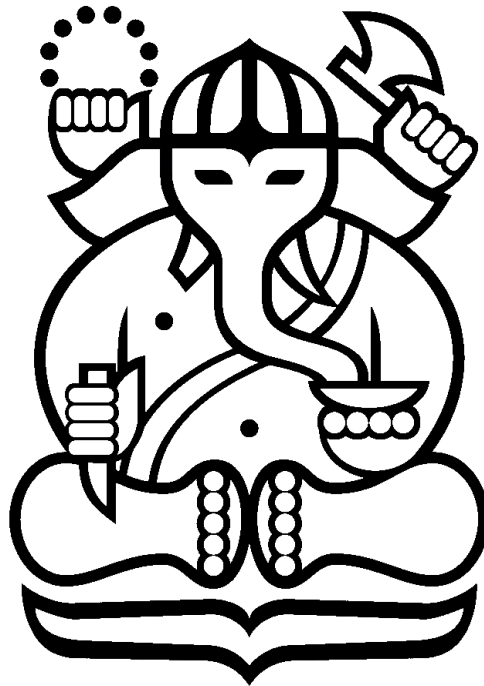


**TUGAS BESAR 2 IF3070 DASAR INTELEGENSI ARTIFISIAL
IMPLEMENTASI ALGORITMA PEMBELAJARAN MESIN**



DISUSUN OLEH KELOMPOK “Pembelajaran HMM” :

MUHAMMAD YAAFI WASESA PUTRA	18222052
TAUFIQ RAMADHAN AHMAD	18222060
JOSIA RYAN JULIANDY SILALAH	18222075
MUHAMMAD ADLI ARINDRA	18222089

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

2024

DAFTAR ISI

DAFTAR ISI.....	1
I. DESKRIPSI PERSOALAN.....	2
II. IMPLEMENTASI KNN.....	3
III. IMPLEMENTASI NAIVE-BAYES.....	6
IV. DATA CLEANING.....	8
4.1 Handling Missing Data.....	8
4.2 Dealing with Outliers.....	15
4.3 Remove Duplicates.....	16
4.4 Feature Engineering.....	16
V. DATA PREPROCESSING.....	18
5.1 Feature Scaling.....	18
5.2 Compile Preprocessing Pipeline.....	18
VI. PERBANDINGAN HASIL PREDIKSI.....	19
6.1 k-Nearest Neighbors.....	19
6.2 Gaussian Naive-Bayes.....	19
VII. KONTRIBUSI ANGGOTA KELOMPOK.....	19
VIII. REFERENSI.....	20

I. DESKRIPSI PERSOALAN

Phishing adalah salah satu serangan siber yang memanfaatkan URL palsu untuk menipu pengguna agar memberikan informasi sensitif seperti kata sandi atau data pribadi. Pada tugas ini, dataset diambil dari sebuah artikel ilmiah yang membahas kerangka kerja baru bernama **PhiUSIIL** untuk mendeteksi URL phishing dengan lebih efektif. PhiUSIIL menggunakan dua pendekatan utama: **URL Similarity Index (USI)** dan **Incremental Learning**. Singkatnya, dataset ini akan menghasilkan tingkat kemiripan antara URL asli dan URL palsu yang digunakan untuk *phishing*.

Pada tugas besar ini, tim kami mengimplementasikan algoritma pembelajaran mesin awal (*from scratch*) untuk dua algoritma utama. Dua algoritma utama itu yakni *K-Nearest Neighbors* (KNN) dan Gaussian Naive-Bayes. Implementasi kedua algoritma ini mencakup perancangan kelas-kelas untuk tiap algoritma. Pada implementasi KNN, algoritma akan menerima tiga input utama: jumlah tetangga, metrik jarak antar data, dan ukuran *batch*. Gaussian Naive-Bayes diharapkan dapat mengimplementasikan model probabilistik yang berbasis distribusi normal untuk setiap fitur.

Untuk menghitung performa dari algoritma yang kami implementasikan, pada tugas ini juga diterapkan model yang sama menggunakan pustaka scikit-learn. Metode ini digunakan untuk membandingkan hasil prediksi dari implementasi manual dan pustaka standar.

Selain implementasi algoritma untuk prediksi pemodelan, tugas ini juga mencakup tahap pra-pemrosesan data, seperti mengatasi data yang hilang, pembersihan data, data yang duplikat, dan sebagainya. Pada dataset yang digunakan fitur target adalah **label**.

II. IMPLEMENTASI KNN

```
class KNN:
    def __init__(self, k=3, metric='euclidean', p=2,
batch_size=500):
        self.k = k
        self.metric = metric.lower()
        self.p = p
        self.batch_size = batch_size
        self.X_train = None
        self.y_train = None
        self.classes_ = None

    def fit(self, X, y):
        self.X_train = np.asarray(X, dtype=np.float32)
        self.y_train = np.asarray(y, dtype=np.int64)
        self.classes_ = np.unique(y)
        return self

    def _compute_distances_batch(self, X_batch):
        if self.metric == 'euclidean':
            test_norm = np.sum(X_batch**2, axis=1)[:,
np.newaxis]
            train_norm = np.sum(self.X_train**2, axis=1)

            distances = -2 * np.dot(X_batch,
self.X_train.T)
            distances += test_norm + train_norm

            return np.sqrt(np.maximum(distances, 0))

        elif self.metric == 'manhattan':
            return np.sum(
                np.abs(X_batch[:, np.newaxis] -
self.X_train),
                axis=2
            )

        elif self.metric == 'minkowski':
            diff = X_batch[:, np.newaxis] - self.X_train
            return np.sum(
                np.abs(diff) ** self.p,
                axis=2
            ) ** (1/self.p)

        else:
            raise ValueError(f"Unsupported metric:
{self.metric}. Choose from 'euclidean', 'manhattan', or
'minkowski'")

    def predict(self, X):
        X = np.asarray(X, dtype=np.float32)
        predictions = np.zeros(X.shape[0], dtype=np.int64)
```

```

        for i in range(0, X.shape[0], self.batch_size):
            batch_end = min(i + self.batch_size,
X.shape[0])
            X_batch = X[i:batch_end]

            distances =
self._compute_distances_batch(X_batch)
            k_nearest_indices = np.argpartition(distances,
self.k, axis=1)[:self.k]
            k_nearest_labels =
self.y_train[k_nearest_indices]

            for j, labels in enumerate(k_nearest_labels):
                predictions[i + j] =
np.bincount(labels.astype(np.int64)).argmax()

        return predictions

    def predict_proba(self, X):
        X = np.asarray(X, dtype=np.float32)
        probabilities = np.zeros((X.shape[0],
len(self.classes_)), dtype=np.float32)

        for i in range(0, X.shape[0], self.batch_size):
            batch_end = min(i + self.batch_size,
X.shape[0])
            X_batch = X[i:batch_end]

            distances =
self._compute_distances_batch(X_batch)
            k_nearest_indices = np.argpartition(distances,
self.k, axis=1)[:self.k]
            k_nearest_labels =
self.y_train[k_nearest_indices]

            for j, labels in enumerate(k_nearest_labels):
                unique_labels, counts = np.unique(labels,
return_counts=True)
                probabilities[i + j,
unique_labels.astype(int)] = counts / self.k

        return probabilities

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

```

Implementasi K-Nearest Neighbors (KNN) pada kode ini menggunakan dirancang untuk melakukan klasifikasi berbasis jarak. Kelas ini diinisialisasi dengan parameter utama, termasuk **k** sebagai jumlah tetangga terdekat, **metric** untuk menentukan metrik jarak yang digunakan (Euclidean, Manhattan, atau Minkowski),

serta **p** yang relevan untuk jarak Minkowski. Fungsi **fit()** digunakan untuk menyimpan data pelatihan (**X_train** dan **y_train**) serta menentukan mana kelas yang unik pada label data. Pada inti algoritma, metode **_compute_distances_batch()** menghitung jarak antara data uji dengan data pelatihan secara efisien menggunakan operasi matriks. Untuk jarak Euclidean, rumus $(a - b)^2 = a^2 + b^2 - 2ab$ diterapkan untuk menghindari *loop* eksplisit dan meningkatkan efisiensi. Jarak Manhattan dihitung dengan menjumlahkan perbedaan absolut, sedangkan jarak Minkowski adalah generalisasi kedua jarak tersebut.

Proses prediksi dilakukan oleh fungsi **predict()**. Fungsi ini membagi data uji menjadi *batch* untuk menghemat memori pada dataset besar. Metode ini menghitung jarak untuk setiap *batch* data uji, kemudian menentukan *k* tetangga terdekat menggunakan **np.argpartition** yang efisien untuk memilih nilai terkecil tanpa mengurutkan seluruh *array*. Prediksi akhir dibuat dengan menentukan mayoritas label di antara *k* tetangga terdekat, menggunakan fungsi **np.bincount()** untuk menghitung frekuensi label. Selain itu, fungsi **predict_proba()** menghitung probabilitas untuk setiap kelas berdasarkan proporsi label di antara tetangga terdekat. Akurasi model dapat dievaluasi menggunakan metode **score()**, yang membandingkan hasil prediksi dengan label sebenarnya. Secara keseluruhan, implementasi ini menggabungkan efisiensi komputasi dengan fleksibilitas metrik jarak untuk menangani dataset besar melalui *batch processing*.

III. IMPLEMENTASI NAIVE-BAYES

```
class GaussianBayes:
    def fit(self, X, y):
        X = np.array(X, dtype=float)
        y = np.array(y)

        self.classes = np.unique(y)
        self.parameters = {}

        for cls in self.classes:
            X_cls = X[y == cls]
            mean = X_cls.mean(axis=0)
            var = X_cls.var(axis=0) + 1e-6
            prior = X_cls.shape[0] / X.shape[0]
            self.parameters[cls] = {'mean': mean, 'var':
var, 'prior': prior}

    def gaussian_prob(self, x, mean, var):
        exponent = np.exp(- ((x - mean) ** 2) / (2 *
var))
        return (1 / np.sqrt(2 * np.pi * var)) * exponent

    def predict(self, X):
        X = np.array(X, dtype=float)

        y_pred = []
        for x in X:
            posteriors = []
            for cls in self.classes:
                params = self.parameters[cls]
                prior = np.log(params['prior'])
                conditional =
np.sum(np.log(self.gaussian_prob(x, params['mean'],
params['var'])))
                posterior = prior + conditional
                posteriors.append(posterior)

            y_pred.append(self.classes[np.argmax(posteriors)])
        return np.array(y_pred)
```

Kode ini terdiri dari tiga fungsi utama, yaitu **fit** untuk mempelajari nilai rata-rata (**mean**), **varians**, dan **prior** pada tiap kelas, **gaussian_prob** untuk

menghitung probabilitas Gaussian dari setiap fitur, dan **predict** untuk memprediksi kelas. Algoritma Gaussian Naive Bayes mengasumsikan setiap fitur x pada suatu kelas mengikuti distribusi normal dengan mean μ dan varian σ^2 . Secara matematis, kepadatan probabilitasnya dihitung menggunakan dengan persamaan sebagai berikut :

$$p(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Saat training, data dipecah berdasarkan label untuk menghitung mean, varian, dan prior masing-masing kelas. Pada saat prediksi, algoritma mengambil log prior dan menjumlahkannya dengan log probabilitas tiap fitur, sesuai persamaan berikut :

$$\hat{y} = \arg \max_{c \in classes} [\ln(\text{prior}_c) + \sum_{j=1}^d \ln(p(x_j | \mu_{c,j}, \sigma^2))]$$

Kelas yang memiliki nilai posterior paling tinggi akan dipilih sebagai *output*. Pendekatan “naive” berarti antar-fitur dianggap independen, membuat perhitungan lebih sederhana namun tetap efektif dalam berbagai kasus.

IV. DATA CLEANING

Untuk tahap pembersihan data, kami menggunakan beberapa metode. Adapun metode-metode tersebut adalah sebagai berikut.

4.1 *Handling Missing Data*

```
def imputeDomain(row):
    if pd.notna(row["Domain"]):
        return row["Domain"]
    if pd.isna(row["URL"]):
        return np.nan
    return row["URL"].split("://")[1].split("/")[0]

def imputeURLLength(row):
    if pd.notna(row["URLLength"]):
        return row["URLLength"]
    if pd.isna(row["URL"]):
        if pd.notna(row["DomainLength"]):
            return row["DomainLength"] + 8
        return np.nan
    return len(row["URL"])

def imputeDomainLength(row):
    if pd.notna(row["DomainLength"]):
        return row["DomainLength"]
    if pd.isna(row["Domain"]):
        return np.nan
    return len(row["Domain"])

def imputeIsDomainIP(row):
    def contains_letter(string):
        return any(char.isalpha() for char in string)

    if pd.notna(row["IsDomainIP"]):
        return row["IsDomainIP"]

    if pd.notna(row["Domain"]) and not contains_letter(row["Domain"]):
        return 1

    return 0

def imputeTLD(row):
    if pd.notna(row["TLD"]):
        return row["TLD"]
    if pd.isna(row["Domain"]):
```

```

        return row["Domain"].split('.')[ -1]
    return np.nan

def imputeCharContinuationRate(row, mean=None):
    if pd.notna(row["CharContinuationRate"]):
        return row["CharContinuationRate"]
    return mean if mean is not None else 0

def imputeTLDEgitimateProb(row, mode_dict):
    if pd.notna(row["TLD"]):
        return mode_dict.get(row["TLD"], np.nan)
    return row["TLDEgitimateProb"]

def imputeURLCharProb(row, mean=None):
    if pd.notna(row["URLCharProb"]):
        return row["URLCharProb"]
    return mean if mean is not None else 0

def imputeTLDLength(row):
    if pd.notna(row["TLDLength"]):
        return row["TLDLength"]
    if pd.isna(row["TLD"]):
        return np.nan
    return len(row["TLD"])

def imputeNoOfSubDomain(row):
    if pd.notna(row["NoOfSubDomain"]):
        return row["NoOfSubDomain"]
    if pd.notna(row["Domain"]):
        return len(row["Domain"].split('.')) - 1
    return np.nan

def imputeNoOfObfuscatedChar(row):
    if row["HasObfuscation"] == 0:
        return 0
    if pd.notna(row["NoOfObfuscatedChar"]):
        return row["NoOfObfuscatedChar"]
    if pd.notna(row["URL"]):
        return row["URL"].count('%') * 3
    return np.nan

def imputeObfuscationRatio(row):
    if pd.notna(row["ObfuscationRatio"]):
        return row["ObfuscationRatio"]
    if pd.isna(row["NoOfObfuscatedChar"]) or
pd.isna(row["URLLength"]) or row["URLLength"] == 0:
        return np.nan
    return row["NoOfObfuscatedChar"] /
row["URLLength"]

```

```

def imputeNoOfLettersInURL(row):
    if pd.notna(row["NoOfLettersInURL"]):
        return row["NoOfLettersInURL"]
    if pd.isna(row["URL"]):
        return np.nan
    return sum(char.isalpha() for char in
row["URL"])

def imputeLetterRatioInURL(row):
    if pd.notna(row["LetterRatioInURL"]):
        return row["LetterRatioInURL"]
        if pd.isna(row["NoOfLettersInURL"]) or
pd.isna(row["URLLength"]) or row["URLLength"] == 0:
            return np.nan
        return row["NoOfLettersInURL"] /
row["URLLength"]

def imputeNoOfDegitsInURL(row):
    if pd.notna(row["NoOfDegitsInURL"]):
        return row["NoOfDegitsInURL"]
    if pd.isna(row["URL"]):
        return np.nan
    return sum(char.isdigit() for char in
row["URL"])

def imputeDegitRatioInURL(row):
    if pd.notna(row["DegitRatioInURL"]):
        return row["DegitRatioInURL"]
        if pd.isna(row["NoOfDegitsInURL"]) or
pd.isna(row["URLLength"]) or row["URLLength"] == 0:
            return np.nan
        return row["NoOfDegitsInURL"] /
row["URLLength"]

def imputeNoOfEqualsInURL(row):
    if pd.notna(row["NoOfEqualsInURL"]):
        return row["NoOfEqualsInURL"]
    if pd.isna(row["URL"]):
        return 0
    return row["URL"].count('=')

def imputeNoOfQMarkInURL(row):
    if pd.notna(row["NoOfQMarkInURL"]):
        return row["NoOfQMarkInURL"]
    if pd.isna(row["URL"]):
        return 0
    return row["URL"].count('?')

def imputeNoOfAmpersandInURL(row):
    if pd.notna(row["NoOfAmpersandInURL"]):

```

```

        return row["NoOfAmpersandInURL"]
    if pd.isna(row["URL"]):
        return 0
    return row["URL"].count('&')

def imputeNoOfOtherSpecialCharsInURL(row):
    if pd.notna(row["NoOfOtherSpecialCharsInURL"]):
        return row["NoOfOtherSpecialCharsInURL"]
    if pd.isna(row["URL"]):
        return 0
    return sum(1 for char in row["URL"] if not
char.isalnum() and char not in ['=', '?', '&', '/',
'.'])

def imputeSpacialCharRatioInURL(row):
    if pd.notna(row["SpacialCharRatioInURL"]):
        return row["SpacialCharRatioInURL"]
        if pd.isna(row["URLLength"]) or
row["URLLength"] == 0:
            return 0
        special_chars = (
            row.get("NoOfOtherSpecialCharsInURL", 0) +
            row.get("NoOfEqualsInURL", 0) +
            row.get("NoOfQMarkInURL", 0) +
            row.get("NoOfAmpersandInURL", 0)
        )
        return special_chars / row["URLLength"]

def imputeIsHTTPS(row):
    if pd.notna(row["IsHTTPS"]):
        return row["IsHTTPS"]
    if pd.notna(row["URL"]):
        return 1 if row["URL"].startswith("https")
    else 0
    return np.nan

def imputeDomainTitleMatchScore(row):
    if pd.notna(row["DomainTitleMatchScore"]):
        return row["DomainTitleMatchScore"]
    if pd.notna(row["URLTitleMatchScore"]):
        return row["URLTitleMatchScore"]

    domain = str(row.get("Domain", "")) if
pd.notna(row.get("Domain", "")) else ""
    title = str(row.get("Title", "")) if
pd.notna(row.get("Title", "")) else ""
    tld_length = int(row.get("TLDLength", 0)) if
pd.notna(row.get("TLDLength", 0)) else 0

    if not domain or not title:

```

```

        return 0

        domain_without_tld = domain[:-tld_length-1] if
len(domain) > tld_length else domain

        match_ratio = SequenceMatcher(None,
domain_without_tld, title).ratio()

        return match_ratio * 100

def                                imputeFilename(row,
default_filename="unknown_file.txt"):
    if pd.isna(row["FILENAME"]):
        return row["FILENAME"]
    if pd.isna(row["URL"]):
        return row["URL"].split("/")[-1]
    return default_filename

def imputeDomainFromFilename(row):
    if pd.isna(row["Domain"]):
        return row["Domain"]
    if pd.isna(row["FILENAME"]):
        return row["FILENAME"].split(".")[0]
    return np.nan

def imputeURLLengthGroup(row, mean_lengths):
    if pd.isna(row["URLLength"]):
        return row["URLLength"]
    group = row.get("TLD", "unknown")
    return mean_lengths.get(group, np.nan)

def imputeIsDomainIPFromPatterns(row):
    if pd.isna(row["IsDomainIP"]):
        return row["IsDomainIP"]
    if pd.isna(row["Domain"]):
        return 1 if all(part.isdigit() for part in
row["Domain"].split(".")) else 0
    return np.nan

def imputeHasObfuscationByHeuristics(row):
    if pd.isna(row["HasObfuscation"]):
        return row["HasObfuscation"]
    if pd.isna(row["URL"]):
        return 1 if "%" in row["URL"] else 0
    return np.nan

def imputeLabelFromPatterns(row):
    if pd.isna(row.get("label")):
        return row["label"].split(";")[0]
    return "unknown"

```

```

def imputeNoOfRedirects(row):
    if pd.isna(row["NoOfURLRedirect"]):
        return row["NoOfURLRedirect"]
    if pd.isna(row["NoOfSelfRedirect"]):
        return row["NoOfSelfRedirect"]
    return 0

def imputeHasSocialFeatures(row):
    if pd.isna(row["HasSocialNet"]):
        return row["HasSocialNet"]
    if pd.isna(row["URL"]):
        social_keywords = ["facebook", "twitter",
"linkedin", "instagram"]
        return 1 if any(keyword in row["URL"] for
keyword in social_keywords) else 0
    return 0

def imputeLineOfCode(row, fill_value=100):
    if pd.isna(row["LineOfCode"]):
        return row["LineOfCode"]
    return fill_value

def imputeLargestLineLength(row, fill_value=200):
    if pd.isna(row["LargestLineLength"]):
        return row["LargestLineLength"]
    return fill_value

def imputeHasFavicon(row):
    if pd.isna(row["HasFavicon"]):
        return row["HasFavicon"]
    return 0

def imputeRobots(row):
    if pd.isna(row["Robots"]):
        return row["Robots"]
    return 0

def imputeIsResponsive(row):
    if pd.isna(row["IsResponsive"]):
        return row["IsResponsive"]
    return 0

def imputeHasDescription(row):
    if pd.isna(row["HasDescription"]):
        return row["HasDescription"]
    return 0

def imputeNoOfPopup(row):
    if pd.isna(row["NoOfPopup"]):

```

```

        return row["NoOfPopup"]
    return 0

def imputeNoOfiFrame(row):
    if pd.isna(row["NoOfiFrame"]):
        return row["NoOfiFrame"]
    return 0

def imputeHasTitle(row):
    if pd.isna(row["HasTitle"]):
        return row["HasTitle"]
    if pd.isna(row["Title"]):
        return 1
    return np.nan

from scipy.stats import entropy

def calculate_entropy(url):
    if pd.isna(url):
        char_counts = pd.value_counts(list(url),
normalize=True)
        return entropy(char_counts)
    return np.nan

def count_punctuation(url):
    if pd.isna(url):
        punctuation_count = sum(1 for char in url
if char in ['.', '/', '-'])
        return punctuation_count
    return np.nan

```

Penting untuk menangani data yang hilang dengan benar karena dapat membuat atau menghancurkan model yang kita buat. Pada tugas ini, kami lebih banyak memilih metode *impute* dikarenakan nilai yang hilang berada pada kisaran 20% - 30% untuk tiap fitur data kami. Secara umum, fungsi-fungsi *impute* yang kami implementasikan melakukan hal berikut.

- Menghitung fitur dari fitur lain
- Menghitung fitur menggunakan mediannya
- Menghapus baris tempat URL yang bernilai NaN

Namun, untuk membantu memaksimalkan penanganan nilai data yang hilang, kami juga menggunakan fungsi **SimpleImputer()** dari pustaka SKLearn.

```
simple_imputer = SimpleImputer(strategy="mean")

def simple_impute(df: pd.DataFrame, train: bool =
False) -> pd.DataFrame:
    if train:
        labels = df["label"]
        features = df.drop("label", axis=1)

        imputed_data =
simple_imputer.fit_transform(features)

        imputed_df = pd.DataFrame(imputed_data,
columns=features.columns, index=df.index)
        imputed_df["label"] = labels
    else:
        imputed_data = simple_imputer.transform(df)
        imputed_df = pd.DataFrame(imputed_data,
columns=df.columns, index=df.index)

    return imputed_df
```

4.2 Dealing with Outliers

Outlier merupakan istilah dalam dunia data untuk nilai pada data pada suatu fitur yang memiliki nilai yang cukup jauh atau bahkan ekstrem jauh dari rentang data yang terdapat pada suatu fitur. Untuk menanganinya hal tersebut, kami mengimplementasikan fungsi berikut untuk menangani *outlier*.

```
def apply_winsorization(df, limits=(0.005, 0.005)):
    df_winsorized = df.copy()

    for column in
df_winsorized.select_dtypes(include=['float64',
'int64']).columns:
        df_winsorized[column] =
winsorize(df_winsorized[column], limits=limits)

    return df_winsorized
```

Pada kasus ini, kami menggunakan **winsorization** dari pustaka Scipy.stats. *Winsorization* adalah teknik statistik yang mengubah *outliers* dalam suatu data

menjadi nilai yang lebih dekat ke batas distribusi data. Pengimplementasian fungsi ini diharapkan mengurangi dampak *outliers* untuk proses modelling nantinya.

4.3 Remove Duplicates

Duplikasi pada data terlebih dalam jumlah yang banyak akan mengurangi kualitas data yang akan dimodelkan. Hal ini maksudnya adalah data tidak memiliki akurasi dan integritas yang baik untuk dilatih dan dijadikan sebagai prediksi. Pada tugas ini, kami mengimplementasikan fungsi `remove_duplicates()` sebagai berikut.

```
def remove_duplicates(df: pd.DataFrame) ->
pd.DataFrame:
    ret = df.copy()
    ret.drop_duplicates(subset=None, keep='first',
inplace=True)
    return ret
```

Kode di atas bekerja dengan pertama kali membuat salinan dari *DataFrame*. Setelah itu, semua kolom dipertimbangkan untuk mendeteksi duplikat. Apabila terdeteksi nilai yang duplikat, kode di atas akan mempertahankan nilai pada baris pertama yang ditemukan dan menghapus sisanya. Selain itu, kode di atas juga memastikan bahwa perubahan dilakukan langsung pada *DataFrame* tanpa membuat salinan tambahan untuk menghemat memori.

4.4 Feature Engineering

Feature Engineering melibatkan pembuatan fitur baru (variabel input) atau mengubah fitur yang sudah ada untuk meningkatkan kinerja model pembelajaran mesin. Feature Engineering bertujuan untuk meningkatkan kemampuan model dalam mempelajari pola dan membuat prediksi akurat dari data. Sering dikatakan bahwa "fitur yang baik menghasilkan model yang baik."

Pada kasus ini, kami memilih menggunakan *feature selection* yaitu dengan melakukan pemilihan fitur yang paling relevan dan informatif dari kumpulan data. Kami menggunakan pemilihan fitur sebagai satu-satunya metode karena

kami hanya perlu menghapus fitur yang berlebihan dan sudah memasukkan fitur yang dibutuhkan. Implementasinya adalah sebagai berikut.

```
def feature_selection(df: pd.DataFrame, train: bool
= False) -> pd.DataFrame:
    ret = df.copy()
    if train:
        ret = ret[["IsDomainIP", "TLDDomains", "TLDLegitimateProb",
"URLCharProb", "NoOfSubDomain", "ObfuscationRatio",
"LetterRatioInURL",
"DegitRatioInURL", "SpacialCharRatioInURL",
"IsHTTPS", "LineOfCode", "LargestLineLength",
"DomainTitleMatchScore",
"HasFavicon", "Robots", "IsResponsive",
"NoOfURLRedirect", "NoOfSelfRedirect",
"HasDescription", "NoOfPopup",
"NoOfiFrame", "HasExternalFormSubmit",
"HasSocialNet", "HasSubmitButton",
"HasHiddenFields",
"HasPasswordField", "Bank", "Pay", "Crypto",
"HasCopyrightInfo", "NoOfImage",
"NoOfCSS", "NoOfJS", "NoOfSelfRef",
"NoOfEmptyRef", "NoOfExternalRef", "HasTitle",
"CharContinuationRate", "URLEntropy",
"PunctuationCount", "DomainToURLRatio",
"NumericCharRatio", "WordCountInURL",
"HasSensitiveKeywords", "label"]]
    else:
        ret = ret[["IsDomainIP", "TLDDomains", "TLDLegitimateProb",
"URLCharProb", "NoOfSubDomain", "ObfuscationRatio",
"LetterRatioInURL",
"DegitRatioInURL", "SpacialCharRatioInURL",
"IsHTTPS", "LineOfCode", "LargestLineLength",
"DomainTitleMatchScore",
"HasFavicon", "Robots", "IsResponsive",
"NoOfURLRedirect", "NoOfSelfRedirect",
"HasDescription", "NoOfPopup",
"NoOfiFrame", "HasExternalFormSubmit",
"HasSocialNet", "HasSubmitButton",
"HasHiddenFields",
"HasPasswordField", "Bank", "Pay", "Crypto",
"HasCopyrightInfo", "NoOfImage",
"NoOfCSS", "NoOfJS", "NoOfSelfRef",
"NoOfEmptyRef", "NoOfExternalRef", "HasTitle",
"CharContinuationRate", "URLEntropy",
"PunctuationCount", "DomainToURLRatio",
"NumericCharRatio", "WordCountInURL",
"HasSensitiveKeywords"]]
    return ret
```

Untuk *feature selection*, kami memilih 43 (di luar fitur *label*) fitur yang dinilai paling baik dan relevan untuk dilakukan pemodelan.

V. DATA PREPROCESSING

5.1 Feature Scaling

Kode berikut mendefinisikan fungsi skala yang menskalakan kolom numerik dari sebuah *DataFrame*, kecuali kolom label, menggunakan dua skalar secara berurutan: ***RobustScaler*** untuk mengurangi pengaruh *outlier*, diikuti oleh ***MinMaxScaler*** untuk menormalkan nilai antara 0 dan 1. Fungsi ini mengembalikan *DataFrame* baru dengan nilai yang diskalakan sambil tetap menjaga data asli tetap utuh.

```
standard_scaler = StandardScaler()

def scale(df: pd.DataFrame, train:bool = False) ->
pd.DataFrame:
    cols = [col for col in df.columns if col !=
"label"]
    ret = df.copy()
    if train:
        ret[cols] =
standard_scaler.fit_transform(df[cols])
    else:
        ret[cols] = standard_scaler.transform(df[cols])
    return ret
```

5.2 Compile Preprocessing Pipeline

```
def custom_pipeline(df: pd.DataFrame, train: bool =
False) -> pd.DataFrame:
    ret = df.copy()
    ret = manual_impute(ret, train)
    ret = remove_duplicates(ret)
    ret = feature_selection(ret, train)
    ret = simple_impute(ret, train)
    if train: ret = apply_winsorization(ret)
    ret = scale(ret, train)
    return ret
```

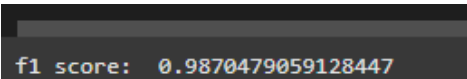

Fungsi ***custom_pipeline*** memproses *DataFrame* melalui fungsi-fungsi *data preprocessing* yang sudah diimplementasikan sebelumnya. Dimulai dengan

membuat salinan data input **df**, lalu melakukan operasi berikut secara berurutan:

- Melakukan imputasi manual menggunakan *manual_impute()*
- Menghapus nilai duplikat dengan *remove_duplicates()*
- Memilih hanya serangkaian fitur yang relevan menggunakan *feature_selection()*
- Melakukan imputasi sederhana dengan *simple_impute()*
- Melakukan penskalaan fitur dengan *scale()*
- Mengatasi *outlier* dengan *apply_winsorization()*

VI. PERBANDINGAN HASIL PREDIKSI

6.1 *k-Nearest Neighbors*

From Scratch	Sklearn
	

Hasil implementasi algoritma KNN menunjukkan hasil yang cukup memuaskan. Dengan menggunakan pustaka sklearn, kami berhasil mendapatkan akurasi 98,03%. Di sisi lain, dengan algoritma *from scratch* yang kami rancang, berhasil mendapatkan akurasi 98,70%. Perbedaannya sangat kecil yaitu sekitar 0,67% yang berarti implementasi algoritma kami sudah mampu mencapai performa yang hampir setara dengan implementasi sklearn. Hal ini menunjukkan bahwa algoritma yang dibangun secara manual sudah cukup bagus dan dapat diandalkan. Hasil yang serupa ini juga mengonfirmasi bahwa implementasi from scratch telah mengikuti prinsip-prinsip dasar kNN dengan benar, termasuk dalam perhitungan jarak, pemilihan tetangga terdekat, dan ukuran kelompok eksekusi data untuk efisiensi memori.

Perbedaan akurasi yang sedikit lebih tinggi pada sklearn kemungkinan disebabkan oleh optimasi internal yang lebih baik dalam *library* tersebut, seperti penanganan kasus khusus atau algoritma pencarian kNN yang lebih efisien. Secara keseluruhan, implementasi kNN from scratch telah berhasil mendekati performa sklearn yang optimal, membuktikan bahwa dasar-dasar

algoritma telah diterapkan dengan tepat. Selisih kecil dalam akurasi bisa dimaklumi karena sklearn sudah sangat teroptimasi dan matang.

6.2 *Gaussian Naive-Bayes*

From Scratch	Sklearn
f1 score: 0.9167590899415018	f1 score: 0.9517298587346351

Berdasarkan perbandingan antara implementasi Naive Bayes menggunakan sklearn dan yang dibuat dari *scratch*, terlihat bahwa kedua model menghasilkan akurasi yang hampir sama. Untuk *scratch* nilai akurasi f1-score adalah 91,6% sedangkan menggunakan sklearn 92,8%. Dimana perbedaannya tidak signifikan yakni hanya 1,2%. Perbedaan akurasi yang begitu kecil menunjukkan bahwa implementasi dari *scratch* berhasil mengimplementasikan fungsi dasar algoritma Naive Bayes dengan baik, termasuk dalam perhitungan mean, varians dan prior.

Hasil akurasi di atas 90% menunjukkan bahwa implementasi Gaussian Naive-Bayes yang kami implementasikan cocok dengan dataset ini. Hal ini tentu didukung dengan proses pembersihan dan pra-pemrosesan data yang kami implementasikan. Kemiripan hasil antara kedua model membuktikan bahwa implementasi dari *scratch* telah mengikuti prinsip dasar Gaussian Naive-Bayes dengan tepat. Perbedaan yang tidak signifikan dalam akurasi kemungkinan besar disebabkan oleh perbedaan dalam penanganan kasus khusus atau optimasi numerik yang ada di dalam library sklearn. Hal ini cukup masuk akal karena pustaka sklearn pasti menerapkan algoritma yang lebih canggih dan rapi.

VII. KONTRIBUSI ANGGOTA KELOMPOK

NIM	Nama	Tugas
18222052	Muhammad Yaafi Wasesa Putra	<ul style="list-style-type: none">• Mengerjakan laporan• Mengerjakan <i>feature engineering</i>• Mengembangkan model
18222060	Taufiq Ramadhan Ahmad	<ul style="list-style-type: none">• Mengerjakan laporan• Membuat <i>Gaussian Naive Bayes</i>• Melakukan <i>Data Cleaning</i>
18222075	Josia Ryan Juliandy Silalahi	<ul style="list-style-type: none">• Mengerjakan laporan• Membuat KNN• Melakukan <i>Data Cleaning</i>
18222089	Muhammad Adli Arindra	<ul style="list-style-type: none">• Mengerjakan laporan• Mengerjakan <i>feature engineering</i>• Mengembangkan model

VIII. REFERENSI

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