作業研究期末報告

預測 2018 NBA 總冠軍

使用程式：python

指導老師: 陸行 教授

彭羿凡 應數三 104701047

2018/6/25

目錄

1.研究目標

隨著 AI 和大數據在近期掀起熱潮，希望能利用機器學習的方　　式，利用以往比賽的數據預測 2018 年度的總冠軍球隊。

2.NBA 賽制簡介

比賽模式:例行賽、季後賽(七戰四勝制)、總冠軍賽(七戰四勝制)
比賽隊伍:例行賽-30 個隊伍 16 隊晉級季後賽
季後賽-分為首輪、二輪、分區決賽
總冠軍賽-2 隊決勝負

3.參數選取

資料來源為 2017~2018 年度
以下介紹部分重要參數:

(1)回合數

回合數(possession)通常廣泛的被運用在計算進攻節奏上,每發生一次球權轉換即一個回合的結束。

定義一個 Possession 只會在下列三種狀況下結束:

1.一個未被進攻方搶下籃板的投籃(進球或防守方搶到籃板)
2.失誤
3.罰球

要統計每一場比賽的球隊回合數需要根據 play-by-play 來逐條判斷，遇到進球、最後一次罰球命中、失誤等情況，則回合數+1。
一般會採用一個公式來估算回合數:

$$\frac{1}{2}\left\{\left[\left(\text{Opp FGA} + 0.4 \times Opp\ FTA - 1.07 \times \left(\frac{Opp\ ORB}{Opp\ ORB + Tm\ DRB}\right)\right)\right.\right.$$

$$\left.\times (Opp\ FGA - Opp\ FG) + Opp\ TOV\right]$$

$$+\left[\left(Tm\ FGA + 0.4 \times TmFTA - 1.07 \times \left(\frac{TmORB}{Tm\ ORB\ \ + Opp\ DRB}\right)\right)\right.$$

$$\left.\left.\times (Tm\ FGA - Tm\ FG) + Tm\ TOV\right]\right\}$$

Tm/Opp FGA：球隊/對手總出手數
Tm/Opp FTA：球隊/對手罰球總數
Tm/Opp ORB：球隊/對手進攻籃板總數
Tm/Opp DRB：球隊/對手防守籃板總數
Tm/Opp FG：球隊/對手總命中數

Tm/Opp TOV：球隊/對手總失誤數
這個公式基於球隊和對手雙方的數據來計算平均數而得，比起只計算一方的數據更加精確。

(2) 進攻效率值 Ortg(Offensive Rating)

球員在場上每 100 進攻回合(Possessions)所能得到的分數。定義是用來衡量球員在場上每 100 進攻回合(Possessions)所能得到的分數。基本上 ORtg 的概念就是把每個環節受到其他影響的部份給排除，例如每顆進球中受到助攻幫助的部分要獨立出來(ORtg 認為這部分是另一名球員助攻的貢獻)，以及每顆進攻籃板後球隊的得分效率等等，來推估球員在整個進攻端為球隊帶來的效益。

$$\text{公式：Ortg} = 100 \times \frac{PProd}{TotPoss}$$

PProd =球員在進攻端為球隊帶來的所有得分數x球隊能在一波進攻之內便完成進攻的效率 + 球員的進攻籃板帶來的得分效益。

TotPoss = 球隊在進攻端所有得到分數的回合數x球隊能在一波進攻內完成進攻的效率 + 球隊每顆進攻籃板之後有得分的回合數。
ORtg 的計算很繁雜，這也顯示 ORtg 為了將每個環節的效益獨立出來

(3) eFG%(effective Field Goal percentage，有效命中率)

$$\text{eFG\%} = \frac{(FGM + 0.5 \times 3PM)}{FGA}$$

FGM:投籃命中 (Field goals made)

FGA:投籃出手 (Field goals Attempted)

FG%:投籃命中率 (Field Goal Percentage)⟹FG%=FGM/FGA

一顆三分球能比一顆兩分球多得 1.5 倍的分數，所以將 FG%(FGM)中三分命中數乘上 1.5，也就是提高 1.5 倍的數字。推得一個結果 eFG%乘以出手數再乘 2 就是得分因為 FGA 是投籃出手，所以乘二就會是分數，三分球的部分，已經先在 eFG%裡已經先乘好 1.5 了。

(4)TS%(True Shooting percentage，真實命中率)

$$\text{TS\%} = \frac{PTS}{2 \times (FGA + 0.44 \times FTA)}$$

簡單來說 TS%比 eFG%多考慮了罰球。有些進攻機會，本來能夠以投籃出手，但因為被犯規而變成罰球出手，但以分數來看仍然是一次進攻機會，eFG%裡卻未計入。所以真實命中率就是希望能考慮到
所有的進攻機會，能夠更準確。

◆以下以定義在回合數(Possessions)下的參數進而得到百分比:
(1)TREB% Total Rebounds 總籃板
(2)OREB% Offensive Rebounds 進攻籃板
(3)AST% Assists 助攻
(4)STL% Steals 抄截
(5)BLK% Blocked shots 阻攻
(6)TOV% Turnover 失誤

◆以下為以單場比賽下的數據得到百分比
(1)2P%=2PM/2PA　二分命中率
(2)3P%=3PM/3PA　三分命中率
(3)FT%=FTM/FTA　罰球命中率

◆以下為非球員因素：
LOC (0、1) 客場、主場
球迷對於自家球隊的支持也是比賽的一大重點，通常擁有主場優勢的球隊，其勝率也較高。

◆總結所有數據：
所以總共計算的 13 個參數為

LOC、2P%、3P%、FT%、eFG%、TS%、Ortg、AST%、STL%、BLK%、 TREB%、OREB%、TOV%

4.程式介紹

使用軟體：Python
使用套件(主要)：keras(建立模型)、pandas(資料分析)

1. keras 可以幫助我們快速的將模型建立出來，可以藉由 Open source 的幫助，將複雜的程式碼簡單化，只需引入所需的套件，就可以完成我們所想要的模型。
2. pandas 在分析資料方面十分的方便，可以讀入 Excel、CSV、html…等等的檔案或是網頁，並且擁有 DataFrame 的功能，可以將資料表格化，對於資料的整理方便且快速。

5.研究方法

◆資料的輸入：

使用 2017~2018 年度的數據，總共有 82*30=2460 筆 data、季後賽所有數據。利用這些資料來建立一個適用於所有隊伍的模型，此模型會輸出球隊的勝率，所以我們可以得到當這個隊伍有此數據的表現時的勝率，藉由勝率的高低來比較兩隊應該是由誰勝出，我們可以把這個勝率當成分數，即分數越高者獲勝。此外，因為季後賽是一個系列戰(7 戰 4 勝)，我們認為以下的方法可以提高預測的準確率，在季後賽的第一輪中，利用例行賽的 data 之常態分佈來模擬真實的 data，因為球員的表現在每一場的比賽並不相同，球員的狀態可能時好時壞，故我們認為這一種方法可以更接近真實的狀況，而季後賽的第二輪以第一輪來模擬，以此類推。至於會使用前一輪數據的原因是為了更接近球隊近況的表現。

◆類神經網路的運作：

1. 每一層的 input(X_i)會和權重(W_{ij})相乘，得到 score(s)，即：$s_i^{(l)} = \sum_{i=1} W_{ij}^{(l)} X_i^{(l-1)}$
2. Score 會被輸入到 Activation Function(σ)，Activation Function 就像一個開關，當 Score 大於某個閥值就可以向後傳遞。
3. 經 Activation Function 轉換後，我們可以得到一個結果： $h_i^{(l)} =$

$\sigma(\sum_{i=1} W_{ij}^{(l)} X_i^{(l-1)}+)$ ，$h_i^{(l)}$ 就是神經元

◆更新權重(W_{ij})的方法—Backpropagation

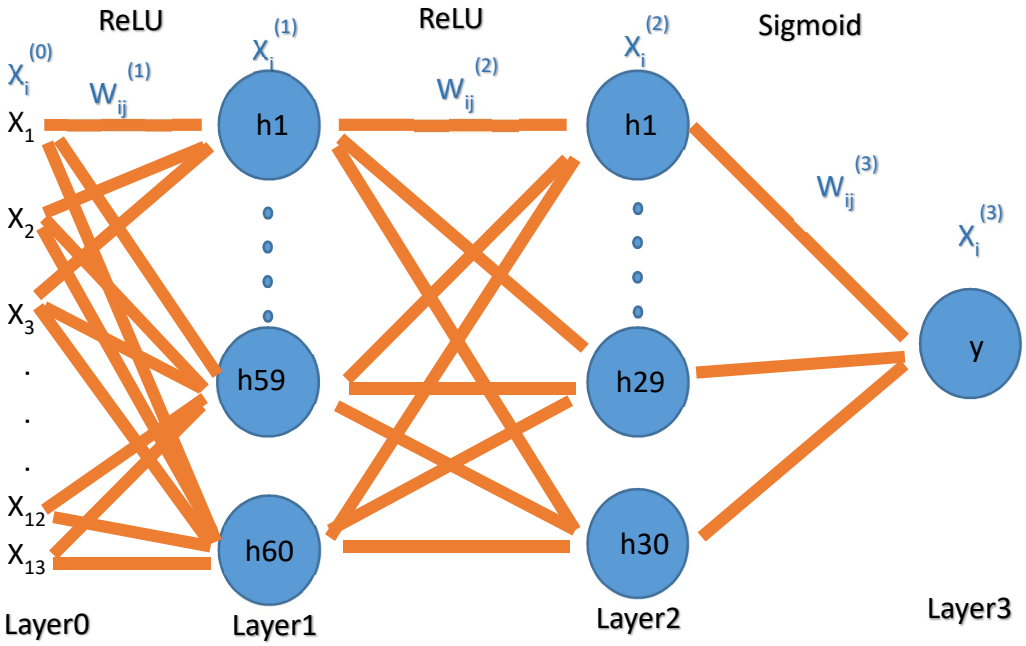
(1)Gradient Descent 的流程：

1. 定義出 Error 函數
2. Error 函數讓我們可以去評估E_{in}
3. 算出它的梯度∇E_{in}
4. 朝著∇E_{in}的反方向更新參數 W，而每次只跨出 η 大小的一步
5. 反覆的計算新參數 W 的梯度，並一再的更新參數 W

(2) 因此我們的更新公式可以表示成：

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta \times \partial L / \partial W_{ij}^{(l)}$$

◆建立出的 model 樣貌：



◆程式建模部分：

In [1]: 讀入我們需要的套件

```
%matplotlib inline
import pandas as pd
import numpy as np
```

In [2]: 讀入Data

```
df = pd.read_excel("2017-18_teamBoxScore.xlsx")
```

In [4]: 將原本資料的dataframe結構轉成array

```
ndarray = df.values
```

In [5]: 看看現在array的格式

```
ndarray.shape
```

Out[5]: (2460, 15)

In [6]: 我們定義一個函數叫做PreprocessData，將我們的資料分為Feature和Label兩部分

```
def PreprocessData(raw_df):
    ndarray = df.values
    Features = ndarray[:,2:15]
    Label = ndarray[:,1]
    return Features , Label
```

In [7]: 我們將Data分成train-data和test-data

```
from sklearn import preprocessing
msk = np.random.rand(len(df))<0.8     //將資料隨機分成總數的0.8和0.2
train_df = df[msk]  //訓練資料為資料總量的0.8
test_df = df[~msk]  //驗證資料為資料總量的0.2
train_Features,train_Label = PreprocessData(train_df)
test_Features,test_Label = PreprocessData(test_df)
print('total:',len(df) ,'train:',len(train_df),'test:',len(test_df))
```

Out[7] total: 2460 train: 1933 test: 527

In [8]: 用keras來建立我們的model

```
from keras.models import Sequential
from keras.layers import Dense,Dropout,Activation
```

In [9]:

```
model = Sequential()
```

In [10]: 加入第一層的神經網路，共13個input和60個神經元全線相連，activation用relu，並在每一次的訓練中遺忘30%的神經元

```
model.add(Dense(units = 60, input_dim = 13, activation = 'relu'))
```

```
model.add(Dropout(0.3))
```

In [11]: 加入第二層的神經網路，60個input和30個神經元全線相連，activation用relu，並在每一次的訓練中遺忘50%的神經元

```
model.add(Dense(units = 30, activation = 'relu'))
model.add(Dropout(0.5))
```

In [12]: 加入最後一層的神經網路，1個unit，即為output(勝或敗的機率)，activation function用sigmoid

```
model.add(Dense(units = 1, activation = 'sigmoid'))
```

In [13]: 最後我們設定loss function 為 binary_crossentropy ，優化的方法為adam

```
model.compile(loss='binary_crossentropy',optimizer='adam' ,metrics=
['accuracy'])
```

In [14]: 將我們的data丟入model中，訓練週期為10次，每一次丟入16筆的資料

```
train_history = model.fit(x = train_Features, y =
train_Label ,epochs=10 , batch_size=16 )
Epoch 1/10
2460/2460 [==============================] - 1s 470us/step - loss:
1.2439 - acc: 0.6362
Epoch 2/10
2460/2460 [==============================] - 0s 145us/step - loss:
0.3805 - acc: 0.8362
Epoch 3/10
2460/2460 [==============================] - 0s 143us/step - loss:
0.2275 - acc: 0.9053
Epoch 4/10
2460/2460 [==============================] - 0s 140us/step - loss:
0.1464 - acc: 0.9366
Epoch 5/10
2460/2460 [==============================] - 0s 146us/step - loss:
0.1045 - acc: 0.9589
Epoch 6/10
2460/2460 [==============================] - 0s 147us/step - loss:
0.0933 - acc: 0.9695
```

```
Epoch 7/10
2460/2460 [==============================] - 0s 146us/step - loss:
0.0774 - acc: 0.9772
2
Epoch 8/10
2460/2460 [==============================] - 0s 143us/step - loss:
0.0543 - acc: 0.9801
Epoch 9/10
2460/2460 [==============================] - 0s 152us/step - loss:
0.0330 - acc: 0.9870
Epoch 10/10
2460/2460 [==============================] - 0s 147us/step - loss:
0.0396 - acc: 0.9846
```

In [15]: 將我們的test-data丟入model中，看看我們訓練的準確率

```
scores = model.evaluate(x = test_Features , y=test_Label)
2460/2460 [==============================] - 0s 73us/step
```

In [16]: 看看我們model的準確率，高達99%呢！

```
scores[1]
Out[16]: 0.9955284552845528
```

6. 研究結果

◆程式預測部分：

In [68]:讀入我們要用來預測季後賽的data

```
BOS = df[200:246]
CLE = df[410:492]
GS = df[738:820]
HOU = df[820:902]
IND = df[902:984]
MIA = df[1230:1312]
MIL = df[1312:1394]
MIN = df[1394:1476]
```

```
NO = df[1530:1558]
OKC = df[1722:1804]
PHI = df[1804:1886]
POR = df[1968:2050]
SA = df[2175:2214]
TOR = df[2250:2296]
UTA = df[2296:2378]
WAS = df[2378:2460]
BOS_1 = pd.read_excel("BOS1.xlsx")
BOS_2 = pd.read_excel("BOS2.xlsx")
CLE_1 = pd.read_excel("CLE1.xlsx")
CLE_2 = pd.read_excel("CLE2.xlsx")
CLE_3 = pd.read_excel("CLE3.xlsx")
HOU_1 = pd.read_excel("HOU1.xlsx")
HOU_2 = pd.read_excel("HOU2.xlsx")
GS_1 = pd.read_excel("GS1.xlsx")
GS_2 = pd.read_excel("GS2.xlsx")
GS_3 = pd.read_excel("GS3.xlsx")
UTA_1 = pd.read_excel("UTA1.xlsx")
PHI_1 = pd.read_excel("PHI1.xlsx")
NO_1 = pd.read_excel("NO1.xlsx")
TOR_1 = pd.read_excel("TOR1.xlsx")
```

In [72]: 我們必須將data處理過後才能丟入model(即格式和先前相同)

```
def InputData(df_team):   //定義一個函數InputData
    df_team = pd.DataFrame(df_team)     //將資料轉為dataframe的形式
    df_team.rename(columns={'TEAM':'team', 'Home/Guest':'Loc',
    'AST%':'ASST%','TRB%':'TREB%','Diff':'teamEDiff','eFG%':'EFG%','O
    RB%':'OREB%','TOV%':'TO%'})     //將資料columns names重新命名
    df_team = np.matrix(df_team)    //將其轉為矩陣的形式
    df_team = df[0:][['2P%', '3P%', 'FT%', 'ASST%', 'TS%',
    'TREB%','STL%', 'BLK%', 'teamEDiff', 'EFG%', 'OREB%',
    'TO%']]          //取我們需要的部分
    df_team_analysis = df_team.describe() //看資料的統計數據
    df_team_analysis = df_team_analysis[1:3]  //取資料的平均和標準差
    df_team_noise=df_team_analysis.values   //將資料轉為array的形式
    g=[]    //我們想要一個陣列，可以幫助我們製造一些接近真實的數據
    for i in range(12):
```

```
        k = np.arange(12)
        g.append(("a_%s")%k[i])
        g[i] = np.random.normal(0 , 0.2 ,1)  //常態分布
    g = np.array(g)
    noise = []
    for j in range(12):
        l = g[j] * df_team_noise [1][j]
        noise.append(l)
    noise = np.matrix(noise)
    noise = noise.T    //noise矩陣的轉秩
    df_team_noise[0] = np.matrix(df_team_noise[0])
    return df_team_noise[0] + noise  //我們製造的data(近似真實資料)
```

In [20]: 主場優勢

```
def Home(team1):
    M = [1]
    Home_data = InputData(team1)
    Home_data = pd.DataFrame(Home_data)
    Home_data.insert(0,'Loc',M)
    Home_data_f = Home_data.values
    team1 = Home_data_f
    return team1
```

In [21]:客場

```
 def Guest(team2):
    N = [0]
    Guest_data = InputData(team2)
    Guest_data = pd.DataFrame(Guest_data)
    Guest_data.insert(0,'Loc',N)
    Guest_data_f = Guest_data.values
    team2 = Guest_data_f
    return team2
```

In [37]:兩隊系列賽(7戰4勝)的函數

```
def series(team1,team2):
    temp1 = InputData(team1)    //暫存
    temp2 = InputData(team2)    //暫存
    i = 0
```

```
    j = 0
    while (i < 4) and (j<4):    //任一隊都還沒贏4場
        team1 = temp1           //將team1_data初始化
        team2 = temp2           //將team2_data初始化
        //第1、2、5、7場team1有主場優勢
        if 1<(i+j)<4:           //當比賽為第3、4場時(team2主場優勢)
            team1 = Guest(team1)
            team2 = Home(team2)
        if 4<(i+j)<6:           //當比賽為第6場時(team2主場優勢)
            team1 = Guest(team1)
            team2 = Home(team2)
        else:                   //當比賽為1、2、5、7場(team1主場優勢)
            team1 = Home(team1)
            team2 = Guest(team2)
        team1_win_percent = model.predict(team1) //將team1_data代入model
        team2_win_percent = model.predict(team2) //將team2_data代入model
        if team1_win_percent > team2_win_percent: //team1贏
            i = i+1     //team1勝場+1
            print("%s : win , %s : lose " %(TEAM[0],TEAM[1]))
        if i == 4:      //team1贏得系列賽
            print("%s win the series" %TEAM[0])
            break
        else:               //team2贏
        j = j+1             //team2勝場+1
        print("%s : lose ,%s : win " %(TEAM[0],TEAM[1]))
        if j == 4:      //team2贏得系列賽
            print("%s win the series" %TEAM[1])
            break
```

Start to predict

我們自己模擬的季後賽！

◆Western conference first round

```
In [109]:
TEAM = ['GS','SA']
series(GS,SA)
TEAM = ['OKC','UTA']
```

series(OKC,UTA)
TEAM = ['HOU','WAS']
series(HOU,WAS)
TEAM = ['POR','NO']
series(POR,NO)
GS : win , SA : lose
GS : lose ,SA : win
GS : win , SA : lose
GS : win , SA : lose
GS : lose ,SA : win
GS : lose ,SA : win
GS : win , SA : lose
GS win the series
OKC : win , UTA : lose
OKC : win , UTA : lose
OKC : lose ,UTA : win
OKC : win , UTA : lose
OKC : lose ,UTA : win
OKC : lose ,UTA : win
OKC : win , UTA : lose
OKC win the series
HOU : win , MIN : lose
HOU : lose ,MIN : win
HOU : win , MIN : lose
HOU : lose ,MIN : win
HOU : win , MIN : lose
HOU : win , MIN : lose
HOU win the series
POR : win , NO : lose
POR : lose ,NO : win
POR : lose ,NO : win
POR : lose ,NO : win
POR : lose ,NO : win
NO win the series

◆Eastern conference first round

In [110]:

```
TEAM = ['CLE','IND']
series(CLE,IND)
TEAM = ['PHI','MIA']
series(PHI,MIA)
TEAM = ['BOS','MIL']
series(BOS,MIL)
TEAM = ['TOR','WAS']
series(TOR,WAS)
CLE : win  , IND : lose
CLE : lose ,IND : win
CLE : win  , IND : lose
CLE : lose ,IND : win
CLE : win  , IND : lose
CLE : win  , IND : lose
```
CLE win the series
```
PHI : lose ,MIA : win
PHI : win  , MIA : lose
PHI : win  , MIA : lose
PHI : lose ,MIA : win
PHI : lose ,MIA : win
PHI : lose ,MIA : win
```
MIA win the series
```
BOS : lose ,MIL : win
BOS : win  , MIL : lose
BOS : win  , MIL : lose
BOS : lose ,MIL : win
BOS : lose ,MIL : win
BOS : lose ,MIL : win
```
MIL win the series
```
TOR : win  , WAS : lose
TOR : win  , WAS : lose
TOR : win  , WAS : lose
TOR : lose ,WAS : win
TOR : win  , WAS : lose
```
TOR win the series

◆Western Conference Semifinals

```
In [111]:
TEAM = ['GS','NO']
series(GS_1,NO_1)
TEAM = ['HOU','OKC']
series(HOU_1,OKC)
GS : lose ,NO : win
GS : win  , NO : lose
GS : win  , NO : lose
GS : win  , NO : lose
GS : lose ,NO : win
GS : lose ,NO : win
GS : win  , NO : lose
```
GS win the series
```
HOU : lose ,OKC : win
HOU : win  , OKC : lose
HOU : win  , OKC : lose
HOU : lose ,OKC : win
HOU : win  , OKC : lose
HOU : win  , OKC : lose
```
HOU win the series

◆Eastern Conference Semifinals

```
In [112]:
TEAM = ['MIA','MIL']
series(MIA,MIL)
TEAM = ['TOR','CLE']
series(TOR_1,CLE_1)
MIA : win  , MIL : lose
MIA : win  , MIL : lose
MIA : lose ,MIL : win
MIA : win  , MIL : lose
MIA : lose ,MIL : win
MIA : lose ,MIL : win
MIA : win  , MIL : lose
```
MIA win the series
```
TOR : lose ,CLE : win
TOR : win  , CLE : lose
```

TOR : lose ,CLE : win
TOR : lose ,CLE : win
TOR : lose ,CLE : win
CLE win the series

◆Western Conference Finals

In [113]:
TEAM = ['HOU','GS']
series(HOU_2,GS_2)
HOU : lose ,GS : win
HOU : lose ,GS : win
HOU : lose ,GS : win
HOU : lose ,GS : win
GS win the series

◆Eastern Conference Finals

In [114]:
TEAM = ['CLE','MIA']
series(CLE_2,MIA)
CLE : win , MIA : lose
CLE : win , MIA : lose
CLE : lose ,MIA : win
CLE : lose ,MIA : win
CLE : win , MIA : lose
CLE : win , MIA : lose
CLE win the series

◆NBA FINALS

In [116]:
TEAM = ['GS' ,'CLE']
series(GS_3,CLE_3)
GS : win , CLE : lose
GS : win , CLE : lose
GS : lose ,CLE : win
GS : win , CLE : lose

GS : win , CLE : lose

GS win the series

若是只預測總冠軍(這才是我們真正的目標！)

在真實的2018季後賽中：

西區第一輪由GS、HOU、UTA、NO勝出，

東區第一輪由CLE、BOS、PHI、TOR勝出，

西區第二輪由GS、HOU勝出，

東區第二輪由CLE、BOS勝出，

西區決賽由GS勝出，

東區決賽由CLE勝出，

故總冠軍賽為GS VS CLE！

In [117]:

```
TEAM = ['GS' ,'CLE']
series(GS_3,CLE_3)
```

GS : lose ,CLE : win

GS : win , CLE : lose

GS : win , CLE : lose

GS : win , CLE : lose

GS : win , CLE : lose

GS win the series

我們藉由model的預測大膽猜測：GS會贏得NBA總冠軍

	實際情形				預測情形			
西區第一輪	UTAH JAZZ	NEW ORLEANS PELICANS	HOUSTON ROCKETS	GOLDEN STATE WARRIORS	HOUSTON ROCKETS	GOLDEN STATE WARRIORS	NEW ORLEANS PELICANS	OKC THUNDER
東區第一輪	CLEVELAND CAVALIERS	TORONTO RAPTORS	BOSTON CELTICS	PHILADELPHIA 76ERS	CLEVELAND CAVALIERS	TORONTO RAPTORS	MIAMI HEAT	MILWAUKEE BUCKS
西區第二輪	GOLDEN STATE WARRIORS	HOUSTON ROCKETS			HOUSTON ROCKETS	GOLDEN STATE WARRIORS		
東區第二輪	BOSTON CELTICS	CLEVELAND CAVALIERS			CLEVELAND CAVALIERS	MIAMI HEAT		
西區決賽	GOLDEN STATE WARRIORS				GOLDEN STATE WARRIORS			
東區決賽	CLEVELAND CAVALIERS				CLEVELAND CAVALIERS			
總冠軍賽	GOLDEN STATE WARRIORS	CLEVELAND CAVALIERS			GOLDEN STATE WARRIORS	CLEVELAND CAVALIERS		

預測結果與實際情形相同，GS 獲得了總冠軍。

7.結論

基本上我們可以從模型中預測出比賽的結果，雖然在一個系列戰中，每一場的勝負不見得相同，但是若是把標準放寬一點，我們的模型在預測哪一隊贏得系列戰上，準確率算是高的(10/14)。

從結果發現無法100%預測，原因可能如下：

1. 裁判影響
2. 傷病因素
3. 教練戰術的使用
4. 使用的輸入參數不夠多

8.研究感想與心得

經過這一次的報告，原本對於機器學習領域就有興趣的我，更加了解機器學習的內容，也看了一些比較難的數學推導，推導的過程並不好理解，相信以後數學更好之後，讀起來可以比較得心應手。這一次也第一次嘗試了自己從網路上蒐集資料，並且用 Python 整理資料，因為是第一次嘗試，所以我用了最簡單的模型，但做出來的結果也不錯，我覺得以上都不是最難的部分，困難點在於如何用新的資料來預測比賽結果，因為資料有些複雜，所以並不能直接將資料直接餵給模型，需要另外打很多的函數才能把資料變成我想要的樣子，模型預測的結果與事實並不完全相符，我個人覺得研究的對象也是一大重點，這次研究 NBA 比賽結果並不是一個有著一定規律的現象，有太多的人為因素會左右比賽的結果，並不是光憑數據就能確定哪一隊會獲勝，所以建立出來的模型也只能大致上預測出結果，甚至有可能會造成強隊恆獲勝的結果。在我考完研究所之後，希望有機會可以多接觸資料科學這一塊領域，當然也要加強自己的數學能力，多讀一些線性和非線性規劃的書，好讓自己以後可以在這塊領域得心應手！

9.參考文獻

1. https://www.youtube.com/watch?v=Gcm4F2bsw-U
2. https://www.sportsv.net/articles/45633(運動視界)
3. https://www.basketball-reference.com/about/ratings.html
4. https://www.ptt.cc/bbs/Jeremy_Lin/M.1399527869.A.166.html
5. https://nba.hupu.com/wiki/%E5%9B%9E%E5%90%88

6. https://www.sportsv.net/articles/45609(運動視界)
7. http://mropengate.blogspot.com/2015/06/ch15-4-neural-network.html
8. http://si.secda.info/buss-math/index.php/2013-01-12-15-28-58/2012-09-23-07-08-48
9. Deep Learning：用 Python 進行深度學習的基礎理論實作
   ISBN：9789864764846 作者：斎藤康毅　譯者：吳嘉芳
   出版社：歐萊禮　出版日期：2017/08/17
10. Deep Learning　　ISBN-13:9780262035613
   作者：Ian Goodfellow, Yoshua Bengio, Aaron Courville
   出版商:The MIT Press　出版日期:2016-11-18

10.工作分配

書面整理：應數三 蘇証皓
PPT 統整：應數四 103701050 業儒
回合數(PPT 白片＆書面)：應數三 104701048 賴炳勳
ORTG PPT(白片＆書面)：應數三 104701049 郭傳浩
剩下 10 項參數(PTT 白片＆書面)：應數三 鄭凱澤
程式(含 PPT 白片＆書面)：應數三 104701047 彭羿凡