NBA player salary predication and rationality analysis

CS579 Project report
Hang Li/Lu Wang

Introduction

The main purpose for this project is to predict the current NBA players' salary, and by the predication, draw some conclusion of the rationality of the players' current salary.

In the current NBA league, some players are improving themselves with a significant speed. However, they still holding a contract with very limited salary. On the other hand, some players are suffering from injury or other problems, make them useless for their team, but at the same time, they still holding a nice contract. There will be numerous reasons for players get a salary that is cannot related to their performance. This paper will try to get the result whether or not the player's salary is more/less than he should get according to his performance.

The hypothesis is that players' salary are related to their popularity, and people's attitude toward them.

Data

In this project, collector module is designed for collecting the data.

Environment and Requirements

The program requires python 2.7 and mongoDB. Use the following command to install the library (example in Linux):

```
$cd iit-cs579-project
$sudo pip install –r requirements.txt
```

Install and run mongoDB:

```
$wget     “https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-2.6.5.tgz”
$tar –zxf “mongodb-linux-x86_64-2.6.5.tgz”
$cd mongodb-linux-x86_64-2.6.5/bin
$mkdir -p ~/mongodb/cs579proj
$mongod --dbpath ~/mongodb/cs579proj --bind_ip 10.1.1.1 --port 2050
```

Copy config.sample.py and rename the new one to config.py, then fill in the required field with the prepared Twitter APP keys and secrets:

```
class SampleConfig(Config):
    CONSUMER_KEY = ''
    CONSUMER_SECRET = ''
    ACCESS_TOKEN = ''
    ACCESS_TOKEN_SECRET = ''

    MONGODB_IP = '10.1.1.1'
    MONGODB_PORT = 2050
```

Start the conllector to collect programs:

```
$python –m collector “Your Keyword” --config SampleConfig
```

The program will load the SampleConfig class as its config file.

Module Structure

```
collector
    /__init__.py
    /__main__.py
    /models.py
    /timing.py
    /twitter_wrapper.py
    /config.sample.py
    /config.py  (should be created by user)
```

models.py defines database models. There are 2 models in total: Tweet and Keyword. Each represents one collection in database.

Documents in ‘tweets’ collection will appears as the following structure:

```
{
    ‘keyword’: str,
    ‘tweet’: dict
}
```

The keyword is the keyword given in the program argument. The tweet field is the tweet field in origin response payload.

Documents in ‘keywords’ collection will appears as the following structure:

```
{
    ‘team’: str,
    ‘player’: str,
    ‘salary’: float,
    ‘city’: str
}
```

timing.py provides a class that can maintain a time window, for twitter API there is a

time window for every user. The API used here, “/search/tweets”, limits 450 requests per 15 minutes. When creating the Timing object, the parameter time_window can define the time window (15 * 60 for example) in seconds.

__main__.py is the entrance of the program when user starts the collector program. To search history tweets, a max_id should specified in the search parameters. When program starts, it will try to find the minimum id in the database and set it to the max_id for searching. From the second search loop, the max_id will be parsed from the payload in previous response. If ‘next_results’ is not in the ‘search_metadata’ in the response, the program will terminate.

Method

Raw dictionary information collection

Raw dictionary include keys which is players’ names, and value as another dictionary. This dictionary will store the count of positive, neutral and negative tweets from person and from group.

Go through each collected tweets in the database, get the label information as player name. Get the text message and do word analysis on it, return a positive/neutral/negative attitude from the tweet. Get the name information of the tweet sender, do name analysis on it, return whether or not this tweet is send from a person or a group/media.

After finish these analysis, choose the corresponding feature, and add one to it for this player.

After went over all the tweets, the feature count for each player are saved on the raw dictionary. Save the dictionary to the database as a matrix format, for each line of the matrix, we have player's name, positive tweets count from person, neutral tweets count from person, negative tweets count from person, positive tweets count from group, neutral tweets count from group, negative tweets count from group.

If a player have zero count on any of these features, we mark this player as 'suck player', and don't save any of the player's information into the database.

Find similar player

Each player has six features. Compare the same feature with different player will reveal the similarity between those two players. Define a method that take a player's name, and return all other players similarity value when compared with this player. A similarity value between two players are the summation of similarity value of each features for those two players. The similarity value of a feature are calculated by rank the difference between the feature counts between two players. If there are n players have smaller difference of feature counts than player 1 and player 2, based on player 1, the similarity value for this feature of player 2 is n.

The player with least similarity value will be the most similar player based on these 6 features.

Find outliners

Outliners are players that don't have strongly related tweets information and salary information. To find out the outliners, 6 player with highest salary and 6 player with lowest salary were selected. Collect the similarity data for each player, draw a graph using the similarity value (most similar to least similar). Get the trend line based on the points in the graph, compute the distance from each point to the trend line, those top 1/6 points with longest distances were be marked as outliners.

After get all the outliners information from these 12 players, rank the times that a certain players' name were marked as outliners. There are two ways to deal with these outliners, one is select top several players as outliners, the other is make all marked players as outliners. The experiment chose the points that appears more than 3 times.

Calculate the predicted salary

Select one player, sort the raw matrix based on other players' similarity value to this player. Note that the outliners will not be considered for this step. Then select 5 most similar player, calculate the average salary of these players, and return the average value as the predicted salary.

Check the average salary accuracy

Calculate the predicted salary for all the players that is neither in suckplayer() nor in outliners. Return the average of the accuracy of each player.

Experiments

First, the experiments begins with creating the raw matrix, using getMatrix(name), to get a matrix that will be used later. Take getMatrix(‘Kobe Bryant’) as an emample:

```
In [55]: pprint(getMatrix('Kobe Bryant'))
```

```
[['Kobe Bryant', 0, 29382, 10730, 15867, 79012, 41911, 20184, 0],
 ['David West', 12.0, 533, 195, 408, 1367, 582, 280],
 ['Tony Parker', 12.5, 1187, 222, 535, 3023, 1222, 357],
 ['Joakim Noah', 12.7, 743, 703, 423, 1738, 834, 922],
 ['Rajon Rondo', 12.9, 1441, 401, 523, 2853, 1340, 1114],
 ['Brook Lopez', 15.7, 275, 268, 169, 1281, 674, 347],
 ['LaMarcus Aldridge', 15.2, 503, 90, 266, 1389, 719, 178],
 ['Zach Randolph', 16.5, 607, 69, 560, 626, 355, 75],
 ['Kyle Lowry', 12.0, 1056, 364, 622, 2371, 1360, 476],
 ['Nikola Pekovic', 12.1, 13, 44, 11, 230, 70, 182],
 ['Gordon Hayward', 14.7, 313, 132, 254, 541, 418, 222],
 ['Chandler Parsons', 14.7, 972, 412, 702, 1989, 1576, 494],
 ['Kevin Durant', 20.0, 6746, 7851, 3890, 68849, 53836, 12034],
 ['Chris Paul', 20.1, 7025, 3130, 3791, 25013, 14937, 6137],
 ['Andre Iguodala', 12.3, 77, 13, 40, 246, 185, 29],
 ['Kevin Love', 15.7, 21341, 984, 13828, 2156, 799, 1443],
 ['Brandon Roy', 14.4, 184, 97, 101, 321, 191, 156],
 ['Roy Hibbert', 14.9, 297, 525, 256, 2110, 1093, 927],
 ['JaVale McGee', 11.3, 2214, 180, 1921, 1867, 1181, 496],
 ['Blake Griffin', 17.6, 4812, 1178, 3346, 9931, 4789, 2408],
 ['Tyreke Evans', 11.2, 210, 95, 110, 415, 229, 122],
 ['Larry Sanders', 11.0, 147, 97, 123, 554, 263, 169],
 ['Kevin Garnett', 12.0, 867, 312, 269, 2037, 756, 495],
 ['AI Jefferson', 13.5, 9, 29, 6, 200, 158, 53],
 ['Nicolas Batum', 11.4, 51, 37, 31, 369, 93, 78],
 ['Andrea Bargnani', 12.0, 239, 129, 62, 656, 181, 353],
 ['Danilo Gallinari', 10.9, 78, 650, 57, 1810, 1244, 1058],
 ['Dwyane Wade', 15.0, 1200, 545, 521, 2616, 1045, 1158],
 ['Serge Ibaka', 12.3, 360, 186, 129, 1084, 537, 240],
 ['Andrew Bogut', 13.0, 1694, 219, 1599, 834, 409, 335],
 ['Derrick Favors', 13.0, 280, 5, 153, 22, 17, 25],
 ['Tyson Chandler', 14.6, 447, 503, 311, 1032, 598, 713],
 ['Eric Gordon', 14.9, 354, 1145, 305, 1419, 1043, 1825],
 ['Jrue Holiday', 11.0, 75, 66, 73, 278, 173, 124],
 ['Marc Gasol', 15.8, 3182, 748, 1942, 7785, 4684, 1336],
 ['Gerald Wallace', 10.1, 104, 28, 78, 153, 134, 29],
 ['Jeremy Lin', 14.9, 1860, 1043, 1093, 5094, 3059, 2210],
 ['Rudy Gay', 19.3, 1067, 513, 650, 2116, 1035, 832],
 ['Eric Bledsoe', 13.0, 5111, 1024, 5942, 6032, 4393, 1435],
 ['Derrick Rose', 18.9, 21698, 6322, 16849, 28781, 18191, 9679],
 ['Josh Smith', 14.0, 860, 506, 901, 2232, 1410, 584],
 ['Russell Westbrook', 15.7, 1594, 465, 874, 7390, 3825, 1377],
 ['Paul George', 15.8, 4279, 1712, 2673, 10412, 6499, 3003],
 ['DeAndre Jordan', 11.4, 2023, 445, 1160, 2566, 1352, 719],
 ['Lebron James', 20.6, 21772, 10935, 12514, 37523, 19021, 17127],
 ['DeMarcus Cousins', 13.7, 3906, 1185, 2594, 6327, 3335, 1991],
 ['Marcin Gortat', 10.4, 116, 73, 120, 269, 144, 74],
 ['David Lee', 15.0, 804, 334, 773, 1931, 961, 495],
 ['Omer Asik', 14.9, 108, 161, 72, 319, 138, 477]]
```

The first line is the information that will used later for compassion, the rest of the matrix represent the features for all players.

After get the raw matrix, it was converted to a similarity matrix using simMatrix(name), following is an example for simMatrix('Kobe Bryant'):

```
In [56]: pprint(simMatrix('Kobe Bryant'))
[['Kobe Bryant', 0, 29382, 10730, 15867, 79012, 41911, 20184, 0],
 ['David West', 12.0, 533, 195, 408, 1367, 582, 280, 170],
 ['Tony Parker', 12.5, 1187, 222, 535, 3023, 1222, 357, 121],
 ['Joakim Noah', 12.7, 743, 703, 423, 1738, 834, 922, 126],
 ['Rajon Rondo', 12.9, 1441, 401, 523, 2853, 1340, 1114, 100],
 ['Brook Lopez', 15.7, 275, 268, 169, 1281, 674, 347, 179],
 ['LaMarcus Aldridge', 15.2, 503, 90, 266, 1389, 719, 178, 184],
 ['Zach Randolph', 16.5, 607, 69, 560, 626, 355, 75, 195],
 ['Kyle Lowry', 12.0, 1056, 364, 622, 2371, 1360, 476, 116],
 ['Nikola Pekovic', 12.1, 13, 44, 11, 230, 70, 182, 259],
 ['Gordon Hayward', 14.7, 313, 132, 254, 541, 418, 222, 197],
 ['Chandler Parsons', 14.7, 972, 412, 702, 1989, 1576, 494, 115],
 ['Kevin Durant', 20.0, 6746, 7851, 3890, 68849, 53836, 12034, 10],
 ['Chris Paul', 20.1, 7025, 3130, 3791, 25013, 14937, 6137, 20],
 ['Andre Iguodala', 12.3, 77, 13, 40, 246, 185, 29, 259],
 ['Kevin Love', 15.7, 21341, 984, 13828, 2156, 799, 1443, 64],
 ['Brandon Roy', 14.4, 184, 97, 101, 321, 191, 156, 224],
 ['Roy Hibbert', 14.9, 297, 525, 256, 2110, 1093, 927, 131],
 ['JaVale McGee', 11.3, 2214, 180, 1921, 1867, 1181, 496, 114],
 ['Blake Griffin', 17.6, 4812, 1178, 3346, 9931, 4789, 2408, 33],
 ['Tyreke Evans', 11.2, 210, 95, 110, 415, 229, 122, 223],
 ['Larry Sanders', 11.0, 147, 97, 123, 554, 263, 169, 215],
 ['Kevin Garnett', 12.0, 867, 312, 269, 2037, 756, 495, 143],
 ['AI Jefferson', 13.5, 9, 29, 6, 200, 158, 53, 268],
 ['Nicolas Batum', 11.4, 51, 37, 31, 369, 93, 78, 257],
 ['Andrea Bargnani', 12.0, 239, 129, 62, 656, 181, 353, 212],
 ['Danilo Gallinari', 10.9, 78, 650, 57, 1810, 1244, 1058, 153],
 ['Dwyane Wade', 15.0, 1200, 545, 521, 2616, 1045, 1158, 99],
 ['Serge Ibaka', 12.3, 360, 186, 129, 1084, 537, 240, 187],
 ['Andrew Bogut', 13.0, 1694, 219, 1599, 834, 409, 335, 148],
 ['Derrick Favors', 13.0, 280, 5, 153, 22, 17, 25, 254],
 ['Tyson Chandler', 14.6, 447, 503, 311, 1032, 598, 713, 152],
 ['Eric Gordon', 14.9, 354, 1145, 305, 1419, 1043, 1825, 119],
 ['Jrue Holiday', 11.0, 75, 66, 73, 278, 173, 124, 245],
 ['Marc Gasol', 15.8, 3182, 748, 1942, 7785, 4684, 1336, 53],
 ['Gerald Wallace', 10.1, 104, 28, 78, 153, 134, 29, 260],
 ['Jeremy Lin', 14.9, 1860, 1043, 1093, 5094, 3059, 2210, 59],
 ['Rudy Gay', 19.3, 1067, 513, 650, 2116, 1035, 832, 110],
 ['Eric Bledsoe', 13.0, 5111, 1024, 5942, 6032, 4393, 1435, 43],
 ['Derrick Rose', 18.9, 21698, 6322, 16849, 28781, 18191, 9679, 9],
 ['Josh Smith', 14.0, 860, 506, 901, 2232, 1410, 584, 102],
 ['Russell Westbrook', 15.7, 1594, 465, 874, 7390, 3825, 1377, 74],
 ['Paul George', 15.8, 4279, 1712, 2673, 10412, 6499, 3003, 30],
 ['DeAndre Jordan', 11.4, 2023, 445, 1160, 2566, 1352, 719, 90],
 ['Lebron James', 20.6, 21772, 10935, 12514, 37523, 19021, 17127, 4],
 ['DeMarcus Cousins', 13.7, 3906, 1185, 2594, 6327, 3335, 1991, 45],
 ['Marcin Gortat', 10.4, 116, 73, 120, 269, 144, 74, 241],
 ['David Lee', 15.0, 804, 334, 773, 1931, 961, 495, 131],
 ['Omer Asik', 14.9, 108, 161, 72, 319, 138, 477, 222]]
```

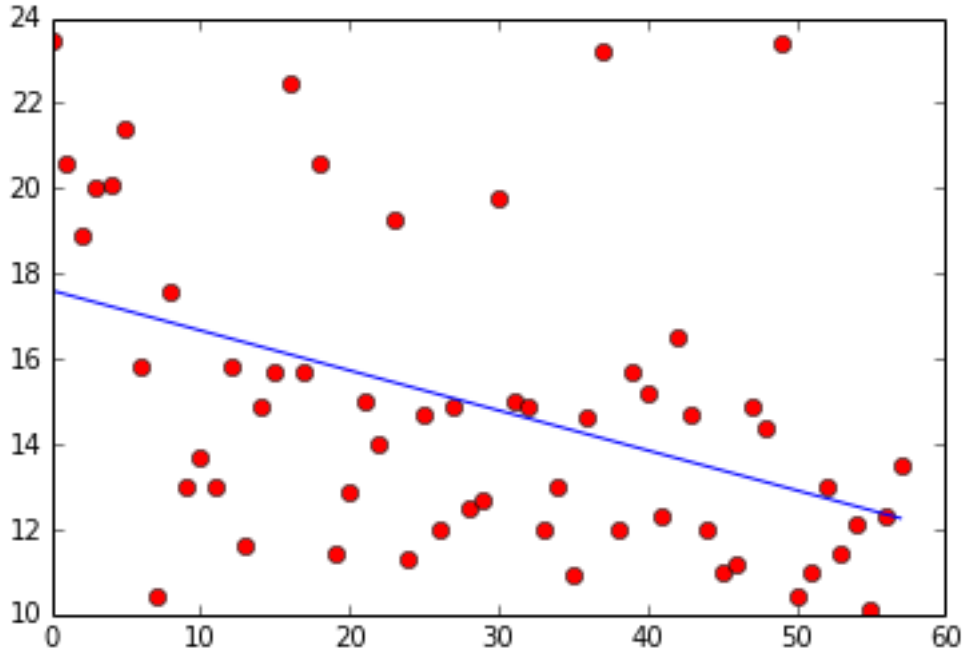
The last number of each line is the similarity value when compared to player Kobe Bryant, the smaller value means more similar.

Change the order of simMatrix by similarity value can get the prediction matrix, by using the predict(name) method, an example is predict('Kobe Bryant'):

In [57]:
```
predict_on_Kobe = predict('Kobe Bryant')
pprint(predict_on_Kobe)
```

```
[['Kobe Bryant', 0, 29382, 10730, 15867, 79012, 41911, 20184, 0],
 ['Lebron James', 20.6, 21772, 10935, 12514, 37523, 19021, 17127, 4],
 ['Derrick Rose', 18.9, 21698, 6322, 16849, 28781, 18191, 9679, 9],
 ['Kevin Durant', 20.0, 6746, 7851, 3890, 68849, 53836, 12034, 10],
 ['Chris Paul', 20.1, 7025, 3130, 3791, 25013, 14937, 6137, 20],
 ['Paul George', 15.8, 4279, 1712, 2673, 10412, 6499, 3003, 30],
 ['Blake Griffin', 17.6, 4812, 1178, 3346, 9931, 4789, 2408, 33],
 ['Eric Bledsoe', 13.0, 5111, 1024, 5942, 6032, 4393, 1435, 43],
 ['DeMarcus Cousins', 13.7, 3906, 1185, 2594, 6327, 3335, 1991, 45],
 ['Marc Gasol', 15.8, 3182, 748, 1942, 7785, 4684, 1336, 53],
 ['Jeremy Lin', 14.9, 1860, 1043, 1093, 5094, 3059, 2210, 59],
 ['Kevin Love', 15.7, 21341, 984, 13828, 2156, 799, 1443, 64],
 ['Russell Westbrook', 15.7, 1594, 465, 874, 7390, 3825, 1377, 74],
 ['DeAndre Jordan', 11.4, 2023, 445, 1160, 2566, 1352, 719, 90],
 ['Dwyane Wade', 15.0, 1200, 545, 521, 2616, 1045, 1158, 99],
 ['Rajon Rondo', 12.9, 1441, 401, 523, 2853, 1340, 1114, 100],
 ['Josh Smith', 14.0, 860, 506, 901, 2232, 1410, 584, 102],
 ['Rudy Gay', 19.3, 1067, 513, 650, 2116, 1035, 832, 110],
 ['JaVale McGee', 11.3, 2214, 180, 1921, 1867, 1181, 496, 114],
 ['Chandler Parsons', 14.7, 972, 412, 702, 1989, 1576, 494, 115],
 ['Kyle Lowry', 12.0, 1056, 364, 622, 2371, 1360, 476, 116],
 ['Eric Gordon', 14.9, 354, 1145, 305, 1419, 1043, 1825, 119],
 ['Tony Parker', 12.5, 1187, 222, 535, 3023, 1222, 357, 121],
 ['Joakim Noah', 12.7, 743, 703, 423, 1738, 834, 922, 126],
 ['Roy Hibbert', 14.9, 297, 525, 256, 2110, 1093, 927, 131],
 ['David Lee', 15.0, 804, 334, 773, 1931, 961, 495, 131],
 ['Kevin Garnett', 12.0, 867, 312, 269, 2037, 756, 495, 143],
 ['Andrew Bogut', 13.0, 1694, 219, 1599, 834, 409, 335, 148],
 ['Tyson Chandler', 14.6, 447, 503, 311, 1032, 598, 713, 152],
 ['Danilo Gallinari', 10.9, 78, 650, 57, 1810, 1244, 1058, 153],
 ['David West', 12.0, 533, 195, 408, 1367, 582, 280, 170],
 ['Brook Lopez', 15.7, 275, 268, 169, 1281, 674, 347, 179],
 ['LaMarcus Aldridge', 15.2, 503, 90, 266, 1389, 719, 178, 184],
 ['Serge Ibaka', 12.3, 360, 186, 129, 1084, 537, 240, 187],
 ['Zach Randolph', 16.5, 607, 69, 560, 626, 355, 75, 195],
 ['Gordon Hayward', 14.7, 313, 132, 254, 541, 418, 222, 197],
 ['Andrea Bargnani', 12.0, 239, 129, 62, 656, 181, 353, 212],
 ['Larry Sanders', 11.0, 147, 97, 123, 554, 263, 169, 215],
 ['Omer Asik', 14.9, 108, 161, 72, 319, 138, 477, 222],
 ['Tyreke Evans', 11.2, 210, 95, 110, 415, 229, 122, 223],
 ['Brandon Roy', 14.4, 184, 97, 101, 321, 191, 156, 224],
 ['Marcin Gortat', 10.4, 116, 73, 120, 269, 144, 74, 241],
 ['Jrue Holiday', 11.0, 75, 66, 73, 278, 173, 124, 245],
 ['Derrick Favors', 13.0, 280, 5, 153, 22, 17, 25, 254],
 ['Nicolas Batum', 11.4, 51, 37, 31, 369, 93, 78, 257],
 ['Nikola Pekovic', 12.1, 13, 44, 11, 230, 70, 182, 259],
 ['Andre Iguodala', 12.3, 77, 13, 40, 246, 185, 29, 259],
 ['Gerald Wallace', 10.1, 104, 28, 78, 153, 134, 29, 260],
 ['AI Jefferson', 13.5, 9, 29, 6, 200, 158, 53, 268]]
```

Expect for the first line, the higher the player is, the more similar it will be when comparing to player Kobe Bryant. The prediction of salary is based on this matrix. Before the prediction, the outliners should be excluded. By creating a graph of the distribution of players with their salary information, the difference between actual salary and predicted salary can be revealed:



In this graph, the red vertices is represent each player, x-axis is rank of players' similarity value, y-axis is salary. The trend line is a monotone decreasing line, because the less similarity it has, the less salary it would be (compared to the highest paid player). The outliners are those players have a greater distances to the trend line (from graph). This experiment will choose the top 1/6 players as the outliners.

Select from the CSV file, from all the players that was searched, choose 6 of them as well paid players, and 6 of them as poor paid players. Using getNameList(n). This experiment use getNameList(6):

```
In [81]: pprint(getNameList(6))
```

```
['Gerald Wallace',
 'Tim Duncan',
 'Marcin Gortat',
 'Danilo Gallinari',
 'Larry Sanders',
 'Jrue Holiday',
 'Kobe Bryant',
 'Amare Stoudemire',
 'Joe Johnson',
 'Carmelo Anthony',
 'Dwight Howard',
 'Lebron James']
```

Predict those names and get the count that outliners are returned. Using sortOuters() method, loop through all of the player above, return the names that appeared more than once:

```
In [90]: pprint(sortOuters())
```

```
[('Derrick Rose', 12),
 ('Rudy Gay', 12),
 ('Danilo Gallinari', 11),
 ('Zach Randolph', 11),
 ('JaVale McGee', 10),
 ('Eric Bledsoe', 6),
 ('Kyle Lowry', 6),
 ('Marcin Gortat', 5),
 ('Blake Griffin', 5),
 ('Jrue Holiday', 2),
 ('Larry Sanders', 1),
 ('Brandon Roy', 1),
 ('Tyreke Evans', 1),
 ('Omer Asik', 1)]

['Derrick Rose',
 'Rudy Gay',
 'Danilo Gallinari',
 'Zach Randolph',
 'JaVale McGee',
 'Eric Bledsoe',
 'Kyle Lowry',
 'Marcin Gortat',
 'Blake Griffin',
 'Jrue Holiday']
```

The method will only return the lower part, which is the players' name. The upper part shows how many times that this player was selected as an outliner.

After remove the outliners, using the predictAvg() method, will get the player's predicted salary, current salary and the accuracy:

```
In [79]: predicted_avg = predictAvg()
```

```
player predicted actual accuracy
Rajon Rondo 14.34 12.90 0.89
Brook Lopez 13.34 15.70 0.85
Kevin Garnett 13.98 12.00 0.83
AI Jefferson 12.06 13.50 0.89
Derrick Rose 15.80 18.90 0.84
Joakim Noah 14.70 12.70 0.84
Kevin Love 14.80 15.70 0.94
Chandler Parsons 13.54 14.70 0.92
Tyson Chandler 13.46 14.60 0.92
JaVale McGee 13.30 11.30 0.82
Danilo Gallinari 14.56 10.90 0.66
Josh Smith 15.00 14.00 0.93
David Lee 14.60 15.00 0.97
Andrew Bogut 13.70 13.00 0.95
Andre Iguodala 11.72 12.30 0.95
Paul George 15.56 15.80 0.98
Roy Hibbert 14.48 14.90 0.97
David West 13.98 12.00 0.83
Blake Griffin 15.56 17.60 0.88
Jeremy Lin 14.50 14.90 0.97
Zach Randolph 13.72 16.50 0.83
Marc Gasol 15.02 15.80 0.95
Dwyane Wade 14.48 15.00 0.97
Larry Sanders 11.60 11.00 0.95
Nikola Pekovic 12.06 12.10 1.00
Omer Asik 12.54 14.90 0.84
Eric Gordon 13.60 14.90 0.91
Tyreke Evans 11.60 11.20 0.96
Jrue Holiday 11.90 11.00 0.92
Andrea Bargnani 12.70 12.00 0.94
Russell Westbrook 15.38 15.70 0.98
Serge Ibaka 13.14 12.30 0.93
Eric Bledsoe 14.62 13.00 0.88
LaMarcus Aldridge 13.98 15.20 0.92
Brandon Roy 11.60 14.40 0.81
Nicolas Batum 12.06 11.40 0.94
Rudy Gay 15.64 19.30 0.81
DeMarcus Cousins 15.54 13.70 0.87
Tony Parker 12.70 12.50 0.98
Kyle Lowry 13.64 12.00 0.86
Gordon Hayward 12.72 14.70 0.87
Derrick Favors 11.58 13.00 0.89
Marcin Gortat 11.86 10.40 0.86
```

```
In [80]: print predicted_avg
```

```
('average accuraccy is', 0.9006716726199744)
```

The average 90% for the player appeared in the table above.

Discussion of pattern performance

If only using the average salary as the predicted value, the average accuracy is around 80%. Following the experiment process, the result is 90%, which is much higher. Thus, this pattern is able to predict that a certain player should have.

There are some values that are set to some constant in the experiment process, change those value will cause the change of final accuracy and the pattern reliability:

1. The number of high/low paid players for the outliner test

   If there are only very few number of people are chosen to test the outliners, will cause one problems:

   If the chosen player himself is an outliner, then all the result that this player returned is not accurate.

   If the player that have an average salary, then the trend line would more likely to be a parallel line of x-axis, and the outliners that it returned will be all the highest and lowest paid players, which does not make sense.

   So in this experiment, only 6 player was chosen from the top and from the bottom.

2. The player that sortOuters() returned:

   It is possible that some of the top/bot player in the list are outliners, the conclusion based on those outliners are not reliable. So when collecting the outliners information, exclude those only appeared only once, it’s very possible that this was caused by some outliners.

3. The number of outliners that returned for every test:

   If there are only a few names were returned, there would be some outliners that were not found. If there are too many names were returned, some of the normal will be marked as outliners. Although if increase the number of outliners that returned for every test, the predictAvg() will increased a lot, the predicted player will trend to have a salary around the average, and the prediction will also around that value, making this algorithm meaningless. Choose to return 1/6 of the player as outliners is like supposing that 1 out of 6 players are paid too high or too low, which is reasonable.

The prediction of the salary outliners will have a significant value when compared to the prediction of normal players. The outliners will either paid too high or too low. The only exception is that the outliners share the same name with another popular person.

Conclusions and future work:

Conclusion

1. The salary for a certain player can be predicted by knowing how popular he is in twitter.
2. More popular a player is, the more salary that he will probably get.

Future work

1. The pattern could be improved by collecting the exact tweets that was looking

for.

2. The pattern could be improved by collecting more players' information.
3. The pattern could be used as a prediction of future salary for those will finishing their contract soon.