Classification dengan KNN (K Nearest Neighbours)

Muhammad Rofi Ariansyah

41155050210066

- KNN adalah model machine learning yang dapat digunakan untuk melakukan prediksi berdasarkan kedekatan karakteristik dengan sejumlah tetangga terdekat.
- Prediksi yang dilakukan dapat diterapkan baik pada classification maupun regression tasks.

Sample Dataset

[1]:
```
import pandas as pd

sensus = {
    'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
    'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
    'jk': [
        'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
        'wanita'
    ]
}

sensus_df = pd.DataFrame(sensus)
sensus_df
```

[1]:

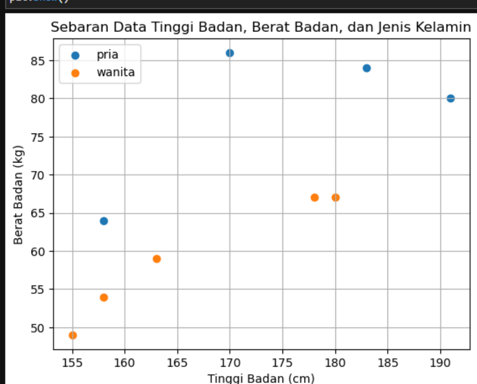
	tinggi	berat	jk
0	158	64	pria
1	170	86	pria
2	183	84	pria
3	191	80	pria
4	155	49	wanita
5	163	59	wanita
6	180	67	wanita
7	158	54	wanita
8	178	67	wanita

Visualisasi Data

[2]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
for jk, d in sensus_df.groupby('jk'):
    ax.scatter(d['tinggi'], d['berat'], label=jk)

plt.legend(loc='upper left')
plt.title('Sebaran Data Tinggi Badan, Berat Badan, dan Jenis Kelamin')
plt.xlabel('Tinggi Badan (cm)')
plt.ylabel('Berat Badan (kg)')
plt.grid(True)
plt.show()
```



Classification dengan KNN

Preprocessing Dataset

[3]:
```
import numpy as np

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

print(f'X_train:\n{X_train}\n')
print(f'y_train: {y_train}')
```

```
X_train:
[[158  64]
 [170  86]
 [183  84]
 [191  80]
 [155  49]
 [163  59]
 [180  67]
 [158  54]
 [178  67]]

y_train: ['pria' 'pria' 'pria' 'pria' 'wanita' 'wanita' 'wanita' 'wanita' 'wanita']
```

[4]:
```
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
print(f'y_train:\n{y_train}')
```

```
y_train:
[[0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]]
```

[5]:
```
y_train = y_train.flatten()
print(f'y_train: {y_train}')
```

```
y_train: [0 0 0 0 1 1 1 1 1]
```

Training KNN Classification Model

[6]:
```
from sklearn.neighbors import KNeighborsClassifier

K = 3
model = KNeighborsClassifier(n_neighbors=K)
model.fit(X_train, y_train)
```

[6]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

Prediksi Jenis Kelamin

[7]:
```
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
X_new
```

[7]:
```
array([[155,  70]])
```

[8]:
```
y_new = model.predict(X_new)
y_new
```

[8]:
```
array([1])
```

[9]:
```
lb.inverse_transform(y_new)
```

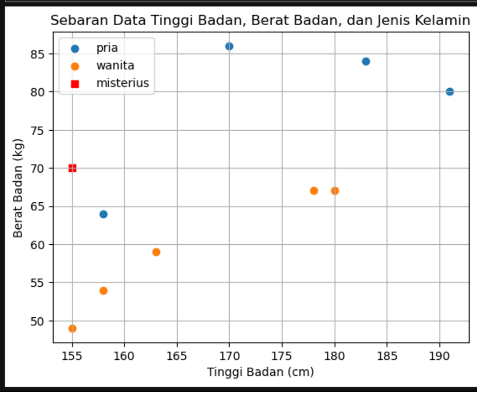
[9]:
```
array(['wanita'], dtype='<U6')
```

Visualisasi Nearest Neighbours

[10]:
```python
fig, ax = plt.subplots()
for jk, d in sensus_df.groupby('jk'):
    ax.scatter(d['tinggi'], d['berat'], label=jk)

plt.scatter(tinggi_badan,
            berat_badan,
            marker='s',
            color='red',
            label='misterius')

plt.legend(loc='upper left')
plt.title('Sebaran Data Tinggi Badan, Berat Badan, dan Jenis Kelamin')
plt.xlabel('Tinggi Badan (cm)')
plt.ylabel('Berat Badan (kg)')
plt.grid(True)
plt.show()
```



Kalkulasi Distance (Euclidean Distance)

$$distance = \sqrt{(t_1 - t_2)^2 + (b_1 - b_2)^2}$$

[11]:
```python
misterius = np.array([tinggi_badan, berat_badan])
misterius
```

[11]:
```
array([155,  70])
```

[12]:
```python
X_train
```

[12]:
```
array([[158,  64],
       [170,  86],
       [183,  84],
       [191,  80],
       [155,  49],
       [163,  59],
       [180,  67],
       [158,  54],
       [178,  67]], dtype=int64)
```

[13]:
```python
from scipy.spatial.distance import euclidean

data_jarak = [euclidean(misterius, d) for d in X_train]
data_jarak
```

[13]:
```
[6.708203932499369,
 21.93171219946131,
 31.304951684997057,
 37.36308338453881,
 21.0,
 13.601470508735444,
 25.179356624028344,
 16.278820596099706,
 23.194827009486403]
```

[14]:
```python
sensus_df['jarak'] = data_jarak
sensus_df.sort_values(['jarak'])
```

[14]:

	tinggi	berat	jk	jarak
0	158	64	pria	6.708204
5	163	59	wanita	13.601471
7	158	54	wanita	16.278821
4	155	49	wanita	21.000000
1	170	86	pria	21.931712
8	178	67	wanita	23.194827
6	180	67	wanita	25.179357
2	183	84	pria	31.304952
3	191	80	pria	37.363083

Evaluasi KNN Classification Model

Testing Set

[15]:
```
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

```
X_test:
[[168  65]
 [180  96]
 [160  52]
 [169  67]]

y_test:
[0 0 1 1]
```

Prediksi terhadap testing set

[16]:
```
y_pred = model.predict(X_test)
y_pred
```

[16]:
```
array([1, 0, 1, 1])
```

Accuracy

Accuracy is the proportion of test instances that were classified correctly.

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

[17]:
```
from sklearn.metrics import accuracy_score

acc = accuracy_score(y_test, y_pred)

print(f'Accuracy: {acc}')
```

```
Accuracy: 0.75
```

Precission

Precision is the proportion of test instances that were predicted to be positive that are truly positive.

$$precission = \frac{tp}{tp + fp}$$

[18]:
```
from sklearn.metrics import precision_score

prec = precision_score(y_test, y_pred)

print(f'Precission: {prec}')
```

```
Precission: 0.6666666666666666
```

Recall

Recall is the proportion of truly positive test instances that were predicted to be positive.

$$recall = \frac{tp}{tp + fn}$$

[19]:
```
from sklearn.metrics import recall_score

rec = recall_score(y_test, y_pred)

print(f'Recall: {rec}')
```

```
Recall: 1.0
```

F1 Score

The F1 score is the harmonic mean of precision and recall.

$$F1 = 2 \times \frac{precission \times recall}{precission + recall}$$

[20]:
```
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_pred)

print(f'F1-score: {f1}')
```

```
F1-score: 0.8
```

Classification Report

[21]:
```python
from sklearn.metrics import classification_report

cls_report = classification_report(y_test, y_pred)

print(f'Classification Report:\n{cls_report}')
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.67      1.00      0.80         2

    accuracy                           0.75         4
   macro avg       0.83      0.75      0.73         4
weighted avg       0.83      0.75      0.73         4
```

Matthews Correlation Coefficient (MCC)

- MCC is an alternative to the F1 score for measuring the performance of binary classifiers.
- A perfect classifier's MCC is 1.
- A trivial classifier that predicts randomly will score 0, and a perfectly wrong classifier will score -1.

$$MCC = \frac{tp \times tn + fp \times fn}{\sqrt{(tp + fp) \times (tp + fn) \times (tn + fp) \times (tn + fn)}}$$

[22]:
```python
from sklearn.metrics import matthews_corrcoef

mcc = matthews_corrcoef(y_test, y_pred)

print(f'MCC: {mcc}')
```

```
MCC: 0.5773502691896258
```

[ ]:

Classification Task dengan Support Vector Machine (SVM)

Muhammad Rofi Ariansyah

41155050210066

Dataset: The MNIST database of handwritten digits

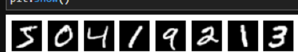
[*]:
```python
from sklearn.datasets import fetch_openml

X, y = fetch_openml('mnist_784', data_home='./dataset/mnist', return_X_y=True)
X.shape
```

[2]:
```python
import matplotlib.pyplot as plt
import matplotlib.cm as cm

pos = 1
for data in X.to_numpy()[:8]:
    plt.subplot(1, 8, pos)
    plt.imshow(data.reshape((28, 28)),
               cmap=cm.Greys_r)
    plt.axis('off')
    pos += 1

plt.show()
```



[3]:
```python
y[:8]
```

[3]:
```
0    5
1    0
2    4
3    1
4    9
5    2
6    1
7    3
Name: class, dtype: category
Categories (10, object): ['0', '1', '2', '3', ..., '6', '7', '8', '9']
```

[4]:
```python
# X_train = X[:60000]
# y_train = y[:60000]
# X_test = X[60000:]
# y_test = y[60000:]

X_train = X[:1000]
y_train = y[:1000]
X_test = X[69000:]
y_test = y[69000:]
```

Classification dengan SVC (Support Vector Classifier)

[5]:
```python
from sklearn.svm import SVC

model = SVC(random_state=0)
model.fit(X_train, y_train)
```

[5]:
```
SVC(random_state=0)
```

[6]:
```python
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       102
           1       0.97      0.99      0.98       119
           2       0.85      0.82      0.84        99
           3       0.97      0.87      0.92       102
           4       0.88      0.95      0.91        92
           5       0.91      0.86      0.88        85
           6       0.93      0.95      0.94       102
           7       0.92      0.94      0.93       115
           8       0.89      0.94      0.91        94
           9       0.92      0.84      0.88        90

    accuracy                           0.92      1000
   macro avg       0.92      0.91      0.91      1000
weighted avg       0.92      0.92      0.92      1000
```

Hyperparameter Tuning dengan `GridSearchCV`

[7]:
```python
from sklearn.model_selection import GridSearchCV

parameters = {
    'kernel': ['rbf', 'poly', 'sigmoid'],
    'C': [0.5, 1, 10, 100],
    'gamma': ['scale', 1, 0.1, 0.01, 0.001]
}

grid_search = GridSearchCV(estimator=SVC(random_state=0),
                           param_grid=parameters,
                           n_jobs=6,
                           verbose=1,
                           scoring='accuracy')

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
```

[7]:
```
GridSearchCV(estimator=SVC(random_state=0), n_jobs=6,
             param_grid={'C': [0.5, 1, 10, 100],
                         'gamma': ['scale', 1, 0.1, 0.01, 0.001],
                         'kernel': ['rbf', 'poly', 'sigmoid']},
             scoring='accuracy', verbose=1)
```

[8]:
```python
print(f'Best Score: {grid_search.best_score_}')

best_params = grid_search.best_estimator_.get_params()
print(f'Best Parameters:')
for param in parameters:
    print(f'\t{param}: {best_params[param]}')
```

```
Best Score: 0.907
Best Parameters:
	kernel: rbf
	C: 10
	gamma: scale
```

Predict & Evaluate

[9]:
```python
y_pred = grid_search.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       102
           1       0.98      0.99      0.98       119
           2       0.87      0.85      0.86        99
           3       0.99      0.89      0.94       102
           4       0.91      0.95      0.93        92
           5       0.92      0.89      0.90        85
           6       0.93      0.94      0.94       102
           7       0.93      0.93      0.93       115
           8       0.89      0.95      0.92        94
           9       0.92      0.88      0.90        90

    accuracy                           0.93      1000
   macro avg       0.93      0.92      0.92      1000
weighted avg       0.93      0.93      0.93      1000
```