k-NN)* dengan fokus pada optimasi melalui penggunaan operasi numpy yang efisien. Algoritma ini menghitung jarak Euclidean antara data uji dan data pelatihan, memilih k tetangga terdekat menggunakan *np.argpartition*, dan menentukan label mayoritas sebagai hasil prediksi. Proses validasi silang digunakan untuk mencari nilai k terbaik berdasarkan rata-rata akurasi melalui *5fold-CV*. Meskipun fleksibel untuk memahami k-NN secara mendalam, algoritma ini memiliki kompleksitas waktu

tinggi O(n x d) sehingga kurang optimal untuk dataset besar tanpa optimasi tambahan seperti *k-d tree* di library seperti *scikit-learn*

2.1.2 Implementasi Algoritma Gaussian Naive-Bayes *from scratch*

```
class NaiveBayes:
    def __init__(self, smoothing=1.0):
        self.smoothing = smoothing
        self.class_priors = {}
        self.feature_params = {}
        self.classes = None

    def calculate_mean_std(self, X):
        mean = np.mean(X, axis=0)
        std = np.std(X, axis=0)
        min_std = 1e-6
        std[std < min_std] = min_std
        return mean, std

    def gaussian_probability(self, x, mean, std):
        exponent = np.clip(-0.5 * ((x - mean) / std) ** 2, -500, 500)
        log_coef = -np.log(std) - 0.5 * np.log(2 * np.pi)
        return np.exp(log_coef + exponent)

    def fit(self, X, y):
        X = np.array(X, dtype=float)
        y = np.array(y)
        self.classes = np.unique(y)
        n_samples = len(y)

        for i in self.classes:
            class_mask = (y == i)
            class_samples = X[class_mask]
            self.class_priors[i] = (len(class_samples) + self.smoothing)
/ (n_samples + self.smoothing * len(self.classes))
            mean, std = self.calculate_mean_std(class_samples)
            self.feature_params[i] = {'mean': mean, 'std': std}

    def predict_proba(self, X):
        X = np.array(X, dtype=float)
        n_samples = len(X)
```

```
        probas = np.zeros((n_samples, len(self.classes)))

        for i, x in enumerate(X):
            class_probs = []
            for j, c in enumerate(self.classes):
                log_prob = np.log(self.class_priors[c])
                mean = self.feature_params[c]['mean']
                std = self.feature_params[c]['std']
                feature_probs = self.gaussian_probability(x, mean, std)
                log_prob += np.sum(np.log(feature_probs + 1e-10))
                probas[i, j] = log_prob
            max_log_prob = np.max(probas[i])
            probas[i] = np.exp(probas[i] - max_log_prob)
            probas[i] /= np.sum(probas[i])

        return probas

    def predict(self, X):
        probas = self.predict_proba(X)
        return self.classes[np.argmax(probas, axis=1)]

    def score(self, X, y):
        return np.mean(self.predict(X) == y)
```

Algoritma 2.1.2 Implementasi Algoritma Gaussian Naive-Bayes *from scratch*

Pada algoritma di atas, ditetapkan *smoothing* dengan nilai 1, hal ini untuk mengatasi probabilitas 0 yang tentu bila terjadi maka hasil yang diperoleh tidak akan akurat. Terdapat *method* calculate_mean_std yang berfungsi untuk menghitung rata-rata dan standar deviasi dari data, standar deviasi yang ditentukan memiliki nilai minimum 1e-6 dengan tujuan menghindari pembagian dengan angka sangat mendekati 0 untuk *method* gaussian_probability yang akan dijelaskan berikutnya.

Pada *method* gaussian_probability, pada metode ini digunakan rumus distribusi normal dengan pembatasan nilai eksponensial antara -500 hingga 500 untuk mencegah overflow. *Method* fit berisi training yang dilakukan terhadap data train. Prosesnya adalah dengan menghitung probabilitas prior untuk tiap kelas(label 0 dan label 1). Pada *method* ini dihitung mean dan standar deviasi mean dan standar deviasi *feature* untuk tiap kelas.

Pada *method* predict_proba, dihitung probabilitas tiap sampel yang ingin diketahui labelnya. Caranya menggunakan rumus distribusi Gaussian dengan mean dan std yang sudah dihitung di *method* fit. Hasil akhirnya berupa matriks probabilitas [jumlah sampel x jumlah kelas]. Pada *method* predict, diambil kelas dengan probabilitas tertinggi sebagai prediksi final. Terdapat *method* tambahan yaitu *method* score, *method* ini digunakan untuk menghitung akurasi prediksi yang dilakukan benar.

2.2 Implementasi Algoritma Pembelajaran Mesin Menggunakan Scikit-Learn

Implementasi algoritma pembelajaran mesin menggunakan *scikit-learn* memanfaatkan Penggunaan library scikit-learn untuk membangun model pembelajaran mesin dengan API yang sudah terstandarisasi. Lebih cepat, mudah digunakan, dan optimal untuk produksi, namun dengan abstraksi yang lebih tinggi terhadap detail internal algoritma.

2.2.1 Implementasi Algoritma KNN menggunakan *scikit-learn*

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np


key_columns = [21, 31, 35, 36, 46, 55] #sudah adjusted zero-index (-1)

X_train_reduced = X_train_balanced[:, key_columns_zero_indexed]
y_train_reduced = y_train_balanced[:]
X_test_reduced = X_val_transformed[:, key_columns_zero_indexed]
y_test_reduced = y_val[:]


knn = KNeighborsClassifier(n_neighbors=17)


knn.fit(X_train_reduced, y_train_reduced)


predictions = knn.predict(X_test_reduced)

accuracy = accuracy_score(y_test_reduced, predictions)
print(f"Accuracy with selected columns: {accuracy:.4f}")
```

Algoritma 2.2.1 Implementasi Algoritma KNN menggunakan *scikit-learn*

Pada algoritma di atas, K-Nearest Neighbor (KNN) diimplementasikan menggunakan library scikit-learn. Data pelatihan dan pengujian disiapkan dengan memilih kolom tertentu yang telah diidentifikasi memiliki kontribusi signifikan terhadap peningkatan akurasi (misalnya, indeks [21, 31, 35, 36, 46, 55]) berdasarkan analisis incremental terhadap jumlah fitur. Nilai **k=17** dipilih berdasarkan hasil validasi silang 5-Fold Cross Validation, di mana **k=17** memberikan akurasi tertinggi dan keseimbangan terbaik antara bias dan variansi.

Langkah-langkah utama meliputi pemilihan kolom relevan, inisialisasi KNeighborsClassifier dengan **n_neighbors=17**, pelatihan model dengan data yang tereduksi, prediksi terhadap data pengujian, dan evaluasi akurasi menggunakan fungsi accuracy_score. Pendekatan ini memastikan hanya fitur yang relevan digunakan, mengurangi kompleksitas tanpa mengorbankan kinerja model

2.2.2 Implementasi Algoritma Gaussian Naive-Bayes menggunakan *scikit-learn*

```
X_train_transformed = submission_pipe.fit_transform(X_train)

X_train_balanced, y_train_balanced =
balance_classes(X_train_transformed, y_train)

X_val = val_data_rem_dup.drop(columns=['id', 'label'])
y_val = val_data_rem_dup['label']

X_val_transformed = submission_pipe.transform(X_val)

nb = GaussianNB()
nb.fit(X_train_balanced, y_train_balanced)

val_predictions = nb.predict(X_val_transformed)
```

Algoritma 2.2.2 Implementasi Gaussian Naive-Bayes menggunakan *scikit-learn*

Pada algoritma di atas, data *training* ditransformasi menggunakan *pipeline* yang telah disiapkan (submission_pipe), kemudian dilakukan penyeimbangan kelas untuk memastikan distribusi data yang merata antar kelas. Selanjutnya, data validasi disiapkan dengan memisahkan *feature* dan label, serta menghapus kolom yang tidak diperlukan seperti 'id' dan 'label'. Data validasi ini juga ditransformasi menggunakan *pipeline* yang sama. Model dilatih menggunakan data training yang sudah diseimbangkan. Model yang sudah dilatih kemudian digunakan untuk

melakukan prediksi pada data validasi. *Feature* lain yang di-*drop* namun tidak tampak karena proses *cleaning* adalah 'FILENAME', 'Domain', 'URL', dan 'Title'.

2.3 Tahap Cleaning dan Preprocessing

Data cleaning dan preprocessing adalah tahap mengubah data mentah menjadi format terstruktur yang siap digunakan model pembelajaran mesin, dengan memastikan data bersih dari noise dan konsisten untuk menghasilkan prediksi yang akurat.

2.3.1 Tahap Cleaning

Data cleaning adalah proses pembersihan dataset dari masalah seperti missing values, outliers, data tidak konsisten dan duplikat untuk memastikan data siap digunakan dalam model pembelajaran mesin.

2.3.1.1 Drop Unnecessary Features

Fitur yang di *drop* adalah fitur yang memiliki korelasi mendekati 0 atau 0 terhadap label, dengan asumsi kolom ini memiliki behaviour sama dengan dataset pada Tucil 2.

Fitur yang di *drop* tersebut adalah 'FILENAME', 'Domain', 'URL', dan 'Title'.

2.3.1.2 Missing Value Percentages

Missing values pada tiap fitur dianalisis dengan melihat persentase tiap missing values pada tiap fitur.

Persentase *missing values* dipilih sebagai cara paling mudah untuk melihat seberapa banyak missing values pada tiap fitur dan apa yang sebaiknya dilakukan untuk menanganinya.

2.3.1.3 Split Numerical and Categorical Variables

Tiap data set (training set & validation set) di split berdasarkan tipe datanya untuk kemudahan dalam menangani *outliers.*

2.3.1.4 Feature Imputation

Pembuatan kelas 'FeatureImputer' yang berisikan fungsi-fungsi untuk menangani *missing values* dan nanti dimasukkan di *pipeline*.

Strategi yang dipilih untuk menanganinya pada fungsi-fungsi tersebut adalah dengan mengganti *numerical missing values* dengan *mean* pada fiturnya dan *categorical missing values* dengan *most frequent*.

Mean dipilih karena *mean* adalah rata-rata ukuran yang cenderung di tengah dan tidak membuat data menjadi condong ke arah yang tertentu. *Most frequent* dipilih untuk memastikan bahwa imputasi tidak memperkenalkan kategori baru pada *categorical feature* tersebut dan menjaga keseimbangan dan distribusi.

2.3.1.5 Dealing with Outliers

Outliers ditangani dengan membuat fungsi *cap_outlier(data, numerical_columns)* yang berfungsi dengan meng-*cap outlier* pada kolom numerik di dataset menggunakan *clipping* berdasarkan metode IQR.

Capping dipilih karena tidak menghilangkan *outliers,* tetapi hanya memberlakukan pembatasan nilai yang membuatnya masih menyimpan semua *rows* dalam dataset.

Penggunaan fungsi ini mengecualikan semua fitur yang hanya mempunyai 2 *unique values* (1 and 0) agar nilai mereka terjaga.

2.3.1.6 Remove Duplicates

Penghapusan duplikat dilakukan dengan menggunakan fungsi *drop_duplicates()* pada tiap dataset yang sudah di-*cap*.

Menghapus fitur yang duplikat dilakukan untuk menghilangkan bias pada saat model dilatih.

2.3.1.7 Feature Engineering

Pada tahap ini, kami merekayasa fitur dengan membuat beberapa fitur baru yang dapat menghasilkan informasi baru bagi pelatihan model.

Fitur 'TLD' dipilih karena memiliki korelasi yang cukup tinggi terhadap label dan merupakan fitur yang berisi nilai-nilai *top level domain*. Namun, karena *unique values*-nya yang tinggi menyebabkannya mempunyai *high cardinality*. Salah satu cara untuk menangani *high cardinality* ini adalah dengan mengambil hanya 10 *unique values* yang sering muncul pada dataset.

Setelah mendapat 10 *most frequent* TLDs, fitur TLD di-*recategorize*, TLDs yang tidak termasuk dalam 10 TLDs tersebut akan berada di grup dengan kategori *others*.

Fitur-fitur ada yang diturunkan, seperti 'URL to Domain Length Ratio' yang mengukur seberapa panjang URL keseluruhan relatif terhadap

domainnya; 'Letters, Digits, and Special Characters to URL Length Ratios; yang menunjukkan proporsi huruf, angka, dan karakter khusus di URL.

Binary features seperti 'HasRedirects' yang menunjukkan *redirect* sering digunakan untuk menutupi tujuan sebenarnya; 'HasPopup' yang menunjukkan banyaknya *popups* dapat terindikasi *phishing*; 'HasExternalReferences' yang dapat berarti jika ada banyak *external references* dapat terindikasi *phishing*.

Combined features seperti menggabungkan 'HasObfuscation' dan 'ObfuscationRatio' untuk membuat indikator ukuran tingkat keparahan pengaburan kode untuk dibaca. Membuat 'SecurityFeatureScore' dengan menggabungkan 'IsHTTPS' dengan nilai 1.5 karena HTTPS menunjukan komunikasi yang aman, 'HasPasswordField' dan 'HasSubmitButton' yang menunjukkan *user-sensitive input* yang biasanya ditiru *phishing sites*.

Semua itu lalu dimasukkan ke 'class FeatureCreator' untuk *pipeline.*

2.3.2 Tahap Preprocessing

Data preprocessing adalah proses transformasi data yang sudah dibersihkan menjadi format yang sesuai untuk algoritma pembelajaran mesin, meliputi scaling fitur numerik, encoding variabel kategorikal, penanganan ketidakseimbangan kelas, pengurangan dimensi, dan normalisasi data.

2.3.2.1 Feature Scaling

Pada tahap ini, algoritma FeatureScaler digunakan untuk melakukan normalisasi atau standarisasi fitur numerik menggunakan salah satu dari tiga metode yang tersedia: minmax, standard, atau robust. Langkah pertama adalah inisialisasi, di mana pengguna menentukan metode skala yang diinginkan. Kemudian, pada fungsi fit, scaler yang sesuai dipilih dan diatur (MinMaxScaler, StandardScaler, atau RobustScaler) berdasarkan metode yang dipilih, dan scaler tersebut dipasangkan pada data masukan untuk menghitung parameter scaling (seperti mean, standar deviasi, atau nilai minimum dan maksimum). Selanjutnya, fungsi transform digunakan untuk menerapkan transformasi pada data, menghasilkan data yang sudah diskalakan dalam bentuk DataFrame dengan kolom dan indeks asli tetap dipertahankan. Algoritma ini memungkinkan integrasi mudah dengan pipeline scikit-learn, memastikan data numerik siap untuk proses pembelajaran mesin.

2.3.2.2 Feature Encoding

Pada tahap ini, algoritma FeatureEncoder dirancang untuk melakukan encoding pada fitur kategorikal dalam dataset dengan menerapkan metode one-hot encoding. Langkah pertama adalah fungsi fit, yang dalam hal ini hanya mengembalikan objek itu sendiri karena tidak ada parameter yang perlu dihitung. Selanjutnya, fungsi transform menerima dataset dan membuat salinannya untuk memastikan data asli tidak berubah. Algoritma kemudian mengidentifikasi kolom-kolom dengan tipe data kategorikal (object), kecuali kolom dengan nama tertentu seperti id yang diabaikan dari proses encoding. Setiap kolom kategorikal diubah menjadi sejumlah kolom baru, masing-masing mewakili kategori unik dalam kolom tersebut, dengan nilai biner (0 atau 1). Setelah itu, kolom asli yang telah diencode dihapus dari dataset. Hasil akhirnya adalah dataset dengan semua fitur kategorikal diubah menjadi format numerik yang kompatibel dengan algoritma pembelajaran mesin.

2.3.2.3 Balancing Classes

Pada tahap ini, algoritma balance_classes bertujuan untuk menangani ketidakseimbangan kelas dalam data target dengan menggunakan teknik SMOTE (Synthetic Minority Oversampling Technique). Langkah pertama adalah mencetak distribusi kelas pada target sebelum dilakukan proses balancing untuk memberikan gambaran awal. Algoritma kemudian memverifikasi apakah kedua kelas hadir dalam target, karena SMOTE membutuhkan setidaknya dua kelas untuk melakukan oversampling; jika tidak, proses akan dihentikan dengan pesan kesalahan. Selanjutnya, SMOTE diterapkan pada data fitur (X_t) dan target (y_t) untuk membuat sampel sintetis pada kelas minoritas berdasarkan tetangga terdekat, menghasilkan dataset baru dengan distribusi kelas yang seimbang. Distribusi kelas pasca-resampling dicetak untuk memastikan bahwa proses telah berhasil. Hasil akhir berupa dataset fitur (X_t_res) dan target (y_t_res) yang telah seimbang, siap digunakan dalam proses pelatihan model pembelajaran mesin.

2.4 Perbandingan hasil prediksi dari algoritma yang diimplementasikan dengan hasil yang didapatkan dengan menggunakan pustaka

Berikut merupakan perbandingan hasil prediksi dari algoritma yang diimplementasikan *from scratch* dengan hasil yang didapatkan dengan menggunakan pustaka *scikit-learn*.

2.4.1 KNN

```
X_train_reduced = X_train_balanced[:1000, :2]
y_train_reduced = y_train_balanced[:1000]
X_test_reduced = X_val_transformed[:200, :2]
#memakai data testing yaitu 1000 baris dalam 2 kolom
pertama untuk mempercepat komputasi

knn = KNearestNeighborUpdated(verbose=True)
knn.fit(X_train_reduced, y_train_reduced)
predictions = knn.predict(X_test_reduced)
print("Predictions:",predictions)
print(accuracy_score(y_val[:200], predictions))
#KNearestNeigborUpdated adalah Algoritma KNN yang dibuat
secara manual


knn = KNeighborsClassifier(n_neighbors=17)
knn.fit(X_train_reduced, y_train_reduced)
predictions = knn.predict(X_test_reduced)
print(accuracy_score(y_val[:200], predictions))
#KNeighborsClassifier yaitu implementasi KNN dari
library scikit-learn
```

Kedua algoritma ini memakai *slice* yang sama dari data yang sama yaitu 1000 baris dalam 2 kolom pertama dari *X_train_balanced* dan *y_train_balanced* , serta memakai data testing yaitu 200 baris pertama dalam 2 kolom pertama *X_val_transformed*. Kemudian nilai k benchmark adalah k=17 , didapat dari *5Fold-CV* saat fitting.

```
print("Predictions:", predictions)
print(accuracy_score(y_val[:200], predictions))
Executed at 2024.12.22 13:47:53 in 1s 409ms

2024-12-22 13:47:52,812 - INFO - Computed distances with shape: (200, 800)
2024-12-22 13:47:52,815 - INFO - Identified 24 nearest neighbors for each test point
2024-12-22 13:47:52,818 - INFO - Prediction complete for 200 test points
2024-12-22 13:47:52,819 - INFO - Fold 4/5, k=24, accuracy=0.9250
2024-12-22 13:47:52,820 - INFO - Predicting with k=27, test data shape: (200, 2)
2024-12-22 13:47:52,828 - INFO - Computed distances with shape: (200, 800)
2024-12-22 13:47:52,831 - INFO - Identified 27 nearest neighbors for each test point
2024-12-22 13:47:52,837 - INFO - Prediction complete for 200 test points
2024-12-22 13:47:52,838 - INFO - Fold 4/5, k=27, accuracy=0.9250
2024-12-22 13:47:52,840 - INFO - Predicting with k=31, test data shape: (200, 2)
2024-12-22 13:47:52,849 - INFO - Computed distances with shape: (200, 800)
2024-12-22 13:47:52,854 - INFO - Identified 31 nearest neighbors for each test point
2024-12-22 13:47:52,857 - INFO - Prediction complete for 200 test points
2024-12-22 13:47:52,861 - INFO - Fold 4/5, k=31, accuracy=0.9250

Predictions: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
0.895
```

Hasil Algoritma KNN Manual yaitu akurasi sebesar 0.895

```
knn = KNeighborsClassifier(n_neighbors=17)


knn.fit(X_train_reduced, y_train_reduced)


predictions = knn.predict(X_test_reduced)

print(accuracy_score(y_val[:200], predictions))
Executed at 2024.12.22 14:03:16 in 46ms

 0.895
```

Hasil Algoritma KNN melalui library *scikit-learn* yaitu akurasi sebesar 0.895

Hasil testing menunjukkan bahwa implementasi KNN melalui algoritma manual dan *scikit-learn* menunjukkan nilai akurasi yang sama

2.4.2 Naive Bayes

Menggunakan pengecekan akurasi berdasarkan data validasi yang diperoleh dari data *train* (dalam hal ini 80% untuk *training*, 20% untuk cek akurasi), diperoleh data hasil sebagai berikut untuk *scratch* dan scikit-learn.

Jenis Algoritma	Akurasi	Kelas	Precision	Recall	F1-score
Scratch	0.6286	*Phishing*	0.9999	0.5985	0.7488
		Non-phishing	0.1682	0.9991	0.2880
Scikit-learn	0.7484	*Phishing*	0.2277	0.7295	0.8428
		Non-phishing	0.9979	0.9815	0.3697

Tabel 2.4.2 Perbandingan hasil prediksi Naive Bayes *from scratch* dengan *scikit-learn*

Berdasarkan tabel di atas, model scikit-learn menunjukkan performa yang lebih unggul dibandingkan implementasi *scratch*. Hal ini terlihat dari nilai akurasi keseluruhan yang lebih tinggi, skor akurasi scikit-learn mencapai 0.7484 sedangkan implementasi *scratch* hanya 0.6286. Dalam mendeteksi situs *phishing*, kedua model memiliki precision yang sangat tinggi (hampir 1), namun scikit-learn memiliki kemampuan deteksi (*recall*) yang lebih baik yaitu 0.7295 dibandingkan implementasi *scratch* yang hanya 0.5985. Untuk kasus *non-phishing*, meskipun kedua model menunjukkan *precision* yang relatif rendah, scikit-learn tetap mengungguli dengan nilai 0.2277 dibanding 0.1682 pada implementasi *scratch*. Kedua model memang sangat baik dalam mengidentifikasi situs *non-phishing* (*recall* tinggi), namun scikit-learn memberikan hasil yang lebih seimbang secara keseluruhan, yang tercermin dari nilai F1-score yang lebih tinggi untuk kedua kelas.

Bab III
Kesimpulan

Pada tugas ini telah dilakukan implementasi algoritma pembelajaran mesin KNN dan Gaussian Naive-Bayes dengan pendekatan *from scratch* dan dengan pustaka *scikit-learn*. Telah dilakukan juga data cleaning dan preprocessing pada dataset yang digunakan.

Adapun perbandingan hasil prediksi dari algoritma KNN dan Gaussian Naive-Bayes yang diimplementasikan *from scratch* dengan hasil yang didapatkan dengan menggunakan pustaka *scikit-learn.*

Berdasarkan hasil perbandingan, model *scikit-learn* menunjukkan performa yang lebih unggul dibandingkan implementasi *from scratch* untuk algoritma KNN.

Berdasarkan hasil perbandingan, model *scikit-learn* menunjukkan performa yang lebih unggul dibandingkan implementasi *from scratch* untuk algoritma Naive Bayes.

Bab IV
Kontribusi

Nama	NIM	Kontribusi
Matthew Lim	18222005	Data Preprocessing, Submission, Debug Model, Documentation
Habib Akhmad Al Farisi	18222029	Naive Bayes, Documentation
Winata Tristan	18222061	KNN, Documentation
Ahmad Habibie Marjan	18222082	Data Preprocessing, Membuat dan merapikan laporan, Documentation, README
Satria Wisnu Wibowo	18222087	Data Cleaning, Data Preprocessing, Documentation

Tabel 4 Kontribusi Pengerjaan Tugas Besar 2 IF3070 Kelompok 32

Bab V
Referensi

5.1 https://www.geeksforgeeks.org/machine-learning-algorithms/
5.2 https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset
5.3 https://www.sciencedirect.com/science/article/abs/pii/S0167404823004558?via%3Dihub
5.4 https://scikit-learn.org/1.5/modules/neighbors.html
5.5 https://scikit-learn.org/1.5/modules/naive_bayes.html