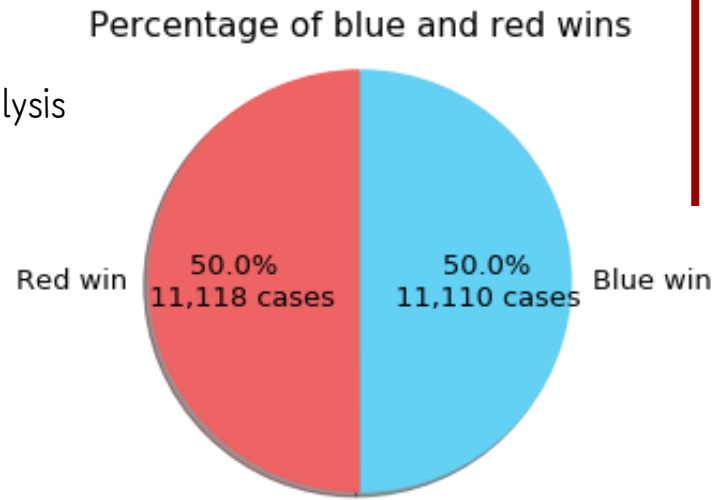
Team 1.
League of Legends Win/Loss Prediction

INDEX

Data Description



- Online game 'League of Legends' Top Tier Rank Match Indicators Analysis
  [latest Season(this year), Cahllenger, Grand Master, Master]

- Direct collection using Riot API (approximately 20,000 games data)

- Target variable: Blue Team win

  - Objects : Provides positive effects to teams when killed
    (Dragon, Baron, RiftHeralds)

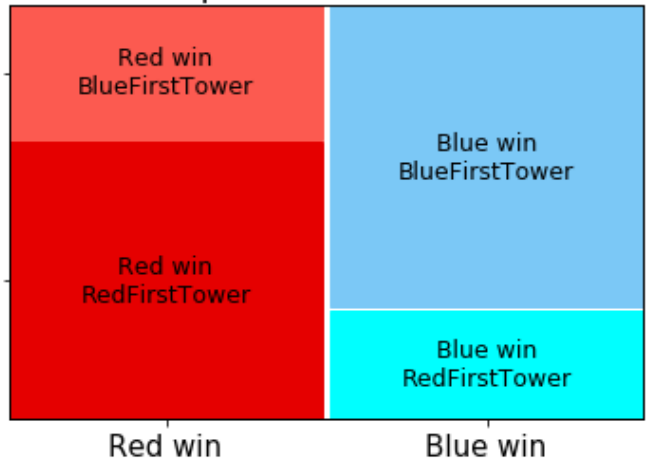
  - Tower / Inhibitors : Structures that must be removed prior to Nexus destruction

  - Performance : Indicators that describe each team's capabilities
    (Kill, Death, Assist, Heal, Level)

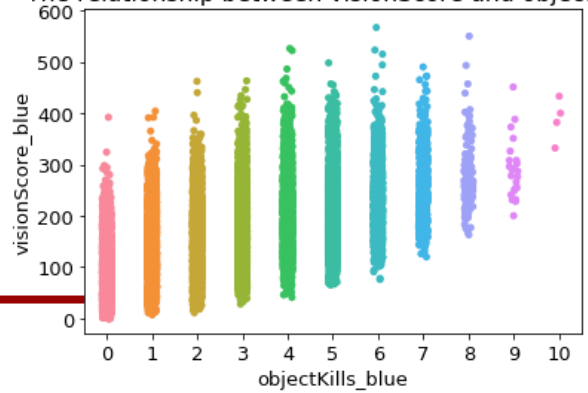
  - Resource : Degree of resource supply to help the team grow (Gold, Minion Kill etc.)

Visualization

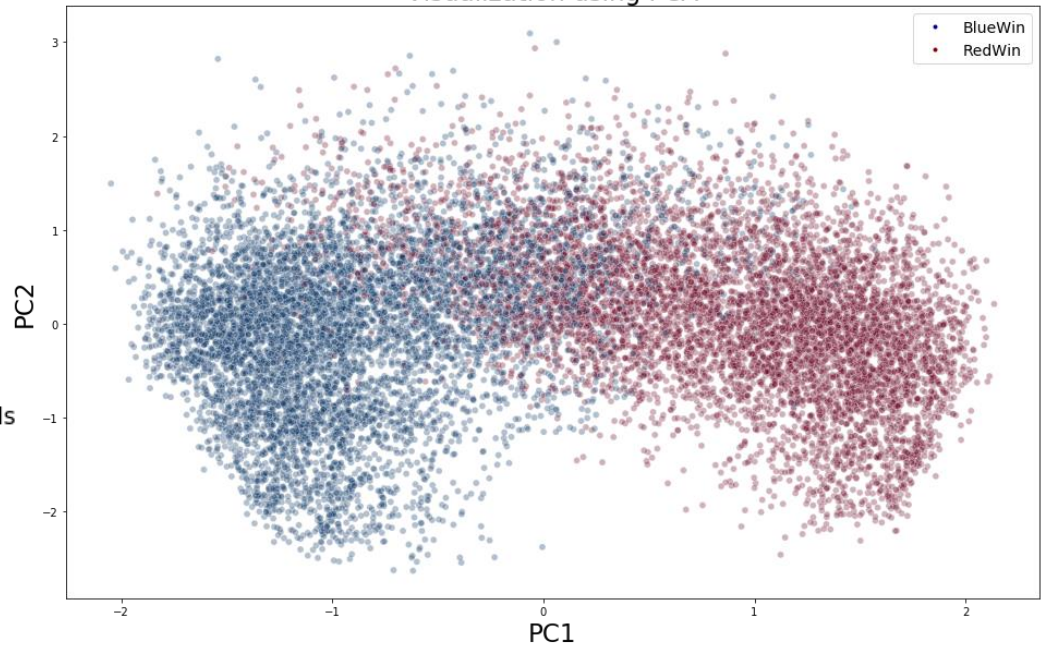
The relationship between firstTower and win



The relationship between visionScore and objectKills



Visualization using PCA



Feature Engineering

1. Feature engineering by rules and variable characteristics

- Add features : blue/red 4 dragon / elder dragon
- Merge features : kda / gap features
- Zero features

2. EFA (Exploratory Factor Analysis)

	VIF Factor	features
18	inf	totalDamageDealtToChampions_blue
36	inf	red4dragons
35	inf	blue4dragons
30	inf	totalDamageDealtToChampions_red
46	inf	damage_gap
20	371.965744	totalDamageTaken_blue
21	338.817946	goldSpent_blue
32	218.679486	totalDamageTaken_red
33	212.813856	goldSpent_red
39	146.853861	gold_diff
40	100.594401	level_diff

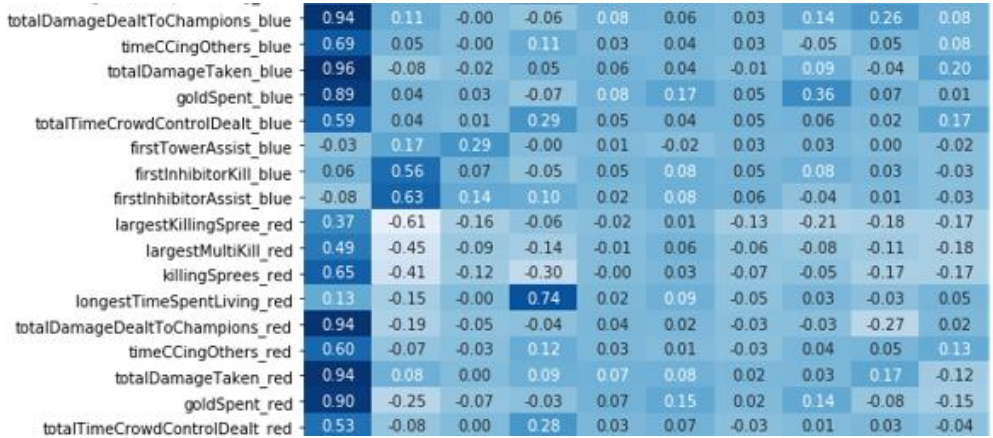
- higth multi collinearity
: exaplanatory variables are highly correlated

- Understanding potential relationships between variables

- Finding latent variables

Feature Engineering

2. EFA (10 variables)



	0	1	2	3	4	5	6	7	8	9
SS Loadings	10.338418	6.698949	2.683211	2.581737	2.201519	1.742404	1.540786	1.407341	1.024785	0.632525
Proportion Var	0.210988	0.136713	0.054759	0.052689	0.044929	0.035559	0.031445	0.028721	0.020914	0.012909
Cumulative Var	0.210988	0.347701	0.402461	0.455149	0.500078	0.535637	0.567082	0.595803	0.616717	0.629626

cumulative variance : 0.63

Factor 0
: team performance
(KDA, damage, heal, level)

Factor 1
: structure destruction, resource
(first Tower/ Inhibitor destruction, gold gap)

Factor 2
: tower desstruction, riftherald kill (Related to tower destruction

Factor 3
: survival time

Factor 4
: Dragon Buff (kill 4 Dragons)

etc (objects kill, liner gap, damage gap).

Model Fit

1. Random Forest

- Use full variables
  (Reduced only by game rules and variable characteristics)

```
predicted = rf.predict(X_test)
accuracy = accuracy_score(y_test, predicted)

print(f'Mean accuracy score: {accuracy:.3}')
```

```
Mean accuracy score: 0.985
```

- Use factors through EFA

```
print(f'Mean accuracy score: {accuracy:.3}')
```

```
Mean accuracy score: 0.962
```

- Important Features

tower_gap	0.199199
inhibitorKills_red	0.151826
inhibitorKills_blue	0.143042
red_kda	0.109693
firstInhibitor_blue	0.092656
blue_kda	0.082312
gold_diff	0.042423
firstInhibitorAssist_blue	0.035262
level_diff	0.020915
damage_gap	0.013179
dragonKills_red	0.010560
firstInhibitorKill_blue	0.009612

1	0.671168
4	0.095004
5	0.069260
0	0.037171
6	0.030220
2	0.028993
7	0.024682
3	0.017729
8	0.013677
9	0.012095

Model Fit

2. GLM (Logistic Regression with L2 penalty)

- Use full variables
  (Reduced only by game rules and variable characteristics)

```
y_pred = log_reg.predict(X_test)

print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.9802051466618679
```

- Use factors through EFA

```
y_pred = log_reg.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.9780457081158899
```

3. SVM

- Use full variables
  (Reduced only by game rules and variable characteristics)

```
y_pred = svm.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.8931077919740867
```

- Use factors through EFA

```
y_pred = svm.predict(X_test)
print('정확도 :', metrics.accuracy_score(y_test, y_pred))
```

```
정확도 : 0.9625697318697138
```

Model Fit

2. GLM (Logistic Regression with L2 penalty)

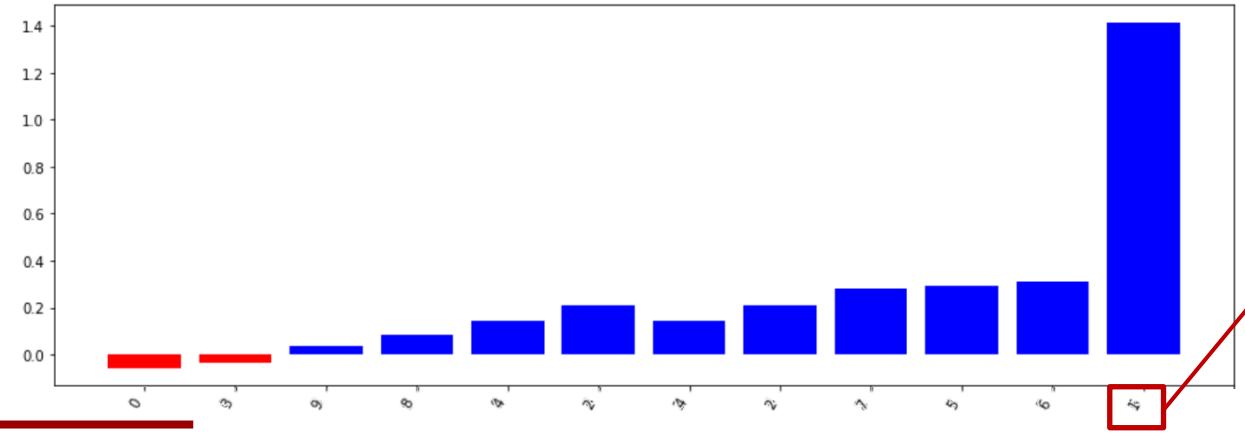
```
coef = log_reg.coef_.reshape(-1)

print(coef.shape)

for feature, w in sorted(enumerate(abs(coef)), key=lambda x:-x[1])[:100]:
    print(X_train.columns[feature], end=' ')
    print(w)
```

```
1 5.129382752073308
6 1.1559080708422291
7 1.068465203885858
5 1.0646265180677654
2 0.7945769613923964
4 0.5074838447520851
8 0.3181102882501763
0 0.20500420706714262
9 0.16235970145253947
3 0.13122861143754494
```

3. SVM

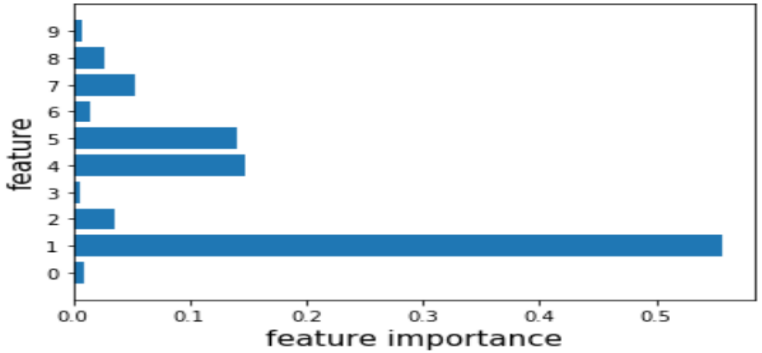


Factor 1

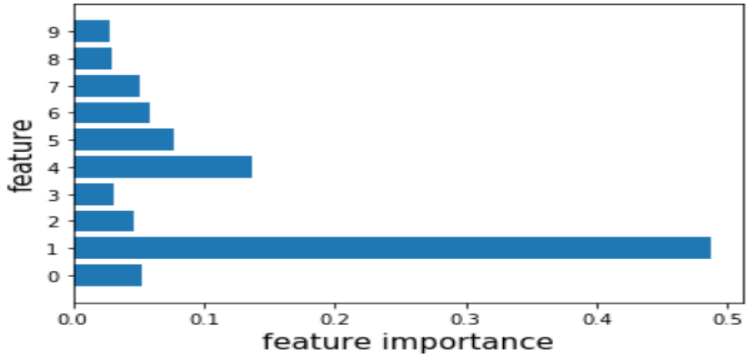
Case Study

1. Depending on game duration

- Short Games (15 ~ 25 min)



- Long Games (30 min ~)



2. An adverse situation

tower_gap	0.205750
inhibitorKills_blue	0.137936
inhibitorKills_red	0.132103
red_kda	0.096596
firstInhibitor_blue	0.056853

Win probability prediction program

- Implement a model that predicts the probability of winning according to the given conditions.

```
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
```

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

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

랜덤포레스트를 사용했을 때, 경기 승리 확률은  34.36 % 입니다.
사용된 랜덤포레스트 모델의 성능은 정확도 70.0 % 입니다.

Generalized Linear Model을 사용했을 때, 경기 승리 확률은  34.15 % 입니다.
승리 확률의 95% 신뢰구간은 ( 32.79 %,  35.55 %)  입니다
사용된 GLM모델의 성능은 정확도 70.0 % 입니다.
```