บทที่ 6 k-Nearest Neighbors

หัวข้อหลัก

- โมเดล k-Nearest Neighbors คือโมเดลการทำนายค่า ที่คิดขึ้นจากสมมติฐานว่า จุดข้อมูลที่อยู่ใกล้เคียงกันจะมีความคล้ายคลึงกัน
- เราสามารถวัดความใกล้เคียงกันของจุดข้อมูลโดยการคำนวณค่าระยะห่างระหว่างจุดข้อมูลในเวกเตอร์สเปซ
- การใช้ k-Nearest Neighbors จำแนกประเภทจุดข้อมูลบน Iris Dataset

6.1 โมเดล k-Nearest Neighbors

โมเดล k-Nearest Neighbors หรือ k-NN classifier เป็นโมเดลการทำนายค่า ที่คิดขึ้นบนสมมติฐานที่กล่าวว่า จุดข้อมูลที่อยู่ใกล้เคียงกัน มักจะมีความคล้ายคลึงกัน มากกว่าจุดข้อมูลที่อยู่ห่างไกลกัน ฉะนั้นเราจึงสามารถทำนายค่าหรือคลาสลาเบลของจุดข้อมูลใหม่ได้โดยดูจากจุดข้อมูลที่อยู่ใกล้เคียงจำนวนหนึ่ง (k จำนวน) แทนที่จะต้องตรวจสอบจุดข้อมูลทั้งหมดแล้วสร้างเป็นโมเดลของรูปแบบในชุดข้อมูล

สิ่งที่จำเป็นต่อการทำงานของ k-NN classifier ก็คือ วิธีการวัดความใกล้เคียงกันระหว่างจุดข้อมูลสองจุด เพื่อให้สามารถเปรียบเทียบได้ว่า จุดข้อมูลคู่ใดมีความใกล้เคียงกันมากที่สุดตั้งแต่ลำดับที่ 1 จนถึงลำดับที่ k ในกรณีที่เราแทนจุดข้อมูลแต่ละจุดด้วยเวกเตอร์ เราสามารถใช้ค่าระยะทางระหว่างจุดสองจุดในเวกเตอร์สเปซในการวัดค่าความใกล้เคียงกันได้ โดยคู่ของจุดที่มีค่าระยะห่างกันน้อยจะมีความใกล้เคียงกันมากกว่าคู่ของจุดที่มีค่าระยะห่างกันมากกว่า

ตัวอย่างเช่น ชุดข้อมูลไอริส [2] ประกอบด้วยข้อมูลของดอกไอริสจำนวน 150 ดอก แต่ละจุดข้อมูลแทนดอกไม้แต่ละดอก ซึ่งมีฟีเจอร์สี่ตัว ได้แก่ ความยาวกลีบดอก (petal length), ความกว้างกลีบดอก (petal width), ความยาวกลีบเลี้ยง (sepal length), และความกว้างกลีบเลี้ยง (sepal width) และมี class label คือสปีชีส์ของดอกไอริส ซึ่งมีค่าที่เป็นไปได้สามค่าคือ ‘Iris Setosa’, ‘Iris Versicolour’, และ ‘Iris Virginica’ ดังแสดงในรูปที่ 6.1

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

รูปที่ 6.1 ตัวอย่าง Iris Dataset

เราสามารถแทนจุดข้อมูลจุดแรก ด้วยเวกเตอร์ $[5.1\ 3.5\ 1.4\ 0.2]^T$ และจุดข้อมูลที่สองด้วยเวกเตอร์ $[4.9\ 3.0\ 1.4\ 0.2]^T$ ระยะห่างระหว่างเวกเตอร์ทั้งสองสามารถหาได้โดยใช้ Euclidean Distance ดังสมการ (6.1)

$$Euclidean\ Distance\ (\bar{u}, \bar{v}) = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2} \qquad \textbf{(6.1)}$$

ระยะห่างระหว่างเวกเตอร์ $\begin{bmatrix}\mathbf{5.1}\\ \mathbf{3.5}\\ \mathbf{1.4}\\ \mathbf{0.2}\end{bmatrix}$ และ $\begin{bmatrix}\mathbf{4.9}\\ \mathbf{3.0}\\ \mathbf{1.4}\\ \mathbf{0.2}\end{bmatrix}$

$$= \sqrt{(5.1-4.9)^2 + (3.5-3.0)^2 + (1.4-1.4)^2 + (0.2-0.2)^2}$$

$$= \sqrt{0.04 + 0.25 + 0 + 0}$$

$$= 0.5385$$

Pseudo-code ของ k-NN classifier แสดงในรูปที่ 6.2 ซอร์สโค้ดภาษาไพธอนของ k-NN classifier แสดงในรูปที่ 6.3

```
kNN_Classify(k, training_dataset, new_datapoint)
    Sort training_dataset by the distance to the new_datapoint ascendingly
    kNN_labels = labels of k points nearest to the new_datapoint
    Return Majority_Vote( kNN_labels )

Majority_Vote( labels )
    If there is only one most common class in labels
        return the most common class label
    If there are more than one most common class in labels
        Remove the label of the farthest neighbor from labels
        return Majority_Vote( labels )
```

รูปที่ 6.2 Pseudo-code ของ k-NN classifier

```python
from typing import List, NamedTuple
from collections import Counter
import math

Vector = List[float]

class LabeledPoint(NamedTuple):
    point: Vector
    label: str

def subtract(v: Vector, w: Vector) -> Vector:
    """Subtracts corresponding elements"""
    assert len(v) == len(w), "vectors must be the same length"

    return [v_i - w_i for v_i, w_i in zip(v, w)]

def dot(v: Vector, w: Vector) -> float:
    """Computes v_1 * w_1 + ... + v_n * w_n"""
    assert len(v) == len(w), "vectors must be same length"

    return sum(v_i * w_i for v_i, w_i in zip(v, w))

def sum_of_squares(v: Vector) -> float:
    """Returns v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)

def squared_distance(v: Vector, w: Vector) -> float:
    """Computes (v_1 - w_1) ** 2 + ... + (v_n - w_n) ** 2"""
    return sum_of_squares(subtract(v, w))

def distance(v: Vector, w: Vector) -> float:
    """Computes the distance between v and w"""
    return math.sqrt(squared_distance(v, w))

def majority_vote(labels: List[str]) -> str:
    """Assumes that labels are ordered from nearest to farthest."""
    vote_counts = Counter(labels)
    winner, winner_count = vote_counts.most_common(1)[0]

    num_winners = len([count
                       for count in vote_counts.values()
                       if count == winner_count])
    if num_winners == 1:
        return winner
    else:
        return majority_vote(labels[:-1])

def knn_classify(k: int,
                 labeled_points: List[LabeledPoint],
                 new_point: Vector) -> str:
    by_distance = sorted(labeled_points,
                         key=lambda lp: distance(lp.point, new_point))
    k_nearest_labels = [lp.label for lp in by_distance[:k]]

    return majority_vote(k_nearest_labels)
```

รูปที่ 6.3 k-Nearest Neighbors Classifier ในภาษาไพธอน (อ้างอิง [1])

6.2 ตัวอย่างการจำแนกประเภทดอกไอริสด้วย k-NN Classifier

ขั้นตอนแรก ดาวน์โหลด Iris dataset จาก UCI Machine Learning Repository [2] มาเก็บไว้ในไฟล์ชื่อ iris.data

```
import requests
data = requests.get(
  "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
)

with open('iris.data', 'w') as f:
    f.write(data.text)
```

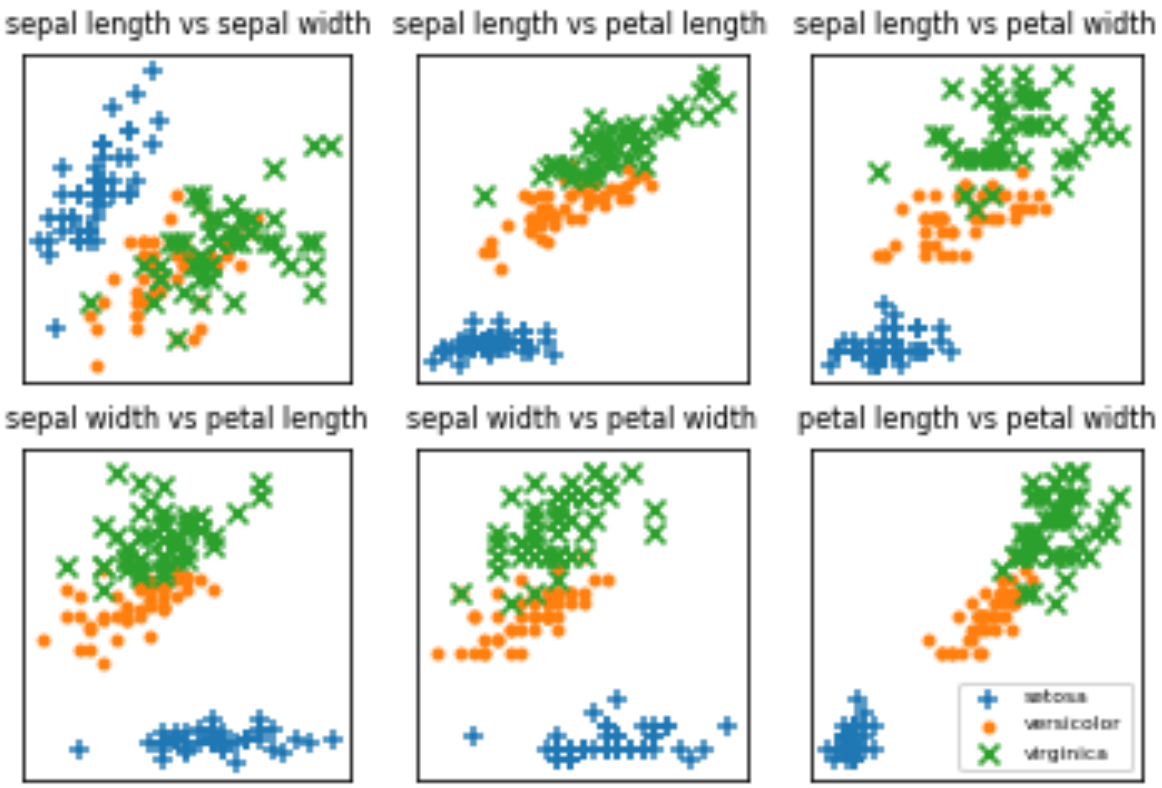
จากนั้นทำการแปลงข้อมูลดิบไปเป็นคลาส LabelPoint และสร้างลิสต์ points_by_species สำหรับแมปจาก คลาสลาเบล (species) ไปเป็นจุดข้อมูลทุกจุดที่อยู่ในคลาสนั้น

```
from typing import Dict
import csv
from collections import defaultdict

def parse_iris_row(row: List[str]) -> LabeledPoint:
    """
    sepal_length, sepal_width, petal_length, petal_width, class
    """
    measurements = [float(value) for value in row[:-1]]
    label = row[-1].split("-")[-1]

    return LabeledPoint(measurements, label)

with open('iris.data') as f:
    reader = csv.reader(f)
    iris_data = [parse_iris_row(row) for row in reader if len(row) > 0]

points_by_species: Dict[str, List[Vector]] = defaultdict(list)
for iris in iris_data:
    points_by_species[iris.label].append(iris.point)
```

ก่อนที่จะสร้างโมเดล k-NN เราจะทำการสำรวจข้อมูลเบื้องต้นก่อนโดยการสร้างกราฟ scatterplot เพื่อแสดงแนวโน้มของความสัมพันธ์ระหว่างฟีเจอร์แต่ละคู่ของจุดข้อมูล โดยเราจะใช้ไลบารี่ matplotlib ในการสร้างกราฟนี้

```
from matplotlib import pyplot as plt
%matplotlib inline

metrics = ['sepal length', 'sepal width', 'petal length', 'petal width']
pairs = [(i, j) for i in range(4) for j in range(4) if i < j]
marks = ['+', '.', 'x']

fig, ax = plt.subplots(2, 3)

for row in range(2):
    for col in range(3):
        i, j = pairs[3 * row + col]
        ax[row][col].set_title(f"{metrics[i]} vs {metrics[j]}", fontsize=8)
        ax[row][col].set_xticks([])
        ax[row][col].set_yticks([])

        for mark, (species, points) in zip(marks, points_by_species.items()):
            xs = [point[i] for point in points]
            ys = [point[j] for point in points]
            ax[row][col].scatter(xs, ys, marker=mark, label=species)

ax[-1][-1].legend(loc='lower right', prop={'size': 6})
plt.show()
```

ซึ่งจะพบว่าฟีเจอร์ petal length และ petal width สามารถแบ่งแยกจุดข้อมูลออกเป็นสปีชีส์สามสปีชีส์ได้ค่อนข้างชัดเจน



ต่อไปเราจะแบ่งชุดข้อมูล Iris dataset ออกเป็น training dataset และ test dataset โดยการสุ่มเลือกจุดข้อมูล 70% จากชุดข้อมูล ไปไว้ใน training dataset และที่เหลืออีก 30% ไปไว้ใน test dataset

```
import random
from typing import TypeVar, List, Tuple
X = TypeVar('X')  # generic type to represent a data point

def split_data(data: List[X], prob: float) -> Tuple[List[X], List[X]]:
    """Split data into fractions [prob, 1 - prob]"""
    data = data[:]                      # Make a shallow copy
    random.shuffle(data)                # because shuffle modifies the list.
    cut = int(len(data) * prob)         # Use prob to find a cutoff
    return data[:cut], data[cut:]       # and split the shuffled list there.


random.seed(12)
iris_train, iris_test = split_data(iris_data, 0.70)
```

จากนั้นสร้างโมเดล k-NN classifier ด้วยชุดข้อมูลฝึกฝน และทำการวัดประสิทธิภาพบนชุดข้อมูลทดสอบ

```
random.seed(12)
iris_train, iris_test = split_data(iris_data, 0.70)

confusion_matrix: Dict[Tuple[str, str], int] = defaultdict(int)
num_correct = 0
for iris in iris_test:
    predicted = knn_classify(5, iris_train, iris.point)
    actual = iris.label

    if predicted == actual:
        num_correct += 1

    confusion_matrix[(predicted, actual)] += 1

pct_correct = num_correct / len(iris_test)

print(f"Accuracy: {pct_correct:.4f}\n")

print("Confusion Matrix")
print("="*50)
for cls, cnt in confusion_matrix.items():
    print(f"Actual[{cls[0]:10s}] - Predicted:[{cls[1]:10s}] || {cnt}")
print("="*50)
```

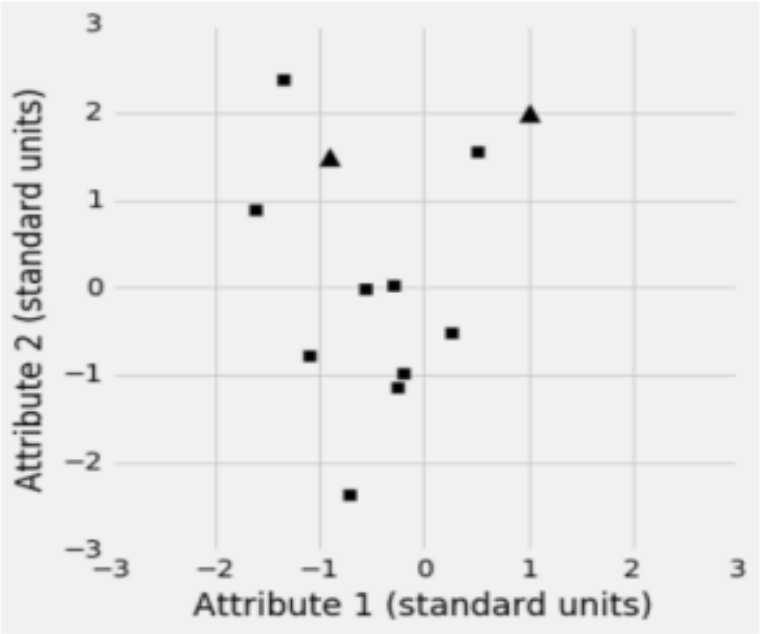
ผลจากการวัดประสิทธิภาพ พบว่า k-nearest neighbors สามารถจำแนกประเภทดอกไอริสได้ที่ความถูกต้อง (accuracy) ประมาณ 97% โดยมีความผิดพลาด 1 ครั้ง คือ ทำนายดอกไอริส virginica เป็น พันธ์ versicolor ดังแสดงใน confusion matrix ด้านล่าง

```
Accuracy: 0.9778

Confusion Matrix
==================================================
Actual[setosa    ] - Predicted:[setosa    ] || 13
Actual[versicolor] - Predicted:[versicolor] || 15
Actual[virginica ] - Predicted:[virginica ] || 16
Actual[virginica ] - Predicted:[versicolor] || 1
==================================================
```

แบบฝึกหัด

<u>จงใช้ข้อมูลต่อไปนี้ตอบคำถามข้อ 1 ถึงข้อ 4</u>

กำหนดชุดข้อมูลฝึกฝน (training dataset) ซึ่งประกอบด้วยจุดข้อมูลทั้งหมด 12 จุดข้อมูล โดยจุดข้อมูลแต่ละจุดถูกจำแนกให้เป็นสมาชิกของคลาสใดคลาสหนึ่งในสองคลาสคือ คลาส **Triangle** (แทนด้วยรูป ▲) และ คลาส Square (แทนด้วยรูป ■) และแต่ละจุดข้อมูลประกอบด้วยแอทริบิวต์สองตัวคือ Attribute1 และ Atrribute2 เมื่อนำจุดข้อมูลทั้ง 12 จุดพล็อตลงบน scatterplot (แกน x คือ Attribute1 และ แกน y คือ Attribute2) จะได้ผลดังรูป



1. จุดข้อมูลจุดใดในข้อต่อไปนี้ จะถูกจำแนกให้อยู่ในคลาส **Triangle** โดย **3-nearest neighbor classifier** บนชุดข้อมูลในรูปที่ 1
   ก. (-0.5, -0.5)
   ข. (-2, 0)
   ค. (0, 2)
   ง. (0, -1)

2. หากเราใช้ **2-nearest neighbor classifier** แบบ majority vote กับชุดข้อมูลในรูปที่ 1 จุดข้อมูลจุดใหม่ ที่มีค่า Attribute1 = -1.2 หน่วย และ Attribute2 = 1.2 หน่วย จะถูกจำแนกให้อยู่ในคลาสใด
   ก. คลาส Triangle (▲) เนื่องจาก จุด (-0.9, 1.4) อยู่ใกล้กับจุดใกล้กับจุดข้อมูลใหม่มากกว่า
   ข. คลาส Square (■) เนื่องจาก จุด (-1.6, 0.9) อยู่ใกล้กับจุดข้อมูลใหม่มากกว่า
   ค. คลาส Square (■) เนื่องจาก จุดที่อยู่ใกล้ที่สุดสองจุดเป็นคลาด Triangle และ คลาส Square อย่างละจุดซึ่งเสมอกันจึงจำเป็นต้องตัดสินโดยใช้จุดที่อยู่ใกล้ที่สุดเป็นลำดับที่สามคือจุด (-1.2, 2.4) ซึ่งเป็นคลาส Square
   ง. อาจเป็นคลาส Triangle (▲) หรือ คลาส Square (■) ก็ได้

3. หากเราใช้ **3-nearest neighbor classifier** แบบ majority vote กับชุดข้อมูลในรูปที่ 1 จุดข้อมูลจุดใหม่ ที่มีค่า Attribute1 = -1.2 หน่วย และ Attribute2 = 1.2 หน่วย จะถูกจำแนกให้อยู่ในคลาสใด
   ก. คลาส Triangle (▲)
   ข. คลาส Square (■)
   ค. ไม่สามารถใช้ 3-nearest neighbor กับชุดข้อมูลในรูปที่ 1 ได้
   ง. อาจเป็นคลาส Triangle (▲) หรือ คลาส Square (■) ก็ได้

4. จากรูปที่ 1 สมมุติว่า จุดข้อมูลจุดใหม่มีค่าของ Attribute1 และ Attribute2 **ต่ำกว่าค่าเฉลี่ยของ Attribute1 และ Attribute2 ตามลำดับทั้งคู่** คำทำนายคลาสของจุดข้อมูลใหม่จุดนี้โดย 3-nearest-neighbor classifier จะมีคลาสเป็นคลาสใด
   ก. คลาส Square (■)
   ข. คลาส Triangle (▲)
   ค. อาจเป็นคลาส Triangle (▲) หรือ คลาส Square (■) ก็ได้
   ง. ไม่สามารถหาคำตอบได้ เนื่องจากข้อมูลไม่เพียงพอ

5. จงใช้ k-Nearest Neighbors classifier เพื่อทำนายค่า miles-per-gallon ของรถยนต์ โดยใช้ชุดข้อมูล Auto MPG dataset จาก UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/auto+mpg

เอกสารอ้างอิง

[1] Joel Grus. Data Science from Scratch (2ed), O'Reilly Media, Inc., 2019.
[2] Iris Data Set. Available at https://archive.ics.uci.edu/ml/datasets/iris