数学建模中期报告

——对本赛季 NBA 总冠军与 MVP 等荣誉归属的试探性预测

2017 年 5 月 7 日

陈旭鹏（2014012882）

王复英（2015011175）

马聿伯（2015012295）

摘要

我们根据开题报告所确立的目标建立了若干子模型。首先我们通过基本的静态数据，运用逻辑回归建立了一个比较简单的机器学习模）来预测 NBA 常规赛各支球队的战绩与排名。在该模型中我们对各支球队计算“等级分”，以此预测每场比赛两队的胜负概率，并最终给出我们的预测。该预测在一定程度上与本赛季的真实排名吻合，但也暴露出了一定的局限性。

为了克服这些局限性，我们在传统模型的基础上又按计划实现了以下几项内容：①利用爬虫进行大规模数据的爬取，为我们接下来使用机器学习，建立更可靠的模型提供可能。②寻找 SportVU 数据，实现真实比赛向动画模拟的转化。③了解如何使用深度学习模型分析动态图像，解决图像的大小，输入，样本划分，标注等问题。④获取各媒体的消息，为下一步补充分析积累资料。

接下来我们将基于位置的精准数据以及文本数据进行处理并撰写 proposal 以获得 stats.nba 提供的 SportVU 数据。此外我们还将进一步深入理解深度学习的相关模型，学习模型结构与参数调优，掌握数据挖掘相关知识，最终再将各部分模型综合起来进行分析，做出合理预测。

目录

一.利用传统模型进行预测与分析

首先我们希望建立一个传统模型对本赛季 NBA 常规赛的战绩与排名做出预测。尽管此时 NBA 常规赛已经落下帷幕，对其做出预测看似毫无意义，其实不然。由于我们使用的数据均来自上一个赛季的球队战绩，因此我们的模型没有受到本赛季球队表现的任何影响。更加重要的是，由于预测的事件已经发生，因此这个模型的精确度可以在客观世界中进行检验，我们也可以依据检验结果在接下来的时间里继续对其进行修正。

之所以称之为“传统模型”，主要有以下两点原因：一是该模型使用的数据类型比较基本，这些数据都是通过人工方式从各种数据分析网址上获取的；二是该模型的核心思想是对每个球队的“等级分”进行逻辑斯蒂回归，是一个比较基本的机器学习模型，已经有一套较为完善的算法体系和求解方法供我们参考使用。我们相信对它的熟悉与应用有助于接下来一系列更加复杂、更加精确的模型的建立与求解。

1.1 数据获取

我们采用了 NBA 数据统计专业网站 Basketball Reference.com 中的统计数据。该网站的中储存有不同球员、队伍、赛季的基本静态统计数据，如得分，篮板，助攻，抢断，失误犯规次数，胜负次数等。我们使用的主要则是上个赛季（2015——2016 赛季，含常规赛与季后赛）各个球队在每场比赛中的数据情况。

我们获得的数据有如下 5 部分：

(1) Team Per Game Stats:每支队伍平均每场比赛的表现统计

数据的具体含义如下表所示：

数据名	含义
Rk -- Rank	排名
G -- Games	参与的比赛场数（都为82 场）
MP -- Minutes Played	平均每场比赛进行的时间
FG--Field Goals	投球命中次数
FGA--Field Goal Attempts	投射次数
FG%--Field Goal Percentage	投球命中次数
3P--3-Point Field Goals	三分球命中次数
3PA--3-Point Field Goal Attempts	三分球投射次数
3P%--3-Point Field Goal Percentage	三分球命中率
2P--2-Point Field Goals	二分球命中次数
2PA--2-point Field Goal Attempts	二分球投射次数
2P%--2-Point Field Goal Percentage	二分球命中率
FT--Free Throws	罚球命中次数
FTA--Free Throw Attempts	罚球投射次数
FT%--Free Throw Percentage	罚球命中率
ORB--Offensive Rebounds	进攻篮板球

数据名	含义
DRB--Defensive Rebounds	防守篮板球
TRB--Total Rebounds	篮板球总数
AST--Assists	助攻
STL--Steals	抢断
BLK -- Blocks	封盖
TOV -- Turnovers	失误
PF -- Personal Fouls	个犯
PTS -- Points	得分

（2）**Opponent Per Game Stats**：所遇到的对手平均每场比赛的统计信息，所包含的统计数据与 **Team Per Game Stats** 中的一致，只是代表的该球队对应的对手的。

（3）Miscellaneous Stats：综合统计数据

数据项	数据含义
Rk (Rank)	排名
Age	队员的平均年龄
W (Wins)	胜利次数

数据项	数据含义
L (Losses)	失败次数
PW (Pythagorean wins)	基于毕达哥拉斯理论计算的赢的概率
PL (Pythagorean losses)	基于毕达哥拉斯理论计算的输的概率
MOV (Margin of Victory)	赢球次数的平均间隔
SOS (Strength of Schedule)	用以评判对手选择与其球队或是其他球队 的难易程度对比，0 为平均线，可以为正负数
SRS (Simple Rating System)	3
ORtg (Offensive Rating)	每 100 个比赛回合中的进攻比例
DRtg (Defensive Rating)	每 100 个比赛回合中的防守比例
Pace (Pace Factor)	每 48 分钟内大概会进行多少个回合
FTr (Free Throw Attempt Rate)	罚球次数所占投射次数的比例
3PAr (3-Point Attempt Rate)	三分球投射占投射次数的比例
TS% (True Shooting Percentage)	二分球、三分球和罚球的总共命中率
eFG% (Effective Field Goal Percentage)	有效的投射百分比（含二分球、三分球）
TOV% (Turnover Percentage)	每 100 场比赛中失误的比例

数据项	数据含义
ORB% (Offensive Rebound Percentage)	球队中平均每个人的进攻篮板的比例
FT/FGA	罚球所占投射的比例
eFG% (Opponent Effective Field Goal Percentage)	对手投射命中比例
TOV% (Opponent Turnover Percentage)	对手的失误比例
DRB% (Defensive Rebound Percentage)	球队平均每个球员的防守篮板比例
FT/FGA (Opponent Free Throws Per Field Goal Attempt)	对手的罚球次数占投射次数的比例

毕达哥拉斯定律：

$$\text{wins} = \frac{runs\ \ scored^2}{runs\ \ scored^2 + runs\ \ allowed^2}$$

我们将用这三个表格来评估球队过去的战斗力。

（4）15-16result：2015-2016 赛季每场比赛的胜负情况，2015~2016 年的 nba 常规赛及季后赛的每场比赛的比赛数据，用以评估 Elo score（在之后的实验小节中解释）

数据项	数据含义
WTeam	获胜方

数据项	数据含义
LTeam	失败方
WLoc	获胜球队是主场（H）还是客场(V)

(5) 16-17Schedule: 2016-2017 赛季的赛程安排

数据项	数据含义
HTeam	主场球队
VTeam	客场球队

1.2 数据分析

在获取到数据之后，我们将利用每支队伍过去的比赛情况和 Elo 等级分来判断每支比赛队伍的可胜概率。在评价到每支队伍过去的比赛情况时，我们将使用到 Team Per Game Stats，Opponent Per Game Stats 和 Miscellaneous Stats（之后简称为 T、O 和 M 表）这三个表格的数据，作为代表比赛中某支队伍的比赛特征。我们最终将实现针对每场比赛，预测比赛中哪支队伍最终将会获胜，但并不是给出绝对的胜败情况，而是预判胜利的队伍有多大的获胜概率。之后我们采用随机数模拟的方法获得每个球队的胜场数。

为了做出对比赛结果的较为合理的预测，我们需要建立一个代表比赛的特征向量。由两支队伍的以往比赛情况统计情况（T、O 和M表），和两个队伍各自的 Elo 等级分构成。

Elo 的最初为了提供国际象棋中，更好地对不同的选手进行等级划分。在现在很多的竞技运动或者游戏中都会采取 Elo 等级分制度对选手或玩家进行等级划分，如足球、篮球、棒球比赛或 LOL，DOTA 等游戏。

在这里我们将基于国际象棋比赛，大致地介绍下 Elo 等级划分制度。假设 A 和 B 的当前等级分为R_A和R_B，则 A 对 B 的胜率期望值为：

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

B 对 A 的胜率期望值为：

$$E_B = \frac{1}{1 + 10^{(R_A - R_B)/400}}$$

如果棋手 A 在比赛中的真实得分S_A（胜 1 分，和 0.5 分，负 0 分）和他的胜率期望值E_A不同，则他的等级分要根据以下公式进行调整：

$$R_A^{new} = R_A^{old} + K(S_A - R_A^{old})$$

在国际象棋中，根据等级分的不同 K 值也会做相应的调整：

- ⩾2400，K=16
- 2100~2400 分，K=24
- ⩽2100，K=32

因此我们将会用以表示某场比赛数据的特征向量为（对阵双方为 A 队和 B 队）：[A 队 Elo score, A 队的 T,O 和 M 表统计数据，B 队 Elo score, B 队的 T,O 和 M 表统计数据]

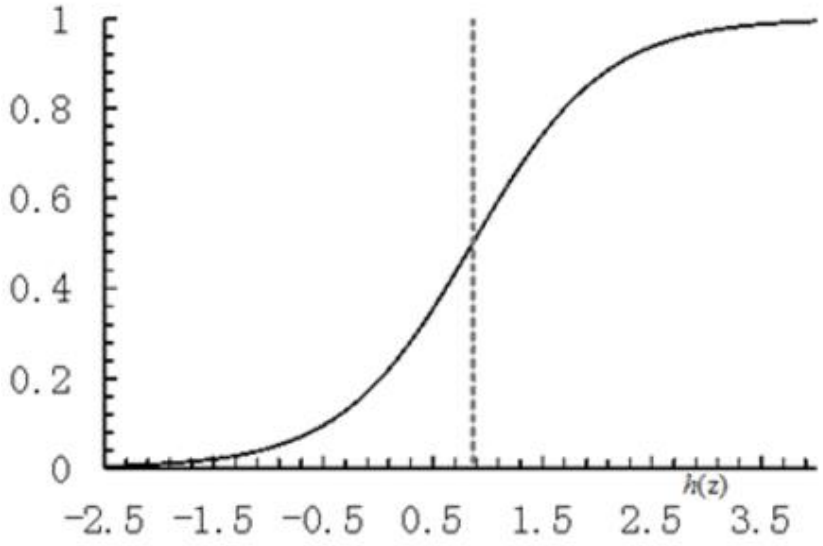
1.3 模型建立

综合考虑后我们采用逻辑回归模型进行预测，逻辑回归就是一种减小预测范围，将预测值限定为[0,1]间的一种回归模型.

(1)构造预测函数 h

逻辑回归实际上是一种分类方法，主要用于二分类问题（即输出只有两种，分别代表两个类别），所以利用了 Logistic 函数（或称为 Sigmoid 函数），函数表达式和曲线如下：

$$g(z) = \frac{1}{1+e^{-z}}$$



逻辑曲线在 z=0 时，十分敏感，在 z>>0 或 z<<0 处，都不敏感，将预测值限定为(0,1)。

构造预测函数为：

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

函数$h_\theta(x)$的值表示结果取 1 的概率，因此对于输入 x 分类结果为类别 1 和类别 0 的概率分别为：

$$P(y=1|\theta) = \frac{1}{1+e^{-\theta^T x}} \qquad P(y=0|\theta) = \frac{e^{-\theta^T x}}{1+e^{-\theta^T x}} = 1 -$$

$$P(y=1|\theta) = P(y=1|x,-\theta)$$

（2）构造损失函数 J

由最大似然估计得到如下 Cost 函数和 J 函数：

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) \ if \ y = 1 \\ -\log(1 - h_\theta(x)) \ if \ y = 0 \end{cases}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{n} Cost(h_\theta(x_i), y_i) = -\frac{1}{m}\left[\sum_{i=1}^{n} y_i \log h_\theta(x_i) + (1 - y_i)\log(1 - h_\theta(x_i))\right]$$

（3）梯度下降法求的最小值

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i^j$$

（4）向量化

约定训练数据的矩阵形式如下，x 的每一行为一条训练样本，而每一列为不同的特称值。y 向量为每个训练样本的标签。

$$x = \begin{bmatrix} x_1 \\ \dots \\ x_m \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \dots \\ y_m \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \dots \\ \theta_n \end{bmatrix}$$

$$A = x \bullet \theta = \begin{bmatrix} x_{10} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} \bullet \begin{bmatrix} \theta_0 \\ \dots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 x_{10} + \theta_1 x_{11} + \dots + \theta_0 x_{1n} \\ \dots \\ \theta_0 x_{m0} + \theta_1 x_{m1} + \dots + \theta_0 x_{mn} \end{bmatrix}$$

$$E = h_\theta(x) - \mathrm{y} = \begin{bmatrix} g(A_1) - y_1 \\ \dots \\ g(A_m) - y_m \end{bmatrix} = \begin{bmatrix} e_1 \\ \dots \\ e_m \end{bmatrix} = g(A) - y$$

因此θ的更新过程可表示为：

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_i^j = \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m} e_i x_i^j = \theta_j - \alpha \frac{1}{m} x^T E$$

1.4 算法解释

获得了具体的数据之后，我们首先利用 python 中的 pandas 库对数据进行整合，对每支队伍我们都能获得其在 15-16 赛季的一系列数据（如场均得分、场均篮板等），作为其特征向量。另外我们还加入了一个重要元素作为特征：elo。我们借鉴国际象棋中的预测方法，设定初始状态下两队的 elo 均为 1600，对主队 elo 增加 100，我们将 elo 作为特征向量的第一行，对每支球队，我们都能给出他的一个特征向量，整合之后共有 66 个特征。

然后我们根据 2015-2016 赛季的比赛结果调整两队的 elo 值，具体调整方法与国际象棋预测的方法相同。为了训练一个模型，我们建立样本集 X，X 共有 1316 行，其每一行代表一个样本（2015-2016 赛季的一场比赛）两个队伍的所有特征。y 为分类问题的类别。在这里我们采用随机生成 0 或 1 的方法：如果生成的 y 为 0，X 的该行就为胜者特征在前，负者特征在后；如果 y=1，X 的该行就为负者特征在前，胜者特征在后。

然后我们对 X 和 y 做逻辑回归，得到一个模型。对于 2016-2017 赛季的每一场比赛，我们只需给出对阵双方及其主客场，模型预测的 y 为（0,1）的数，如果 y<0.5，我们将其归到 y=0 的类，即后面的队伍获胜，前面的队伍失败；如果 y>0.5，我们将其归到 y=1 的类，即前面的队伍胜利，后面的队伍失败。这样我们就预测得到了 2016-2017 赛季每场比赛的获胜概率。

为了估计每支球队的获胜场次，在得到每场比赛球队获胜的概率后，我们通过产生（0,1）随机数的方法模拟该比赛的结果。如果随机数小于概率，我们认为获胜；如果随机数小于概率，我们认为存在其他因素导致出现爆冷，获胜概率大的球队反而失败。

1.5 代码实现

我们主要用 python 中的机器学习库 sklearn 来实现逻辑回归模型。

（1）插入要使用的相关模块

```
import pandas as pd

import math

import csv

import random

import numpy as np

from sklearn import linear_model

from sklearn.model_selection import cross_val_score
```

（2）设置模型回归训练时需要用到的参数变量

```
#当每支队伍没有 elo 等级分时，赋予其基础 elo 等级分

base_elo = 1600 #设定基础等级分为 1600

team_elos = {}

team_stats = {}

X = []

y = []

folder = 'data'     #表示存放数据（即我们获得的 5 个表格）的目录
```

（3）初始化数据，

从 **T、O 和 M** 表格中读入数据，去除一些无关数据并将这三个表格通过 Team 属性列进行连接。（注：下面的参数 Mstat, Ostat, Tstat 分别表示从 M、O、T 表中读取的数据，其格式为 pandas 的 dataframe 结构。）

```
def initialize_data(Mstat, Ostat, Tstat):
    new_Mstat = Mstat.drop(['Rk', 'Arena'], axis=1)  #删除 RK 列（排名）、Arena 列（球馆名称）
    new_Ostat = Ostat.drop(['Rk', 'G', 'MP'], axis=1)  #删除 RK 列、G 列（参加的比赛数）、MP 列（平均每场比赛的进行时间）
    new_Tstat = Tstat.drop(['Rk', 'G', 'MP'], axis=1)  #删除 RK、G。MP 列
    team_stats1 = pd.merge(new_Mstat, new_Ostat, how='left', on='Team')
    team_stats1 = pd.merge(team_stats1, new_Tstat, how='left', on='Team') #合并数据集
    return team_stats1.set_index('Team', inplace=False, drop=True) #建立索引
```

（4）获取每支队伍的 Elo Score 等级分函数，当在开始没有等级分时，将其赋予初始 base_elo 值：

```
def get_elo(team):
    try:
        return team_elos[team]
    except:
        #  当最初没有 elo 时，给每个队伍最初赋 base_elo（1600）
        team_elos[team] = base_elo
        return team_elos[team]
```

（5）定义计算每支球队的 Elo 等级分函数：

```
# 计算每个球队的 elo 值
def calc_elo(win_team, lose_team):
    winner_rank = get_elo(win_team)     #获胜方的等级
    loser_rank = get_elo(lose_team)      #失败方的等级
    rank_diff = winner_rank - loser_rank
    exp = (rank_diff  * -1) / 400
    odds = 1 / (1 + math.pow(10, exp))   #计算出胜率的期望
    #  根据 rank 级别修改 K 值
    if winner_rank < 2100:
        k = 32
    elif winner_rank >= 2100 and winner_rank < 2400:
        k = 24
    else:
        k = 16
    new_winner_rank = round(winner_rank + (k * (1 - odds))) #对获胜方的等
级根据比赛的真实情况做修正
    new_rank_diff = new_winner_rank - winner_rank
    new_loser_rank = loser_rank - new_rank_diff   #失败方的等级做相应调整
    return new_winner_rank, new_loser_rank
```

（6）建立比赛数据集

基于我们初始好的统计数据，及每支队伍的 **Elo score** 计算结果，建立对应 2015~2016 年常规赛和季后赛中每场比赛的数据集（在主客场比赛时，我们认为主场作战的队伍更加有优势一点，因此会给主场作战队伍相应加上 100 等级分）：

```
def   build_dataSet(all_data):      #主函数中参数为 result_data
    X = []
    skip = 0
    for index, row in all_data.iterrows():      #获取每行的 index、row
        Wteam = row['WTeam']
        Lteam = row['LTeam']
        #获取每个队伍最初的 elo 值
        team1_elo = get_elo(Wteam)
        team2_elo = get_elo(Lteam)
        #  给主场比赛的队伍加上 100 的 elo 值
        if row['WLoc'] == 'H':
            team1_elo += 100
        else:
            team2_elo += 100
        #  把 elo 当为评价每个队伍的第一个特征值
        team1_features = [team1_elo]
        team2_features = [team2_elo]
        #  从 team_stats 中添加双方队伍的统计信息
```

```
            for key, value in team_stats.loc[Wteam].iteritems():
                team1_features.append(value)
            for key, value in team_stats.loc[Lteam].iteritems():
                team2_features.append(value)
            # X 的每一行为一个样本的特征值，其内容是两个队伍的特征。两队伍的特征值的顺序为随机生成的
            # y 值为随机生成的 0 或者 1
            if random.random() > 0.5:
                X.append(team1_features + team2_features)
                y.append(0)
            else:
                X.append(team2_features + team1_features)
                y.append(1)
            if skip == 0:
                print (X)
                skip = 1
            #  根据这场比赛的结果更新队伍的 elo 值
           new_winner_rank, new_loser_rank = calc_elo(Wteam, Lteam)
             team_elos[Wteam] = new_winner_rank
            team_elos[Lteam] = new_loser_rank
      return np.nan_to_num(X), y
```

（7）逻辑回归

在 main 函数中调用这些数据处理函数，使用 sklearn 的 Logistic Regression 方法建立回归模型：

```
if __name__ == '__main__':
    #读取 M、O、T 表中的数据
     Mstat = pd.read_csv(folder + '/15-16Miscellaneous_Stat.csv')
     Ostat = pd.read_csv(folder + '/15-16Opponent_Per_Game_Stat.csv')
     Tstat = pd.read_csv(folder + '/15-16Team_Per_Game_Stat.csv')
     team_stats = initialize_data(Mstat, Ostat, Tstat)
     result_data = pd.read_csv(folder + '/2015-2016_result.csv')
     X, y = build_dataSet(result_data)
     # 训练网络模型
     model = linear_model.LogisticRegression()
     model.fit(X, y)
     #利用 10 折交叉验证计算训练正确率
     print("Doing cross-validation..")
     print(cross_val_score(model, X, y, cv = 10, scoring='accuracy', n_jobs=-
1).mean())    #输出交叉验证的正确率
```

（8）预测概率

最终利用训练好的模型在 16~17 年的常规赛数据中进行预测。利用模型对一场新的比赛进行胜负判断，并返回其胜利的概率：

```
def predict_winner(team_1, team_2, model):
    features = []
    # team 1，客场队伍，添加特征值
    features.append(get_elo(team_1))      #赋予初始 elo 值
    for key, value in team_stats.loc[team_1].iteritems():
        features.append(value)            #添加特征
    # team 2，主场队伍，添加特征值
    features.append(get_elo(team_2) + 100)
    for key, value in team_stats.loc[team_2].iteritems():
        features.append(value)
    features = np.nan_to_num(features)
    return model.predict_proba([features])
```

（9）预测 2016-2017 赛季比赛结果

```
print('Predicting on new schedule..')
schedule1617 = pd.read_csv(folder + '/16-17Schedule.csv')
result = []
for index, row in schedule1617.iterrows():
    team1 = row['Vteam']
    team2 = row['Hteam']
    pred = predict_winner(team1, team2, model)   #根据模型预测获胜的概率
    prob = pred[0][0]
```

```
        if prob > 0.5:
                winner = team1
                loser = team2
                result.append([winner, loser, prob])
        else:
                winner = team2
                loser = team1
                result.append([winner, loser, 1 - prob])
with open('16-17Result.csv', 'w') as f:
        writer = csv.writer(f)
        writer.writerow(['win', 'lose', 'probability'])
        writer.writerows(result)
```

（10）随机数模拟

注：我们将刚才得到的表格存为 excel 格式，并在其后加了两列，第一列为随机产生的（0,1）的实数，第二列为概率减去随机数的正负。name.xlsl 第一列为 30 支球队的队名。

```
import xlrd
data = xlrd.open_workbook('16-17result.xlsx')
table = data.sheets()[0]
nrows = table.nrows
name = xlrd.open_workbook('name.xlsx')
```

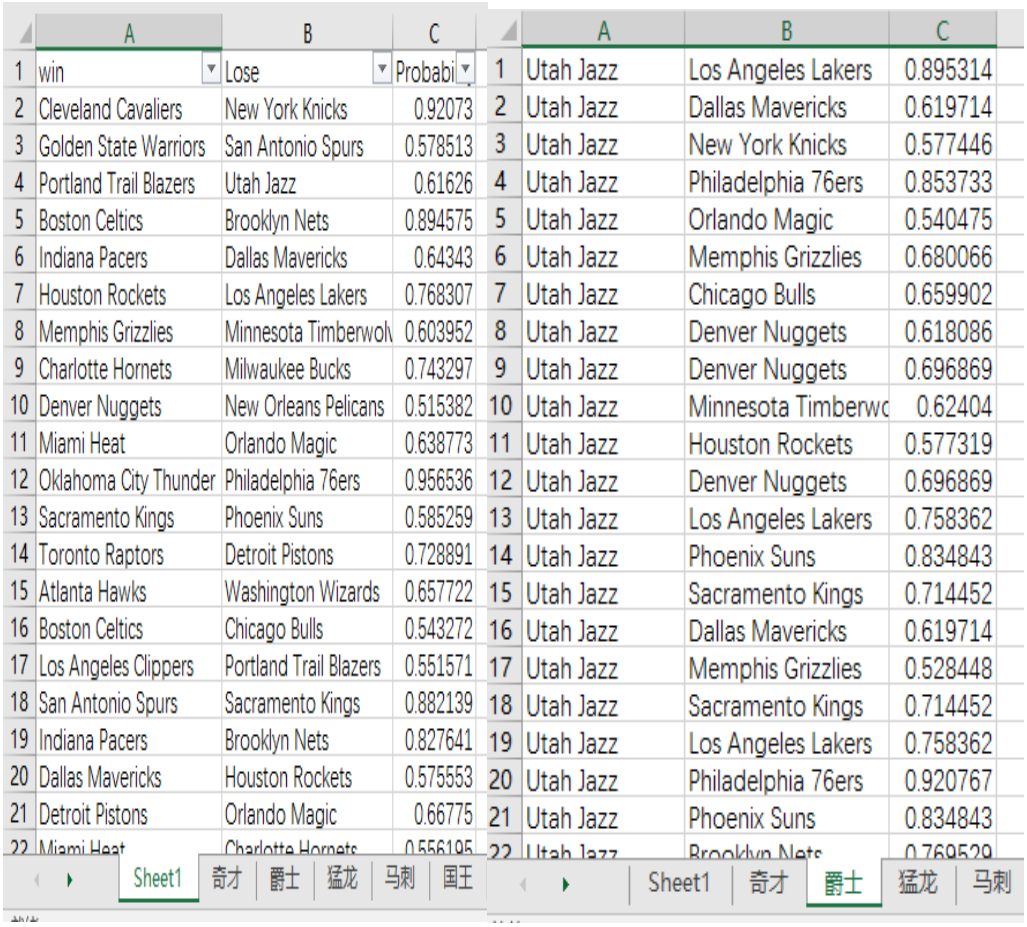
```
nametable =name.sheets()[0]
namerows = nametable.nrows
a=[0]*30
dic={}
for name in range(namerows):
for i in range(nrows):

if((table.row(i)[4].value==1)and(table.row(i)[1].value==nametable.row(name)[0
].value)):
                a[name]=a[name]+1

if((table.row(i)[4].value==1)and(table.row(i)[0].value==nametable.row(name)[0
].value)):
                a[name]=a[name]+1
    dic[nametable.row(name)[0].value]=a[name]
import csv
with open("16-17.csv", "w") as csvFile:
    csvWriter = csv.writer(csvFile)
    for k,v in dic.items():
        csvWriter.writerow([k,v])
```

1.6 模型求解

通过以上我们获得了 2016-2017 赛季常规赛的预测结果，我们对本赛季的任何一场比赛都进行了胜负判断，并得到了预计获胜球队的胜利概率。（为了便于分析，我们整理得到了每支球队全赛季 82 场比赛胜利或失败的概率。）



	A	B	C
1	win	Lose	Probabi
2	Cleveland Cavaliers	New York Knicks	0.92073
3	Golden State Warriors	San Antonio Spurs	0.578513
4	Portland Trail Blazers	Utah Jazz	0.61626
5	Boston Celtics	Brooklyn Nets	0.894575
6	Indiana Pacers	Dallas Mavericks	0.64343
7	Houston Rockets	Los Angeles Lakers	0.768307
8	Memphis Grizzlies	Minnesota Timberwol	0.603952
9	Charlotte Hornets	Milwaukee Bucks	0.743297
10	Denver Nuggets	New Orleans Pelicans	0.515382
11	Miami Heat	Orlando Magic	0.638773
12	Oklahoma City Thunder	Philadelphia 76ers	0.956536
13	Sacramento Kings	Phoenix Suns	0.585259
14	Toronto Raptors	Detroit Pistons	0.728891
15	Atlanta Hawks	Washington Wizards	0.657722
16	Boston Celtics	Chicago Bulls	0.543272
17	Los Angeles Clippers	Portland Trail Blazers	0.551571
18	San Antonio Spurs	Sacramento Kings	0.882139
19	Indiana Pacers	Brooklyn Nets	0.827641
20	Dallas Mavericks	Houston Rockets	0.575553
21	Detroit Pistons	Orlando Magic	0.66775
22	Miami Heat	Charlotte Hornets	0.556195

Sheet1 | 奇才 | 爵士 | 猛龙 | 马刺 | 国王

	A	B	C
1	Utah Jazz	Los Angeles Lakers	0.895314
2	Utah Jazz	Dallas Mavericks	0.619714
3	Utah Jazz	New York Knicks	0.577446
4	Utah Jazz	Philadelphia 76ers	0.853733
5	Utah Jazz	Orlando Magic	0.540475
6	Utah Jazz	Memphis Grizzlies	0.680066
7	Utah Jazz	Chicago Bulls	0.659902
8	Utah Jazz	Denver Nuggets	0.618086
9	Utah Jazz	Denver Nuggets	0.696869
10	Utah Jazz	Minnesota Timberwo	0.62404
11	Utah Jazz	Houston Rockets	0.577319
12	Utah Jazz	Denver Nuggets	0.696869
13	Utah Jazz	Los Angeles Lakers	0.758362
14	Utah Jazz	Phoenix Suns	0.834843
15	Utah Jazz	Sacramento Kings	0.714452
16	Utah Jazz	Dallas Mavericks	0.619714
17	Utah Jazz	Memphis Grizzlies	0.528448
18	Utah Jazz	Sacramento Kings	0.714452
19	Utah Jazz	Los Angeles Lakers	0.758362
20	Utah Jazz	Philadelphia 76ers	0.920767
21	Utah Jazz	Phoenix Suns	0.834843
22	Utah Jazz	Brooklyn Nets	0.769529

Sheet1 | 奇才 | 爵士 | 猛龙 | 马刺

仅仅得到预期胜负的概率值并不能断定一场比赛的结果，原因很简单：即使一只球队取胜的概率为 99%，我们也无法保证在真实情况下实力偏弱的球队爆冷取胜，实现那 1%的奇迹。

为了模拟这种概率上的随机性，我们利用 matlab 生成从 0 至 1 的随机数。我们称某场比赛(A 队 vs B 队)胜率大于 50%的球队为预计获胜方（A 队），预计获胜方取胜的概率计为 p。当所取随机数小于 p 时，我们认为 A 队在意料之中的击败了 B 队；反之则认为 B 队爆冷战胜了 A 队。

就这样，我们利用对每一场比赛胜负概率的判断做出了对每一场比赛胜负情况的估计，略加整理后即可得到该模型对本赛季球队战绩与排名的预估：

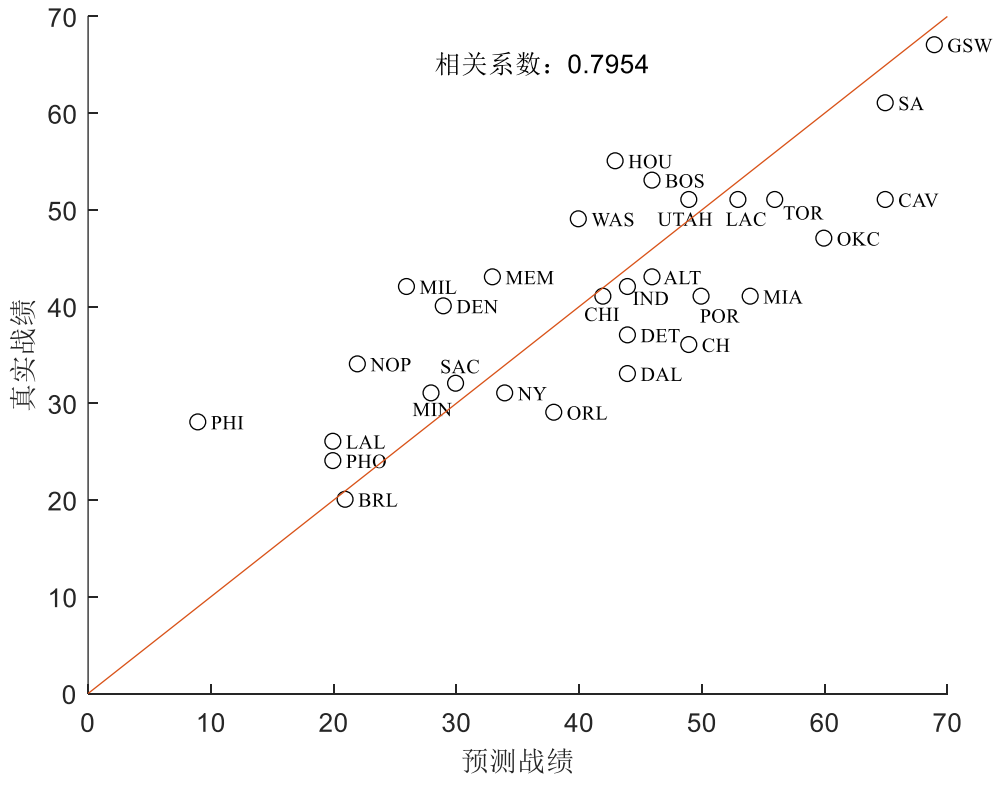
1	骑士	65	17
2	猛龙	56	26
3	热火	54	28
4	黄蜂	49	33
5	凯尔特人	46	36
6	老鹰	46	36
7	步行者	44	38
8	活塞	44	38
9	公牛	42	40
10	奇才	40	42
11	魔术	38	44
12	尼克斯	34	48
13	雄鹿	26	56
14	篮网	21	61
15	76人	9	73

东部球队战绩与排名

1	勇士	69	13
2	马刺	65	17
3	雷霆	60	22
4	快船	53	29
5	开拓者	50	32
6	爵士	49	33
7	小牛	44	38
8	火箭	43	39
9	灰熊	33	49
10	国王	30	52
11	掘金	29	53
12	森林狼	28	54
13	鹈鹕	22	60
14	太阳	20	62
15	湖人	20	62

西部球队战绩与排名

预测战绩与真实战绩的散点图



如果预测结果与真实情况完全符合，则点都落在直线 y=x 上，但是实际情况并非如此。我们用这两组数据的相关系数来评价预测的精确度。

我们把模型生成的结果与本赛季常规赛的真实排名对比后发现，该模型可以在一定程度上反应本赛季的真实情况，但不可否认其结果存在较大误差，这也暴露出了我们所用传统模型的局限性。

排名	球队	胜	负	胜率
1	凯尔特人[11]	53	29	64.6%
2	骑士[1]	51	31	62.2%
3	猛龙	51	31	62.2%
4	奇才[1]	49	33	59.8%
5	老鹰	43	39	52.4%
6	雄鹿	42	40	51.2%
7	步行者	42	40	51.2%
8	公牛	41	41	50%
9	热火	41	41	50%
10	活塞	37	45	45.1%
11	黄蜂	36	46	43.9%
12	尼克斯	31	51	37.8%
13	魔术	29	53	35.4%
14	76人	28	54	34.1%
15	篮网	20	62	24.4%

东部球队战绩与排名

排名	球队	胜	负	胜率
1	勇士[11]	67	15	81.7%
2	马刺[1]	61	21	74.4%
3	火箭	55	27	67.1%
4	快船	51	31	62.2%
5	爵士[1]	51	31	62.2%
6	雷霆	47	35	57.3%
7	灰熊	43	39	52.4%
8	开拓者	41	41	50%
9	掘金	40	42	48.8%
10	鹈鹕	34	48	41.5%
11	小牛	33	49	40.2%
12	国王	32	50	39%
13	森林狼	31	51	37.8%
14	湖人	26	56	31.7%
15	太阳	24	58	29.3%

西部球队战绩与排名

1.7 局限性分析

经过初步分析，我们认为该模型的局限性主要来自以下几个方面：

l 数据类型较为单一

正如之前提到过的，该模型虽然属于机器学习模型，但其所使用的数据主要以有限的静态数据为主。从 Basketball Reference.com 人工获得的数据主要存在两个问题，一是数据量不够丰富。该网站的数据信息对于一个业余 NBA 爱好者已经足够眼花缭乱，但对于需要海量数据的机器学习模型的建立而言显然仍略显单薄。二是数据同质化问题突出。尽管这些数据可以在很大程度上反映这场比赛

的走势，但其同样有很多力不能逮的地方：比如“垃圾时间”的得分与“关键时刻”的得分价值远不能同日而语，但体现在这些数据上却是完全一样的。

II 考虑因素不够细致

从代码实现可以看出，该模型通过对不同球队“等级分”之间的比较来判断双方对阵的胜负概率，却忽略了一些在竞技体育中非常重要的影响因素。例如：①模型中只是简单的为主队加 100 分，但却忽视了赛程对球队状态的影响——在 NBA 中，如果一支球队连续征战数个客场，那么它的战斗力将会显著下降；②模型中假设球队阵容完整且健康，但在真实情况下随着赛季的不断深入，伤病总会多多少少影响球队的状态，在关键时刻甚至会对球队造成毁灭性的打击（如 2014——2015 赛季总决赛其间勒夫和欧文的相继受伤使得胜利的天平迅速向勇士倾斜），因此偶发伤病的影响也应该纳入对胜率期望的计算之中。

③对球队等级分的“评定”截止于上赛季结束后，未考虑休赛期的球员流动与选秀情况，遑论本赛季开始后的球队交易（例如杜兰特的出走使得对雷霆的战绩预测出现极大误差）。而这些都是我们在下一步分析总决赛冠军及 MVP 时所必须要考虑到的。

III 应用范围有限

该模型只能应用于常规赛战绩和排名的预测，对 NBA 总冠军以及 MVP 归属的预测存在一定难度。因为前者的七局四胜制会导致极大的偶然性，且有时会出现两队“球风相克”、“球员相克”的特殊情况，这是回归类型方法较难解决的；而后者则需要考虑球员个人对球队体系的帮助与影响，这同样是该传统模型难以进行量化的。

综上所述，我们在接下来的一段时间内将有针对性地对上文所提到的这些局

限性进行改进与修正，并尝试建立更加精确的非传统模型，对当季 NBA 的一系列排名及荣誉归属进行试探性的预测。

二 利用爬虫进行更大规模数据的收集

为了能够超越传统预测模型的局限，我们需要更多更具体的数据来体现球队水平、球员竞技状态等情况。除了在传统模型中收集的 BBR 网站上的基础数据，我们需要更多的数据支持我们分析。

我们锁定了 NBA 数据统计专业网站 stats.nba.com。该网站数据更为齐全，适合我们进行进一步的数据挖掘。

基于 python 的爬虫是挖掘数据的强大的工具，可是由于 NBA 的数据纷繁复杂，种类极其多，为每个网页编写程序进行爬取较为费时费力，因此我们需要寻找更为合适的方法。

首先我们寻找到了一个提供 NBA 数据分析 API 的网站 probasketballapi.com。该网站提供七天免费试用的 API，可以爬取多种数据。包括: Basic Resources: Teams, Players, Games, Shot Charts; Advanced Stats: Advanced Team Stats, Advanced Player Stats; Box Score: Team Boxscores, Player Boxscore; Four Factors: Team Four Factors, Player Four Factors; Misc Stats: Team Misc Stats, Player Misc; Stats; SportsVU: Team SportsVU Data, Player SportsVU Data; Player Usage: Player Usage Data. 注意此 API 提供的 SportVU 数据也和其他数据一样，属于纯数据类型。

该网站提供了基于 PHP 的接口，为了统一处理，我们将代码改写为 python。

```
import pycurl
import json
from io import BytesIO


buffer = BytesIO()
url='http://api.probasketballapi.com/sportsvu/team'
api_key = 'DcMv4T2f0h5kXY6BZOdzFLPq1GWiUH8E'
team_id=1610612753
season=2014
query_string='api_key='+api_key+'&team_id='+str(team_id)+'&season='
+str(season)
mycurl=pycurl.Curl()
mycurl.setopt(pycurl.URL,url)
mycurl.setopt(pycurl.POSTFIELDS,query_string)
mycurl.setopt(pycurl.WRITEDATA,buffer)
mycurl.perform()
mycurl.close()


contents=buffer.getvalue()
results=json.loads(contents.decode('iso-8859-1'))
results
```

这样，只用改写 url 以及不同的 query_string 即可调取不同的所需要的数据，并将其写入 csv 中方便之后的调用。

```
#写入 csv 中
with open("nba2.csv","w",newline="") as datacsv:
    headers = [k for k in results[0]]
    #headers =
['game_id','team_id','opponent_id','period','season','spd','dist','orbc','drbc','tchs','sast','ftast','pass','cfga','ufgm','ufga','dfgm','dfga']
    writer = csv.DictWriter(datacsv,fieldnames=headers)
    #csv 文件插入一行数据，把下面列表中的每一项放入一个单元格（可以用循环插入多行）
    writer.writeheader() # CSV 第一行需要自己加入
    for result in results:
        writer.writerow(result)# rows 就是表单提交的数据
```

接下来我们又找到了 github 中提供的 stats.nba.com 的‘官方’接口 nba_py，发现该接口提供的数据类型更为丰富具体，且调取更加方便。该接口共提供了数十种不同的数据，不仅包括赛季的平均数据，还包括每场比赛的具体数据，这也为我们接下来的数据挖掘与建模提出了巨大的挑战。

下面举一个简单的例子展示如何使用 nba_py 进行数据调取

```
from nba_py import game
r=game.Boxscore('0041400122').player_stats()
r.to_csv('/Users/james/Desktop/2.csv')
```

首先我们载入 game 模块，然后我们选择调取某场比赛的 Boxscore 数据中关于运动员的数据。0041400122 为该场比赛的 GameID。如果我们查看返回的数据类型会发现是 list，下面只需要简单的命令即可写入 csv 中，我们打开 csv 文件发现生成了这样一个表格：

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
GAME_ID	TEAM_ID	TEAM_ABBREV	TEAM_CITY	PLAYER_ID	PLAYER_NAME	START_POSI	COMMENT	MIN	FGM	FGA	FG_PCT	FG3M	FG3A	FG3_PCT	FTM
41400122	1610612749	MIL	Milwaukee	203507	Giannis Antetoki	F		37:52:00	2	11	0.182	0	0	0	2
41400122	1610612749	MIL	Milwaukee	101141	Ersan Ilyasova	F		23:44	3	10	0.3	0	3	0	2
41400122	1610612749	MIL	Milwaukee	2585	Zaza Pachulia	C		22:32	3	8	0.375	0	0	0	1
41400122	1610612749	MIL	Milwaukee	203114	Khris Middleton	G		38:19:00	8	20	0.4	3	7	0.429	3
41400122	1610612749	MIL	Milwaukee	203487	Michael Carter-\	G		32:51:00	5	12	0.417	0	1	0	2
41400122	1610612749	MIL	Milwaukee	203089	John Henson			25:28:00	4	9	0.444	0	1	0	0
41400122	1610612749	MIL	Milwaukee	201162	Jared Dudley			17:02	1	4	0.25	1	2	0.5	0
41400122	1610612749	MIL	Milwaukee	201573	Jerryd Bayless			15:09	3	6	0.5	0	0	0	2
41400122	1610612749	MIL	Milwaukee	201564	O.J. Mayo			27:03:00	3	10	0.3	0	3	0	2
41400122	1610612749	MIL	Milwaukee	203898	Tyler Ennis		DNP - Coach's Decision								
41400122	1610612749	MIL	Milwaukee	203268	Jorge Gutierrez		DNP - Coach's Decision								
41400122	1610612749	MIL	Milwaukee	203948	Johnny O'Bryant III		DNP - Coach's Decision								
41400122	1610612749	MIL	Milwaukee	203101	Miles Plumlee		DNP - Coach's Decision								
41400122	1610612741	CHI	Chicago	2399	Mike Dunleavy	F		33:24:00	4	12	0.333	4	9	0.444	0
41400122	1610612741	CHI	Chicago	2200	Pau Gasol	F		33:35:00	4	12	0.333	0	1	0	3
41400122	1610612741	CHI	Chicago	201149	Joakim Noah	C		32:19:00	3	9	0.333	0	0	0	0
41400122	1610612741	CHI	Chicago	202710	Jimmy Butler	G		46:00:00	10	19	0.526	3	9	0.333	8
41400122	1610612741	CHI	Chicago	201565	Derrick Rose	G		38:05:00	4	14	0.286	2	6	0.333	5
41400122	1610612741	CHI	Chicago	201959	Taj Gibson			11:00	0	0	0	0	0	0	0
41400122	1610612741	CHI	Chicago	203503	Tony Snell			13:21	1	3	0.333	1	3	0.333	0
41400122	1610612741	CHI	Chicago	202703	Nikola Mirotic			22:21	3	9	0.333	1	3	0.333	1
41400122	1610612741	CHI	Chicago	201166	Aaron Brooks			9:55	2	3	0.667	1	2	0.5	0
41400122	1610612741	CHI	Chicago	203946	Cameron Bairstow		DNP - Coach's Decision								
41400122	1610612741	CHI	Chicago	203926	Doug McDermott		DNP - Coach's Decision								
41400122	1610612741	CHI	Chicago	1737	Nazr Mohammed		DNP - Coach's Decision								
41400122	1610612741	CHI	Chicago	202734	ETwaun Moore		DNP - Coach's Decision								

Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
FTM	FTA	FT_PCT	OREB	DREB	REB	AST	STL	BLK	TO	PF	PTS	PLUS_MINUS
2	2	1	2	9	11	4	2	2	2	4	6	-1
2	2	1	1	5	6	0	0	1	0	1	8	-6
1	2	0.5	1	5	6	3	1	2	0	2	7	-2
3	3	1	1	5	6	0	1	1	1	2	22	-13
2	3	0.667	0	5	5	2	1	1	1	4	12	-6
0	1	0	3	3	6	0	1	2	0	4	8	-7
0	0	0	1	2	3	0	4	0	0	1	3	2
2	2	1	0	2	2	1	0	0	0	2	8	-3
2	2	1	0	3	3	3	0	0	0	2	8	-9
0	0	0	1	5	6	3	0	0	0	1	12	4
3	6	0.5	2	14	16	3	0	4	3	1	11	4
0	0	0	4	15	19	5	0	1	3	3	6	0
8	14	0.571	2	7	9	2	0	1	0	2	31	12
5	5	1	3	4	7	9	1	1	3	3	15	8
0	0	0	1	0	1	0	0	3	3	4	0	-4
0	0	0	0	2	2	1	0	0	1	2	3	9
1	2	0.5	1	3	4	1	1	0	0	2	8	11
0	0	0	0	0	0	2	0	0	0	1	5	1

包括了对战球队（芝加哥公牛队与密尔沃基雄鹿队），球员名称与 ID，首发球员位置，DNP 原因，出场时间，进球数，出手数，命中率，三分进球数，三分球命中率，罚球命中数，罚球出手数，罚球命中率，进攻篮板，防守篮板，总篮板，助攻，抢断，封盖，犯规数，得分，以及正负值。这些数据非常有用，对于分析具体的球员表现，分析重点比赛场次很有意义。

基于这两个 API，我们可以在几十种不同的模块中收集很多的数据，NBA 一直以来都提供非常具体充实的数据供球迷查询分析，供专家预测，供球队制订战

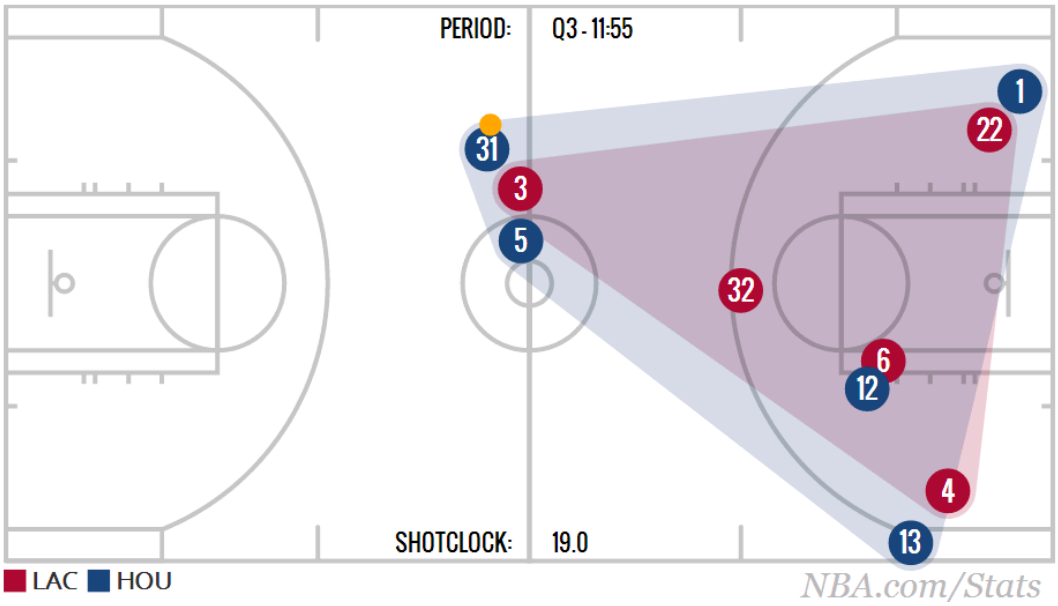
术与建队策略。可以说我们的数据收集到这一步，已经完成了‘大数据’的要求，每年的数千场比赛的数百种数据均可收集，我们的挑战会来自于整理数据所花费的时间成本已经梳理数据结构并进行分析的模型复杂性。

三 基于位置的‘大数据’收集

我们并没有将数据收集满足于上述步骤。经过我们对国内外主流篮球论坛的分析，借鉴了众多对比赛分析预测的经验与失误之后，认识到传统数据分析的最大问题（不管是最基本的数据分析还是包含了上一部分的‘大数据’分析），就是在于数据并不能反映球场的真实情况。一场比赛 48 分钟（不计加时），球场局势瞬息万变，往往因为一个球员的防守或者进攻问题（防守不好，对位被针对，投篮失准被防空，被夹击）等等问题以及战术调整等等产生剧烈的变化。哪怕数据统计‘精确到’每场，依然远不足以反映球场形势。仅举几个简单的例子，去年总决赛 G7 最后将近五分钟的时间内，勇士与骑士一分未得，然而这五分钟两队的攻防令人窒息，相当精彩，在数据上却完全无法体现，更不用说伊戈达拉的快攻如何被詹姆斯封盖，欧文的单打三分如何一剑封喉等等瞬间，均无法在纯数字数据中得到体现。今年的常规赛中骑士队因为球员年龄较大，詹姆斯习惯性常规赛留力，不过分追求战绩等问题，很多比赛数据相当不好看，但是季后赛却继续横扫对手，统治力十足。可见数据是会骗人的，基于基本的数据做出的预测往往会令人质疑，但是这些主观的球场的表现，又无法被量化，因此产生了本段阐述的，数据无法反应真实表现的情况。

我们在寻找数据的过程中发现了这样的一个基于 python 绘制球员实时轨迹

的程序。可以显示球员在场上的实时位置，相当于动画模拟了球员的场上表现，这为我们分析球员的真实表现，分析球队的具体战术产生了启发。



（http://savvastjortjoglou.com/nba-play-by-play-movements.html）（原图像为长度为数秒的动图）

我们研究该程序后发现，该程序的位置信息来源同样来自于 stats.nba，不过是一种最近几年新引进的技术 SportVU。该技术通过在球场架设多部超高速超清专用摄像机，每秒捕捉球场动态 25 次，来获得过去无法获得的无比准确丰富的球场信息。然而进一步的寻找令人失望，该数据并未向公众提供，无法通过 API 获取。

为此我们首先联系了上述文章的作者，经过询问后被告知，在 SportVU 技术的早期推广阶段可以获取该数据，最近几年的数据已经不再公开。但是在 https://github.com/savvastj/BasketballData/tree/master/2016.NBA.Raw.SportVU.Game.Logs 页面我们发现了 2016 年上半年的部分场次的 SportVU 打包数据。我们下载了其中一场比赛的数据做初步的分析。我们发现该数据非常的庞大，一场数据即可产生约 70M 数据，数据格式为 json，我们将做初步的数据处理，然后

存储到 csv 文件中。

```
#载入 json 数据并存储到 csv 文件中
import json
import numpy as np
import pandas as pd
with open('data.json', 'r') as f:
    data = json.load(f)
headers=["team_id", "player_id", "x_loc","y_loc", "radius","moment",
"game_clock", "shot_clock"]
    # 初始化新列表 player_moments
player_moments= []
for data1 in data:
    data1m=data1["moments"]
    for moment in data1m:
        for player in moment[5]:
            player.extend((data1m.index(moment),moment[2],
moment[3]))
            player_moments.append(player)
df =pd.DataFrame(player_moments, columns=headers)
# 创建 player 列表，将主队运动员的数据赋值给 player
players =data1["home"]["players"]
# 添加客队运动员的数据
```

```
        players.extend(data1["visitor"]["players"])
        # 创建新的字典 id_dict
        id_dict ={}
        # 在字典中增加我们想要的值,
        for    player in players:
              id_dict[player['playerid']]
=[player["firstname"]+""+player["lastname"],player["jersey"]]
        id_dict.update({-1:['ball', np.nan]})
        df["player_name"]= df.player_id.map(lambda x: id_dict[x][0])
        df["player_jersey"]= df.player_id.map(lambda x: id_dict[x][1])
        df.to_csv('nnba.csv',sep=',',
na_rep='',float_format=None,header=True,index=True)
```

打开 csv 文件可以看到一场比赛会产生约 100 万行的海量数据，该数据为每 0.04 秒一次采样，包括在场十位球员的姓名，球衣号，位置坐标，篮球的坐标与高度，基于这样的表格，我们可以利用已经实现的程序绘制正常比赛的动画图像，分析具体的球场细节信息。

	team_id	player_id	x_loc	y_loc	radius	moment	game_clock	shot_clock	player_name	player_jersey
0	-1	-1	23.4254	45.12734	3.64299	0	711.26	11.99	ball	
1	1610612761	2449	19.08811	13.91147	0	0	711.26	11.99	LuisScola	4
2	1610612761	201960	10.11935	13.54703	0	0	711.26	11.99	DeMarreCarroll	5
3	1610612761	200768	20.81838	44.51006	0	0	711.26	11.99	KyleLowry	7
4	1610612761	201942	14.16535	38.7957	0	0	711.26	11.99	DeMarDeRozan	10
5	1610612761	202685	20.89057	34.20938	0	0	711.26	11.99	JonasValanciunas	17
6	1610612766	101107	13.30744	22.19981	0	0	711.26	11.99	MarvinWilliams	2
7	1610612766	201587	9.1404	18.58649	0	0	711.26	11.99	NicolasBatum	5
8	1610612766	202689	19.31103	42.47285	0	0	711.26	11.99	KembaWalker	15
9	1610612766	203469	16.44724	32.53167	0	0	711.26	11.99	CodyZeller	40
10	1610612766	203798	15.49183	37.7022	0	0	711.26	11.99	PJHairston	19
11	-1	-1	24.0045	44.6889	2.99994	1	711.22	11.97	ball	
12	1610612761	2449	18.93523	13.8225	0	1	711.22	11.97	LuisScola	4
13	1610612761	201960	9.98343	13.71795	0	1	711.22	11.97	DeMarreCarroll	5
14	1610612761	200768	21.2291	44.29041	0	1	711.22	11.97	KyleLowry	7
15	1610612761	201942	13.59663	38.65339	0	1	711.22	11.97	DeMarDeRozan	10
16	1610612761	202685	20.99513	34.48187	0	1	711.22	11.97	JonasValanciunas	17
17	1610612766	101107	13.12847	22.14854	0	1	711.22	11.97	MarvinWilliams	2
18	1610612766	201587	8.97947	18.67319	0	1	711.22	11.97	NicolasBatum	5
19	1610612766	202689	19.69412	42.27407	0	1	711.22	11.97	KembaWalker	15
20	1610612766	203469	16.46477	32.57282	0	1	711.22	11.97	CodyZeller	40
21	1610612766	203798	15.07591	37.49843	0	1	711.22	11.97	PJHairston	19
22	-1	-1	24.62517	44.30089	2.24555	2	711.18	11.95	ball	

为了获得完整的数据，挑选我们认为重要的比赛进行分析，我们访问了 https://www.stats.com/sportvu-basketball-media/?lang=zh-hans 网站来试图寻找获得 SportVU 数据的方法。我们联系了 STATS 公司的工作人员 Sales Development Representative Gabrielle DuFour。她在了解了我们的想法之后告知我们如果是用于科教领域的研究，可以申请使用该数据，她帮助我们联系了负责 SportVU 技术的主管 Charlie Rohlf。我们需要提供一份 proposal，由于 Rohlf 下周才能出差回来，因此我们需要再下周再做具体的联系，因此为了获取完整的数据使用资格，我们接下来需要再写一份英文版的方案。

四 利用深度学习进行预测

深度学习将会成为我们此次模型预测中的最强有力的工具。分析具体的比赛不但费时（几个人无法看完几千场比赛来做出分析），而且过于主观，容易产生大量的疏漏，而这种经验性质的问题利用近几年发展起来的强大的图像分析模型 CNN 将会非常有效，我们将会将位置数据输入模型，让机器学习球场的情况，球队的战术，学会分析球场情况，当我们输入最新的球队比赛时可以做出最佳的判断。

在开题报告中我们提到，如果希望使用深度学习的模型，最大的问题在于数据量的问题，太小的数据量是不足以满足深度学习的要求的。可以看到上一部分不仅解决了数据量的问题，甚至还带来了超量的数据。因此我们要进行一系列的处理。

我们希望选择 CNN 来分析一场比赛的整个过程，我们知道 CNN 在图像识

别领域大放异彩，如果只是把真实的比赛录像输入模型，不仅数据量巨大（高大数 GB），而且会由于摄像机位等问题无法总览比赛，而我们上一部分收集的基于 SportVU 的数据将解决此问题，让我们利用 CNN 分析比赛的希望变为可能。

在使用 CNN 作为模型的目标确定后，我们还需要确定如何输入数据，以及 label 的问题，如果我们把一整场的数据输入后，仅以球队是否获胜作为 label，那就浪费了如此海量的数据，因此我们希望以每次得分作为 label，这将满足我们利用深度学习来分析战术的目的。这样机器将会学习大量的战术，以及通过分析战术来判断能否得分以及得分的数量。将整场的比赛累加即可得到最终的分数预测，达到预测胜负的目的。注意到这个模型考虑到了非常精细的球员对位和对手信息，因此所做的预测会远比传统模型真实，更具有信服度。

我们首先需要解决数据标注和输入问题，原始数据中并未提供得分变化，因此我们还需要解决该问题，由于数据较复杂，如何划分样本，输入数据也需要仔细的考虑。经过进一步的思考之后我们提出了两种解决得分问题的方法：一是向相关数据公司咨询能否得到相关得分信息，二是利用已有的数据分析出是否得分: 一共有两个方法，一个是追踪球的三维坐标变化，如果球的轨迹穿过了篮筐则认为进球，但是该方法的缺陷是球的速度较快，哪怕在 0.04 秒的时间内也会有较大的坐标变动，因此可能无法准确判断球是否穿过了篮筐，二是根据计时器以及球员位置判断，即根据 24 秒计时器判断，如果计时器重新回到 24 秒，则有两种情况：球投中了，球权转化，此时两队球员的坐标会逐渐移动到另一个半场，二是球没有投进，进攻方抢到了进攻篮板球，球员坐标还在本半场，或者球没有投进，但是防守方抢到了防守篮板球，球员坐标移动到另外半场。注意我们区别投中后球权转化与没投中对方抢到防守篮板的球权转化的方法是 :投中的情况下另

一方会在球场外发球，我们可以根据这一点加以区分。

由上面的分析可知，基于数据本身的坐标信息判断得分与否为我们带来了很多额外的工作量，我们还是希望能够直接获得得分数据本身。另外，上述分析为我们的比赛划分提供了新的思路：我们可以依靠'24 秒'对比赛进行分割，不管是投进，还是没投进但是抢到了进攻篮板，没投进抢到了防守篮板，计时器都会重新回到 24 秒，基于这样的划分我们可以给数据不同的标注，作为模型学习到战术的反馈。

经过分析后我们发现，CNN 等模型擅长的是对图像的特征提取，如果我们仅仅输入数据本身，CNN 可能无法提取到任何有效特征，因此我们将会先把整场数据转化为动态图像，当做真实比赛的模拟，然后让 CNN 分析这样的一场比赛。这为我们提出了一系列的挑战，首先是数据量的问题，经过图像化后的数据量可能会达到上百兆，由于计算能力的问题我们只能选取少部分重要比赛，或者某些比赛的重要时刻进行分析，而无法做大量比赛的分析。二是我们将不能使用传统的进行二维图像分析，而必须寻找适合视频分析的模型，这方面的研究还不是很多。传统的 CNN 模型只用解决空域的问题，而我们必须处理空域与时域的问题，因此必须选择使用 3D-CNN 类型的模型。由于我们的数据是时序相关的，利用 RNN 也是非常常见的选择，我们将会阅读相关论文以及一些已经实现的代码。3D-CNN 识别视频因为比图片识别在标注和计算量，数据量等问题上都难很多，还要考虑时间相关性等因素，效果并没有 CNN 那么神奇，但是考虑到我们的数据是较为简单的点状模拟图，已经高度抽象，因此对于模型的规模和深度都不会有非常大的要求，效果也许会提高一些。

另外，我们认为可以在已有数据中增加更多的内容，以实现我们对'分析战术'

的要求，我们可以很容易地爬取到球员本场比赛的数据，以及基本的身体信息及过往对战，赛季数据等等数据信息，这些信息可以让模型更有经验地分析对位信息，但是如何把这些信息嵌入进输入数据中也是我们接下来要考虑的问题。

因此我们接下来会重点解决两大类问题：

①样本问题：样本划分（以 24 秒划分），数据标注（得分以及更多更有利于战术分析的数据），实现数据向图像的转换并解决存储和输入问题。

②模型问题：尝试 3D-CNN，CNN+RNN 等适合视频识别的模型。我们将使用台式机及带有 GPU 的集群测试模型，由于任务需求的特殊性，在模型上我们会有较大的改进的余地。

五 基于媒体报道的补充分析

为了充分的考虑各种影响比赛的因素，我们决定进一步加入场外因素的影响。我们认为媒体的报道不仅会影响球员的心态，也可以为我们提供额外的信息进行分析。

我们首先选定 twitter 作为我们获得信息的来源。通过推特提供的模块 tweepy，我们可以爬取特定用户的最多 3200 条最近动态，以 NBA 官方账户为例。

```
import tweepy
import csv
#Twitter API credentials
consumer_key = "kaFo5gLtx3KiUJO7ZEjVDbSb5"
```

```
    consumer_secret                                                    =
"OW7Ab1BqaFwBFY98IwesOEzdjXz7Nvx8NDXhemtA22wKVt9miM"
    access_key                =                "857083088559525888-
35Ycl5LYlDkHDv2TBDhMuST6t5xfThK"
    access_secret                                                      =
"hHCdBhznV48aWZg1OoFpWy8PV4S48140D1IqL6FiToJjh"



    def get_all_tweets(screen_name):
        #Twitter only allows access to a users most recent 3240 tweets with
this method

        #authorize twitter, initialize tweepy
        auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
        auth.set_access_token(access_key, access_secret)
        api = tweepy.API(auth)

            #initialize a list to hold all the tweepy Tweets
        alltweets = []

            #make initial request for most recent tweets (200 is the maximum
allowed count)
```

```
        new_tweets = api.user_timeline(screen_name = screen_name,count=200)

            #save most recent tweets
        alltweets.extend(new_tweets)

            #save the id of the oldest tweet less one
        oldest = alltweets[-1].id - 1

            #keep grabbing tweets until there are no tweets left to grab
        while len(new_tweets) > 0:
            print("getting tweets before %s" % (oldest))

                    #all subsiquent requests use the max_id param to prevent duplicates
            new_tweets = api.user_timeline(screen_name = screen_name,count=200,max_id=oldest)

                    #save most recent tweets
            alltweets.extend(new_tweets)

                    #update the id of the oldest tweet less one
```

```
            oldest = alltweets[-1].id - 1

            print("...%s tweets downloaded so far" % (len(alltweets)))

            #transform the tweepy tweets into a 2D array that will populate the csv
            outtweets = [[tweet.id_str, tweet.created_at, tweet.text.encode("utf-8")] for tweet in alltweets]

            #write the csv
            with open('%s_tweets.csv' % screen_name, mode='w', encoding='utf-8') as f:
                writer = csv.writer(f)
                writer.writerow(["id","created_at","text"])
                writer.writerows(outtweets)

                pass


    if __name__ == '__main__':
        #pass in the username of the account you want to download
        get_all_tweets("NBA")
```

我们实现了爬取 NBA 官方账户的文本信息并存储到 csv 中，可以看到 twitter 时间与文本信息。不过文本信息的处理要比纯数据信息复杂得多，我们希望使用一些基本的 NLP 知识提取一些可用的有效的信息。我们还将继续爬取各支球队的账户，重点球员的账户以及各体育媒体的账户，作为我们预测模型的补充分析。

六 待处理工作与后续安排

最近一段时间我们的主要工作都集中于初步建立传统的分析模型，搜集各种数据，学习爬虫、学习深度学习相关知识等工作。为了获得最佳的预测效果，我们在数据收集整理阶段下了较大的功夫。

接下来一段时间我们会进行更具体的数据整理与建模工作，我们会将更具体的数据、基于位置的精准数据以及文本数据进行处理并撰写 proposal 以获得 stats.nba 提供的 SportVU 数据。同时我们还将进一步深入理解深度学习的相关模型，学习调整模型结构与参数调优，掌握更多的数据挖掘相关知识与工具，学习简单的自然语言处理知识等，最终再将各部分模型综合起来进行分析，做出合理的预测，可谓任重而道远。

时间表

第十二周	纯数字型大数据整理完毕，建立模型预测。进一步联系并获取 SportVU 数据，进行数据整理及标注，学习 CNN 相关知识，尝试简单的模型。
第十三周	建立 CNN 学习 SportVU 数据。爬媒体数据，学习 NLP 相关工具
第十四周	CNN 调整与优化，测试 CNN 模型，分析媒体数据。
第十五周	综合各部分模型，对今年进行预测尝试，调整模型。
第十六周	撰写论文。

七 参考文献

【1】逻辑回归，http://blog.csdn.net/pakko/article/details/37878837

【2】NBA 常规赛结果预测——利用 Python 进行比赛数据分析

https://www.shiyanlou.com/courses/782/labs/2647/document

【3】stats.nba.com

【4】probasketballapi.com

【5】Python client for NBA statistics located at stats.nba.com

https://github.com/seemethere/nba_py

【6】Basketball Reference.com

【7】 周志华，《机器学习》，北京：清华大学出版社，2016，57—60