

Laporan Tugas Besar

IF3070 Dasar Artificial Intelligence



Dipersiapkan oleh:
Kelompok 17 K03

Muhammad Reffy Haykal	/ 18222103
M Rivaldi Mahari	/ 18222119
Regan Adiesta Mahendra	/ 18222122
Samuel Franciscus Togar H	/ 18222131

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI	2
BAB 1 Implementasi KNN	3
BAB 2 Penjelasan Singkat Implementasi Naive-Bayes	5
BAB 3 Cleaning dan Preprocessing	12
BAB 4 Perbandingan Hasil Prediksi Dari Algoritma	14
BAB 5 Kesimpulan	15
Kontribusi Kelompok	16
Referensi	17

BAB 1

Implementasi KNN

K-Nearest Neighbors (KNN) adalah algoritma bersifat non parametric yang berarti tidak membuat asumsi apa pun tentang distribusi data yang mendasarinya dan *lazy learning* yang berarti tidak menggunakan titik data training untuk membuat model. KNN digunakan untuk klasifikasi dan regresi. Dalam klasifikasi, KNN menentukan kelas sebuah sampel berdasarkan kelas dari k tetangga terdekatnya.

Berikut merupakan implementasi KNN :

```
def compute_distances(self, X):
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    dists = np.sqrt(np.sum(np.square(self.X_train), axis=1) +
                    np.sum(np.square(X), axis=1)[:, np.newaxis] -
                    2 * np.dot(X, self.X_train.T))
    return dists
```

Fungsi `compute_distance` untuk menghitung jarak antara dua vektor menggunakan formula euclidean distance :

$$\sqrt{\sum_{i=1}^n (x1i - x2i)^2}$$

Implementasi yang dilakukan menjadikan KNN sebagai class yang memiliki beberapa fungsi

```
def __init__(self):
    pass

def train(self, X, y):
    self.X_train = X
    self.y_train = y
```

Fungsi `__init__` untuk menginisialisasi instance dari kelas KNN dan `train` untuk menyimpan data latih `X_train` (fitur) dan `y_train` (label)

```
def predict(self, X, k=1, num_loops=0):
    if num_loops == 0:
        dists = self.compute_distances(X)
    else:
        raise ValueError('Invalid value %d for num_loops' % num_loops)
    return self.predict_labels(dists, k=k)
```

Fungsi predict_labels berparameter K (jumlah tetangga terdekat) dimulai dengan menghitung jarak euclidean antara sampel uji x menggunakan metode compute_distances. Lalu menggunakan hasil jarak untuk menentukan prediksi label dengan metode predict_labels

```
def predict_labels(self, dists, k=1):
    num_test = dists.shape[0]
    y_pred = np.zeros(num_test)
    for i in range(num_test):
        # Mendapatkan k tetangga terdekat
        closest_y = self.y_train[np.argsort(dists[i])[:k]]
        # Memilih label yang paling sering muncul
        y_pred[i] = np.argmax(np.bincount(closest_y))
    return y_pred
```

Fungsi predict_labels ini menentukan prediksi label untuk setiap sampel uji berdasarkan jarak yang dihitung. Untuk setiap sampel uji, indeks k tetangga terdekat diambil berdasarkan jarak terpendek lalu Ambil label dari tetangga tersebut. Lalu menentukan label yang paling sering muncul di antara k tetangga dengan np.bincount dan ambil argumen maksimum sebagai prediksi kelas.

BAB 2

Implementasi Naive-Bayes

Gaussian Naive Bayes (GNB) adalah teknik klasifikasi yang digunakan dalam pembelajaran mesin berdasarkan pendekatan probabilistik dan distribusi Gaussian. Gaussian Naive Bayes berasumsi bahwa setiap parameter, disebut juga fitur atau prediktor, mempunyai kapasitas independen dalam memprediksi variabel keluaran.

Berikut ini adalah penjelasan dari kode implementasi Naive-Bayes:

```
class GaussianNB:
    def __init__(self):
        self.classes = None
        self.mean = {}
        self.var = {}
        self.priors = {}
```

Pada bagian ini, dilakukan persiapan variabel-variabel penting untuk menyimpan informasi statistik dari data pelatihan. Terdapat 4 variabel yang akan dipakai yaitu:

- `self.classes`: Menyimpan daftar unik kelas dalam dataset
- `self.mean`: Menyimpan nilai rata-rata/ *mean* dari setiap fitur untuk masing-masing kelas
- `self.var`: Menyimpan variansi/*variance* dari setiap fitur untuk masing-masing kelas.
- `self.priors`: Menyimpan probabilitas awal/*prior* dari setiap kelas, yaitu rasio jumlah sampel kelas tersebut terhadap total sampel.

```
def fit(self, X, y):
    self.classes = np.unique(y)
    for cls in self.classes:
        X_cls = X[y == cls]
        self.mean[cls] = np.mean(X_cls, axis=0)
        self.var[cls] = np.var(X_cls, axis=0)
        self.priors[cls] = X_cls.shape[0] / X.shape[0]
```

Selanjutnya, dilakukan pelatihan oleh metode `fit()`. Metode ini menyiapkan semua statistik yang diperlukan untuk prediksi. Pertama, metode menerima data fitur di X dan label di Y sebagai input, lalu metode mengidentifikasi semua kelas unik dari label menggunakan `np.unique(y)` dan menyimpannya dalam `self.classes`. Kemudian, untuk setiap kelas/`cls`, data yang sesuai dengan kelas tersebut difilter dari

X. Dari data ini dihitung rata-rata dan variansi untuk setiap fitur, yang kemudian disimpan dalam **self.mean** dan **self.var**, dan *prior* untuk kelas tersebut dihitung dan disimpan dalam **self.priors**.

```
def _gaussian_pdf(self, class_idx, x):
    mean = self.mean[class_idx]
    var = self.var[class_idx]
    numerator = np.exp(-(x - mean) ** 2 / (2 * var))
    denominator = np.sqrt(2 * np.pi * var)
    return numerator / denominator
```

Setelah model dilatih, metode **_gaussian_pdf()** digunakan untuk menghitung Probability Density Function (PDF) dari distribusi Gaussian (Normal) untuk nilai fitur tertentu. Berikut ini adalah fungsinya:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- μ : Mean
- σ^2 : Variansi

Dalam metode ini, diterima input indeks kelas di *class_idx* dan nilai fitur di *X*. Berdasarkan mean dan variansi dari fitur yang disimpan di **self.mean** dan **self.var**, dihitung *likelihood* data terhadap kelas tertentu menggunakan rumus distribusi Gaussian.

```
def predict(self, X):
    predictions = []
    for x in X:
        posteriors = []
        for cls in self.classes:
            prior = np.log(self.priors[cls])
            likelihood = np.sum(np.log(self._gaussian_pdf(cls, x)))
            posterior = prior + likelihood
            posteriors.append(posterior)
        predictions.append(self.classes[np.argmax(posteriors)])
    return np.array(predictions)
```

Setelah semua statistik sudah siap, selanjutnya adalah tahap prediksi. Prediksi dilakukan oleh metode **predict()**, yang menerima data baru di X sebagai input. Untuk setiap sampel dalam data, metode ini menghitung probabilitas posterior untuk setiap kelas. Probabilitas posterior dihitung dengan menjumlahkan log prior dari kelas yang diambil dari **self.priors**, dan log *likelihood* dari setiap fitur yang dihitung menggunakan **_gaussian_pdf()** sebelumnya. Setelah menghitung posterior untuk semua kelas, kelas dengan probabilitas posterior tertinggi dipilih sebagai prediksi untuk sampel tersebut. Proses ini diulang untuk semua sampel dalam data baru, dan hasil prediksi dikembalikan sebagai *array*.

BAB 3

Cleaning dan Preprocessing

Cleaning dan Preprocessing adalah langkah persiapan data untuk analisis dan pengembangan model machine learning. Dua langkah ini digunakan untuk membangun model yang bersih, konsisten, dan dalam format yang baik.

Berikut ini adalah penjelasan dari kode implementasi Cleaning:

1. Handling missing data :

Imputasi

```
def handle_missing_data(dataframe):  
    mean_imputer = SimpleImputer(strategy='mean')  
    median_imputer = SimpleImputer(strategy='median')  
    mode_imputer = SimpleImputer(strategy='most_frequent')
```

i. Mean (digunakan untuk kolom yang distribusi normal)

```
if Mean_col:  
    dataframe[Mean_col]=  
mean_imputer.fit_transform(dataframe[Mean_col])
```

ii. Median (digunakan untuk kolom yang distribusi penyebara yang luas)

```
if Median_col:  
    dataframe[Median_col]=  
median_imputer.fit_transform(dataframe[Median_col])
```

iii. Modus (digunakan untuk kolom yang bersifat kategorikal)

```
if Modus_col:  
    dataframe[Modus_col]=  
mode_imputer.fit_transform(dataframe[Modus_col])
```

iv. SimpleImputer (digunakan untuk menangani nilai hilang dalam dataset)

```
constant_fill_columns=  
dataframe.columns[dataframe.isnull().any()].tolist()  
dataframe[constant_fill_columns]=  
dataframe[constant_fill_columns].fillna(0)  
return dataframe
```

2. Dealing with Outliers

```
for col in num_cols:  
    # Menghitung Q1, Q3, dan IQR untuk mendeteksi outlier  
    Q1 = X_train[col].quantile(0.25)  
    Q3 = X_train[col].quantile(0.75)  
    IQR = Q3 - Q1  
    lower_bound = Q1 - 1.5 * IQR
```



```

upper_bound = Q3 + 1.5 * IQR

# Clipping outlier
X_train[col] = np.where(X_train[col] < lower_bound, lower_bound,
X_train[col])
X_train[col] = np.where(X_train[col] > upper_bound, upper_bound,
X_train[col])

# Imputasi dengan nilai median untuk outlier
median_value = X_train[col].median()
X_train[col] = np.where(X_train[col] < lower_bound,
median_value, X_train[col])
X_train[col] = np.where(X_train[col] > upper_bound,
median_value, X_train[col])

# Imputasi dengan nilai mean untuk outlier
median_value = X_train[col].mean()
X_train[col] = np.where(X_train[col] < lower_bound,
median_value, X_train[col])
X_train[col] = np.where(X_train[col] > upper_bound,
median_value, X_train[col])

# Imputasi dengan nilai modus untuk outlier
median_value = X_train[col].mode()
X_train[col] = np.where(X_train[col] < lower_bound,
median_value, X_train[col])
X_train[col] = np.where(X_train[col] > upper_bound,
median_value, X_train[col])

# Transformasi log untuk kolom yang skewed
if skew(X_train[col]) > 1:
    X_train[col] = np.log1p(X_train[col])

print("Data Setelah Penanganan outlier:")
print(X_train.head(16))

```

3. Remove Duplicate

```

duplicate_count = X_train.duplicated().sum()
print(f"Jumlah baris duplikat sebelum dihapus: {duplicate_count}")

```

```

X_train = X_train.drop_duplicates()

duplicate_count_after = X_train.duplicated().sum()
print(f"Jumlah baris duplikat setelah dihapus: {duplicate_count_after}")

print("Ukuran dataset setelah penghapusan duplikasi:", X_train.shape)

```

4. Feature Engineering

- a. `apply_feature_engineering`: fungsi ini menambah fitur baru ke dataset dengan melakukan transformasi.

```

def apply_feature_engineering(df):
    df['LogURLLength'] = np.log1p(df['URLLength'])
    df['URLToDomainLengthRatio'] = df['URLLength'] / df['DomainLength']
    domain_length_bins = [0, 10, 20, 30, 40, np.inf]
    domain_length_labels = ['short', 'medium', 'long', 'very_long', 'extremely_long']
    df['DomainLengthBinned'] = pd.cut(df['DomainLength'], bins=domain_length_bins, labels=domain_length_labels)
    . . . . .
    return df

```

- b. `handle_infinity_values`: fungsi ini mengganti nilai *inf* dan *-inf* di dataset dengan NaN untuk menghindari kesalahan pemrosesan data.

```

def handle_infinity_values(df):
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    return df

```

Data Preprocessing

1. Feature Scaling

```

class FeatureScaler(BaseEstimator, TransformerMixin):
    def __init__(self, scaler_type='minmax'):
        # Menyimpan jenis scaler yang digunakan
        self.scaler_type = scaler_type
        # Pilih scaler berdasarkan input
        if self.scaler_type == 'minmax':
            self.scaler = MinMaxScaler()
        elif self.scaler_type == 'standard':
            self.scaler = StandardScaler()

```

```

        else:
            raise ValueError("scaler_type harus 'minmax' atau 'standard'")

    def fit(self, X, y=None):
        # Melakukan fitting pada data
        self.scaler.fit(X)
        return self

    def transform(self, X):
        # Melakukan transformasi pada data yang sudah difit
        X_scaled = self.scaler.transform(X)
        # Mengembalikan hasil dalam format DataFrame dengan nama kolom yang sama dengan X
        return pd.DataFrame(X_scaled, columns=X.columns)

```

2. Encoding Categorical Variables

```

class CategoricalEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, encoding_type='onehot'):
        self.encoding_type = encoding_type
        self.encoder = None

    def fit(self, X, y=None):
        if self.encoding_type == 'onehot':
            self.encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
            self.encoder.fit(X) # Changed sparse to sparse_output
        elif self.encoding_type == 'label':
            self.encoder = {col: LabelEncoder().fit(X[col]) for col in X.columns}
        return self

    def transform(self, X):
        if self.encoding_type == 'onehot':
            # Menerapkan OneHotEncoder dan mengembalikan DataFrame
            encoded_array = self.encoder.transform(X)
            columns = self.encoder.get_feature_names_out(input_features=X.columns)
            return pd.DataFrame(encoded_array, columns=columns, index=X.index)

```

```

        elif self.encoding_type == 'label':
            # Menerapkan LabelEncoder untuk setiap kolom
            return X.apply(lambda col:
self.encoder[col.name].transform(col))

```

3. Handling Imbalanced Classes

```

class ImbalanceHandler(BaseEstimator, TransformerMixin):
    def __init__(self, method='smote'):
        if method == 'smote':
            self.handler = SMOTE(random_state=42)
        elif method == 'undersampling':
            self.handler = RandomUnderSampler(random_state=42)
        else:
            raise ValueError("Metode yang valid adalah 'smote' atau
'undersampling'.")

    def fit_resample(self, X, y):
        X_res, y_res = self.handler.fit_resample(X, y)
        print(f"Distribusi kelas setelah penyeimbangan:
{Counter(y_res)}") # Fixed indentation
        return X_res, y_res

```

BAB 4

Perbandingan Hasil Prediksi Dari Algoritma

Akurasi hasil prediksi naive-bayes:

```
[ ] from sklearn.metrics import accuracy_score, classification_report
    #prediksi pada validation set
    y_pred = model.predict(X_val)

    #evaluasi hasil prediksi
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Akurasi pada validation set: {accuracy:.4f}")

    print("\nlaporan evaluasi model:")
    print(classification_report(y_val, y_pred))
```

➡ Akurasi pada validation set: 0.4860

laporan evaluasi model:					
	precision	recall	f1-score	support	
0	0.48	0.47	0.48	492	
1	0.49	0.50	0.50	508	
accuracy			0.49	1000	
macro avg	0.49	0.49	0.49	1000	
weighted avg	0.49	0.49	0.49	1000	

Hasil prediksi menunjukkan nilai 0.48 atau 48%, di gambar terlihat bahwa model Naive Bayes kesulitan dalam menangkap pola pada data, karena asumsi independensi antar fitur yang tidak sesuai.


Akurasi hasil prediksi KNN:

```
# memilih label yang paling sering muncul
y_pred[i] = np.argmax(np.bincount(closest_y))
return y_pred

knn = KNearestNeighbor()
knn.train(X_train, y_train)

# Prediksi menggunakan X_train
y_pred = knn.predict(X_train, k=3) # Menggunakan k=3

# Menghitung akurasi
accuracy = np.mean(y_pred == y_train)
print(f"Label Asli: {y_train}")
print(f"Prediksi: {y_pred}")
print(f"Akurasi: {accuracy * 100:.2f}%")
```

 <ipython-input-40-7fd9916549e1>:23: RuntimeWarning: invalid value encountered in sqrt
dists = np.sqrt(np.sum(np.square(self.X_train), axis=1) +
Label Asli: [1 0 0 ... 1 1 0]
Prediksi: [0. 1. 0. ... 1. 1. 0.]
Akurasi: 66.79%

Hasil Prediksi menggunakan KNN akurasi 66,79% hal ini menunjukkan bahwa model cukup baik, namun ada ruang untuk perbaikan dengan kemungkinan penyebab tidak maksimal data mungkin tidak terstandarisasi dengan benar .

BAB 5

Kesimpulan

Secara keseluruhan, algoritma **K-Nearest Neighbors (KNN)** cenderung memberikan hasil yang baik pada dataset dengan pola distribusi yang tidak terlalu kompleks, terutama jika nilai parameter jumlah tetangga ditentukan secara tepat. Namun, KNN memiliki kelemahan dalam hal waktu komputasi, terutama pada dataset berukuran besar, karena algoritma ini membutuhkan perhitungan jarak untuk setiap titik data. Sebaliknya, **Naive Bayes** unggul dalam efisiensi waktu karena pendekatannya yang berbasis probabilistik, meskipun akurasinya dapat menurun jika terdapat korelasi antar fitur, mengingat asumsi independensi antar fitur pada algoritma ini.

Berdasarkan hasil evaluasi, implementasi menggunakan *library* dinilai lebih efektif dalam pengaturan parameter dan penghitungan, menghasilkan performa yang lebih optimal dibandingkan implementasi dari *scratch*. Pada dataset yang kami gunakan, KNN memiliki keunggulan dalam metrik akurasi, sementara Naive Bayes lebih unggul dalam metrik recall, terutama ketika kelas target tidak seimbang.

KONTRIBUSI KELOMPOK

Nama anggota	NIM	Tugas
Muhammad Reffy Haykal	18222103	<ul style="list-style-type: none">- Data Splitting- Preprocessing (Feature Engineering)- Compile Preprocessing Pipeline- Error Analysis
M Rivaldi Mahari	18222119	<ul style="list-style-type: none">- Data Cleaning dan Preprocessing- KNN
Regan Adiesta Mahendra	18222122	<ul style="list-style-type: none">- KNN
Samuel Franciscus Togar Hasurungan	18222131	<ul style="list-style-type: none">- Modelling and Validation (Naive Bayes)- Improvements- Submission- Perbandingan Hasil Prediksi Algoritma- Kesimpulan- Setup Laporan

REFERENSI

Martins, C. (2023, November 2). *Gaussian naive Bayes explained with Scikit-Learn*. Built In. <https://builtin.com/artificial-intelligence/gaussian-naive-bayes>

[Spesifikasi Tugas Besar 2 IF3070 Dasar Inteligensi Artifisial 2024/2025](#)

PHIUSIIL Phishing URL (Website) - UCI Machine Learning Repository. (n.d.). <https://archive.ics.uci.edu/dataset/967/phiusiil+phishing+url+dataset>

1.9. Naive Bayes. (n.d.). Scikit-learn. https://scikit-learn.org/1.5/modules/naive_bayes.html

1.6. Nearest neighbors. (n.d.). Scikit-learn. <https://scikit-learn.org/1.5/modules/neighbors.html>