201645084_3_Q_노영훈

과제 : KNN 분류에서 최적 하이퍼 파라미터 찾기

1. (KNN 분류)에서 최적의 이웃 숫자 찾기(그래프도 그림)

데이터 준비

```
# 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 ]


# 생선 데이터로 통합
length = bream_length+smelt_length
weight = bream_weight+smelt_weight
fish_data = []
fish_data = [[l, w] for  l, w in  zip (length,weight)]
fish_data = np.column_stack((length, weight))
fish_target = [1 ]*35  +[0 ]*14

input_arr = np.array(fish_data)
target_arr = np.array(fish_target)


# 훈련 데이터와 실전 데이터로 분류
np.random.seed(42 )
index = np.arange(49 )
np.random.shuffle(index)
train_input = input_arr[index[:35 ]] # 훈련 데이터
train_target = target_arr[index[:35 ]]  # 훈련 라벨

test_input = input_arr[index[35 :]]
test_target = target_arr[index[35 :]]
```

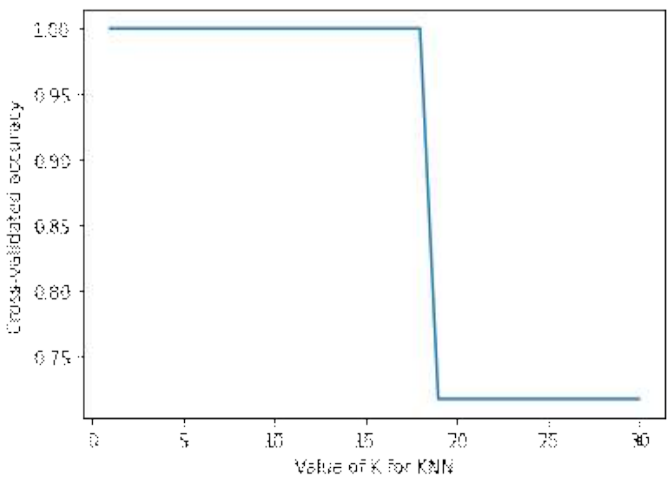
훈련 및 정규화

```
# 훈련
from  sklearn.model_selection import  train_test_split
train_input, test_input, train_target, test_target = train_test_split(fish_data, fish_target, stratify=fish_target, ran
dom_state=42 )
print (train_input.shape, test_input.shape)
print (train_target.shape, test_target.shape)
```

```
# 정규화(모든 데이터가 동일한 정도의 중요도가 반영되게 하는 작업)
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

# KNN 피팅
from  sklearn import  neighbors, datasets

import  numpy as  np
import  matplotlib.pyplot as  plt
from  matplotlib.colors import  ListedColormap
X = train_scaled
y = train_target
clf = neighbors.KNeighborsClassifier(5 )
clf.fit(X,y)
y_pred=clf.predict(X)
from  sklearn.metrics import  confusion_matrix
confusion_matrix(y,y_pred)
```

cross-validation을 통한 K 찾기(정확성이 가장 높은 K가 최적의 이웃 숫자이다)

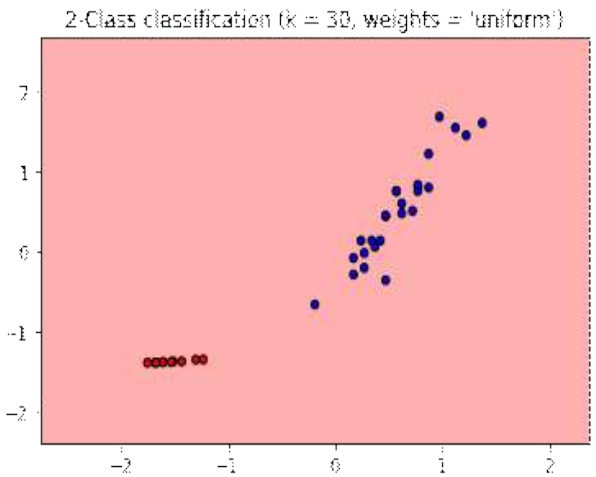
```
# 최적의 k 찾기
from  sklearn.model_selection import  cross_val_score
k_range = range (1 ,31 )
k_scores = []
best_k =[]
best_score =0
for  k in  k_range:
    knn = neighbors.KNeighborsClassifier(k)
    # 모델명 : knn, 훈련데이터 : X, 타깃 : y, 폴드 수 : cv(몇 겹으로 교차검증을 할 것인지)
    scores = cross_val_score(knn, X, y, cv = 10 , scoring='accuracy')
    k_scores.append(scores.mean())
    k_score = scores.mean()
    if  k_score >= best_score:
      best_score = k_score
      best_k.append(k)

plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-validated accuracy')
plt.show()
print ("최적의 k = ", best_k)
최적의 k = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
```

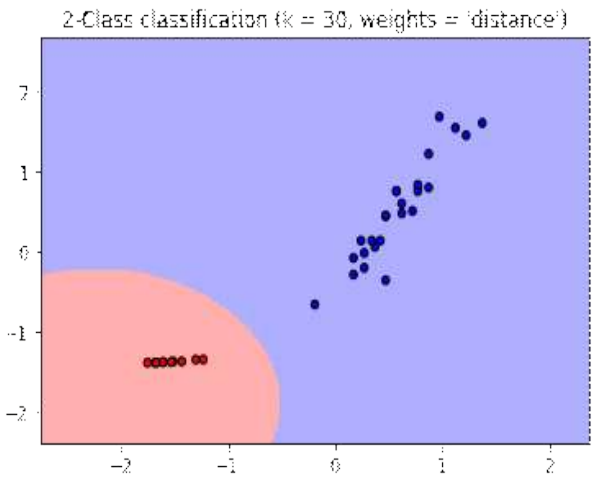


최적의 K는 1부터 18까지입니다.

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

```
n_neighbors = 30
h = .02
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
for  weights in  ['uniform', 'distance']:
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    x_min, x_max = X[:, 0 ].min () - 1 , X[:, 0 ].max () + 1
    y_min, y_max = X[:, 1 ].min () - 1 , X[:, 1 ].max () + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    plt.scatter(X[:, 0 ], X[:, 1 ], c=y, cmap=cmap_bold,
                edgecolor='k', s=20 )
    plt.xlim(xx.min (), xx.max ())
    plt.ylim(yy.min (), yy.max ())
    plt.title("2-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))
plt.show()
```





교차검증이란?

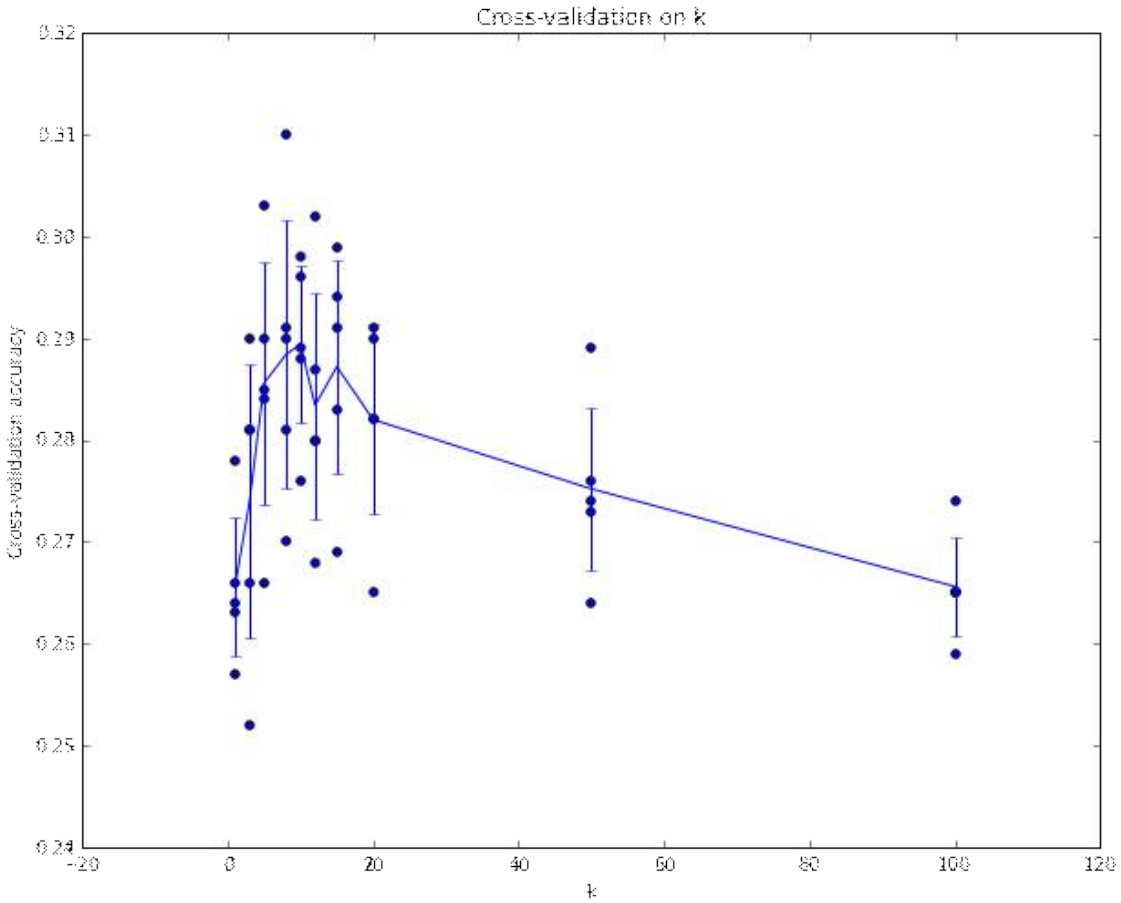
이렇듯 고정된 train set과 test set으로 평가를 하고, 반복적으로 모델을 튜닝하다 보면 test set에만 과적합 되어버리는 결과가 생긴다.
이를 해결하고자 하는 것이 바로 **교차 검증(cross validation)**이다.

장점 :
1. 모든 데이터 셋을 평가에 활용할 수 있다.
 - 평가에 사용되는 데이터 편중을 막을 수 있다.

(특정 평가 데이터 셋에 overfit 되는 것을 방지할 수 있다.)
 - 평가 결과에 따라 좀 더 일반화된 모델을 만들 수 있다.

2. 모든 데이터 셋을 훈련에 활용할 수 있다.
 - 정확도를 향상시킬 수 있다.
 - 데이터 부족으로 인한 underfitting을 방지할 수 있다.

단점 : Iteration 횟수가 많기 때문에 모델 훈련/평가 시간이 오래 걸린다.



5-Fold Cross-validation을 이용한 kNN 알고리즘의 Hyperparameter 튜닝 예. [Image from Stanford Univ. CS231n]

5-Fold Cross-validation을 하는 과정은 앞서 설명드린 과정을 그래도 한 것입니다. 위의 그래프에서 가로축은 kNN의 k(테스트 데이터의 Metric에 대하여 가장 가까운 k개의 데이터)를, 세로축은 k의 값에 따라 Validation Fold를 변경해가며 얻은 Cross-validation의 정확도를 나타낸 것입니다.

5-Fold Cross-validation을 수행하였으므로 세로축 각 k에 대하여 5개씩의 결과를 얻었습니다. 이에 대한 평균값을 구한 것이 파란색 라인입니다. 파란색 라인을 따라가다 보면 대략 k = 7에서 가장 높은 정확도를 얻었음을 알 수 있습니다. 따라서, 주어진 Dataset에 대해서는 kNN 모델의 k값을 7로 정해서 사용하는 것이 가장 효과적임을 알 수 있습니다.

참고문헌 :

[Artificial Intelligence / Posts] 교차검증 (Cross-validation)
https://cinema4dr12.tistory.com/1275

KNN 교차검증 적용
https://velog.io/@guns/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-%EC%8A%A4%ED%84%B0%EB%94%94-3%EC%9D%BC%EC%B0%A8-K-NN-K-Nearest-Neighbor