Compitition Report

Description

1.[分析方法]

->使用 KNeighborsClassifier 做 machine learning

KNeighborsClassifier 所採用的方式是先訂一個中心點，接著蒐集最靠近中心點的 K 個點，然後分析那些點是哪些特質佔大多數，因此預測中心點即有那個特質。

2.[code]

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')

#將 opponent 參數化(train 和 test 都需要)
# d_opp_test = df_test['opponent'].values

#列出所有對手名稱(test)
# opp_test=[]
# for i in range(len(d_opp_test)):
#     if d_opp_test[i] in opp_test:
#         continue
#     else:
#         opp_test.append(d_opp_test[i])

#給每個對手一個 value，從 1 開始 (字典)
```

```
# opp_dic_test={}
# v=1
# for i in opp_test:
#     opp_dic_test[i] = v
#     v += 1

#把對手換成數值
# for i in range(len(d_opp_test)):
#     d_opp_test[i] = opp_dic_test[d_opp_test[i]]
# df_test['opponent'] = d_opp_test

#列出所有對手名稱(train)
# d_opp_train = df_train['opponent'].values
# opp_train=[]
# for i in range(len(d_opp_train)):
#     if d_opp_train[i] in opp_train:
#         continue
#     else:
#         opp_train.append(d_opp_train[i])

#給每個對手一個 value，從 1 開始 (字典)
# opp_dic_train={}
# v=1
# for i in opp_train:
#     opp_dic_train[i] = v
#     v+=1

#把對手換成數值
# for i in range(len(d_opp_train)):
```

```
# d_opp_train[i] = opp_dic_train[d_opp_train[i]]
# df_train['opponent'] = d_opp_train

#簡化時間(把所剩的分鐘數化成秒並和所剩秒數合併)
df_train['seconds_remaining'] = df_train['minutes_remaining']*60 +
df_train['seconds_remaining']
df_test['seconds_remaining'] = df_test['minutes_remaining']*60 +
df_test['seconds_remaining']

#補缺值(將缺值處填入 100)
df_train = df_train.fillna(100)
df_test = df_test.fillna(100)


#[訓練資料]補缺值(將整行資料代換成 shot_distance 的平均)
#c = df_train['shot_made_flag'].values #c 是 shot_made_flag 的整行資料
#for i in range(len(c)):
#    if c[i] == 1: #當 shot_made_flag=1 時 將 1 換成所有 shot_made_flag=1 時的
shot_distance 平均
#       c[i] = df_train[df_train.shot_made_flag==1].shot_distance.mean()
#    if c[i] == 0: #當 shot_made_flag=0 時 將 0 換成所有 shot_made_flag=0 時的
shot_distance 平均
#       c[i] = df_train[df_train.shot_made_flag==0].shot_distance.mean()
#df_train = df_train.fillna(df_train.shot_distance.mean()) #將缺值補入
shot_distance 總平均

#[測試資料] 同上
#d = df_test['shot_made_flag'].values
#for i in range(len(d)):
```

```
#    if d[i] == 1:
#        d[i] =df_test[df_test.shot_made_flag==1].shot_distance.mean()
#   if d[i] == 0:
#        d[i] =df_test[df_test.shot_made_flag==0].shot_distance.mean()
#df_train = df_train.fillna(df_train.shot_distance.mean())

#df_train['shot_zone_area'] = c
#df_test['shot_zone_area'] = d



y_id = df_test['shot_id'].values

#以'shot_made_flag','shot_distance' 作分析
x_train = df_train[['shot_made_flag','shot_distance']].values
y_train = df_train['action_type'].values

x_test = df_test[['shot_made_flag','shot_distance']].values

k = KNeighborsClassifier(n_neighbors=30) #取最鄰近中心的 30 個點
k.fit(x_train,y_train)
y_test = k.predict(x_test)

result = pd.DataFrame(np.column_stack((y_id.tolist(), y_test.tolist())))
result.to_csv('result.csv',index=False) #轉成 csv 檔
```

備註:#的 code 為試過但沒有提高機率的程式檔，有些已遺失，所以只附上部分檔案。

Analysis

★methods we have tried

1.K-Neighbors Regression

2.DummyClassifier

3.KNeighborsClassifier

★parameters and misdata affect the accuracy

1.[parameters]

->一開始放入所有可能影響 action type 的數據 ['lat','lon','minutes_remaining','period','seconds_remaining','shot_distance','shot_made_flag','shot_zone_area']，但發現不同因子會互相影響，因而干擾判斷，所以決定簡化數據，方法如下:

(1)時間:

將所剩的分數秒數合併成所剩下的總秒數。

(2)位置:

◆因為 lat、lon 的尺度太大，易導致複雜度上升而降低預測準確率，因此不採

用此兩行數據，'shot_zone_area'是類別型資料，把位置分成六種:

[Right Side(R)、Left Side(L)、Right Side Center(RC)、Left Side Center(LC)、Center(C)、Back Court(BC)]

◆而 Shot zone area 資料分得比較粗略，而且性質和'loc_x'、'loc_y'重複，只是讓數據變複雜，影響判斷，因此也不採用。

◆但是我們最後意外發現採用'shot_distance'　比採用'loc_x'、'loc_y'準確率還高，可能是因為採用'loc_x'、'loc_y'有左右之分，而 action type 跟在左邊還是右邊投並沒有太大的關連。

->最終採用'shot_distance'

(3)對手:

將其數據參數化

2.[misdata]

(1)補 0

(2)補 1

(3)補平均值

(4)補其他很大的值

結論:經過以上四種試驗方式，發現(1)(2)(3)皆會影響本身資料的結構，因而可能導致誤判，因此我們認為採取第四種方式是最好的，補的值只要很大，讓他離本來 0、1 組成的群集夠遠就好了，如此一來，就把 train 裡的數據分成兩群，一群的'shot_made_flag' 有(0，1)之異；另一群'shot_made_flag'則皆相同，如此一來解決缺值問題，也將那特殊的一群納入訓練資料中。

★problems we met

1.在過程中發現，如果只單純用兩個因素作分析，例如:(位置、是否進球)、(時間、是否進球)會發現，它們都會影響 action type，而且預測出來的準確率不低，代表它們都是重要的影響因子，但如果**同時納入考慮後，因為多了不同性質的因子，就像是多了一個維度，會使預測準確率降低，雖然一直都有考量到此問題，但不知如何改善。**

2.不知道如何調整某些因子的權重。

★the strong and weak points of our methods

1.[strong points]

(1)處理方法較單純

(2)精度高

(3)對異常值不敏感

2.[weak points]

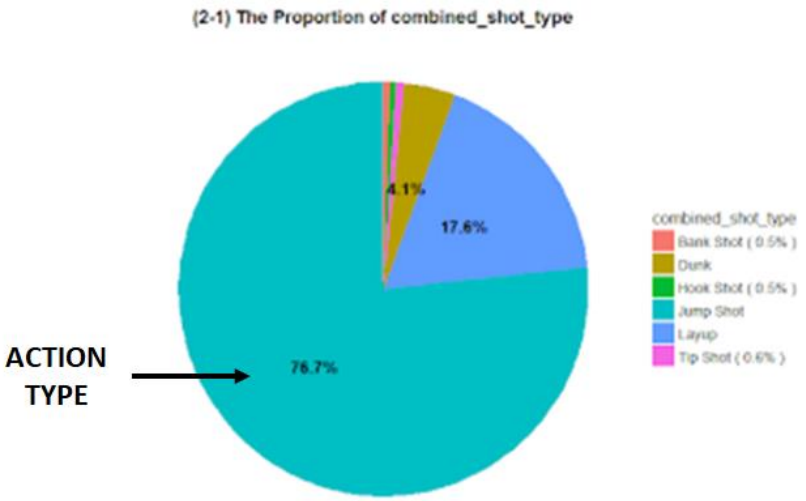
(1)不適合多維度

(2)計算複雜度高

(3)空間複雜度高

Visualization

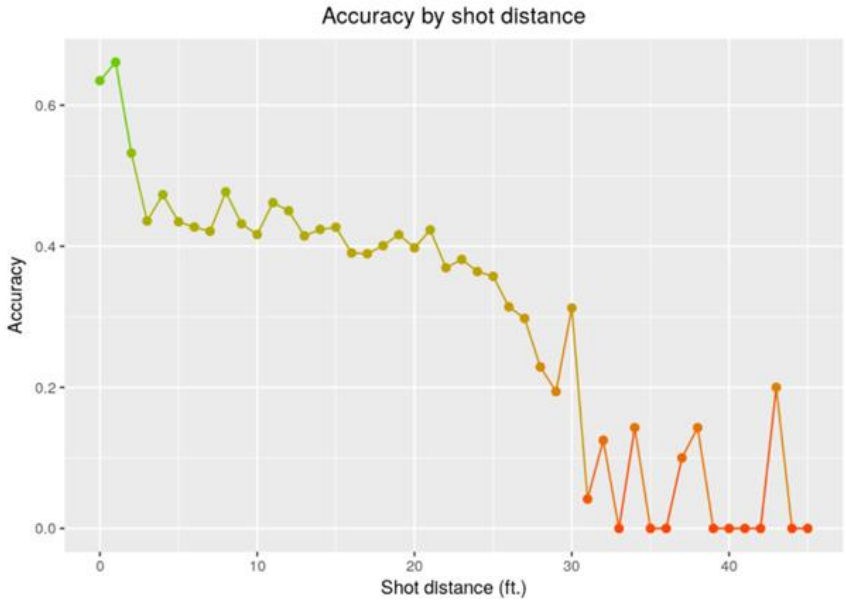
1.[投籃方式比例]



◆由圖可見，投籃方式以 jump shot 為主。

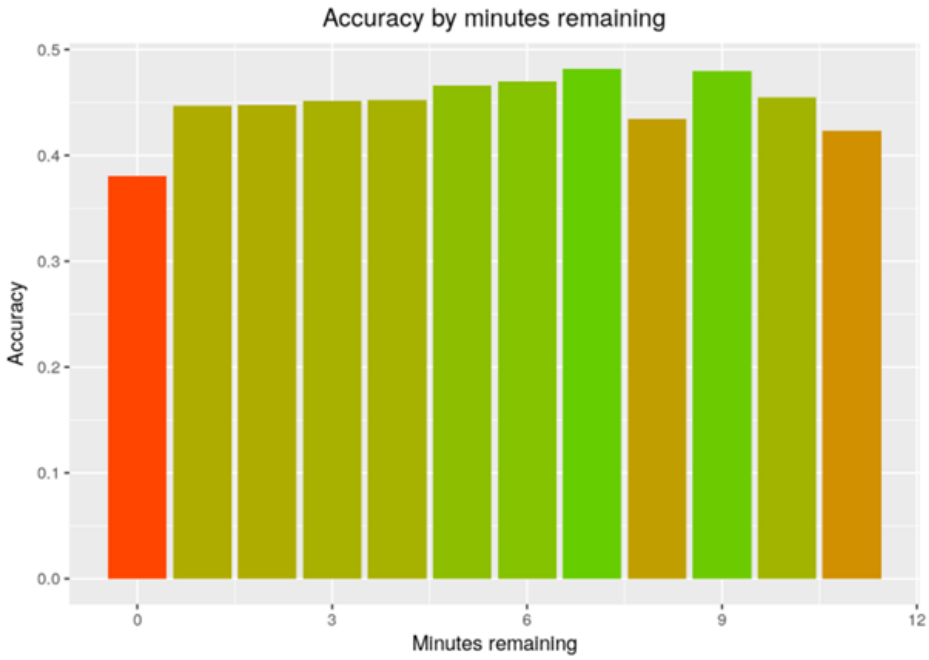
->因此我們試著使用 DummyClassifier 來預測，機率達 0.6225，但此方式非機器學習的好方式，因此捨棄此方法，另尋更好的方法。

2.[距離與準確度之關係]



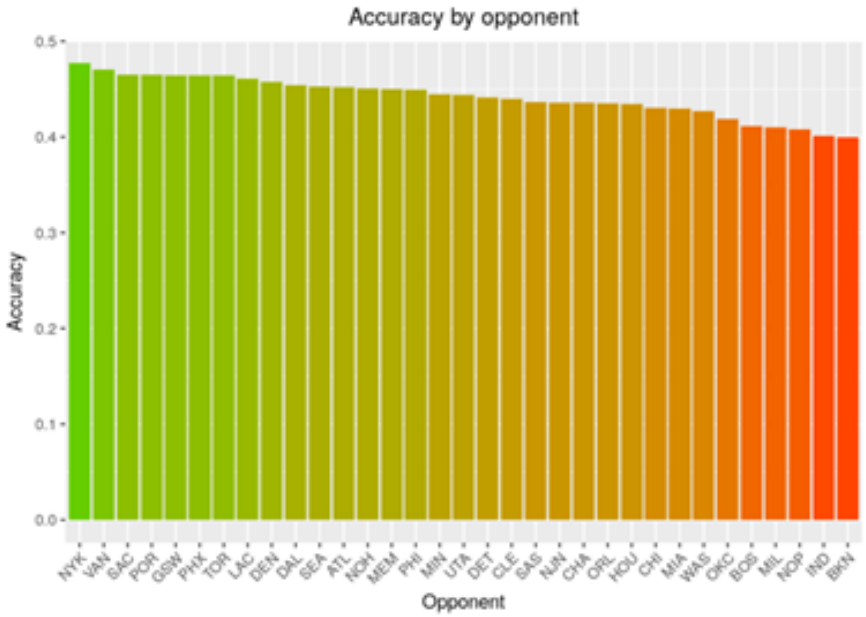
->籃下準確率高達六成，中距離則下降至四成，遠距離則為一至兩成。隨著距離愈遠，準確率越低，下降幅度因不同距離而有所不同，其中，中距離的準確率是只有微幅下降，這很有可能是 kobe Bryant 擅長的投球距離。

3.[剩下的時間與準確度之關係]



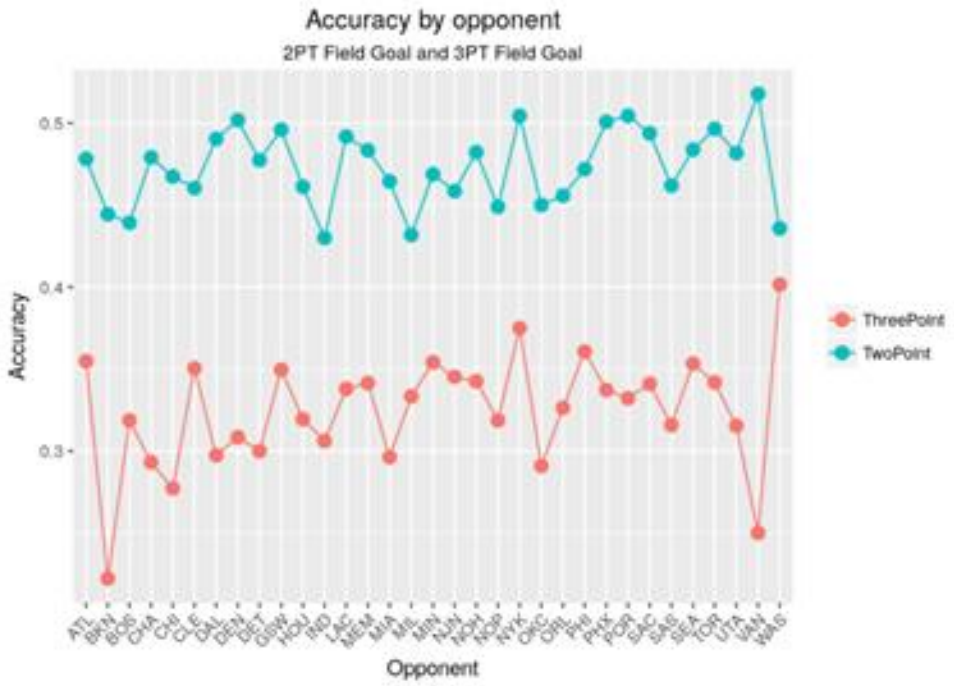
->剩下的時間與準確度大致上沒有太大的相關，除了最後一分鐘的進球率較低以外(不到四成)，有可能是在最後一分鐘球員較累而使進球率降低，又或者兩隊比分差較大，使用拖延戰術且未將球投進，所以使進球率下降。

4.[對手與準確度之關係]



->由圖可知，對手是 NYK 時，進球率是最高的(約 0.48)；而對手是 BKN 進球率是最低的(約 0.41)。這可能跟對手的習慣性有關，若較熟悉對手準確率便很可能能提高。

5.[對手與準確度之關係]



->由圖可知，二分球進球率明顯高於三分球。而三分球與二分球之間的相關性並不高，舉例來說:若內線防守良好，在內線投球得分相對困難，就會選擇多在外線出手，透過三分線的進球率幫助球隊獲勝。當然，球隊的狀況良好的話，很有可能兩分球及三分球的進球率皆高，所以，我認為有相關但相關性不高，當然也可以試試看透過寫程式去印證。

References

1.http://gymining.blogspot.tw/2016/05/kobe-bryant-shot-selection.html

2.https://www.kaggle.com/xvivancos/kobe-bryant-shot-selection?scriptVersionId=3202343

3.http://gymining.blogspot.tw/2016/05/kobe-bryant-shot-selection.html