

Laporan Tugas Besar Kecerdasan Buatan II

Diajukan untuk memenuhi Tugas Besar

Mata Kuliah Kecerdasan Buatan



Disusun Oleh :

Glorious Satria Dhamang Aji

1302213105, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas
Telkom,

glorioussatria@student.telkomuniversity.ac.id

Kevin

1302210019, S1 Rekayasa Perangkat Lunak, Fakultas Informatika, Universitas
Telkom,

kevinwijaya@student.telkomuniversity.ac.id

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

TELKOM UNIVERSITY

KOTA BANDUNG

2023

Deskripsi Tugas.....	3
Pengenalan Metode.....	3
KNN.....	4
Naïve Bayes.....	5
Proses yang diimplementasikan ke dalam program.....	6
Membaca data latih/uji.....	6
Naïve Bayes.....	6
KNN.....	7
Pelatihan atau training model.....	8
Naïve Bayes.....	8
Data Preprocessing.....	8
Partisi Data.....	8
Normalisasi.....	8
Standarisasi.....	9
Model Generation.....	10
Class Separation.....	10
Data Summarization.....	11
Gaussian Equation.....	12
Probabilities Calculation.....	13
Prediction.....	14
KNN.....	14
Data Preprocessing.....	14
Normalisasi.....	14
Model Generation.....	15
Euclidean Distance.....	15
Manhattan Distance.....	16
Code Translation.....	17
Prediction.....	18
Pengujian atau testing model.....	19
Data Folding.....	19
Naïve Bayes.....	20
Accuracy Validation.....	20
KNN.....	21
Accuracy Validation.....	21
Evaluasi model.....	22
Baca Data Uji.....	22
Model Usage.....	23
Menyimpan output ke file.....	25
Lampiran.....	26

Deskripsi Tugas

Diberikan sebuah file Excel bernama "traintest.xlsx" yang terdiri dari dua lembar, yaitu "train" dan "test". Kedua lembar ini mengandung dataset untuk masalah klasifikasi biner, di mana setiap baris data umumnya terdiri dari nomor baris, fitur input (dari kolom A hingga D), dan kelas output (kolom E). Nilai fitur input adalah bilangan bulat dalam rentang tertentu, sementara output kelas adalah nilai biner (0 atau 1).

Lembar "train" memiliki 296 baris data lengkap dengan output kelas yang sesuai (0 atau 1). Lembar ini digunakan untuk melatih model sesuai dengan metode yang dipilih. Sementara lembar "test" terdiri dari 10 baris data, namun output kelasnya (0 atau 1) disembunyikan. Lembar ini akan digunakan untuk menguji model yang telah dilatih. Hasil keluaran dari program untuk data uji ini akan dibandingkan dengan kelas sebenarnya untuk mengevaluasi kinerja model.

	A	B	C	D	E
1	id	x1	x2	x3	y
2	1	60	64	0	1
3	2	54	60	11	0
4	3	65	62	22	0
5	4	34	60	0	1
6	5	38	69	21	0
7	6	33	58	10	1
8	7	63	61	0	1
9	8	57	64	0	1
10	9	46	58	3	1

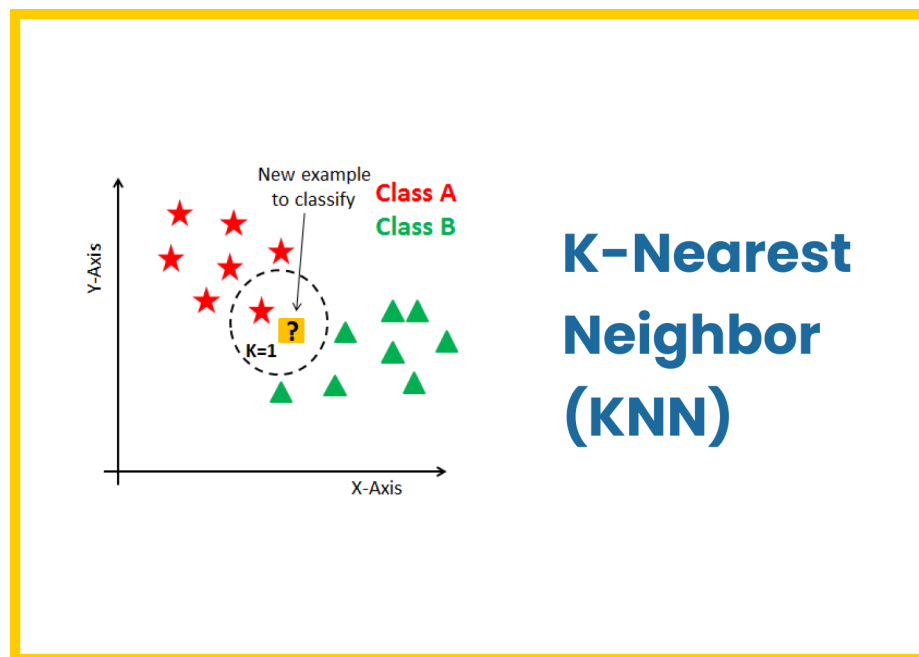
Gambar 1.0 Ilustrasi Data traintest

Link akses dataset : [Link DataSet](#)

Pengenalan Metode

Kami menggunakan 2 metode di dalam tugas ini. 2 metode tersebut adalah K-Nearest Neighbor dan Naïve Bayes. Penggunaan 2 metode ini bermaksud untuk melihat dan menganalisis metode mana yang lebih akurat dalam kasus pengklasifikasian biner ini. Berikut akan dijelaskan pengertian dari kedua metode tersebut.

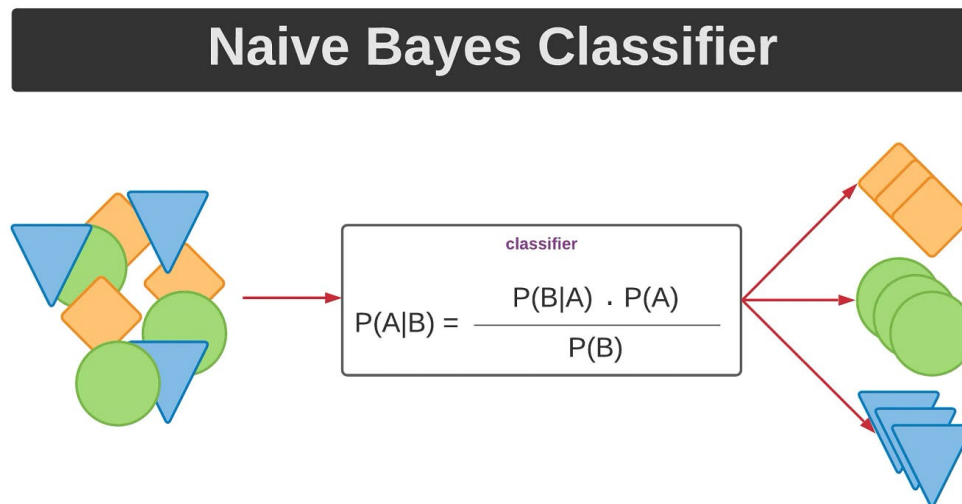
KNN



Gambar 1.1 K-nearest Neighbor

K-Nearest Neighbor (KNN) adalah metode klasifikasi yang memprediksi kelas dari suatu objek berdasarkan kelas dari objek-objek terdekat. Metode ini menghitung jarak antara objek yang akan diprediksi dengan objek-objek lain pada dataset. Kemudian, metode ini memilih kelas yang paling sering muncul pada k objek terdekat sebagai kelas dari objek yang akan diprediksi.

Naïve Bayes



Gambar 1.2 Naïve Bayes Classifier

Naïve Bayes adalah metode klasifikasi yang menggunakan teorema Bayes untuk memprediksi kelas dari suatu objek. Metode ini mengasumsikan bahwa setiap fitur pada dataset saling independen satu sama lain. Metode ini menghitung probabilitas setiap kelas pada dataset dan probabilitas setiap fitur pada setiap kelas. Kemudian, metode ini mengalikan probabilitas setiap fitur pada setiap kelas dan memilih kelas dengan probabilitas tertinggi sebagai kelas dari objek yang akan diprediksi.

Proses yang diimplementasikan ke dalam program

Dikarenakan penggunaan metode yang lebih dari 1, akan dijelaskan bagaimana implementasi yang digunakan di kedua metode tersebut untuk melihat perbedaan dari segi alur dan cara kerja implementasi metode yang dipilih

Membaca data latih/uji

Naïve Bayes

A screenshot of a Jupyter Notebook cell with a dark background. The code is written in Python and uses the pandas, google.colab, and io libraries. It shows the process of uploading a file from Google Drive and reading it as a CSV file. The code is as follows:

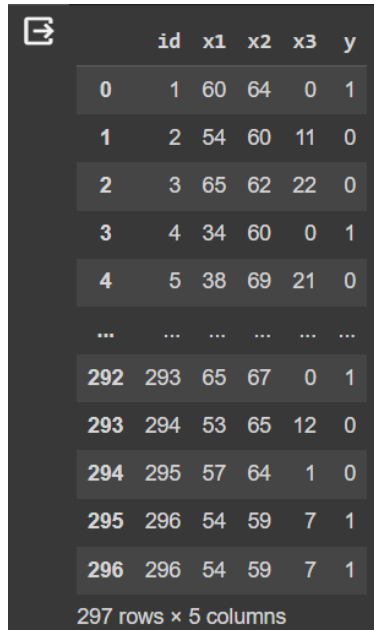
```
import pandas as pd
from google.colab import files
import io

uploaded = files.upload()

data =
io.BytesIO(uploaded["traindata.csv"])
df = pd.read_csv(data, sep = None, engine = 'python')
df
```

Gambar 1.3 Membaca Data Uji

Untuk membaca data latih dan uji, digunakan library `google.colab` dan `io`. Library `google.colab` digunakan untuk mengambil method `file` sebagai pembaca file yang akan digunakan sebagai data latih dan uji. Kemudian kami menggunakan library `pandas` untuk memvisualisasikan dataset agar lebih mudah diolah di kemudian hari.



	id	x1	x2	x3	y
0	1	60	64	0	1
1	2	54	60	11	0
2	3	65	62	22	0
3	4	34	60	0	1
4	5	38	69	21	0
...
292	293	65	67	0	1
293	294	53	65	12	0
294	295	57	64	1	0
295	296	54	59	7	1
296	296	54	59	7	1

297 rows × 5 columns

Gambar 1.4 Output dari membaca Train data

KNN

```

import pandas as pd
from google.colab import files
import io

uploaded = files.upload()

data =
io.BytesIO(uploaded["traindata.csv"])
df = pd.read_csv(data, sep = None, engine = 'python')
df

```

Gambar 1.5 Membaca Data Uji

Untuk membaca data latih dan uji, digunakan library `google.colab` dan `io`. Library `google.colab` digunakan untuk mengambil method file sebagai pembaca file yang akan digunakan sebagai data latih dan uji.

Pelatihan atau training model

Naïve Bayes

Data Preprocessing

Partisi Data

Secara keseluruhan, potongan kode ini menghasilkan dua set data terpisah: X yang berisi fitur atau variabel independen, dan y yang berisi variabel target atau dependen.

```
#partisi data
X = df.drop(['y','id'], axis = 1)
y = df.y
```

Gambar 1.6 Partisi Data

Normalisasi

$$z = \frac{x - \min(x)}{[\max(x) - \min(x)]}$$

Gambar 1.7 Rumus Normalisasi

```
#normalisasi
df = (X - X.min()) / (X.max() -
X.min())
```

Gambar 1.8 Normalisasi

Normalisasi adalah proses mengubah nilai-nilai dalam suatu dataset ke dalam skala yang seragam atau standar. Tujuan normalisasi adalah untuk memastikan bahwa setiap fitur (variabel) dalam dataset memiliki dampak yang setara terhadap analisis atau pemodelan yang dilakukan.

Standarisasi

$$z = \frac{x_i - \mu}{\sigma}$$

Gambar 1.9 Rumus Standardization



```
#standarisasi
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])
```

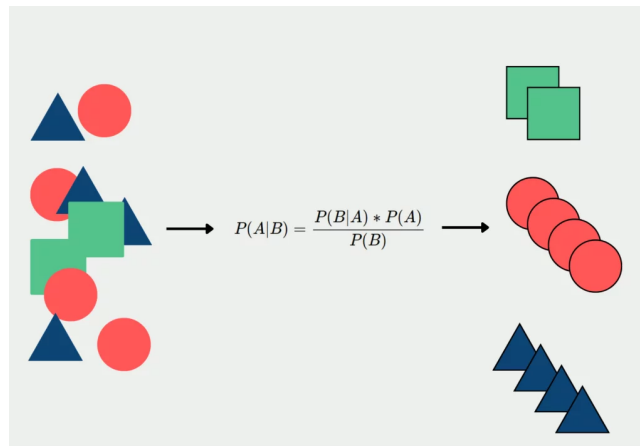
Gambar 2.0 Standardisasi

Standarisasi adalah proses mengubah distribusi nilai suatu variabel ke dalam distribusi normal standar dengan rata-rata sama dengan 0 dan deviasi standar sama dengan 1. Ini dilakukan dengan mengurangi rata-rata dari setiap nilai dalam variabel dan kemudian membaginya dengan deviasi standar. Hasil standarisasi adalah bahwa setiap kolom numerik dalam dataframe X sekarang memiliki rata-rata yang sama dengan 0 dan deviasi standar yang sama dengan 1. Standarisasi berguna terutama ketika kita ingin menghilangkan efek dari perbedaan unit atau satuan pengukuran antar variabel, atau ketika algoritma yang kita gunakan mengasumsikan bahwa fitur-fitur memiliki distribusi normal dan memiliki skala yang seragam.

Model Generation

Class Separation

Fungsi `separate_by_class` di dalam konteks Naïve Bayes memiliki tujuan untuk memisahkan data ke dalam kelas-kelas yang berbeda berdasarkan nilai target y . Dalam klasifikasi Naïve Bayes, perbedaan kelas-kelas ini diperlukan untuk menghitung statistik yang diperlukan untuk membuat prediksi. Fungsi ini memisahkan data menjadi subset-subset yang sesuai dengan setiap nilai unik dalam kolom target y . Sederhananya adalah membagi kelas y menjadi 2 sub-himpunan yang adalah himpunan y bernilai 1 dan y bernilai 0.



Gambar 2.1 Ilustrasi Class Separation

```
def separate_by_class(df):  
    splitted_df = {}  
    for kelas in df.y.unique():  
        splitted_df[kelas] = df[df.y == kelas].reset_index(drop=True)  
    return splitted_df  
  
separate_by_class(df)[0].head()
```

Gambar 2.2 Class Separation

Data Summarization

Karena Naïve Bayes menggunakan sentralitas dan dispersi data, maka digunakanlah *Mean*, *Standard Deviation*, dan *Count*. Dalam Naïve Bayes, menggunakan mean (rata-rata), deviasi standar, dan jumlah data adalah kunci untuk memahami dan memodelkan distribusi probabilitas fitur-fitur dalam setiap kelas. Rata-rata digunakan untuk memberikan perkiraan nilai tengah dari data dalam suatu kelas, sedangkan deviasi standar memberikan informasi tentang sebaran data di sekitar rata-rata, terutama dalam konteks distribusi probabilitas Gaussian. Asumsi distribusi normal ini memungkinkan Naïve Bayes untuk menggunakan rumus Gaussian untuk menghitung probabilitas kondisional. Sementara itu, jumlah data digunakan untuk menghitung probabilitas prior dari setiap kelas, yang merupakan langkah awal dalam menghitung probabilitas kondisional. Dengan informasi ini, Naïve Bayes dapat membuat prediksi dengan memanfaatkan asumsi independensi fitur, menghasilkan model yang sederhana dan efisien untuk klasifikasi.

$$\begin{aligned}\mu &= \frac{1}{n} \sum_{i=1}^n x_i && \text{Mean} \\ \sigma &= \left[\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2 \right]^{0.5} && \text{Standard deviation} \\ f(x) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} && \text{Normal distribution}\end{aligned}$$

Gambar 2.3 Rumus Statistika Data

Kemudian setelah memisahkan class y yang bernilai 1 dan 0 serta menghitung rumus Mean, Standard Deviation, dan Count, maka dikelompokkan lah data tersebut di setiap class yang sudah dipisahkan.

{1:	x1	x2	x3	y
mean	-0.048141	0.012378	-0.173716	1.0
std	1.020798	0.998426	0.818043	0.0
count	219.000000	219.000000	219.000000	219.0,

0:	x1	x2	x3	y
mean	0.135164	-0.034753	0.487740	0.0
std	0.939087	1.016475	1.278957	0.0
count	78.000000	78.000000	78.000000	78.0}

Gambar 2.4 Informasi Tiap Class

Gaussian Equation

```
import math

def gaussian_equation(x,mean,std):
    exponen = math.exp(-((x-mean)**2/(2*std**2)))
    return ( 1 / (math.sqrt(2*math.pi) * std)) * exponen

gaussian_equation(1,1,1)
```

Gambar 2.4 Implementasi Gaussian Equation

Gaussian equation mengacu pada rumus distribusi normal atau kurva Gaussian, yang juga dikenal sebagai distribusi Gauss atau distribusi bell-shaped. Rumus ini digunakan untuk menggambarkan distribusi probabilitas dari suatu variabel acak kontinu. Dalam konteks Naïve Bayes, distribusi normal sering digunakan untuk mengasumsikan bahwa fitur-fitur dalam setiap kelas mengikuti distribusi Gaussian. Ini adalah salah satu asumsi yang dibuat oleh Naïve Bayes untuk menyederhanakan perhitungan probabilitas kondisional. Oleh karena itu, ketika kita melihat istilah "Gaussian equation" dalam konteks Naïve Bayes, itu merujuk pada penggunaan rumus distribusi normal untuk menghitung probabilitas kondisional dari fitur-fitur dalam setiap kelas.

Probabilities Calculation

```
def calculation_probabilites_class(x, summaries):
    n = sum([summaries[kelas].loc['count'].y for kelas in summaries])
    probs = {}
    for kelas in summaries:
        probs[kelas] = summaries[kelas].loc['count'].y / n
        for col in summaries[kelas].columns[:-1]:
            mean, std, _ = summaries[kelas].loc[:, col]
            probs[kelas] *= gaussian_equation(x[col], mean, std) if std != 0 else 0
    # Normalisasi probabilitas
    total_prob = sum(probs.values())
    for kelas in probs:
        if total_prob != 0:
            probs[kelas] /= total_prob
        else:
            probs[kelas] = 0
    return probs

summaries = summarize_by_class(df)
print("Probabilitas kelas baris pertama: ")
calculation_probabilites_class(df.iloc[0], summaries)
```

Gambar 2.4 Probabilities Calculation

Berdasarkan perhitungan Gaussian yang sudah didefinisikan, sekarang saatnya untuk memprediksi peluang sebuah baris termasuk di dalam kelas mana, entah itu 1 atau 0. Prediksi ini dapat digunakan menggunakan perhitungan peluang dari Perhitungan Gaussian. Dalam konteks source code yang dibuat, perhitungan peluang ini ada di dalam Fungsi `calculation_probabilites_class` untuk menghitung probabilitas kelas dari suatu instance data baru (x). Fungsi ini memanfaatkan statistik ringkasan yang telah dihitung sebelumnya untuk setiap kelas, termasuk probabilitas prior dan distribusi probabilitas Gaussian untuk setiap fitur dalam kelas tersebut. Dalam perhitungannya, probabilitas kelas dihitung sebagai produk dari probabilitas prior dan probabilitas kondisional dari setiap fitur dalam kelas, diukur oleh rumus distribusi normal atau Gaussian equation. Selanjutnya, probabilitas tersebut dinormalisasi untuk memastikan bahwa jumlah total probabilitas kelas adalah 1. Hasilnya adalah dictionary probabilitas kelas, di mana setiap kunci adalah kelas yang mungkin dan nilainya adalah probabilitas kelas yang diperkirakan untuk instance data baru.

```
Probabilitas kelas baris pertama:
{1: 0.8456642281280323, 0: 0.15433577187196768}
```

Gambar 2.5 Probabilitas Kelas baris pertama df

Prediction

```
def predict(train, test):  
    preds = []  
    summaries = summarize_by_class(train)  
    for _, row in test.iterrows():  
        probs = calculation_probabilites_class(row,  
        summaries)  
        preds.append(max(probs, key=probs.get))  
    return preds
```

Gambar 2.5 Predict

Kemudian dibuatlah fungsi prediksi untuk memprediksi

KNN

Data Preprocessing

Partisi Data

```
features = df[['x1', 'x2', 'x3']]  
target = df["y"]
```

Gambar 2.6 Partisi data

Kami memisahkan kolom target dan fitur. Kolom target adalah hasil klasifikasi biner pada dataset. Sedangkan kolom fitur adalah atribut yang mempengaruhi hasil klasifikasi biner yang ada pada dataset. Pemisahan ini dilakukan untuk mempermudah proses pelatihan data di kemudian hari.

Normalisasi

Lalu kami menggunakan normalisasi untuk menghindari bias yang mungkin muncul akibat perbedaan skala atau rentang nilai antar fitur dalam dataset. Karena latar belakang metode KNN yang mengukur jarak antar poin, jika ada fitur yang memiliki skala sangat berbeda, maka fitur dengan rentang nilai yang lebih besar akan mendominasi perhitungan jarak.

```
features = (features - features.min()) /  
(features.max()-features.min())  
  
features.describe()  
  
features.iloc[0] - features.iloc[1]
```

Gambar 2.7 Normalisasi dengan Metode KNN

Model Generation

Terdapat 2 metode untuk menghitung jarak antar poin fitur. Metode tersebut adalah Euclidean distance dan Manhattan distance

Euclidean Distance

$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

gambar 2.8 Rumus Euclidean

Rumus Euclidean Distance digunakan untuk mengukur jarak antara dua titik dalam ruang Euclidean, dan berasal dari konsep geometri yang dikenal sebagai Teorema Pythagoras. Dalam konteks metode k-Nearest Neighbors (KNN), rumus ini digunakan untuk menghitung

jarak antara titik data uji dan titik data dalam dataset pelatihan. Rumus ini mencerminkan panjang vektor yang menghubungkan dua titik dalam ruang fitur, di mana setiap dimensi atau fitur diwakili oleh koordinat dalam vektor. Penggunaan akar kuadrat dan jumlah kuadrat perbedaan antar koordinat memastikan bahwa jaraknya selalu positif dan sesuai dengan struktur geometris ruang Euclidean. Dengan memahami geometri Euclidean, rumus ini memberikan cara yang konsisten dan intuitif untuk mengukur kedekatan atau kesamaan antara titik-titik data, yang sangat bermanfaat dalam konteks klasifikasi atau regresi menggunakan metode k-NN.

Manhattan Distance

$$distance = \sum_{1}^n |p_i - q_i|$$

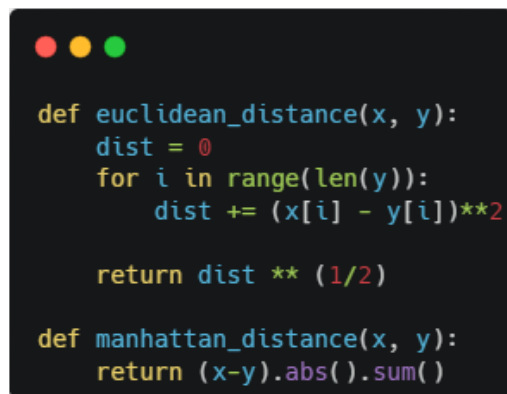
gambar 2.9 Rumus Manhattan

Rumus Manhattan Distance, juga dikenal sebagai jarak kota atau jarak taksi, berasal dari konsep perpindahan dalam kota atau grid, diberi nama "Manhattan" karena jalur grid yang teratur seperti jalan-jalan di Kota New York. Rumus ini digunakan untuk mengukur jarak antara dua titik dengan menyusuri jalur grid, di mana perpindahan hanya dapat terjadi sepanjang sumbu horizontal dan vertikal. Dalam konteks dua dimensi, rumus ini mencerminkan total perpindahan horizontal dan vertikal yang diperlukan untuk mencapai satu titik dari titik lainnya. Konsep ini terkait erat dengan ide perpindahan dalam sistem grid yang membentuk sudut kanan, dan rumus Manhattan Distance sering digunakan dalam berbagai aplikasi seperti algoritma k-Nearest Neighbors (k-NN) dan masalah optimasi rute.

Perbedaan 2 metode tersebut terletak di rumus. Dalam Euclidean distance, perpindahan semua arah diukur dan diperlukan sebagai

panjang vektor, sedangkan dalam Manhattan distance, perpindahan hanya dapat terjadi sepanjang sumbu horizontal atau vertikal. Euclidean distance lebih cenderung memberikan bobot yang lebih besar pada perbedaan dalam satu dimensi, sementara Manhattan distance lebih memperhitungkan perbedaan dalam semua dimensi dengan proporsi yang sama.

Code Translation



```
def euclidean_distance(x, y):  
    dist = 0  
    for i in range(len(y)):  
        dist += (x[i] - y[i])**2  
  
    return dist ** (1/2)  
  
def manhattan_distance(x, y):  
    return (x-y).abs().sum()
```

gambar 3.0 Implementasi Metode Euclidean dan Manhattan pada Metode KNN

Source code di atas adalah hasil terjemahan rumus untuk menghitung Euclidean Distance dan Manhattan Distance. Euclidean Distance dihitung di dalam perulangan sepanjang baris yang dihitung kemudian dikalikan bilangan $\frac{1}{2}$. Sedangkan Manhattan Distance menghitung summation atau jumlah dari keseluruhan jarak yang dihitung.

Prediction

```
def predict_knn(x,k,X,y):
    distance = []
    for _, row in X.iterrows():
        distance.append(manhattan_distance(x,row))
        #distance.append(euclidean_distance(x,row))

    data = X.copy()
    data['distance'] = distance
    data['clas'] = y
    data.sort_values(by="distance").reset_index(drop=True)

    y_pred = data.iloc[:k].clas.mode()

    return y_pred[0]
```

gambar 3.1 Implementasi Metode Euclidean dan Manhattan pada Metode KNN

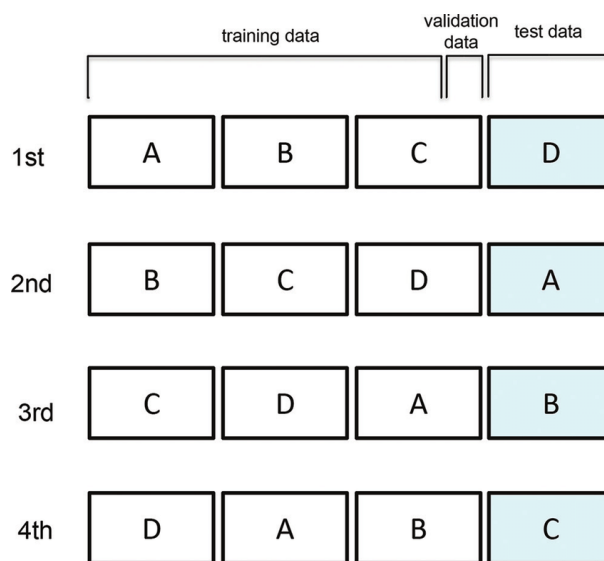
Bagian ini mendefinisikan fungsi-fungsi yang diperlukan untuk pengujian model, termasuk fungsi jarak (Euclidean dan Manhattan) dan fungsi prediksi kelas. Fungsi-fungsi ini nantinya digunakan dalam proses evaluasi model.

Fungsi `predict_knn` merupakan implementasi algoritma k-Nearest Neighbors KNN dengan menggunakan jarak Manhattan untuk melakukan prediksi kelas suatu titik data baru (x). Dalam fungsi ini, setiap titik data dalam data latih (X) diukur jaraknya terhadap titik data uji menggunakan metode jarak Manhattan. Hasil jarak tersebut disusun bersama dengan kelas masing-masing titik data dalam *dataframe* baru yang disebut `data`. Data tersebut kemudian diurutkan berdasarkan jarak, dengan titik data terdekat mendapatkan posisi teratas. Selanjutnya, fungsi ini melakukan prediksi kelas dengan memilih kelas yang paling sering muncul di antara k tetangga terdekat. Output dari fungsi ini adalah prediksi kelas untuk titik data uji. Fungsi ini memberikan fleksibilitas untuk menggunakan matriks jarak yang berbeda dengan mengganti pemanggilan fungsi `manhattan_distance` dengan `euclidean_distance`, yang

dapat disesuaikan sesuai kebutuhan. Dengan demikian, fungsi `predict_knn` memberikan kemampuan untuk melakukan klasifikasi berbasis KNN dengan jarak Manhattan pada data yang diberikan.

Pengujian atau testing model

Data Folding



gambar 3.2 Ilustrasi Data Folding

Data Folding memastikan bahwa model dinilai pada beragam subset data, mengurangi kemungkinan overfitting atau evaluasi yang terlalu berfokus pada karakteristik tertentu dari dataset. Dengan cara ini, data folding memberikan estimasi kinerja model yang lebih reliabel dan meminimalkan risiko over-optimistik pada hasil evaluasi. Pada konteks kasus kali ini, dilakukan pembagian dataset menjadi tiga bagian untuk proses pengujian dan pelatihan pada metode pengujian silang (cross-validation). Ukuran setiap bagian dihitung berdasarkan panjang total dataset, dan kemudian dilakukan alokasi data sesuai dengan aturan pengujian silang yang telah ditetapkan.

```

# Menentukan ukuran setiap bagian
size = len(df)
size_per_fold = size // 3

# Mengambil bagian pertama untuk testing dan dua bagian sisanya untuk training
fold_first = (df.iloc[:size_per_fold].reset_index(drop=True),
df.iloc[size_per_fold:].reset_index(drop=True))

# Mengambil 50 persen data dengan menyilang masing-masing untuk training dan testing
fold_second = (pd.concat([df.iloc[:size_per_fold], df.iloc[2*size_per_fold:]]).reset_index(drop=True),
pd.concat([df.iloc[size_per_fold:2*size_per_fold]]).reset_index(drop=True))

# Mengambil quarter akhir untuk testing dan dua quarter awal untuk training
fold_third = (df.iloc[size_per_fold:].reset_index(drop=True),
df.iloc[:2*size_per_fold].reset_index(drop=True))

```

gambar 3.3 Ilustrasi Data Folding

Bagian pertama digunakan sebagai set pengujian, sedangkan dua bagian sisanya digabungkan untuk membentuk set pelatihan pada fold pertama. Pada fold kedua, dilakukan penggabungan data antara bagian pertama dan ketiga untuk set pelatihan, sementara bagian kedua digunakan sebagai set pengujian. Pada fold ketiga, bagian kedua menjadi set pengujian, sementara dua bagian sisanya digunakan sebagai set pelatihan. Proses ini dilakukan dengan tujuan untuk melakukan evaluasi performa model dengan memastikan bahwa setiap bagian dataset digunakan sebagai set pengujian sekali dan sebagai set pelatihan dua kali, mencakup seluruh dataset secara keseluruhan.

Naïve Bayes

Accuracy Validation

Berdasarkan hasil pengukuran akurasi yang sudah dilakukan, didapati bahwa data yang memiliki akurasi tertinggi adalah fold ketiga. Fold ketiga memiliki akurasi 79%. Untuk itu Fold ketiga akan digunakan untuk generasi data di data uji yang akan dilakukan di step selanjutnya.

```

Fold 1: Akurasi = 72.73%
Fold 2: Akurasi = 77.78%
Fold 3: Akurasi = 79.29%
Akurasi tertinggi diperoleh pada Fold ke-3 dengan akurasi 79.29%

```

gambar 3.4 Hasil Pengukuran Akurasi

KNN

Accuracy Validation

Berdasarkan hasil pengukuran akurasi yang sudah dilakukan, didapati bahwa terdapat stagnasi akurasi di setiap iterasi yang dijalankan. Hal ini berarti bahwa data ini kurang cocok untuk model KNN dan karena dilakukan latihan data terhadap 2 model, masih terdapat alternatif model lain yang dapat digunakan.

```
Accuracy for k=1: 78.79%
Accuracy for k=3: 78.79%
Accuracy for k=5: 78.79%
Accuracy for k=7: 78.79%
Accuracy for k=9: 78.79%
Accuracy for k=11: 78.79%
Accuracy for k=13: 78.79%
Accuracy for k=15: 78.79%
Accuracy for k=17: 78.79%
Accuracy for k=19: 78.79%
Accuracy for k=21: 78.79%
```

gambar 3.5 Hasil Pengukuran Akurasi

Evaluasi model

Setelah melihat bahwa tingkat akurasi dari model Naive Bayes lebih tinggi daripada KNN, diambil Naive Bayes sebagai model yang akan digunakan untuk generasi klasifikasi biner terhadap data uji.

Baca Data Uji

	id	x1	x2	x3	y
0	297	43	59	2	?
1	298	67	66	0	?
2	299	58	60	3	?
3	300	49	63	3	?
4	301	45	60	0	?
5	302	54	58	1	?
6	303	56	66	3	?
7	304	42	69	1	?
8	305	50	59	2	?
9	306	59	60	0	?

gambar 3.6 Baca Data Uji

Terdapat 10 baris data uji yang masih belum diklasifikasikan secara biner. Tugas dari model Naive Bayes adalah untuk mengklasifikasikan kesepuluh data tersebut secara biner dan mengelompokkan mana data yang memiliki kelas nilai 1 atau 0.

Model Usage

```
def generate_data_from_model():
    preds = []
    train = fold_third[0]
    test = df
    summaries = summarize_by_class(train)
    for _, row in test.iterrows():
        # Cek apakah nilai y pada baris saat ini adalah '?'
        if row['y'] == '?':
            # Hitung probabilitas menggunakan fungsi calculation_probabilites_class
            probs = calculation_probabilites_class(row.drop('y'), summaries)
            # Ambil kelas dengan probabilitas tertinggi sebagai hasil prediksi
            predicted_class = max(probs, key=probs.get)
            # Ganti nilai '?' dengan hasil prediksi
            row['y'] = predicted_class
        preds.append(row['y'])
    return preds

generate_data_from_model()
```

gambar 3.7 Generate model

Kemudian dibuat fungsi untuk generasi data hasil dari pelatihan model Naive Bayes. Fungsi `generate_data_from_model()` ini digunakan untuk menghasilkan prediksi kelas baru pada dataset yang mengandung nilai target ('y') yang awalnya tidak diketahui atau ditandai dengan tanda tanya ('?'). Fungsi ini menggunakan model Naive Bayes yang telah dilatih pada bagian pelatihan dari dataset (fold ketiga) untuk menghitung probabilitas kelas untuk setiap baris pada dataset yang membutuhkan prediksi. Jika nilai target pada suatu baris adalah '?', fungsi akan menggantinya dengan kelas yang memiliki probabilitas tertinggi berdasarkan model Naive Bayes. Hasilnya adalah prediksi kelas baru untuk seluruh dataset, dengan nilai-nilai target yang semula tidak diketahui telah digantikan dengan prediksi yang dihasilkan oleh model.

Hasil dari fungsi tersebut adalah prediksi kelas dari setiap baris data uji.

	id	x1	x2	x3	y
0	297	43	59	2	1
1	298	67	66	0	1
2	299	58	60	3	1
3	300	49	63	3	1
4	301	45	60	0	1
5	302	54	58	1	1
6	303	56	66	3	1
7	304	42	69	1	1
8	305	50	59	2	1
9	306	59	60	0	1

gambar 3.8 Data ter-generasi

Terlihat bahwa setiap baris data uji yang telah terklasifikasikan mengarah ke nilai 1 dalam biner. Hal ini adalah hasil dari prediksi Naive Bayes yang sudah didefinisikan sebelumnya.

Menyimpan output ke file

Menyimpan output ke file .csv bisa dilakukan dengan method yang ada di library Pandas, yaitu `df.to_csv`. Method ini merubah *DataFrame* menjadi bentuk .csv yang bisa diakses oleh Microsoft

Pandas To CSV

`df.to_csv(index=False)`

Index	Name	num_customers	Bill
0	Foreign Cinema	50	289.0
1	Liho Liho	45	224.0
2	500 Club	102	80.5
3	The Square	65	25.3

name,num_customers,AvgBill
Foreign Cinema,50,289.0
Liho Liho,45,224.0
500 Club,102,80.5
The Square,65,25.3

The text above may look gross, but
this is how 99% of small data
tables are saved

gambar X.0 ilustrasi method .to_csv

predicted_data.csv ×			
1 to 10 of 10 entries			
Filter			
x1	x2	x3	y
43	59	2	1
67	66	0	1
58	60	3	1
49	63	3	1
45	60	0	1
54	58	1	1
56	66	3	1
42	69	1	1
50	59	2	1
59	60	0	1
Show 10 per page			

gambar 3.9 file .csv yang tergenerasi

Lampiran

Source Code :

[Naïve Bayes](#)

[KNN](#)