통계적머신러닝 Final Report **2015150017 통계학과 최성웅**

온라인 게임 '리그 오브 레전드' 승/패 예측 및 승리 확률 시뮬레이터 구현

1. '리그 오브 레전드' 란?

온라인 게임 '리그 오브 레전드'는 다섯 명으로 이루어진 두 팀이 대칭으로 구성된 경기장 내의 양 진영에서 출발하여 승부를 겨루는 게임이다. 일반적으로, 'top lane', 'mid lane'과 경기장 전체를 아우르는 'jungle' 포지션에 한 명씩 배치되고 경기장 아래의 'bottom lane'에 두 명이 배치된다. 경기장 좌측 하단과 오른쪽 상단에는 각 'Blue team'과 'Red team'의 **넥서스(Nexus)**, 즉 파괴되면 패배로 이어지는 중요한 구조물이 위치해 있고 넥서스로부터 주기적으로 **'미니언'**이라는 캐릭터가 출발하여 'top', 'bottom', 'mid' lane으로 이동한다. 각 팀의 구성원 다섯 명은 해당 미니언과 경기장 곳곳에 위치한 **'몬스터(monster)'**를 사냥하면서 중요 자원인 **골드(Gold)**를 획득함과 동시에 **레벨(Level)**을 높인다. 획득한 골드를 통해 구입한 아이템의 성능과 레벨에 비례하여 팀 구성원의 힘, 즉 상대에게 가할 수 있는 **데미지(Damage)**가 높아지고 그것을 기반으로 상대 팀 구성원을 처치하면서 상대 진영의 넥서스로 향하는 길에 위치한 **타워(Tower)와 억제기(Inhibitor)**를 빠르게 파괴해야 한다. 결과적으로, 상대 팀의 항복이 선언되기 전에 타워와 억제기 그리고 마지막으로 넥서스를 파괴하면 승리하게 된다. 지도 내에 빨간색 원으로 표시된 곳에는 **바론(Baron)과 드래곤(Dragon), 전령(RiftHerald)**이라는 대형 오브젝트가 위치하고 있으며, 주기적으로 나타나는 해당 오브젝트를 팀원들과 함께 처치하면 이동 속도, 골드 획득, 구조물 철거 속도 상승 등의 유리한 효과를 얻을 수 있다. 기본적으로 아군과 아군의 미니언이 없는 곳은 암흑 시야이기 때문에 상대 팀의 위치를 확인하기 위해서는 **시야(Vision)** 관련 아이템을 구입하여 경기장 곳곳에 설치하는 것이 중요하다.

위의 규칙에 기반했을 때, 특정 팀의 승리와 패배를 예측하는 데에는 많은 지표가 사용될 수 있다. 예를 들면, 많은 프로 선수들은 획득 골드양이 10000 이상 차이가 날 때는 역전하기 거의 불가능하다고 말한다 그 뿐 아니라, 오브젝트인 바론이나 드래곤을 처음 처치하는 팀 혹은 아군이 먼저 상대를 처치, 즉 **킬(Kill) 스코어**를 따냈을 때 승리 확률이 높다고 말한다. 팀의 승리와 패배에 영향을 미치는 주요한 요소가 있는 것은 확실하다. 프로 선수가 아닌, 일반적으로 게임을 즐기는 유저들도 그러한 주요 요소가 있다는 것을 직관적이고 경험적으로 이해하고 있다. 하지만 해당 분석을 통해서 실제로 이러한 지표들이 주어졌을 때, 승패 여부를 적절하게 예측할 수 있는지 혹은 어떤 지표가 상대적으로 승패에 더 큰 영향을 미치는지를 통계적 기법과 머신러닝 모델을 사용하여 확인하려고 한다.

2. 데이터 정제화 (불필요한 정보 제거) 및 변수 설명

분석에는 작년 특정 시즌 내 최상위 랭크(Challenger, Grand Master, Master)의 약 20000 경기 데이터를 사용하였다. 그리고 target variable은 'win_blue', 즉 블루 팀의 승패 여부로 설정하였다.

```
df.head()
```

	gameId	platformId	gameCreation	gameDuration	queueId	mapId	seasonId	gameVersion	gameMode	gameType	...
0	4.417695e+09	KR	1.590851e+12	1264.0	420.0	11.0	13.0	10.11.322.2991	CLASSIC	MATCHED_GAME	...
1	4.417596e+09	KR	1.590849e+12	1739.0	420.0	11.0	13.0	10.11.322.2991	CLASSIC	MATCHED_GAME	...
2	4.417436e+09	KR	1.590846e+12	1339.0	420.0	11.0	13.0	10.11.322.2991	CLASSIC	MATCHED_GAME	...
3	4.417280e+09	KR	1.590842e+12	2341.0	420.0	11.0	13.0	10.11.322.2991	CLASSIC	MATCHED_GAME	...
4	4.417211e+09	KR	1.590840e+12	1479.0	420.0	11.0	13.0	10.11.322.2991	CLASSIC	MATCHED_GAME	...

5 rows × 222 columns

```
print(df.shape)
```

```
(22228, 222)
```

df.head와 df.shape를 이용하여 전체적인 데이터 모습을 확인해보았을 때, raw data 내에는 블루 팀과 레드 팀의 승패 여부를 표시하는 'win_blue', 'win_red'를 제외하고 총 220개의 변수가 존재한다는 것을 어렵지 않게 확인 할 수 있었다. 직관적으로 생각했을 때도 너무 많은 설명 변수가 있음을 알 수 있다. 과도한 설명 변수를 데이터 분석에 사용하게 되면, '차원의 저주'에 이르거나 모델의 일반화에 실패하여 과적합 문제가 발생할 수 있다. 일정 이상으로 변수가 많게 되면, 많으면 많을수록 제대로 된 모델 학습 및 결과 도출이 어려워지게 되는 것이다. 때문에, raw data의 불필요한 게임 내 지표들을 삭제하는 과정이 불가피했다.

가장 먼저, 승/패에 영향을 미치지 않는 경기 내 유저들의 아이디, 닉네임, 선택한 캐릭터 등 유저 정보를 삭제하였다. 이후, 블루 팀의 정보가 곧 레드 팀의 정보로 직결되는 변수들을 추렸다. 예를 들어, 블루 팀의 승패 여부를 표시하는 'win_blue'가 1이라면 'win_red'가 0이 되는 것이 당연하다. 또한, 블루 팀이 첫 번째 드래곤 혹은 첫 번째 억제기의 처치 및 철거 여부를 나타내는 'firstDragon_blue', 'firstInhibitor_blue'가 1일 때도 'firstDragon_red', 'firstInhibitor_red'는 당연히 0이

될 것이다. 이런 논리로, 한 진영의 정보가 다른 진영의 정보로 이어지는 변수들은 레드 팀과 관련된 것을 삭제하였다.

또한, 정보 면에서 포함 관계에 있는 변수들을 추려서 삭제하였다. 예를 들어, 팀 구성원이 특정 횟수만큼 연속적으로 킬 스코어를 달성했음을 나타내는 'doubleKills_blue', 'tripleKills_blue', 'quadraKills_blue', 'pentaKills_blue' 등의 변수는 팀 구성원들의 전체 킬 스코어(Kill score)를 표시하는 'Kills_blue'에 상당 부분 포함된다고 볼 수 있다. 따라서 한 변수에 포함될 수 있다고 판단되는 변수들을 모두 삭제하였다.

다음은, 특정 열의 총합이 모두 0일 때 해당 변수를 삭제하였다. 아래는 위에서 설명한 변수 삭제 과정의 대략적인 예시이다. 해당 과정을 모두 진행했을 때, raw data의 약 220개 변수가 46개로 줄었음을 확인할 수 있다.

[5]:
```
# 분석에 불필요한 변수들
dropcols1 = ['gameId', 'platformId', 'gameCreation', 'queueId', 'mapId', 'seasonId',
             'gameVersion', 'gameMode', 'gameType', 'teams', 'participants',
             'participantIdentities', 'teamId_blue','teamId_red', 'bans_blue', 'bans_red',
             'participantId_blue','playerScore0_blue','playerScore1_blue','playerScore2_blue','playerScore3_blue',
             'playerScore4_blue','playerScore5_blue','playerScore6_blue','playerScore7_blue','playerScore8_blue',
```

[8]:
```
dropcols3 = []
for i in df.columns:
    if df.loc[:, i].sum() == 0:
        dropcols3.append(i)
```

[9]:
```
df.drop(columns = dropcols3, inplace = True)
print(df.shape)
```

```
(22228, 46)
```

마지막으로, 게임 규칙에 기반하여 변수 생성 및 결합을 진행했다. 예를 들어, 특정 팀이 총 네 마리의 드래곤을 처치하거나 처치 횟수에 관계없이 다섯 번째 드래곤을 처치하면 게임 내에서 유리한 효과를 얻을 수 있는데 이러한 규칙들을 반영한 변수들 추가적으로 생성하였다.

[10]:
```
# 4드래곤 변수와 장로드래곤 변수 만듦

df['blue4dragons'] = 0
df['red4dragons'] = 0
df['blue_olddragon'] = 0
df['red_olddragon'] = 0

for i in range(0, df.shape[0]):
    if df.loc[i, 'dragonKills_blue'] == 4:
        df.loc[i, 'blue4dragons'] = 1
    elif df.loc[i, 'dragonKills_blue'] >= 5:
        df.loc[i, 'blue_olddragon'] = df.loc[i, 'dragonKills_blue'] - 4
```

그리고 양팀에 모두 있는 연속형 변수를, 격차를 나타내는 변수로 나타내어 결합하고 결합에 사용된 변수들은 삭제하였다.

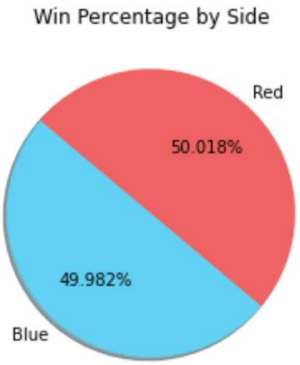
[11]:
```
# 양 팀에 모두 있는 연속형 변수를 그 차이를 나타내는 변수로 결합

df['gold_diff'] = df['goldEarned_blue'] - df['goldEarned_red']
df['level_diff'] = df['champLevel_blue'] - df['champLevel_red']
```

[13]:
```
df.drop(columns = dropcols4, inplace = True)
print(df.shape)
```

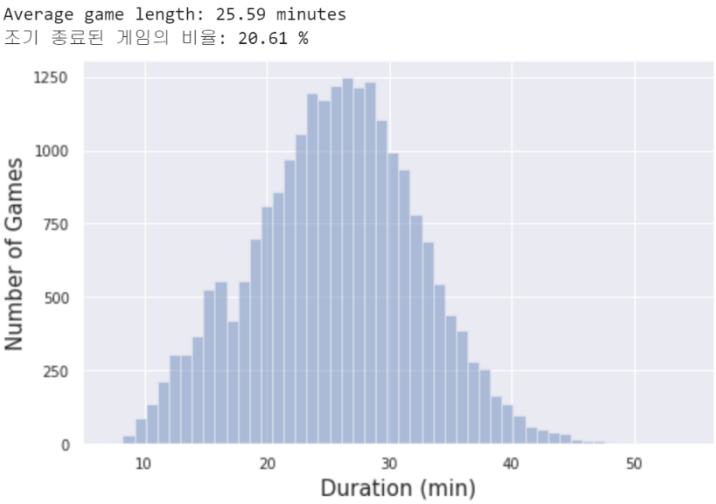
```
(22228, 38)
```

위의 과정을 모두 마쳤을 때, target variable인 'win_blue'와 총 경기 진행 시간을 나타내는 'gameDuration'을 제외하고 총 36개의 설명 변수가 남았음을 확인할 수 있다.

3. 데이터 시각화

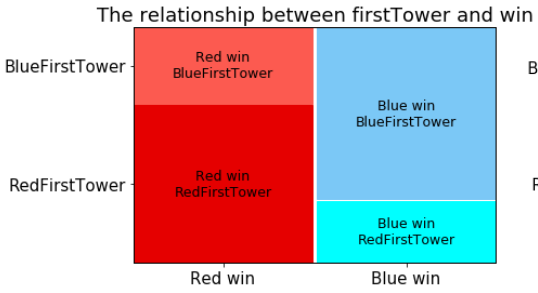


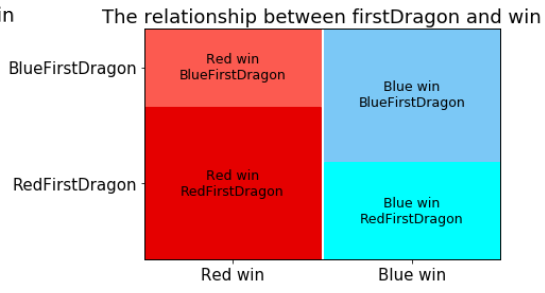
본격적인 분석에 앞서, 몇 가지 간단한 시각화를 진행하였다. 분량상 몇 가지 시각화 내용만을 보고서에 담았다. 먼저 왼쪽의 차트 그림을 통해 알 수 있듯이, 데이터 내의 블루 팀 승리 횟수와 레드 팀 승리 횟수는 거의 같다. 이는 다른 given information 없이도 승패 예측에 있어서 약 50%의 정확도를 기대할 수 있음을 의미한다. 전체 데이터 내에 결측값은 존재하지 않았다.



전체 경기의 평균 진행 시간은 약 26분이며 항복을 선언할 수 있는 시점인 20분을 기준으로 그 이전에 조기 종료된, 즉 한 진영의 압도적인 우세로 승패가 결정된 경기의 비율은 약 20.6%였다.

아래는 목적 변수와 특정한 패턴을 함께하는 공변량의 존재 여부를 확인하기 위해서 mosaic plot을 그려본 것이다.





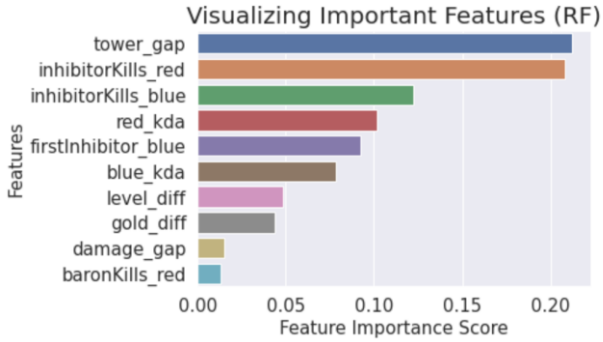
그 결과, 첫 타워 철거 여부와 첫 드래곤 처치 여부를 나타내는 'firstTower'와 'firstDragon'이 해당 팀의 승리 여부와 크게 연관되어 있는 것을 알 수 있었다. 더불어, 설명 변수 간의 연관성도 확인해보았다. 몇 가지 설명변수들이 서로 유의미한 상관관계를 가지는 것으로 보였는데, 그 중 한가지는 '시야 점수(vision score)'와 '오브젝트 처치 횟수(objectKills)'였다. 경기장 내에 시야 관련 아이템을 잘 배치시켜 상대 팀 구성원의 위치를 조기에 파악하고 적절한 시점에 오브젝트를 처치하거나 상대보다 한발짝 빨리 오브젝트를 처치함으로써 획득한 골드로 빠르게 성장하여 경기장 내의 자유로운 이동을 통해 시야 아이템을 잘 배치시킬 수 있다는 것으로 해석해볼 수

있다. 한편, 골드의 획득양과 소비량 간의 유의미한 양의 상관관계 또한 확인할 수 있었다. 이러한 설명 변수 간의 유의미한 상관관계는 이후 분석 진행 과정에서 다중 공선성 문제를 해결하고 차원의 저주를 피하기 위해 차원 축소를 진행하는 강한 근거가 되었다.

4. 차원 축소 (Dimension Reduction) 및 모델 적합

```
print('==== RandomForest clf with all features ====')
print('train accuracy:', clf.score(X_train,y_train))
print('test accuracy:', clf.score(X_test, y_test))
```

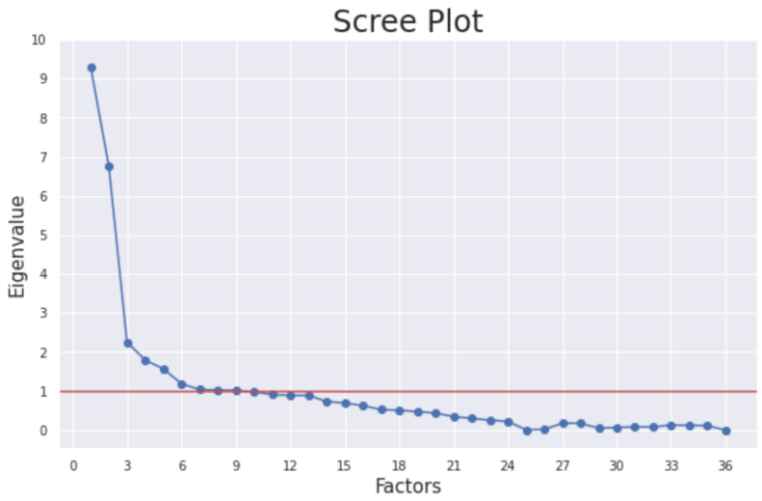
```
0.9840605895973967
{'max_depth': 8, 'min_samples_leaf': 2, 'min_samples_split': 5}
==== RandomForest clf with all features ====
train accuracy: 0.9922874220708272
test accuracy: 0.932905982905983
```



위의 결과는 raw data 의 약 200 개 설명변수를 36 개를 줄인 후, 모든 설명 변수를 이용하여 RandomForest Classfier 를 적합 시켜 정확도와 변수 중요도를 출력한 것으로, 모델의 초모수는 GridsearchCV 를 이용하여 결정하였다. Test accuracy 가 train accuracy 보다 다소 낮은 것을 확인할 수 있는데 이는 너무 많은 설명 변수로 인해 과적합 문제가 발생한 것으로 보였다. 더불어, 변수 중요도를 시각화 했을 때 'inhibitorKills_red'와 'inhibitorKills_blue', 'red_kda'와 'blue_kda' 등, 같은 category 내에서 양 진영의 상태를 표시하는 변수가 함께 높은 중요도를 보였다. 이는 양 진영의 대칭적인 변수가 가지는 유의미한 상관 관계 때문임을 직관적으로 이해할 수 있다. 실제로 아래처럼 VIF factor 를 직접 출력했을 때, 일부가 아주 높은 값을 보이고 있다. 때문에, 이러한 과적합 문제와 다중공선성의 문제를 해결하기 위해 차원 축소를 진행하였다.

	VIF Factor	features
18	inf	totalDamageDealtToChampions_red
33	inf	damage_gap
14	inf	totalDamageDealtToChampions_blue
16	315.580640	totalDamageTaken_blue
17	314.749921	goldSpent_blue
21	192.751964	goldSpent_red
20	192.740920	totalDamageTaken_red
26	106.348402	gold_diff

아래는 주성분 분석을 진행하기에 앞서 적절한 주성분 개수를 파악하기 위해 scree plot 을 그린 것이다. Eigen value 가 1 이상일 때를 기준으로, scree point 가 되는 지점을 확인하였고 결과적으로 n_components=9 로 진행하였다.




```
the number of Eigenvalues over 1 =  9
```

이후, PCA를 통해 9개의 주성분으로 기존의 설명변수를 표현하였고 해당 데이터에 각각 Logistic Regression, RandomForest Classifier, SVM Classifier를 적합시켰다. RandomForest Classifier와 SVM Classifier의 초모수는 GridsearchCV를 이용하여 결정하였다. Accuracy를 비교했을 때, train과 test의 결과가 거의 유사하게 나타나 36개의 전체 설명변수를 사용했을 때 발생한 과적합 문제가 완전히 해결되었음을 확인할 수 있었다.

- Logistic Regression

```
print('==== Logistic Regression ====')
print('train accuracy:', accuracy_score(y_train,y_train_pred))
print('test accuracy:', accuracy_score(y_test,y_test_pred))
```

```
==== Logistic Regression ====
train accuracy: 0.9570666495276046
test accuracy: 0.9599640125955915
```

- RandomForest Classifier

```
print('==== RandomForest Classfier ====')
print('train accuracy:', clf.score(pca_train,y_train))
print('test accuracy:', clf.score(pca_test,y_test)
```

```
0.9541101446085648
{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}
==== RandomForest Classfier ====
train accuracy: 0.9766694517642522
test accuracy: 0.9732163742690059
```

- Support Vector Machine Classifier

```
print('==== SVM Classifier ====')
print('train accuracy:', svc.score(pca_train,y_train))
print('test accuracy:', svc.score(pca_test, y_test))
```

```
0.958416486810611
{'C': 1.0}
==== SVM Classifier ====
train accuracy: 0.9634937978019152
test accuracy: 0.9620632778527516
```

5. Case Study

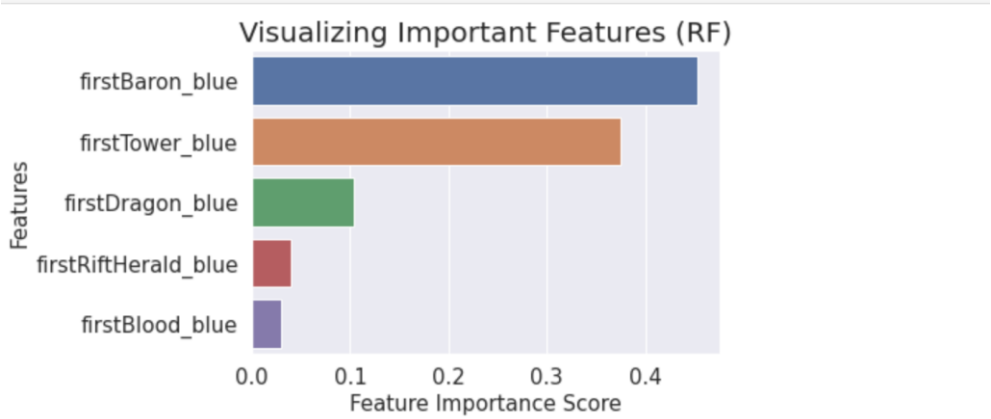
5-1. 경기 초반 중요 지표를 이용한 승패 예측 및 변수 중요도 확인

220개의 raw data variables를 EDA 및 게임 규칙 기반으로 36개까지 축소한 뒤, 과적합과 다중공선성 문제의 해결을 위해 주성분 분석을 진행하였다. 결과적으로 9개의 주성분을 이용하여 target variable, 즉 블루 팀의 승패 여부를 적절하게 예측하고 과적합 문제를 완벽히 해결할 수 있었다. 하지만 90%를 뛰어넘는 정확도를 보면서 해당 분석에 큰 오점이 있음을 알 수 있었다. 지금까지 사용한 변수들은 이미 승패가 결정된 종료된 경기들의 지표이다. 블루 팀이 승리한 경기의 주요 경기 지표들은 당연히 블루 팀의 높은 골드 획득량, 높은 킬 스코어, 많은 오브젝트 처치 횟수를 가리키고 있을 것이다. 이미 끝난 경기의 완성된 지표를 이용하여 승패를 예측하는 것은 승패를 예측하겠다는 보고서의 목적에 적합하지 않을 뿐 아니라 경기에 참여하는 유저들에게도 흥미롭지 않다고 판단하였다. 따라서, 블루 팀의 첫 타워 철거 여부, 첫 오브젝트 처치 여부 등 경기 초기에 결정되는 중요 지표 5개를 이용하여 승패 예측을 진행하였다. 그 결과는 다음과 같다.

- RandomForest Classifier

```
print('==== RandomForest CLF with "early-stage-features"====')
print('train accuracy:', clf.score(X_train,y_train))
print('test accuracy:', clf.score(X_test, y_test))
```

```
0.7732504310101251
{'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split': 2}
==== RandomForest CLF with "early-stage-features"====
train accuracy: 0.7732502088823189
test accuracy: 0.7750322387164492
```

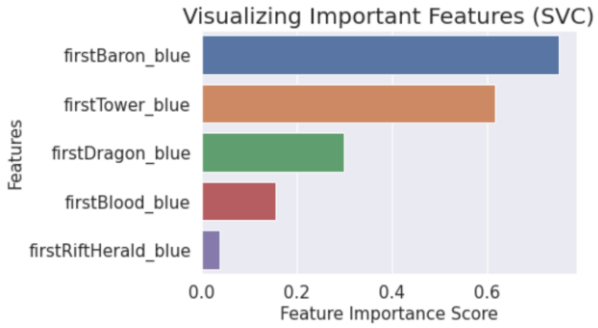


주성분 9개를 이용하여 승패 예측을 진행했을 때보다 정확도가 0.2 정도 떨어졌다. 하지만 경기 초반 지표 5개만을 이용했을 때도 70% 이상의 비율로 승패를 정확히 예측할 수 있다는 사실이 놀라웠다. 다섯 개 변수 중 가장 중요도가 높은 것은 오브젝트 '바론(Baron)'의 첫 처치 여부였고 뒤이어 타워, 드래곤의 철거 및 처치가 중요했다.

- **Support Vector Machine Classifier**

```
print('==== SVM Classifier with "early-stage-features" ====')
print('train accuracy:', svc.score(X_train,y_train))
print('test accuracy:', svc.score(X_test, y_test))
```

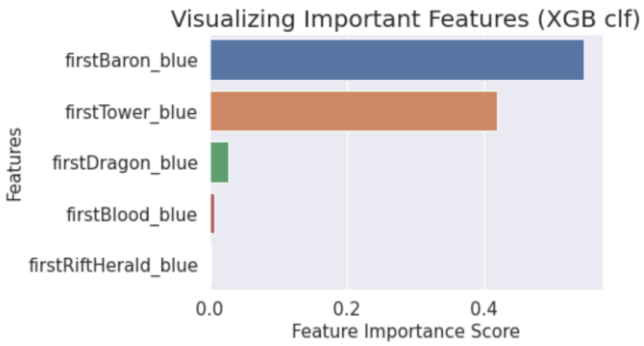
```
0.757118020271438
{'C': 0.01}
==== SVM Classifier with "early-stage-features" ====
train accuracy: 0.7610386271611286
test accuracy: 0.7554355975408606
```



- **XGBoost Classifier**

```
print('==== XGBoost CLF with "early-stage-features" ====')
print('train accuracy:', xg_clf.score(X_train,y_train))
print('test accuracy:', xg_clf.score(X_test, y_test))
```

```
==== XGBoost CLF with "early-stage-features" ====
train accuracy: 0.7732502088823189
test accuracy: 0.7650322387164492
```



SVM Classifier와 XGBoost Classifier를 사용했을 때도 거의 유사한 정확도와 변수 중요도 순위를 보인다. 해당 Case study에서도 GridsearchCV를 이용하여 모형 초모수를 결정하였다.

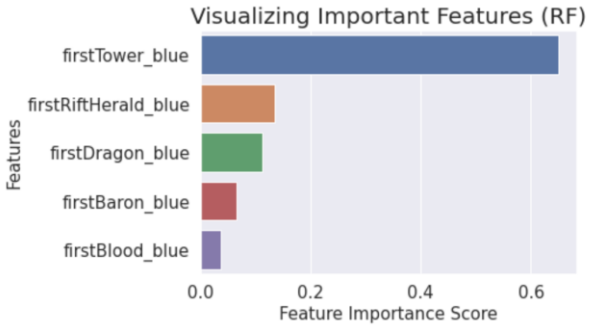
5-2. 조기 종료된 경기의 승패 예측 및 변수 중요도 확인

'리그 오브 레전드'에서는 20분부터 항복 선언을 할 수 있다. 해당 Case study에서는 이러한 항복 선언 가능 시점이 오기도 전에 승패가 결정된, 즉 특정 진영의 압도적인 우세로 조기 종료된 경기들의 승패 예측을 첫 번째 Case study에서 사용한 경기 초반 중요 지표 5개를 이용하여 진행하였다.

- **RandomForest Classifier**

```
print('==== RandomForest CLF with "early-stopped Games"====')
print('train accuracy:', clf.score(X_train,y_train))
print('test accuracy:', clf.score(X_test, y_test))
```

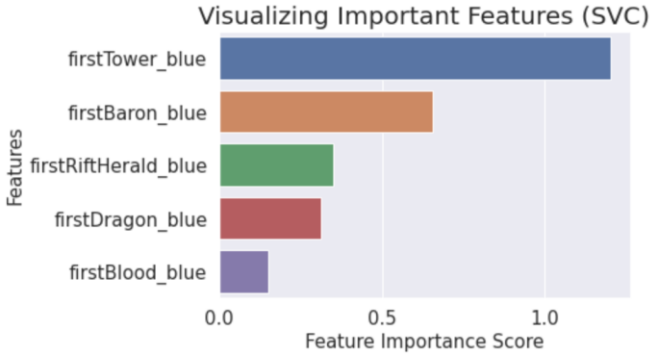
```
0.8914877497460341
{'max_depth': 6, 'min_samples_leaf': 4, 'min_samples_split': 2}
==== RandomForest CLF with "early-stopped Games"====
train accuracy: 0.8964764577486748
test accuracy: 0.8981818181818182
```



- **Support Vector Machine Classifier**

```
print('==== SVM Classifier with "early-stopped Games" ====')
print('train accuracy:', svc.score(X_train,y_train))
print('test accuracy:', svc.score(X_test, y_test))
```

```
0.8896174255565079
{'C': 0.1}
==== SVM Classifier with "early-stopped Games" ====
train accuracy: 0.892422825070159
test accuracy: 0.8945454545454545
```



경기 초반 중요 지표를 이용한 예측 결과, '조기 종료된 경기'의 승패 여부는 전체 경기의 승패 여부에 비해 더 높은 정확도(70% -> 80%)로 예측될 수 있었다. 즉, 경기 초반 중요 지표가 항복 선언 가능 시점 이전의 압도적인 우세에 큰 영향을 미친다고 유추할 수 있다. 변수 중요도 순위에도 다소 변화가 보였다. '바론'의 첫 처치여부가 가장 중요했던 첫 번째 Case Study와 달리, '타워'의 첫 철거 여부가 더 중요한 것으로 나타났다. 이는 경기장 양 극단의 넥서스를 파괴함으로써 경기가 끝나는 게임 특성상, 넥서스에 이르는 길을 지키는 타워를 파괴하는 것이 경기 전반의 우세한 흐름을 결정하는 바론의 처치보다 해당 환경에서 중요하게 작용하는 것으로 해석할 수 있다.

6. 승리 확률 시뮬레이터

몇 가지 Case Study까지 진행했을 때 머릿속에 떠오른 것은, 승패 예측에 있어서 게임을 플레이하는 유저들이 가장 관심을 가질 요소가 무엇인지에 대한 의문이었다. 플레이어들은 자신이 처한 환경에서 어떤 행동을 취해야 팀의 승리 확률을 최대로 높일 수 있을지 혹은 특정 환경에서 어떤 행동을 취할 때의 대략적인 승리 확률에 관심이 있다. 때문에, 다양한 경기 내 상황을 가정하고 그 때마다의 승리 확률을 계산할 수 있는 승리 확률 계산 프로그램을 구현하였다. 해당 프로그램의 실행 과정은 다음과 같다.

```
현재 경기 상황을 입력하세요.

입력할 수 있는 경기 상황은 다음과 같습니다.

 firstBlood_ours
 firstTowerKill_ours
 firstInhibitorKill_ours
 firstBaron_ours
 firstDragon_ours
 firstRiftHerald_ours
 4dragons_ours
 olddragon_ours
 kda_ours
 kda_enemy
 level_diff_ours_minus_enemy
 inhibitorKills_gap_ours_minus_enemy
 baronKills_gap_ours_minus_enemy
 dragonKills_gap_ours_minus_enemy
 riftHeraldKills_gap_ours_minus_enemy

경기 상황 입력을 마치면 '입력완료'라고 입력해주세요.

조건을 주고자 하는 경기 상황
 firstTowerKill_ours
값을 입력해주세요.(ex) True/False)
 True
조건을 주고자 하는 경기 상황
 firstBaron_ours
값을 입력해주세요.(ex) True/False)
 True
조건을 주고자 하는 경기 상황
 입력완료
```

매 실행마다 입력할 수 있는 경기 상황이 나타나고 실행자는 자신이 가정하고 싶은 상황을 원하는 대로 입력할 수 있다. 왼쪽의 실행 예시는 첫 타워 철거와 오브젝트인 드래곤 처치를 아군이 해냈을 때, RandomForest Classifier와 GLM 모형 적합을 통해 예측한 아군의 승리 확률 및 정확도와 신뢰구간을 출력한 모습이다. 이처럼 유저들은 다양한 상황을 가정하고 여러 옵션을 추가로 입력해보면서 가장 높은 승리 확률을 보여주는 행동을 찾아나갈 수 있다. 즉, 평균 경기 플레이 시간인 약 26분 대신 1분이 채 되지 않는 시간을 할애하여 다양한 시뮬레이션을 진행해볼 수 있는 것이다.

```
총  16671 개의 경기를 학습하고  5557 개의 경기를 예측해 모델의 성능을 측정합니다.

랜덤포레스트를 사용했을 때, 경기 승리 확률은  85.03 % 입니다.
사용된 랜덤포레스트 모델의 성능은 정확도 74.97 % 입니다.

Generalized Linear Model을 사용했을 때, 경기 승리 확률은  90.55 % 입니다.
승리 확률의 95% 신뢰구간은 ( 89.8 %,  91.24 %)  입니다
사용된 GLM모델의 성능은 정확도 74.97 % 입니다.
```

7. 결론

온라인 게임 '리그 오브 레전드'의 최상위 랭크 경기들을 이용하여 승패 예측을 진행하였다. 해당 과정에서 과적합과 다중공선성 문제를 해결하기 위해 차원 축소를 진행하였다. 결과적으로 높은 정확도를 유지하면서도 다중공선성 문제를 해결할 수 있었지만 분석 목적에 적합한 Case Study 를 추가로 진행하였고 경기 초반 중요 지표만을 이용해서도 높은 확률로 승패 여부 예측이 가능하며 해당 지표가 압도적인 우세로 인한 경기의 조기 종료에 큰 영향을 미친다는 사실을 발견하였다. 더불어, 유저들에게 가장 필요한 부분을 고민하였고 그 결과, 다양한 시뮬레이션을 통해 승리 확률을 계산 및 승패 예측을 진행할 수 있는 프로그램을 구현하였다. 다음에는 전체 경기 지표가 아닌, 경기 내에서 분당으로 갱신된 지표를 이용하여 경기에 참여하는 유저들이 승패를 예측하고 더 많은 환경 하에서 적절한 행동을 찾을 수 있도록 돕는 프로그램을 만들어보고 싶다.

[부록]

1. Data Loading 과정

'리그 오브 레전드' 승/패 기록 및 경기 지표 데이터를 얻기 위해서 게임 제작사 홈페이지에 접속하여 Riot API를 이용하였다. 제작사 'Riot Games'는 유저들에게 그들의 데이터 베이스에 접근할 수 있는 툴을 제공한다. 유저들은 'developer.riotgames.com'에서 API key를 얻을 수 있다. Riot API를 이용하여 json format으로 데이터를 받아오고, 그것을 pandas data frame 형태로 변형시켰다. 다음은 전체 과정을 진행한 코드이다. 해당 ipynd를 pdf로 변형하는 데에 어려움이 있어 캡처 도구를 이용하여 첨부하였다.

In [1]:
```
import requests
import pandas as pd
import numpy as np
import time

api_key =  'RGAPI-4b716c74-5faf-4cff-88a4-617f81317dc5'
sohwan = "https://kr.api.riotgames.com/lol/summoner/v4/summoners/by-name/" +'hide on bush' +'?api_key=' + api_key
r = requests.get(sohwan)
r
```

Out[1]: <Response [200]>

In [2]:
```
challenger = 'https://kr.api.riotgames.com/lol/league/v4/challengerleagues/by-queue/RANKED_SOLO_5x5?api_key=' + api_key
r = requests.get(challenger)#챌린저데이터 호출
league_df = pd.DataFrame(r.json())

league_df.reset_index(inplace=True)#수집한 챌린저데이터 index정리
league_entries_df = pd.DataFrame(dict(league_df['entries'])).T #dict구조로 되어 있는 entries컬럼 풀어주기
league_df = pd.concat([league_df, league_entries_df], axis=1) #옆끼리 결합

league_df = league_df.drop(['index', 'queue', 'name', 'leagueId', 'entries', 'rank'], axis=1)
league_df.info()
league_df.to_csv('챌린저데이터.csv',index=False,encoding = 'cp949')
```

In [5]:
```
for i in range(len(league_df)):
    try:
        sohwan = 'https://kr.api.riotgames.com/lol/summoner/v4/summoners/by-name/' + league_df['summonerName'].iloc[i] + '?api_key=' + api_key
        r = requests.get(sohwan)

        while r.status_code == 429:
            time.sleep(10)
            sohwan = 'https://kr.api.riotgames.com/lol/summoner/v4/summoners/by-name/' + league_df['summonerName'].iloc[i] + '?api_key=' + api_key
            r = requests.get(sohwan)

        account_id = r.json()['accountId']
        league_df.iloc[i, -1] = account_id

    except:
        pass
```

In [11]:
```
match_info_df = pd.DataFrame()
season = str(13)
for i in range(len(league_df2)):
    try:
        match0 = 'https://kr.api.riotgames.com/lol/match/v4/matchlists/by-account/' + league_df2['account_id'].iloc[i]  +'?season=' + season + '&api_key=' + api_key
        r = requests.get(match0)

        while r.status_code == 429:
            time.sleep(5)
            match0 = 'https://kr.api.riotgames.com/lol/match/v4/matchlists/by-account/' + league_df2['account_id'].iloc[i]  +'?season=' + season + '&api_key=' + api_key
            r = requests.get(match0)

        match_info_df = pd.concat([match_info_df, pd.DataFrame(r.json()['matches'])])

    except Exception as e:
        print(i)
        print(e)
```

```
match_fin = pd.DataFrame()



for i in range(16601, len(match_info_df)):

    api_url='https://kr.api.riotgames.com/lol/match/v4/matches/' + str(match_info_df['gameId'].iloc[i]) + '?api_key=' + api_key
    r = requests.get(api_url)

    if r.status_code == 200: # response가 정상이면 바로 맨 밑으로 이동하여 정상적으로 코드 실행
        pass

    elif r.status_code == 429:
        print('api cost full : infinite loop start')
        print('loop location : ',i)
        start_time = time.time()

        while True: # 429error가 끝날 때까지 무한 루프
            if r.status_code == 429:

                print('try 10 second wait time')
                time.sleep(10)

                r = requests.get(api_url)
                print(r.status_code)

            elif r.status_code == 200: #다시 response 200이면 loop escape
                print('total wait time : ', time.time() - start_time)
                print('recovery api cost')
                break
```

```
    elif r.status_code == 403: # api갱신이 필요
        print('you need api renewal')
        print('break')
        break

    # 위의 예외처리 코드를 거쳐서 내려왔을 때 해당 코드가 실행될 수 있도록 작성
    mat = pd.DataFrame(list(r.json().values()), index=list(r.json().keys())).T
    match_fin = pd.concat([match_fin,mat])


    if i%100 == 0:
        match_fin.to_csv('챌린저데이터(kr_match_fin)tmp4.csv',index=False,encoding = 'cp949')
        f = open('last_saved_index(challenger).txt', 'a')
        data = '₩nlast saved index in loop  : {}'.format(i)
        f.write(data)
        f.close()
```

```
challenger_tot = pd.concat([data1, match_fin])
challenger_tot.drop_duplicates(subset='gameId', inplace=True)
challenger_tot.to_csv('match_fin(챌린저데이터).csv', index=False, encoding='cp949')
```

```
challenger_tot.head()
```

	gameId	platformId	gameCreation	gameDuration	queueId	mapId	seasonId	gameVersion	gameMode	gameType	teams	participants
0	4.42418e+09	KR	1.59112e+12	2169	420	11	13	10.11.322.2991	CLASSIC	MATCHED_GAME	[{'teamId': 100, 'win': 'Win', 'firstBlood': T...	[{'participant 1, 'teamId': 100, 'champion...

```
stat_df = pd.DataFrame()
key_error_ls = []



for j in range(match_fin.shape[0]):
    blue_ply_df = pd.DataFrame()
    red_ply_df = pd.DataFrame()


    try :
        for i in range(10):
            if i<=5.5 :
                one_ply = pd.DataFrame(pd.DataFrame(match_fin.participants.values[j]).stats[i], index=[i])
                blue_ply_df = pd.concat([blue_ply_df, one_ply])
            else :
                one_ply = pd.DataFrame(pd.DataFrame(match_fin.participants.values[j]).stats[i], index=[i])
                red_ply_df = pd.concat([red_ply_df, one_ply])

    except KeyError :
        key_error_ls.append(j)
        continue

    blue_ply_df.drop(['item0', 'item1','item2', 'item3', 'item4', 'item5', 'item6'], axis=1, inplace=True)
    red_ply_df.drop(['item0', 'item1','item2', 'item3', 'item4', 'item5', 'item6'], axis=1, inplace=True)

    blue_tot_stat = pd.DataFrame(blue_ply_df.apply(sum)).T
    tmp_arr = np.array(blue_tot_stat.columns) +'_blue'
    blue_tot_stat.columns = tmp_arr.tolist()

    red_tot_stat = pd.DataFrame(red_ply_df.apply(sum)).T
    tmp_arr2 = np.array(red_tot_stat.columns) +'_red'
    red_tot_stat.columns = tmp_arr2.tolist()


    blue_team_stat = pd.DataFrame(pd.DataFrame(match_fin.teams.values[j]).iloc[0]).T
    blue_stat_index = blue_team_stat.columns.tolist()
    blue_team_stat.columns = [x +'_blue'for x in blue_stat_index]

    red_team_stat = pd.DataFrame(pd.DataFrame(match_fin.teams.values[j]).iloc[1]).T
    #red_team_stat.reset_index(inplace=True)
    red_stat_index = red_team_stat.columns.tolist()
    red_team_stat.columns = [x +'_red'for x in red_stat_index]

    team_tot_stat = pd.concat([blue_team_stat, red_team_stat.reset_index(drop=True)], axis=1)
    ply_tot_stat = pd.concat([blue_tot_stat, red_tot_stat.reset_index(drop=True)], axis=1)

    all_stat = pd.concat([team_tot_stat, ply_tot_stat], axis=1)
    all_stat.index = [j]
```

```
    try :
        stat_df = pd.concat([stat_df, all_stat])

    except :
        pass
        #print('fail to concat: the number of row in match_fin is ',j, 'problem is ',tmp_ls)
```

2. 전체 분석 코드

분석을 위해 실행한 전체 코드는 아래와 같다.

(ipynd 파일 pdf 변환 시 깨짐 및 한글 출력 문제 발생)