Machine Learning 3rd Term project

Clustering – Cluster NBA Player's position

2021741062 김현준

데이터 선정

군집화는 비 지도학습을 사용하는 알고리즘으로, 지도학습을 사용하는 분류, 회귀 알고리즘과는 데이터의 차이가 있고, 학습을 하는 과정이 살짝 다르다고 볼 수 있음.

군집화는 Target 정답 없이 데이터들의 양상을 보고 비슷한 양상을 띄는 데이터를 묶는 과정으로, 분류 회귀 알고리즘과 다르게 많은 수의 행은 오히려 독이 될 수 있다고 생각하여, 적당한 크기를 가지고 있는 데이터를 찾고자 하였음.

비 지도학습은 Target 레이블이 데이터에 없어도 상관없이 학습을 진행할 수 있지만, 이번 학습에서는 데이터의 한 라벨을 Target 레이블로 선정하고, 군집화를 하였을 때 원하는 대로 잘 군집화가 이루어졌는지 확인해보고자 함.

이번에 사용하는 데이터 셋은 2023년 NBA 선수들의 개인 스텟을 정리한 데이터로, 포지션에 따라 선수들의 개인 스텟이 다를 것이라고 예상, 포지션에 따른 군집화가 될 수 있을지 기대하며 데이터를 선정하였음. 여기서 말하는 스텟은 골 성공률, 시도 횟수 등과 같은 지표를 의미함.

NBA Players stats(2023 season)

NBA Players stats in 2023

데이터 전처리

앞에서도 말했지만 군집화는 비 지도학습으로 Target 데이터가 학습에 사용되지 않으며 필요도 없음. 하지만 이번 프로젝트의 목적은 과연 선수들의 개인 스텟만으로 이 선수가 어떤 포지션에서 뛰는지 분류해낼 수 있을까? 에 대한 궁금증에서 시작되었기 때문에 선수들의 포지션을 Target 으로 바꾸어 마지막 군집화를 진행 후 비교를 해볼 생각임.

축구는 좋아하지만 농구는 잘 알지 못해서 포지션에 대한 공부부터 했어야 했는데, 일반적으로 알고 있던 농구의 5 개 포지션 센터, 스몰 포워드, 파워 포워드, 슈팅 가드, 포인트 가드와는 다르게 포지션의 분류를 보니 C, SF, PF, SG, PG 와 F G 가 따로 또 분류가 되어있음을 확인할 수 있었음.

```
pos_counts = df['POS'].value_counts()
print(pos_counts)
```

```
POS
SG    96
C     78
SF    77
PG    77
PF    74
F     66
G     66
Name: count, dtype: int64
```

이를 위해 추가 조사를 진행하였는데 SF, PF 를 F 로 묶어서 표현하는 경우는 있어도 F 를 따로 빼둔 경우는 없었음. 이에 각 POS 별로 평균적인 수치를 비교하여 같은 포지션에서 뛰는 선수들이 맞는지 확인하는 과정을 거쳤음.

사용하고 있는 데이터에 제공되고 있는 열은 총 30 개로

선수의 이름, 포지션, 팀, 나이, 플레이한 게임 수, 승리 게임 수, 패한 게임 수, 이번 시즌에 선수가 플레이한 시간(분), 플레이어가 얻은 포인트, 필드골의 횟수, 필드 골 성공 비율, 자유투의 횟수, 성공 자유투 비율, 총 공격 리바운드 수, 총 수비 리바운드 수, 총 어시스트 수, 총 턴 오버 수, 도루 횟수, 블록 횟수, 파울의 수, nba 판타지 포인트 수, 더블 더블 성공 횟수, 트리플 더블 성공 횟수 가 있음.

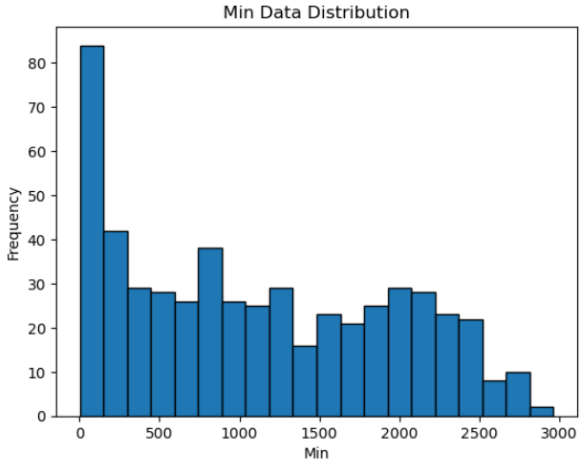
이때 각 포지션 별 스탯을 비교하기 위해 중요한 데이터는 지금의 데이터에서 nba 판타지 포인트 수, 더블 더블 성공 횟수, 트리플 더블 성공 횟수 같은 데이터를 빼야하며, 선수들마다 경기한 경기의 시간이 다르기 때문에 분당 횟수로 바꾸어 수치 비교가 의미 있는 결과를 낼 수 있도록 하려고 함. 또한 선수들의 플레이한 게임 수, 승리 게임 수, 패한 게임의 수는 선수들의 개인 스텟에는 유의미한 결과지만 포지션별 군집화를 하는 현 상황에서는 의미 없는 데이터이기 때문에 과감하게 삭제하였음.

```
columns_to_drop = ['PName','Team','GP','W','L','FP','DD2','TD3','+/-']
df = df.drop(columns=columns_to_drop)
df
```

	POS	Age	Min	PTS	FGM	FGA	FG%	3PM	3PA	3P%	...	FTA	FT%	OREB	DREB	REB	AST	TOV	STL	BLI
0	SF	25	2732.2	2225	727	1559	46.6	240	686	35.0	...	622	85.4	78	571	649	342	213	78	5
1	C	29	2284.1	2183	728	1328	54.8	66	200	33.0	...	771	85.7	113	557	670	274	226	66	11
2	PG	24	2390.5	2138	719	1449	49.6	185	541	34.2	...	694	74.2	54	515	569	529	236	90	3
3	PG	24	2416.0	2135	704	1381	51.0	58	168	34.5	...	739	90.5	59	270	329	371	192	112	6
4	PF	28	2023.6	1959	707	1278	55.3	47	171	27.5	...	772	64.5	137	605	742	359	246	52	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
529	F	24	6.2	3	1	2	50.0	1	1	100.0	...	0	0.0	0	0	0	1	0	1	
530	G	19	8.0	3	1	4	25.0	1	4	25.0	...	0	0.0	0	2	2	0	0	0	
531	F	27	7.9	2	1	2	50.0	0	0	0.0	...	2	0.0	2	0	2	1	2	0	
532	F	26	3.0	2	1	1	100.0	0	0	0.0	...	0	0.0	0	0	0	0	1	0	
533	G	24	2.1	2	0	1	0.0	0	1	0.0	...	2	100.0	0	0	0	0	0	1	

534 rows × 21 columns

이렇게 제거한 데이터를 사용하려고 했으나 Min 데이터를 보면 알 수 있듯이 행 529~533 의 Min 데이터는 초반 행들의 데이터에 비하면 너무 적다는 것을 알 수가 있음. 이 상태에서 Min 데이터로 총 횟수들을 나누어 분당 데이터를 그냥 구해버리게 된다면 1 분만 뛰고 1 골 넣은 선수가 2200 분을 뛰고 1500 골을 넣은 선수보다 잘 한다는 이상한 데이터가 되어버리기 때문에, 너무 적게 뛴 선수는 안타깝지만 배제를 해야 했음. Df.describe 를 통해 Min 데이터의 평균값과 분포도를 알 수 있었고 히스토그램으로 나타낸 Mindata 의 분포는 다음과 같음.



생각보다 균등하게 분배가 되어있었고, 최소 100 분은 뛰어야 유의미한 결과가 날 것이라고 생각하여 뛴 시간이 100 분 미만인 선수들은 삭제하였음. 삭제한 후에는 468 개의 행을 가진 데이터로 필터링 되었음.

이제 분당 데이터로 데이터를 수정해야 하는데, 선수가 달성한 필드 골, 성공시킨 3 점 필드골의 총 개수, 자유투 총 횟수와 성공률과 같은 데이터들 또한 포지션별 영향도 있으나 전반적으로 선수의 개인 기량에 따른 데이터이기 때문에, 이 부분들도 다 제거한 후에 분당 데이터로 데이터를 업데이트 하였음.

```
columns_to_drop = ['PTS','FGM','3PM','FTM','FTA','FT%']
df_filtered = df_filtered.drop(columns=columns_to_drop)
df_filtered
```

	POS	Age	Min	FGA	FG%	3PA	3P%	OREB	DREB	REB	AST	TOV	STL	BLK	PF
0	SF	25	2732.2	1559	46.6	686	35.0	78	571	649	342	213	78	51	160
1	C	29	2284.1	1328	54.8	200	33.0	113	557	670	274	226	66	112	205
2	PG	24	2390.5	1449	49.6	541	34.2	54	515	569	529	236	90	33	166
3	PG	24	2416.0	1381	51.0	168	34.5	59	270	329	371	192	112	65	192
4	PF	28	2023.6	1278	55.3	171	27.5	137	605	742	359	246	52	51	197
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
470	G	22	104.6	40	30.0	28	14.3	4	11	15	4	5	6	2	12
475	G	29	172.4	37	24.3	23	34.8	0	15	15	23	7	5	0	17
477	PF	27	171.6	24	45.8	4	25.0	18	30	48	7	11	2	8	35
489	SF	28	107.4	27	25.9	20	25.0	2	15	17	9	2	4	4	7
494	F	39	112.6	15	46.7	9	11.1	3	14	17	19	9	4	3	11

468 rows × 15 columns

그리고 분당 데이터로 환산하기 위해 Min 의 값으로 FGA, 3PA, ORED, DRED, REB, AST, TOV, STL, BLK, PF 값을 나누었음. 나눈 다음에 Age, Min 의 값 또한 삭제하였음.

```
columns_to_divide = ['FGA', '3PA', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'PF']

for column in columns_to_divide:
    df_filtered[column] = df_filtered[column] / df_filtered['Min']

df_filtered
```

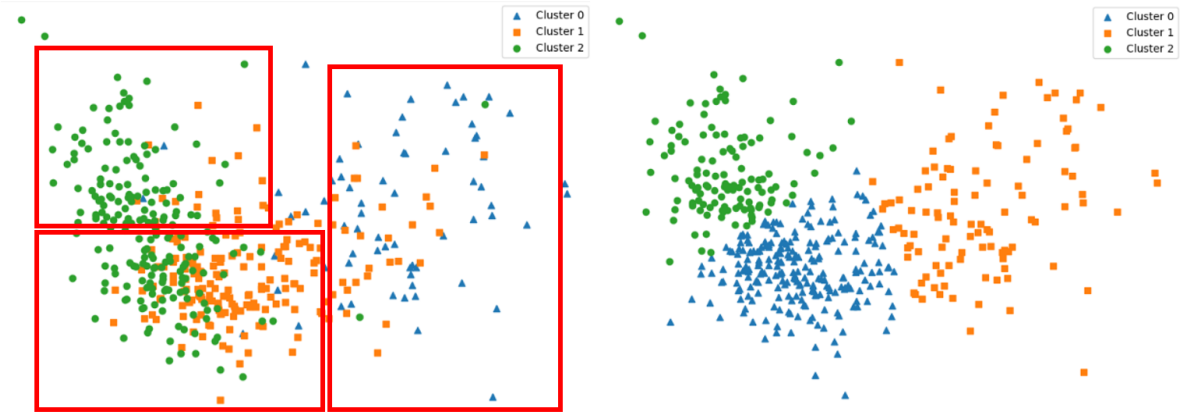
	POS	Age	Min	FGA	FG%	3PA	3P%	OREB	DREB	REB	AST	TOV	STL	BLK
0	SF	25	2732.2	0.570602	46.6	0.251080	35.0	0.028548	0.208989	0.237538	0.125174	0.077959	0.028548	0.018666
1	C	29	2284.1	0.581411	54.8	0.087562	33.0	0.049472	0.243860	0.293332	0.119960	0.098945	0.028895	0.049035
2	PG	24	2390.5	0.606149	49.6	0.226312	34.2	0.022589	0.215436	0.238026	0.221293	0.098724	0.037649	0.013805
3	PG	24	2416.0	0.571606	51.0	0.069536	34.5	0.024421	0.111755	0.136175	0.153560	0.079470	0.046358	0.026904
4	PF	28	2023.6	0.631548	55.3	0.084503	27.5	0.067701	0.298972	0.366673	0.177407	0.121566	0.025697	0.025203
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
470	G	22	104.6	0.382409	30.0	0.267686	14.3	0.038241	0.105163	0.143403	0.038241	0.047801	0.057361	0.019120
475	G	29	172.4	0.214617	24.3	0.133411	34.8	0.000000	0.087007	0.087007	0.133411	0.040603	0.029002	0.000000
477	PF	27	171.6	0.139860	45.8	0.023310	25.0	0.104895	0.174825	0.279720	0.040793	0.064103	0.011655	0.046620
489	SF	28	107.4	0.251397	25.9	0.186220	25.0	0.018622	0.139665	0.158287	0.083799	0.018622	0.037244	0.037244
494	F	39	112.6	0.133215	46.7	0.079929	11.1	0.026643	0.124334	0.150977	0.168739	0.079929	0.035524	0.026643

이후 POS 에 따라 값을 조정하였는데, 현재 POS 는 총 7 가지의 Target 을 가지고 있기 때문에 이대로 진행할 시, 7 개로 군집화를 진행해야 포지션별 비교를 할 수 있음. 7 개의 군집화는 현실적으로 많이 어렵기 때문에 다양한 케이스로 데이터들을 묶고 테스트를 진행하며 제일 군집화가 잘 되는 케이스를 찾아보고자 하였음.

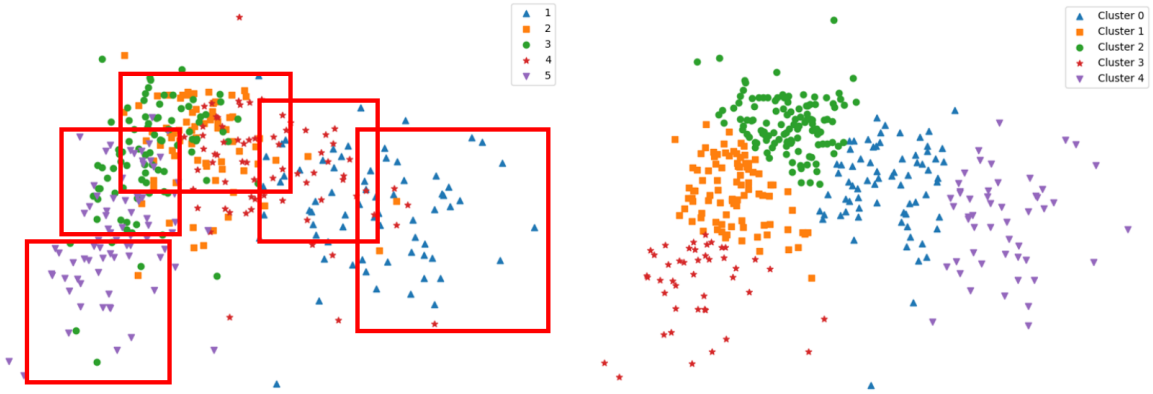
1. C = 0, SF, PF, F = 1, SG, PG, G = 2
2. C = 0, SF = 1, SG = 2, PF = 3, PG = 4
3. C = 0, F = 1, G = 2

첫번째 케이스의 경우 F 가 원래는 있는 포지션이 아니긴 하지만 SF, PF 둘 다 가능한 선수를 F 로 칭한다는 정보를 얻어 큰 포워드와, 큰 가드로 데이터를 정리하고자 하였음. 하지만 LDA 와 K – Means 를 진행한 결과 군집화가 적당히 되고 있지 않으며, 각 클래스 별로 겹쳐진 부분도 있어 좋지 않은 전처리가 되었음을 알 수 있었음.

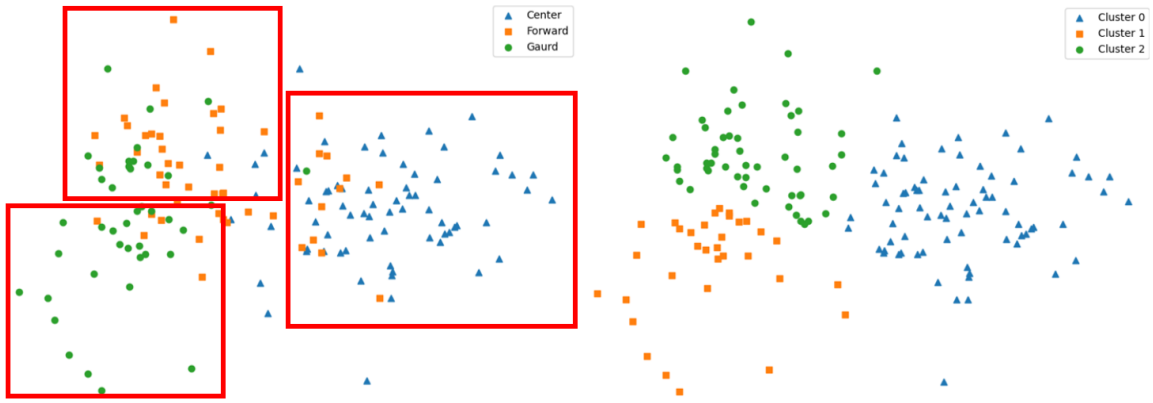
아래의 그래프가 1 번 케이스로 전처리를 진행한 부분이며 왼쪽이 LDA, 오른쪽이 군집을 3 개로 제한하고 학습시킨 K – Means 로 군집화를 진행한 부분임. Target 데이터가 없었다면 군집화가 굉장히 잘 되었다고 생각할 수 있었으나, 원래의 Data 와 비교해본 결과 올바르게 분류되지 않았음을 파악할 수 있음.



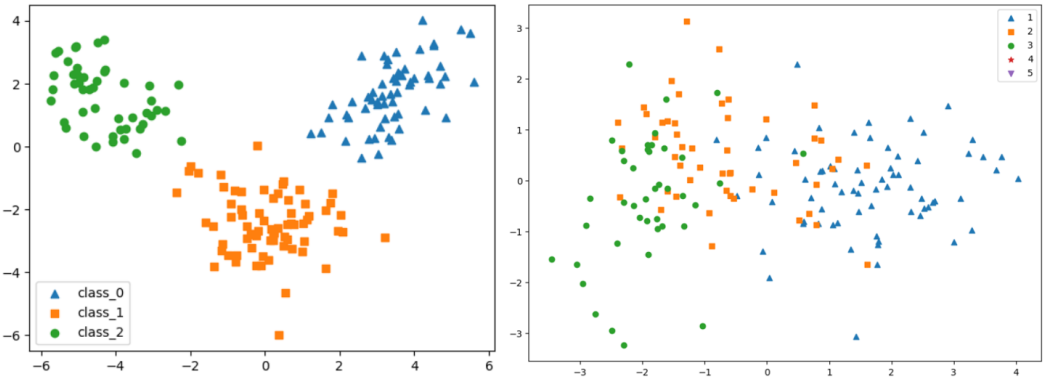
두번째 케이스의 경우에는 총 분류하는 군집의 개수가 5 개로 설정되었으며 LDA, K- means 결과는 다음과 같음. 이 경우에도 정상적으로 군집화가 진행되고 있지 않음을 알 수 있음.



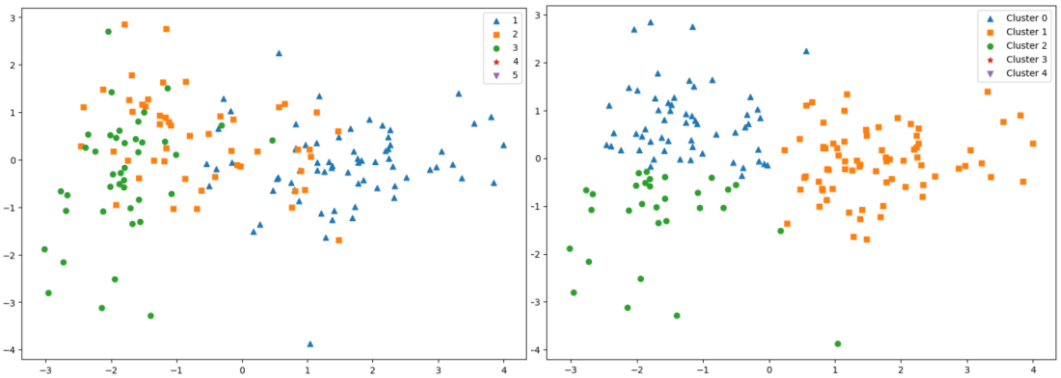
세번째 케이스의 경우 인터넷 서칭을 통해 F, G 라고만 적혀 있는 선수들은 SF, PF 와 SG, PG 포지션을 둘 다 수행할 수 있는 선수들이라는 것을 알고는 C, F, G 로만 센터 포워드 가드로 생각하면 더 군집화를 잘 수행할 수 있지 않을까? 하는 생각으로 수행해본 결과임. 3 개의 case 중에서는 가장 LDA 와 K – Means 의 차이가 크지 않음을 확인할 수 있음.



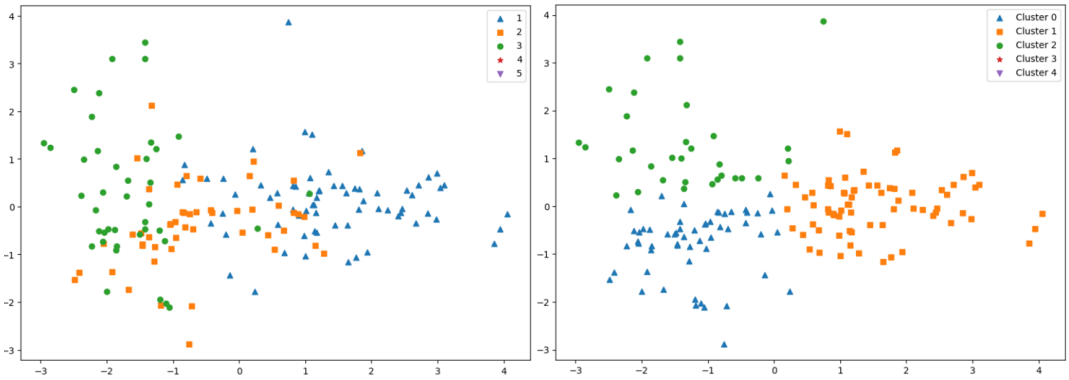
하지만 현재 LDA 로 전 처리된 데이터를 확인하면 LDA 로 전처리를 했음에도 불구하고 클래스간 거리가 너무 가깝다는 것을 확인할 수 있음. 아래의 두 그래프 중 왼쪽은 예전에 HW#2 로 진행한 Wine 데이터에 대해 LDA 를 수행한 결과임. 두개의 그래프를 비교하면 확실히 현재 LDA 그래프가 거리가 많이 떨어져 있지 않음을 확인할 수 있음.



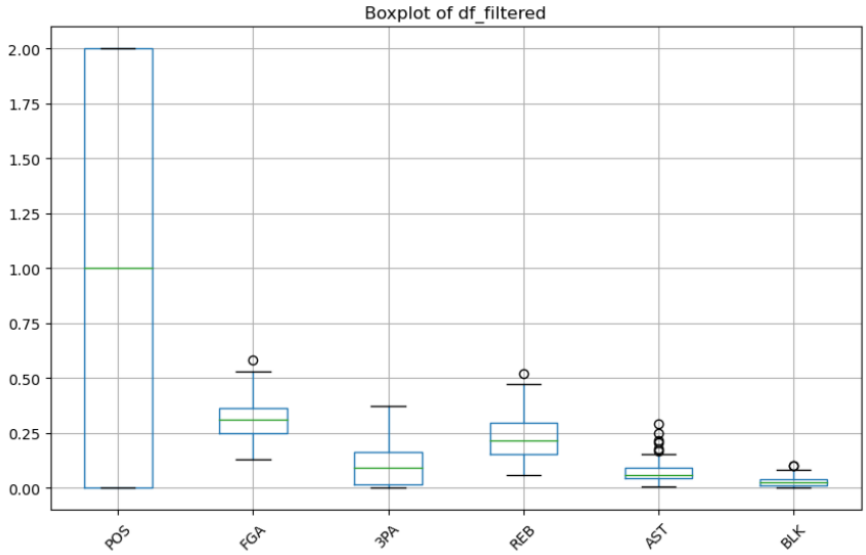
기존의 데이터에서 FG%, 3P%, OREB, DREB 4 개의 클래스를 추가로 삭제한 후 LDA 를 진행한 결과임. 기존의 데이터보다 LDA 가 더 잘 진행됨을 확인할 수 있음.



열의 개수가 그동안 너무 많았기에 차원 축소를 진행할 경우 너무 많은 데이터 손실이 나며 다음과 같은 문제가 발생했다고 생각하여, 부가적인 스텟을 전부 삭제해 보았음. 삭제한 데이터는 FGA, 턴 오버 수와 플레이어의 도루 횟수임. 처음에 비해 겹치는 부분이 적어지며 군집화가 제일 잘 되고 있다고 판단할 수 있기에, 다음과 같이 전처리한 데이터를 최종 군집화에 사용하였음



LDA 를 사용한 결과를 보고 이상치가 너무 많다고 생각하여 이상치 제거도 시도해보았으나, 막상 Box Plot 을 그린 후 이상치를 제거하였을 때 유의미한 결과는 없었기에 최종 전처리에서 사용하지는 않았음.

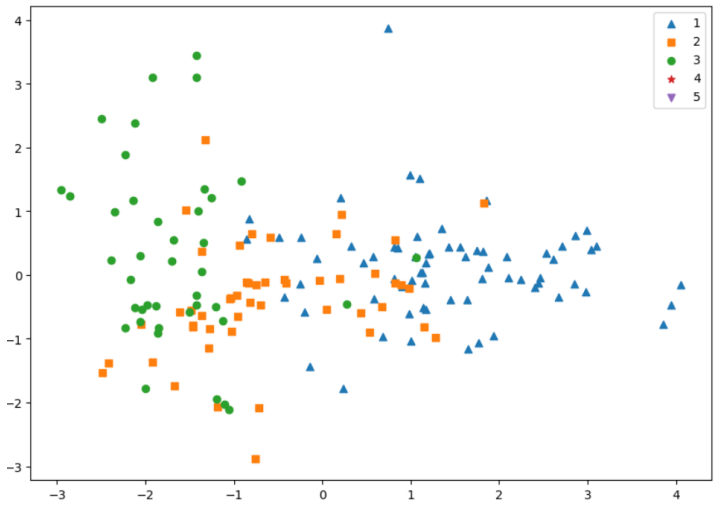


Box plot 을 사용한 결과 다음과 같이 어시스트 부분에서 이상치가 좀 검출되었으나, 실제로 이상치를 체크하여 제거했을 때는 10 개 미만의 값이 검출되어 큰 영향을 끼치지는 못하였음.

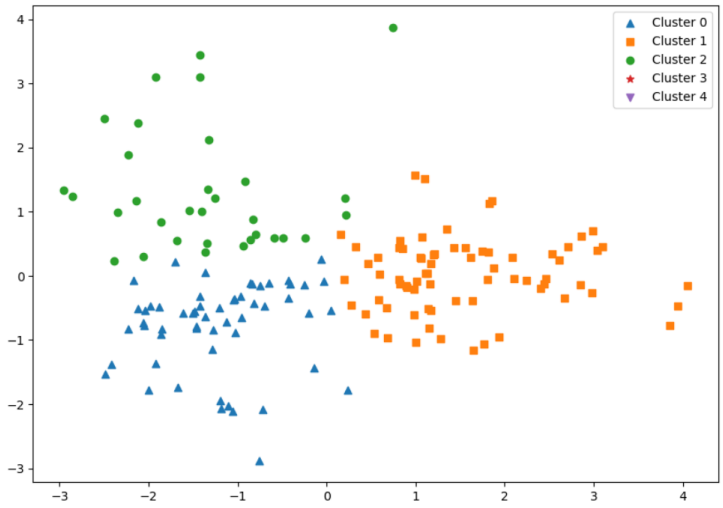
군집화 결과

사용한 데이터 전처리 후 LDA

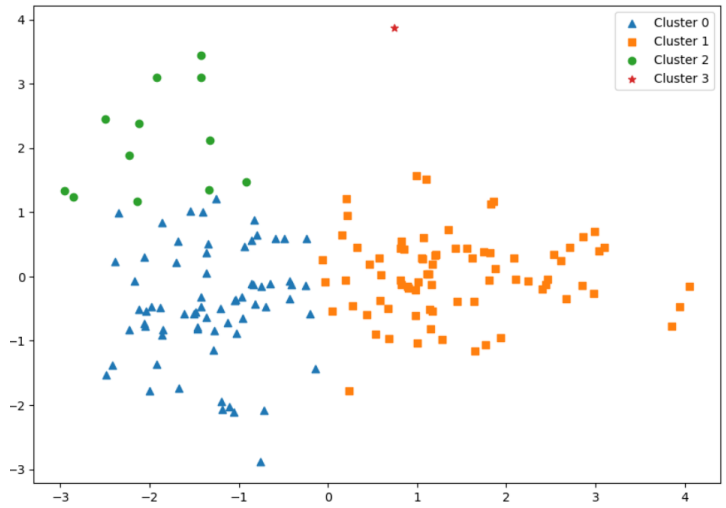
Target 은 포지션 3 개로 Center, Guard, Forward 로 구성하였음. 파란색 세모가 Center, 주황색 네모가 Forward, 초록색 동그라미가 Guard 임.



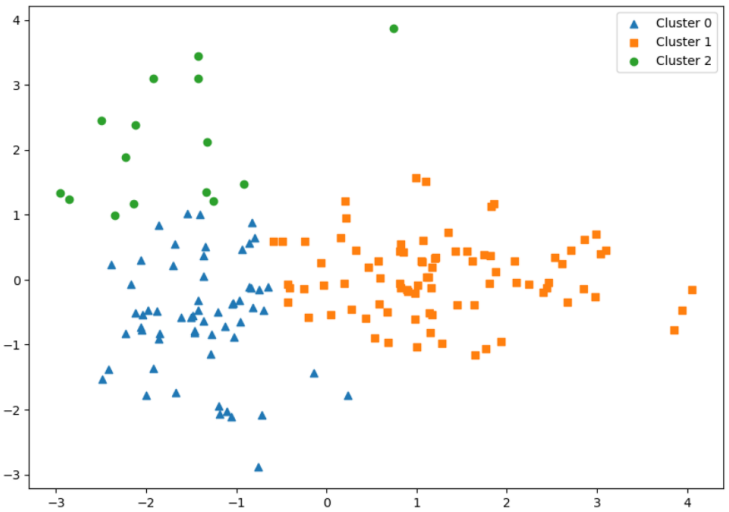
K – Means (clusters = 3)



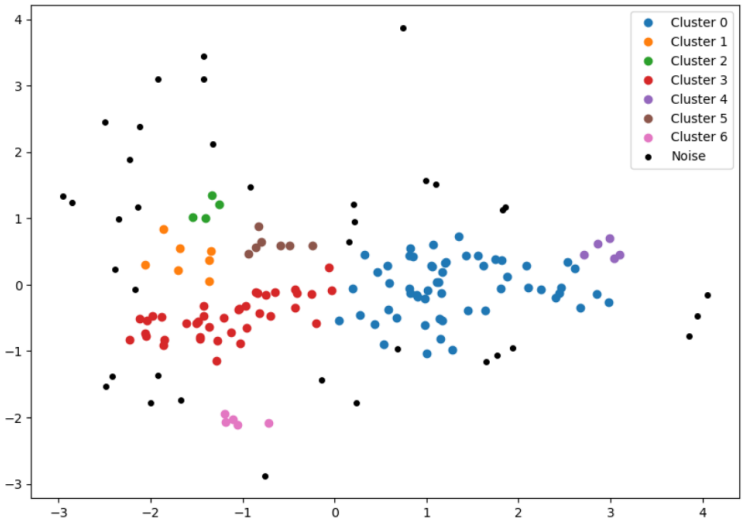
Mean – Shift



GMM



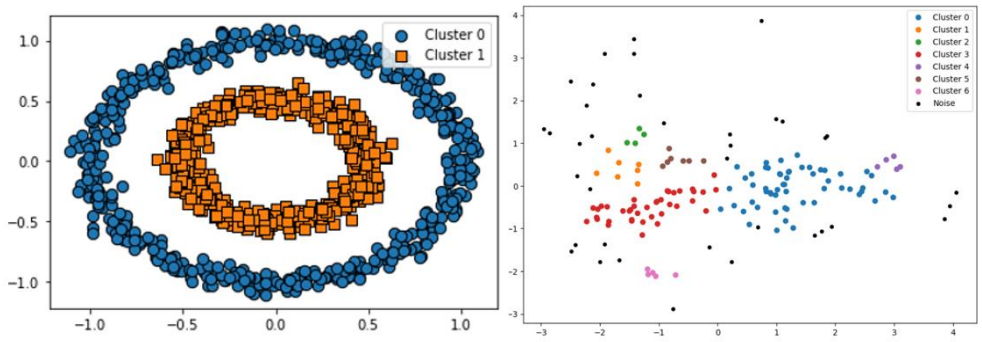
DBSCAN



군집화 결과 분석 및 고찰

다양한 알고리즘을 사용해서 군집화를 수행한 결과 K – Means와 Mean – Shift, GMM은 거의 비슷한 양상을 보였으나, DBSCAN의 경우에는 원하는 것처럼 군집화가 잘 되고 있지 않음을 확인할 수 있음. 이는 DBSCAN의 Clustering 결과를 보면 3개로 뭉쳐지기 않고 노이즈 데이터가 많이 있음. 다른 알고리즘들 또한 LDA 상에서 겹친 부분은 어쩔 수 없이 오차가 발생하였음.

DBSCAN의 경우 epsilon과 min_samples 파라미터를 사용하는데 epsilon의 경우 클러스터링을 수행할 때 고려하는 최대 거리, min_samples의 경우 클러스터를 형성하기 위해 필요한 최소 포인트 수임. 현재 **LDA 처리가 진행된 데이터가 서로 너무 가깝게 분포하고 있기에 다음과 같은 문제가 발생했다고 생각**함. 수치적으로 확인해보기 위해 다양한 알고리즘 중 K – Means 알고리즘에 **실루엣 군집 평가를 진행**한 결과 총 실루엣 점수는 0.43, 각 요소별 실루엣 계산은 cluster 0이 0.43, 1이 0.52, 2가 0.25로 평가되었음. 이 말은 즉 현재 군집화가 중간 정도의 품질을 가지고 있으며, 완전히 구분된 것은 아니지만 어느정도 구분이 가능한 상태라는 것을 의미함.



전 처리 과정에서 PCA가 아닌 LDA를 사용해서 전처리를 한 이유는, PCA와 다르게 LDA는 클래스 간 분산을 최대화하고 클래스 안에서의 분산을 최소화하기에 클래스 별로 거리를 멀리 띄울 수 있다는 장점이 있기 때문이었음. 하지만 스텟 별로 큰 차이가 나지 않았기에 퍼지지 않은 데이터

가 만들어짐.

축구에서는 골을 넣는 선수, 골을 막는 선수, 중간에서 볼을 배급하는 선수가 명확하게 공격수 미드필더 수비수로 나뉘어져 있어 선수들 개개인의 스텟을 보면 이 사람이 공격수로 활동하는지, 수비수로 활동하는지를 인지할 수 있음. (보통 축구에서 골은 공격수가, 수비 리커버리는 수비수가 진행하기 때문) **하지만 농구의 경우 명확하게 골을 넣는 사람과 골을 수비하는 사람이 정해져 있지 않고 5명이 같이 움직이기 때문에 선수들의 스텟을 비교하였을 때 큰 차이가 나지 않음**.

POS	3PM	3PA	3P%	OREB	DREB	REB	AST
C	19.653846	55.782051	23.937179	96.153846	218.717949	314.871795	74.320513
F	26.257576	77.242424	31.468182	36.045455	89.484848	125.530303	41.075758
G	20.545455	59.666667	29.506061	11.803030	41.000000	52.803030	35.030303
PF	52.540541	150.918919	32.228378	74.432432	222.756757	297.189189	110.689189
PG	85.350649	236.012987	34.022078	29.571429	143.961039	173.532468	251.935065
SF	77.688312	211.636364	34.463636	48.714286	173.753247	222.467532	116.363636
SG	96.822917	261.833333	35.888542	36.156250	151.697917	187.854167	155.270833

Min	PTS	FGM	FGA	FG%	TOV	STL
1017.233333	482.756410	189.512821	328.192308	57.402564	59.025641	24.615385
594.909091	245.000000	93.181818	196.363636	46.356061	28.727273	17.666667
395.472727	153.818182	56.227273	133.000000	40.887879	18.469697	12.409091
1319.018919	596.797297	221.121622	444.459459	48.405405	70.445946	35.594595
1407.524675	730.792208	260.402597	577.558442	43.666234	96.597403	47.558442
1395.584416	653.844156	235.350649	506.415584	45.676623	64.584416	41.051948
1423.453125	701.770833	250.843750	557.479167	44.504167	80.166667	47.583333

3점슛과 2점슛의 성공 비율은 포지션별로 큰 차이가 나지 않았으며 나머지 스텟들의 경우에도 눈으로 보기에는 큰 차이가 나는 것처럼 보이지만, Min으로 나누어 분당 스텟으로 전환할 경우에는 큰 차이가 나지 않음을 확인할 수 있었음. 시간으로 나누었을 경우 너무 값이 작게 나와 눈으로 확인할 때는 100을 곱하여 확인하였는데, 100을 곱한 값을 보아도 큰 차이가 나지 않음을 확인할 수 있음.

```
pos_averages = df_filtered.groupby('POS').mean()

# 결과를 출력합니다.
print(pos_averages * 100)
```

POS	FGA	3PA	REB	AST	BLK
0	30.383859	5.285746	30.537640	6.515958	4.222733
1	31.905319	13.802348	19.541928	6.356073	2.156326
2	31.832870	13.791237	14.363619	9.418144	1.136690

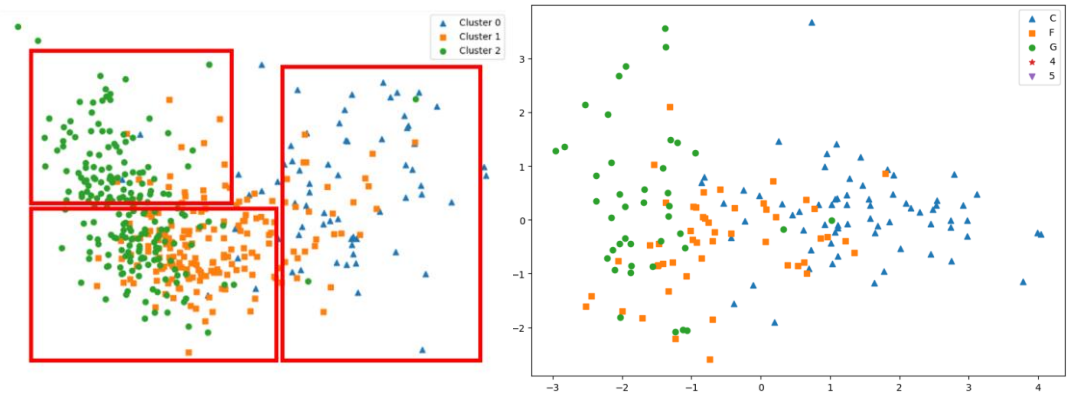
```
pos_averages = df_filtered.groupby('POS').mean()

# 결과를 출력합니다.
print(pos_averages*100)
```

POS	3PA	OREB	DREB	AST	BLK
0	5.285746	9.923207	20.614433	6.515958	4.222733
1	13.802348	5.734917	13.807011	6.356073	2.156326
2	13.791237	3.268785	11.094833	9.418144	1.136690

위의 데이터 중 C, F, G 만을 사용하여 데이터 전처리를 진행하고 라벨별로 각 열의 평균을 구해보았을 때, 리커버리 횟수 총합인 REB 말고 수비, 공격 리버커리 횟수인 OREB, DREB를 사용해도 큰 차이가 나지 않음을 확인할 수 있었음. (아래 사진은 OREB와 DREB를 사용한 후 LDA를 한 결과). 위에서는 추가 설명을 하지 않았지만, 전 처리 과정 중 FG%와 3P% 또한 선수의 기량을 나

타내는 수치라고 생각하였기에 drop하여 사용하였음. **이번 분석은 각 포지션별**로 슈팅을 시도하는 횟수나 리버커리 수, 어시스트 수의 차이가 있으며, 이 **차이를 바탕으로 포지션 군집이 가능할까**? 라는 궁금증에서 시작하였기 때문에 각 지표들 중 순수하게 선수들의 실력을 나타내는 지표는 과감하게 삭제하여 선수가 시도한 골, 리바운드 등 **성공보다는 시도한 스텟을 더 집중적으로 사용**하고자 하였음.

실제로 모든 열을 다 포함했을 때보다 특정 열만을 사용해서 전 처리를 진행했을 때 더 좋은 결과를 얻을 수 있었음.



하지만 이렇게 다양한 시도를 통해 전 처리 성능을 개선하고 LDA를 사용하였을 때 클래스 별로 많이 떨어지게 하여 원활한 군집화를 하고자 하였지만 각 Target 들의 데이터가 큰 차이가 나지 않았기 때문에 군집화를 진행 시 섞이는 부분들이 있었음.

```
pos_averages = df_filtered.groupby('POS').mean()

# 결과를 출력합니다.
print(pos_averages * 100)
```

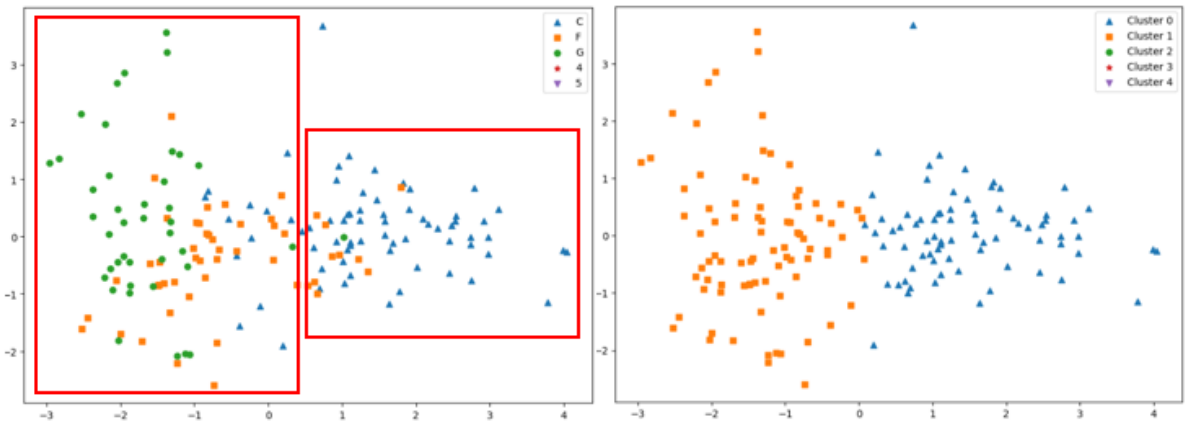
```
            FGA         3PA         REB         AST         BLK
POS
0     30.383859    5.285746   30.537640   6.515958   4.222733
1     31.905319   13.802348   19.541928   6.356073   2.156326
2     31.832870   13.791237   14.363619   9.418144   1.136690
```

각 포지션 별로 데이터를 다시 보면 센터 포지션과 가드, 포워드 포지션의 스텟 차이는 유의미하지만, 가드와 포워드 사이에는 그리 큰 다름이 없는 것을 확인해볼 수 있음.

이에 추가적인 조사를 진행한 결과 **센터의 경우**에는 전통적으로 바스켓 근처에서 수비를 담당하는 역할을 맡기 때문에 **다른 포지션들에 비해 3점 슛과 블로킹과 리바운드에 많이 관여함**. 하지만 **가드와 포워드 포지션의 선수**들은 득점 및 어시스트와 같은 분야에서 비슷하게 움직이기 때문에 **유사한 통계를 보임**. 가드는 볼 핸들링과 외곽 플레이를 담당하기에 어시스트와 3점 슛을 담당하며 포워드는 가드와 센터 사이에서 득점, 리바운드, 어시스트에 기여함.

그러하기에 스텟을 보고 가드와 포워드를 분류하기에는 굉장히 어려우며, 지금 상황과 같이 전처리를 위해 많은 열을 제거한 상황에서는 그 분류가 더 어려운 상황이라고 말할 수 있음. 만약 센

터와 센터가 아닌 포지션만을 분류하고자 했다면 다음과 같이 굉장히 잘 분류가 되는 모습을 확인할 수 있음. 또한 현대 농구는 기존의 농구처럼 기존에 정해진 룰, 전통적인 역할이 점점 흐려지고 있기 때문에 스텟만 보고 선수들의 포지션을 분류하기가 애매하며, 센터랑 포워드를 같이 수행하는 선수들도 있기에 분류가 더욱 힘들 수밖에 없다는 것을 알았음.



농구가 아닌 야구나 축구 선수들의 데이터를 사용했더라면 더욱 좋은 군집화를 할 수 있었을 것이라고 생각하여 아쉬움이 남음. 완벽하지는 않았으나 포지션, 가드, 센터로 포지션을 분류할 수 있었으며 이를 다른 종목에 적용할 경우 더 세분화된 포지션 분류를 할 수 있을 것이라 생각함.