졸업프로젝트 보고서

머신러닝을 이용한 게임 승부 예측

(League of Legends)

2013171046 심재훈

고려대학교 공과대학 전기전자과

1. Introduction

❑ Problem Statement

- League of Legends(이하 롤)에서는 10명의 플레이어가 5명씩 두 팀을 나누어 경기를 진행한다. 순위를 매기는 랭크 게임에서는 140가지가 넘는 챔피언 중에서 서로 번갈아 한가지 챔피언을 선택한다. 게임의 승리를 위해서는 챔피언을 선택하는 과정에서 전략적으로 고르는 것이 중요한데 이 과정에 머신러닝을 적용해 보려고 한다. 각 팀에서 챔피언을 5개씩 골랐을 때, 어느 팀이 유리한지 승률을 예측해보려고 한다.

❑ Pains and Needs

- 140가지가 넘는 챔피언 중 10가지 챔피언을 고르는 방법은 140C10로 매우 많다. 하지만 수집할 수 있는 데이터는 약 20만개로 모든 경우를 고려해 보기는 힘들다. 또한 같은 챔피언들 사이의 경기더라도 결과는 자주 바뀌므로 이를 고려해서 모델링을 해야 한다. 모델링을 할 때 게임에 대한 직관을 이용해 승리에 핵심적인 요인을 파악하고 이를 표현할 수 있는 feature를 만들어야 한다.

❑ Importance

- 게임 승부 예측은 다른 스포츠 경기에 확장하기 쉽다. 야구나 축구에서 인원 배치를 어떻게 하면 좋을지 고민하는 감독을 위해 이 모델을 변형해서 도움을 줄 수 있을 것이다. 또한 전반적으로 sparse한 데이터에 대해 어떤 feature engineering을 취해야 모델링 예측 향상에 도움이 될지 알아 볼 수 있을 것이다.

2. Related Work (Project)

- **풀고자 하는 연구 내용(Problem)을 다른 논문 /프로젝트들의 저자들은 어떻게 풀고 있는가? 해당 문제를 풀기 위해서 기존에는 어떤 접근법(approach)들이 존재 했는가? 그것들의 장단점은 무엇인가?**

롤에서의 승부 예측을 머신 러닝 관점으로 분석해본 자료는 없었다. 그러나 프로 경기의 경우 해설자들이나 실력자들이 전문가의 관점으로 분석해서 승부 예측을 한 경우는 많다. 이들은 각 챔피언에 대한 이해와 롤의 이해를 바탕으로 승부를 예측하는데 사람들의 주관에 따라 예측이 크게 빗나가는 경우도 많았고, 소수만이 이런 분석을 해낼 수 있다는 단점이 있었다. 이들 전문가의 분석에 의한 승부 예측은 원인을 이해할 수 있기에 예측 결과를 쉽게 받아들인 다는 점이다. 롤 외의 분야지만 비슷한 스포츠 승부 예측의 경우 많은 사례가 보인다. 머신러닝 기반 야구 경기 예측의 경우 Neural Network를 활용해 승부 예측을 진행한 결과가 있다. [1] 평균적인 예측 승률과 실제승률의 오차는 3.39%라고 한다.

3. Solution Approach (Main Idea)

- **아이디어란 주어진 문제를 효과적인 체계적으로 풀어 나가기 위한 일련의 처리 과정을 기술하는 것 입니다. What is your main idea to the problem?**

롤에서는 야구에서 투수와 포수같이 역할 군이 있는데 주로 싸우게 될 위치(라인)에 따라 탑, 정글, 미드, 원딜, 서폿으로 구분한다. 게임은 평균적으로 30분간 진행되는데 게임 시작 후 15분간은 라인에 머무르면서 게임을 진행(라인전) 한다. 게임 초반의 결과는 이후에도 영향을 크게 미쳐 중요성이 큰 편이다. 라인전에서는 챔피언간 1대1 상호작용이 많기 때문에 주로 이를 고려하고 챔피언을 고르게 된다. 이를 충분히 고려하지 못하거나 이해가 부족한 경우 게임 승패에 지대한 영향을 준다. 이 분석을 바탕으로 머신러닝에 쓰일 컬럼을 데이터에서 뽑아내려고 한다.

[1] http://commres.net/wiki/c/mt/2017/lecturer_note/group-02

4. Experiment / 성능 평가

분석 과정

1. 데이터 수집 및 전처리

데이터 수집은 Riot API를 통해 4개의 지역(KR, NA1, EUN1,EUW1) 에서 2개의 상위 티어(Challenger, Master)의 2가지 랭크타입 (SOLO, FLEX)의 게임결과를 불러오는 과정을 통해 이루어졌다. 이를 table 형식으로 가공하고 중복된 데이터를 제거한 결과 20만개의 데이터를 얻어 낼 수 있었다. 데이터 중 역할 군을 나타내는 컬럼은 mislabeled 된 경우가 많았는데 이를 위해 전처리를 해야 됐다. 각 챔피언마다 어울리는 역할 군이 있다는 지식을 활용해 데이터에서 각 챔피언이 어떤 역할 군을 가는지 통계 냈다. 이후 역할 군 컬럼에 대해서 5가지 역할 군이 겹치지 않게 통계치를 이용해 역할을 추정했다. 이 추정은 검증을 해봤을 때 정확했다.

140개 챔피언 중 각 역할 군에 자주 가는 챔피언은 한정되어 있다. 가지고 있는 데이터의 75%는 역할군마다 TOP 50개만을 이용해 구성해 낼 수 있다. 보기 드문 데이터를 가지고 학습을 시키는 경우 편차가 크기 때문에 이를 제외했다. 총 15만7천개의 데이터를 이용해 학습을 시켰다.

	A	B	C	D	E	F
1		Aatrox	Akali	Camille	Chogath	Darius
2	Aatrox	0.029845	0.036198	-0.21479	0.075128	0.115314
3	Akali	0.080889	0.041271	-0.07591	0.087902	-0.066
4	Camille	0.079682	0.068033	-0.05065	0.042479	0.065543
5	Chogath	0.005637	0.142155	-0.10438	0.034872	0.033936
6	Darius	-0.19248	0.070246	-0.05633	0.164057	0.050955

탑 vs 탑의 승률 평균(normalized)

	A	B	C
1	Aatrox	TOP	0.726383
2	Aatrox	MID	0.156019
3	Aatrox	ADC	0.048362
4	Aatrox	JUN	0.047635
5	Aatrox	SUP	0.0216
6	Ahri	MID	1
7	Akali	MID	0.668422
8	Akali	TOP	0.255946
9	Akali	ADC	0.033697
10	Akali	SUP	0.022981
11	Akali	JUN	0.018954
12	Alistar	SUP	0.998183
13	Alistar	MID	0.000908
14	Alistar	TOP	0.000545
15	Alistar	ADC	0.000182
16	Alistar	JUN	0.000182

역할 군 예측

2. 학습 과정

팀간 상호작용이 많이 일어나는 조합을 고려해서 컬럼을 만들었다. Team1 의 탑과 Team2 의 탑을 예를 들면, 각 챔피언마다 상대 챔피언을 만날 경우의 평균 승률을 계산하고 이를 normalize 했다. 만약 데이터가 없는 경우 중위 값을 이용했다. 라인-상대라인, 라인-상대정글, 라인-아군정글, 라인-아군라인의 상호작용을 뽑아내 총 21개의 컬럼을 새롭게 만들었다.

Train, Test Set 비율을 2대1로 나눈 후 이 21개의 컬럼과 승부 결과 값을 Random Forest Classifier(RFC)로 학습시켰다. Test Set에 대한 결과 값은 10가지 케이스에 대해 평균 55.2%이 나왔다. 비교하기 위해 Team1이 전부 이기는 경우를 고려했는데 이 경우 50.8% 였다. Naïve 한 예측보다 약 4.4%프로 예측률이 올라갔다. 단순한 방법으로 컬럼을 새롭게 뽑아내지 않고 categorical 데이터인 열에 대해 OneHotEncoding을 하고 학습을 시켜보기도 했다. 이 경우 평균 54 %이 나왔는데 승부 예측에 중요하지 않은 컬럼까지 학습을 시켜서 예측률이 떨어진 것 같다.

RFC 외에도 Logistic Regression, Naïve_Bayes, KNN Classifier, Discriminant Analysis를 써봤으나 53% 아래의 예측률이 나와서 decision tree 계열의 알고리즘을 고르게 됐다. 기본적으로 default parameter을 이용했다.

Train, Test Set의 비율을 95: 5로 했을 경우 예측률이 55.2%에서 56%로 올라가게 됐다.

Parameter Tuning 과정을 통해 예측률을 높여봤다. MLP Classifier 와 RFC에 대해서 이 과정을 해봤다. MLP의 경우 hidden_layer_sizes를 조절해봤는데 default 값 (100,) 에 비하여 (50,)인 경우 결과 값이 높았다. 하지만 이 경우에도 53.6%의 예측률 밖에 안 나와 최종적으로 RFC를 이용하기로 했다. RFC의 경우 조절할 parameter로 n_esitimator, max_depth, max_features가 있었는데 default 10, None, "auto" 에 비해 100, None, "log2"로 조절해주는 것이 결과값이 높았다. 결과값은 튜닝 전 56%에서 56.9%로 올라가게 됐다.

기간을 달리해 데이터를 수집 후 같은 모델링을 했을 때, 결과는 54.5%가 나왔다.

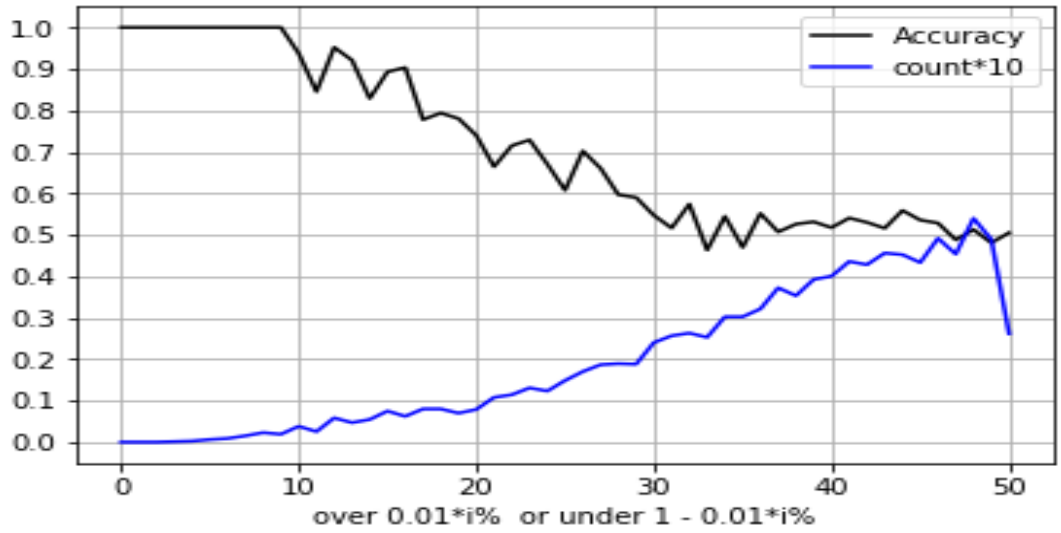
코드 실행 과정

1. getBestSummonersStats : 각 지역별로 마스터, 챌린저 티어의 summonerId 호출하기
2. addAccountIDWithBestSummoner: 1 번에서 호출된 summonerId 의 accountId 를 찾기
3. getAllGameId: 각 플레이어들이 최근 200 판동안 했던 게임의 id 를 호출하기
4. splitGameID: 3 번의 게임 id 를 지역별로 나누기
5. getGameDataFromGameID: 지역별로 나뉜 게임 id 로 json 형식의 게임 데이터를 호출하기
6. gameDataDictToTableV2: 5 번에서 호출된 json 파일을 table 형태로 가공하기
7. integrateDataTable: 지역별로 나누어진 데이터를 통합하기
8. mapChampionPosition: 7 번의 데이터에서 각 챔피언별로 포지션 선택 비율 뽑아내기
9. positionPrediction: 8 번의 테이블을 이용해 7 번의 데이터 중 포지션 컬럼을 가공하기
10. changePosition: 9 번의 데이터를 TOP, MID, JUN, ADC, SUP 순으로 바꾸기
11. rearrayV2: 10 번의 데이터를 최종 데이터로 쓸 컬럼만 뽑아내 가공하기
12. predictWin_final_v5: 최종 데이터를 이용해 모델링하기

5. Conclusion

Discussion

56.9%의 결과값은 Naïve한 예측보다 6.1% 높은 값이다. 100판중에서 6판을 이 알고리즘을 통해 승부 예측을 더 잘해낼 수 있다는 의미인데 만족스러운 수치는 아니다. 하지만 한 쪽으로 치우친 데이터의 경우 예측률은 높이 올라간다. 이 알고리즘이 보기에 결과가 뻔한 input의 경우, 즉 이길 확률이 90% 이상이거나 10%이하로 예측하는 경우 예측률이 98%까지 올라간다.



❑ **Summary / Contributions / Future work**

게임의 경우 게임사의 Patch와 플레이어의 상호작용에 따라 데이터가 많이 바뀐다. 따라서 정확한 예측을 위해서는 지속적으로 데이터를 업데이트 해줄 필요가 있다. 더 많은 데이터를 수집하고, 승부와 관련 있는 컬럼을 더 많이 찾아내 학습을 시키면 예측률을 높일 수 있을 것이다. 데이터 중에서 게임과 관련된 구체적인 수치들은 이용하지 않았는데 이 데이터 또한 가공해서 활용할 만한 가치가 보인다.

6. 별첨1(Source, data files description)

#dataFiles

#accountIdDB.xlsx

region, summoner_id(primary key), accountId(primary key)

#championID.xlsx

id(primary key), key, name, title

#gameIdDB.xlsx

region, gameType, gameId(primary key)

#KR_RANKED_SOLO_5x5_challengerList.xlsx

rank_order, summoner_name,summoner_id(primary key), accountId(primary key), tier, LP ,wins, losses

#KR_RANKED_SOLO_5x5_data.txt

원본 json형식의 dataset

#KR_SOLO_gameDataTable2.xlsx

main dataset. decription below(그림 6)

KR_SOLO_gameDataTableDescription

#champion_rate.xlsx

KR_SOLO_gameDataTable2.xlsx 에서 승률, 픽률, 밴률 비율을 뽑아낸 데이터

#champion_win_ban.xlsx

KR_SOLO_gameDataTable2.xlsx 에서 챔피언별로 승리, 밴, 패치버젼을 뽑아낸 데이터

#code

#getChampionIDList.py

챔피언 ID list를 구하기 위한 api 호출

산출물: championID.xlsx

#getBestSummonersStats.py

region, gametype, tier별로 챌린저 마스터 리그 인원들의 리스트 뽑기

산출물: KR_RANKED_SOLO_5x5_challengerList.xlsx

#getAccountIDBySummonerID.py

summonerID로 AccountID 받기

산출물: accountIdDB.xlsx

#addAccountIDWithBestSummoner.py

account ID로 챌린저 마스터 리그 인원들의 리스트 뽑기

산출물: KR_RANKED_SOLO_5x5_masterList.xlsx

#getAllGameId.py

accountId를 이용해서 gameID를 얻기. 해당 플레이어들이 최근 플레이한 100~200경기의 gameID를 뽑을수 있다.

산출물: gameIdDB.xlsx

#splitGameID.py

GameID를 gameType, region 별로 나누어 저장하기

산출물: dataKRSolo.xlsx

#getGameDataFromGameID.py

gameID 리스트로부터 각 gameID의 정보 뽑아내기

산출물: KR_RANKED_SOLO_5x5_data.txt

#getSummonerName.py

accountId를 이용해서 SummonerName 뽑아내기

#gameDataDictToTable.py

json 형태의 파일을 input으로 하여 이를 변환하여 일정 형식의 table을 output으로 산출

산출물: KR_SOLO_gameDataTable2.xlsx

#descriptiveAnalysisV1.py

기술통계 분석.

산출물: champion_win_ban.xlsx, champion_rate.xlsx

#championRelationV1.py

챔피언, 변수간 관계분석

#descriptive.R

변수간 상관관계 분석, ANOVA 분석, boxplot 그리기