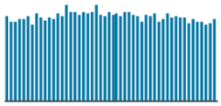
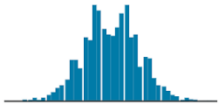
NBA Shot Log Analysis: A MapReduce program for Fear Score and Comfortable Zone Identification

Task Description:

By analyzing the NBA shot log data (Data), we need to answer the following:

GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN
 21.4m 21.4m	1808 unique values	A 50% H 50%	W 50% L 50%	 -53 53
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24

Task 1: For each pair of the players (A, B), we define the fear score of A when facing B is the hit rate, such that B is closest defender when A is shooting. Based on the fear score, for each player, please find out who is his ”most unwanted defender”.

Task 2: For each player, we define the comfortable zone of shooting is a matrix of,

[SHOT DIST, CLOSE DEF DIST, SHOT CLOCK]

Please develop a MapReduce-based algorithm to classify each player’s records into 4 Comfortable zones. Considering the hit rate, which zone is the best for James Harden, Chris Paul, Stephen Curry and Lebron James.

Project Deliverables:

Goal 1: Finding the “most unwanted defender” for each player based on the fear score

Solving this problem with MapReduce, the first procedure is to collect the player information from the dataset. In the mapper, the player, player’s defender, and the outcome of the shot. The following information is outputted for the reducer to parse through.

```
for line in sys.stdin:
    line = line.strip()
    row = csv.reader([line], delimiter = ',')
    row = list(row)[0]

    player = row[19]
    defender = row[14]
    outcome = row[13]

    print(player + '\t' + defender + '\t' + outcome + '\t' + '1')
```

The reducer in this solution stores the information from the mapper in a dictionary where each player is handled separately. The players can be held separately because the dataset is sorted by the players. If this was not the case, the amount of memory used would be much greater, so there would have to be more rounds of MapReduce. As the data points are read by the reducer it checks whether the defender has been accounted for prior. If the defender has been appended to the dictionary already for this specific player, the total shots and missed shots is incremented.

It is necessary to have both the total shots and total missed shots because the fear score will be calculated by the ratio. The more missed shots from a player-defender pair, the higher the fearscore. After the first iteration of the solution, there were lots of 1.0 fear scores, meaning that 100% of shots from a defender were missed. This is due to a low number of samples in the dataset, so a minimum number of shots was included to normalize the results to better reflect a fear score. If a total number of shots did not reach the minimum score, it was divided by four.

```
jon ingles Jackson, Reggie 0.6666666666666666
jon leuer Adams, Steven 0.25
jonas jerebko Aldemir, Furkan 0.25
jonas valanciunas Adams, Steven 0.25
jordan farmar Vucevic, Nikola 0.7058823529411765
jordan hill Aldrich, Cole 0.25
jose calderon Bonner, Matt 0.6666666666666666
jose juan barea Aminu, Al-Farouq 0.25
jrue holiday Allen, Tony 0.25
jusuf nurkic Curry, Stephen 0.625
kawhi leonard Ajinca, Alexis 0.25
kelly olynyk Batum, Nicolas 0.7222222222222222
kemba walker Adrien, Jeff 0.25
kendrick perkins Curry, Stephen 0.7777777777777778
kenneth faried Acy, Quincy 0.25
kent bazemore Ibaka, Serge 0.6666666666666666
kentavious caldwell-pope Adams, Steven 0.25
kevin garnett Carroll, DeMarre 0.7333333333333333
kevin love Nene 0.6470588235294118
kevin seraphin Aldridge, LaMarcus 0.5882352941176471
```

Figure: Above is the output from the reducer in alphabetical order. The first column is the player, the second column is the defender with the highest fear score, and the third is the fear score.

Goal 2: Classify each player's records into 4 comfortable zones and find which comfortable zone is best for players based on their hit rate

1. Problem Description:

For each player, a data point / matrix is defined **[SHOT_CLOCK, SHOT_DISTANCE, CLOSEST_DEFENDER_DISTANCE]**, and parallel k-means clustering algorithm is applied to find the 4 clusters/ zones iteratively based on the player's hit rate. Fig.1 is the 3D representation of the matrix, wherein four comfortable zones/clusters are shown. The zone with the highest hit rate (% of shots made or high success rate) is considered as the most comfortable zone of the player.

2. Dataset Description:

The NBA dataset contains 21 Columns (e.g., player_name, CLOSE_DEF_DIST) and 128,069 rows (all players information). Missing values are dropped using specific code functions. Sample dataset is shown in Fig.2, which shows shots taken during the 2014-2015 season, who took the shot, where on the floor was the shot taken from, who was the nearest defender, how far away was the nearest defender, time on the shot clock, and much more. The column titles are generally self-explanatory.

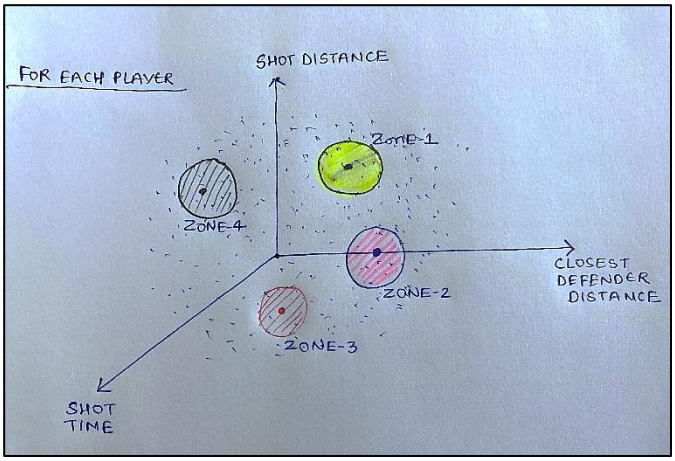


Fig. 1 Sketch showing the four comfortable zones of players based on their hit rate. (SHOT_TIME is SHOT_CLOCK)

3. Source of Dataset:

The source of the data set is Kaggle: https://www.kaggle.com/dansbecker/nba-shot-logs%20

GAME_ID	MATCHUP	LOCATION	W	FINAL_MA	SHOT_NUM	PERIOD	GAME_CLC	SHOT_CLO	DRIBBLES	TOUCH_TI	SHOT_DIST	PTS_TYPE	SHOT_RESI	CLOSEST_D	CLOSEST_I	CLOSE_DEF_DIST	PTS	player_nar	player_id
21400899	MAR 04, 2(	A	W	24	1	1	01:09	10.8	2	1.9	7.7	2	made	Anderson,	101187	1.3	1	2	brian robe	203148
21400899	MAR 04, 2(	A	W	24	2	1	00:14	3.4	0	0.8	28.2	3	missed	Bogdanovi	202711	6.1	0	0	brian robe	203148
21400899	MAR 04, 2(	A	W	24	3	1	00:00		3	2.7	10.1	2	missed	Bogdanovi	202711	0.9	0	0	brian robe	203148
21400899	MAR 04, 2(	A	W	24	4	2	11:47	10.3	2	1.9	17.2	2	missed	Brown, Ma	203900	3.4	0	0	brian robe	203148
21400899	MAR 04, 2(	A	W	24	5	2	10:34	10.9	2	2.7	3.7	2	missed	Young, Tha	201152	1.1	0	0	brian robe	203148
21400899	MAR 04, 2(	A	W	24	6	2	08:15	9.1	2	4.4	18.4	2	missed	Williams, D	101114	2.6	0	0	brian robe	203148
21400899	MAR 04, 2(	A	W	24	7	4	10:15	14.5	11	9	20.7	2	missed	Jack, Jarret	101127	6.1	0	0	brian robe	203148
21400899	MAR 04, 2(	A	W	24	8	4	08:00	3.4	3	2.5	3.5	2	made	Plumlee, M	203486	2.1	1	2	brian robe	203148

Fig.2 NBA Sample data set

4. Process Design and Description (MapReduce Program):

This is a parallel k-means clustering problem which iteratively finds the normalized centroids for the entire dataset. Fig.3 illustrates the map-reduce design for this problem. It will take two rounds of map-reduce; input is passed from NBA shot logs dataset. *Mapper-1.py* is designed to read the input csv file, take player and player's stats and outputs it to the first reducer (*reducer-1.py*). The next step is to centroids initialization which will be done by reducer-1.py. It goes player by player to extract average stats when the shot is made, initialized centroids to random points in the range of the data, outputs each players id, average stat line, and its current centroid to the second mapper

Mapper-2.py takes the player and their stat line as well as the current centroid then finds the closest centroid and then groups the player into a category with other players who are also closest to the other centroid. Lastly, *reducer-2.py* will perform the centroids initialization task iteratively and find the normalized clusters or comfortable zones for each player. It will basically take the player, stat line, and group and then averages the stat lines by group to produce the new centroids, and then passes back to the mapper (information about player, stat line, and updated/new centroids).

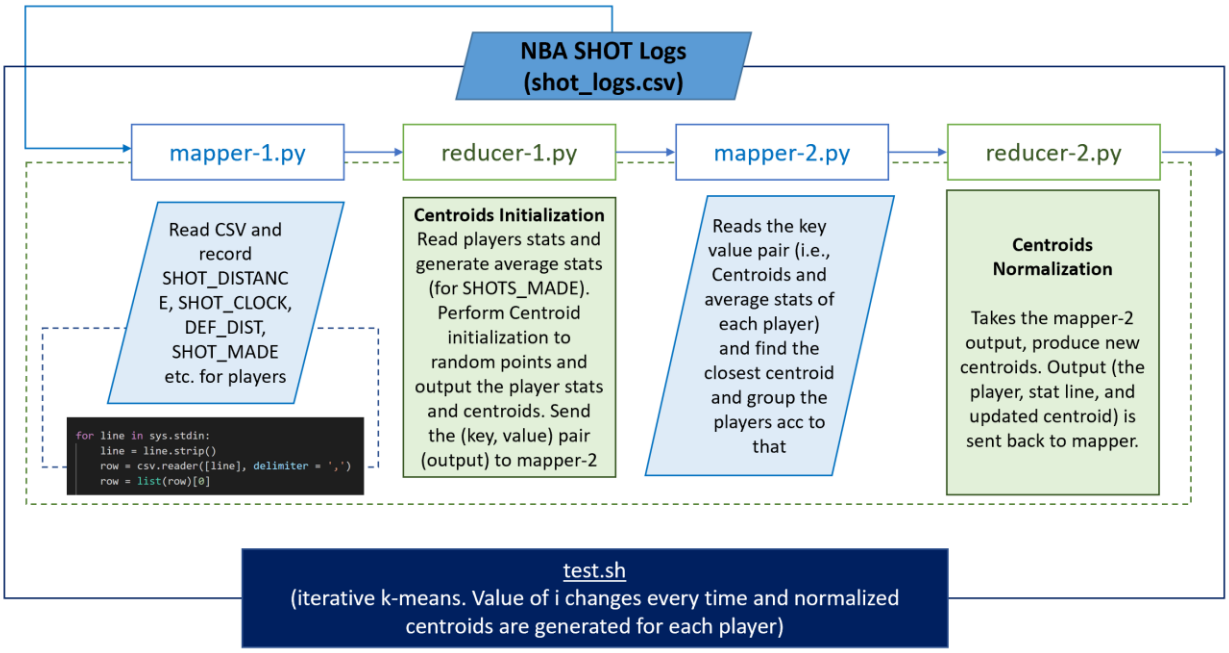


Fig.3 Illustrates the map-reduce design for this problem. It is an iterative process, runs iteratively to find the best and normalized centroids (or zones with high hit rates) for each player

5. Mapper and Reducer Results (Code and Output):

Step -1 (mapper-1.py): Input the dataset (shot_logs.csv)

```
import sys
import string
import csv

for line in sys.stdin:
    line = line.strip()
    row = csv.reader([line], delimiter = ',')
    row = list(row)[0]

    key = row[19]
    shot_clock = row[8]
    shot_dist = row[11]
    close_def = row[16]

    print(key + '\t' + '1' + '\t' + shot_clock + '\t' + shot_dist + '\t' +
close_def)
```

```
jarrett jack    1       8.5     18.8    3.9
jarrett jack    1       2.9     17.6    8.3
jarrett jack    1       12.5    18.7    3.9
jarrett jack    1       14.5    5.5     4.1
jarrett jack    1       9.8     22.3    3.3
jarrett jack    1       3.4     8.6     2.2
jarrett jack    1       17      15.2    1.9
jarrett jack    1       13.5    10.8    0.7
jarrett jack    1       21      4.8     2.8
jarrett jack    1               4.8     5
jarrett jack    1       9.3     4.7     3
jarrett jack    1       9.5     3.6     4
jarrett jack    1       14.3    13.3    5
jarrett jack    1       19.2    1.3     2.9
jarrett jack    1       18.8    2       0
jarrett jack    1       16.4    9.3     2.8
jarrett jack    1       15      1.7     0.8
jarrett jack    1       12.9    13.8    3.5
jarrett jack    1       13.2    16.8    2.9
jarrett jack    1       8       5.4     2.7
jarrett jack    1       21.9    18.5    4.6
jarrett jack    1       6       1.4     1.4
jarrett jack    1       21.8    17      6.9
jarrett jack    1       11.2    2.2     4.2
jarrett jack    1       10.3    7.5     0.9
jarrett jack    1       4       6.8     0.9
```

Fig. 4 mapper-1 output (Col.1 is Player name, Col.2 is instance, Col.3 is shot_clock, Col.4 is shot_dist, Col. 5 is defender_dis)

Step-2 (reducer-1.py) Centroids Initialization :

```
#!/usr/bin/python
#!/usr/bin/python
from operator import itemgetter
from collections import defaultdict
import sys

player_prev = ''
player_mat = []
player_totals = [0,0,0,0]
counter = 0
cent_1 = [5,5,5]
cent_2 = [18,18,4]
cent_3 = [15,6,5]
cent_4 = [7,14,3]
cent_list = [cent_1,cent_2,cent_3,cent_4]

for line in sys.stdin:
    line = line.strip()

    #print(line)
    player, num, shot_clock, shot_dist, close_def = line.split('\t')

    try:
        shot_clock = float(shot_clock)
    except:
        shot_clock = float(0)

    if player == player_prev:
        player_mat = [int(num),shot_clock,float(shot_dist),float(close_def)]
        for i in range(4):
            player_totals[i] += player_mat[i]
        player_prev = player
    else:
        for i in range(1,4):
            if player_totals[0]!=0:
                player_totals[i] = player_totals[i]/player_totals[0]
        player_totals = player_totals[1:4]
        counter += 1
        r_l = cent_list[counter%4]
        if player_totals[0] != 0:
            print(player_prev + '\t' + str(player_totals[0])+ '\t' + str(player_t
otals[1])+ '\t' + str(player_totals[2]) + '\t'+ str(r_l[0]) + '\t' + str(r_l[1])
+ '\t' + str(r_l[2]))
```

```
        player_totals = [0,0,0,0]
        player_totals += player_mat
        player_prev = player
```

The output of this code is shown in Fig. 5

Output from first or second reducer function

lebron james	12.7316561845	11.2849056604	4.31970649895	18	18	4
lou williams	12.2522058824	16.9941176471	4.47095588235	15	6	5
luc mbah a moute	12.981	12.0645 4.589	7	14	3	
luis scola	12.5721461187	10.0219178082	3.83789954338	5	5	5
luke babbitt	11.8847222222	20.025	6.14444444444	18	18	4
luol deng	12.3003663004	11.3586080586	4.06593406593	15	6	5
manu ginobili	10.6091346154	12.4706730769	3.97596153846	7	14	3
marc gasol	10.2896039604	9.47128712871	3.99504950495	5	5	5
marcin gortat	13.2148387097	5.75064516129	3.25516129032	18	18	4
marco belinelli	11.5128571429	17.0107142857	5.085	15	6	5
marcus morris	12.9307053942	15.6659751037	4.6601659751	7	14	3
marcus smart	13.1364485981	17.3878504673	4.4476635514	5	5	5
marcus thornton	13.3550387597	15.3511627907	3.91472868217	18	18	4
mario chalmers	11.8980861244	11.2851674641	4.08421052632	15	6	5

Fig.5: Shows the input for the second mapper function either from the first or second reducer function. The columns 2-4 are the statistical matrix while columns 5-7 are the current centroid the instance has.

Step-3 (mapper-2.py):

```
import sys
import math

player_cent = {'player':[],'stat_matrix':[] , 'current_cent':[] ,'dist_matrix':[]
,'min_index_group':[]}

for line in sys.stdin:
    line = line.strip()
    line = line.split('\t')
    try:
        player_cent['player'].append(line[0])
        player_cent['stat_matrix'].append(line[1:4])
        player_cent['current_cent'].append(line[4:7])
    except:
        pass

cent_list = []
for i in player_cent['current_cent']:
    if not i in cent_list:
        cent_list.append(i)
```

```
def e_dist(stat_matrix,cent_list):
    mat_dist = []
    for i in range(0,len(cent_list)):
        dist_sqrd = 0
        for j in range(0,len(cent_list[i])):
            dist_sqrd += (float(stat_matrix[j]) - float(cent_list[i][j]))**2

        dist = math.sqrt(dist_sqrd)
        mat_dist.append(dist)

    player_cent['dist_matrix'].append(mat_dist)

for k in range(0,len(player_cent['stat_matrix'])):
    e_dist(player_cent['stat_matrix'][k],cent_list)

for i in player_cent['dist_matrix']:
    player_cent['min_index_group'].append(i.index(min(i)))

print(player_cent['dist_matrix'])

for i in range(0,len(player_cent['stat_matrix'])):
    print(player_cent['player'][i] + '\t' + str(player_cent['stat_matrix'][i][0])
+ '\t' + str(player_cent['stat_matrix'][i][1])+ '\t' + str(player_cent['stat_matr
ix'][i][2]) + '\t' + str(player_cent['min_index_group'][i]))
```

The output of this code is shown in Fig. 6.

Output from second mapper function

```
ty lawson       13.1585139319   12.4659442724   4.43034055728   1
tyler hansbrough        13.9891304348   4.13260869565   2.74782608696   0
tyler zeller    13.4021929825   5.74210526316   3.15964912281   0
tyreke evans    12.562278481    8.31898734177   3.69417721519   0
tyson chandler  15.8947598253   3.46331877729   3.3     0
udonis haslem   11.1263157895   7.30701754386   3.56842105263   0
victor oladipo  13.1070336391   9.60581039755   4.05657492355   0
vince carter    12.0086021505   17.535483871    4.62150537634   3
wayne ellington 11.2632978723   17.0127659574   4.33085106383   1
wesley johnson  12.2010204082   16.5321428571   4.97857142857   3
wesley matthews 12.5850746269   16.9205970149   4.88776119403   3
wilson chandler 13.3574257426   12.8623762376   3.97623762376   1
zach lavine     13.5455128205   12.1282051282   3.90256410256   0
zach randolph   11.6287461774   7.15810397554   2.82262996942   0
```

Fig.5: This is passed from the mapper-2.py function to the reducer-2 function. The last column is the index of centroid. The reducer calculates the updated centroid by aggregating all instances in the same group.

Step-4 (reducer-2.py) Centroids Normalization:

```
import sys

player_cent = {'player':[],'stat_matrix':[] , 'new_cent':[] ,'cent_group':[]}

for line in sys.stdin:
    line = line.strip()
    line = line.split('\t')
    try:
        player_cent['player'].append(line[0])
        player_cent['stat_matrix'].append(line[1:4])
        player_cent['cent_group'].append(line[4])
    except:
        pass


cent_dict = {'cent_0':[],'cent_1':[] , 'cent_2':[] ,'cent_3':[]}
for i in range(0,len(player_cent['stat_matrix'])):
    try:
        lst_name = 'cent_' + str(player_cent['cent_group'][i])
        cent_dict[lst_name].append(player_cent['stat_matrix'][i])
    except:
        pass

new_cent = []
for i in range(0,4):
    lst_name = 'cent_' + str(player_cent['cent_group'][i])
    sums = [0,0,0]
    length = len(cent_dict[lst_name])
    for j in cent_dict[lst_name]:
        for k in range(0,len(j)):
            j[k]= float(j[k])
            sums[k] += j[k]
    for l in range(0,len(sums)):
        sums[l] = sums[l]/length

    new_cent += sums

new_cent = [new_cent[0:3],new_cent[3:6],new_cent[6:9],new_cent[9:12]]


for i in range(0,len(player_cent['cent_group'])):
    player_cent['new_cent'].append(new_cent[int(player_cent['cent_group'][i])])
```

```
for i in range(0,len(player_cent['new_cent'])):
    try:
        print(player_cent['player'][i] + '\t' + str(player_cent['stat_matrix'][i]
[0])+ '\t' + str(player_cent['stat_matrix'][i][1])+ '\t' + str(player_cent['stat_
matrix'][i][2])+ '\t' + str(player_cent['new_cent'][i][0])+ '\t' + str(player_cen
t['new_cent'][i][1])+ '\t' + str(player_cent['new_cent'][i][2]))
    except:
        pass
```

The output of this code is shown in Fig. 5

6. Code Testing (test.sh file):

```
#!/bin/sh
../../start.sh
/usr/local/hadoop/bin/hdfs dfs -rm -r /pro_1/part2/input/
/usr/local/hadoop/bin/hdfs dfs -rm -r /pro_1/part2/output/
/usr/local/hadoop/bin/hdfs dfs -mkdir -p /pro_1/part2/input/
/usr/local/hadoop/bin/hdfs dfs -copyFromLocal ../../mapreduce-
test-data/part2/shot_logs.csv /pro_1/part2/input/
/usr/local/hadoop/bin/hadoop jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-
2.9.2.jar \
-file ../../mapreduce-test-python/pro_1/part2/mapper_1.py -
mapper ../../mapreduce-test-python/pro_1/part2/mapper_1.py \
-file ../../mapreduce-test-python/pro_1/part2/reducer_1.py -
reducer ../../mapreduce-test-python/pro_1/part2/reducer_1.py \
-input /pro_1/part2/input/* -output /pro_1/part2/output/
for i in {0..5}
do
/usr/local/hadoop/bin/hadoop jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-
2.9.2.jar \
-file ../../mapreduce-test-python/pro_1/part2/mapper_2.py -
mapper ../../mapreduce-test-python/pro_1/part2/mapper_2.py \
-file ../../mapreduce-test-python/pro_1/part2/reducer_2.py -
reducer ../../mapreduce-test-python/pro_1/part2/reducer_2.py \
-input /pro_1/part2/input/* -output /pro_1/part2/output/
done
/usr/local/hadoop/bin/hdfs dfs -cat /pro-1/part2/output/
../../stop.sh
```

7. **Final Output of Four Players:**
   The comfortable zones for James Harden, Chris Paul, Lebron James, Stephan Curry are shown in below table:

james harden	12.339323467230436	13.16046511627906	3.8689217758985195	12.579797048545766	12.243114793351713	4.253021397200293
chris paul	12.423584905660372	15.654716981132074	4.745518867924525	12.603851444525631	11.367830760906081	4.138001170631598
lebron james	12.73165618448638	11.284905660377364	4.319706498951785	12.603851444525631	11.367830760906081	4.138001170631598
stephen curry	14.992110874200424	15.464818763326232	4.615138592750533	12.579797048545766	12.243114793351713	4.253021397200293

Here: Column1.Player_name, Column 2. SHOT_CLOCK, Column 3. SHOT_DIST, Column 4. DEF_DIST, Column 5. CENTROID SHOT_CLOCK, Column 6. CENTROID SHOT_DIST, Column 7. CENTROID DEF_DIST