Big Data Mining and Text Analytics (16:198:674)

Project Report

Winning Fantasy Sports Using Spark

Pavan Nihal Renducintala (172002406)

Srinivas Gaddam (174009745)

Contents

1 Project goals and why it is interesting

Determining the value of a player in fantasy sports is essential and it is not a trivial task. The ultimate goal of this project is to produce a ranked list of players in terms of their value, but being able to assign a numerical value in order to compare players directly would also be useful for many use cases.

Many sports fans turn into statisticians to get an edge in their fantasy sports leagues. Depending on one's technical skills, this "edge" is usually no more sophisticated than simple spreadsheet analysis, but some particularly intense people go to the extent of creating their own player rankings and projection systems. This project is interesting because it incorporates the use of technology to determine the right players to choose in fantasy leagues in order to win.

2 Data Gathering and Preprocessing

We begin by grabbing our data from Basketball-Reference.com, which allows us to export season-based statistics by year as CSV files. We will gather data from the current season all the way back to the 1979-1980 season. This gives us 26 full seasons of data to use for our analysis.

We note that the CSV files have some missing data, repeated headers, etc., which we will have to account for during our data processing.

We will begin our processing by first loading then into HDFS by using the following command.

```
hadoop fs -put BasketballStats /user/cloudera/
```

After looking at the data clearly, one important step was to include "year" in each file as each line had no indication of the year. We did that by setting up the spark shell and running the following:

```
//process files so that each line includes the year
for (i <- 1980 to 2016){
    println(i)
    val yearStats = sc.textFile(s"/user/cloudera/BasketballStats/data/leagues_NBA_$i*").repartition(sc.defaultParallelism)
    yearStats.filter(x => x.contains(",")).map(x =>  (i,x)).saveAsTextFile(s"/user/cloudera/BasketballStatsWithYear/data/$i/")
}
```

Another important data cleaning task was to remove junk rows and clean up data entry errors as well. For this we began by reading all our data into a single RDD. Then filter our rows we don't want and cache the result.

```
//read in all stats
val stats=sc.textFile("/user/cloudera/BasketballStatsWithYear/data/*/*").repartition(sc.defaultParallelism)

//filter out junk rows, clean up data entry errors as well
val cleanStats=stats.filter(x => !x.contains("FG%")).filter(x => x.contains(",")).map(x=>x.replace("*","").replace(",,",",0,"))

//remove duplicate rows, use "TOT" row when available
val filteredStats=cleanStats.map{x=>
    val pieces=x.split(",")
    val year=pieces(0)
    val name=pieces(2)
    val team=pieces(5)
    (year+"_"+name,Map(team->x))
}.reduceByKey(_++_).map{ case(x,y)=>
    if (y.contains("TOT")){
    y("TOT")
    }else{
    y.last._2
    }
}
```

3 Describing the contents of data in detail

As this project is all about, Fantasy Team, we have all the statistics related to the players in the data. The clean data looks like this :-

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P	2P	2PA	2P	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST
1	Alex Acker	SG	26	TOT	25	0	8	1.2	3	0.395	0.3	0.8	0.35	0.9	2.2	0.411	0.441	0.2	0.4	0.5	0.3	0.6	1	0.5
2	Hassan Adams	SG	24	TOR	12	0	4.3	0.3	1.1	0.308	0	0		0.3	1.1	0.308	0.308	0.3	0.5	0.5	0.1	0.5	0.6	0.1
3	Arron Afflalo	SG	23	DET	74	8	16.7	1.8	4.1	0.437	0.6	1.4	0.402	1.2	2.6	0.456	0.508	0.8	1	0.817	0.4	1.4	1.8	0.6
4	Maurice Ager	SG	24	NJN	20	0	4.9	0.8	2.2	0.349	0	0.3	0	0.8	1.9	0.405	0.349	0.2	0.4	0.5	0.2	0.4	0.5	0.2
5	Blake Ahearn	PG	24	SAS	3	0	6.3	0.7	2	0.333	0.7	1.3	0.5	0	0.7	0	0.5	0.7	0.7	1	0	0.3	0.3	0.7
6	Alexis Ajinca	PF	20	CHA	31	4	5.9	0.8	2.2	0.362	0	0.1	0	0.8	2.2	0.373	0.362	0.6	0.9	0.714	0.3	0.7	1	0.1
7	LaMarcus Aldridge	PF	23	POR	81	81	37.1	7.4	15.3	0.484	0.1	0.3	0.25	7.3	15	0.489	0.486	3.2	4.1	0.781	2.9	4.6	7.5	1.9
8	Joe Alexander	SF	22	MIL	59	0	12.1	1.7	4.2	0.416	0.3	0.8	0.348	1.5	3.4	0.432	0.449	1	1.4	0.699	0.7	1.2	1.9	0.7
9	Malik Allen	PF	30	MIL	49	3	11.8	1.5	3.5	0.429	0	0	0	1.5	3.4	0.432	0.429	0.2	0.4	0.476	0.7	1.4	2.1	0.7
10	Ray Allen	SG	33	BOS	79	79	36.4	6.3	13.2	0.48	2.5	6.2	0.409	3.8	7	0.542	0.575	3	3.2	0.952	0.8	2.7	3.5	2.8
11	Tony Allen	SG	27	BOS	46	2	19.3	3	6.2	0.482	0.1	0.6	0.222	2.8	5.6	0.51	0.493	1.7	2.4	0.725	0.5	1.8	2.3	1.4
12	Morris Almond	SG	23	UTA	25	1	10.2	1.3	3.2	0.407	0.2	0.7	0.294	1.1	2.6	0.438	0.438	0.8	1	0.808	0.4	1.1	1.4	0.3
13	Rafer Alston	PG	32	TOT	77	76	31.8	4.1	10.6	0.385	1.5	4.3	0.338	2.6	6.4	0.417	0.454	2	2.7	0.75	0.4	2.5	2.9	5.3
14	Louis Amundson	PF	26	PHO	76	0	13.7	1.8	3.3	0.536	0	0	0	1.8	3.3	0.538	0.536	0.7	1.6	0.442	1.7	1.9	3.6	0.4
15	Chris Andersen	C	30	DEN	71	1	20.6	2.3	4.1	0.548	0	0.1	0.2	2.2	4	0.56	0.551	1.8	2.5	0.718	2.3	3.9	6.2	0.4
16	Ryan Anderson	PF	20	NJN	66	30	19.9	2.4	6.2	0.393	1	2.9	0.365	1.4	3.3	0.417	0.478	1.5	1.8	0.845	1.6	3.1	4.7	0.8
17	Carmelo Anthony	SF	24	DEN	66	66	34.5	8.1	18.3	0.443	1	2.6	0.371	7.2	15.7	0.455	0.469	5.6	7.1	0.793	1.6	5.2	6.8	3.4
18	Joel Anthony	C	26	MIA	65	28	16.1	0.9	1.8	0.483	0	0		0.9	1.8	0.483	0.483	0.5	0.7	0.652	1.4	1.7	3	0.4
19	Gilbert Arenas	PG	27	WAS	2	2	31.5	3	11.5	0.261	1	3.5	0.286	2	8	0.25	0.304	6	8	0.75	0.5	4	4.5	10
20	Trevor Ariza	SF	23	LAL	82	20	24.4	3.3	7.3	0.46	0.7	2.3	0.319	2.6	4.9	0.526	0.511	1.5	2.1	0.71	1.4	2.9	4.3	1.8
Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	FG%	3P	3PA	3P	2P	2PA	2P	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST
21	Hilton Armstrong	C	24	NOH	70	29	15.6	2	3.5	0.561	0	0	0	2	3.5	0.566	0.561	0.9	1.4	0.633	1.2	1.6	2.8	0.4

In the dataset we have the following data for each player(In a single season):

Table 1: Data Variables in given Dataset

Rk	Rank
Pos	Position
Age	Age of Player at the start of February 1st of that season.
Tm	Team
G	Games
GS	Games Started
MP	MinutesPlayed Per Game
FG	Field Goals Per Game
FGA	Field Goal Attempts Per Game
FG%	Field Goal Percentage
3P	3-Point Field Goals Per Game
3PA	3-Point Field Goal Attempts Per Game
3P%	3-Point Field Goal Percentage
2P	2-Point Field Goals Per Game
2PA	2-Point Field Goal Attempts Per Game
2P%	2-Point Field Goal Percentage
eFG%	Effective Field Goal Percentage
FT	Free Throws Per Game
FTA	Free Throw Attempts Per Game
FT%	Free Throw Percentage
ORB	Offensive Rebounds Per Game
DRB	Defensive Rebounds Per Game
TRB	Total Rebounds Per Game
AST	Assists Per Game
STL	Steals Per Game
BLK	Blocks Per Game
TOV	Turnovers Per Game
PF	Personal Fouls Per Game
PS/G	Points Per Game

*eFG%-Effective Field Goal Percentage This statistic adjusts for the fact that a 3-point field goal is worth one more point than a 2-point field goal.

Though we have the data, we cannot compare the data (statistics) directly as it consists of different seasons and we need to find a standard base for comparision. So to have a good comparison, it is better for us to use normal distribution. So to analyze the given data we calculate z-scores and normalized z-scores so that we can easily compare statistics.

In this analysis we are going to consider 9 main statistics which are:

FG%, FT%, 3P, TRB, AST, STL, BLK, TOV, PTS,

We normalize all the above statistics by subtracting mean from each element and dividing it by standard deviation. (We get the z scores from this calculation) .

For FG% and FT% we don't take the number of shots taken into consideration.To overcome this, we multiply with the ratio of FGA(person) to the mean FGA (average of that league) and multiply it to z-score.

Now after getting all the individual z-scores, we will add up all the individual z-scores and we calculate zTot. (zTot is player's performance relative to others during that season).

Z-score might be misleading in some types of fantasy games(Head to Head). For example if a person has +17 in one individual z-score and -1 in remaining then his aggregate is 9 which is same as player who has all 1 and aggregate is 9. But in Head to Head they only care about who won (don't care about score margin)

For that type of fantasy games we introduce Normalized z-score which we restrict each individual z-score to range of -1 to 1. i.e. divide the each element by absolute maximum value.

BballData: In this class we will store the values from given dataset and also z-scores and normalized z-scores as arrays and normalized z-scores respectively.

Bbparse: Function which takes input from our dataset and calculates new values and it gives back the BballData object.

BbStatCounter:It is a object which is used to compute the overall values for the dataset. In this object we will use a class from Spark. i.e. StatCounter class so that we calculate all the mathematical operations required for z-scores and normalized z-scores such as maximum, minimum, variance across all the main statistics we considered above.

processStats:It is a function which is used to calculate the minimum, maximum, variance, mean, average.It takes an Resilient Distributed Datasets of Strings ,which contains all our statistical data ,and we will send it to parse via Bbparse and then it categorizes the data by year,then we compute the minimum,maximum,variance,standard deviation for each numerical data.and it is stored in the array.It is important function for processing data.

Now we begin the processing of the data.

The first step is Calculating the z-Scores.

We pass the Filtered dataset to processStats so that we can acquire the total statistics into map and broadcast.We broadcast a value so that it is accessible to every node (local copy).In this step we send all the statistical values for processing i.e 2-pointers,3-pointers and every statistical value we have.

```
//process stats and save as map
val txtStat=Array("FG","FGA","FG%","3P","3PA","3P%","2P","2PA","2P%","eFG%","FT","FTA","FT%","ORB","DRB","TRB","AST","STL","BLK","TOV","PF","PTS")
val aggStats=processStats(filteredStats,txtStat).collectAsMap

//collect rdd into map and broadcast
val broadcastStats=sc.broadcast(aggStats)
```

Now as we have the data in map and broadcast we need to calculate the z-scores.

Now we send the the above data to calculate the z-scores of the 9 important statistics and again we map and broadcast the array. Next we send the above stats again and calculate the normalized z-stats and again we map the array.

```
//********************
//Compute Z-Score Stats Per Year
//********************

//parse stats, now tracking weights
val txtStatZ=Array("FG","FT","3P","TRB","AST","STL","BLK","TOV","PTS")
val zStats=processStats(filteredStats,txtStatZ,broadcastStats.value).collectAsMap

//collect rdd into map and broadcast
val zBroadcastStats=sc.broadcast(zStats)


//********************
//Compute Normalized Stats Per Year
//********************

//parse stats, now normalizing
val nStats=filteredStats.map(x=>bbParse(x,broadcastStats.value,zBroadcastStats.value))
```

The initial processing the data is completed .Now we need to store the data so that we can query the data.So for this we will create a table and we will use SQL for query purposes.

We will create a DataFrame table and save it as table .To do these we use CreateDataFrame and saveAsTable function .So we change RDD to RDD[Row] and define schema.

```
//map RDD to RDD[Row] so that we can turn it into a dataframe
val nPlayer = nStats.map(x => Row.fromSeq(Array(x.name,x.year,x.age,x.position,x.team,x.gp,x.gs,x.mp) ++ x.stats ++ x.statsZ ++ Array(x.valueZ) ++ x.
    statsN ++ Array(x.valueN)))
```

```
//create schema for the data frame
val schemaN = StructType(
      StructField("name", StringType, true) ::
      ...
```

. . .
)

```
//create data frame
val dfPlayersT=sqlContext.createDataFrame(nPlayer,schemaN)

//save all stats as a temp table
dfPlayersT.registerTempTable("tPlayers")

//calculate exp and zdiff, ndiff
val dfPlayers=sqlContext.sql("select age-min_age as exp,tPlayers.* from tPlayers join (select name,min(age)as min_age from tPlayers group by name) as t1
    on tPlayers.name=t1.name order by tPlayers.name, exp ")

//save as table
dfPlayers.saveAsTable("Players")
//filteredStats.unpersist()
```

4 Programmatic Data Analysis using Spark SQL

After the above analysis we have very large chunks of data .Now we can start analysing the data.We will use spark SQL. Lets see some sample analysis:

Check on the race to become an MVP(Most Valueble Player) and see the results on Spark-Shell

We do this by checking the players and their z scores

```
scala> sql("Select name, zTot from Players where year=2016 order by zTot desc").take(10).foreach(println)
[Stephen Curry,19.225140373459713]
[Kevin Durant,15.799353456355771]
[James Harden,13.468369657962842]
[Kawhi Leonard,13.2363791864673]
[Anthony Davis,12.175842820706395]
[Chris Paul,11.955601275340271]
[Russell Westbrook,11.238867658560213]
[Hassan Whiteside,11.168074417229784]
[LeBron James,10.624655035977618]
[Kyle Lowry,10.322277991164933]
```

```
scala> sql("Select name, nTot from Players where year=2016 order by nTot desc").take(10).foreach(println)
[Stephen Curry,3.786415095914835]
[Kevin Durant,2.780663744102033]
[Kawhi Leonard,2.689545347592425]
[James Harden,2.4934388824887086]
[Chris Paul,2.4832088437975663]
[Anthony Davis,2.442810950355021]
[Russell Westbrook,2.3535182550768834]
[LeBron James,2.236596964707329]
[Kyle Lowry,2.1984724401500344]
[Paul Millsap,2.080800689897187]
```

We see similar players in both lists, but the ordering is slightly different. James Harden and Kawhi Leonard are neck and neck in z-score, but Leonard has a slight edge in normalized z-score, which suggests that he contributes more significantly across a wider spectrum of stats than does Harden, or that Harden may be overkill in the stats that he is good in.

Similarly, lets see another analysis

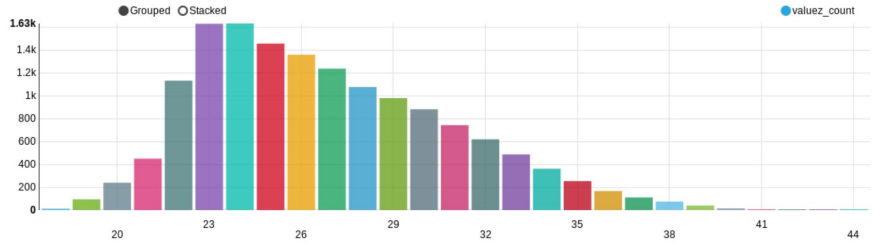
```
scala> sql("select name,3p,z3p,year from players order by 3p desc").take(10).foreach(println)
[Stephen Curry,5.1,6.18946211163782,2016]
[Stephen Curry,3.6,4.395853442084638,2015]
[Klay Thompson,3.5,3.9448574050889116,2016]
[Stephen Curry,3.5,4.369359176456364,2013]
[Ray Allen,3.4,4.735522316886375,2006]
[George McCloud,3.3,4.225692703960359,1996]
[Stephen Curry,3.3,3.886769608832293,2014]
[Dennis Scott,3.3,4.225692703960359,1996]
[Ray Allen,3.3,4.863432467545103,2002]
[Klay Thompson,3.1,3.651613974257507,2015]
```

```
scala> sql("select name,3p,z3p,year from players order by z3p desc").take(10).foreach(println)
[Joe Hassett,1.3,10.204767074071892,1981]
[Darrell Griffith,1.1,8.695692525219064,1984]
[Mike Dunleavy,1.1,8.695692525219064,1984]
[Mike Dunleavy,0.8,7.507037255311244,1983]
[Joe Hassett,1.0,7.433566476899131,1982]
[Darrell Griffith,1.2,6.938020916099367,1985]
[Brian Taylor,1.2,6.934806295220304,1980]
[Michael Adams,2.5,6.672086264728188,1991]
[Don Buse,0.9,6.650823393818539,1982]
[Danny Ainge,1.8,6.274029091222401,1988]
```

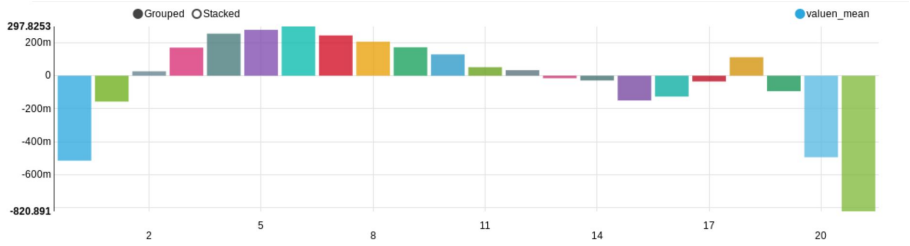
The first list is top 3-pointers over all the seasons.In which we can see that Stephen Curry is leading in it as he is most 3 point shooters in all time per season.But in second we compare the normalized 3 pointers i.e comparision of 3-pointers by player to total 3 pointers in which we can see Stephen Curry is not even in top 5 .It is Joe Hassett who was player in 1981 when 3-pointers are not scored as regularly as today and he has best shooting compared to others in that season.

Similarly, lets do an other analysis and see visualizations on Apache Hue rather than spark shell

Lets see how number of players vary according to age group



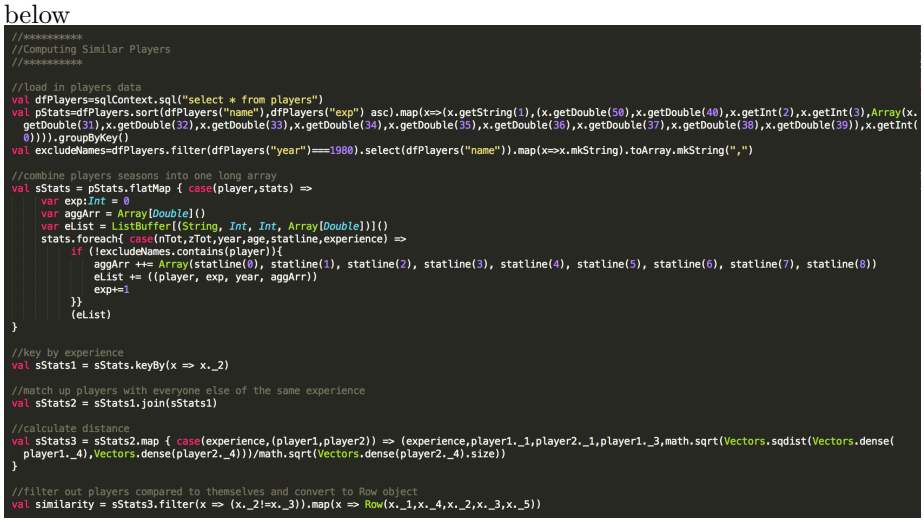
Now lets see the performance of players based on their experience



5 Machine Learning

There were two different aspects of Machine learning in this project. Lets look both of the in detail.

1. Similarity between players : The detailed steps about how we found the similarities between steps can be found from the comments in the pictures below



```
//**********
//Computing Similar Players
//**********

//load in players data
val dfPlayers=sqlContext.sql("select * from players")
val pStats=dfPlayers.sort(dfPlayers("name"),dfPlayers("exp") asc).map(x=>(x.getString(1),(x.getDouble(50),x.getDouble(40),x.getInt(2),x.getInt(3),Array(x.
  getDouble(31),x.getDouble(32),x.getDouble(33),x.getDouble(34),x.getDouble(35),x.getDouble(36),x.getDouble(37),x.getDouble(38),x.getDouble(39)),x.getInt(
  0)))).groupByKey()
val excludeNames=dfPlayers.filter(dfPlayers("year")===1980).select(dfPlayers("name")).map(x=>x.mkString).toArray.mkString(",")

//combine players seasons into one long array
val sStats = pStats.flatMap { case(player,stats) =>
     var exp:Int = 0
     var aggArr = Array[Double]()
     var eList = ListBuffer[(String, Int, Int, Array[Double])]()
     stats.foreach{ case(nTot,zTot,year,age,statline,experience) =>
          if (!excludeNames.contains(player)){
               aggArr ++= Array(statline(0), statline(1), statline(2), statline(3), statline(4), statline(5), statline(6), statline(7), statline(8))
               eList += ((player, exp, year, aggArr))
               exp+=1
          }}
          (eList)
}

//key by experience
val sStats1 = sStats.keyBy(x => x._2)

//match up players with everyone else of the same experience
val sStats2 = sStats1.join(sStats1)

//calculate distance
val sStats3 = sStats2.map { case(experience,(player1,player2)) => (experience,player1._1,player2._1,player1._3,math.sqrt(Vectors.sqdist(Vectors.dense(
  player1._4),Vectors.dense(player2._4)))/math.sqrt(Vectors.dense(player2._4).size))
}

//filter out players compared to themselves and convert to Row object
val similarity = sStats3.filter(x => (x._2!=x._3)).map(x => Row(x._1,x._4,x._2,x._3,x._5))
```

```
//schema for similar players
val schemaS = StructType(
      StructField("experience", IntegerType, true) ::
      StructField("year", IntegerType, true) ::
      StructField("name", StringType, true) ::
      StructField("similar_player", StringType, true) ::
      StructField("similarity_score", DoubleType, true) :: Nil
)


//create data frame
val dfSimilar = sqlContext.createDataFrame(similarity,schemaS)
dfSimilar.cache

//save as table
dfSimilar.saveAsTable("similar")
```

2. Performance prediction of players for current year : The detailed steps about how we found the predictions about performance of players for the current year can be found from the comments in the pictures below

```
val statArray = Array("zfg","zft","z3p","ztrb","zast","zstl","zblk","ztov","zpts")

for (stat <- statArray){
  //set up vector with features
  val features = Array("exp", stat+"0")
  val assembler = new VectorAssembler()
  assembler.setInputCols(features)
  assembler.setOutputCol("features")

  //linear regression
  val lr = new LinearRegression()

  //set up parameters
  val builder = new ParamGridBuilder()
  builder.addGrid(lr.regParam, Array(0.1, 0.01, 0.001))
  builder.addGrid(lr.fitIntercept)
  builder.addGrid(lr.elasticNetParam, Array(0.0, 0.25, 0.5, 0.75, 1.0))
  val paramGrid = builder.build()

  //define pipline
  val pipeline = new Pipeline()
  pipeline.setStages(Array(assembler, lr))

  //set up tvs
  val tvs = new TrainValidationSplit()
  tvs.setEstimator(pipeline)
  tvs.setEvaluator(new RegressionEvaluator)
  tvs.setEstimatorParamMaps(paramGrid)
  tvs.setTrainRatio(0.75)

  //define train and test data
  val trainData = sqlContext.sql("select name, year, exp, mp, " + stat + "0," + stat + "  as label from ml where year<2017")
  val testData = sqlContext.sql("select name, year, exp, mp, " + stat + "0," + stat + "  as label from ml where year=2017")

  //create model
  val model = tvs.fit(trainData)

  //create predictions
  val predictions = model.transform(testData).select("name", "year", "prediction","label")

  //Get RMSE
  val rm = new RegressionMetrics(predictions.rdd.map(x => (x(2).asInstanceOf[Double], x(3).asInstanceOf[Double])))
  //println("Mean Squared Error " + stat + " : " + rm.meanSquaredError)
  println("Root Mean Squared Error " + stat + " : " + rm.rootMeanSquaredError)

//save as temp table
  predictions.registerTempTable(stat + "_temp")

}
```

```
//add up all individual predictions and save as a table
val regression_total=sqlContext.sql("select zfg_temp.name, zfg_temp.year, z3p_temp.prediction + zfg_temp.prediction + zft_temp.prediction + ztrb_temp.
  prediction + zast_temp.prediction + zstl_temp.prediction + zblk_temp.prediction + ztov_temp.prediction + zpts_temp.prediction as prediction, z3p_temp.
  label + zfg_temp.label + zft_temp.label + ztrb_temp.label + zast_temp.label + zstl_temp.label + zblk_temp.label + ztov_temp.label + zpts_temp.label as
  label from z3p_temp, zfg_temp, zft_temp, ztrb_temp, zast_temp, zstl_temp, zblk_temp, ztov_temp, zpts_temp where zfg_temp.name=z3p_temp.name and z3p_temp
  .name=zft_temp.name and zft_temp.name=ztrb_temp.name and ztrb_temp.name=zast_temp.name and zast_temp.name=zstl_temp.name and zstl_temp.name=zblk_temp.
  name and zblk_temp.name=ztov_temp.name and ztov_temp.name=zpts_temp.name")
regression_total.saveAsTable("regression_total")
```

6 Insights form Machine Learning

1. Aspect 1 of Machine Learning : Lets look at the similarities between players and order them by their similarity score

```
SELECT * FROM similar ORDER BY similarity_score desc LIMIT 10
```

	experience	year	name	similar_player	similarity_score
1	1	1994	Shaquille O'Neal	Linton Townes	4.8540276959849198
2	5	1998	Shaquille O'Neal	Michael Adams	4.746425177310913
3	5	1991	Michael Adams	Shaquille O'Neal	4.746425177310913
4	0	2016	Duje Dukan	Shaquille O'Neal	4.7402911631692062
5	0	1993	Shaquille O'Neal	Duje Dukan	4.7402911631692062
6	8	2001	Shaquille O'Neal	Michael Adams	4.7388738281479625
7	8	1994	Michael Adams	Shaquille O'Neal	4.7388738281479625
8	7	1993	Michael Adams	Shaquille O'Neal	4.7266539710241631
9	7	2000	Shaquille O'Neal	Michael Adams	4.7266539710241631
10	2	2013	Quentin Richardson	Shaquille O'Neal	4.7164049051587158

2. Aspect 2 of Machine Learning : Lets look at the prediction for the Race for MVP for the current year

```
SELECT * FROM regression_total ORDER BY prediction DESC LIMIT 10
```

	name	year	prediction
1	Stephen Curry	2017	15.563982282699799
2	Kevin Durant	2017	12.528156974234882
3	James Harden	2017	11.188605604440932
4	Kawhi Leonard	2017	10.84722854171317
5	Anthony Davis	2017	10.342249399013458
6	Chris Paul	2017	9.3206252373673024
7	Russell Westbrook	2017	9.2994036298801603
8	Hassan Whiteside	2017	9.2133347765070521
9	Karl-Anthony Towns	2017	8.6132864055270275
10	Draymond Green	2017	8.251551010836863

After reading articles and blogs about current and previous NBA seasons, our results seemed very accurate

7 Conclusion

We started off this project by enriching our data which allows us to assign a value to a player which in turn helps us to determine best players in various categories. We then calculated at which point of his career is a player going to be at his peak. We also calculated how much the value of a player is going to change from one year to next. We got some insights from machine learning about how a player is gonna play in this current season which is the main criterion that helps a person to choose a player in his **Fantasy League**. Finally, This project was a great learning for both of us.

8 Technologies

Cloudera QuickStarts for CDH 5.10 on VirtualBox- Virtual machine
Scala - Language to write Spark code
Apache Hadoop - Distributed data storage and processing
Apache Impala - Writing high-preformance, low-latency SQL queries to visualize results on Hue
Spark-Shell - Compile scala code
SparkSQL - Write SQL queries on Spark-Shell
Apache Hue - Visualize Impala queries data

9 Link to code

https://github.com/nihal223/Winning-Fantasy-Sports

10 References

[1] Cloudera's blog - We got some of our ideas from this blog