Final Project 3

使用 DNN 預測 NBA 明星賽球員

徐嘉澤

摘要

NBA 每個賽季會在季中的時候進行全明星賽，會由觀眾、記者投票選出兩個分區的明星球員（共 24 位），得票數反應了該球員的「人氣」。

我們的猜想是 該球員的數據 和 觀眾投票的意向 呈某種程度的正相關，場上成績較好的球員可能會比較容易被選為全明星。所以我們的目標是蒐集球員在明星賽前的成績，使用 DNN 來預測該球員是否可以進入明星賽，屬於一個二元分類的問題。

1. 研究動機與目的

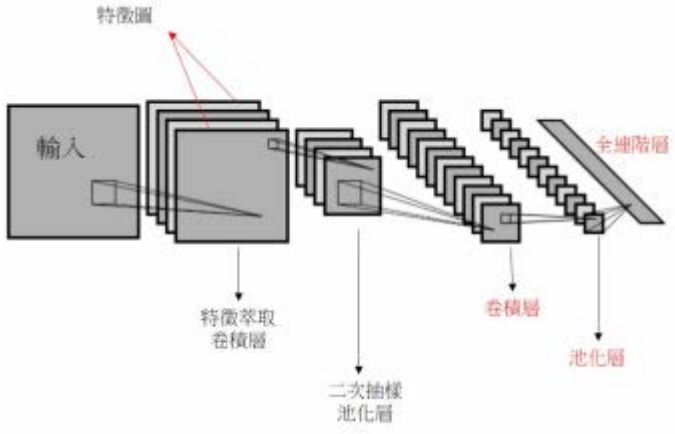
每一次的 NBA 賽紀開始我與幾位同學都會開始推崇自己喜愛的球員可以被選上明星賽，有些球員明明表現平平卻可以被選上參賽明星賽，也有些球員明明表現很好卻未被選上，於是在這樣的情況下讓人不禁開始思考其中緣故究竟為何？於是根據近幾年深度學習的興起，本次研究將從官方網站中擷取每個年份的球員數據加以預測是否會被選上明星賽，雖然誰被選上並無實質上的意義，但是為喜愛的事情上添增一份生活的情趣，我想研究的最大目的不僅僅是可以獲得實質的回饋，而是能貫徹自己所愛的事物加以深入了解。

而此次的研究將使用 DNN 的方式進行訓練及預測，最終期望可以研究出一模型可以準確預測當年度被選上明星賽的球員。

2. 文獻回顧

2.1 DNN

LeCun 在 1995 年提出卷積神經網路的基本架構及運算，由卷積、二次抽樣、全連接層組成，用來作為手寫文件的辨識方法。



圖一、卷積神經網路基本架構

Krizhevsky 在 2012 年的影像辨識大會提出深度卷積神經網路 Alexnet 及仰賴 GPU 來進行運算，以高準確率的便是奪得冠軍，獲得各界之矚目。

Abel-Hamid 在 2014 年的語音辨識運用卷積神經網路並分析其三個主要特質:局部性權重分享、池化在語音辨識的好處，其中針對資料須適時選擇池化方式。

- 卷積層

卷基層的功用在於吋曲特爭執，一個卷積運算包含輸入、濾波器、濾波器在移動時的步伐長度、濾波器在移動時的填充長度、輸出。圖二為卷積運算後的維度變化。

$$out = \frac{input - filter + 2 * pad}{stride} + 1$$

圖二、計算卷積運算後之維度變化公式

- 池化層

池化層，即 LeNet 中的二次抽樣有平均池化跟最大池化，目的是為了將資料量減少並保留重要資訊的方法，把原本的衣料做一個最大化或是平均化的降為計算。一鶻池化運算包含輸入、濾波器、濾波器在移動時的不乏長度、輸出。圖三為池化運算後的維度變化。

$$out = \frac{input - filter}{stride} + 1$$

圖三、池化運算後的維度變化公式

2.2 優化器

訓練深度學習網路，選擇適當的優化器是十分重要的一環，優化器將用於反向傳播理論中，來加速神經網路的訓練，使訓練時間能縮短並訓練出最佳模型。而本次研究使用的優化器為”ADAM”如圖二所示：

$$W \leftarrow W - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

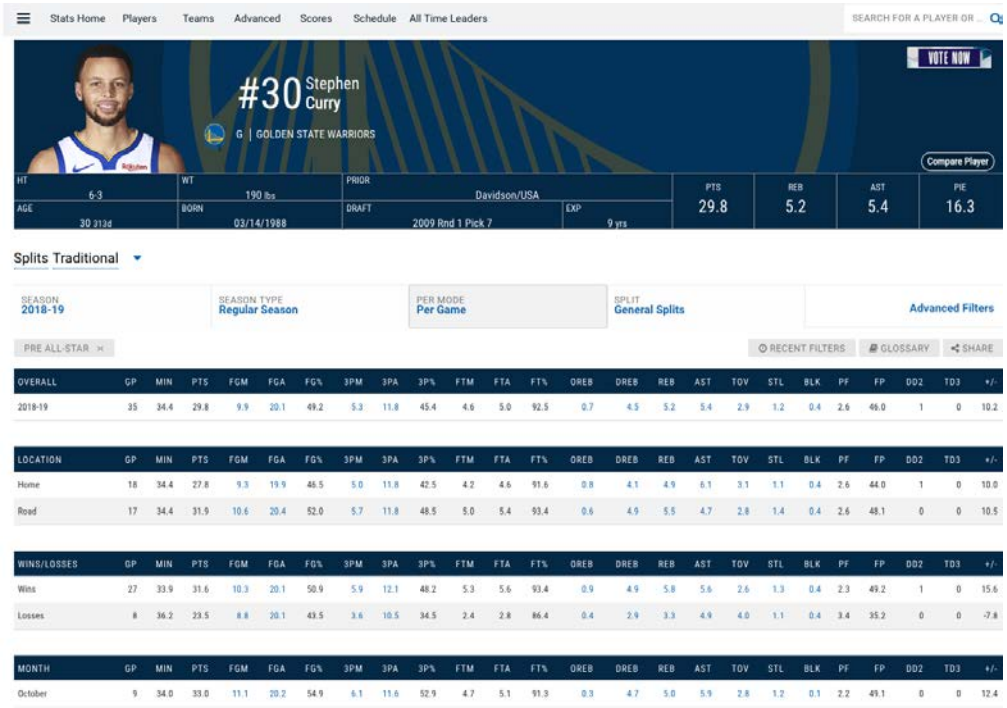
圖二、ADAM 演算法

ADAM 保留了 MOMENTUM 對過去梯度的方向做梯度速度調整與 ADAM 對過去梯度的平方值做 LEARNING RATE 的調整，再加上 ADAM 有做參數的”偏離校正”，使得每一次的學習率都會有個確定的範圍，會讓參數的更新較為平穩。

3. 研究方法

3.1 蒐集數據

首先因為沒有現成的明星賽前 Dataset 只能設法從網路上爬蟲，大多數網站的資料都是整個年度的球員數據，只有在官方網站的球員資料庫才能找到相關的資料。 NBA 官網是動態載入的網頁，所以就用了 selenium 來自動抓取資料。



圖一、官方網站球員資料

蒐集了 01~18 年，327 筆明星賽球員，3200 筆非明星賽球員，共 3527 筆球員資料，分別有傳統、進階、雜項等等五個向度的數據，共 83 種。

蒐集的資料共有底下幾種欄位：

	allstar	name	pid	Overall	GP	MIN	PTS	FGM	FGA	FG%	3PM	3PA	3P%	FTM	FTA	FT%	OREB	DREB	REB	AST	TOV	STL	BLK
0	1	James Hardi	201935	17	50	35.8	31.3	9.3	20.8	44.8	4.1	10.7	38.4	8.6	9.9	86.5	0.5	4.6	5.1	9	4.2	1.8	0.7
1	1	Anthony Da	203076	17	51	36.4	27.4	10	18.6	54.1	0.8	2.1	36.7	6.5	7.9	82	2.4	8.3	10.7	2.4	2.1	1.3	2.1
2	1	LeBron Jame	2544	17	56	37	26.5	10.2	18.7	54.4	1.8	4.8	36.2	4.5	6.1	73.7	1.1	7	8.1	8.9	4.4	1.5	1
3	0	Damian Lilla	203081	17	51	36.5	26.1	8.5	18.9	44.7	3	8.2	37	6.2	6.7	92.4	0.7	3.8	4.5	6.6	3.1	1	0.4
4	1	Giannis Ant	203507	17	53	37	27.8	10.2	18.9	54	0.5	1.8	29	6.8	9.1	75.3	2.2	8.2	10.4	4.8	2.8	1.4	1.3
5	1	Kevin Duran	201142	17	50	34.5	26	9.2	17.6	52.3	2.6	6.1	42.1	5	5.6	88.6	0.5	6.3	6.8	5.5	3.1	0.8	1.9
6	1	Russell Wes	201566	17	57	36.2	25.4	9.5	21.5	44.4	1.3	4.4	29.6	5.1	7	72.6	1.7	7.6	9.4	10.4	4.6	1.9	0.2
7	1	Kyrie Irving	202681	17	53	32.8	24.7	9	18.5	48.6	2.7	6.7	39.7	4.1	4.6	88.8	0.5	3	3.6	5	2.3	1.1	0.3
8	0	LaMarcus Al	200746	17	54	33.9	22.4	8.9	17.7	50.1	0.5	1.4	32.9	4.2	5.1	83.5	3.4	5	8.4	2	1.5	0.5	1.2
9	0	Victor Oladi	203506	17	52	34.6	24.4	8.9	18.4	48.4	2.3	6.2	38.1	4.3	5.3	81	0.5	4.8	5.3	4.1	2.8	2.1	0.8
10	1	DeMar DeRo	201942	17	57	34	23.7	8.3	18.1	46	1.2	3.5	32.7	5.9	7.1	82.7	0.8	3.2	3.9	5.2	2.2	1.2	0.3
11	0	Joel Embiid	203954	17	44	31.4	23.7	8.3	17.1	48.7	1	3.3	29.7	6.1	7.7	78.5	2.2	8.9	11.1	3.1	4	0.6	1.8
12	0	Bradley Beal	203078	17	57	36.4	23.6	8.6	18.7	46.1	2.4	6.6	37	3.9	5	79.2	0.8	3.7	4.5	4.2	2.4	1.2	0.5
13	0	Lou William	101150	17	55	32.4	23.2	7.4	16.8	44	2.8	7.4	37.8	5.6	6.2	89.5	0.5	2.1	2.5	5.3	2.9	1.1	0.2
14	1	Jimmy Butle	202710	17	55	37.3	22.4	7.5	15.8	47.6	1.2	3.5	35.8	6.2	7.1	86.5	1.3	4.1	5.5	5	1.9	1.9	0.4
15	1	Kemba Walk	202689	17	55	35.1	22.9	7.6	17.9	42.3	2.9	7.6	37.9	4.8	5.7	85.2	0.4	2.9	3.4	5.8	2.2	1.2	0.3
16	1	Paul George	202331	17	56	36.5	22.5	7.6	17	44.8	3.3	7.7	43.2	4	4.9	80.4	0.9	4.5	5.4	3.1	2.7	2.2	0.5
17	0	Blake Griffin	201933	17	41	34.4	22.2	7.6	17.5	43.4	1.9	5.8	33.2	5.1	6.4	79.1	1.4	6.5	8	5.6	2.9	0.8	0.3
18	0	CJ McCollum	203468	17	57	36.1	21.7	8.3	18.4	45	2.5	5.9	42.1	2.7	3.1	85.9	0.6	3.3	3.9	3.2	1.9	1	0.4
19	0	Karl-Anthon	1626157	17	61	35.1	20.2	7.4	13.6	54.6	1.5	3.5	42.1	3.9	4.5	85.8	2.9	9.2	12.1	2.4	1.9	0.8	1.5
20	0	Donovan Mi	1628378	17	55	32.1	19.6	7.3	16.6	43.9	2.3	6.6	35.4	2.7	3.2	83.6	0.6	2.9	3.5	3.5	2.6	1.5	0.4
21	0	Khris Middle	203114	17	57	36.7	20	7.3	15.8	46.3	1.8	5.2	34.6	3.5	4.1	86.2	0.5	4.8	5.3	4.1	2.3	1.3	0.2
22	1	Klay Thomps	202691	17	57	34.1	20	7.8	15.9	49.4	3.2	7.1	45.5	1.1	1.3	86.3	0.4	3.5	3.9	2.5	1.8	0.7	0.6
23	0	T.J. Warren	203933	17	56	32.4	19.3	8	16	50	0.3	1.4	20.3	3	4.1	74.6	1.8	3.3	5.1	1.4	1.3	1	0.6
24	0	Dennis Schr	203471	17	55	31.3	19.5	7.5	17.2	43.6	1.1	3.8	28.7	3.4	3.9	87	0.7	2.4	3.1	6.3	2.6	1.1	0.1
25	0	Jrue Holiday	201950	17	57	36.7	18.6	7.3	15	48.8	1.5	4.8	32.5	2.4	3.1	78.7	0.6	3.7	4.3	5.5	2.5	1.5	0.7
26	0	Harrison Ba	203084	17	56	34.4	18.3	6.8	15.2	44.9	1.4	4	35	3.2	3.9	82.6	1.1	5.5	6.6	1.9	1.5	0.7	0.2
27	0	Chris Paul	101108	17	39	31.9	19.2	6.5	13.9	47	2.7	6.7	39.7	3.5	3.8	90.7	0.6	5	5.7	8.3	2.3	1.8	0.3
28	0	Tobias Harri	202699	17	54	32.9	18	6.7	14.9	45	2.3	5.7	40.2	2.3	2.7	82.4	0.8	4.5	5.3	2	1.1	0.8	0.3
29	0	Nikola Jokic	203999	17	51	31.5	16.9	6.2	12.7	48.5	1.3	3.6	36.3	3.3	3.9	85	2.7	7.9	10.6	5.9	2.7	1.2	0.8
30	0	Eric Gordon	201569	17	51	31.9	18.5	6.1	14.6	41.8	3	9	33.4	3.2	3.9	81.9	0.4	1.9	2.3	2.4	2	0.6	0.4

3.2 初步分析

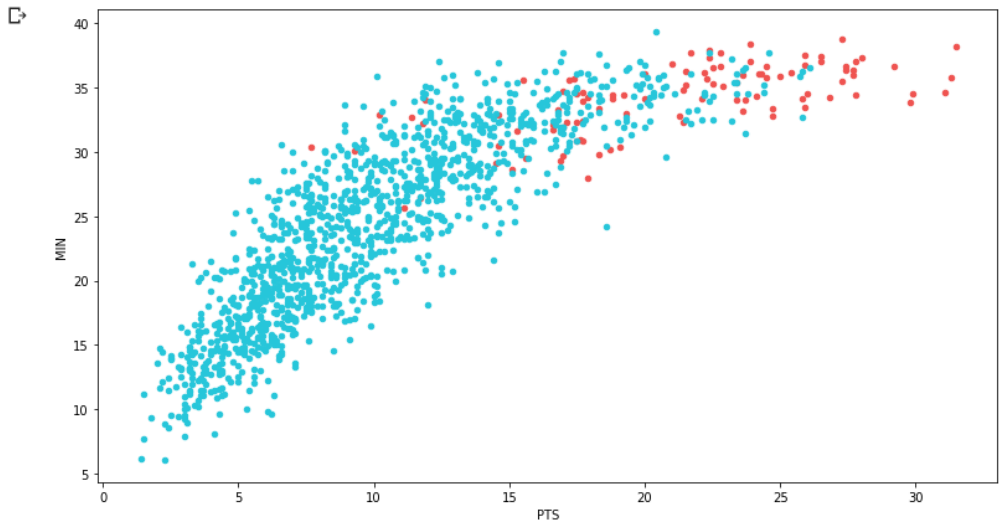
在這 83 項數據中，某些項目的數據較優秀可能會影響觀眾的投票意向，所以將資料視覺化看看大概長怎麼樣。

程式碼：

```
# Run in jupyter notebook

%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
data = pd.read_excel('13-17Player.xlsx')
def scatter_plot(x,y):
    ax = data.query('allstar==1').plot.scatter(x=x,y=y,color='#ef5350')
    data.query('allstar==0').plot.scatter(x=x,y=y,color='#26C6DA',ax=ax,figsize=(13,7))
```

結果：

```
[ ]  scatter_plot('PTS','MIN')
```

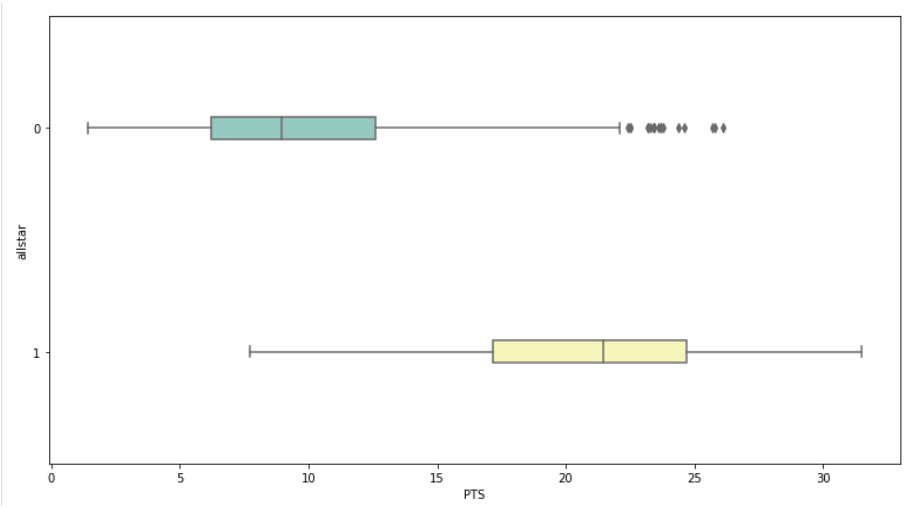


得分箱型圖。

程式碼：

```
fig ,ax = plt.subplots(figsize=(13,7))
def box_plot(x):
    sns.boxplot(data=data.sample(1000),x=x,y='allstar',width=0.1,palette='Set3',ax=ax,orient='h')

box_plot('PTS')
```

結果：



3.3DNN

主要我們是想要利用球員的成績，來預測他是否可以進明星賽，這個簡單的 task 用傳統 Machine Learning 的方法也可以得到不錯的 performance。不過現在 Deep learning 在某些情況下表現更好，以下為訓練的過程。

- Data

  資料餵入：

```python
import pandas as pd
import numpy as np

# Read data from excel
train_data0107 = pd.read_excel(DATA_PATH+"01-07Player.xlsx")
train_data0812 = pd.read_excel(DATA_PATH+"08-12Player.xlsx")
train_data1317 = pd.read_excel(DATA_PATH+"13-17Player.xlsx")
train_df = pd.concat([train_data0107,train_data0812])
test_df = train_data1317

# Numpy array
train_x = train_df.drop(['allstar','name','pid','Overall'],axis=1).values
test_x = test_df.drop(['allstar','name','pid','Overall'],axis=1).values
train_y = train_df['allstar'].values
test_y = test_df['allstar'].values

print(train_x.shape) # (2200, 83)
print(test_x.shape) # (1327, 83)
```

```
(2200, 84)
(1327, 84)
```

由於資料並無缺失也沒特別的情況需要處理，唯一值得提的是由於這屬於不平衡的資料，稍後會再用別的方式來處理這個問題。

```python
from keras import callbacks
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import Adam
from sklearn.utils import shuffle, class_weight
```

- Model structure

  input_dim 是輸入的維度，我們有 83 維的資料。

  接著加上幾層 100 維的 fully connected hidden layer。因為資料和目標都滿簡單的，所以基本上只需要幾層 hidden layer ，在參數量不會很多的時候就可以 fit 我們的目標，使用 relu 作為 activation function。最後為 1 維的 output layer 用 sigmoid 。

  binary_crossentropy 為二元分類的 loss function，以 Adam 作為 optimizer。

```
# Sequential model
model = Sequential()
model.add(Dense(10,input_dim=84,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=Adam(lr=0.0001),
              metrics=['accuracy'])
```

- Fit

  設定一下 callback function 讓 val_loss 最小的 model 被保存起來，作為最終較好的 model ，也可以使用 earlystopping 等方法。

  因為資料不太平均，在 model.fit() 傳入 class_weight 讓 model 在 training 的時候可以根據資料的比例調整 loss function。

```python
epochs = 1000 # 訓練次數
batch_size = 512 # gradient decent 的 batch 大小

# Class weight
class_weight = class_weight.compute_class_weight('balanced',np.unique(train_y),train_y)

# Callback function
callback = callbacks.ModelCheckpoint('NBA_weights.{epoch:02d}-{val_loss:.2f}.hdf5',
                                     monitor='val_loss',
                                     verbose=1,
                                     save_best_only=True,
                                     mode='auto',
                                     period=10)
# Fit
history = model.fit(train_x,
                    train_y,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=0.3,
                    class_weight = class_weight,
                    callbacks=[callback])
```

3.4 預測

訓練好 model 之後，我們使用 13~17 年的球員成績作為測試資料，加上混淆矩陣看一下最終的成果。

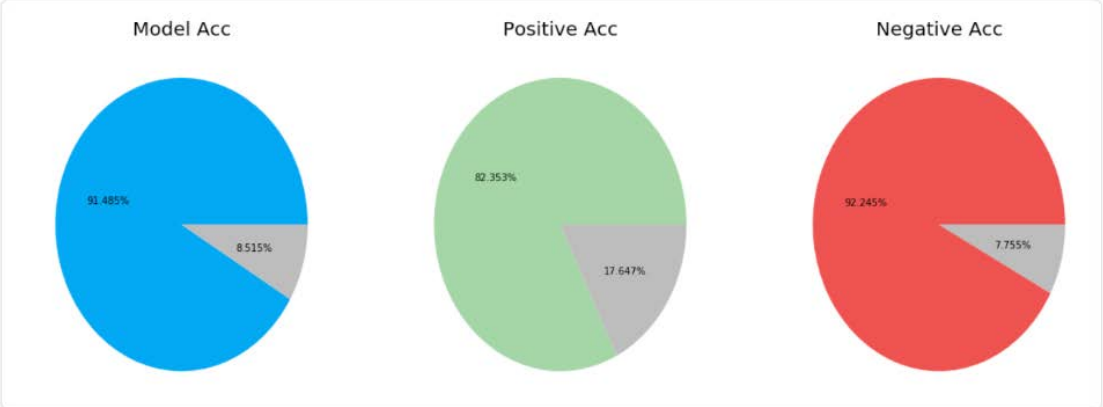
在平均分佈的資料裡，以 0.5 做為 sigmoid 的分界。現在處理的是不平衡的資料，在訓練時我們加入了 class weight ，在預測時也要根據比例調整預測的門檻值。

程式碼：

```python
confution_matrix = {
    'TP':0,
    'TN':0,
    'FP':0,
    'FN':0
}

positive_count = 0
negative_count = 0

# Threshold
# 根據 class weight 調整門檻
TH = class_weight[0]/(class_weight[0]+class_weight[1])

for idx,p in enumerate(model.predict(test_x)):
    real = test_df.iloc[idx]['allstar']  # real data
    p = p>= TH
    positive_count += real
    negative_count += not real
    # confusion matrix
    if real and p:
        confution_matrix['TP']+=1
    elif not real and not p:
        confution_matrix['TN']+=1
    elif real and not p:
        confution_matrix['FP']+=1
    elif not real and p:
        confution_matrix['FN']+=1
```

結果：

CM	Positive	Negetive
True	84	1130
False	18	95



4. 結論與 Future Work

- 結論
  - 投票意向和球員場上成績有關
  - 有些無法從球員成績看出的因素，使得準確度不容易達到非常高
    - （在模型中很難看出這些猜測是否正確，這只是可能的假設)
    - 球員的名氣（老將可能成績較差但仍受觀眾喜愛）
    - 球員私人行為（發表歧視言論等等）
    - 作弊手段得到數據
    - 球員所在的隊伍

我們蒐集的資料是「明星賽前的成績」，如此一來就無法在結果真正揭曉前「預知」結果，雖然這個專案最終沒有得到特別有意義成果，但從資料蒐集整理、建立模型架構、訓練、分析，完整訓練了一個簡單的 DNN。

- Future Work

  未來可以經過一些調整之後得到更有意義的結果。

  - 分析此模型的結果，看看是哪些球員過譽了？
  - 蒐集比較完整的資料
  - 做比較深入的特徵工程
  - 使用其他技巧，像是用 NLP 分析球壇傳聞等等
  - 針對其他目標做訓練，如：預測東西區 24 人名單，總冠軍預測

5. 參考文獻

1. https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db
2. https://ndltd.ncl.edu.tw/cgi-bin/gs32/gsweb.cgi/ccd=J47c8J/record?r1=5&h1=3#XXX
3. https://blog.hanklu.tw/post/nba-allstar-predict/