

Tugas Besar II

IF3170 Inteligensi Buatan

KNN and Naive Bayes Algorithm



Disusun oleh:

13521087 Razzan Daksana Yoni

13521089 Kenneth Ezekiel Suprantonni

13521095 Muhamad Aji Wibisono

13521101 Arsa Izdiyar Islam

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

DAFTAR ISI

DAFTAR ISI	2
BAB I	
PENDAHULUAN	3
1.1. Latar belakang	3
1.2. Implementasi	3
1.3. Spesifikasi Pengerjaan Tugas	3
BAB II	
ISI	5
2.1. Data Preparation	5
2.1.1. Pemilihan Fitur	5
2.1.1.1. Fitur Nominal	6
2.1.1.2. Fitur Numerik	7
2.1.2. Penggabungan Fitur	8
2.1.3. Pembersihan Data	9
2.1.4. Data Splitting	10
2.2. Penjelasan Algoritma	10
2.2.1. KNN	10
2.2.2. Naive Bayes	10
2.3. Implementasi	11
2.3.1. KNN	11
2.3.2. Naive Bayes	13
2.3.3. Implementasi Menggunakan Scikit-Learn	16
BAB III	
EXPERIMENT	17
3.1. Eksperimen	17
3.1.1. KNN vs. KNN Scikit-Learn	17
3.1.2. Naive Bayes vs. Naive Bayes Scikit-Learn	17
3.2. Analisis dan Insight	17
3.2.1. KNN vs. KNN Scikit-Learn	17
3.2.2. Naive Bayes vs. Naive Bayes Scikit-Learn	18
BAB IV	
PEMBAGIAN TUGAS	19
BAB V	
LAMPIRAN	20

BAB I

PENDAHULUAN

1.1. Latar belakang

Setelah melakukan EDA dan pemrosesan, pada tugas kecil 2, penulis diminta untuk mengimplementasikan algoritma pembelajaran mesin yang telah penulis pelajari di kuliah, yaitu KNN dan Naive-Bayes. Data yang digunakan sama seperti tugas kecil 2. Latihlah model dengan menggunakan data latih, kemudian validasi hasil dengan menggunakan data validasi untuk mendapatkan insight seberapa baik model melakukan generalisasi.

1.2. Implementasi

1. **[Poin 40]** Algoritma KNN.
2. **[Poin 50]** Algoritma Naive-Bayes.
3. **[Poin 10]** Implementasi algoritma 1 dan 2 dengan pustaka seperti *scikit-learn*
4. **[Bonus]** Kaggle submission pada [link berikut](#).

Perhatikan bahwa untuk **poin 1 dan 2, implementasi dilakukan *from scratch***. Pustaka yang boleh digunakan hanya pustaka utilitas seperti numpy dan pandas. **Model harus bisa di-save dan di-load**. Implementasinya dibebaskan (misal menggunakan .txt, .pkl, dll).

Untuk bonus, nilai diberikan berdasarkan ranking *leaderboard* Kaggle yang dirincikan sebagai berikut:

- Rank 1-3 = 10 poin
- Rank 4-5 = 5 poin
- Rank 6-10 = 3 poin
- Submit saja = 1 poin

Dalam leaderboard, gunakan nama kelompok. Identifikasi dilakukan berdasarkan nama kelompok, jadi cukup 1 orang saja yang berada dalam tim Kaggle.

1.3. Spesifikasi Pengerjaan Tugas

1. Tugas dikerjakan berkelompok, dan 1 kelompok terdiri atas 4 mahasiswa (gabungan 2 kelompok tugas kecil).

2. Tugas dikumpulkan seperti repository yang sudah ada. Hanya saja, ditambah dengan folder docs yang berisi laporan. Cantumkan repository Github private dengan menambahkan akses pada akun-akun asisten
3. Laporan dalam format .pdf berisi informasi sbb:
4. Penjelasan singkat implementasi KNN.
5. Penjelasan singkat implementasi Naive-Bayes.
6. Perbandingan hasil prediksi dari algoritma yang diimplementasikan dengan hasil yang didapatkan dengan menggunakan pustaka. Jelaskan insight yang kalian dapatkan dari perbandingan tersebut.
7. Catatan: perbandingan hasil dapat menggunakan metrics precision, recall, atau akurasi.
8. Jika melakukan submisi Kaggle, jelaskan singkat pemrosesan apa saja yang dilakukan.
9. Kontribusi setiap anggota dalam kelompok.
10. Penamaan file yang dikumpulkan: Tubes2_[nama kelompok].pdf/zip (misal: Tubes2_ITBwibu.pdf/zip). Pengumpulan hanya dilakukan oleh satu orang saja.
11. Pengumpulan yang terlambat tidak diperbolehkan, batas akhir adalah hari Rabu, 29 November 2023 pukul 23.00 WIB.
12. Dilarang bekerja sama antar kelompok, kecurangan akan berakibat nilai E pada mata kuliah IF3170.
13. Mahasiswa wajib mengisi daftar kelompok. Apabila tidak mengisi, nilai berpotensi tidak tercatat sehingga kosong.

BAB II

ISI

2.1. Data Preparation

2.1.1. Pemilihan Fitur

Sebelum dilakukan pengembangan model, perlu dilakukan pemilihan terhadap fitur untuk meningkatkan kinerja model, mengurangi kompleksitas, dan menghindari overfitting.

Pada dataset yang diberikan, terdapat beberapa fitur seperti berikut dengan price_range adalah kolom target:

Tabel 2.1.1.1. Tabel setiap jenis fitur yang terdapat pada data

Nama Fitur	Jenis Data
Battery_Power	Numerik
Clock_speed	Numerik
fc	Numerik
int_memory	Numerik
m_dep	Numerik
mobile_wt	Numerik
n_cores	Numerik
pc	Numerik
px_height	Numerik
px_width	Numerik
ram	Numerik
sc_h	Numerik
sc_w	Numerik
talk_time	Numerik

blue	Nominal
dual_sim	Nominal
four_g	Nominal
three_g	Nominal
touch_screen	Nominal
wifi	Nominal
price_range	Ordinal

Penulis menggunakan beberapa cara dan dasar teori untuk menentukan signifikansi dari masing - masing kolom.

2.1.1.1. Fitur Nominal

Pada kolom - kolom yang berjenis nominal, penulis menggunakan Chi-Squared Test untuk mencari p-value dari kolom tersebut terhadap kolom target, yaitu price_range. Chi-Squared Test bekerja dengan menghitung nilai statistik Chi-Squared lalu melakukan perhitungan p-value dari nilai Chi-Squared yang sudah diperoleh dengan derajat kebebasan sesuai dengan jumlah kolom yang ingin dicari. P-value akan merepresentasikan keterkaitan antara kolom yang dipilih dengan kolom target, dengan p-value < 0.05 menunjukkan keterkaitan yang erat.

Pada implementasinya p-value menggunakan Chi-Squared Test dicari dengan fungsi buatan scipy seperti berikut:

```
# check for nominal columns (boolean columns) using contingency table
from scipy.stats import chi2_contingency

def show_chi_contingency(col):
    print("column:", col)
    contingency = pd.crosstab(index=df[col], columns=df['price_range'], normalize='index')
    res = chi2_contingency(contingency)
    print("p =", res[1])
    return contingency
```

Gambar 2.1.1.1. Fungsi untuk mencari p-value dengan Chi-Squared Test

Didapatkan p-value setiap kolom nominal adalah seperti berikut

Tabel 2.11.1.1. Tabel p-value dari Chi-Squared Test untuk setiap fitur nominal

Nama Fitur	p-value
blue	0.9999306682714457
dual_sim	0.9999360384691366
four_g	0.9999600883636923
three_g	0.9997915321532349
touch_screen	0.99993024036726
wifi	0.9999157156926507

Dengan semua p-value di mendekati 1, disimpulkan bahwa tidak ada fitur nominal yang berpengaruh signifikan terhadap kolom target dan tidak ada yang dapat digunakan dalam pencarian

2.1.1.2. Fitur Numerik

Pada kolom - kolom yang berjenis numerik, penulis menggunakan Analysis of Variance (ANOVA) untuk mencari p-value dari kolom tersebut terhadap kolom target, yaitu price_range. P-value pada hasil Analysis of Variance menentukan seberapa variasi dari suatu kolom akan mempengaruhi variasi dari kolom target dengan p-value < 0.05 menunjukkan variasi yang tinggi. Pada implementasinya penulis menggunakan library sklearn.feature_selection, yaitu f_classif

Didapatkan p-value setiap kolom numerik adalah seperti berikut

Tabel 2.11.2.1 Tabel p-value dari Analysis of Variance Test untuk setiap fitur numerik

Nama Fitur	p-value
Battery_Power	1.199878164181477e-12
Clock_speed	0.947345457999534
fc	0.9844593072179413
int_memory	0.02193652500628816

m_dep	0.4192414002698195
mobile_wt	0.0024343700642573767
n_cores	0.3031233402282094
pc	0.9475768095830291
px_height	3.634821713583304e-10
px_width	1.4591634777808455e-11
ram	0.0
sc_h	0.08589662330366395
sc_w	0.20583807634591528
talk_time	0.44294679598239306

Berdasarkan Analysis of Variance terhadap fitur-fitur yang bertipe numerik, nilai p dari dari fitur-fitur berikut lebih tinggi dari 0.05 dan kurang baik untuk digunakan

clock_speed, fc, m_dep, n_cores, pc, sc_h, sc_w, dan talk time

Sementara fitur-fitur berikut memiliki nilai p lebih rendah dari 0.05 dan baik untuk digunakan

battery_power, int_memory, mobile_wt, px_height, px_width, ram

2.1.2. Penggabungan Fitur

Selain memilih fitur, penggabungan fitur yang berkorelasi tinggi dan/atau merepresentasikan hal yang serupa juga dapat dilakukan untuk meningkatkan kinerja model. Pencarian dilakukan algoritma seperti berikut:

```
# Detecting correlation between each other
df_high_correlation = df.drop('price_range', axis=1).corr().abs()

high_correlation_pairs = []
high_correlation_threshold = 0.4

for index, row in df_high_correlation.iterrows():
    for column, value in row.items():
        if value > high_correlation_threshold and column > index:
            high_correlation_pairs.append((index, column))
```

Gambar 2.1.2.1. Algoritma pencarian fitur yang berkorelasi tinggi

Penulis menyimpulkan untuk bahwa menggabungkan sc_w dan sc_h serta px_height dan px_width mungkin dilakukan karena korelasi cukup tinggi dan hal yang direpresentasikan cukup mirip.

Pada kesimpulannya, penulis memilih fitur battery_power, int_memory, mobile_wt, px_height, px_width, ram dalam pengembangan model.

2.1.3. Pembersihan Data

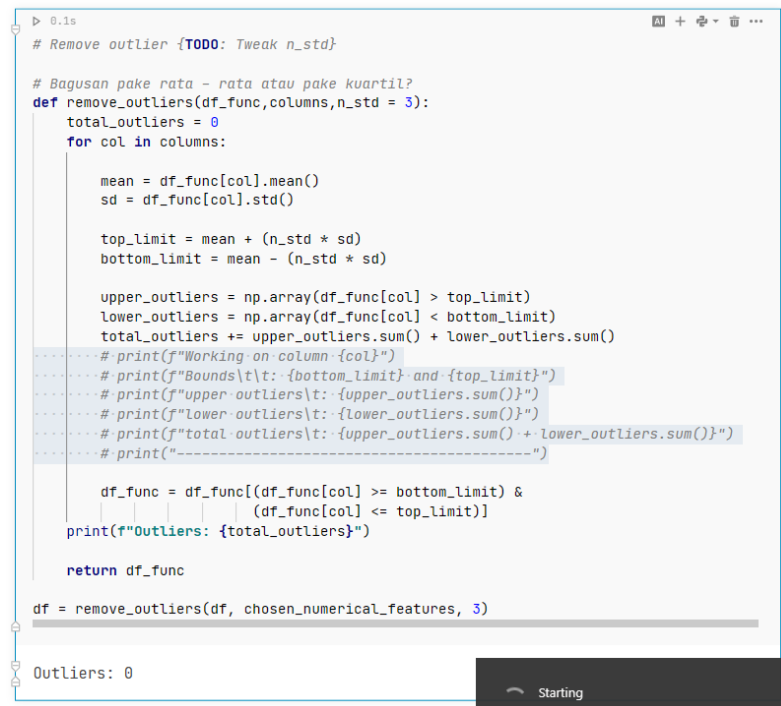
Setelah melakukan pemilihan dan penggabungan fitur, penulis melakukan pembersihan terhadap data tersebut, yaitu menghilangkan duplikat. Penulis menemukan pada data tersebut tidak ada duplikat.

```
# Check for duplicates just to be sure
print(f"Duplicates: {df.duplicated().sum()}")
```

Duplicates: 0

Gambar 2.1.3.1. Hasil pencarian duplikasi pada data dengan fitur yang sudah dipilih

Penulis lalu melakukan pembersihan pada data yang merupakan outlier pada tiap kolom yang dipilih. Penulis menemukan pada data tersebut tidak ada duplikat.



```
0.1s
# Remove outlier {TODO: Tweak n_std}

# Bagus pake rata - rata atau pake kuartil?
def remove_outliers(df_func, columns, n_std = 3):
    total_outliers = 0
    for col in columns:
        mean = df_func[col].mean()
        sd = df_func[col].std()

        top_limit = mean + (n_std * sd)
        bottom_limit = mean - (n_std * sd)

        upper_outliers = np.array(df_func[col] > top_limit)
        lower_outliers = np.array(df_func[col] < bottom_limit)
        total_outliers += upper_outliers.sum() + lower_outliers.sum()
        # print(f"Working on column {col}")
        # print(f"Bounds\t\t: {bottom_limit} and {top_limit}")
        # print(f"upper outliers\t: {upper_outliers.sum()}")
        # print(f"lower outliers\t: {lower_outliers.sum()}")
        # print(f"total outliers\t: {upper_outliers.sum() + lower_outliers.sum()}")
        # print("-----")

    df_func = df_func[(df_func[col] >= bottom_limit) &
                      (df_func[col] <= top_limit)]
    print(f"Outliers: {total_outliers}")

    return df_func

df = remove_outliers(df, chosen_numerical_features, 3)

Outliers: 0
Starting
```

Gambar 2.1.3.2. Hasil pembersihan outlier pada data dengan fitur yang sudah dipilih

2.1.4. Data Splitting

Untuk pengetesan model, penulis sudah diberikan dua data yang telah ditentukan sebagai data training dan data testing dari spesifikasi tugas besar.

2.2. Penjelasan Algoritma

2.2.1. KNN

Algoritma K-Nearest Neighbors atau KNN adalah supervised learning classifier, yang menggunakan pendekatan untuk menentukan klasifikasi atau prediksi tentang pengelompokan dari sebuah data. Pendekatan klasifikasi ini dilakukan dengan mencari jarak atau distance rata-rata tetangga terdekat sebanyak k. Untuk menentukan jarak tetangga terdekat pendekatan ini, penulis menggunakan Euclidean Distance. Klasifikasi pada algoritma ini dilakukan dengan mencari jarak data yang dicari dengan euclidean setiap data pada dataset acuan. Pencarian jarak dilakukan dengan menambahkan 1 pada tipe data nominal dan Euclidean Distance pada data numerik. Lalu, dicari data dengan distance sedekat mungkin sebanyak k. Setelah itu klasifikasi ditentukan dengan jumlah data target yang paling banyak.

2.2.2. Naive Bayes

Algoritma Naive Bayes berlandaskan pada prinsip dasar probabilitas Bayes, menggunakan pendekatan yang dianggap 'naif'. Pendekatan ini mengasumsikan independensi kondisional antara setiap pasangan fitur dalam model. Dalam konteks ini, "naif" merujuk pada asumsi bahwa keberadaan suatu fitur dalam suatu kelas tidak dipengaruhi oleh keberadaan fitur lain.

Algoritma ini terbagi menjadi dua tahap, yaitu tahap pembelajaran (*learning*) dan tahap klasifikasi (*classification*).

Tahap Pembelajaran

Dalam tahap ini, dihitung frekuensi kemunculan setiap nilai atribut untuk setiap label kelas. Dari frekuensi yang didapatkan, dihitung Probabilitas kondisional $P(a_i | v_j)$, yaitu kemungkinan setiap nilai a_i dari atribut a untuk seluruh kelas label kelas v_j .

Tahap Klasifikasi

Dalam tahap ini, dilakukan prediksi, dimana algoritma menggabungkan probabilitas dari setiap label dengan mempertimbangkan atribut data yang akan diprediksi. Prediksi dilakukan dengan mengkomputasi proyeksi dari probabilitas atribut dari semua data pada tahap pembelajaran. Selanjutnya akan dihitung probabilitas $P(v_j | a_1, a_2, \dots, a_n)$ untuk setiap label kelas v dan dipilih label dengan nilai probabilitas tertinggi sebagai label dari data tersebut.

Model

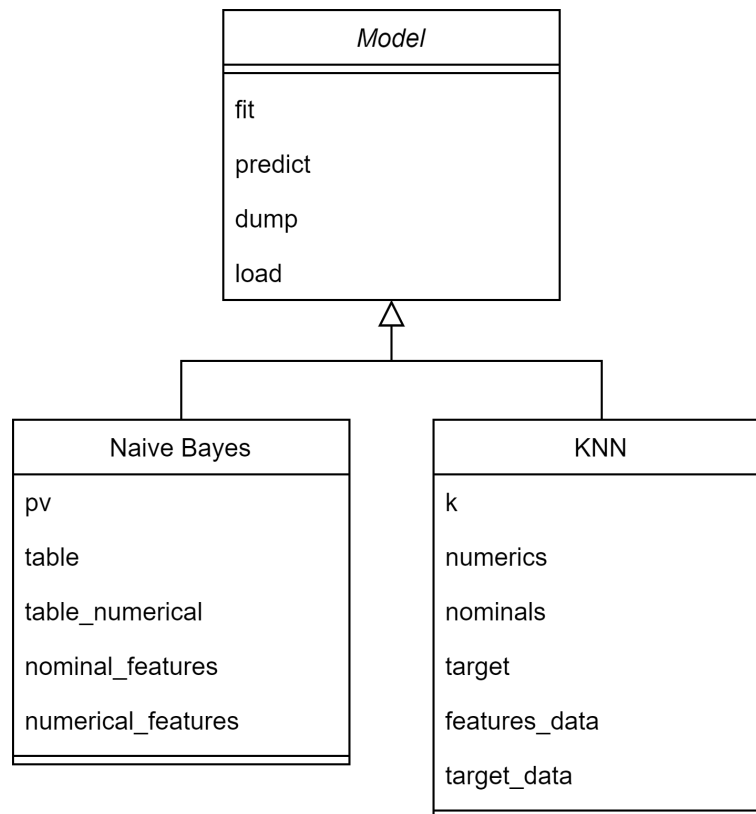
Model ini menggunakan dua jenis probabilitas:

$P(v_j)$: Probabilitas prior dari kelas v_j .

$P(a_i | v_j)$: Probabilitas kemunculan nilai a_i dari atribut a ketika kelas adalah v_j .

2.3. Implementasi

Implementasi digunakan sebuah struktur model seperti berikut:



Gambar 2.3.1 Struktur Model

2.3.1. KNN

```
class KNN(Model):
    def __init__(self, nominals, numericals, target, k=5):
        self.k = k
        self.numerics = numericals
        self.nominals = nominals
        self.target = target
        self.features_data = None
        self.target_data = None

    def fit(self, x_data, y_data):
        self.features_data = x_data
        self.target_data = y_data
```

```

def predict(self, x_data):
    if self.features_data is None or self.target_data is None:
        raise Exception("Invalid data")

    predictions = []
    for i, row in x_data.iterrows():
        # Get the distance between the data
        ## Nominal
        nom_dist = np.sum(row[self.nominals] !=
self.features_data[self.nominals], axis=1)
        ## Numeric (using Euclidean)
        num_dist = np.sqrt(np.sum(np.square(row[self.numerics] -
self.features_data[self.numerics]), axis=1))

        distances = nom_dist + num_dist

        # Get the k nearest neighbors
        neighbors = np.argsort(distances)[:self.k]

        # Get the most common class
        classes = Counter(self.target_data.iloc[neighbors])
        max_class = classes.most_common(1)[0][0]

        predictions.append([i, max_class])
    return pd.DataFrame(predictions, columns=['id', 'price_range'])

def __str__(self):
    return "KNN(k={})".format(self.k)

knn = KNN(chosen_nominal_features, chosen_numerical_features, "price_range")
knn.fit(x_train, y_train)

pred = knn.predict(x_test)
accuracy = accuracy_score(y_test, pred)
precision = precision_score(y_test, pred, average='weighted')
recall = recall_score(y_test, pred, average='weighted')

print("-----")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

```

2.3.2. Naive Bayes

```
def calculate_probability(x, mean, std):  
    """  
    Calculate the probability of a value in a normal distribution.  
  
    Parameters:  
    - x: The value for which to calculate the probability.  
    - mean: The mean of the normal distribution.  
    - std: The standard deviation of the normal distribution.  
  
    Returns:  
    - probability: The probability of the given value in the distribution.  
    """  
    exponent = -((x - mean) ** 2) / (2 * std ** 2)  
    coefficient = 1 / (math.sqrt(2 * math.pi) * std)  
    probability = coefficient * math.exp(exponent)  
    return probability  
  
# Model Naive Bayes  
class NaiveBayes(Model):  
  
    def __init__(self, nominals, numericals):  
        self.pv = {}  
        self.table = {}  
        self.table_numerical = {}  
        self.nominal_features = nominals  
        self.numerical_features = numericals  
  
    def fit(self, x_data, y_data):  
        # Get the probability P(Vi)  
        target = 'price_range'  
        for i in range(y_data.min(), y_data.max()+1):  
            self.pv[i] = len(y_data[y_data == i])/len(y_data)  
  
        data = x_data.join(y_data)  
  
        # Get the probability of value a given vi for every nominal feature  
        for feature in self.nominal_features:
```

```

        values = x_data[feature].unique()
        for j in values:
            j_data = data[data[feature] == j]
            for k in range(y_data.min(), y_data.max()+1):
                k_data = j_data[j_data[target] == k]
                self.table[(feature,j,k)] = len(k_data) / len(data) /
self.pv[k]

        # Get the gaussian distribution of a value given vi for every
numerical feature
        for feature in self.numerical_features:
            for k in range(y_data.min(), y_data.max() + 1):
                k_data = data[data[target] == k]
                if not k_data.empty:
                    mean = k_data[feature].mean()
                    std = k_data[feature].std()
                    self.table_numerical[(feature, k)] = {'mean': mean,
'std': std}

def predict(self, x_data):
    predictions = []
    for i, row in x_data.iterrows():
        class_probability = []
        for k in range(min(self.pv.keys()), max(self.pv.keys()) + 1):
            p = self.pv[k]
            for feature in x_data.columns:
                if feature in self.numerical_features:
                    value = row[feature]
                    mean = self.table_numerical[(feature, k)]['mean']
                    std = self.table_numerical[(feature, k)]['std']
                    p *= calculate_probability(value, mean, std)
                else:
                    value = row[feature]
                    if (feature, value, k) in self.table:
                        p *= self.table[(feature, value, k)]

            class_probability.append(p)
        predictions.append([i, np.argmax(class_probability)])
    return pd.DataFrame(predictions, columns=['id', 'price_range'])

```



```

def __str__(self):
    string = "P(V):\n"
    for key, value in self.pv.items():
        string += f"P(V{key}): {value}\n"
    string += "Table:\n"
    for key, value in self.table.items():
        string += f"{key[0]} P({key[1]} | {key[2]}): {value}\n"
    string += "Numerical Table:\n"
    for key, value in self.table_numerical.items():
        string += f"P({key[0]} | {key[1]}): {value}\n"
    return string

nb = NaiveBayes(chosen_nominal_features, chosen_numerical_features)
nb.fit(x_train, y_train)
print(str(nb))

df_pred = nb.predict(x_test)
print(df_pred)
accuracy = accuracy_score(y_test, df_pred['price_range'])
precision = precision_score(y_test, df_pred['price_range'],
                             average='weighted')
recall = recall_score(y_test, df_pred['price_range'], average='weighted')

print("-----")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

```

2.3.3. Implementasi Menggunakan Scikit-Learn

```

print("#-1-KNN-----")
knn_library = KNeighborsClassifier()

knn_library.fit(a, y_train)
pred = knn_library.predict(d)

accuracy = accuracy_score(y_test, pred)
precision = precision_score(y_test, pred, average='weighted')

```

```
recall = recall_score(y_test, pred, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

print("#-2-Naive-Bayes-----")
nb_library = GaussianNB()

nb_library.fit(a, y_train)
pred = nb_library.predict(d)

accuracy = accuracy_score(y_test, pred)
precision = precision_score(y_test, pred, average='weighted')
recall = recall_score(y_test, pred, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

BAB III

EXPERIMENT

3.1. Eksperimen

3.1.1. KNN vs. KNN Scikit-Learn

Tabel 3.1.1.1 Hasil prediksi KNN vs. KNN Scikit-Learn

KNN	Accuracy: 0.925 Precision: 0.9253007631703358 Recall: 0.925
KNN Scikit-Learn	Accuracy: 0.925 Precision: 0.9253007631703358 Recall: 0.925

3.1.2. Naive Bayes vs. Naive Bayes Scikit-Learn

Tabel 3.1.2.1 Hasil prediksi Naive Bayes vs. Naive Bayes Scikit-Learn

Naive Bayes	----- Accuracy: 0.7933333333333333 Precision: 0.7946776384842273 Recall: 0.7933333333333333
Naive Bayes Scikit-Learn	Accuracy: 0.7933333333333333 Precision: 0.7946776384842273 Recall: 0.7933333333333333

3.2. Analisis dan Insight

3.2.1. KNN vs. KNN Scikit-Learn

Pada algoritma KNN, hasil yang didapatkan menggunakan algoritma yang dibuat oleh penulis dan menggunakan library adalah sama persis dan waktu yang dibutuhkan untuk eksekusi tidak berbeda jauh.

3.2.2. Naive Bayes vs. Naive Bayes Scikit-Learn

Pada algoritma Naive Bayes, hasil yang didapatkan menggunakan algoritma yang dibuat oleh penulis dan menggunakan library adalah sama persis dan waktu yang dibutuhkan untuk eksekusi tidak berbeda jauh.

BAB IV

PEMBAGIAN TUGAS

NIM	Nama	Pekerjaan
13521087	Razzan Daksana Yoni	KNN
13521089	Kenneth Ezekiel Supranton	Naive Bayes, Kaggle Submission
13521095	Muhamad Aji Wibisono	Scikit-learn, Feature selection
13521101	Arsa Izdihar Islam	Feature selection

BAB V

LAMPIRAN

Datalore:

<https://datalore.jetbrains.com/notebook/Tve7D5SbxralruE3sP0tUj/atWxu2PxY5ENPF98gE0O06/>

Github:

https://github.com/KenEzekiel/Tubes2_ArsalIntelligence.git