강원지역혁신플랫폼

기계학습

Machine Learning

K-최근접 이웃 분석 실습(1)

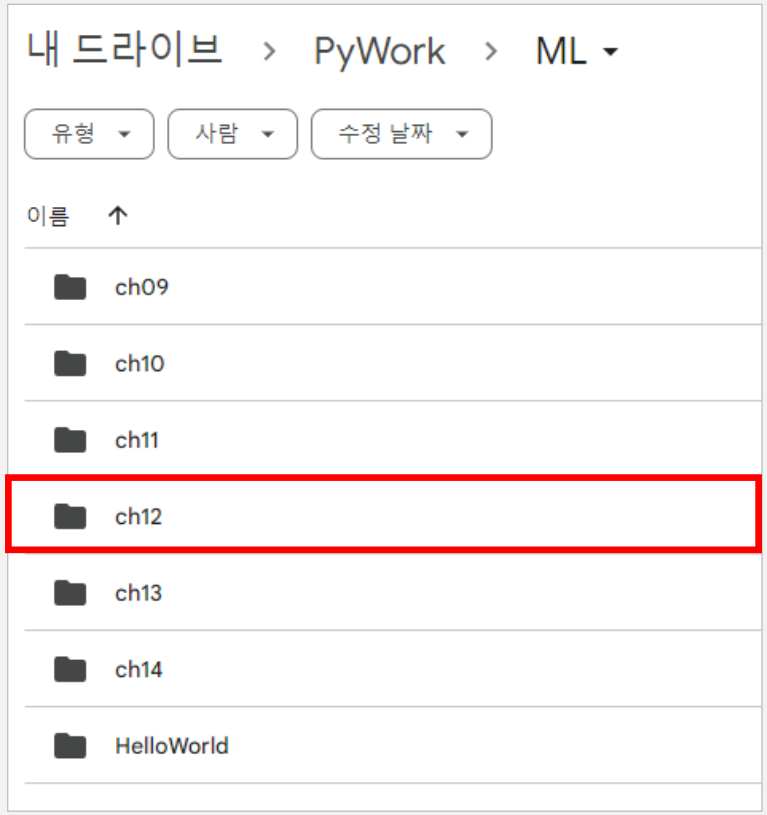
학습목표

농구 선수 포지션 예측에 K-최근접 이웃 모델을 활용할 수 있습니다.

01 | 실습

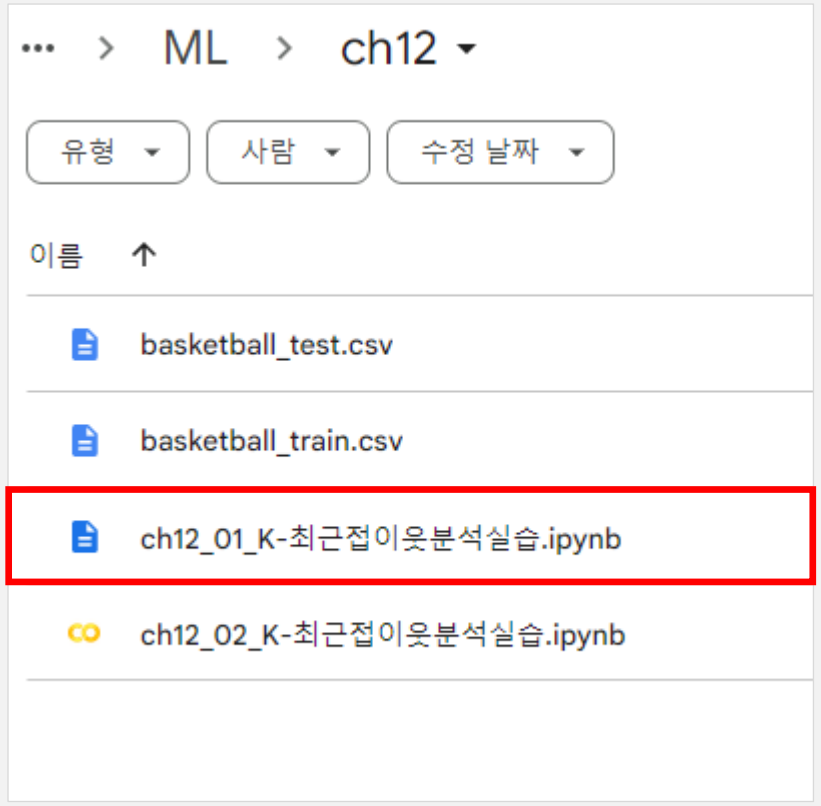
(권장) 아래와 같은 경로에 실행 소스가 존재하면 환경 구축 완료

- 구글 드라이브 "PyWork > ML" 폴더로 이동함
  - 아래의 [ch12] 폴더를 클릭하면 됨



01 | 실습

- “ML > ch12 >” 폴더를 클릭함
  - 아래의 [ch12_01_K-최근접이웃분석실습.ipynb] 스크립트를 클릭함



02 | K-최근접 이웃(K-NN) 분석 실습

K-최근접 이웃(K-NN) 분석 실습

- 다음은 K-NN 모델로 농구선수의 3점슛(3P), 블로킹(BLK), 리바운드(TRB) 데이터 셋으로 농구선수 포지션 예측해 보자.
  - 훈련 데이터와 테스트 데이터는 아래의 경로를 참고함
    - 훈련 데이터 : https://raw.githubusercontent.com/wikibook/machine-learning/2.0/data/csv/basketball_train.csv
    - 테스트 데이터 : https://raw.githubusercontent.com/wikibook/machine-learning/2.0/data/csv/basketball_test.csv

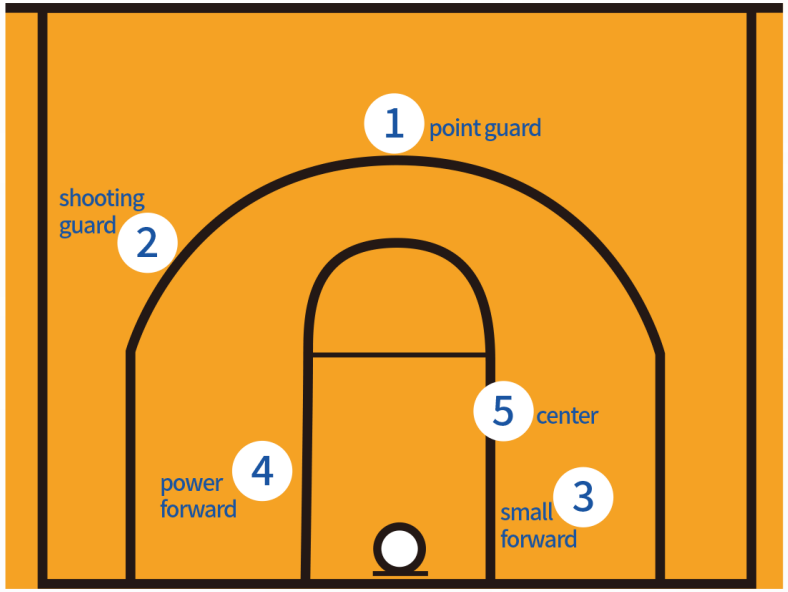
02 | K-최근접 이웃(K-NN) 분석 실습

농구 포지션은 크게 세가지로 나눔

- 가드(Guard), 포워드(Forward), 센터(Center)

농구 포지션을 세부적으로 나누게 되면 다섯가지로 나눔

- 포인트 가드(Point Guard), 슛팅 가드(Shooting Guard),
  스몰 포워드(Small Forward), 파워 포워드(Power Forward), 센터(Center)



PG:포인트 가드
SG:슛팅 가드
SF:스몰 포워드
PF:파워 포워드
C:센터

출처: https://namu.wiki/w/농구/포지션

02 | K-최근접 이웃(K-NN) 분석 실습

다음은 농구선수의 3점슛(3P), 블로킹(BLK), 리바운드(TRB) 데이터 셋으로 농구선수 포지션 데이터셋은 다음과 같음

- 훈련 데이터와 테스트 데이터 집합의 관측치와 속성 구조는 다음과 같이 구성됨
  - 훈련 데이터 집합은 80개의 관측치
  - 테스트 데이터 집합은 20개의 관측치

Player	Pos	3P	TRB	BLK
JaVale McGee	C	0	3.2	0.9
Manu Ginobili	SG	1.3	2.3	0.2
Nene Hilario	C	0	4.2	0.6
...	...	...	...	...

※ 3점슛(3P), 블로킹(BLK), 리바운드(TRB)

02 | K-최근접 이웃(K-NN) 분석 실습

다음은 농구선수의 3점슛(3P), 블로킹(BLK), 리바운드(TRB) 데이터 셋을 읽어오는 코드이다.

- 실행결과 훈련 데이터는 80개 관측치와 5개 속성, 테스트 데이터는 20개 관측치와 5개 속성으로 구성된 것을 볼 수 있음

```
train_file_url = "https://raw.githubusercontent.com/wikibook/machine-learning/2.0/data/csv/basketball_train.csv"
test_file_url = "https://raw.githubusercontent.com/wikibook/machine-learning/2.0/data/csv/basketball_test.csv"
train = pd.read_csv(train_file_url)
test = pd.read_csv(test_file_url)
print(train.shape)   # (80, 5)
print(train[:5])
print('-' * 30)
print(test.shape)    # (20, 5)
print(test[:5])
```

```
(80, 5)
            Player Pos   3P  TRB  BLK
0  Denzel Valentine  SG  1.3  2.6  0.1
1       Kyle Korver  SG  2.4  2.8  0.3
2      Troy Daniels  SG  2.1  1.5  0.1
3      Tim Hardaway  SG  1.9  2.8  0.2
4    Dewayne Dedmon   C  0.0  6.5  0.8
------------------------------
(20, 5)
                Player Pos   3P  TRB  BLK
0          JaYale McGee   C  0.0  3.2  0.9
1         Manu Ginobili  SG  1.3  2.3  0.2
2          Nene Hilario   C  0.0  4.2  0.6
3        Evan Fournier  SG  1.9  3.1  0.1
4  Georgios Papagiannis   C  0.0  3.9  0.8
```

02 | K-최근접 이웃(K-NN) 분석 실습

다음은 범주형 데이터 유형의 빈도수를 반환하는 count_frequency() 함수를 정의하는 코드이다.

- count_frequency() 함수에 리스트를 입력해서 빈도 역순으로 반환함

```
# 빈도 출력 함수 : (key: 리스트요소, value: 빈도수)로 이루어진 딕셔너리 생성
def count_frequency(x_list):

  count = {}

  for item in x_list:
    count[item] = count.get(item, 0) + 1

  # 내림차순: 빈도수 높은 순으로 정렬
  sorted_count = sorted(count.items(), key = operator.itemgetter(1), reverse = True)

  return sorted_count
```

02 | K-최근접 이웃(K-NN) 분석 실습

- 다음은 count_frequency() 함수로 훈련 데이터 셋의 농구선수 Pos 속성의 빈도수를 출력하는 코드이다.
  - 실행결과 슛팅 가드(SG) 41명, 센터(C) 39명으로 구성된 것을 알 수 있음

```
count_frequency(train.Pos)
```

```
[('SG', 41), ('C', 39)]
```

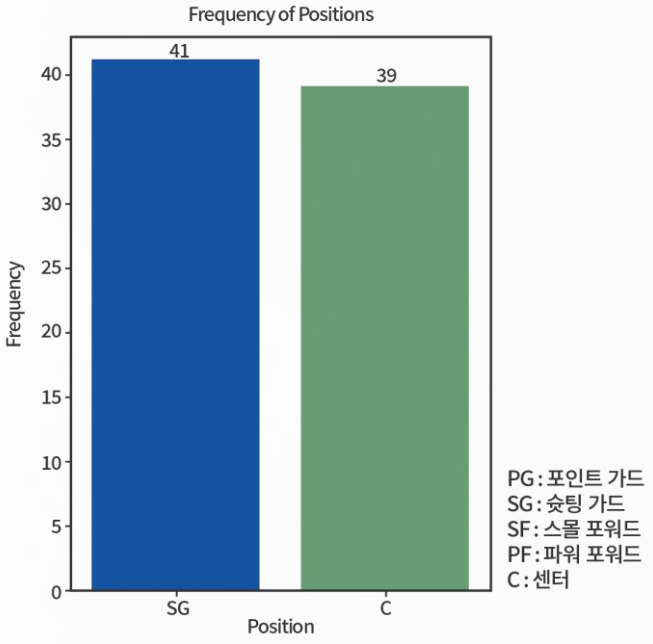
PG:포인트 가드
SG:슛팅 가드
SF:스몰 포워드
PF:파워 포워드
C:센터

02 | K-최근접 이웃(K-NN) 분석 실습

다음은 훈련 데이터 셋의 농구선수 Pos 속성의 빈도 그래프를 출력하는 코드이다.

- Seaborn 라이브러리의 countplot() 함수를 이용함
- 실행결과 SG(슛팅 가드) 41명, C(센터) 39명으로 구성된 것을 볼 수 있음

```
# 빈도 그래프 그리기
plt.figure(figsize=(6, 8))
ax = sns.countplot(x='Pos', data=train, palette='viridis')

# 각 막대 위에 빈도 수 표시
for p in ax.patches:
   ax.annotate(format(p.get_height(), '.0f'),
            (p.get_x() + p.get_width() / 2., p.get_height()),
            ha = 'center', va = 'center',
            xytext = (0, 10),
            textcoords = 'offset points')
plt.title('Frequency of Positions')
plt.xlabel('Position')
plt.ylabel('Frequency')
plt.show()
```

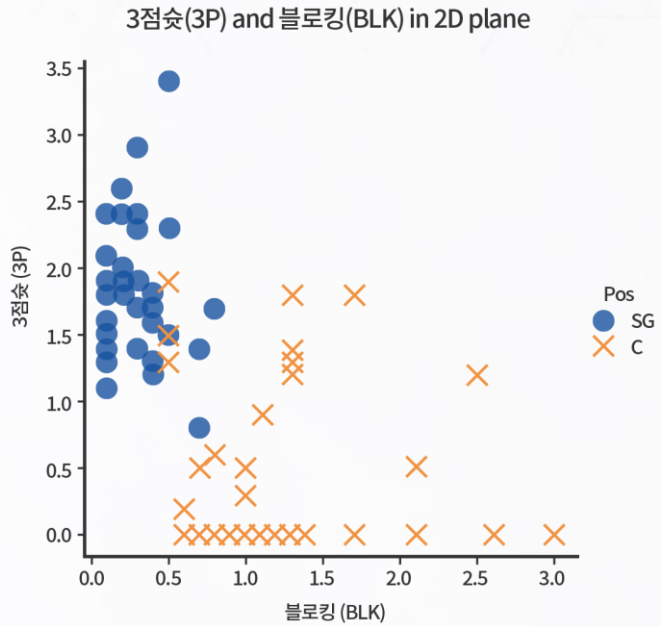


02 | K-최근접 이웃(K-NN) 분석 실습

다음은 훈련 데이터의 3점슛(3P)과 블로킹(BLK) 변수로 산점도를 그리는 코드이다.

- 실행결과 슛팅가드(SG)는 3점슛이 많고, 센터(C)는 블로킹이 많은 것을 볼 수 있음
- seaborn 라이브러리의 lmplot() 함수는 산점도, 회귀선, 신뢰 구간을 동시에 표현해줌

```
# 3점슛과 블로킹 데이터 시각화
sns.lmplot(x='BLK', y='3P', data=train, fit_reg=False,
           scatter_kws={"s": 150},
           markers=["o", "x"], hue="Pos")
plt.title('3점슛(3P) and 블로킹(BLK) in 2D plane')
plt.xlabel('블로킹 (BLK)')
plt.ylabel('3점슛 (3P)')
plt.show()
```



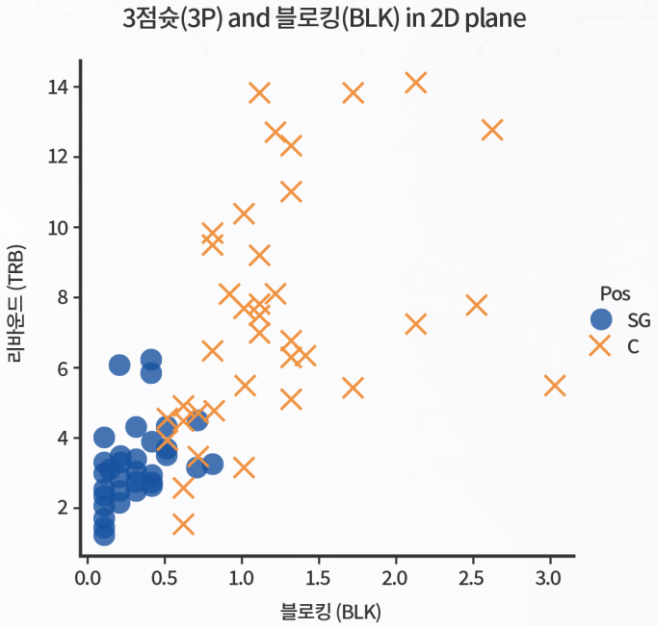
PG:포인트가드/ SG:슛팅가드/ SF:스몰포워드/ PF:파워포워드/ C:센터

02 | K-최근접 이웃(K-NN) 분석 실습

다음은 훈련 데이터의 블로킹(BLK)과 리바운드(TRB) 변수로 산점도를 그리는 코드이다.

- 실행결과 슛팅가드(SG)는 블로킹과 리바운드 모두 낮은 빈도임
  - 센터(C)는 블로킹과 리바운드 모두 높은 빈도인 것을 볼 수 있음

```
sns.lmplot(x='BLK', y='TRB', data=train, fit_reg=False,
           scatter_kws={"s": 150},
           markers=["o", "x"], hue="Pos")
plt.title('3점슛(3P) and 블로킹(BLK) in 2D plane')
plt.xlabel('블로킹 (BLK)')
plt.ylabel('리바운드 (TRB)')
plt.show()
```



PG:포인트가드/ SG:슛팅가드/ SF:스몰포워드/ PF:파워포워드/ C:센터

02 | K-최근접 이웃(K-NN) 분석 실습

다음은 최적의 K를 찾기 위한 10-겹 교차검증 수행하는 코드이다.

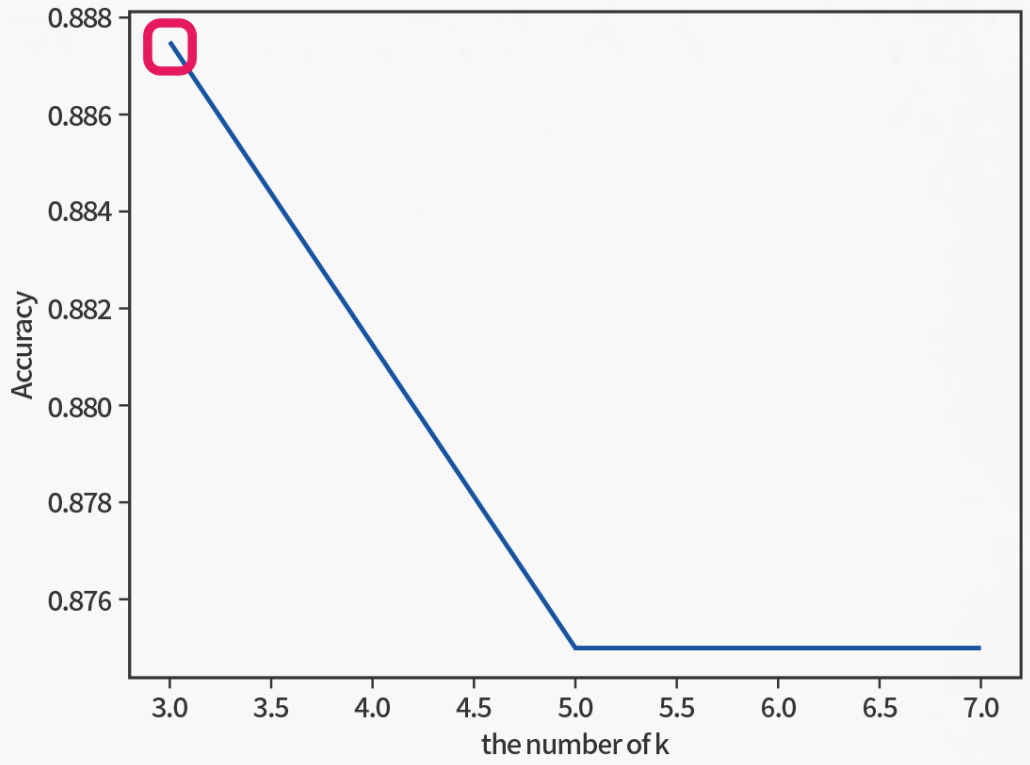
- 다만 K값은 3부터 √관측치의 개수 범위 이내이면서 2씩 증가시킴

```
max_k_range = int(math.sqrt(train.shape[0]))
k_list = []
for i in range(3, max_k_range, 2):
    k_list.append(i)

cross_validation_scores = []
x_train = train[['3P', 'BLK' , 'TRB']]
y_train = train[['Pos']]

# 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_train, y_train.values.ravel(), cv=10, scoring='accuracy')
    cross_validation_scores.append(scores.mean())
cross_validation_scores

# visualize accuracy according to k
plt.plot(k_list, cross_validation_scores)
plt.xlabel('the number of k')
plt.ylabel('Accuracy')
plt.show()
```

02 | K-최근접 이웃(K-NN) 분석 실습

✦ 실행결과는 아래 그림과 같음

› 아래결과에서 K값은 3 ~ 7 범위 중에서 K=3인 경우 정확도가 가장 높은 것을 알 수 있음



02 | K-최근접 이웃(K-NN) 분석 실습

다음은 최적의 K값을 확인하는 코드이다.

- 실행결과는 최적의 K값은 3인 것을 알 수 있음

```
cvs = cross_validation_scores
k = k_list[cvs.index(max(cross_validation_scores))]
print("The best number of k : " + str(k))          # The best number of k : 3
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델1

모델1 : 독립 변수는 3점슛(3P), 블로킹(BLK)

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK) 변수로 K-NN 모델을 학습하는 코드이다.

- 여기서 K = 3으로 설정함
  - 아래와 같이 fit() 함수로 모델을 학습함

```
knn1 = KNeighborsClassifier(n_neighbors=k)     # knn 모델 생성

x_train = train[['3P', 'BLK']]          # select data features
y_train = train[['Pos']]                # select target value

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)       # 정규화
knn1.fit(x_train, y_train.values.ravel())   # setup knn using train data
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델1

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 평가하는 코드이다.

- 모델 평가 결과 정확도가 100%인 것을 볼 수 있음

```
x_test = test[['3P', 'BLK']]   # select data feature to be used for prediction
y_test = test[['Pos']]         # select target value

x_test = scaler.transform(x_test) # 정규화
knn1.score(x_test, y_test)          # 1.0
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델1

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 예측하는 코드이다.

- 아래와 같이 모델이 예측한 결과를 확인할 수 있음

```
pred = knn1.predict(x_test)
Pred
```

```
array(['C', 'SG', 'C', 'SG', 'C', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'C',
       'SG', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'C'], dtype=object)
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델1

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK) 변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델의 예측값과 실제값을 비교하는 코드이다.

- 아래와 같이 모델의 예측값과 실제값이 동일한 것을 확인할 수 있음

```
comparison = pd.DataFrame({'prediction':pred, 'ground_truth':y_test.values.ravel()})
comparison
```

	prediction	ground_truth
0	C	C
1	SG	SG
2	C	C
3	SG	SG
4	C	C
5	C	C
6	C	C
7	SG	SG
8	SG	SG
9	C	C
10	SG	SG
11	C	C
12	SG	SG
13	C	C
14	C	C
15	SG	SG
16	SG	SG
17	C	C
18	SG	SG
19	C	C

→ 예측값과 실제값이 일치

02 | K-최근접 이웃(K-NN) 분석 실습: 모델2

모델2 : 독립 변수는 3점슛(3P), 리바운드(TRB)

다음은 훈련 데이터 셋의 3점슛(3P), 리바운드(TRB) 변수로 K-NN 모델을 학습하는 코드이다.

- 여기서 K = 3으로 설정함
  - 아래와 같이 fit() 함수로 모델을 학습함

```
knn2 = KNeighborsClassifier(n_neighbors=k) # knn 모델 생성

x_train = train[['3P', 'TRB']]                  # select data features
y_train = train[['Pos']]                        # select target value
x_train = scaler.fit_transform(x_train)         # 정규화
knn2.fit(x_train, y_train.values.ravel())       # setup knn using train data
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델2

다음은 훈련 데이터 셋의 3점슛(3P), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 평가하는 코드이다.

- 모델 평가 결과 정확도가 95%인 것을 볼 수 있음

```
x_test = test[['3P', 'TRB']]  # select data feature to be used for prediction
y_test = test[['Pos']]        # select target value

x_test = scaler.transform(x_test) # 정규화
knn2.score(x_test, y_test)    # 0.95
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델2

- 다음은 훈련 데이터 셋의 3점슛(3P), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 예측하는 코드이다.
  - 아래와 같이 모델이 예측한 결과를 확인할 수 있음

```
pred = knn2.predict(x_test)
Pred
```

```
array(['C', 'SG', 'C', 'SG', 'C', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'SG',
       'SG', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'C'], dtype=object)
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델2

다음은 훈련 데이터 셋의 3점슛(3P), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델의 예측값과 실제값을 비교하는 코드이다.

- 아래와 같이 예측값과 실제값이 불일치한 것을 확인할 수 있음

```
comparison = pd.DataFrame({'prediction':pred, 'ground_truth':y_test.values.ravel()})
comparison
```

	prediction	ground_truth
0	C	C
1	SG	SG
2	C	C
3	SG	SG
4	C	C
5	C	C
6	C	C
7	SG	SG
8	SG	SG
9	C	C
10	SG	SG
11	SG	C
12	SG	SG
13	C	C
14	C	C
15	SG	SG
16	SG	SG
17	C	C
18	SG	SG
19	C	C

→ 11번째 인덱스의 예측값과 실제값이 불일치

02 | K-최근접 이웃(K-NN) 분석 실습: 모델3

모델3 : 독립 변수는 블로킹(BLK), 리바운드(TRB)

다음은 훈련 데이터 셋의 블로킹(BLK), 리바운드(TRB) 변수로 K-NN 모델을 학습하는 코드이다.

- 여기서 K = 3으로 설정함
  - 아래와 같이 fit() 함수로 모델을 학습함

```
knn3 = KNeighborsClassifier(n_neighbors=k) # knn 모델 생성

x_train = train[['BLK', 'TRB']]                  # select data features
y_train = train[['Pos']]                         # select target value

x_train = scaler.fit_transform(x_train)   # 정규화
knn3.fit(x_train, y_train.values.ravel()) # setup knn using train data
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델3

다음은 훈련 데이터 셋의 블로킹(BLK), 리바운드(TRB) 변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 평가하는 코드이다.

- 모델 평가 결과 정확도가 85%인 것을 볼 수 있음

```
x_test = test[['BLK', 'TRB']]  # select data feature to be used for prediction
y_test = test[['Pos']]         # select target value

x_test = scaler.transform(x_test)  # 정규화
knn3.score(x_test, y_test)         # 0.85
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델3

다음은 훈련 데이터 셋의 블로킹(BLK), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 예측하는 코드이다.

- 아래와 같이 모델이 예측한 결과를 확인할 수 있음

```
pred = knn3.predict(x_test)
pred
```

```
array(['SG', 'SG', 'C', 'SG', 'SG', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'C',
       'SG', 'SG', 'C', 'SG', 'SG', 'C', 'SG', 'C'], dtype=object)
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델2

다음은 훈련 데이터 셋의 블로킹(BLK), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델의 예측값과 실제값을 비교하는 코드이다.

- 아래와 같이 예측값과 실제값이 불일치한 것을 확인할 수 있음

```
comparison = pd.DataFrame({'prediction':pred, 'ground_truth':y_test.values.ravel()})
comparison
```

	prediction	ground_truth
0	SG	C
1	SG	SG
2	C	C
3	SG	SG
4	SG	C
5	C	C
6	C	C
7	SG	SG
8	SG	SG
9	C	C
10	SG	SG
11	C	C
12	SG	SG
13	SG	C
14	C	C
15	SG	SG
16	SG	SG
17	C	C
18	SG	SG
19	C	C

→ 0, 4, 13번째 인덱스의 예측값과 실제값이 불일치

02 | K-최근접 이웃(K-NN) 분석 실습: 모델4

모델4 : 독립 변수는 3점슛(3P), 블로킹(BLK), 리바운드(TRB)

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK), 리바운드(TRB) 변수로 K-NN 모델을 학습하는 코드이다.

- 여기서 K = 3으로 설정함
  - 아래와 같이 fit() 함수로 모델을 학습함

```
knn4 = KNeighborsClassifier(n_neighbors=k)

x_train = train[['3P', 'BLK', 'TRB']]          # select data features to be used in train
y_train = train[['Pos']]                       # select target

x_train = scaler.fit_transform(x_train)  # 정규화
knn4.fit(x_train, y_train.values.ravel()) # 데이터 학습
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델4

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 평가하는 코드이다.

- 모델 평가 결과 정확도가 100%인 것을 볼 수 있음

```
x_test = test[['3P', 'BLK', 'TRB']]     # select features to be used for prediction
y_test = test[['Pos']]                  # select target

x_test = scaler.transform(x_test)       # 정규화
knn4.score(x_test, y_test)              # 모델 평가 : 1.0
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델4

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK), 리바운드(TRB)변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델을 예측하는 코드이다.

- 아래와 같이 모델이 예측한 결과를 확인할 수 있음

```
pred = knn4.predict(x_test)     # 예측
pred
```

```
array(['C', 'SG', 'C', 'SG', 'C', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'C',
       'SG', 'C', 'C', 'SG', 'SG', 'C', 'SG', 'C'], dtype=object)
```

02 | K-최근접 이웃(K-NN) 분석 실습: 모델4

다음은 훈련 데이터 셋의 3점슛(3P), 블로킹(BLK), 리바운드(TRB) 변수로 학습된 K-NN 모델로 테스트 데이터 셋으로 모델의 예측값과 실제값을 비교하는 코드이다.

- 아래와 같이 예측값과 실제값이 일치하는 것을 확인할 수 있음

```
comparison = pd.DataFrame({'prediction':pred, 'ground_truth':y_test.values.ravel()})
comparison
```

	prediction	ground_truth
0	C	C
1	SG	SG
2	C	C
3	SG	SG
4	C	C
5	C	C
6	C	C
7	SG	SG
8	SG	SG
9	C	C
10	SG	SG
11	C	C
12	SG	SG
13	C	C
14	C	C
15	SG	SG
16	SG	SG
17	C	C
18	SG	SG
19	C	C

→ 예측값과 실제값이 일치

02 | K-최근접 이웃(K-NN) 분석 실습: 모델 평가 비교

모델 평가 결과 비교

농구선수의 3점슛(3P), 블로킹(BLK), 리바운드(TRB) 데이터 셋으로 농구선수 포지션 예측하는 모델 평가 결과는 다음과 같음

- 아래와 같이 모델1과 모델4의 성능이 제일 좋은 것을 볼 수 있음

모델	종속 변수	독립 변수	정확도
모델1	포지션(Pos)	3점슛(3P), 블로킹(BLK)	100%
모델2		3점슛(3P), 리바운드(TRB)	95%
모델3		블로킹(BLK), 리바운드(TRB)	85%
모델4		3점슛(3P), 블로킹(BLK), 리바운드(TRB)	100%