빅데이터 분석 및 시각화 개론

리플레이 데이터 분석을 통한 최적의 전략 도출:

히어로즈 오브 더 스톰의 사례

201411162 허재형

201411029 김승년

201411007 권오현

1. Introduction

사람들은 게임에서 이기기 위해 많은 노력을 기울인다. 게임에서 이기는 것을 통해 사람들은 재미를 느끼고, 우월감을 느낄 수 있기 때문이다. 우리는 게임의 주 소비층인 대한민국의 20대 남성으로서, 우리가 가진 이 취미에 대한 흥미와 열정을 사용할 수 있는 방법을 찾고자 했다. 우리는 사람들이 게임에서 이기는 것을 돕기 위한 솔루션을 제시하고자 한다. 게임에서 얻을 수 있는 데이터를 분석한 뒤, 플레이어가 이길 수 있는 최적의 방법을 제시하고자 한다. 우

리가 이 분석에서 사용한 게임은 히어로즈 오브 더 스톰이라는 게임이다. 본 게임은 미국의 게임 개발사 블리자드 엔터테인먼트에서 만들어진 게임으로, 5대 5로 나뉘어 상대 팀을 이기는 것이 목표인 게임이다. 각 팀에 속하는 플레이어는 다양한 캐릭터 중 자신이 원하는 한 캐릭터를 선택한뒤 게임에 임한다. 여러 종류의 맵이 게임에 존재하며, 각 맵별로 다른 특정 목표를 먼저 만족시킨 팀이 이긴다. 5대 5 팀전이라는 요소는 각 플레이어들에게 이기기 위한 동력을 부여해 준다.

게임에서 이기기 위한 사람들의 관심과 노력은 게임 전적 분석 사이트를 탄생시켰다. 우리나라에서 지난 몇년간 유행했던 게임인 오버워치, 리그 오브 레전드, 배틀그라운드 등의 게임들은 모두 전적 분석 사이트가 개설되어 있다. 전적 분석 사이트는 각 플레이어의 전적을 분석해 주는 한편, 전세계 플레이어들의 데이터를 통계적으로 분석하는 역할도 한다. 우리가 분석 대상으로 삼은 히어로즈 오브 더 스톰 역시 이러한 전적 분석 사이트가 존재한다. 한 가지 특이한 점은 히어로즈 오브 더 스톰의 전적 분석 사이트는 다른 전적 분석 사이트와는 다르게 데이터 분석을 위한 RAW DATA를 제공하고 있다는 점이다. 이 데이터는 다운받은 시점 1달 전까지 전 세계의 플레이 데이터로, 한달동안 전 세계에서 진행된 게임들의 결과에 대한 다양한 정보를 담고 있다. 승패부터 다양한 수치까지 존재하는 이 데이터를 잘 이용해, 플레이어들이 이길 수 있는 방법을 제시할 것이다.

2. Analysis

1) 기본가정

	A	B	C	D
1	153846.2		승률	0.5
2	76923.08	기댓값	패배율	0.5
3	38461.54	분산		
4	196.1161	표준편차		
5	1177	6시그마		
6	75746	0.492351		
7	78100	0.507649		

팀 게임의 특성 상, 개별 영웅의 승률은 수 많은 게임을 통해 50%에 수렴한다고 볼 수 있을 것이다. 이기는 팀이 있다면 지는 팀도 있을 것이기 때문이다. 우리가 얻은 데이터는 총 950만 건의 게임에 대한 정보를 가지고 있고, 히어로즈 오브 더 스톰에는 65명의 캐릭터가 존재한다. 따라서 개별 영웅의 평균 승률이 50%라고 가정한다면, 우리가 뽑은 표본에서 영웅들의 승률은 6시그마 범위 안에서 49.2%~50.8% 내에 분포해야 한다고 볼 수 있을 것이다. 만약 우리가 어떤 가설을 세운뒤, 그 가설에 대한 분석 결과가 이 범위를 벗어나는 경우, 분석된 데이터에 대해 우리가 세운 가설이 검증되며 그때의 승률은 평균을 넘어서는 유의미한 결과라고 볼 수 있을 것이다.

2) 숙련도와 승률간의 관계

우리가 분석할 히어로즈 오브 더 스톰(이하 히오스)은 개별 플레이어가 한 캐릭터를 선택해 게임을 플레이 한다. 플레이어는 캐릭터를 고른 뒤 게임을 플레이 한다. 한 게임이 끝나면, 플레이어는 그 캐릭터에 대해 경험치를 얻는다. 얻어진 경험치는 누적되어 캐릭터의 레벨을 올린다. 특정한 캐릭터도 많은 게임을 플레이하면 할 수록 그 캐릭터에 대한 플레이어의 레벨은 높아질 것이다. 그렇기에 개별 플레이어의 특정 캐릭터에 대한 레벨이 높다면, 그 플레이어는 그 캐릭터를 많이 플레이해보았고, 많이 플레이해보았다는 것은 그 캐릭터를 많이 다루어 보았으며 그 캐릭터에 더 익숙해졌다는 것을 의미한다. 또한 더 나아가서 일반적으로 많이 경험해본다면 숙련도가 높아지고, 캐릭터를 더 잘 다루게 되어 게임을 좀 능숙하게 이끌고 이겨나갈 것이라고 생각할수 있다. 우리는 이러한 생각에서 출발하여, '게임을 많이 플레이 해 경험이 있는 사람의 승률이 더 높을 것이다" 라는 가설을 세운 뒤, 이 가설을 검증하고자 했다.

2-1) 숙련도와 승률의 관계 : 분석

숙련도와 승률의 상관관계에 대해 분석하기 위해, 게임별 리플레이가 존재하는 replay_character.csv 파일을 불러온다. 그 뒤, 파일에서 is winner(승리여부) 랑 hero level(숙련도) 칼럼을 따로 묶어 새로운 데이터프레임을 만든다.

```
In [7]: # 일단 Replay_character 파일에서 is winner 랑 Hero Level이랑 묶기
        Rep_Char01 = Rep_Char1[['Is Winner','Hero Level']]
        Rep_Char01.head()
```

이제 분석을 위해 전체 데이터 중에서 이긴 팀과 진 팀의 총 캐릭터 레벨을 합친 뒤, 캐릭터 레벨 합이 더 높은 쪽이 이겼던 경우를 세 보기로 한다. 각 리플레이 결과에 대해 이긴 팀과 진 팀의 캐릭터 레벨을 모두 더한다. 그런 뒤, 이긴 팀의 캐릭터 레벨 합이 더 높은 경우를 count한다. 시범적으로 1만 건의 데이터에 대해 분석해 본 결과는 이긴 팀쪽의 캐릭터 레벨 합이 높은 경우가 샘플로 뽑은 1만 건 중 5282건으로 52.82%였다. 6시그마를 벗어나는 유의미한 분석 결과라 볼 수 있다.

```
In [5]: h=0;
        for i in range(0,10000):
            b = 10*i + 10;
            a = Rep_Char01[10*i:b]
            a1 = a[a['Is Winner'] == 0]
            a2 = a1['Hero Level'].sum()
            a3 = a[a['Is Winner'] == 1]
            a4 = a3['Hero Level'].sum()
            if a2<a4:
                h = h+1
            else:
                h = h
        h
Out[5]: 5282
```

이후 더 많은 행을 분석해 본 결과, 950000 건의 데이터 중 510475건으로 53.73% 였다.

```
In [14]: h=0;
         for i in range(0,950000):
             b = 10*i + 10;
             a = Rep_Char01[10*i:b]
             a1 = a[a['Is Winner'] == 0]
             a2 = a1['Hero Level'].sum()
             a3 = a[a['Is Winner'] == 1]
             a4 = a3['Hero Level'].sum()
             if a2<a4:
                 h = h+1
             else:
                 h = h
             if i%10000==0:
                 print("now ongoing...",i)
         h
```

즉 우리의 가설이 맞았다는 것을 알 수 있다. 캐릭터 레벨의 합이 더 높은 쪽이 이길 확률이 53%였다.

3) 적에게 가한 피해량과 승률의 관계

히오스에서 적 팀을 이기기 위해서는 맵별로 세부 사항은 다를 수 있지만 크게는 적 캐릭터를 공격해 제거하고(제거된 캐릭터는 일정 시간 후 부활한다), 적의 구조물을 공격해 파괴해야 한다. 적 캐릭터에게 더 많은 피해를 입힐 수록, 적 캐릭터가 살아서 아군을 방해하는 일이 적어질 것이다. 또한 적 구조물에 대한 공격을 통해 게임을 이기게 된다. 이러한 사실로부터 우리는 적에게 가한 피해량이 높을수록 승률이 높을 것이라는 가설을 세우게 되었다. 우리가 얻은 replay_character.csv 데이터에는 적에게 가한 피해량이 3가지로 나뉘어 있다. 영웅에 대한 피해량, 구조물에 대한 피해량, 그리고 합산 피해량이다. 우리는 이 각각의 피해량에 대해 피해량과 승률 간의 상관관계를 알아보고자 하였다.

3-1) 적에게 가한 피해량과 승률의 관계 :분석

```
In [15]: k1 = k.head(1000)
         k1.describe()
```

Out[15]:

	loser Hero Damage	winner Hero Damage	loser siege Damage	winner siege Damage	loser total Damage	winner total Damage
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	179177.048000	217697.836000	241209.252000	245637.640000	276472.771000	284158.428000
std	67335.021804	77805.853609	90884.840496	90686.647266	104858.136914	101046.922747
min	20277.000000	37578.000000	56389.000000	54205.000000	63644.000000	71506.000000
25%	128440.250000	160318.750000	173999.250000	177947.500000	200118.250000	208957.750000
50%	169695.500000	205944.500000	229175.000000	232666.500000	260906.500000	270730.500000
75%	219425.250000	263204.250000	290575.250000	297276.000000	333228.750000	338802.250000
max	473458.000000	542962.000000	641930.000000	643446.000000	728436.000000	728571.000000

위에서 언급한 세 가지 분야의 피해량에 대해 이긴 팀과 진 팀별로 구분하여 합산한 결과는 위와 같다.

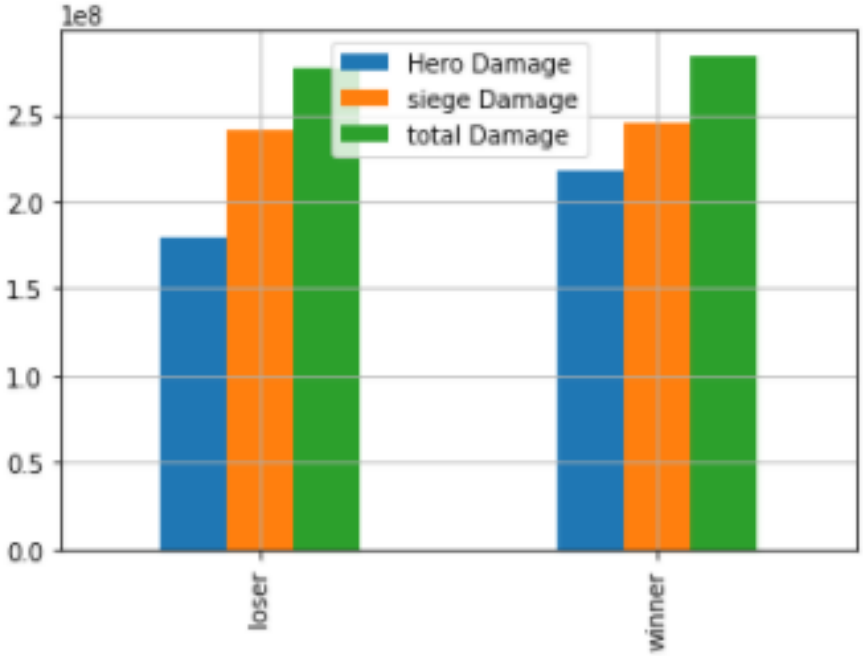
```
In [29]: k3 = pd.DataFrame(np.random.randn(2,3))

         k3.columns = ['Hero Damage', 'siege Damage', 'total Damage']

         k3['Hero Damage'].iloc[0] = k1['loser Hero Damage'].sum()
         k3['siege Damage'].iloc[0] = k1['loser siege Damage'].sum()
         k3['total Damage'].iloc[0] = k1['loser total Damage'].sum()
         k3['Hero Damage'].iloc[1] = k1['winner Hero Damage'].sum()
         k3['siege Damage'].iloc[1] = k1['winner siege Damage'].sum()
         k3['total Damage'].iloc[1] = k1['winner total Damage'].sum()


         k3.index=['loser','winner']
         k3.head()
```

Out[29]:

	Hero Damage	siege Damage	total Damage
loser	179177048.0	241209252.0	276472771.0
winner	217697836.0	245637640.0	284158428.0

리플레이 데이터들의 모든 피해량을 합산한 결과는 위와 같고, 이를 도식화하면

```
In [31]: k3.plot(kind='bar')
         plt.grid()
```



과 같이 나온다.

3-2) 적에게 가한 피해량과 승률의 상관관계에 관한 분석 결과

위 그래프에서 볼 수 있듯이, 이긴 팀과 진 팀의 딜량 차이는 현저하다고 볼 수 있다. 특히 이러한 현상은 구조물 피해량보다 영웅 피해량에서 더 두드러지게 드러난다. 이긴 팀의 영웅 피해량이 진 팀의 영웅 피해량보다 20%정도 더 높은 것을 볼 수 있다. 이를 통해, 우리는 영웅을 직접 타격한 피해량이 구조물에 대한 피해량보다 승률에 좀 더 영향을 미친다고 볼 수 있을 것이다. 2)번의 분석 결과와 비교해 도출해 보자면, 개인이 손에 익은 캐릭터를 플레이해 이기는 것 보다는 단순하게 피해를 더 많이 입힐 수 있는 캐릭터를 골라 플레이 하는 것이 이기는데에는 더 도움이 될 것이라는 생각을 할수 있을 것이다. 아울러 만약 승패를 통계적으로 분석한다면, 통계 분석에 관한 모형을 만들 때 피해량을 중요한 변수로 고려해야 할 것을 의미하기도 한다.

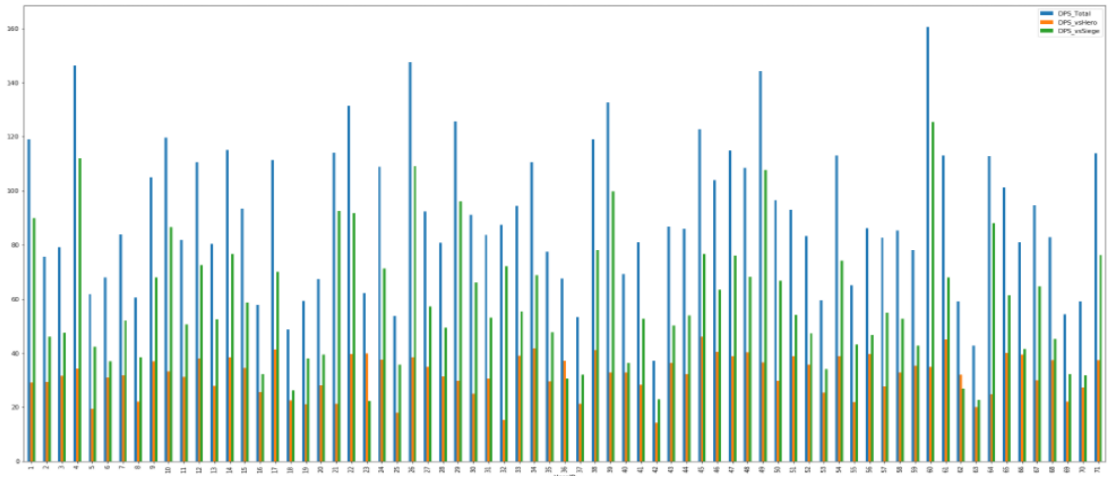
4) 맵별 효율이 높은 캐릭터

히오스에는 14개의 맵이 있다. 각 맵은 다른 구성을 가지고 있어 플레이어들에게 각기 다른 전략을 선택하게 만든다. 자연스럽게 각 맵별로 전략에 맞는 캐릭터가 있고, 전략에 맞지 않는 캐릭터가 있을 수 있다. 혹은, 맵에 상관 없이 좋은 캐릭터와 나쁜 캐릭터가 있을 수 있다. 또한 히오스는 각각의 캐릭터가 서로 다른 역할을 수행한다. 캐릭터의 역할로는 크게 전선에서 적에게 피해를 가하는 캐릭터가 있으며, 적이 가하는 피해를 대신 맞아줄 수 있는 캐릭터가 있으며, 아군이 입은 피해를 회복할 수 있는 캐릭터가 있다. 각각의 변량에 대해, 맵별로 최적의 효율을 내는 캐릭터가 있을 것이라 생각할 수 있고, 이를 선택하면 이길 수 있는 확률이 높아질 것이다. 우리는 이런 가설을 가지고, 실제로 맵별로 최적의 효율을 낼 수 있는 캐릭터들에 대해 분석해 보고자 했다.

4-1) 최적의 효율을 내는 캐릭터에 대한 분석

효율을 판단하기 위해 우리는 replays.csv 파일을 사용했다. 해당 파일을 불러온 뒤, 각 리플레이에 해당하는 게임이 얼마나 오래 진행되었는지 시간 정보를 사용한다. 이 시간 정보에서, 리플레이의 각캐릭터가 죽어있던 시간을 뺀다면 해당 캐릭터가 게임 안에서 '생존해 있던 시간' 을 구할 수 있을 것이다. 여기에서 각 캐릭터가 가진 값들, 예를 들자면 적에게 가한 피해량이나 받은 피해량, 등을 이 '생존해 있던 시간' 으로 나눠주게 된다면 시간당 그 캐릭터의 효율을 알게 될 것이다. 이를 분석해 본 결과는 다음과 같다.

```
In [67]: dps_by_hero.plot.bar(figsize=(30,15))
Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x2404dccdf28>
```



각 캐릭터별로 적 캐릭터에 가한 피해량, 적 구조물에 가한 피해량, 합한 피해량에 대한 시간 당 피해량을 구할 수 있었다. 추가적으로 가한 피해량 뿐만이 아니라 다른 역할군의 캐릭터들이 가지는 수치인 받은 피해량과 아군 치유량에 대해서도 같은 분석을 적용했다. 모든 분석 결과는 1003번 맵에서 행해진 게임에 대해서만 적용된 결과이다. 1003번 맵은 '저주받은 골짜기' 맵으로, 양 진영이 대칭적인 구조를 가지는 히오스에서 가장 표준적인 맵이다. 이 곳에서 행한 분석을 통해 가장 기본적인 캐릭터 효율을 알아낼 수 있을 것이라 생각했다. 분석은 다음과 같이 진행되었고,

```
mapID = 1001      # 1001 ~ 1020
want = 1          # 0: DPS_Total, 1: DPS_vsHero, 2: DPS_vsSiege, 3: DPS_DMG_Taken, 4: DPS_Healing
tmp = ""

a=Rep
a1 = a[a['MapID'] == mapID]
b = pd.merge(Rep_Char, a1, how='inner', left_on = 'ReplayID', right_on='ReplayID')
b['Alive Time']=0
for i in range(0, 10000):
    b['Alive Time'][i]=(datetime.datetime.strptime(b['Replay Length'][i], "%H:%M:%S") - datetime.datetime.strptime(b['Time Spent Dead'][i], '
b_sampled = b[0:10000]
b_sampled['Total_Damage'] = b_sampled['Hero Damage']+b_sampled['Siege Damage']

b_sampled['DPS_Total'] = b_sampled['Total_Damage']/b_sampled['Alive Time']
b_sampled['DPS_vsHero'] = b_sampled['Hero Damage']/b_sampled['Alive Time']
b_sampled['DPS_vsSiege'] = b_sampled['Siege Damage']/b_sampled['Alive Time']
b_sampled['DPS_DMG_Taken'] = b_sampled['Damage Taken']/b_sampled['Alive Time']
b_sampled['DPS_Healing'] = (b_sampled['Healing']+b_sampled['Self Healing'])/b_sampled['Alive Time']
dps_by_hero = b_sampled[['HeroID', 'DPS_Total', 'DPS_vsHero', 'DPS_vsSiege', 'DPS_DMG_Taken', 'DPS_Healing']].groupby('HeroID')[['DPS_Total',

if want ==0 :
    tmp = 'DPS_Total'
elif want == 1:
    tmp = 'DPS_vsHero'
elif want == 2:
    tmp = 'DPS_vsSiege'
elif want == 3:
    tmp = 'DPS_DMG_Taken'
elif want == 4:
    tmp = 'DPS_Healing'
NO1 = dps_by_hero[tmp].sort_values(ascending = False).index[0]
Hero_Map[Hero_Map['ID']==NO1]['Name']
```

그 결과는 다음과 같았다.

Total DPS 1, 2, 3위: 라그나로스, 해머, 아즈모단

vsHero DPS 1, 2, 3위: 갈, 줄진, 타이커스

vsSiege DPS 1, 2, 3위: 라그나로스, 아즈모단, 해머

탱 1, 2, 3위: 레오릭, 렉사르, 초

힐 1, 2, 3위: 모랄, 루시우, 빛나래

4-2) 효율이 높은 캐릭터에 관한 분석 결과

시간당 적에게 가한 피해량이 높은 캐릭터들이 실제로 승률이 높은지에 대한 검정이 추가적으로 진행되었다. 검정 결과 시간당 총 피해량 1,2,3위는 각각 승률이 51,51,55%로 일반적인 경우보다 높다는 것을 알 수 있었다. 시간당 적 캐릭터에 대한 피해량 1,2,3위 역시 승률이 50% 이상이었고, 시간당 받아낸 피해량 1,2,3위에 대해선 승률이 51,53,53%였다. 그렇지만 치유량 1,2,3위에 대해서는 1위만 50% 였을 뿐 2,3위는 승률이 47%였다.

이를 통해 알 수 있는 것은 실제로 시간당 '효율' 이 좋은 캐릭터들이 승률도 높다는 것이다. 이는 캐릭터에 대한 피해량, 구조물에 대한 피해량, 총 피해량, 그리고 받은 피해량에 대해 모두 적용되는 내용이다. 그렇지만 치유량에 대해서는 이러한 경향성이 나타나지 않는다. 이는 치유라는 것이 결국 아군이 받은 피해를 메꿔주는 역할일 뿐이지, 적극적으로 게임에 많이 기여하지는 않는 다는 생각을 하게 해준다. 결과적으로 우리가 위 분석에서 세운 뒤 검증된 가설인, 피해량이 높은 캐릭터가 더 많이 이길 것이라는 가설이 한번 더 검증된 것으로 볼 수 있다. 아울러 더 나아가, 통계적으로 모형을 만들어 분석할 때 적에게 가한 피해량을 중요하게 다루어야 한다는 것을 의미한다.

추가적으로 피해량 효율에 대해 검증이 실시되었다. 1003번 맵의 다른 데이터를 사용한 cross validation이 수행되었다. 다른 1만 개의 데이터 세트 총 5개에 대해서도 1, 2, 3위의 경향성에는 큰 변동이 없었다. 4번 샘플에서 3위가 '줄' 로 바뀌었을 분, 다른 데이터 세트에서 모두 1,2,3위는 동일했다.

3. 이기기 위한 가장 좋은 방법은?

위에서 분석된 결과를 보자면, 짧은 시간 안에 적에게 만은 피해량을 줄 수 있거나, 적에게 많은 피해를 입고 살아남을 수 있는 캐릭터를 고르는 것이 이기는 데 도움을 줄 수 있을 것이라는 것을 알 수 있다. 이를 통해 생각해 본, 이기기 위한 캐릭터 고르는 법은 다음과 같다. 먼저, 우리팀원들이 모두 캐릭터를 선택했고, 나만 선택하면 되는 경우라고 생각해보자. 그런 경우, 우선 아군에 부족한 역할군이 어떤 것인지 파악한 다음, 위 분석 결과처럼 맵별/역할군별 효율이 높은 캐릭터 중 하나를 선택해 플레이한다면 이길 확률이 높아질 것이다. 그렇지만 언제나 이 방법을 사용할 수는 없을 것이다. 또한, 히오스의 게임 모드 중에는 한번에 아군이 모두 캐릭터를 골라 플레이해야 하는 경우가 존재한다. 플레이어들은 사실 이미 많은 경험과 전적 분석 사이트들을 통해 이기는 조합이 무엇인지, 이 맵에선 어떤 캐릭터가 좋은지를 알 수 있을 것이다. 그렇지만 우리의 목적은 데이터 분석을 통해 이기는 솔루션을 제공하는 것이다. 따라서 우리는 우리가 위에서 분석한 바를 응용해, 최적의 캐릭터 조합을 고르게 하는 모델을 만들어 보고자 한다.

게임의 승패가 어떠한 방정식과 같은 모델링을 사용해 예측될 수 있다고 생각하면, 기존의 데이터를 바탕으로 회귀모형을 만들어볼 수 있을 것이다. 게임의 승패는 binary data이다. 따라서 우리는 게임의 승패를 로지스틱 회귀모형을 사용해 분석하기로 했다. 우리가 세운 방정식은 다음과 같다.

Y(승리의 여부) = a(계수)*(우리가 선택한 캐릭터들의 각 분야별 효율 순위의 합) +b(계수)*(카운터 픽의 수) +c(계수)*(조합 시너지 효과)

여기서 우리는 게임이 단순하게 많은 피해량으로 결정되는 것이 아니라는 가정을 하고, 회귀모형에 두 가지의 변수를 추가하기로 했다. 먼저, 카운터 픽 이라는 개념을 도입했다. 카운터 픽이라 함은, 특정한 캐릭터에 대해 그 캐릭터와 상성이 좋지 않아 적으로 나오는 경우 이기기 힘들어지는 캐릭터를 말한다. 우리는 카운터 픽을 특정한 캐릭터의 적으로 어떠한 캐릭터가 나온 경우의 승률이 유의미하게 낮은 경우로 생각하기로 했다. 만약 카운터 픽이 선택되었다면, 이 변수는 1이 된다. 카운터 픽이 선택되지 않았다면 이 변수는 0이 된다.

다음으로 생각된 변수는 시너지 효과이다. 시너지 효과는 아군에 있는 특정 두 캐릭터가 서로를 보완할 수 있게 되는 등 상성이 좋은 경우를 의미한다. 시너지 효과에 대해서 우리는 두 캐릭터가 같은 팀에 존재하는 경우의 승률이 유의미하게 높은 경우 시너지 효과가 있다고 생각했다. 시너지 효과가 있는 경우 이 변수는 1이 되고, 그렇지 않다면 0이 될 것이다.

다음과 같이 시너지 효과를 계산하기 위해 replay_characters.csv 파일의 데이터를 재배열 했다. 그 후, 특정한 두 캐릭터가 같은 팀에 있는 경우만을 찾았다.

```
In [199]: k4.iloc[0]['hero5']
Out[199]: 71

In [204]: tmp = [2, 16]
k6 = pd.DataFrame(columns = ['ReplayID', 'hero1','hero2','hero3','hero4','hero5','isWin'])
k7 = pd.DataFrame(columns = ['ReplayID', 'hero1','hero2','hero3','hero4','hero5','isWin'])
count=0
for i in range(0, 20000):
    if tmp[0]==k4.iloc[i]['hero1'] or tmp[0]==k4.iloc[i]['hero2'] or tmp[0]==k4.iloc[i]['hero3'] or tmp[0]==k4.iloc[i]['hero4'] or tmp[0]==k
        k6.loc[count]=k4.iloc[i]
        count+=1

count2=0
for i in range(0, count):
    if tmp[1]==k6.iloc[i]['hero1'] or tmp[1]==k6.iloc[i]['hero2'] or tmp[1]==k6.iloc[i]['hero3'] or tmp[1]==k6.iloc[i]['hero4'] or tmp[1]==k
        k7.loc[count2]=k6.iloc[i]
        count2+=1
k7

In [205]: z=k7['isWin'].sum()/k7['isWin'].count()
z
Out[205]: 0.5555555555555558

In [208]: a=k5[k5['HeroID']==tmp[0]]['Is Winner'].sum()/k5[k5['HeroID']==tmp[0]]['Is Winner'].count()
b=k5[k5['HeroID']==tmp[1]]['Is Winner'].sum()/k5[k5['HeroID']==tmp[1]]['Is Winner'].count()
(a+b)/2>z #true면 시너지 낮음, false면 높음
Out[208]: False
```

두 캐릭터가 같은 편에서 플레이한 경우의 승률을 계산해, 유의미한 경우를 시너지가 있다고 판단했다.

이제 로지스틱 회귀를 사용해 각 변수들의 계수를 구해 볼 것이다.

$$h_\theta(x) = g(\theta^T x)$$

이런 cost function을 만들고

```
In [5]: def sigmoid(z):
            #SIGMOID Compute sigmoid functoon
            #   J = SIGMOID(z) computes the sigmoid of z.

            # You need to return the following variables correctly
            g = np.zeros(z.shape)

            # ====================== YOUR CODE HERE ======================
            # Instructions: Compute the sigmoid of each value of z (z can be a matrix,
            #               vector or scalar).

            g = (1/(1+np.exp(-z)))

            # =============================================================

            return g
```

Now you will implement the cost function and gradient for logistic regression. Complete the code in costFunction.m to return the cost and gradient.

Recall that the cost function in logistic regression is

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(h_\theta(x^{(i)}) - (1-y^{(i)})\log(1-h_\theta(x^{(i)})]$$

이렇게 오차 함수를 만든 후,

```
    for i in range(m):

        J = J + (-y[i]*math.log(sigmoid(np.dot(th.T,X[i])))-(1-y[i])*math.log(1-sigmoid(np.dot(th.T,X[i]))))/m

    # =============================================================
```

```
In [7]: def gradient(X, y, th, _lambda=0):
            # Initialize some useful values
            m = len(y)

            # You need to return the following variables correctly
            grad = np.zeros(th.shape)

            # ====================== YOUR CODE HERE ======================


            for i in range(m):

                grad = grad + ((sigmoid(np.dot(th.T,X[i]))-y[i])*X[i])/m

            # =============================================================

            return grad
```

Gradient를 계산해서, cost function을 최소화하는 경우를 계산했다.

```
In [9]: cost(X, y, np.array(initial_theta))
Out[9]: 0.6931471805599406
```

오차의 합인 cost function은 위와 같은 경우일 때 최소화가 되고, 이때 우리 방정식의 계수를 구해보면, 앞에서부터

각 변수에 대해

```
In [12]: optimal_theta
Out[12]: array([-24.40141901,   0.2001471 ,   0.19532949])
```

와 같은 결과가 나오게 된다.

만들어진 식을, 식을 만드는데 사용된 데이터 셋에 대해 적용해 본 결과

```
In [14]: def predict(t, x):
    #PREDICT Predict whether the label is 0 or 1 using learned logistic
    #regression parameters theta
    #   p = PREDICT(theta, X) computes the predictions for X using a
    #   threshold at 0.5 (i.e., if sigmoid(theta'*x) >= 0.5, predict 1)

    m = X.shape[0] # Number of training examples

    # You need to return the following variables correctly
    p = np.zeros(m)

    # ==================== YOUR CODE HERE ====================
    # Instructions: Complete the following code to make predictions using
    #               your learned logistic regression parameters.
    #               You should set p to a vector of 0's and 1's
    #
    for i in range(m):
        a = (sigmoid(np.dot(t.T,x[i])))

        if a >= 0.5:
            p[i] = 1
        else:
            p[i] = 0

    # ============================================================

    return p
```

Let's predict the admission probably of a student with scores 45 and 85:

Training set accuracy:

```
In [15]: print 'logistic regression, training accuracy : ' +str(np.mean(predict(optimal_theta, X) == y))
logistic regression, training accuracy : 0.89
```

89% 정도의 정확도를 보여 주었다.

4. 결론 및 한계점, 추후 발전 방안

승률에 기여하는 요인들에 대한 가설을 세운 뒤 분석해 가설을 검증하고, 이를 통해 승률에 어떤 요인들이 영향을 끼치는지 알아볼 수 있었다. 더 나아가서, 우리의 원래 목적이던 '이기는 방법' 을 알기 위해 승패에 관여하는 변수들로 구성된 방정식을 제작해 보고, 이를 검증해 보았다.

이를 통해 우리는 개별 유저가 게임에서 이기도록 도울 수 있을 것이다. 또한, 게임 개발팀은 이미 하고 있는 일이겠지만 해당 데이터의 분석을 통해 게임의 밸런스를 파악하여 문제가 있다면 수정할 수도 있을 것이다. 이러한 시도를 통해 사람들은 이길 수 있다는 생각을 점점 하게 되고, 이를 바탕으로 게임에 대한 흥미 역시 점점 늘어날 것으로 생각된다.

한계점 역시 존재한다. 먼저 기술력 부족으로 승률 방정식이 너무 적은 수의 데이터만 가지고 만들어 졌다는 한계점이 존재한다. 그렇지만 시너지 효과 계산의 자동화와 비슷한 방식으로 카운터픽의 계산을 자동화할 수 있을 것이다. 이러한 방법을 통한다면 더 빠른 승률 방정식의 계산이 가능해 질 것이다. 데이터를 통한 검증이 아닌,

실증적인 검증이 시간적 부족 때문에 행해지지 못했다는 것 역시 우리의 분석이 가진 한계라고 볼 수 있을 것이다.

이렇게 세운 승률 방정식을 검증한 뒤, 기존 데이터와 완전히 무관한 12월 8일 이후에 얻어진 데이터에 이를 대입해 검증하는 것 역시 한 방법이 될 수 있다. 해당 기간 동안 대규모의 밸런스 관련된 패치가 없었다면, 추세는 일정하게 유지될 것이다. 새로 얻어진 데이터를 통해 승률 방정식을 계산해본 뒤 기존의 승률 방정식과 비교해 보는 것도 한가지 방법이 될 수 있을 것이다. 히어로즈 오브 더 스톰의 전적 분석 사이트는 이미, 특정한 캐릭터의 조합을 입력하면 그 조합의 분석된 승률을 나타내 주는 서비스를 진행하고 있다. 이 서비스보다 더 나은 결과를 보여주는 것이 우리의 최종적인 목표이다.