**주제: 메이저리그 선수 데이터를 활용한 포지션 예측**

1. 학습 목적

메이저리그 내야수 능력치를 학습하여 마이너리그 내야수의 포지션을 예측 모델을 생성한다.

1. 데이터 수집/선별
2. selenium, BeautifulSoup을 이용하여 메이저리그 홈페이지와 마이너리그 홈페이지에서 모든 선수의 **데이터를 크롤링** 해온다.
3. 데이터의 기준은 메이저리그 선수 stats 데이터가 더 많은 feature를 가지고 있으나 예측할 때 사용할 마이너리그 선수 stats 데이터의 feature를 기준으로 한다. (마이너리그는 수비 stats 데이터가 없음)
4. 메이저리그 공격 stats 데이터 url

<http://mlb.mlb.com/stats/sortable.jsp?c_id=mlb#elem=%5Bobject+Object%5D&tab_level=child&click_text=Sortable+Player+hitting&game_type='R'&season=2019&season_type=ANY&league_code='MLB'&sectionType=sp&statType=hitting&page=1&ts=1580532864437&playerType=ALL>

1. 메이저리그 수비 stats 데이터 url

["http://mlb.mlb.com/stats/sortable.jsp?c\_id=mlb#elem=%5Bobject+Object%5D&tab\_level=child&click\_text=Sortable+Player+fielding&game\_type='R'&season=2019&season\_type=ANY&league\_code='MLB'&sectionType=sp&statType=fielding&page=1&ts=1580539033522&playerType=ALL"]](%22http://mlb.mlb.com/stats/sortable.jsp?c_id=mlb#elem=%5Bobject+Object%5D&tab_level=child&click_text=Sortable+Player+fielding&game_type='R'&season=2019&season_type=ANY&league_code='MLB'&sectionType=sp&statType=fielding&page=1&ts=1580539033522&playerType=ALL"])

1. 마이너리그 공격 stats 데이터 url

<http://www.milb.com/milb/stats/stats.jsp?sid=milb&t=l_bat&lid=117>

1. 공격 데이터만 가지고 학습 후 예측을 하니 60% 이하의 예측을 보여 수비데이터를 포함하여 메이저리그 선수의 데이터만으로 학습하고 예측하기로 했다.
2. 메이저리그 선수 데이터를 “train\_test\_split”을 통해 80 : 20으로 나눠서 test 데이터를 통해 예측하도록 하겠다.
3. 데이터 전처리
4. 데이터를 다시 load할 때를 대비하여 excel파일로 1차 정제된 데이터를 백업
5. excel 파일을 panda로 read
6. 컬럼 이름 편집: 컬럼이름 간소화, 특수문자 제거
7. 불필요한 컬럼 삭제: 데이터 중에 포지션 예측에 불필요한 선발 게임 경기수, 랭킹, 빈 컬럼 등)
8. 공격 stats와 수비 stats 데이터가 분리되어 있으므로, 이름(player), 팀(team), 포지션(pos) 정보를 기준으로 병합한다. 이름만으로 하면 동명이인이 있을 수 있기 때문에 3개 컬럼이 일치하는 데이터로 통합한다.
9. POS에서 내야수만 추출 ("1B", "2B", "3B", "SS")
10. 일정 타석과 수비 이닝의 최소값을 정해 그 이하인 선수는 제거
11. 추가 불필요한 컬럼 삭제

* Player(이름), Team(팀), AB(타석수), INN(수비이닝) 정보 삭제

1. info()를 통해 데이터의 정보를 확인

* 데이터 타입, 결측치 등을 확인
* 타율, OBP 등은 데이터 타입이 object로 되어 있어 수치 데이터로 변경이 필요

1. 결측치 여부 확인

* isnull().count()를 통해 컬럼마다 결측치 여부를 확인

1. AVG, OBP, SLG, FPCT등 object type을 수치형으로 변경

* **데이터 전처리가 끝났음으로 데이터 분석을 해본다.**

1. 포지션 별로 각 수치 평균을 확인해본다. Pos로 groupby와 mean으로 확인

* 눈으로만 봐도 1루수는 타 내야수보다 일정 컬럼에서 구분이 된다.

1. 데이터를 describe()을 통해 평균, 분산, 최소/최대값, 4분위수값을 확인한다.

* 이상치를 대략 확인할 수 있으며 어떻게 분포되어 있는지 확인한다.

1. describe()를 통해 더 자세히(그래프) 분포를 보고 싶으면 seaborn을 통해 distplot을 사용하여 확인한다.

* 데이터의 치우침 정도를 확인할 수 있다.

1. feature간에 상관도 분석을 한다. corr matrix로 확인 가능

* cmap-"RdBu"
* 양의 상관관계가 높을 수록 - 진한 파란색
* 음의 상관관계가 높을 수록 - 진한 빨간색
* 상관도 분석을 통해서 정말 Pos 예측에 영향을 안 미치는 feature가 있다면 제거하는 것도 좋다.
* **데이터 레이블로 인코딩과 세트 분리**

1. Pos 컬럼은 object여서 LabelEncoder()를 통하여 레이블로 할당한다.
2. 이제 Pos을 Target 데이터셋으로 그 외는 입력변수 데이터셋으로 분리해준다.
3. train\_test\_split을 통해 train, test 데이터 셋으로 8:2 비율로 분리한다.
4. 모델 선정
5. Machine Learning 모델들, 스태킹 모델, Deep Learning 모델로 다양하게 학습해본다.
6. Machine Learning 모델: 주요 모델은 간단히 설명

* KNN 알고리즘
* Decision Tree 알고리즘
* Random Forest 알고리즘: 결정트리 기반으로 여러 개의 결정트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습 수행 뒤 최종적으로 모든 분류기가 보팅을 통해 예측 결정
* 아다부스타 알고리즘
* GBM 알고리즘
* XGBoost: GBM에 기반하여 느린수행시간 및 과적합규제 문제를 해결, 자체 내장된 교차검증 사용, 분류에서 뛰어난 예측 성능, 결손값 자체 처리
* Stacking: 개별적인 여러 알고리즘을 서로 결합해 예측 결과를 도출한다. 배깅, 부스팅과 공통점을 가지고 있으나 개별 알고리즘으로 예측한 데이터를 기반으로 다시 예측을 수행한다. 즉 개별 알고리즘의 예측 결과 데이터 세트를 최종적인 메타 데이터 세트로 만들어 별도의 ML 알고리즘으로 최종 학습을 수행하고 테스트 데이터를 기반으로 다시 최종 예측을 수행
* Deep Learning: Single Layer, Multy Layer에 Scaling, 가중치 초기값, drop out 등을 통해 다양한 모델을 구성하고 각각 학습해 본다.

1. 학습
2. 다양한 Machine Learning 모델을 생성한다.
3. Training Data 셋을 통해 각 모델을 학습한다.

knn\_clf.fit(data[0], data[1])

rf\_clf.fit(data[0], data[1])

dt\_clf.fit(data[0], data[1])

ada\_clf.fit(data[0], data[1])

gb\_clf.fit(data[0], data[1])

xgb\_clf.fit(data[0], data[1])

lgbm\_clf.fit(data[0], data[1])

1. 성능 평가 (객관적인 평가를 위해 random\_state=11로 고정)
2. 기본 ML 모델을 학습 예측

KNN 정확도: 0.6122

RF 정확도: 0.6531

DT 정확도: 0.6327

ADA 정확도: 0.5102

GB 정확도: 0.6735

**XGB 정확도: 0.7255**

LGBM 정확도: 0.6735

1. 상기 모델들을 Stacking 모델로 학습 예측

최종 모델의 예측 정확도: **0.7143**

1. Scaling 후 기본 ML 모델을 학습 예측

* MinMaxScaler, StandardScaler, 분산, 평균을 통한 정규화 방식으로 했는데
* MinMaxScaler로 정규화했을 때 가장 좋은 성능을 보임

KNN 정확도: 0.5510

RF 정확도: 0.6735

DT 정확도: 0.6122

ADA 정확도: 0.5102

GB 정확도: 0.6531

**XGB 정확도: 0.7755**

LGBM 정확도: 0.6939

1. 상기 모델들을 Stacking 모델로 학습 예측

최종 모델의 예측 정확도: **0.7551**

1. 교차검증을 통해 기본 ML 모델을 학습 예측

교차 검증 0 정확도: 0.7049

교차 검증 1 정확도: 0.6885

교차 검증 2 정확도: 0.5738

교차 검증 3 정확도: 0.5246

교차 검증 평균 정확도: **0.6230**

* + 정확도가 가장 좋았던 XGBoost교차 검증을 했을 때 정확도가 내려감
  + 아마 XGBoost 자체적으로 교차검증을 하기 때문일 것으로 예상됨.

1. GridSearchCV를 통해서 최고의 하이퍼파라미터 찾아본다.

각 모델들은 사람이 정해줘야 하는 하이퍼파라미터 값들이 있다.

하나하나 넣어서 확인할 수 있지만 GridSearchCV를 사용하면 몇 가지 하이퍼파라미터 중에 최고의 정확도를 예측하는 하이퍼파라미터 값들을 찾아 준다.

일단 XGBoost에 들어갈 하이퍼파라미터들 넣어서 예측해 본다.

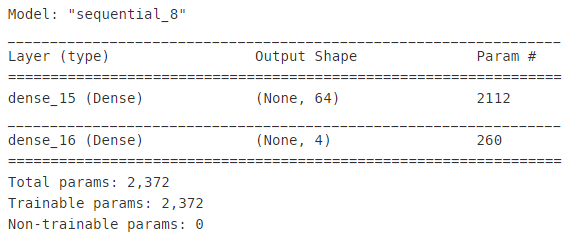
GridSearchCV 최고 평균 정확도 수치:**0.5795**

GridSearchCV 최적 하이퍼 파라미터: {'max\_depth': 8, 'min\_samples\_split': 2}

정확도가 낮은 이유는 하이퍼 파라미터를 2개만 적용했기 때문으로 예상된다.

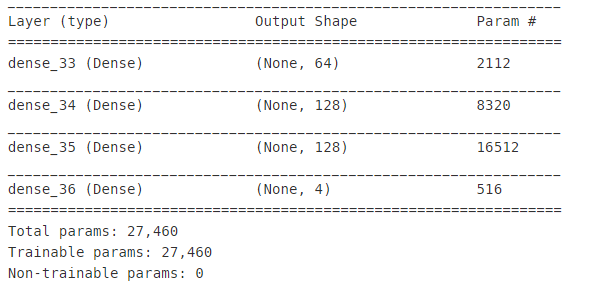
1. Deep Learning을 통해 예측해본다.

* 싱글 레이어



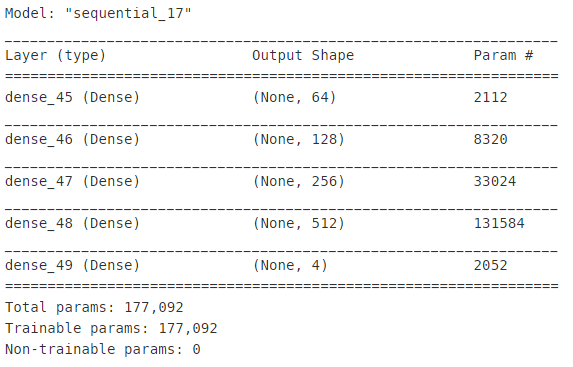
**Adam정확도:0.6531, PMSProp정확도:0.5918, SGD정확도0.5918**

* 멀티 레이어



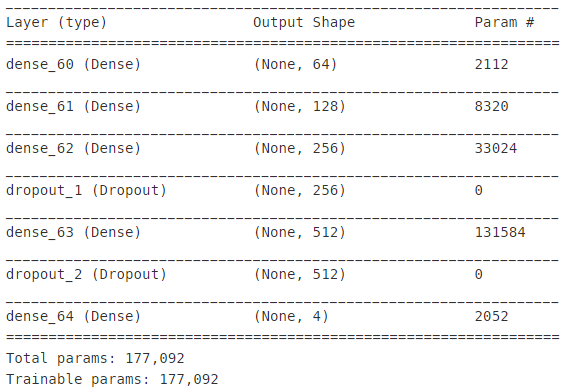
**Adam정확도:0.5102, PMSProp정확도:0.5714, SGD정확도0.6122**

* 더 깊은 멀티 레이어에 가중치 초기값(he) 사용하였다.



**Adam정확도:0.4694, PMSProp정확도:0.5306, SGD정확도0.4898**

* 더 깊은 멀티 레이어에 Dropout을 사용하였다.



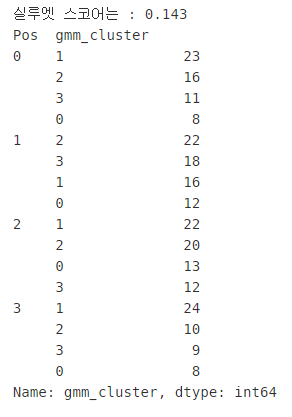
**Adam정확도:0.5918, PMSProp정확도:0.5714, SGD정확도0.5510**

1. 결과

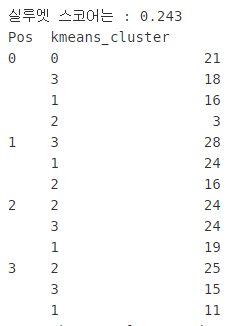
가장 높은 정확도를 보인 모델은 XGBoost모델로 random\_state=11로 고정하였을 때, 77%의 정확도를 보였다. 물론 random\_state을 랜덤하게 해주면 정확도가 계속 변하지만 평균적으로 돌려보았을 때도 XGBoost가 제일 높은 정확도를 보여줬다. 한가지 딥러닝 모델을 좀더 다양하게 구성해보면 정확도가 올라갈 것으로 예상되나 층을 깊게 하고 다양한 옵션을 적용해도 타 Machine Learning 모델보다 정확도가 떨어졌으며, 딥러닝이 만능이 아닌 것을 확인할 수 있었다. 좀더 아쉬웠던 부분은 메이저리그는 매년 정보가 있어서 2019년도 정보 244개로만 학습한 것이 아쉬웠고 과제를 제출하고 최근 10년 또는 과거 데이터를 포함하여 더 많은 데이터를 확보한 상태에서 학습을 시켜볼 예정이다.

마지막으로 Kmean과 GMM 클러스터링 방식으로 군집화를 진행해 보았으나

**GMM 결과**



KMean 결과



실루엣 스코어 : KMean 0.243으로 1에 가깝게 나와야 군집이 잘되는 것이나 잘 군집화가 되어지지 않았다.