Week2 - Discussion Post (KMeans)

James Quacinella

Simple Example

To start, lets make sure hadoop and RHadoop are working. Using an exmaple from https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md, I will first import the various libraries and then, for now, turn hadoop off.

```
library(rJava)
library(rhdfs)
```

```
##
## HADOOP_CMD=/home/yarn/hadoop/bin/hadoop
##
## Be sure to run hdfs.init()
```

```
hdfs.init()
library(rmr2)
```

```
## Warning: S3 methods 'gorder.default', 'gorder.factor', 'gorder.data.frame',
## 'gorder.matrix', 'gorder.raw' were declared in NAMESPACE but not found
```

```
## Please review your hadoop settings. See help(hadoop.settings)
```

```
rmr.options(backend = "local")
```

Simple MapReduce with no Hadoop

Below, we create some data from the Binomial distribution and look to map reduce to tell us the various counts of numbers generated. I added print statements to help me understand how things are being passed around:

```
groups = rbinom(32, n = 50, prob = 0.4)
groups_dfs_local = to.dfs(groups)

from.dfs(
  mapreduce(
    input = groups_dfs_local,
    map = function(k, v) {
      print("Value map:")
      print(v);
      keyval(v, 1);
    },
    reduce =
      function(k, vv) {
        print(paste("Key reduce: ", k));
```

```r
        print("Values reduce:")
        print(vv);
        keyval(k, length(vv)) ;
      }
  )
)
```

```
## [1] "Value map:"
##  [1] 16 18  8 16 14 16 11 11 10 18  8 15 12 11 17 16 13 14  9  9 10 11 13
## [24] 17 17 12 11  8 10 13 15 11 11 13 12 17 13 18 13 15 13 12 12 12 11 12
## [47] 16 13 10  8
## [1] "Key reduce:  16"
## [1] "Values reduce:"
## [1] 1 1 1 1 1
## [1] "Key reduce:  18"
## [1] "Values reduce:"
## [1] 1 1 1
## [1] "Key reduce:  8"
## [1] "Values reduce:"
## [1] 1 1 1 1
## [1] "Key reduce:  14"
## [1] "Values reduce:"
## [1] 1 1
## [1] "Key reduce:  11"
## [1] "Values reduce:"
## [1] 1 1 1 1 1 1 1 1
## [1] "Key reduce:  10"
## [1] "Values reduce:"
## [1] 1 1 1 1
## [1] "Key reduce:  15"
## [1] "Values reduce:"
## [1] 1 1 1
## [1] "Key reduce:  12"
## [1] "Values reduce:"
## [1] 1 1 1 1 1 1 1
## [1] "Key reduce:  17"
## [1] "Values reduce:"
## [1] 1 1 1 1
## [1] "Key reduce:  13"
## [1] "Values reduce:"
## [1] 1 1 1 1 1 1 1 1
## [1] "Key reduce:  9"
## [1] "Values reduce:"
## [1] 1 1
```

```
## $key
##  [1] 16 18  8 14 11 10 15 12 17 13  9
##
## $val
##  [1] 5 3 4 2 8 4 3 7 4 8 2
```

Simple MapReduce with Hadoop

Lets turn Hadoop back on and watch it work:

```
rmr.options(backend = "hadoop")
```

```
## NULL
```

```
groups_dfs = to.dfs(groups)

from.dfs(
  mapreduce(
    input = groups_dfs,
    map = function(k, v) {
      keyval(v, 1);
    },
    reduce =
      function(k, vv) {
        keyval(k, length(vv)) ;
      }
  )
)
```

```
## $key
##  [1]  9 10 11 12 13 14 15 16 17 18 19
##
## $val
##  [1] 4 7 6 6 4 9 8 1 2 2 1
```

NOTE: the output is sorted, proving it went to hadoop even though the output does not show here. Also note that hadoop has a high startup cost.

K-Means Example

Continuing with the example, we will perform K-Means clustering on a sample data set. First, lets grab the data:

```
green_taxi_data_csv <- read.csv("~/Code/Masters/IS622/Week2/green_tripdata_2015-01.trimmed.csv")
green_taxi_data <- as.matrix(green_taxi_data_csv[,c("Trip_distance","Fare_amount")])
```

Notice that I am trimming the data to the columns that I want to cluster. I did not choose the obvious thing to cluster, the pickup or dropoff locations, since they didn't seem to have much variation. This makes sense since the green taxis work in a much more limited area. I took inspiration from Rohan's code.

Without Hadoop

Now lets load it into the DFS, but lets stay local for now:

```
rmr.options(backend = "local")
```

```
## NULL
```

```
green_taxi_data_dfs_local <- to.dfs(green_taxi_data)
```

Next, we will setup our map-reduce job:

```
## @knitr kmeans-signature
kmeans.mr =
  function(
    P,
    num.clusters,
    num.iter,
    combine,
    in.memory.combine) {
## @knitr kmeans-dist.fun
    dist.fun =
      function(C, P) {
        apply(
          C,
          1,
          function(x)
            colSums((t(P) - x)^2))}
## @knitr kmeans.map
    kmeans.map =
      function(., P) {
        nearest = {
          if(is.null(C))
            sample(
              1:num.clusters,
              nrow(P),
              replace = TRUE)
          else {
            D = dist.fun(C, P)
            nearest = max.col(-D)}}
        if(!(combine || in.memory.combine))
          keyval(nearest, P)
        else
          keyval(nearest, cbind(1, P))}
## @knitr kmeans.reduce
    kmeans.reduce = {
      if (!(combine || in.memory.combine) )
        function(., P)
          t(as.matrix(apply(P, 2, mean)))
      else
        function(k, P)
          keyval(
            k,
            t(as.matrix(apply(P, 2, sum))))}
## @knitr kmeans-main-1
    C = NULL
```

```
    for(i in 1:num.iter ) {
      C =
        values(
          from.dfs(
            mapreduce(
              P,
              map = kmeans.map,
              reduce = kmeans.reduce)))
      if(combine || in.memory.combine)
        C = C[, -1]/C[, 1]
## @knitr end
#      points(C, col = i + 1, pch = 19)
## @knitr kmeans-main-2
      if(nrow(C) < num.clusters) {
        C =
          rbind(
            C,
            matrix(
              rnorm(
                (num.clusters -
                   nrow(C)) * nrow(C)),
              ncol = nrow(C)) %*% C) }}
        C}
## @knitr end
```

Lets run the map-reduce job and see the results:

```
kmeans.mr(
      green_taxi_data_dfs_local,
      num.clusters = 12,
      num.iter = 5,
      combine = FALSE,
      in.memory.combine = FALSE)
```

```
##       Trip_distance Fare_amount
##   [1,]    0.69201910    4.517481
##   [2,]    1.34685111    7.250133
##   [3,]    2.11229475   10.039581
##   [4,]    2.95879970   12.326141
##   [5,]    5.81493671   20.781148
##   [6,]    7.43016458   26.813754
##   [7,]   11.92375074   44.452382
##   [8,]    3.95792050   15.903539
##   [9,]   15.07699513    3.004392
##  [10,]    0.43619863  -10.416952
##  [11,]    0.10782609 -156.439565
##  [12,]    0.06053254  -52.023964
```

With Hadoop

Lets load the data now into HDFS:

```r
rmr.options(backend = "hadoop")
```

```
## NULL
```

```r
green_taxi_data_dfs <- to.dfs(green_taxi_data)
```

Lets re-run the map-reduce job and see the results:

```r
kmeans.mr(
      green_taxi_data_dfs,
      num.clusters = 12,
      num.iter = 5,
      combine = FALSE,
      in.memory.combine = FALSE)
```

```
##        Trip_distance Fare_amount
##   [1,]     1.8226316     8.70000
##   [2,]     2.8935000    13.20000
##   [3,]     0.9006169     5.28263
##   [4,]     3.4825000    12.75000
##   [5,]     3.4050000    11.93750
##   [6,]     1.4033333    12.25000
##   [7,]     5.2779268    19.27927
##   [8,]     3.7159302    14.69186
##   [9,]     8.9386486    31.32883
##  [10,]     0.7450000    16.62500
##  [11,]     2.5343750    10.92188
##  [12,]     2.8336000    12.18000
```