K-Nearest Neighbor

Mohammad Febryan Khamim

1 Pengantar K-Nearest Neighbor (KNN)

K-Nearest Neighbor **(KNN)** adalah algoritma *supervised machine learning* yang umumnya digunakan untuk klasifikasi dan regresi. **Cara kerjanya** adalah dengan menemukan 'k' terdekat titik (tetangga) dari suatu data input baru.

Dari **k** tetangga terdekat tersebut, selanjutnya:

- **Prediksi** : Jumlah kelas terbanyak
- **Regresi** : Menentukan rata-rata **k** titik tersebut

KNN tidak mengasumsikan data terdistribusi tertentu (Normal, Gaussian, dan sebagainya) karena hanya berdasarkan jarak antardata. Selain itu, KNN tak perlu parameter yang banyak. Untuk menemukan model terbaik, hanya diperlukan sebuah langkah untuk menentukan nilai '**k**' paling optimal.

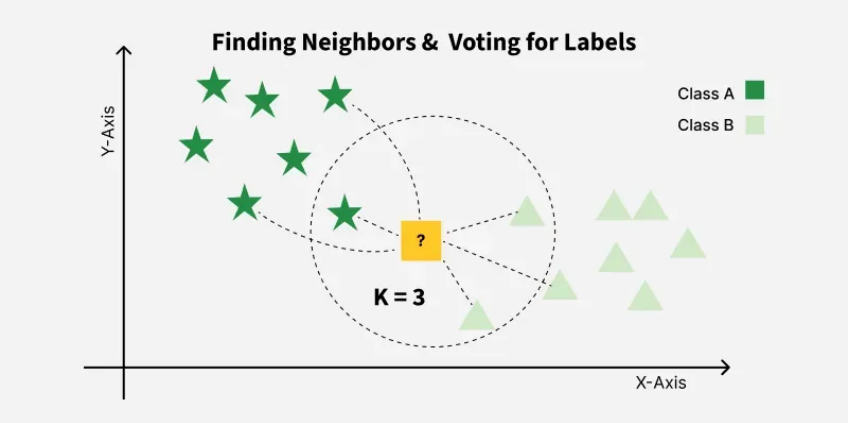

Figure 1: Ilustrasi K-Nearest Neighbors

K-Nearest Neighbor disebut sebagai algoritma pembelajar yang malas (*lazy learner*) karena algoritma ini tak melakukan pembelajaran langsung dari data latih. Algoritma ini menyimpan seluruh *dataset* dan baru melakukan perhitungan atau prediksi dilakukan.

2 Penentuan nilai 'k'

- '**k**' adalah jumlah tetangga yang hendak diamati ketika menentukan keputusan
- Memilih '**k**' penting untuk memutuskan hasil terbaik
- Data banyak *noise* atau *outlier* → Semakin besar nilai **k** maka semakin baik
- Semakin besar nilai **k** → Model gagal menangkap pola-pola penting → *underfitting*

- Dapat menggunakan : *Cross Validation & Elbow Method*
- Lebih baik digunakan nilai **k** ganjil

3 Kelebihan dan Kekurangan KNN

Kelebihan	Kekurangan
• Algoritma mudah dipahami • Tak perlu proses *training* yang rumit • Fleksibel untuk berbagai distribusi data • Mudah untuk menambahkan data	• Lambat untuk *dataset* besar • Butuh banyak memori • Sensitif terhadap skala data • Sensitif terhadap *noise / outlier* • Kurang efisien untuk fitur kategorikal

4 Penggunaan KNN

Algoritma dengan intuisi yang sederhana ini, dapat digunakan pada beberapa kasus, di antaranya sebagai berikut.

- Ukuran data kecil-menengah
- Jumlah fitur tak terlalu banyak
- Data bersih

5 Metrik Jarak yang Digunakan dalam KNN

Untuk menentukan jaraik antartitik pada KNN, terdapat beberapa metrik jarak yang digunakan, yakni sebagai berikut.

1. **Euclidean Distance**
$$\mathrm{d_{euc}}(x, x_i) = \sqrt{\sum_{j=1}^{d}(x_j - X_{ij})}$$
2. **Manhattan Distance**
$$\mathrm{d_{man}}(x, y) = \sum_{i=1}^{n}|x_i - y_i|$$
3. **Minkowski Distance**
$$\mathrm{d_{min}} = (\sum_{i=1}^{n}(x_i - y_1)^p)^{\frac{1}{p}}$$

6 Algoritma KNN

Berikut ini adalah langkah-langkah atau algoritma dalam menyelesaikan tugas ML, yakni:

1. Menentukan nilai **k** optimal

2. Menghitung dan menentukan **k** tetangga terdekat
3. Menghitung jarak titik saat ini dengan **k** tetangga terdekat
4. Voting : Klasifikasi | Rata-rata : Regresi

7 Catatan

- Dalam KNN, data atau fitur yang memiliki *range* nilai yang lebih besar akan sangat memengaruhi perhitungan jarak → butuh standardisasi.
- Bagaimana memilih Metrik Jarak terbaik?
  - Euclidean : *Default* untuk data yang bersih
  - Manhattan : Lebih tahan *outlier* dan data grid
  - Minkowski : Fleksibel dan bisa disesuaikan

8 Source Code Python

Listing 1: Python Code

```
# Import libraries yang dibutuhkan
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# Load Dataset
df = pd.read_csv("Classified Data",index_col=0)
df.head()

# Standardisasi data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])

# Train Test Split
from sklearn.model_selection import train_test_split
X = df_feat
y = df['TARGET CLASS']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
   =0.30, random_state=101)

# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)

# Evaluasi Model
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))

# Menentukan nilai K yang optimal
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

# Plotting error rate vs K value
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
   marker='o',
```

```
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')

# Menggunakan k=1
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))

# Menggunakan k optimal
# NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```