|  |  |  |
| --- | --- | --- |
| Student ID: 201820736 | Name: 정민규 | Score: |

**Data Mining Project #1**

**Intoduction**

Riot사의 게임인 League of Legends(이하 LoL)은 많은 게이머들에게 사랑받고 있는 게임이다. LoL에는 기본적인 게임 룰을 가진 소환사의 협곡이라는 것이 있다. 이 게임은 167개의 챔피언(사용자가 선택할 수 있는 캐릭터)을 사용자가 골라 5 대 5 대전을 한다. 본 프로젝트의 목표는 두 가지이다.

1. 우리팀 4명의 챔피언과 적팀 5명의 챔피언이 정해졌을 때, 어떠한 챔피언을 고르는 것이 승률이 가장 좋을 것인가.
2. 사전 선택한 2명이 고를 챔피언들 중 어떠한 조합이 가장 승률이 좋은가.

LoL을 가끔 즐기는 사용자들은 휙휙 변하는 게임의 양상에 적응하기 힘들다. 따라서 추천해주는 챔피언을 위주로 이들이 게임을 다시 시작한다면 게임에 대한 낯선 이미지가 덜해질 것이라 판단했다. 또한 LoL은 10년 이상 플레이 되어온 게임으로 신규 유저의 진입장벽이 다소 높은 편이다. 본 프로젝트는 이를 함께 해결해 줄 수 있을 것이라 생각한다.

**Problem Defintion**

상위랭커들의 최근 플레이한 데이터를 op.gg에서 직접 크롤링하여 수집한다. 한 게임에 대한 데이터는 [#, user\_ID, user\_ID\_tag, result\_value, game\_time, our\_team, our\_team\_object, our\_team\_total\_kill, our\_team\_total\_gold, enemy\_team, enemy\_team\_object, enemy\_team\_total\_kill, enemy\_team\_total\_gold, game\_players]로 정의한다.

* #: row를 구분하기 위한 번호이다.
* user\_ID, user\_ID\_tag: 누구의 게임을 가져왔는지 구분하기 위함이다.
* result\_value: 가져온 게임 정보의 유저가 이겼다면 1, 졌다면 0이다.
* game\_time: 한 게임에 소요된 시간을 말한다.
* our\_team: [lane, champion, spell 1, spell 2, kill, death, assist, total damage, total taken damage, wards, cs] \* 5로 구성되어 있다.
  + lane: 게임에 사용되는 플레이어의 포지션을 의미한다. top, jungle, mid, ad carry, supporter가 있다.
  + champion: 플레이어가 어떠한 챔피언을 플레이 했는지를 말한다. LoL에는 2024년 6월 기준 167개의 챔피언이 존재한다.
  + spell: 플레이어가 게임에 들어가기 전에 사전에 설정할 수 있는 게임적 스킬을 말한다.
  + kill: 플레이어가 한 게임동안 처치한 적 챔피언의 수를 말한다.
  + death: 플레이어가 한 게임동안 처치 당한 수를 말한다.
  + assist: 플레이어가 한 게임동안 적 챔피언의 처치에 기여한 수를 말한다.
  + total damage: 플레이어가 한 게임동안 적에게 가한 피해량의 총합을 말한다.
  + total taken damage: 플레이어가 한 게임동안 적에게 받은 피해량의 총합을 말한다.
  + wards: 플레이어가 한 게임동안 사용한 와드(시야를 밝히는 아이템)의 수를 말한다.
  + cs: Creep Score로, 플레이어가 한 게임동안 챔피언을 제외한 모든 개체들을 처치한 수를 말한다.
* our\_team\_object: 게임의 승리에 중요하게 작용하는 요소들에 관한 데이터이다. [baron, dragon, riftherald, voidgrub, tower, inhibitor]로 이루어져 있으며, 각각 처치한 수를 기록했다.
* our\_team\_total\_kill: 우리팀이 한 게임동안 적 챔피언을 처치한 수의 총합을 말한다.
* our\_team\_total\_gold: 우리팀이 한 게임동안 얻은 게임내 재화의 총합을 말한다.
* enemy\_team,enemy\_team\_object, enemy\_team\_total\_kill, enemy\_team\_total\_gold: 위의 our\_team들과 같은 의미를 가진다. 단, 우리팀이 아닌 적팀에 대한 정보를 말한다.
* game\_players: 한 게임을 플레이한 10명의 서로 다른 플레이어들의 id를 말한다.

위 데이터를 이용하여 다음과 같은 예제를 해결하는 것이 목적이다.

1. 우리팀 플레이어 4명이 "Skarner", "Graves", "Twisted Fate", "Rakan"로 챔피언을 정했고, 적팀 플레이어 5명이 "Jayce", "Fiddlesticks", "Kayle", "Corki", "Braum"로 챔피언을 선택했다면, 남은 우리팀 플레이어 1명이 어떠한 챔피언을 선택했을 때 승률이 높을 것인가? 만약 lucian을 선택했을 때의 승률이 59%로 가장 높다면 이를 사용자에게 추천해준다.
2. 어떠한 조합이 가장 많이 등장했고 승리했는지를 바탕으로 임의의 조합을 추천해준다. 이는 LoL에서 2인 Ranked Solo 게임(Ranked Solo와 같지만 게임 시작 전 최대 두 명의 플레이어가 같은 게임을 할 수 있음) 을 플레이했을 때 추천해주기 위함이다. 예를 들어 한 명의 플레이어가 ‘Lucian’을 선택했을 경우, ‘Nami’를 추천해준다.



Figure 1. 예제 1번의 예시

**Data Understanding**

본 데이터는 op.gg에서 제공하는 상위랭커(2024년 6월 8일 기준 1등부터 1000등)들의 전적을 기반으로 크롤링한다. selenium 등을 이용하여 크롤링한 후, 데이터를 csv파일로 저장하여 사용한다. csv파일에서 하나의 행이 하나의 게임을 나타낸다.

A screenshot of a computer

Description automatically generated

Figure 2. 하나의 행을 출력해본 결과

직접 크롤링하여 얻은 데이터는 79943개이다. 이는 figure 1에 있는 게임 시작 전 화면에 해당한다(figure 1과 figure 2는 다른 데이터). 이하의 data understanding은 data-preprocessing을 거친 후, 53347개의 게임 데이터에 대한 분석이다.

1. 승패 게임 분석

A blue and orange pie chart

Description automatically generated

Figure 3 승패 게임 수 그래프

Figure 3을 보면 상위랭커들의 승리한 횟수가 많음을 알 수 있다. 전체 데이터들 중 승리한 게임 수는 30360이고, 패배한 게임 수는 22987이다.

1. 챔피언 별 플레이 횟수 및 스펠 별 사용횟수

A graph of a number of people

Description automatically generated

Figure 4. 각 챔피언 별 플레이 상위 10개

Figure 4를 보면 어떠한 챔피언이 많이 플레이 되었는지 알 수 있다. 이를 이용해 현재 메타를 잘 살펴볼 수 있다. 각 플레이 횟수를 보면 "Nautilus",8486, "Kai'Sa",7838, "Viego",7309, "Ashe",6361, "Skarner",6261, "Tristana",5934, "Nidalee",5593, "Corki",5532, "Hwei",5502, "Varus",5365이다.

A graph of a number of spells

Description automatically generated

Figure 5. 각 스펠 별 사용횟수

Figure 5를 보면 각 스펠 별 사용횟수를 알 수 있다. "Flash", 255723, "Teleport", 90856, "Ignite", 54875, "Smite", 53350, "Heal", 25778, "Cleanse", 18236, "Ghost", 16510, "Barrier", 13151, "Exhaust", 4991로 사용되었다. 각 수치가 큰 이유는 각 게임 별 10명의 플레이어가 선택한 스펠이기 때문에 모든 수치의 총합을 10으로 나누면 총 게임 수인 53347이다.

1. 승률이 좋은 챔피언

A graph of green bars

Description automatically generated with medium confidence

Figure 6. 승률이 높은 챔피언

Figure 6을 통하여 어떠한 챔피언이 현재 승률이 높은 지 알 수 있다.

|  |  |  |
| --- | --- | --- |
| Champion | Win Rate | Games |
| Kayle | 0.75 | 88 |
| Morgana | 0.6222222222222222 | 180 |
| Renata Glasc | 0.6170212765957447 | 423 |
| Rek’Sai | 0.6143057503506312 | 2139 |
| Evelynn | 0.6142857142857143 | 70 |
| Annie | 0.6085011185682326 | 447 |
| Nunu & Willump | 0.6084183673469388 | 784 |
| Tahm Kench | 0.6068376068376068 | 234 |
| Skarner | 0.6066123622424533 | 6261 |
| Thresh | 0.6063257065948856 | 2972 |

자세한 주요 수치들을 표로 보면 위와 같다. 여기서 주목할 점이 게임의 총 플레이 횟수에 따른 승률이 일반적인 상관관계를 가지지 않는다는 것이다. 챔피언 간의 밸런스가 잘 잡혀 있다고 가정했을 때, 승률 하나만으로는 고승률의 챔피언이 정말로 좋은지는 판단하기 어렵다. 허나 플레이 된 횟수와 함께 승률이 높은 챔피언이 일반적으로 현재 메타에서 좋은 챔피언이라고 생각할 수 있으며 범용적인 챔피언일 확률이 높을 것이다. 게임 내에서 플레이할 수 있는 챔피언이 많은 까닭에 소위 말하는 장인(하나의 챔피언에 대해 이해도가 높으며 플레이를 많이 하는 플레이어를 일컫는 말)들이 상위 랭커일 때, 실질적인 메타에 대해서 크게 반영을 하지 못할 수도 있다. 다만 그러한 챔피언들이 정말로 좋아진 경우에는 플레이 되는 게임 수가 자연스레 많아지며 좋은 챔피언이라 분류될 수 있을 것이다. 위 표에서 예시를 들면 Kayle은 플레이 횟수가 88판으로 적지만 6261판 플레이 된 Skarner 보다는 승률이 높다. 그러나 일반적인 유저가 플레이하기에는 Skarner를 해보는 것이 좋을 수 있다. 물론 실제로 Skarner는 최근에 리메이크(기존에 있던 챔피언의 스킬 등을 새로이 만드는 작업)된 챔피언이며 많은 사람들이 플레이 해보고 있긴 하다. 이는 리메이크 된 챔피언은 출시 초반에 일반적인 챔피언 밸런스보다 아주 뛰어나거나 아주 안 좋거나 하기 때문이다. 따라서 이후 패치에 따라 챔피언 밸런싱이 이루어질 가능성이 높다. 그러나 현재 플레이하기에는 좋다는 것을 위 데이터로 알 수 있다.

1. 승률이 좋은 조합 / 낮은 조합

조합을 이용하여 어떠한 조합이 포함되어 있는 게임들 중, 이긴 게임의 수를 나누어 승률을 구해봤을 때, 다음과 같은 순위로 조합이 강력함을 알 수 있었다.

A screen shot of a computer program

Description automatically generated

Figure 7. 승률이 높은 조합

승률이 높은 조합과 마찬가지로 어떠한 조합이 승률이 낮은 지 구할 수 있었다.

A screen shot of a computer

Description automatically generated

Figure 8. 승률이 낮은 조합

1. 가장 많이 등장한 조합

크롤링한 데이터에서 어떠한 조합이 가장 많이 플레이 되었는지 구할 수 있었다.

A screen shot of a computer program

Description automatically generated

Figure 9. 가장 많이 플레이 된 조합

Figure 9를 보면 가장 많이 플레이 된 Nautilus가 역시 가장 많이 플레이 된 조합 상위에 자주 보이는 것을 알 수 있다.

1. lane 별 챔피언 승률

결과가 얼마나 잘 나왔는지 직접적으로 비교해보기 위하여 5개의 lane별 챔피언의 승률을 크롤링한다. 총 167개의 챔피언을 대상으로 5개의 lane에 갔을 때, 승률을 구한 데이터는 다음과 같다.

A screen shot of a computer

Description automatically generated

Figure 10. AD Carrylane의 각 챔피언별 승률 데이터

Figure 10을 보면 하나의 행은 [#,챔피언,플레이 수,평점,승률,게임당 픽률,게임당 밴률,cs,gold]이다.

* #: 행 번호를 말한다.
* 챔피언: 챔피언의 이름을 말한다.
* 플레이 수: 챔피언이 플레이된 게임의 수를 말한다.
* 평점: KDA라고도 하며, kill death assist를 말하며 일반적인 공식은 (kill + assist)/death이다.
* 승률: 이 챔피언이 플레이 된 게임들에서의 승률을 말한다.
* 게임당 픽률: 게임을 시작했을 때, 그 lane에서 선택되는 비율을 말한다.
* 게임당 밴률: 게임을 시작했을 때, 그 게임에 한해서 선택되지 못하는 챔피언을 밴 당한 챔피언이라 한다. 플레이 할 챔피언을 선택하기 전에 사전에 선택한다. 일반적으로 좋다고 여겨지는 챔피언이나 플레이어 본인이 상대하기 힘든 챔피언을 밴한다.
* cs, gold: 앞서 설명한 정의와 동일하다.

이 데이터를 이용한다면 추천된 챔피언이 얼마나 타당하게 추천되었는지 실험적으로 확인해 볼 수 있을 것이다.

**Data pre-processing**

Data pre-processing은 다음과 같은 것들을 진행하였다.

1. 중복된 게임 제거

상위랭커들의 게임은 비슷한 실력의 사람들이 플레이하기 때문에 같은 게임이 존재할 수 있다. 이를 위하여 플레이어들의 이름을 이용한다. 여기서 모든 플레이어들의 이름이 같고, 이외의 게임 관련 정보가 순서만 다르다면 같은 게임으로 간주한다. 이와 같이 중복된 게임을pandas의 drop으로 제거한다.

1. 게임 종류 확인 및 제거

이는 초반부 데이터 크롤링 때 고려하지 못한 것으로 플레이어들이 게임을 진심으로 한다는 것을 가정하기 위해서는 중요한 게임들만 고려해야 한다. 하지만 이는 쉬운 일이 아니므로 자신의 실력이 기록되는 Ranked Solo 게임만을 데이터로 사용하기로 한다. 순수한 재미를 위해 남에게 피해를 줄 수 있는 플레이들이 포함된 게임 데이터들을 배제하기 위함이다.

1. 챔피언과 스펠을 정수로 매핑

조합 관련 컴퓨팅 및 이후에 진행할 것들을 고려해보았을 때, 직접 데이터를 사용하는 것보다 정수로 매핑하여 사용하는 것이 효율이 좋을 것이라 생각했다. 따라서 csv 파일을 변경하여 각 챔피언들을 1에서 167사이의 숫자로, 스펠들을 1에서 9사이의 숫자로 변경한다.

1. Data pre-processing 이후 필요 없어진 행 제거

사실 상 User\_id, User\_tag\_id, players등은 본 프로젝트의 목표에 필요하지 않으므로 이러한 행들은 제거한다.

이외의 것들은 직접 데이터를 크롤링하였기 때문에 missing data, Unicode 등의 문제는 고려할 필요가 없다. 이를 직접 확인 해보았으나 사실상 한 게임의 데이터에서 하나의 정보라도 빠지면 그 게임이 유효한지 판단할 수 없으므로 크롤링할 때 예외처리로 미리 제거를 하였다. 또한 data의 비대칭성이 결국 승률이 좋은 챔피언을 추천하는 데에 기인하므로 한쪽에 치우친 데이터 (예를 들면 승리 데이터가 더 많으므로)를 그대로 사용하는 것이 좋을 것 같다고 판단하였다. python 모듈들 중 이를 해결하기 위한 SMOTE(synthetic minority oversampling technique)이 있으나, 이를 직접 사용하기에는 상대적으로 소수인 데이터의 크기와 다수인 데이터의 크기가 크게 차이 나지 않기 때문에 사용하지 않기로 한다.

**Data Mining Project #2**

* 프로젝트를 진행하다 보니 프로젝트 1 과는 여럿 변경된 점이 있다. 변경된 부분은 본 문서에 반영하고 수정하였다.

**Method**

본 프로젝트에서 해결하고자2가지를 각 예시에 따라 진행해보았다.

1. 우리팀 플레이어 4명이 "Skarner", "Graves", "Twisted Fate", "Rakan"로 챔피언을 정했고, 적팀 플레이어 5명이 "Jayce", "Fiddlesticks", "Kayle", "Corki", "Braum"로 챔피언을 선택했다면, 남은 우리팀 플레이어 1명이 어떠한 챔피언을 선택했을 때 승률이 높을 것인가? 만약 lucian을 선택했을 때의 승률이 59%로 가장 높다면 이를 사용자에게 추천해준다.

본 예제를 naïve하게 준비한 데이터에서 찾는 것을 먼저 해보았다. 목표는 top 5를 추천 하는 것이지만 실제로 준비한 데이터에서만 찾으려 하니 Kai'Sa가 있는 게임 데이터들만 고려하여 하나의 챔피언만을 추천해주었다. 따라서 이를 위한 승률이 높은 모델을 예측 후, 추천하는 것으로 진행하였다.

해야 할 것은 주어진 조합에 따라 이 조합이 승리했는지, 혹은 패배했는지를 기반으로 하고 싶기에 classifier를 사용한다. 이는 결국 단순 0, 또는 1을 분류하는 문제이므로 로컬에서 학습 속도가 빠르기 때문에 더욱 메리트가 있어 보인다. 이에 따라 세가지 classifier를 사용하고 비교해보기로 한다.

1. random forest classifier

random forest classifier은 decision tree를 여러 개 사용하여 예측을 하는 방법을 말한다. 이는 각각의 decision tree가 독립적으로 학습되어 앙상블로 최종 예측을 내리는 학습 방법이다. 즉, 모든 decision tree의 예측을 이용하는 것이다. 또한 과적합을 방지할 수 있으므로 비교적 적은 데이터를 가진 본 프로젝트에서 사용하기 좋을 것 같다고 생각했다.

1. gradient boosting classifier

gradient boosting classifier은 random forest classifier과 비슷하게 decision tree(weak learner)를 사용하지만, 이전 모델의 오차(residual error)를 보정하며 weak learner를 학습시키는 앙상블 방법이다. 이름에 gradient가 들어가 있듯이 loss function의 gradient를 따라 loss를 줄이는 방향으로 학습을 진행한다. 마지막으로 모든 weak learner들의 예측을 이용하여 최종 예측을 내린다. 이 분류기의 과적합을 방지하기 위해서는 learning rate나 decision tree의 depth를 조정해야 한다. 또한 random forest classifier과 마찬가지로 앙상블 기법을 사용하기 때문에 모델이 복잡하며 해석이 어려운 점이 있다.

1. xgboost classifier

xgboost classfier은 gradient boosting 알고리즘을 개선하여 효율과 성능을 향상시킨 분류기이다. 이 분류기는 L1, L2 정규화를 이용하여 과적합을 방지한다. 또한 sparse data를 효율적으로 처리할 수 있다. 다만 본 프로젝트의 data는 sparse하지 않기 때문에 큰 성능 향상을 기대하긴 어렵다. random forest, gradient boosting classifer의 문제점 중 하나인 decision tree의 과적합을 최대 depth 대신 최대 leaves로 pruning하여 decision tree의 복잡성을 줄임으로 해결한다. 또한 병렬 처리가 가능하여 학습 속도가 빠른 편이다. 단, 이러한 것들로 인하여 hyperparameter가 많아져 tuning이 복잡한 편이다.

예제 1을 기반으로 위 classifier들을 학습시키고 예측할 때, 우리가 가진 정보는 챔피언들의 이름 뿐이므로 이외의 것들을 함께 설정해야 한다. 미리 설정할 값들은 game\_time, our\_team\_total\_kill, our\_team\_total\_gold, our\_team\_object, enemy\_team\_total\_kill, enemy\_team\_total\_gold, enemy\_team\_object이다. 이들은 게임을 이긴 팀에 대하여 각 값들의 평균을 내어 사용한다. 이외의 데이터는 평균을 내어 사용하기 어렵기 때문에 미리 설정하지 않는다.

A computer screen shot of a number

Description automatically generated

Figure 11. 평균으로 구한 값들

위 값들을 반올림하여 정수로 변환하여 각 classifier 학습에 사용한다. 또한 각 classifier들의 hyperparameter를 최적으로 찾기 위하여 GridSearchCV를 이용한다. 이는 hyperparameter를 튜닝하며 교차 검증을 통해 모델의 성능을 평가, 그리고 가장 성능이 좋은 hyperparameter 조합을 선택할 수 있게 해준다.

1. 어떠한 조합이 가장 많이 등장했고 승리했는지를 바탕으로 임의의 조합을 추천해준다. 이는 LoL에서 2인 Ranked Solo 게임(Ranked Solo와 같지만 게임 시작 전 최대 두 명의 플레이어가 같은 게임을 할 수 있음) 을 플레이했을 때 추천해주기 위함이다. 예를 들어 한 명의 플레이어가 ‘Lucian’을 선택했을 경우, ‘Nami’를 추천해준다.

이 예제를 해결하기 위해 apriori 알고리즘 사용한다. 이를 위해association rule을 생성하여 특정 챔피언이 선택된 경우에 다른 챔피언을 추천한다. 그리고 비교적 적은 데이터를 최대한 활용하기 위하여 각 데이터가 가진 result\_value가 0인 경우, 적팀을 우리팀으로 생각하여 활용한다. 이렇게 한다면 53347개의 행을 모두 사용할 수 있다. apriori알고리즘은 어떠한 대상을 포함한 집합이 많이 등장하는지 알아내는 것을 효율적으로 하기 위한 알고리즘이다. 예를 들어 여러 사람의 장보기 후 영수증을 보았을 때, 사람들이 맥주를 산 경우 이 사람이 감자칩을 같이 구매하는 경향이 있는지 없는지를 알 수 있다. 즉 어떠한 두 대상의 support가 설정한 minimum support를 넘기는 집합을 구하는 알고리즘이다. 따라서 각 대상 간의 연관 규칙을 알 수 있다. 이 때, 조합에 따라 등장한 횟수가 적은 게임들은 적절히 없애는 것이 중요하다. 예를 들어 ‘Lucian’, ‘Nami’ 조합이 등장한 게임의 횟수가 1이고 승리했다면, 단 한 판으로 인하여 승률이 100%가 되는 상황이 발생한다. 따라서 조합이 등장한 게임의 횟수를 세어 미리 필터링하여 실험을 진행하는 것이 좋아 보인다.

실험은 python으로 코드를 짜서 진행한다. 이 때, 각 예제에 따라 사용하기 좋은 모듈들을 알맞게 사용한다. 본 프로젝트에서 크게 사용하는 모듈은 pandas, scikit-learn, xgboost, mlxtend등이 있다.

**Experimental Result**

각 예제들을 실험해본 결과는 다음과 같다.

먼저 예제를 예측해보기 위하여 다음으로 설정하였다.

* our\_team\_partial = ["Skarner", "Graves", "Twisted Fate", "Rakan"]
* enemy\_team = ["Jayce", "Fiddlesticks", "Kayle", "Corki", "Braum"]
* game\_time = 1449
* our\_team\_total\_kill = 30
* our\_team\_total\_gold = 53270
* enemy\_team\_total\_kill = 18
* enemy\_team\_total\_gold = 44910
* our\_team\_object = [1, 2, 1, 3, 7, 1]
* enemy\_team\_object = [0, 1, 0, 2, 2, 0]

our\_team\_partial은 미리 선택된 우리팀의 4개의 챔피언을, enemy\_team은 미리 선택된 적팀 5개의 챔피언을 말한다. 이외의 것들은 앞서 설명한대로 구한 평균을 반올림한 값이다. 가장 먼저 해본 것은 hyperparameter를 튜닝하지 않고 각 classifier들을 학습시켜보았다. 그 결과는 아래 그림과 같다.

**A screenshot of a computer program

Description automatically generated**

Figure 12. 예제 1의 실험 결과

각 classifier들을 보면 예상 승률을 100%로 생각되어지는 챔피언이 많음을 알 수 있다. 예제 1에서 주어진 조건은 우리팀이 "Skarner", "Graves", "Twisted Fate", "Rakan"를 선택했을 때이다. 이 경우는 사실

ad carry를 비워두고 진행한 것이었다. 하지만 결과에서는 실질적으로 ad carry lane에서 자주 쓰이는 챔피언이 등장하기 보다는 승률에 따라 승리할 확률이 높은 챔피언들을 추천해주었다. 그러나 3가지 분류기 모두 ad carry 챔피언을 추천 해주었다. random forest classifier는 corki를, gradient boosting classifier은 corki를, 그리고 xgboost classifier은 smolder를 추천 해주었다. 실제 ad carry 챔피언별 승률을 보면 corki는 20위, smolder은 16위에 있는 챔피언이다. 순위가 낮다고 생각되지만, 위 조합에서는 이러한 챔피언들이 유효하다는 것을 알려준다. 이하는 GridSearchCV를 이용하여 최적의 hyperparameter를 찾아내고 학습시킨 결과이다. 이때, 각 classifier별 최적의 hyperparameter와 cross-validation의 최고 정확도는 다음과 같다.

1. random forest classifier
   1. max\_depth: 30
   2. min\_samples\_leaf: 1
   3. min\_samples\_split: 5
   4. n\_estimators: 200
   5. best cross-validation accuracy: 0.99
2. gradient boosting classifier
   1. learning\_rate: 0.2
   2. max\_depth: 7
   3. n\_estimators: 200
   4. subsample: 0.9
   5. best cross-validation accuracy: 0.99
3. xgboost classifier
   1. learning\_rate: 0.2
   2. max\_depth: 7
   3. n\_estimators: 200
   4. subsample: 0.9
   5. best cross-validation accuracy: 0.99

이에 따른 결과는 다음 그림과 같다.

A screenshot of a computer program

Description automatically generated

Figure 13. GridSearchCV를 이용한 결과

위 결과에서 각 classifier들이 추천해준 ad carry 챔피언을 비교해 봤을 때, random forest classifier는 aphelios, varus를, gradient boosting classifier은 xayah를, 그리고 xgboost classifier은 smolder를 추천 해주었다. 먼저 aphelios의 승률 순위는 15위, varus는 10위, xayah는 19위, smolder는 16위이다. 이를 통해 GridSearchCV를 이용하여 hyperparameter를 튜닝한 것이 조금 더 나은 성능을 보임을 알 수 있다. 또한 추천해주는 챔피언이 조금씩 달라지는 것을 보아 각 classifier들의 학습 및 평가 방법에 따라 어떠한 조합이 위로 올라오는지 알 수 있다. 또한 챔피언들의 각 ad carry lane에서의 승률 순위를 비교해보면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | GridSearchCV(x) | | GridSearchCV(o) | |
| Classifier | 챔피언 | 승률 순위 | 챔피언 | 승률 순위 |
| Random Forest | Orianna | 86 | Aphelios | 15 |
| LeBlanc | 94 | Varus | 10 |
| Corki | 20 | Riven | 109 |
| Kayle | 55 | Syndra | 48 |
| Seraphine | 27 | Pantheon | 49 |
| 평균 순위 | 56.4 | 평균 순위 | 46.2 |
| Gradient Boosting | Trundle | 89 | Briar | 111 |
| Orianna | 86 | Trundle | 89 |
| LeBlanc | 94 | Morgana | 54 |
| Corki | 20 | Singed | 102 |
| Kayle | 55 | Xayah | 19 |
| 평균 순위 | 68.8 | 평균 순위 | 75.0 |
| XGBoosting | Smolder | 16 | Nunu & Willump | 119 |
| Nunu & Willump | 119 | Smolder | 16 |
| Karthus | 30 | Kindred | 58 |
| Taric | 142 | Yone | 34 |
| Swain | 31 | Taric | 142 |
| 평균 순위 | 67.6 | 평균 순위 | 73.7 |

위 결과를 보았을 때, hyperparameter을 튜닝하여 최적이라 선택된 모델이 gradient boosting classifier과 xgboosting classifier에 대해서는 평균 순위가 더 커진 것을 알 수 있다. 하지만 random forest의 경우에는 평균순위가 더 작아졌다. 또한 각 모델별로 평균 순위를 기반으로 한 순위를 매겨보자면 random forest, gradient boosting, xgboosting 순서대로 성능이 좋은 것 같다고 생각할 수 있다.

이 예제에 대해서 먼저 챔피언 2개를 조합했을 때, 그 조합이 등장한 횟수(TotalCount)를 세어본다. 아래 그림은 TotalCount를 센 것이다.

A graph of a distribution of total count

Description automatically generated

Figure 14. 챔피언 2개의 조합이 등장한 게임 수의 분포도

Figure 14를 통해서 알 수 있는 점으로 TotalCount가 적은 조합이 매우 많음을 알 수 있다. 이를 출력해서 알아보면 TotalCount가 1인 조합의 수는 2007이며 가장 많이 등장한 조합에 대한 TotalCount는 1821이다. 또한 TotalCount의 총 가짓수는 477개이다. 따라서 많이 등장하지 않은 조합을 적절히 걸러내기 위해 대략적으로 TotalCount가 낮은 25% 데이터(TotalCount가 10보다 작은 데이터들)를 제거한다. 이로서 특정 챔피언이 아주 적은 게임으로 높은 승률을 가지는 데이터를 걸러낼 수 있다. TotalCount의 threshold를 10으로 설정하고 나서 조합별 분석을 해보았다.

먼저 TotalCount가 가장 많은 순서대로 상위 10개의 조합을 뽑아보았다.

A screenshot of a computer screen

Description automatically generated

Figure 15. 상위 5개의 승률이 높은 챔피언 조합

제일 왼쪽에 있는 것은 조합을 구할 때 부여된 번호이다. 가장 많이 나온 조합들을 분석을 해보면 Kai’Sa와 Nautilus의 조합이 가장 많이 등장했으며, 승률 또한 높은 편인 것을 알 수 있다. 그리고 주목할만한 부분이 가장 많이 나온 조합들을 잘 보면 ad carry와 supporter가 함께 잡히는 경우가 많다는 것이다. 이는 Ranked Solo를 둘이서 플레이 할 때 일반적으로 ad carry와 supporter(19, 104, 38, 194, 420), mid와 jungle, top과 jungle의 조합으로 듀오를 많이 하는 것을 반영하는 것 같다. 하지만 이러한 조합이 아닌 310(jungle, supporter), 119(jungle, ad carry), 16(jungle, supporter), 101(jungle, ad carry), 26(jungle, supporter)를 보면 저 두 조합이 많이 플레이 되고 승률 또한 나쁘지 않음을 미루어 보았을 때, 만약 팀에 저 챔피언이 있을 경우, 다른 하나를 고려하는 것 또한 좋은 선택임을 알 수 있다.

이하는 상위 5개의 승률이 높은 챔피언 조합이다.

A screen shot of a computer

Description automatically generated

Figure 16. 이하는 상위 5개의 승률이 높은 챔피언 조합

Figure 16은 승률이 높은 챔피언 조합을 출력해본 것이다. 예상대로 전체 게임 수가 작을 수록 이러한 데이터들이 많음을 알 수 있다. Qiyana의 경우, 전체 데이터에서 2139 게임이 진행 되었는데 실제 op.gg 승률은 47.67%이다. 이는 상위 랭커의 게임들을 보면 장인들이 플레이하는 게임들이 많은데, 범용적인 챔피언들(조작이나 스킬 구성이 쉽거나 초보자들의 진입장벽이 낮고 성능이 좋은 챔피언) 보다는 플레이를 완전히 숙달하기에 어렵기 때문에 일반적인 전체 승률은 낮은 편이다. 비슷한 예로 Gangplank의 경우도 마찬가지이다. 또한 장인들은 하나의 챔피언을 끝까지 파는 경우가 많기에 이들 중에서 잘하는 이들은 다른 챔피언을 거의 플레이하지 않고도 상위 랭커가 되는 경우도 많다. 따라서 TotalCount가 작으면서 승률이 좋은 것은 초보자에게 추천하기 까다로운 챔피언들이 종종 있을 경우가 크다. 그러므로 TotalCount가 클수록 조금의 보정을 해준 다음 추천하는 것이 바람직해 보인다. 또한 추천을 받고자 하는 챔피언의 lane을 기준의 승률 데이터와 조합별 데이터셋에서의 승률을 이용하여 보정한다. 정확한 과정은 다음과 같다.

* 챔피언이 등장한 게임과 그 챔피언을 포함한 조합의 수를 TotalCount로 구한다.
  + 가중치(w\_tc)는 0.3으로 한다. Log를 취하는 방법도 있지만 선형적으로 한 이유는 많이 등장하는 조합이라면 당연히 플레이 수가 많을 것이므로 승률이 상대적으로 낮을 수 있기 때문이다. 또한 상위 랭커들의 플레이 데이터이므로 그들의 판단을 근거로 한다.
* 추가적으로 크롤링한 챔피언 별 전체 승률(WinRate)을 가져온다.
  + 가중치(w\_wr)는 0.1로 한다. 이는 크롤링한 데이터셋에 크게 귀속되어 있으므로 가중치를 적게 주는 것이다.
* 선택한 챔피언이 조합에 등장했을 때, 어떠한 챔피언이 등장할 확률을 Confidence로 구한다.
  + 가중치(w\_cf)는 0.5로 한다. 이는 상관관계에 따라 가중치를 더 줄 필요가 있어보이기 때문이다.
* 조합별 데이터셋에서의 승률 Combination을 구한다.
  + 가중치(w\_cb)는 0.1로 한다. 이는 WinRate와 비슷한 이유에서이다.
* 가중치가 부여된 Confidence는 w\_tc \* TotalCount + w\_wr \* WinRate + w\_cf \* Confidence + w\_cb \* Combination로 구한다.

보정하기 전후에 추천 시스템에 ‘Lucian’을 넣으면 각각 ‘Nami’가 나온다. 이는 데이터셋에서Lucian이 있는 경우에 ‘Nami’가 있을 때가 압도적이기 때문이다. 따라서 ‘Lucian’ 대신 여러 챔피언들로 각각에 대해서 실험을 진행해보았다.

|  |  |  |
| --- | --- | --- |
|  | 선택된 챔피언 | 추천된 챔피언 |
| 추천 시스템 보정 전 | Lucian | Nami |
| Kai’Sa | 'Camille', 'Rell', 'Skarner', 'Taliyah', 'Nautilus' |
| Ashe | 'Viego', 'Braum', 'Nidalee', 'Skarner', 'Rumble' |
| 추천 시스템 보정 후 | Lucian | Nami |
| Kai’Sa | ‘Skarner', 'Nautilus', 'Camille', 'Taliyah', 'Nidalee' |
| Ashe | 'Skarner', 'Viego', 'Braum', 'Rumble', 'Nidalee' |

보정을 하고나서 달라진 점이 몇몇 있다. 먼저 추천되는 챔피언들의 우선 순위가 바뀌었다. 다만 보정 전후 모두 전체적으로 추천된 챔피언들이 개별적으로 플레이 수가 높은 챔피언들이며 이들을 추천 하는 순서가 바뀐 것에 주목할 필요가 있다.

**Conclusion**

이번 프로젝트의 결론은 다음과 같다.

1. 데이터의 중요성

본 프로젝트에서 사용한 Ranked Solo에서 나올 수 있는 모든 경우의 수는 챔피언만 따졌을 때, 약 8.307\*10^39이다. 물론 일반적으로 자신의 점수를 올리기 위해서 각 lane 별 좋은 챔피언만 하기 때문에 저만큼의 모든 경우가 나올 수는 없다. 하지만 절대적으로 크롤링한 데이터의 수가 적은 것 같다는 생각을 떨치긴 어려웠다. 일반적으로 데이터가 많으면 과적합도 방지할 수 있을 뿐만 아니라 모델들이 더욱 다양한 패턴을 학습할 수 있을 것이다. 이를 위해 cross-validation을 사용한 것이긴 하다.

1. 구체적인 모델링의 필요성

이는 2번째 예제를 해결하는 과정에서 느꼈다. 가장 아쉬운 것으로 챔피언이 갈 수 있는 lane을 국한시켜버린 것이다. 예제에서는 ad carry에 대해서 분석을 했지만 사실 ‘Ashe’라는 챔피언은 mid, supporter로도 플레이 되기 때문이다. 따라서 추천을 할 때 미리 lane을 지정하고 하는 것이 조금 더 올바른 추천이지 않을까 하는 아쉬움이 있다.

1. 전체 결론

실제로 플레이하는 게임에 대해서 이토록 분석해본 것이 처음이라 재미있었다. 물론 수강하는 과목이기에 열심히 한 것도 있으나 개인적으로 프로젝트를 진행하면서 궁금한 것들이 자꾸만 생겨 더욱이 흥미를 가진 것도 있었다. 다만 크롤링을 하면서 내가 필요한 데이터가 어디까지인지, 얼마나 필요한지에 대해 감을 잡지 못한 것이 아쉽긴 했다. 나중에 기회가 된다면 개인적으로 더 깊게 연구해보고 싶다.