TUGAS PEMROGRAMAN 03 - LEARNING

Mata Kuliah Pengantar Kecerdasan Buatan

Disusun oleh:

Berlian Muhammad Galin Al Awienoor	(1301204378)
Kiki Dwi Prasetyo	(1301204027)

KELOMPOK 7

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
UNIVERSITAS TELKOM
2021/2022

DAFTAR ISI

PENDAHULUAN

A. Latar Belakang

k-Nearest Neighbors (kNN) adalah adalah suatu metode yang menggunakan algoritma *supervised learning,* dimana hasil dari sampel uji yang baru diklasifikasikan berdasarkan mayoritas dari kategori pada kNN. Mengklasifikasi objek baru berdasarkan atribut dan sampel latih adalah tujuan dari algoritma ini. *k-Nearest Neighbors* adalah salah satu algoritma klasifikasi paling dasar dan penting dalam *Machine Learning.* kNN memiliki domain pembelajaran yang diawasi dan menemukan aplikasi intens dalam pengenalan pola, penambangan data, dan deteksi intrusi. kNN berfungsi untuk melakukan klasifikasi suatu data berdasarkan data pembelajaran (*train data sets*), yang diambil dari k tetangga terdekatnya (*nearest neighbors*). Dengan k merupakan banyaknya tetangga terdekat.

B. Rumusan Masalah

Pada tugas pemrograman kali ini diberikan file traintest.xlsx yang terdiri dari dua *sheet: train* dan *test,* yang berisi dataset untuk problem klasifikasi biner *(binary classification).* Setiap *record* atau baris data dalam dataset tersebut secara umum terdiri dari nomor baris data (*id*), fitur input (*x*1 sampai *x*3), dan output kelas (*y*). Fitur input terdiri dari nilai-nilai *integer* dalam *range* tertentu untuk setiap fitur. Sedangkan output kelas bernilai biner (0 atau 1).

id	x1	x2	x3	y
1	60	64	0	1
2	54	60	11	0
3	65	62	22	0
4	34	60	0	1
5	38	69	21	0

Sheet *train* berisi 296 baris data, lengkap dengan target output kelas (*y*). Gunakan *sheet* ini untuk tahap pemodelan atau pelatihan *(training)* model sesuai metode yang Anda gunakan. Adapun *sheet test* berisi 10 baris data, dengan output kelas (*y*) yang disembunyikan. Gunakan sheet ini untuk tahap pengujian *(testing)* model yang sudah dilatih. Nantinya output program Anda untuk data uji ini akan dicocokkan dengan target atau kelas sesungguhnya.

C. Tujuan

Tujuan dari disusunnya laporan ini adalah sebagai berikut:

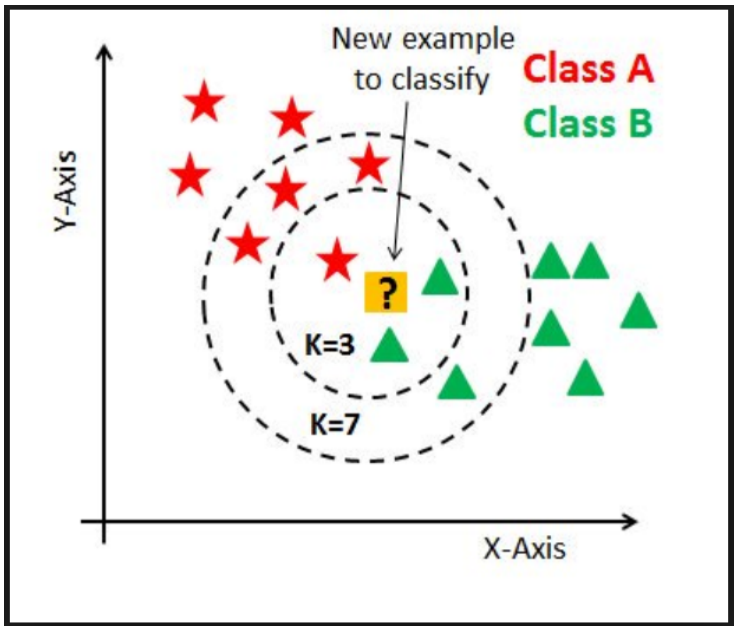
1. Untuk memenuhi tugas mata kuliah Pengantar Kecerdasan Buatan
2. Untuk menunjukkan dan menjelaskan hasil analisis dengan metode kNN
3. Untuk menunjukkan dan menjelaskan hasil akhir yang didapat dari dataset yang diberikan

PEMBAHASAN

A. *k-Nearest Neighbors*

k-Nearest Neighbors (kNN) adalah adalah suatu metode yang menggunakan algoritma *supervised learning,* dimana hasil dari sampel uji yang baru diklasifikasikan berdasarkan mayoritas dari kategori pada kNN. Mengklasifikasi objek baru berdasarkan atribut dan sampel latih adalah tujuan dari algoritma ini. *k-Nearest Neighbors* adalah salah satu algoritma klasifikasi paling dasar dan penting dalam *Machine Learning.* kNN memiliki domain pembelajaran yang diawasi dan menemukan aplikasi intens dalam pengenalan pola, penambangan data, dan deteksi intrusi. kNN berfungsi untuk melakukan klasifikasi suatu data berdasarkan data pembelajaran (*train data sets*), yang diambil dari k tetangga terdekatnya (*nearest neighbors*). Dengan k merupakan banyaknya tetangga terdekat.

kNN memiliki karakteristik seperti mengklasifikasi langsung dari tetangga terdekat, merupakan *Instance Based Learning* (IBL) dan disebut sebagai *Lazy learner,* dimana kNN tidak melakukan proses belajar dari data latih.



1. Algoritma kNN

   Secara sederhana, algoritma kNN bisa dituliskan hanya dengan dua langkah. Yang pertama adalah pelatihan, yaitu menyimpan setiap pola latih. Tidak ada proses pembangunan model klasifikasi seperti pada ID3. Langkah kedua adalah klasifikasi. Setiap kali mengklasifikasikan sebuah pola, kNN harus memeriksa semua pola latih untuk menemukan sejumlah *k* pola terdekat. Proses pelatihan kNN menghasilkan nilai *k* yang yang diharapkan mampu memberikan akurasi tertinggi untuk menggeneralisasi data-data yang akan datang. Proses pelatihan pada dasarnya adalah melakukan observasi terhadap sejumlah *k* sampai dihasilkan *k* yang paling optimum.

   Jadi, algoritma kNN mudah diimplementasikan hanya dengan mengatur satu parameter *k*. Keputusan kelas bisa ditelusuri, sehingga mempermudah kita untuk memperbarui model. Karena bekerja secara lokal, hanya memperhitungkan sejumlah *k* pola atau objek data, kNN secara umum mampu memberikan performansi relatif tinggi untuk himpunan data yang terkelompok secara lokal dan nonlinier.

2. Kelebihan Algoritma kNN

   Terdapat kelebihan dari penggunaan algoritma kNN, berikut adalah kelebihan dari kNN.

   a) Algoritma kNN kuat dalam mentraining data yang noisy
   b) Algoritma kNN sangat efektif jika datanya besar
   c) Mudah diimplementasikan

3. Kekurangan Algoritma kNN

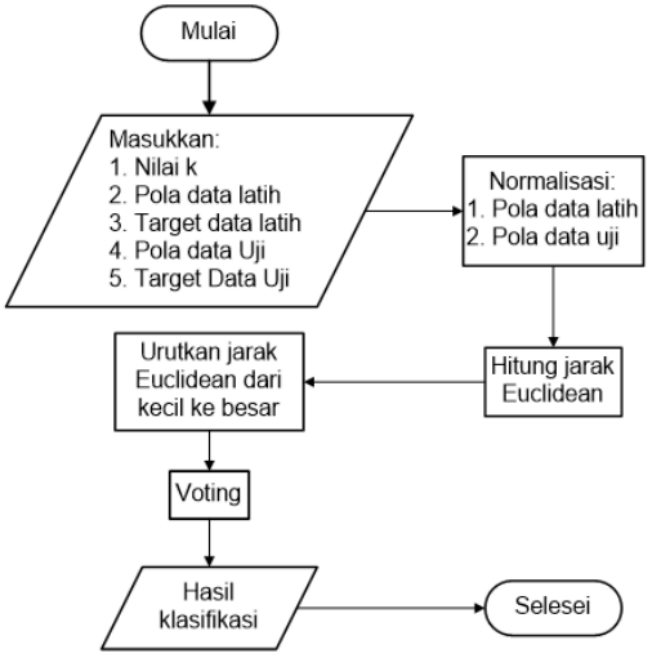
   Sementara itu,terdapat juga kekurangan dari algoritma kNN.

   a) Algoritma kNN perlu menentukan nilai parameter k
   b) Sensitif pada data pencilan
   c) Rentan pada variabel yang non-informatif

4. Cara Kerja Algoritma KNN

   Secara umum, cara kerja algoritma kNN adalah sebagai berikut.

   a) Tentukan jumlah tetangga (k) yang akan digunakan untuk pertimbangan penentuan kelas
   b) Hitung jarak dari data baru ke masing-masing data point di dataset
   c) Ambil sejumlah k data dengan jarak terdekat, kemudian tentukan kelas dari data baru tersebut



5. Normalisasi

Normalisasi data merupakan salah satu teknik yang penting untuk dipahami dalam praproses data. Dalam analisis dan eksplorasi data sering kali ditemukan banyak *features* atau variabel di dalam dataset yang akan dianalisis. Normalisasi data adalah proses membuat beberapa variabel memiliki rentang nilai yang sama, tidak ada yang terlalu besar maupun terlalu kecil sehingga dapat membuat analisis statistik menjadi lebih mudah.

Contoh metode normalisasi data adalah *Min-Max Scaling*. Cara kerjanya setiap nilai pada sebuah fitur dikurangi dengan nilai minimum fitur tersebut, kemudian dibagi dengan rentang nilai atau nilai maksimum dikurangi nilai minimum dari fitur tersebut. Cara ini akan menghasilkan nilai baru hasil normalisasi yang berkisar antara 0 dan 1

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

6. *Euclidean Distance*

Euclidean Distance didefinisikan sebagai jarak antara dua titik. Dengan kata lain, *Euclidean Distance* antara dua titik dalam ruang *Euclidean* didefinisikan sebagai panjang segmen garis antara dua titik. Karena jarak Euclidean dapat ditemukan dengan menggunakan titik koordinat dan teorema Pythagoras, jarak ini kadang-kadang disebut jarak Pythagoras.

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{ip} - x_{jp})^2}$$

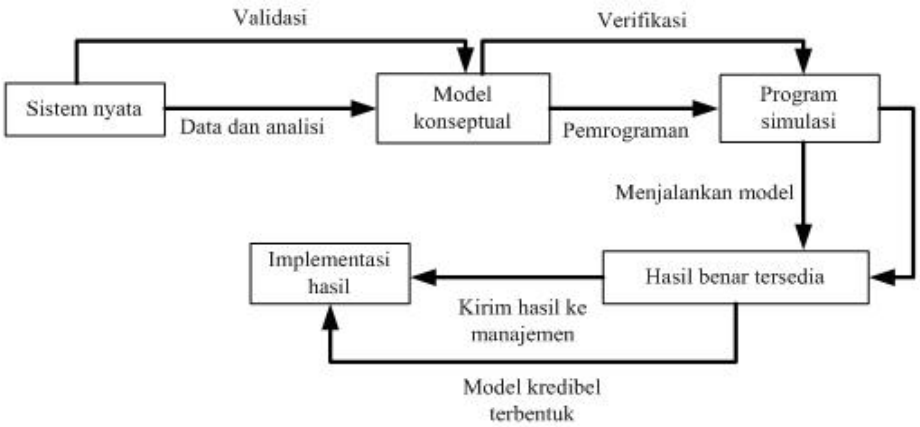
7. *Manhattan Distance*

Manhattan Distance adalah metrik jarak antara dua titik dalam ruang vektor berdimensi N. Ini adalah jumlah panjang proyeksi segmen garis antara titik-titik ke sumbu koordinat. Dalam istilah sederhana, ini adalah jumlah perbedaan mutlak antara ukuran di semua dimensi dari dua titik.

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}|$$

8. Validasi

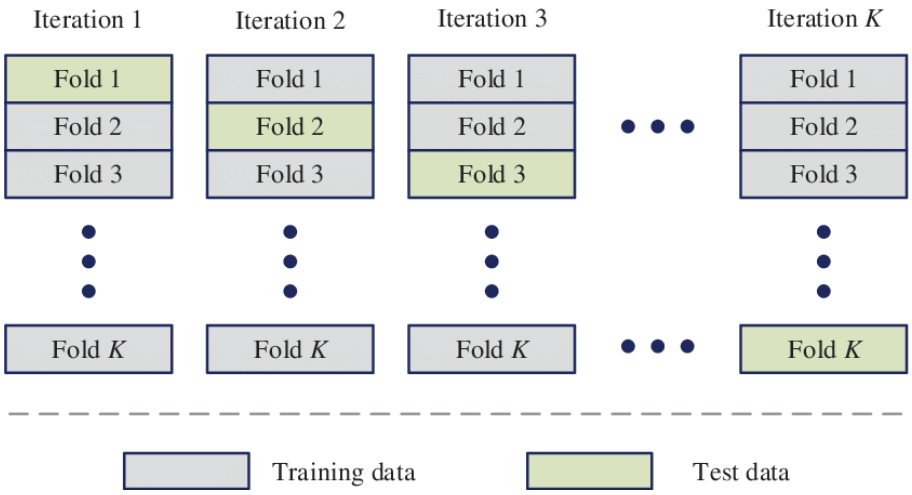
Pada teknik learning, mungkin performansi model yang dibangun sudah sangat tinggi. Namun, ternyata performansinya rendah sekali untuk data-data baru. Bukan modelnya yang harus disalahkan, tetapi perlu cara yang tepat untuk memvalidasi model tersebut. Intinya, tidak akan bisa membangun model yang baik selama tidak bisa melakukan validasi model secara benar. Validasi model sangat penting dilakukan dengan cara yang benar. Model yang salah, tidak valid, bisa mengakibatkan kerugian sangat besar, bahkan fatal.



9. *k-Fold Cross-Validation*

k-Fold Cross-Validation adalah metode statistik yang digunakan untuk memperkirakan keterampilan model *machine learning* atau prosedur pengambilan sampel ulang yang digunakan untuk mengevaluasi model pembelajaran mesin pada sampel data terbatas. Ini biasanya digunakan dalam *machine learning* terapan untuk membandingkan dan memilih model untuk masalah pemodelan prediktif tertentu karena mudah dipahami, mudah diterapkan, dan menghasilkan perkiraan keterampilan yang umumnya memiliki bias lebih rendah daripada metode lain.

Metode ini memiliki parameter tunggal *k* yang mengacu pada jumlah grup yang akan dibagi menjadi sampel data tertentu. Menggunakan sampel terbatas untuk memperkirakan bagaimana model diharapkan tampil secara umum ketika digunakan untuk membuat prediksi pada data yang tidak digunakan selama pelatihan model. Ini adalah metode yang populer karena mudah dipahami dan karena umumnya menghasilkan perkiraan keterampilan model yang kurang bias atau kurang optimis dibandingkan metode lain, seperti pemisahan *train/test* sederhana.



IMPLEMENTASI PENGKODEAN ALGORITMA

Untuk melakukan analisis dan evaluasi algoritma *k-Nearest Neighbors*, berikut adalah tahapan implementasi untuk proses algoritma kNN yang telah kami buat sedemikian rupa dari tahap awal (*pre-processing*) sampai tahap akhir (*meng-outputkan result*) dengan menggunakan bahasa pemrograman Python.

1. Import Dataset

   Dataset yang digunakan dalam tugas pemograman ini terdata 2 *sheet*, yaitu *sheet train* dan *sheet test.*

   Import dataset train

```
#import data train dari traintest.xlsx
dfTrain = pd.read_excel("https://github.com/berlianm/k-Nearest-Neighbors/blob/main/traintest.xlsx?raw=true", sheet_name='train')
dfTrain
```

	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
...	...	...	...	...	...
291	292	59	64	1	1
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 rows × 5 columns

Import dataset test

```
#import data test dari traintest.xlsx
dfTest = pd.read_excel("https://github.com/berlianm/k-Nearest-Neighbors/blob/main/traintest.xlsx?raw=true", sheet_name='test')
dfTest
```

	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	?

2. Membaca data latih/uji

   Membaca data latih/uji menggunakan teknik visual, digunakan untuk menemukan tren, pola, atau untuk memeriksa asumsi dengan bantuan ringkasan statistik dan representasi grafik.

   2.1 Deskripsi Detail Dataset

   Deskripsi detail dari dataset *train* dan dataset *test*

Describe data *train*

```
dfTrain.describe()
```

	id	x1	x2	x3	y
count	296.000000	296.000000	296.000000	296.000000	296.000000
mean	148.500000	52.462838	62.881757	4.111486	0.736486
std	85.592056	10.896367	3.233753	7.291816	0.441285
min	1.000000	30.000000	58.000000	0.000000	0.000000
25%	74.750000	44.000000	60.000000	0.000000	0.000000
50%	148.500000	52.000000	63.000000	1.000000	1.000000
75%	222.250000	61.000000	65.250000	5.000000	1.000000
max	296.000000	83.000000	69.000000	52.000000	1.000000

Describe data *test*

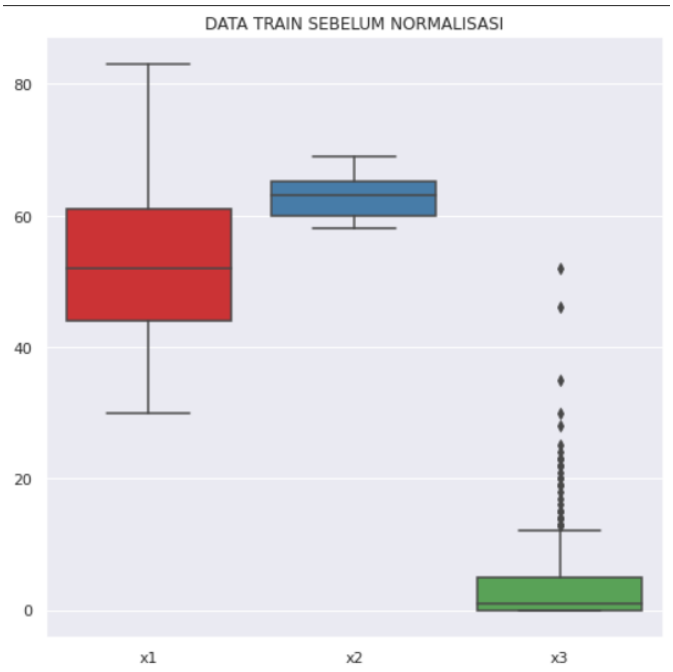
```
dfTest.describe()
```

	id	x1	x2	x3
count	10.00000	10.000000	10.000000	10.000000
mean	301.50000	52.300000	62.000000	1.500000
std	3.02765	7.972871	3.771236	1.269296
min	297.00000	42.000000	58.000000	0.000000
25%	299.25000	46.000000	59.250000	0.250000
50%	301.50000	52.000000	60.000000	1.500000
75%	303.75000	57.500000	65.250000	2.750000
max	306.00000	67.000000	69.000000	3.000000

2.2 Visualisasi Dataset (Sebelum Normalisasi)

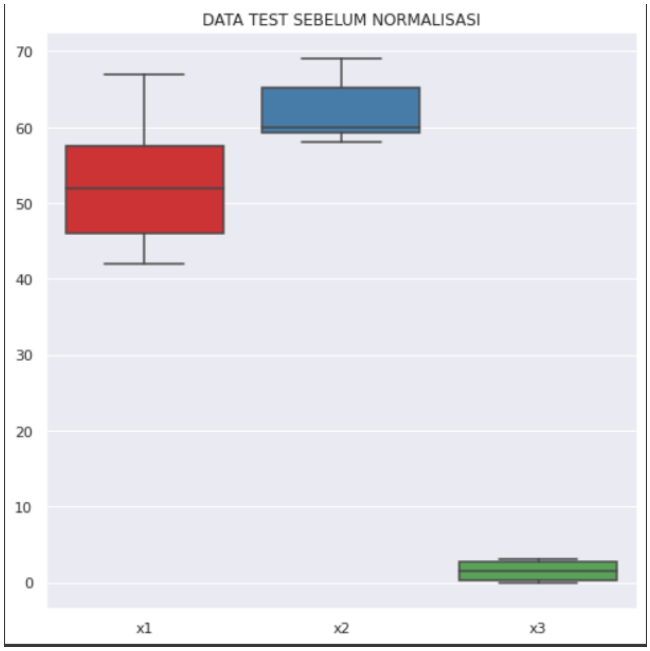
Menampilkan *boxplot* dari data *train* yang belum dinormalisasi

```
#menampilkan data train sebelum normalisasi (boxplot)
sns.set_theme(style="darkgrid")
ax = sns.boxplot(data=dfTrain[['x1', 'x2', 'x3']], orient="v", palette="Set1")
sns.set(rc={'figure.figsize':(8, 8)})
ax.set_xticklabels(['x1', 'x2', 'x3'])
plt.title('DATA TRAIN SEBELUM NORMALISASI')
```



Menampilkan *boxplot* dari data *test* yang belum dinormalisasi

```
#menampilkan data test sebelum normalisasi (boxplot)
sns.set_theme(style="darkgrid")
ax = sns.boxplot(data=dfTest[['x1', 'x2', 'x3']], orient="v", palette="Set1")
sns.set(rc={'figure.figsize':(8, 8)})
ax.set_xticklabels(['x1', 'x2', 'x3'])
plt.title('DATA TEST SEBELUM NORMALISASI')
```



3. Normalisasi (*Min-Max Scaling*)

Seperti yang dijelaskan diatas, normalisasi adalah proses membuat beberapa variabel memiliki rentang nilai yang sama, tidak ada yang terlalu besar maupun terlalu kecil sehingga dapat membuat analisis statistik menjadi lebih mudah. Disini kami menggunakan rumus *min-max scaling.*

Normalisasi data *train*

```
#normalisasi data train (min-maks scaling)
df_train_normalisasi = pd.DataFrame(index=dfTrain.index, columns=dfTrain.columns)

xMaks = dfTrain["x1"].max()
xMin  = dfTrain["x1"].min()

for i in range(len(dfTrain)):
    x = dfTrain["x1"][i]
    xBaru = (x - xMin) / (xMaks - xMin)
    df_train_normalisasi["x1"][i] = xBaru

xMaks = dfTrain["x2"].max()
xMin  = dfTrain["x2"].min()

for i in range(len(dfTrain)):
    x = dfTrain["x2"][i]
    xBaru = (x - xMin) / (xMaks - xMin)
    df_train_normalisasi["x2"][i] = xBaru

xMaks = dfTrain["x3"].max()
xMin  = dfTrain["x3"].min()

for i in range(len(dfTrain)):
    x = dfTrain["x3"][i]
    xBaru = (x - xMin) / (xMaks - xMin)
    df_train_normalisasi["x3"][i] = xBaru

df_train_normalisasi['x1'] = df_train_normalisasi['x1'].astype(float)
df_train_normalisasi['x2'] = df_train_normalisasi['x2'].astype(float)
df_train_normalisasi['x3'] = df_train_normalisasi['x3'].astype(float)
df_train_normalisasi['y']  = dfTrain['y']
df_train_normalisasi['id'] = dfTrain['id']
df_train_normalisasi
```

	id	x1	x2	x3	y
0	1	0.566038	0.545455	0.000000	1
1	2	0.452830	0.181818	0.211538	0
2	3	0.660377	0.363636	0.423077	0
3	4	0.075472	0.181818	0.000000	1
4	5	0.150943	1.000000	0.403846	0
...	...	...	...	...	...
291	292	0.547170	0.545455	0.019231	1
292	293	0.660377	0.818182	0.000000	1
293	294	0.433962	0.636364	0.230769	0
294	295	0.509434	0.545455	0.019231	0
295	296	0.452830	0.090909	0.134615	1

296 rows × 5 columns

Normalisasi data *test*

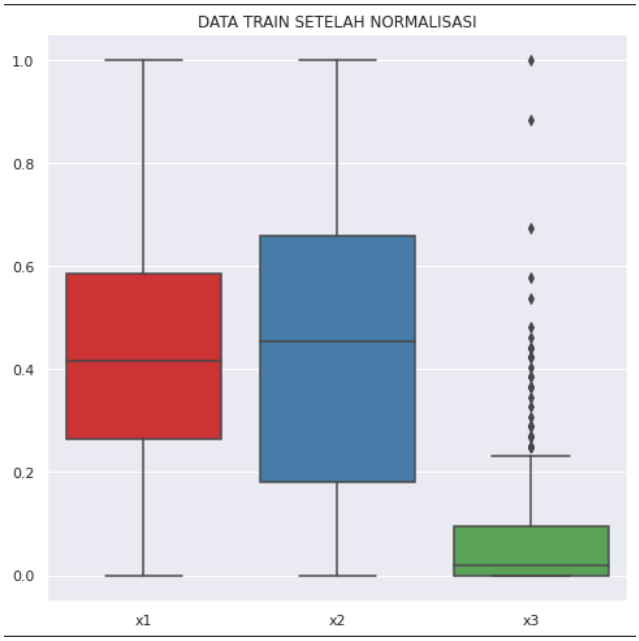
```
#normalisasi data test (min-maks scaling)
df_test_normalisasi = pd.DataFrame(index=dfTest.index, columns=dfTest.columns)

xMaks = dfTest["x1"].max()
xMin  = dfTest["x1"].min()

for i in range(len(dfTest)):
    x = dfTest["x1"][i]
    xBaru = (x - xMin) / (xMaks - xMin)
    df_test_normalisasi["x1"][i] = float(xBaru)

xMaks = dfTest["x2"].max()
xMin  = dfTest["x2"].min()

for i in range(len(dfTest)):
    x = dfTest["x2"][i]
    xBaru = (x - xMin) / (xMaks - xMin)
    df_test_normalisasi["x2"][i] = float(xBaru)

xMaks = dfTest["x3"].max()
xMin  = dfTest["x3"].min()

for i in range(len(dfTest)):
    x = dfTest["x3"][i]
    xBaru = (x - xMin) / (xMaks - xMin)
    df_test_normalisasi["x3"][i] = float(xBaru)

df_test_normalisasi['x1'] = df_test_normalisasi['x1'].astype(float)
df_test_normalisasi['x2'] = df_test_normalisasi['x2'].astype(float)
df_test_normalisasi['x3'] = df_test_normalisasi['x3'].astype(float)
df_test_normalisasi['id'] = dfTest['id'].astype(str)
df_test_normalisasi['y'] = dfTest['y']
df_test_normalisasi
```

	id	x1	x2	x3	y
0	297	0.04	0.090909	0.666667	?
1	298	1.00	0.727273	0.000000	?
2	299	0.64	0.181818	1.000000	?
3	300	0.28	0.454545	1.000000	?
4	301	0.12	0.181818	0.000000	?
5	302	0.48	0.000000	0.333333	?
6	303	0.56	0.727273	1.000000	?
7	304	0.00	1.000000	0.333333	?
8	305	0.32	0.090909	0.666667	?
9	306	0.68	0.181818	0.000000	?

3.1 Visualisasi Dataset (Sesudah Normalisasi)

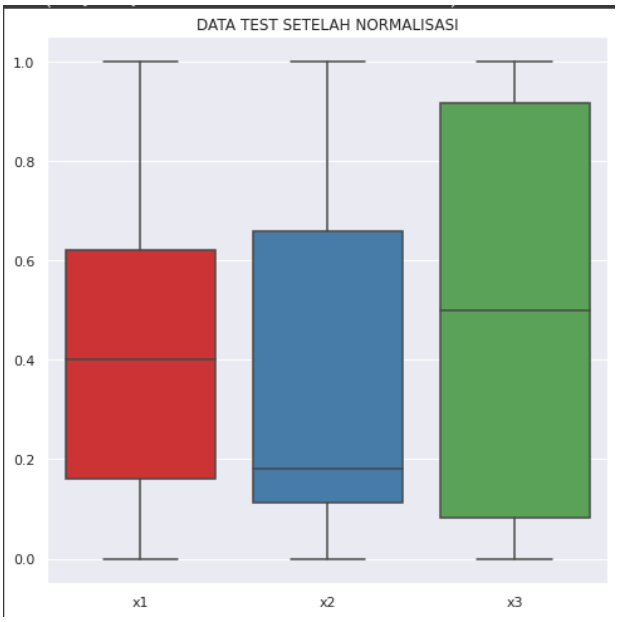
Menampilkan *boxplot* data *train* setelah dinormalisasi

```
#menampilkan data train sesudah normalisasi (boxplot)
sns.set_theme(style="darkgrid")
ax = sns.boxplot(data=df_train_normalisasi[['x1', 'x2', 'x3']], orient="v", palette="Set1")
sns.set(rc={'figure.figsize':(8, 8)})
ax.set_xticklabels(['x1', 'x2', 'x3'])
plt.title('DATA TRAIN SETELAH NORMALISASI')
```



Menampilkan *boxplot* data *test* setelah dinormalisasi

```
#menampilkan data test sesudah normalisasi (boxplot)
sns.set_theme(style="darkgrid")
ax = sns.boxplot(data=df_test_normalisasi[['x1', 'x2', 'x3']], orient="v", palette="Set1")
sns.set(rc={'figure.figsize':(8, 8)})
ax.set_xticklabels(['x1', 'x2', 'x3'])
plt.title('DATA TEST SETELAH NORMALISASI')
```



4. Metode *Euclidean*

   Melakukan perhitungan *euclidean distance* sesuai dengan rumus

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ip} - x_{jp})^2}$$

```
start_time_kNN = time.time()

#metode Eucledian
def Euclidean(df_train_normalisasi, test):
    result = []
    for i in range(len(df_train_normalisasi)):
        jarak_x = math.sqrt((((df_train_normalisasi['x1'][i] - test['x1']) ** 2) +
                             ((df_train_normalisasi['x2'][i] - test['x2']) ** 2) +
                             ((df_train_normalisasi['x3'][i] - test['x3']) ** 2)))
        result.append([jarak_x, df_train_normalisasi['y'][i]])

    return result
```

5. Metode *Manhattan*

   Melakukan perhitungan *manhattan distance* sesuai dengan rumus

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{ip} - x_{jp}|$$

```
#metode Manhattan
def Manhattan(df_train_normalisasi, test):
    result = []
    for i in range(len(df_train_normalisasi)):
        jarak = (abs(df_train_normalisasi['x1'][i] - test['x1']) +
                 abs(df_train_normalisasi['x2'][i] - test['x2']) +
                 abs(df_train_normalisasi['x3'][i] - test['x3']))
        result.append([jarak , df_train_normalisasi['y'][i]])

    return result
```

6. Proses kNN

Melakukan proses kNN dengan memanggil metode euclidean dan manhattan untuk memproses penyortingan, pengambilan k data teratas serta melakukan label vote.

```
#memanggil fungsi Euclidean (sorting, k data teratas, melakukan label vote)
def kNN_Euclidean(df_train_normalisasi, df_test_normalisasi, k):
    hasil = []

    for i in range(len(df_test_normalisasi)):
        jarak = Euclidean(df_train_normalisasi, df_test_normalisasi.iloc[[i]])
        jarak = sorted(jarak, key=lambda x:x[0])
        jarak_K = jarak[:k]
        y_satu = 0
        y_nol = 0

        for j in range(k):
            if jarak_K[j][1] == 1:
                y_satu += 1
            else:
                y_nol += 1

        if y_satu > y_nol:
            hasil.append([df_test_normalisasi.loc[i, 'id'], 1])
        else:
            hasil.append([df_test_normalisasi.loc[i, 'id'], 0])

    df_Hasil = pd.DataFrame(hasil, columns = ['id', 'y'])

    return df_Hasil
```

```python
#memanggil fungsi Manhattan (sorting, k data teratas, melakukan label vote)
def kNN_Manhattan(df_train_normalisasi, df_test_normalisasi, k):
    hasil = []
    for i in range(len(df_test_normalisasi)):
        jarak = Manhattan(df_train_normalisasi, df_test_normalisasi.iloc[i])
        jarak = sorted(jarak, key=lambda x:x[0])
        jarak_K = jarak[:k]
        y_satu = 0
        y_nol = 0

        for j in range(k):
            if jarak_K[j][1] == 1:
                y_satu += 1
            else:
                y_nol += 1

        if y_satu > y_nol:
            hasil.append([df_test_normalisasi.loc[i, 'id'], 1])
        else:
            hasil.append([df_test_normalisasi.loc[i, 'id'], 0])

    df_Hasil = pd.DataFrame(hasil, columns = ['id', 'y'])

    return df_Hasil
```

```python
def kNN(df_train_normalisasi, df_test_normalisasi, k):
    return {
        'manhattan' : kNN_Manhattan(df_train_normalisasi, df_test_normalisasi, k),
        'euclidean' : kNN_Euclidean(df_train_normalisasi, df_test_normalisasi, k)
    }
```

7. Validasi

Untuk memastikan kebenaran dari perhitungan maka dilakukan proses validasi.

```
def validasi(df_train_normalisasi, k):
    #fold1
    test_fold1  = df_train_normalisasi.iloc[:59].drop('y', axis = 1)
    train_fold1 = df_train_normalisasi.iloc[59:].reset_index().drop('index', axis = 1)

    #fold2
    test_fold2  = df_train_normalisasi.iloc[59:118].drop('y', axis = 1).reset_index().drop('index', axis = 1)
    train_fold2 = pd.concat([df_train_normalisasi.iloc[:59], df_train_normalisasi.iloc[118:]]).reset_index().drop('index', axis = 1)

    #fold3
    test_fold3  = df_train_normalisasi.iloc[118:177].drop('y', axis = 1).reset_index().drop('index', axis = 1)
    train_fold3 = pd.concat([df_train_normalisasi.iloc[:118], df_train_normalisasi.iloc[177:]]).reset_index().drop('index', axis = 1)

    #fold4
    test_fold4  = df_train_normalisasi.iloc[177:236].drop('y', axis = 1).reset_index().drop('index', axis = 1)
    train_fold4 = pd.concat([df_train_normalisasi.iloc[:177], df_train_normalisasi.iloc[236:]]).reset_index().drop('index', axis = 1)

    #fold5
    test_fold5  = df_train_normalisasi.iloc[236:295].drop('y', axis = 1).reset_index().drop('index', axis = 1)
    train_fold5 = df_train_normalisasi.iloc[0:236].reset_index().drop('index', axis = 1)

    #hasil validasi
    hasil_fold1  = kNN(train_fold1, test_fold1, k)
    hasil_fold2  = kNN(train_fold2, test_fold2, k)
    hasil_fold3  = kNN(train_fold3, test_fold3, k)
    hasil_fold4  = kNN(train_fold4, test_fold4, k)
    hasil_fold5  = kNN(train_fold5, test_fold5, k)
    akurasi = []
```

```
valid = 0
#fold 1
for i in range(len(hasil_fold1['euclidean'])):
    if hasil_fold1['euclidean']['y'][i] == df_train_normalisasi['y'][i]:
        valid += 1
akurasi_fold = valid / len(hasil_fold1['euclidean'])
akurasi.append(akurasi_fold)

valid = 0
#fold 2
for i in range(len(hasil_fold1['euclidean'])):
    if hasil_fold2['euclidean']['y'][i] == df_train_normalisasi.iloc[59:118]['y'][i+59]:
        valid += 1
akurasi_fold = valid / len(hasil_fold1['euclidean'])
akurasi.append(akurasi_fold)

valid = 0
#fold 3
for i in range(len(hasil_fold1['euclidean'])):
    if hasil_fold3['euclidean']['y'][i] == df_train_normalisasi.iloc[118:177]['y'][i+118]:
        valid += 1
akurasi_fold = valid / len(hasil_fold1['euclidean'])
akurasi.append(akurasi_fold)

valid = 0
#fold 4
for i in range(len(hasil_fold1['euclidean'])):
    if hasil_fold4['euclidean']['y'][i] == df_train_normalisasi.iloc[177:236]['y'][i+177]:
        valid += 1
akurasi_fold = valid / len(hasil_fold1['euclidean'])
akurasi.append(akurasi_fold)

valid = 0
#fold 5
for i in range(len(hasil_fold1['euclidean'])):
    if hasil_fold5['euclidean']['y'][i] == df_train_normalisasi.iloc[236:295]['y'][i+236]:
        valid += 1
akurasi_fold = valid / len(hasil_fold1['euclidean'])
akurasi.append(akurasi_fold)


return akurasi
```

8. Menyimpan Output ke File

   Output dari *euclidean* disimpan ke file xlsx

```
dataResult = pd.ExcelWriter('Result_Euclidean.xlsx')
kNN['euclidean'].to_excel(dataResult)
dataResult.save()
```

   Output dari *manhattan* disimpan ke file xlsx

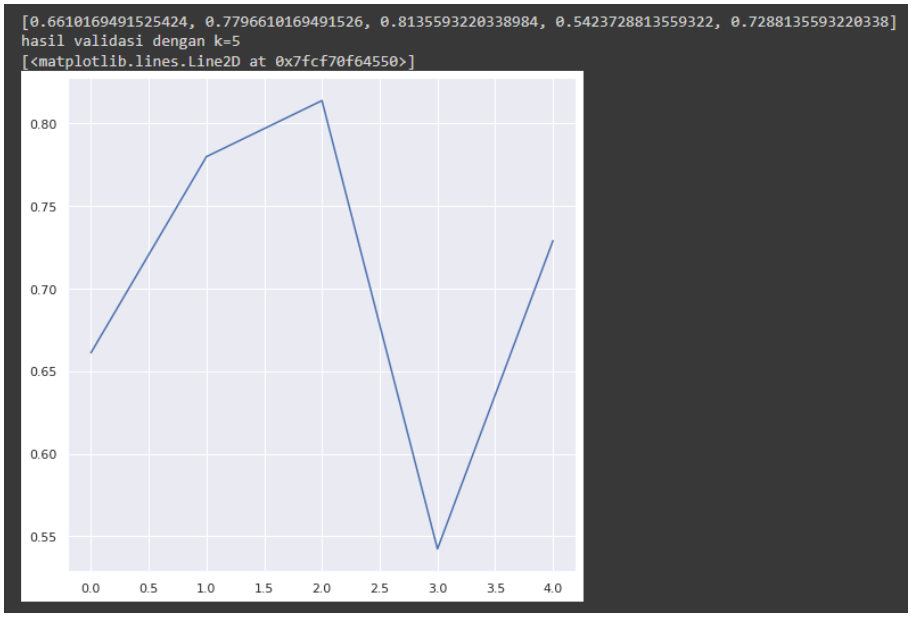
```
dataResult = pd.ExcelWriter('Result_Manhattan.xlsx')
kNN['manhattan'].to_excel(dataResult)
dataResult.save()
```

HASIL DAN EVALUASI

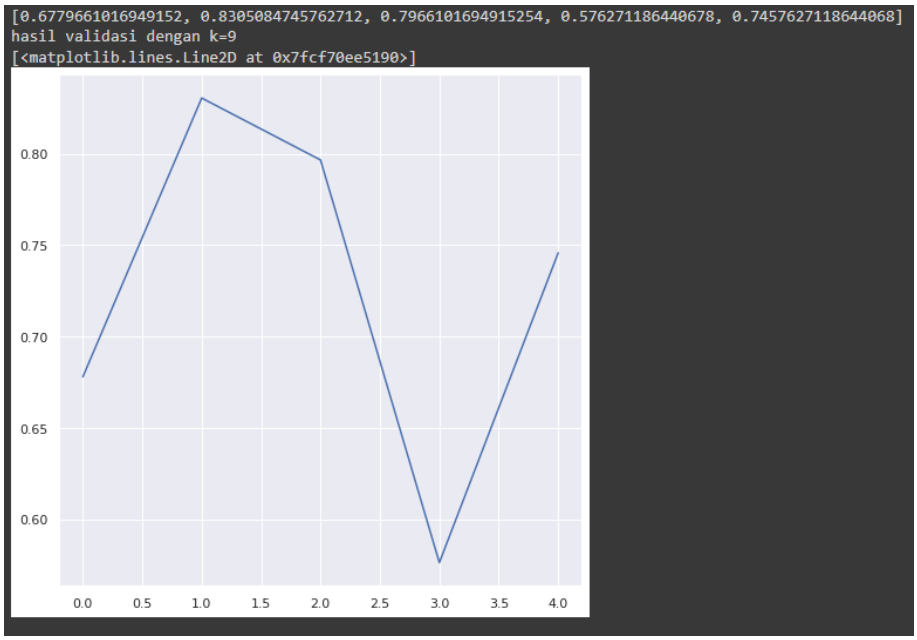
A. Hasil dan Evaluasi

Berikut merupakan hasil dan evaluasi dari proses implementasi algoritma *k-Nearest Neighbors* yang telah kami buat.

k = 5

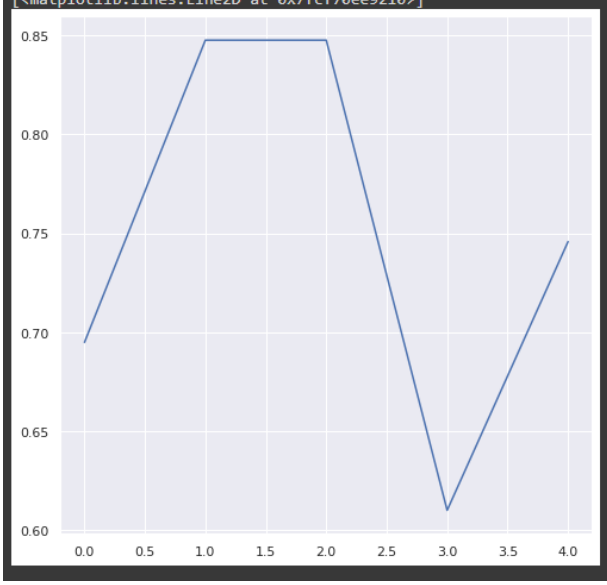


k = 9



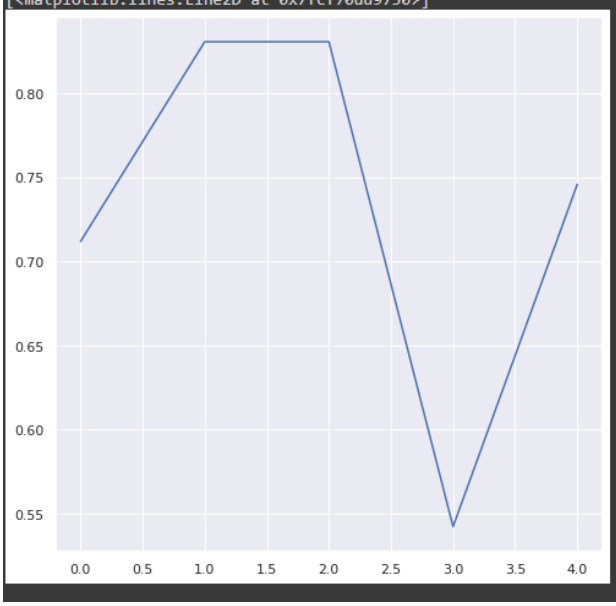
k = 15

```
[0.6949152542372882, 0.847457627118644, 0.847457627118644, 0.6101694915254238, 0.7457627118644068]
hasil validasi dengan k=15
[<matplotlib.lines.Line2D at 0x7fcf70ee9210>]
```



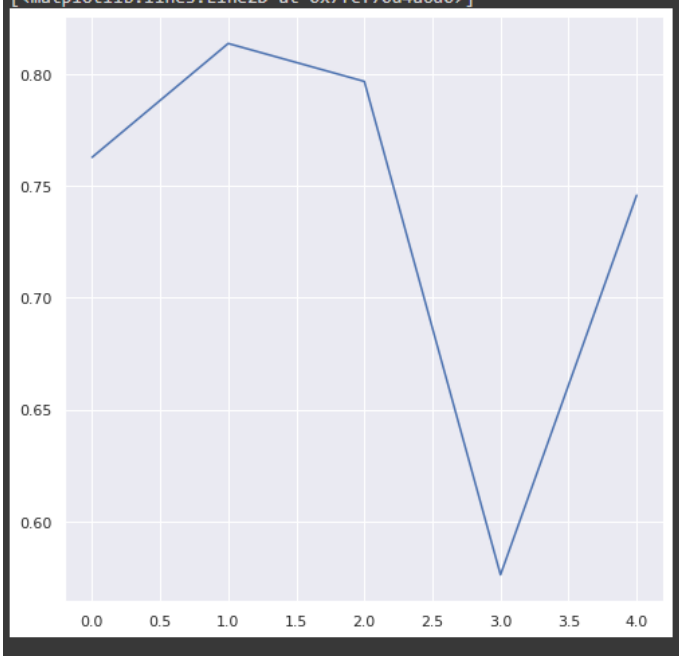
k = 19

```
[0.711864406779661, 0.8305084745762712, 0.8305084745762712, 0.5423728813559322, 0.7457627118644068]
hasil validasi dengan k=19
[<matplotlib.lines.Line2D at 0x7fcf70dd9750>]
```



k = 25

```
[0.7627118644067796, 0.8135593220338984, 0.7966101694915254, 0.576271186440678, 0.7457627118644068]
hasil validasi dengan k=25
[<matplotlib.lines.Line2D at 0x7fcf70d4a6d0>]
```



Tabel Hasil Evaluasi

k \ Fold	1	2	3	4	5
5	66%	77%	81%	54%	72%
9	67%	83%	79%	57%	74%
15	69%	**84%**	**84%**	61%	74%
19	71%	83%	83%	54%	74%
25	76%	81%	79%	57%	74%
Rata Rata	69,8%	81,6%	81,2%	56,6%	73,6%

B. Hasil Akhir

Sesuai hasil evaluasi sebelumnya, dimana akurasi tertinggi pada nilai k = 15 tepatnya, maka akan dilakukan proses *euclidean* dan *manhattan* untuk menentukan nilai y sebagai hasil akhir. Berikut hasil dari kedua proses tersebut.

Euclidean

	id	y
0	297	0
1	298	1
2	299	0
3	300	0
4	301	1
5	302	0
6	303	0
7	304	1
8	305	0
9	306	1

Manhattan

	id	y
0	297	1
1	298	1
2	299	0
3	300	0
4	301	1
5	302	0
6	303	1
7	304	1
8	305	0
9	306	1

Jadi, setelah menggunakan rumus *Min-Max Scaling* untuk proses normalisasi (*pre-processing*), beberapa metode dalam pencarian jarak (jarak atribut numerik), yakni *Euclidean* dan *Manhattan*, serta menggunakan *k-fold cross-validation* untuk proses validasi. Hasil akurasi tertinggi yang didapat adalah pada k = 15 dengan nilai sebesar 84%. Yang selanjutnya diproses pada *Euclidean* dan *Manhattan* agar memperoleh hasil akhir dari data testing (y) seperti pada gambar diatas.

PENUTUP

A. Kesimpulan

Secara sederhana, algoritma kNN bisa dituliskan hanya dengan dua langkah. Yang pertama adalah pelatihan, yaitu menyimpan setiap pola latih. Langkah kedua adalah klasifikasi. Setiap kali mengklasifikasikan sebuah pola, kNN harus memeriksa semua pola latih untuk menemukan sejumlah *k* pola terdekat. Proses pelatihan kNN menghasilkan nilai *k* yang yang diharapkan mampu memberikan akurasi tertinggi untuk menggeneralisasi data-data yang akan datang. Proses pelatihan pada dasarnya adalah melakukan observasi terhadap sejumlah *k* sampai dihasilkan *k* yang paling optimum. Jadi, algoritma kNN merupakan algoritma yang mudah diimplementasikan, karena hanya dengan mengatur satu parameter *k*. Akan tetapi, akurasinya sangat bergantung dengan bagus tidaknya dataset yang diberikan.

B. Daftar Pustaka

Sun, Shiliang, and Rongqing Huang. "An Adaptive K-Nearest Neighbor Algorithm." *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*, 2010. *Crossref*, https://doi.org/10.1109/fskd.2010.5569740.

Advernesia. "Pengertian dan Cara Kerja Algoritma K-Nearest Neighbors (KNN)." *Advernesia*, 29 Dec. 2020, www.advernesia.com/blog/data-science/pengertian-dan-cara-kerja-algoritma-k-nearest-neighbours-knn.

GeeksforGeeks. "K-Nearest Neighbours." *GeeksforGeeks*, 19 June 2022, www.geeksforgeeks.org/k-nearest-neighbours.

https://www.ibm.com/id-en/topics/knn

C. Lampiran

1. Link GitHub:
   https://github.com/berlianm/k-Nearest-Neighbors

2. Link Hasil Pengerjaan Pemrograman:
   https://colab.research.google.com/drive/1WkNTPgpfUHII9lnsH4liWHSCgPX-z4Vk?usp=sharing

3. Link Dokumen Laporan:
   https://docs.google.com/document/d/1yVjhN2IIy9c4lqb7p2kkEbxktpEHxe9YGa_3XU3SdNw/edit?usp=sharing

4. Link Dokumen Presentasi:
   https://docs.google.com/presentation/d/1IU9S1k2YZ1TA7Jkj_hetaOKDIiVu0Ox2ZjQn5N7eBJc/edit?usp=sharing

5. Link Hasil Akhir Testing (Excel):
   https://drive.google.com/drive/folders/1pBCqTFZwyJtx7gZQki-wJ8tq1DggIOpB?usp=sharing