201645084_3_Q_노영훈

과제 : KNN 분류에서 최적 하이퍼 파라메터 찾기?

1. (KNN 분류)에서 최적의 이웃 숫자 찾기(그래프도 그림)
- 회귀분석을 통해 최적의 이웃 숫자를 찾아낸다.

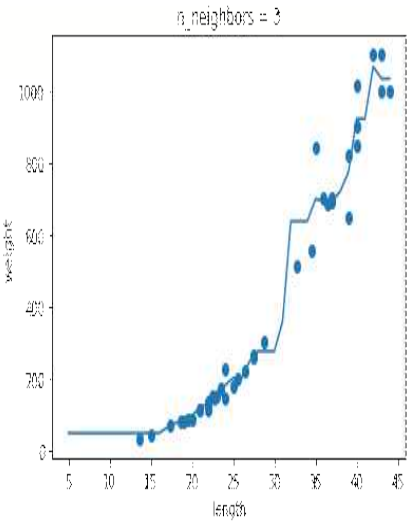
```
# 데이터 준비
perch_length = np.array([8.4 , 13.7 , 15.0 , 16.2 , 17.4 , 18.0 , 18.7 , 19.0 , 19.6 , 20.0
, 21.0 ,
       21.0 , 21.0 , 21.3 , 22.0 , 22.0 , 22.0 , 22.0 , 22.0 , 22.5 , 22.5 , 22.7 ,
       23.0 , 23.5 , 24.0 , 24.0 , 24.6 , 25.0 , 25.6 , 26.5 , 27.3 , 27.5 , 27.5 ,
       27.5 , 28.0 , 28.7 , 30.0 , 32.8 , 34.5 , 35.0 , 36.5 , 36.0 , 37.0 , 37.0 ,
       39.0 , 39.0 , 39.0 , 40.0 , 40.0 , 40.0 , 40.0 , 42.0 , 43.0 , 43.0 , 43.5 ,
       44.0 ])
perch_weight = np.array([5.9 , 32.0 , 40.0 , 51.5 , 70.0 , 100.0 , 78.0 , 80.0 , 85.0 , 85.0
, 110.0 ,
       115.0 , 125.0 , 130.0 , 120.0 , 120.0 , 130.0 , 135.0 , 110.0 , 130.0 ,
       150.0 , 145.0 , 150.0 , 170.0 , 225.0 , 145.0 , 188.0 , 180.0 , 197.0 ,
       218.0 , 300.0 , 260.0 , 265.0 , 250.0 , 250.0 , 300.0 , 320.0 , 514.0 ,
       556.0 , 840.0 , 685.0 , 700.0 , 700.0 , 690.0 , 900.0 , 650.0 , 820.0 ,
       850.0 , 900.0 , 1015.0 , 820.0 , 1100.0 , 1000.0 , 1100.0 , 1000.0 ,
       1000.0 ])
```

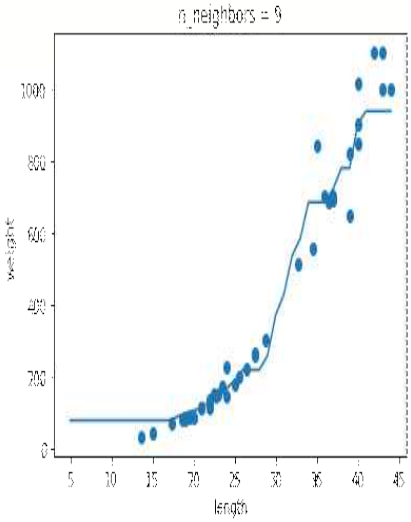
```
# 훈련 세트 준비
from  sklearn.model_selection import  train_test_split
train_input, test_input, train_target, test_target = train_test_split(
    perch_length, perch_weight, random_state=42 )
train_input = train_input.reshape(-1 ,1 )
test_input = test_input.reshape(-1 ,1 )
```

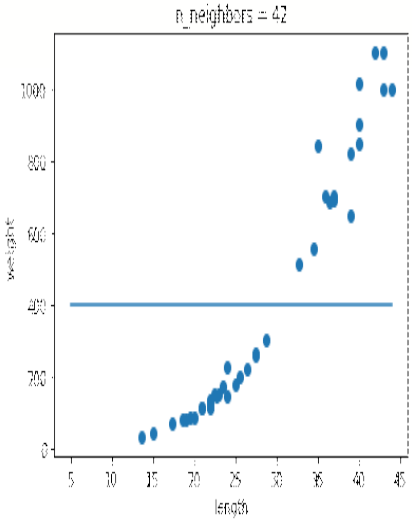
```
# 회귀모델 훈련
from  sklearn.neighbors import  KNeighborsRegressor
knr = KNeighborsRegressor()
knr.fit(train_input, train_target)
print (knr.score(test_input, test_target)) # (1 - (타깃 - 예측)^2의 합)/(타깃 -예측)^2의 합
from  sklearn.metrics import  mean_absolute_error
test_prediction = knr.predict(test_input) # 테스트 세트에 대한 예측을 만듭니다
mae = mean_absolute_error(test_target, test_prediction)
# 테스트 세트에 대한 평균 절댓값 오차를 계산합니다
print (mae)
```

```
# '오차율이 가장 적게 나오는 이웃 숫자가 최적의 이웃 숫자다'라는 가정으로 접근
# 최적의 이웃 숫자 구하기.
best_rate=1
for i in range (1 ,train_input.size+1 ):
  knr.n_neighbors = i # default = 5
  knr.fit(train_input, train_target)
  trainset = knr.score(train_input, train_target)
  testset = knr.score(test_input, test_target)
  error_rate = trainset-testset
  if  best_rate > abs (error_rate):
    best_rate = abs (error_rate)
    best_k = i
# 오차율이 너무 큰 양수면 과대적합, 너무 작은 음수는 과소적합.
print ("best_k = ",best_k,"\nbest_error_rate = ",best_rate,"\n")
result : min_k = 9 min_error_rate = 6.626178285862316e-05
```

```
# 그래프 시각화
# k-최근접 이웃 회귀 객체를 만듭니다
knr = KNeighborsRegressor()
# 5에서 45까지 x 좌표를 만듭니다
x = np.arange(5 , 45 ).reshape(-1 , 1 )
# n = 3, 9, 42일 때 예측 결과를 그래프로 그립니다.
for  n in  [3 , 9 , 42 ]:
    # 모델 훈련
    knr.n_neighbors = n # 이웃 숫자 결정
    knr.fit(train_input, train_target) # 정해진 이웃 숫자를 바탕으로 훈련
    # 지정한 범위 x에 대한 예측 구하기
    prediction = knr.predict(x)
    # 훈련 세트와 예측 결과 그래프 그리기
    plt.scatter(train_input, train_target)
    plt.plot(x, prediction)
    plt.title('n_neighbors = {}'.format (n))
    plt.xlabel('length')
    plt.ylabel('weight')
    plt.show()
```







이 데이터에서 오차율이 가장 적은 최적의 이웃 숫자는 9입니다.
최적의 이웃 숫자 K = 9

2. (KNN 분류)에서 거리 기반으로 최적 거리 찾기(그래프도 그림)

- 표준 점수를 통해 편차가 큰 데이터를 표준화한다.

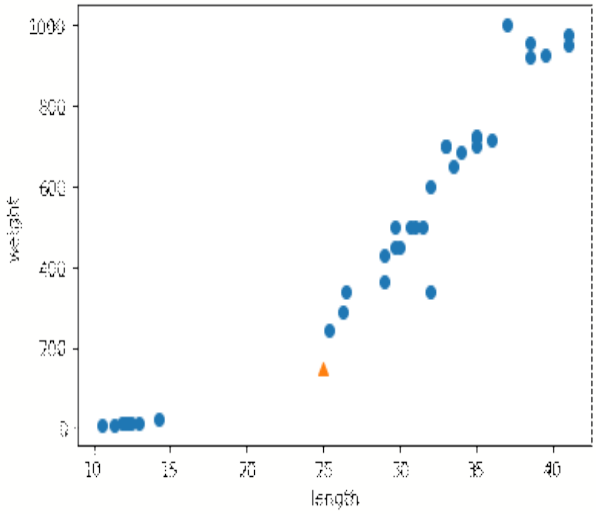
표준 점수 : (특성 - 평균) / 표준편차

```
# 도미의 데이터
bream_length = [25.4 , 26.3 , 26.5 , 29.0 , 29.0 , 29.7 , 29.7 , 30.0 , 30.0 , 30.7
, 31.0 , 31.0 , 31.5 , 32.0 , 32.0 , 32.0 , 33.0 , 33.0 , 33.5 , 33.5 , 34.0 , 34.0
, 34.5 , 35.0 , 35.0 , 35.0 , 35.0 , 36.0 , 36.0 , 37.0 , 38.5 , 38.5 , 39.5 , 41.0
, 41.0 ]
bream_weight = [242.0 , 290.0 , 340.0 , 363.0 , 430.0 , 450.0 , 500.0 , 390.0 , 450.0
, 500.0 , 475.0 , 500.0 , 500.0 , 340.0 , 600.0 , 600.0 , 700.0 , 700.0 , 610.0 , 650.0
, 575.0 , 685.0 , 620.0 , 680.0 , 700.0 , 725.0 , 720.0 , 714.0 , 850.0 , 1000.0 , 920.0
, 955.0 , 925.0 , 975.0 , 950.0 ]
# 빙어의 데이터
smelt_length = [9.8 , 10.5 , 10.6 , 11.0 , 11.2 , 11.3 , 11.8 , 11.8 , 12.0 , 12.2
, 12.4 , 13.0 , 14.3 , 15.0 ]
smelt_weight = [6.7 , 7.5 , 7.0 , 9.7 , 9.8 , 8.7 , 10.0 , 9.9 , 9.8 , 12.2 , 13.4
, 12.2 , 19.7 , 19.9 ]
```

수상한 도미 한 마리

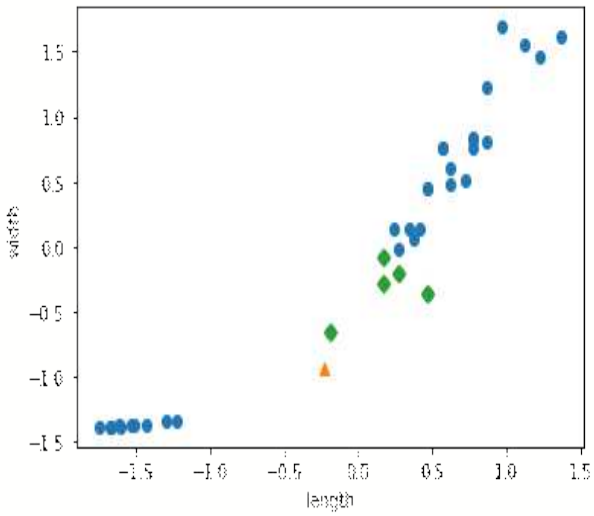
```
from  sklearn.neighbors import  KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(train_input, train_target)
kn.score(test_input, test_target)

print (kn.predict([[25 , 150 ]])) # 빙어로판별

import  matplotlib.pyplot as  plt

plt.scatter(train_input[:,0 ], train_input[:,1 ])
plt.scatter(25 , 150 , marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()

distances, indexes = kn.kneighbors([[25 , 150 ]])

plt.scatter(train_input[:,0 ], train_input[:,1 ])
plt.scatter(25 , 150 , marker='^')
plt.scatter(train_input[indexes,0 ], train_input[indexes,1 ], marker='D')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```

```
# 길이보다 무게에 편향되게 판별됨.
plt.scatter(train_input[:,0 ], train_input[:,1 ])
plt.scatter(25 , 150 , marker='^')
plt.scatter(train_input[indexes,0 ], train_input[indexes,1 ], marker='D')
plt.xlim((0 , 1000 ))
plt.xlabel('length')
plt.ylabel('weight')
plt.show()

# 하단의 산점도를 보면 길이는 '5'단위인 반면 무게는 '200'단위라 편차가 심하다.
```




```
# 훈련
from  sklearn.model_selection import  train_test_split
train_input, test_input, train_target, test_target = train_test_split(fish_data, fish_t
arget, stratify=fish_target, random_state=42 )
print (train_input.shape, test_input.shape) # 배열 모양 확인
print (train_target.shape, test_target.shape) # 배열 모양 확인
mean = np.mean(train_input, axis=0 ) # 훈련 데이터의 평균
std = np.std(train_input, axis=0 ) # 훈련 데이터의 표준편차
train_scaled = (train_input - mean) / std # 훈련데이터의 (특성-평균)/표준편차

# 테스트 케이스[25,150] 표준 점수로 바꾸기.
new = ([25 , 150 ] - mean) /std # (테스트 케이스 특성 - 평균) / 표준편차
plt.scatter(train_scaled[:,0 ], train_scaled[:,1 ]) # 훈련 데이터 x, y축
plt.scatter(new[0 ], new[1 ], marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()

distances, indexes = kn.kneighbors([new]) # [new]에 가까운 5개 객체 거리와 인덱스 가져오기.

distances # 표준 점수화된 5개 객체 거리
# Output : array([[0.2873737 , 0.7711188 , 0.89552179, 0.91493515, 0.95427626]])

indexes # 표준 점수화된 기준으로 테스트 케이스 new에 가장 근접한 5개의 인덱스
# Output : array([[21, 14, 34, 32, 5]])

print (kn.predict([new])) # new = ([25,150] - mean)/ std
# Output : [1.]  도미  => 표준 점수화된 기준으로 판별하면 정상적으로 도미라고 판별한다.
# 훈련 데이터 : 파란원 / 최단 거리에 5개의 원 : 초록색 / 세모 : 테스트 케이스
```

```
plt.scatter(train_scaled[:,0 ], train_scaled[:,1 ])
plt.scatter(new[0 ], new[1 ], marker='^')
plt.scatter(train_scaled[indexes,0 ], train_scaled[indexes,1 ], marker='D')
plt.xlabel('length')
plt.ylabel('width')
plt.show()
```



설명 : 표준 점수공식으로 길이와 무게를 표준화해서 길이와 무게 간의 큰 편차로 인해 테스트 케이스가 오류가 나는 상황을 방지했다.