분류모델의 성능 평가

충북대학교 소프트웨어학과
류관희

목 차

01

분류모델 실습

- K-NN 모델 이해
- k-NN 모델 실습
- K-NN 모델 파이션 구현

02

분류모델 성능평가지표

- 분류 모델의 작동 원리
- 분류 정확도 평가 지표
- Confusion Matrix

03

분류모델 성능향상

- 데이터 분석 과정
- 파라메터 조정
- 훈련데이터와 시험 데이터 비율 조정
- 데이터 Scaling 적용

학습목표

충북대학교
SW중심대학사업단

이번 파트에서는 분류 모델의 성능 평가를 이해하기 위해 k-NN 모델 적용 데이터 분류 분석을 공부한다.

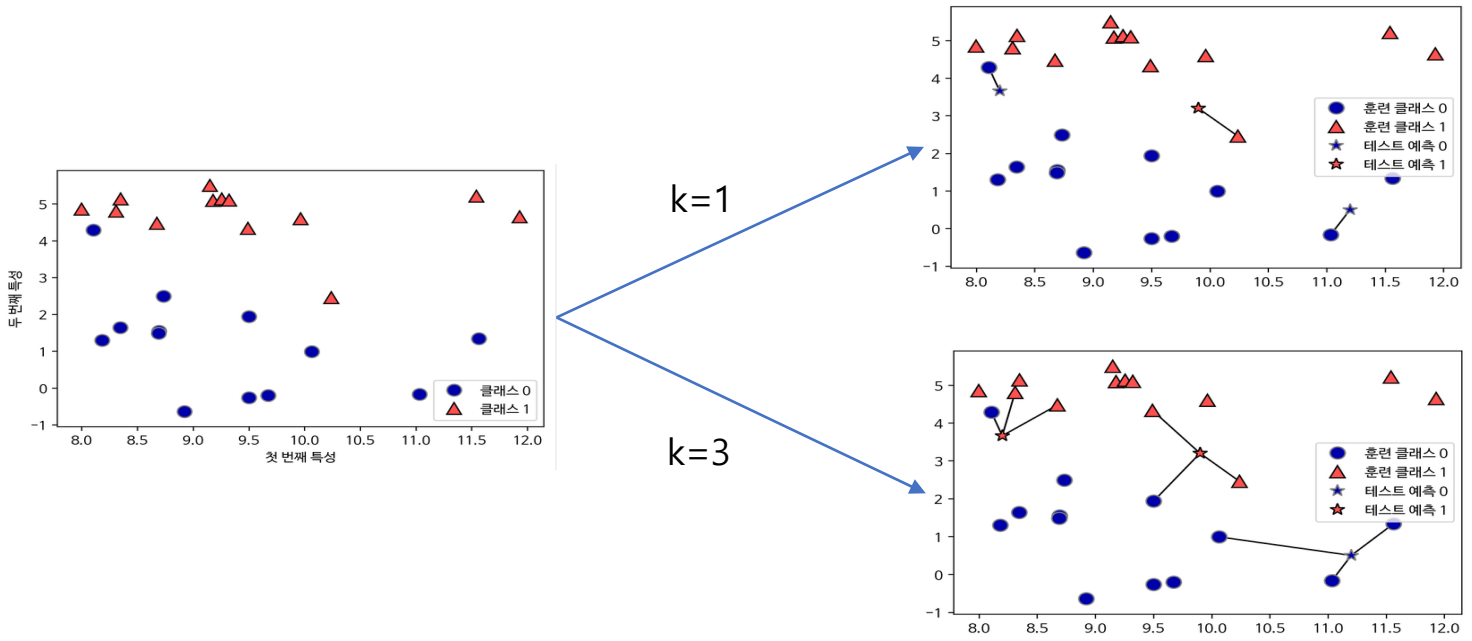
- K-NN 모델 이해
- K-NN 모델 실습
- K-NN 모델의 파이션 구현

분류 모델 실습

- KNN 알고리즘을 이해하셨나요?
  - KNN 의미는 무엇인가?
  - KNN 알고리즘이란?
  - KNN 알고리즘은 어디에서 사용할 수 있는가?
  - KNN과 linear regression 차이점은 무엇인가?
  - KNN 알고리즘은 어떻게 작동하는가?
  - KNN 알고리즘의 장단점은 무엇인가?

분류 모델 실습

- KNN 알고리즘이란?
  - K Nearest Neighbor (KNN) 기계 학습 분야에서 가장 간단한 알고리즘
  - 분류하고자 하는 데이터와 가장 가까운 k개의 이웃을 선택하여 분류함



분류 모델 실습

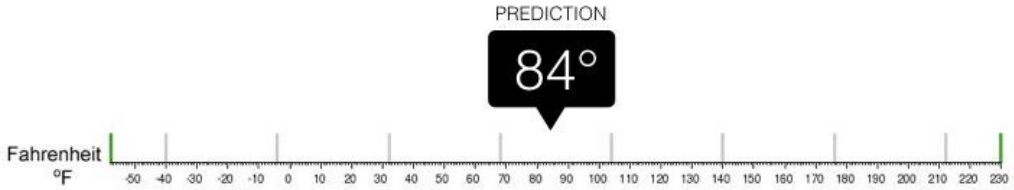
- KNN 적용 분야
  - 얼굴 및 필기체 인식
  - 질병 탐지
  - 스팸 메일 분류
  - 금융 애플리케이션
  - 엔터테인먼트 애플리케이션

분류 모델 실습

- KNN과 linear regression 차이점

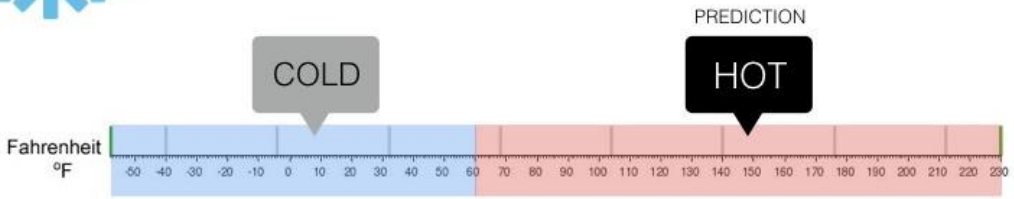
Regression

What is the temperature going to be tomorrow?



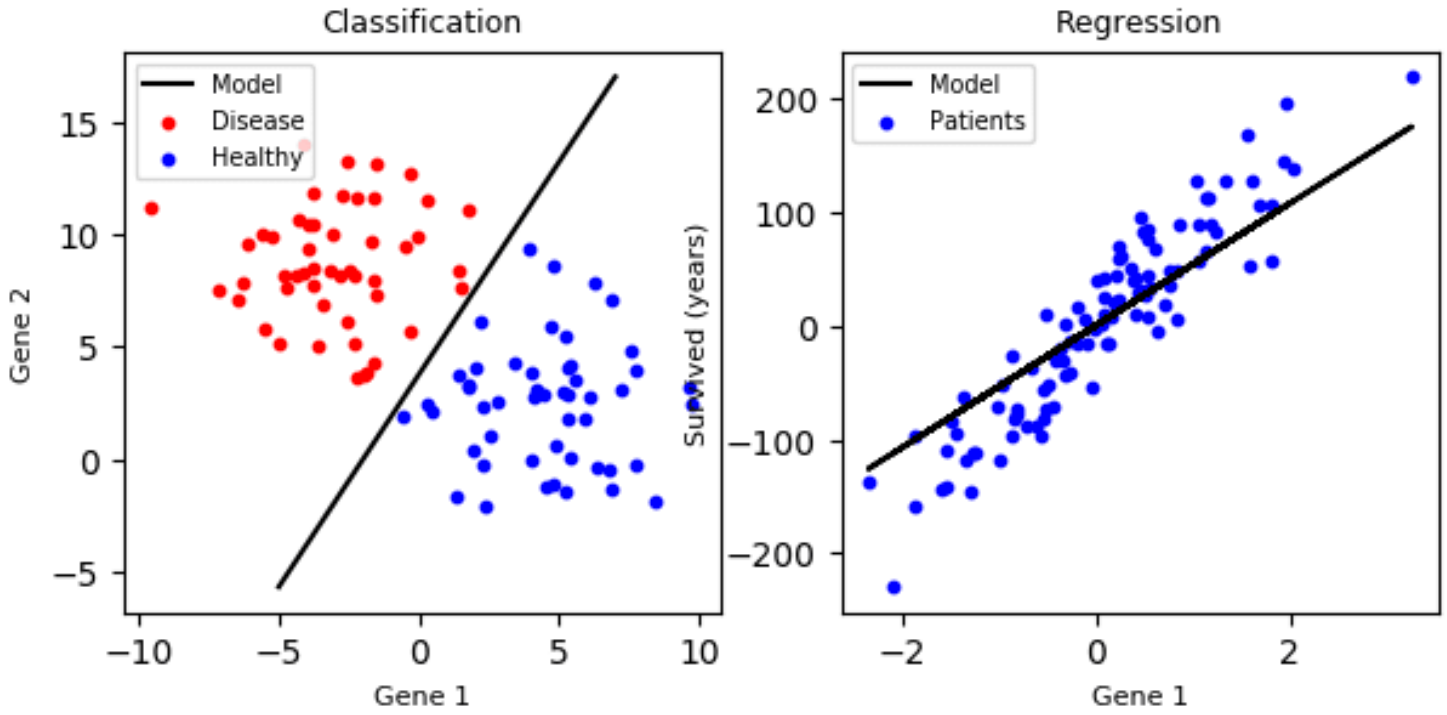
Classification

Will it be Cold or Hot tomorrow?



분류 모델 실습

- KNN과 linear regression 차이점



분류 모델 실습

KNN 알고리즘은 어떻게 작동하는가?

단계 1
매개 변수 k (가장 가까운 데이터 포인트) 결정

단계 2
유클리드 거리를 $\mathrm{dist}(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$를 계산

단계 3
2단계에서 계산된 거리 값 정렬

단계 4
정렬된 데이터에서 상위-k 선택

단계 5
이러한 행의 가장 빈번한 클래스를 기반으로 테스트 포인트에 클래스 할당

분류 모델 실습

- KNN 알고리즘의 장단점
  - 장점
    - 이해하기 쉬운 모델
    - 많은 조정을 하지 않아도 좋은 성능 발휘
    - 매우 빠르게 모델을 만들 수 있음
    - 복잡한 방법을 적용하기 전에 좋은 시작점
  - 단점
    - 적절한 k 선택이 필요
    - 훈련 데이터가 매우 크면(특성의 수, 샘플의 수가 클 경우) 예측이 느리고 잘 작동 하지 않음
    - Nominal 속성과 누락 데이터는 추가 처리가 필요

분류 모델 실습

- Python에서의 KNN 알고리즘

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors = 1)

training_points = [
  [0.5, 0.2, 0.1],
  [0.9, 0.7, 0.3],
  [0.4, 0.5, 0.7]
]
training_labels = [0, 1, 1]

classifier.fit(training_points, training_labels)

unknown_points = [
  [0.2, 0.1, 0.7]
]
guesses = classifier.predict(unknown_points)

print(guesses)
```

분류 모델 실습

- 실습
  - 키와 몸무게에 대한 scatter plot 그리기
  - 키가 161, 몸무게가 61일 때의 'T Shirt Size' 예측

```
unknown_points = [
  [161, 61]
]
```

  - k=3, k=5, k= 10 일 때의 결과 확인

training_points

training_labels

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

분류 모델 실습

- KNN 알고리즘 실습

```
training_points = [
    [158, 58],
    [158, 59],
    [158, 63],
    [160, 59],
    [160, 60],
    [163, 60],
    [163, 61],
    [160, 64],
    [163, 64],
    [165, 61],
    [165, 62],
    [165, 65],
    [168, 62],
    [168, 63],
    [168, 66],
    [170, 63],
    [170, 64],
    [170, 68]
]

training_labels = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

분류 모델 실습

- KNN 알고리즘 실습

```
#We want to find the class of the following point
unknown_points = [
    [161, 61],
]

#Learning with KNN algorithm when K=3
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(training_points, training_labels)

guesses = classifier.predict(unknown_points)

print(guesses)
```

문제풀이

- K-NN을 이용한 데이터 분석 과정을 설명하시오.

- 파이션을 이용한 k-NN 모델 구현 과정을 설명하시오.

요약

- 데이터 분류 분석 성능 측면에서 다음과 같은 과정을 공부하였음.
  - K-NN 모델 이해
  - K-NN 모델 실습
  - 파이션을 이용한 k-NN 모델의 구현

01

분류모델 실습

- K-NN 모델 이해
- k-NN 모델 실습
- K-NN 모델 파이션 구현

02

분류모델 성능평가지표

- 분류 모델의 작동 원리
- 분류 정확도 평가 지표
- Confusion Matrix

03

분류모델 성능향상

- 데이터 분석 과정
- 파라메터 조정
- 훈련데이터와 시험 데이터 비율 조정
- 데이터 Scaling 적용

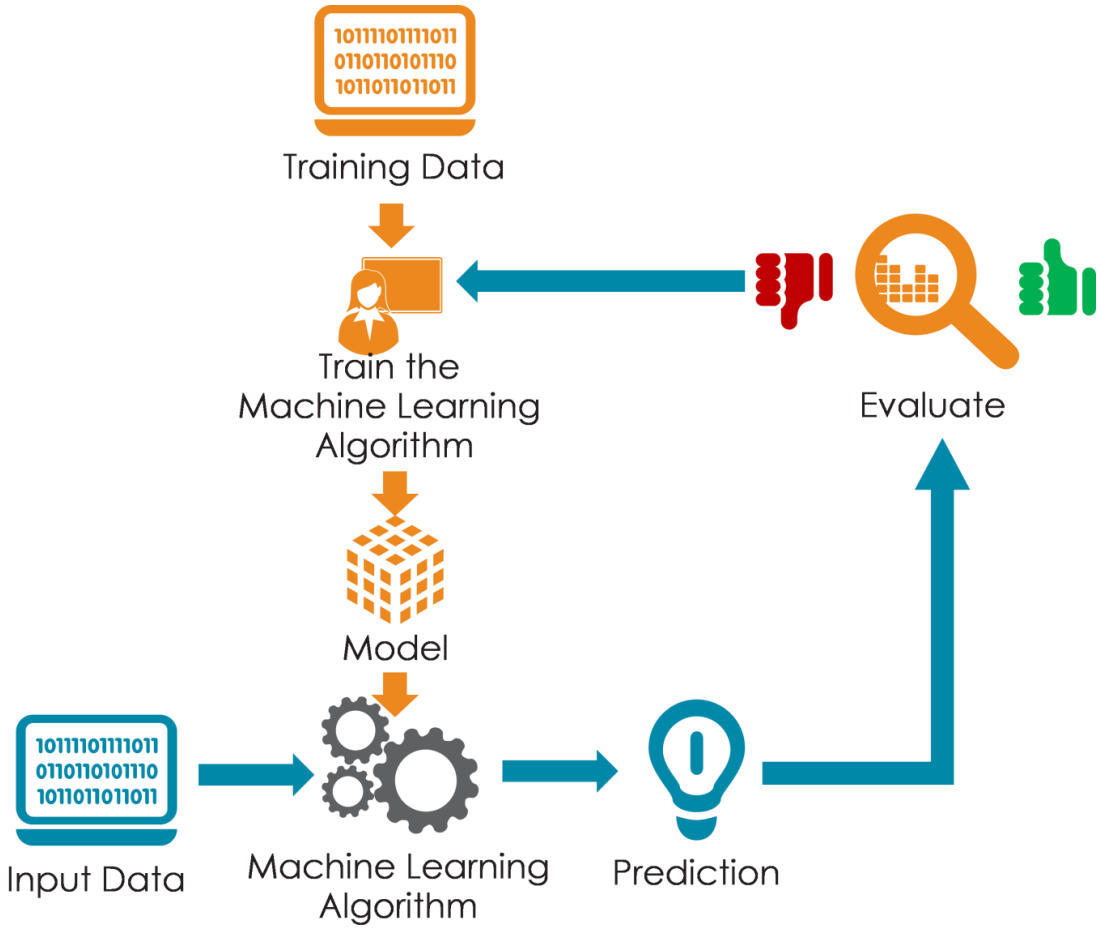
학습목표

이번 파트에서는 분류 모델의 성능 평가를 이해하기 위해 다음과 같은 내용을 공부한다.

- 분류 모델의 작동 원리
- 분류 정확도 평가 지표
- Confusion Matrix

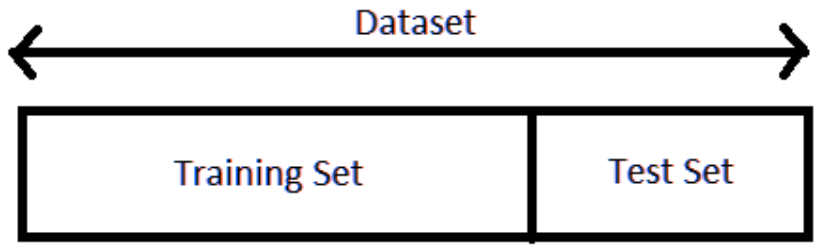
분류 모델 성능 평가 지표

- 빅데이터 분류분석의 작동 방식



분류 모델 성능 평가 지표

- Train과 test로 분할
  - Train dataset
    - 모델을 훈련하는데 사용하는 실제 데이터셋
      - 모델이 이 데이터를 보고 학습함
  - Test dataset
    - 모델 평가에 사용되는 데이터셋



- 보통 training과 testing 단계에서 데이터를 20%-80% 정도 분할

분류 모델 성능 평가 지표

- Train과 test로 분할
  - sklearn 라이브러리

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
Train_test_split(training_points, training_labels, test_size=0.2, random
_state=4)
```

  - test_size=0.2
    - Test 데이터셋은 20%, training 데이터셋은 80%
  - random_state=4
    - random_state 하이퍼 파라미터를 사용하지 않는 한, 데이터는 무작위로 할당

분류 모델 성능 평가 지표

- Train과 test로 분할 실습

```
training_points = [
    [158, 58],
    [158, 59],
    [158, 63],
    [160, 59],
    [160, 60],
    [163, 60],
    [163, 61],
    [160, 64],
    [163, 64],
    [165, 61],
    [165, 62],
    [165, 65],
    [168, 62],
    [168, 63],
    [168, 66],
    [170, 63],
    [170, 64],
    [170, 68]
]

training_labels = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

분류 모델 성능 평가 지표

- Train과 test로 분할 실습

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(training_points, training_labels,
test_size=0.2, random_state=4)

#Learning with KNN algorithm when K=3
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors = 1)
classifier.fit(X_train, y_train)

guesses = classifier.predict(X_test)

print(guesses)
```

분류 모델 성능 평가 지표

- 분류 정확도
  - Confusion matrix
  - Accuracy
  - Error rate
  - Precision
  - Recall
  - F measure

분류 모델 성능 평가 지표

- 분류 정확도
  - Sklearn 라이브러리

```
from sklearn.metrics import confusion_matrix
from sklearn import metrics

print(confusion_matrix(y_test, guesses))

print(metrics.accuracy_score(y_test, guesses))

print(metrics.precision_score(y_test, guesses, average='binary'))

print(metrics.recall_score(y_test, guesses, average='binary'))

print(metrics.f1_score(y_test, guesses, average='binary'))
```

분류 모델 성능 평가 지표

- Confusion matrix

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

분류 모델 성능 평가 지표

- Confusion matrix
  - 가장 일반적인 성능 측정은 한 클래스와 다른 클래스를 구별하는 모델의 능력을 고려
    - 관심 클래스는 positive
    - 다른 모든 것은 negative

  - Positive 클래스와 negative 클래스 예측 간의 관계는 2x2 confusion matrix로 표시 될 수 있음
    - 예측이 네 가지 범주 중 하나에 속하는지 여부를 표로 만들기
      - True Positive (TP): 관심 클래스로 올바르게 분류
      - True Negative (TN): 관심 클래스가 아닌 것으로 올바르게 분류
      - False Positive (FP): 관심 클래스로 잘못 분류
      - False Negative (FN): 관심 클래스가 아닌 것으로 잘못 분류

분류 모델 성능 평가 지표

- Confusion matrix 실습

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

training_points = cancer.data
training_labels = cancer.target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(training_points, training
_labels, test_size=0.2, random_state=4)

#Learning with KNN algorithm when K=3
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)

guesses = classifier.predict(X_test)
```

분류 모델 성능 평가 지표

- Confusion matrix 실습

```
#Check confusion matrix with sklrean metrics
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, guesses))
```

```
[[29  5]
 [ 9 71]]
```

분류 모델 성능 평가 지표

- Accuracy
  - 2 x 2 confusion matrix를 사용하면 예측 정확도 (success rate이라고도 함)를 다음과 같이 정의 할 수 있음:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- Error rate
  - 오류율 또는 잘못 분류된 예의 비율은 다음과 같이 지정:

$$\text{error rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = 1 - \text{accuracy}$$

분류 모델 성능 평가 지표

- Precision
  - Precision은 진정으로 긍정인 positive 사례의 비율로 정의
  - 다시 말해, 모델이 positive 클래스를 예측할 때 얼마나 자주 올바른지

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall
  - 반면, recall은 결과가 얼마나 완전한지 측정

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- F1 Measure

  F1=2(precision x recall)/(precision+recall)

- G Measure

  G=sqrt(precision x recall)

분류 모델 성능 평가 지표

- Task 1
  - 암 데이터셋의 정확도와 오류율 계산
  - Accuracy
    - (29 + 71) / (29 + 71 + 9 + 5)
    - 결과: 0.877
  - Error rate
    - (9 + 5) / (29 + 71 + 9 + 5)
    - 결과: 0.122

분류 모델 성능 평가 지표

- Task 2
  - 암 데이터셋의 정확도와 오류율 계산

  - Precision
    - 71/ (71 + 6)
    - 결과: 0.934

  - Recall
    - 71/ (71 + 9)
    - 결과: 0.887

분류 모델 성능 평가 지표

- Task 3
  - F-measure
    - (2 * 0.934 * 0.887)/ (0.934 + 0.887) = 1.656/1.821
    - 결과: 0.91

분류 모델 성능 평가 지표

- Accuracy, precision, recall and F measure 실습

```
#Check classification accuracy with sklrean metrics
from sklearn import metrics
print(metrics.accuracy_score(y_test, guesses))

#Check precision with sklrean metrics
from sklearn import metrics
print(metrics.precision_score(y_test, guesses, average='binary'))

#Check recall with sklrean metrics
from sklearn import metrics
print(metrics.recall_score(y_test, guesses, average='binary'))

#Check F measure with sklrean metrics
from sklearn import metrics
print(metrics.f1_score(y_test, guesses, average='binary'))
```

문제풀이

- 분류모델 성능평가를 위한 Confusion Matrix를 설명하시오.
- 분류 모델의 다음과 같은 성능 평가 지표를 설명하시오.
  - Accuracy
  - Error rate
  - Precision
  - Recall
  - F measure
  - G measure

요약

- 분류모델 성능평가를 위한 Confusion Matrix를 공부하였음.
- 분류 모델의 다음과 같은 성능 평가 지표를 공부하였음.
  - Accuracy
  - Error rate
  - Precision
  - Recall
  - F measure
  - G measure

01

분류모델 실습

- K-NN 모델 이해
- k-NN 모델 실습
- K-NN 모델 파이션 구현

02

분류모델 성능평가지표

- 분류 모델의 작동 원리
- 분류 정확도 평가 지표
- Confusion Matrix

03

분류모델 성능향상

- 데이터 분석 과정
- 파라메터 조정
- 훈련데이터와 시험 데이터 비율 조정
- 데이터 Scaling 적용

학습목표

충북대학교
SW중심대학사업단

이번 파트에서는 분류 모델의 성능 향상을 이해하기 위해 다음과 같은 내용을 공부한다.

- 데이터 분석 과정
- 파라메터 조정
- 훈련데이터와 시험 데이터 비율 조정
- 데이터 Scaling 적용

분류 모델의 성능 향상

- 빅데이터 설계 프로세스는 두 가지 주요 단계로 구성
  - 데이터 관리, 데이터 훈련 및 지속적인 정확도 개선
- 빅데이터 설계를 위한 단계
  1. 라이브러리 로딩
  2. 데이터셋 로딩
  3. 데이터 관찰
  4. 통계 분석 (불필요한 열 제거하기, 중복 데이터 찾기 및 제거하기, Null 값 제거 등)
  5. Training 및 testing 데이터셋 분할
  6. 훈련 모델과 정확도 확인
  7. 하이퍼 파라미터 (k의 수)를 조정하여 정확도 향상
  8. Training 및 testing 데이터셋 비율 변경
  9. 데이터 정규화
  10. 이상 값 처리

분류 모델의 성능 향상

- Step 1
  - 이 자습서에서 분석을 수행하는 데 사용할 여러 라이브러리 로딩
    - 이미 라이브러리를 설치했다고 가정

```
import pandas as pd
import numpy as np


import matplotlib.pyplot as plt


from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
```

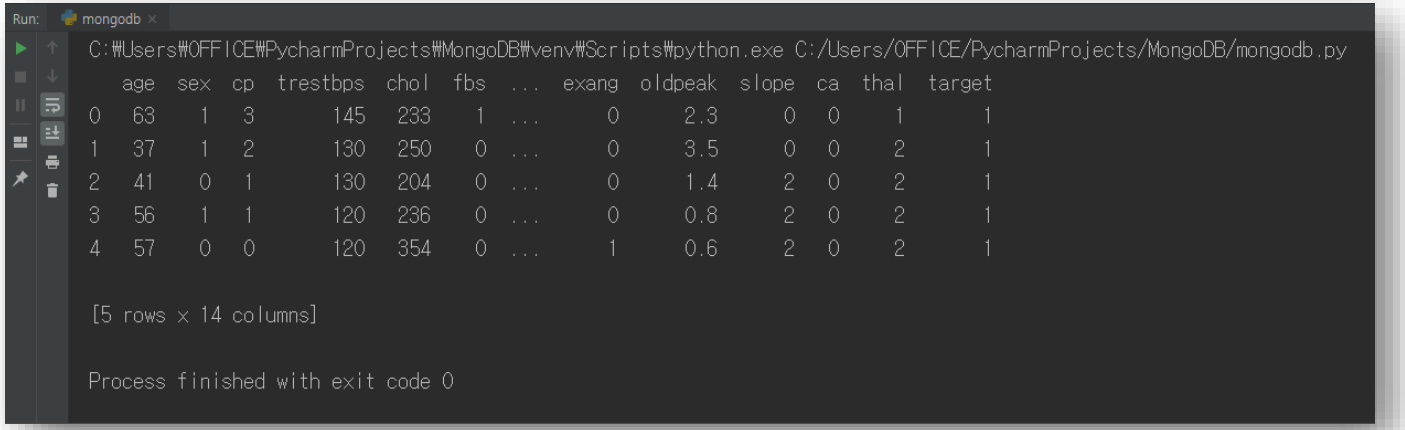
분류 모델의 성능 향상

- Step 2
  - 사용할 데이터셋을 로드, 데이터셋은 심장병 검사를 받은 환자의 과거 데이터를 포함
    - df = pd.read_csv('heart.csv')
- Step 3
  - 데이터에 더 익숙해지도록 데이터의 일반적인 정보를 살펴보기

```
#print(df.head())
#print(df.shape)
#print(df.info())
```

분류 모델의 성능 향상

- Step 3
  - print(df.head())의 결과
    - 데이터셋의 상위 5 개 레코드를 표시

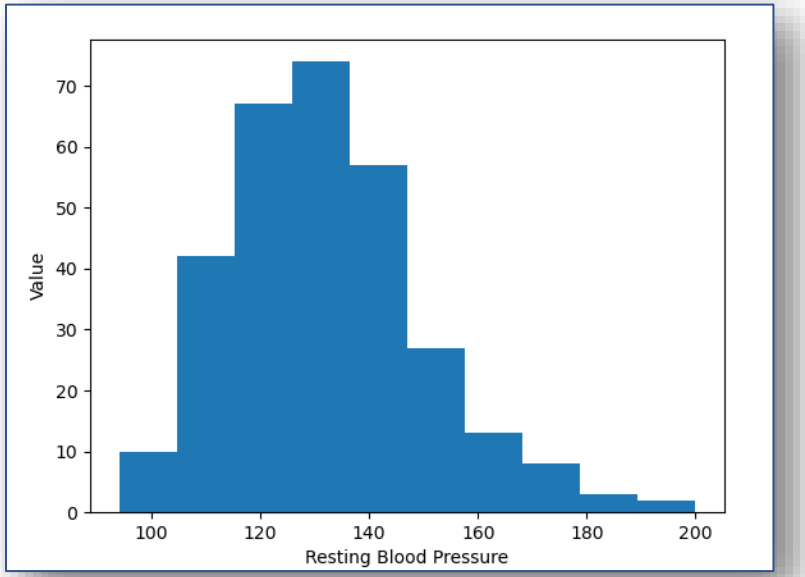


  - Task 1
    - print(df.shape)와 print(df.info())를 직접 확인

분류 모델의 성능 향상

- Step 4
  - 데이터를 더 잘 이해하기 위해 탐색적 데이터 분석(EDA)를 수행
    - trestbps 속성의 히스토그램

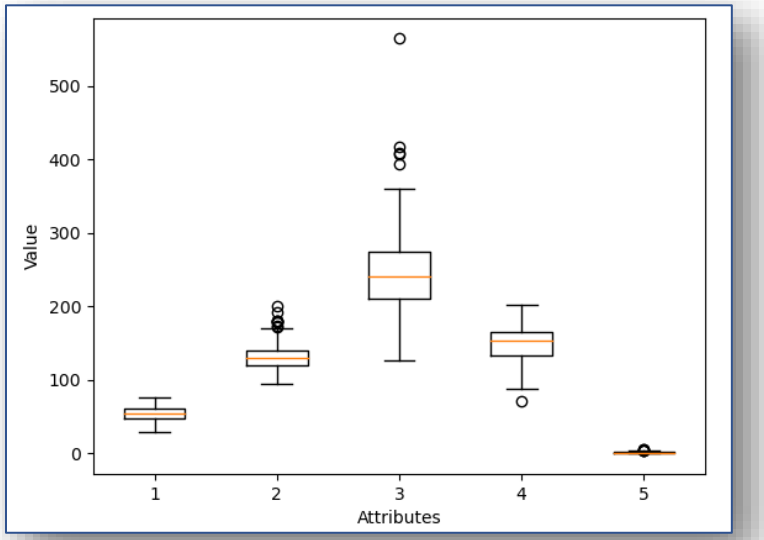
```
plt.hist(df['trestbps'])
plt.xlabel('Resting Blood Pressure')
plt.ylabel('Value')
plt.show()
```



분류 모델의 성능 향상

- Step 4
  - 모든 속성의 이상 값

```
data_to_boxplot = [df['age'], df['trestbps'], df['chol'],
                                              df['thalach'], df['oldpeak']]
plt.boxplot(data_to_boxplot)
plt.xlabel('Attributes')
plt.ylabel('Value')
plt.show()
```



분류 모델의 성능 향상

- Step 4
  - 누락된 값
  - Print(df.isnull().sum())

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64

Process finished with exit code 0
```

분류 모델의 성능 향상

- Step 5
  - 정확도를 확인하기 위해 training 및 test 데이터셋으로 분할
  - Training data ->70%, test data -> 30%

```
training_points = df.drop(columns=['target'])
training_labels = df['target']

X_train, X_test, y_train, y_test = train_test_split(
            training_points,
            training_labels,
            test_size=0.3,
            random_state=4)
```

분류 모델의 성능 향상

- Step 6
  - 모델 훈련(k=5)과 정확도 확인

```
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
guesses = classifier.predict(X_test)


print(guesses)
print(confusion_matrix(y_test, guesses))
print(metrics.accuracy_score(y_test, guesses))
```

초기 분류 정확도는 0.5934065934065934

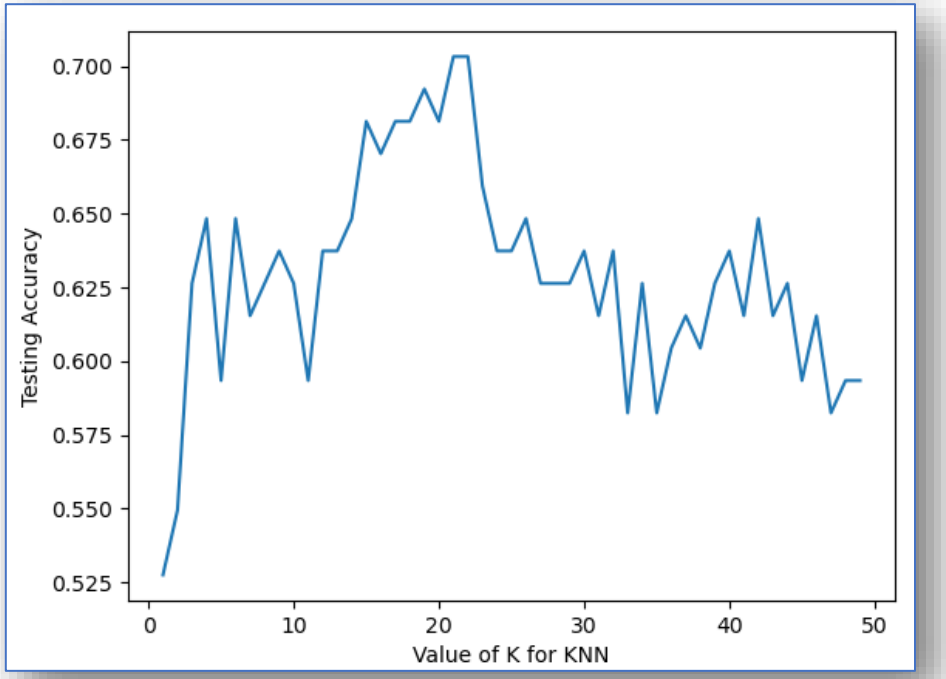
분류 모델의 성능 향상

- Step 7
  - 하이퍼 파라미터(k의 수)를 조정하여 정확도 개선

```
k_range = range(1, 50)

accuracy_scores = []

for k in k_range:
    classifier = KNeighborsClassifier(n_neighbors = k)
    classifier.fit(X_train, y_train)
    guesses = classifier.predict(X_test)
    accuracy_scores.append(metrics.accuracy_score(y_test, gues
ses))
print(accuracy_scores)

#Visualize the result of KNN accuracy with matplotlib
plt.plot(k_range, accuracy_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
plt.show()
```

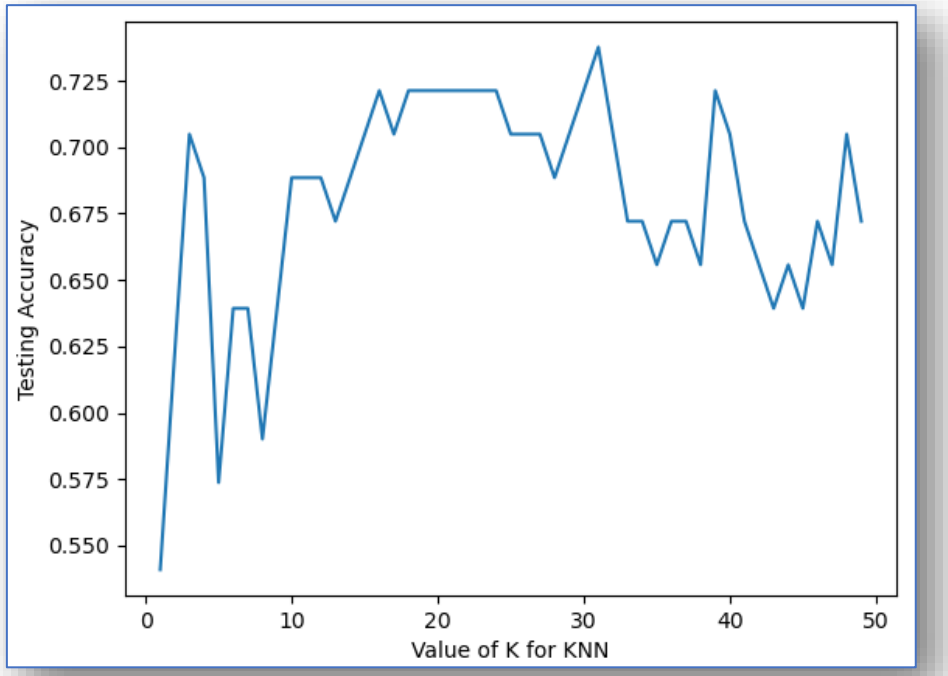
분류 모델의 성능 향상

- Step 7
  - 하이퍼 파라미터 조정 결과
  - 가장 높은 정확도 : 0.7032967032967034



분류 모델의 성능 향상

- Step 8
  - Training 및 test 데이터셋 비율 변경
    - Training data -> 80%, test data ->20%
  - 가장 높은 정확도: 0.7377049180327869



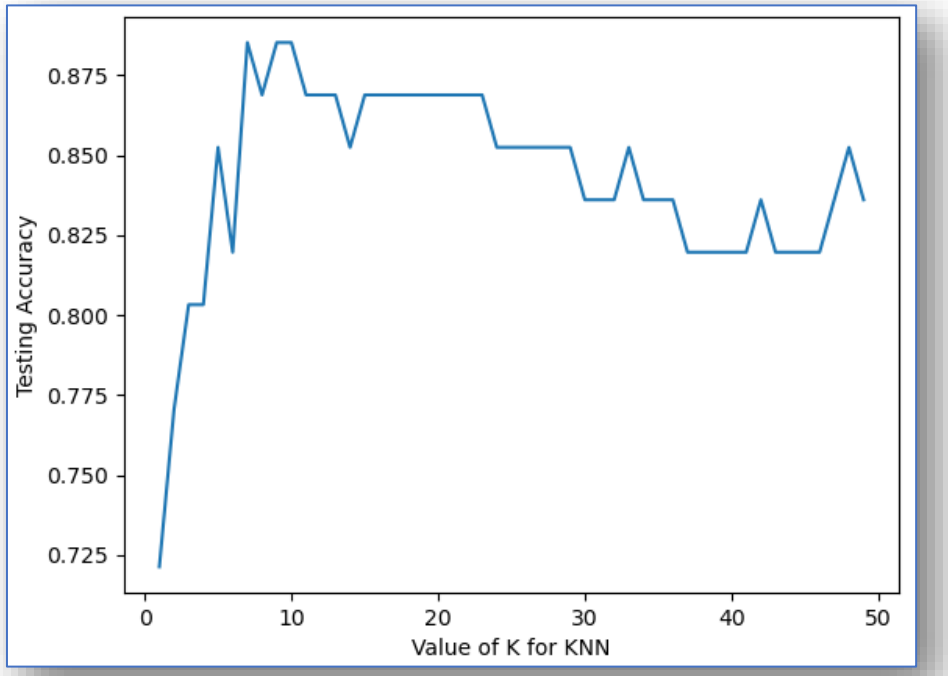
분류 모델의 성능 향상

- Step 9
  - Standard Scaling

```
from sklearn.preprocessing import StandardScaler

#Create copy of dataset.
df_model = df.copy()

#Rescaling features age, trestbps, chol, thalach, oldpeak.
scaler = StandardScaler()

features = [['age', 'trestbps', 'chol', 'thalach', 'oldpeak']]
for feature in features:
    df_model[feature] = scaler.fit_transform(df_model[feature])

training_points = df_model.drop(columns=['target'])
training_labels = df_model['target']
```

분류 모델의 성능 향상

- Step 9
  - Standard Scaling
  - 가장 높은 정확도 : 0.8852459016393442



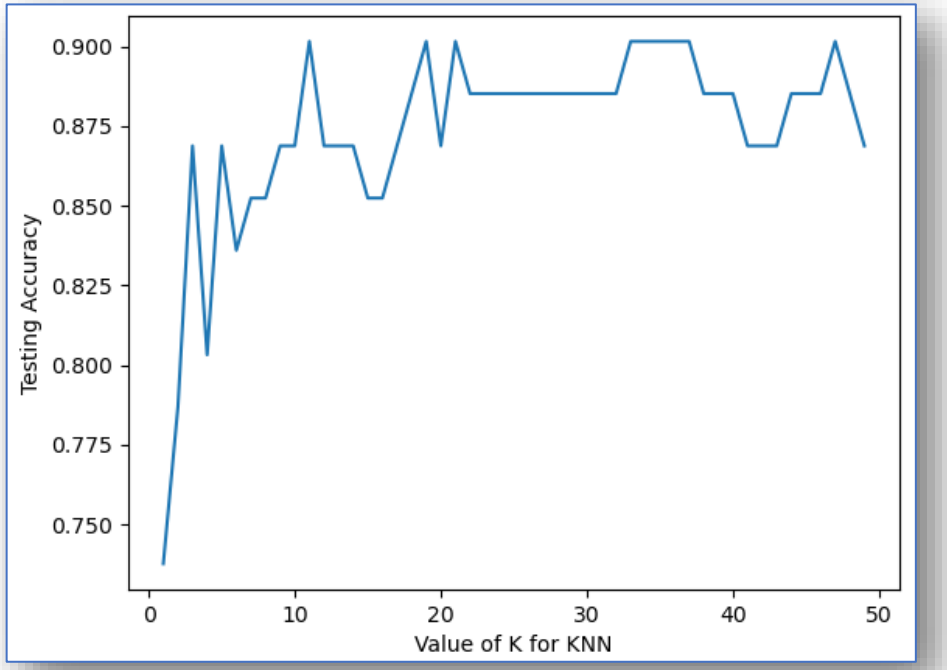
분류 모델의 성능 향상

- Step 9
  - Min-Max Normalization

```
from sklearn.preprocessing import MinMaxScaler

#Create copy of dataset.
df_model = df.copy()

#Rescaling features age, trestbps, chol, thalach, oldpeak.
scaler = MinMaxScaler()

features = [['age', 'trestbps', 'chol', 'thalach', 'oldpeak']]
for feature in features:
    df_model[feature] = scaler.fit_transform(df_model[feature])

training_points = df_model.drop(columns=['target'])
training_labels = df_model['target']
```

분류 모델의 성능 향상

- Step 9
  - Min-Max Scaling
  - 가장 높은 정확도 : 0.9016393442622951



문제풀이

- 분류모델 성능향상을 위해 사용하는 다음과 같은 방법을 설명하시오.
  - 분류모델의 파라메터 조정
  - 학습과 시험 데이터의 비율 조정
  - Scaling을 통한 데이터 변환 조정

요약

- 분류모델 성능향상을 위해 사용하는 다음과 같은 방법을 공부하였음.
  - 분류모델의 파라메터 조정
  - 학습과 시험 데이터의 비율 조정
  - Scaling을 통한 데이터 변환 조정