Pokok Bahasan V
K-Nearest Neighbor (KNN).

Kode Pokok Bahasan: TIK.RPL03.004.00.01

Deskripsi Pokok Bahasan:
Membahas bagaimana penerapan K-Nearest Neighbor untuk melakukan Klasifikasi.

No	Elemen Kompetensi	Indikator Kinerja	Jml Jam	Hal
1.	Menampilkan hasil Klasifikasi dari kasus yang diberikan.	Mampu melakukan analisis terhadap klasifikasi data dari diagram yangmuncul.	1	
2.	Melakukan perhitungan hasil prediksi dengan confusion matriks.	Melakukan perhitungan hasil prediksi dengan confusion matriks berdasarkan dataset pada studi kasus.	1	

TUGAS PENDAHULUAN
Hal yang harus dilakukan dan acuan yang harus dibaca sebelum praktikum :
1. Menginstal Python pada PC masing-masing praktikan.
2. Menginstal Jupyter notebook pada PC masing-masing praktikan.
3. Menginstal library scikit-learn.

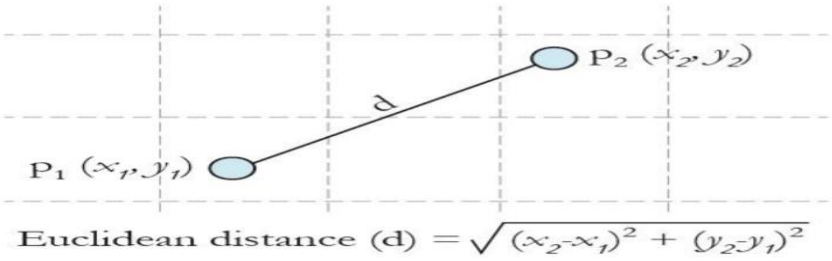
DAFTAR PERTANYAAN
1. Apa itu algoritma K-Nearest Neighbor?
algoritma untuk melakukan klasifikasi terhadap objek dengan data pembe- lajaran yang jaraknya paling dekat dengan objek tersebut.
2. Apa kegunaan K-Nearest Neighbor?
memiliki keunggulan dapat mengklasifikasikan data calon pegawai yang tidak diketahui dengan adanya data latih dan data uji. KNN dapat menprosedur yang berbasis matematis untuk mengevaluasi nilai kriteria-kriteria tersebut menjadi sebuah keterangan klasifikasi.
3. Sebutkan tahapan dari proses algoritma K-Nearest Neighbor!
Langkah-1: Pilih nilai banyaknya tetangga K.
Langkah-2: Hitung jarak dari jumlah tetangga K (bisa menggunakan salah satu metrik jarak, misalnya Euclidean distance)
Langkah-3: Ambil tetangga terdekat K sesuai jarak yang dihitung.

TEORI SINGKAT

Algoritma K-Nearest Neighbor (K-NN) adalah sebuah metode klasifikasi terhadap sekumpulan data berdasarkan pembelajaran data yang sudah terklasifikasikan sebelumya. Termasuk dalam supervised learning, dimana hasil query instance yang

baru diklasifikasikan berdasarkan mayoritas kedekatan jarak dari kategori yang ada dalam K-NN

KNN, Dapat digunakan untuk tujuan klasifikasi, Tidak menyusun model atau mengekstrak aturan logika tertentu sebagai hasil dari analisis, Identikasi k buah individu tetangga terdekat dilakukan dengan terlebih dahulu menghitung jarak dari individu yang akan diduga dengan setiap individu yang ada pada gugus data training. Jika ini sudah dilakukan maka tinggal mencari k buah amatan yang jaraknya paling kecil. Penghitungan jarak dari dua amatan A dan B dapat menggunakan formula Euclid distance.



LAB SETUP

Hal yang harus disiapkan dan dilakukan oleh praktikan untuk menjalankan praktikum modul ini.

1. Menginstall library yang dibutuhkan untuk mengerjakan modul.
2. Menjalankan Jupyter notebook.

ELEMEN KOMPETENSI I

Deskripsi:
Menampilkan hasil klasifikasi dari kasus yang diberikan.

Kompetensi Dasar:
Menampilkan hasil Klasifikasi dari kasus yang diberikan.

Latihan 1.1.1
Penjelasan Singkat :
Pada latihan ini anda akan diminta melakukan analisis terhadap klasifikasi data dari diagram yang muncul.

Langkah- Langkah Praktikum :
1. import library yang dibutuhkan

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
```

from sklearn.model_selection import train_test_split
import seaborn as sns

```
In [1]: import numpy as np
        import pandas as pd
        from matplotlib import pyplot as plt
        from sklearn.datasets import load_breast_cancer
        from sklearn.metrics import confusion_matrix
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.model_selection import train_test_split
        import seaborn as sns
```

2. Menampilkan column features

breast_cancer.feature_names

```
In [15]: breast_cancer = load_breast_cancer()

In [16]: breast_cancer.feature_names

Out[16]: array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
                'mean smoothness', 'mean compactness', 'mean concavity',
                'mean concave points', 'mean symmetry', 'mean fractal dimension',
                'radius error', 'texture error', 'perimeter error', 'area error',
                'smoothness error', 'compactness error', 'concavity error',
                'concave points error', 'symmetry error',
                'fractal dimension error', 'worst radius', 'worst texture',
                'worst perimeter', 'worst area', 'worst smoothness',
                'worst compactness', 'worst concavity', 'worst concave points',
                'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

3. Menampilkan class target

breast_cancer.target_names

```
In [17]: breast_cancer.target_names

Out[17]: array(['malignant', 'benign'], dtype='<U9')
```

4. Menyeseleksi features column dan mngubah class target menjadi categorical

X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
X = X[['mean area', 'mean compactness']]
y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
y = pd.get_dummies(y, drop_first=True)

```
In [18]: X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
         X = X[['mean area', 'mean compactness']]
         y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
         y = pd.get_dummies(y, drop_first=True)
```

5. Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

```
In [19]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

6. Memanggil dan menjalankan algoritma KNN

knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)

```
In [20]: knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
         knn.fit(X_train, y_train)

<ipython-input-20-52466d8b3ccf>:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples, ), for example using ravel().
  knn.fit(X_train, y_train)

Out[20]: KNeighborsClassifier(metric='euclidean')
```

7. Melakukan prediksi dan menampilkan hasil prediksi

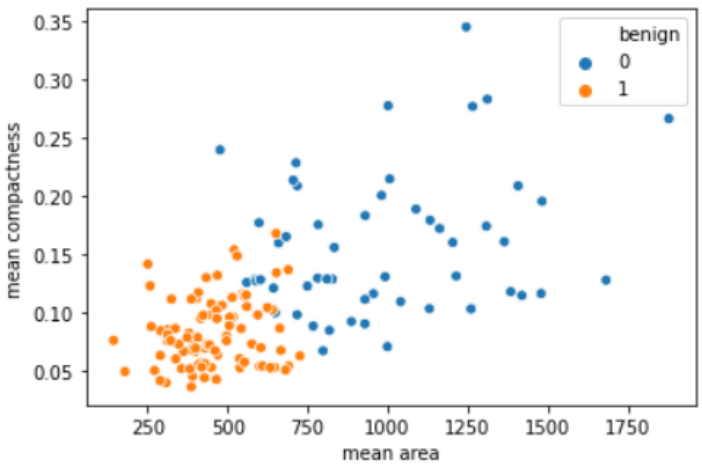
```
y_pred = knn.predict(X_test)
y_pred
```

```
In [21]: y_pred = knn.predict(X_test)
         y_pred

Out[21]: array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
                0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
                0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0,
                1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1], dtype=uint8)
```

8. Visualisasi dengan sns

```
sns.scatterplot(
   x='mean area',
   y='mean compactness',
   hue='benign',
   data=X_test.join(y_test, how='outer')
)
```

```
In [22]: sns.scatterplot(
             x='mean area',
             y='mean compactness',
             hue='benign',
             data=X_test.join(y_test, how='outer')
         )

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1afa9f6a190>
```

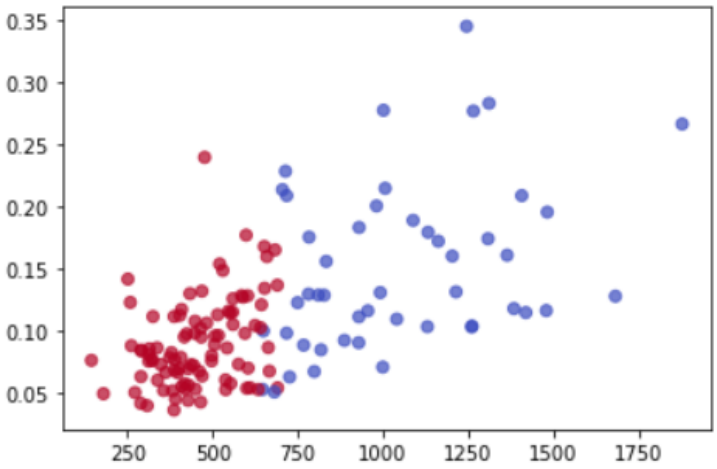


9. visualisasi dengan matplotlib

```
plt.scatter(
```

```
    X_test['mean area'],
    X_test['mean compactness'],
    c=y_pred,
    cmap='coolwarm',
    alpha=0.7
)
```

```
In [23]: plt.scatter(
             X_test['mean area'],
             X_test['mean compactness'],
             c=y_pred,
             cmap='coolwarm',
             alpha=0.7
         )

Out[23]: <matplotlib.collections.PathCollection at 0x1afaa076fd0>
```



10. Menampilkan hasil Confusion matriks hasil prediksi

```
confusion_matrix(y_test, y_pred)
```

```
In [24]: confusion_matrix(y_test, y_pred)

Out[24]: array([[45,  9],
                [ 3, 86]], dtype=int64)
```

Interpretasi Confusion matriks :

DIISISI
Interpretasi yang bisa dijelaskan adalah banyaknya data yang didapat dari berbagai sumber, maka pemrosesan dan analisis data yang efisien menjadi sulit, dikarenakan distribusi data yang tidak merata antar kelas.

Elemen Komptensi 1.2.1

Tugas Laporan :

1. Siapkan data iris.csv

2. Buka Jupyter Notebook

3. Mengimport module

```
from random import seed
from random import randrange
from csv import reader
from math import sqrt
```

4. Melakukan load file CSV

```
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
            csv_reader = reader(file)
            for row in csv_reader:
                    if not row:
                            continue
                    dataset.append(row)
    return dataset
```

5. Mengubah string menajdi float

```
def str_column_to_float(dataset, column):
    for row in dataset:
            row[column] = float(row[column].strip())
```

6. Mengubah string menjadi integer

```
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
            lookup[value] = i
    for row in dataset:
            row[column] = lookup[row[column]]
    return lookup
```

7. Mencari nilai min dan max untuk setiap kolom

```
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
            col_values = [row[i] for row in dataset]
            value_min = min(col_values)
            value_max = max(col_values)
            minmax.append([value_min, value_max])
```

```
    return minmax
```

8. Mengubah skala data

```
    def normalize_dataset(dataset, minmax):
        for row in dataset:
            for i in range(len(row)):
                row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] -
    minmax[i][0])
```

9. Melakukan split data

```
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

10. Menghitung akurasi

```
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

11. Mengevaluasi algoritma menggunakan cross validation split

```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores
```

12. Menghitung jarak Euclidean

```
def euclidean_distance(row1, row2):
        distance = 0.0
        for i in range(len(row1)-1):
                distance += (row1[i] - row2[i])**2
        return sqrt(distance)
```

13. Menemukan neighors yang paling mirip

```
def get_neighbors(train, test_row, num_neighbors):
        distances = list()
        for train_row in train:
                dist = euclidean_distance(test_row, train_row)
                distances.append((train_row, dist))
        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(num_neighbors):
                neighbors.append(distances[i][0])
        return neighbors
```

14. Membuat prediksi dengan neighbors

```
def predict_classification(train, test_row, num_neighbors):
        neighbors = get_neighbors(train, test_row, num_neighbors)
        output_values = [row[-1] for row in neighbors]
        prediction = max(set(output_values), key=output_values.count)
        return prediction
```

15. Algoritma KNN

```
def k_nearest_neighbors(train, test, num_neighbors):
        predictions = list()
        for row in test:
                output = predict_classification(train, row, num_neighbors)
                predictions.append(output)
        return(predictions)
```

16. Menjalankan KNN

```
    # Test the kNN on the Iris Flowers dataset
    seed(1)
    filename = 'knn.csv'
    dataset = load_csv(filename)
    for i in range(len(dataset[0])-1):
        str_column_to_int(dataset, i)
    # convert class column to integers
    str_column_to_int(dataset, len(dataset[0])-1)
    # evaluate algorithm
    n_folds = 5
    num_neighbors = 5
    scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds,
    num_neighbors)
    print('Scores: %s' % scores)
```

```
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

# define model parameter
num_neighbors = 5
# define a new record
row = [0.4, 0.1]
# predict the label
label = predict_classification(dataset, row, num_neighbors)
print('Data=%s, Predicted: %s' % (row, label))
```

Screenshot tiap cell code:

In [1]:
```
from random import seed
from random import randrange
from csv import reader
from math import sqrt
```

In [16]:
```
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset
```

In [17]:
```
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())
```

In [18]:
```
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup
```

In [19]:
```
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax
```

In [20]:
```
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
```

In [21]:
```
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for _ in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split
```

In [22]:
```
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

In [23]:
```
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
    for fold in folds:
        train_set = list(folds)
        train_set.remove(fold)
        train_set = sum(train_set, [])
        test_set = list()
        for row in fold:
            row_copy = list(row)
            test_set.append(row_copy)
            row_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [row[-1] for row in fold]
        accuracy = accuracy_metric(actual, predicted)
        scores.append(accuracy)
    return scores
```

In [24]:
```
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)
```

In [25]:
```
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

In [26]:
```
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

In [27]:
```
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    return(predictions)
```

In [30]:
```
# Test the kNN on the Iris Flowers dataset
seed(1)
filename = 'D:/File Kuliah Semester 5/Penambangan Data/Prak-6/knn.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_int(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# evaluate algorithm
n_folds = 5
num_neighbors = 5
scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds, num_neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

# define model parameter
num_neighbors = 5
# define a new record
row = [0.4, 0.1]
# predict the label
label = predict_classification(dataset, row, num_neighbors)
print('Data=%s, Predicted: %s' % (row, label))
```

```
Scores: [75.0, 50.0, 75.0, 50.0, 50.0]
Mean Accuracy: 60.000%
Data=[0.4, 0.1], Predicted: 1
```

CEK LIST

Elemen Kompetensi	No Latihan	Penyelesaian	
		Selesai	Tidak selesai
1	1.1.1	✓	
2	1.2.1	✓	

FORM UMPAN BALIK

Elemen Kompetensi	**Tingkat Kesulitan**	**Tingkat Ketertarikan**	**Waktu Penyelesaian dalam menit**
Menampilkan hasil Klasifikasi dari kasus yang diberikan. yang diberikan.	☐ Sangat Mudah ☐ Mudah ☐ ✓ Biasa ☐ Sulit ☐ Sangat Sulit	☐ Tidak Tertarik ☐ Cukup Tertarik ☐ Tertarik ☐ ✓ Sangat Tertarik	
Melakukan perhitungan hasil prediksi dengan confusion matriks.	☐ Sangat Mudah ☐ Mudah ☐ ✓ Biasa ☐ Sulit ☐ Sangat Sulit	☐ Tidak Tertarik ☐ Cukup Tertarik ☐ Tertarik ☐ ✓ Sangat Tertarik	