Welcome to Week 3 of the Big Data Capstone

This document provides a running example of completing the Week 3 assignment :

- A shorter version with fewer comments is available as script:
  sparkMLlibClustering.py
- To run these commands in Cloudera VM: first run the setup script:
  setupWeek3.sh
- You can then copy paste these commands in pySpark.
- To open pySpark, refer to : Week 2 and Week 4 of the Machine Learning course
- Note that your dataset may be different from what is used here, so your results may not match with those shown here

Finally, make sure that your working directory contains the data files (.csv) for the fastest support. We recommend workng in your home directory (type cd ~ in your terminal). Please run any scripts using your terminal for proper settings.

Note: this document has been written using R (with Knitr) as an example and verification of the techniques used in the class. I'm going to be using lubridate and ggplot2 libraries for generating this document.

Step 1: Attribute Selection

Import Data

First let us read the contents of the file adclicks. csv. The following commands read in the CSV file in a table format and removes any extra whitespaces. So, if the CSV contained 'userid' it becomes 'userid'.

Note that you must change the path to adclicks. csv to the location on your machine, if you want to run this command on your machine.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```
ad_clicks <- read.csv("ad-clicks.csv")

#time format is year-month-day: yyyy-mm-dd
#               hh:mm:ss where hh is 24 hour clock
#
# datetime is in UTC (GMT) so no local conversions
ad_clicks$timestamp <- ymd_hms(ad_clicks$timestamp)
```

Let us display the first 5 lines of adclicksDF:

```r
head(ad_clicks,5)
```

```
##             timestamp txId userSessionId teamId userId adId  adCategory
## 1 2016-05-26 15:13:22 5974          5809     27    611    2 electronics
## 2 2016-05-26 15:17:24 5976          5705     18   1874   21      movies
## 3 2016-05-26 15:22:52 5978          5791     53   2139   25   computers
## 4 2016-05-26 15:22:57 5973          5756     63    212   10     fashion
## 5 2016-05-26 15:22:58 5980          5920      9   1027   20    clothing
```

Next, We are going to add an extra column to the adclicks table and make it equal to 1. We do so to record the fact that each ROW is 1 adclick. You will see how this will become useful when we sum up this column to find how many ads did a user click.

```r
ad_clicks$adCount <- 1
```

Let us display the first 5 lines of adclicksDF and see if a new column has been added:

```r
head(ad_clicks,5)
```

```
##             timestamp txId userSessionId teamId userId adId  adCategory
## 1 2016-05-26 15:13:22 5974          5809     27    611    2 electronics
## 2 2016-05-26 15:17:24 5976          5705     18   1874   21      movies
## 3 2016-05-26 15:22:52 5978          5791     53   2139   25   computers
## 4 2016-05-26 15:22:57 5973          5756     63    212   10     fashion
## 5 2016-05-26 15:22:58 5980          5920      9   1027   20    clothing
##   adCount
## 1       1
## 2       1
## 3       1
## 4       1
## 5       1
```

Next, let us read the contents of the file buyclicks. csv. As before, the following commands read in the CSV file in a table format and removes any extra whitespaces. So, if the CSV contained 'userid' it becomes 'userid'. Note that you must change the path to buyclicks. csv to the location on your machine, if you want to run this command on your machine.

```r
buy_clicks <- read.csv("buy-clicks.csv")

buy_clicks$timestamp <- ymd_hms(buy_clicks$timestamp)
```

Let us display the first 5 lines of buyclicksDF:

```r
head(buy_clicks,5)
```

```
##             timestamp txId userSessionId team userId buyId price
## 1 2016-05-26 15:36:54 6004          5820    9   1300     2     3
## 2 2016-05-26 15:36:54 6005          5775   35    868     4    10
## 3 2016-05-26 15:36:54 6006          5679   97    819     5    20
## 4 2016-05-26 16:36:54 6067          5665   18    121     2     3
## 5 2016-05-26 17:06:54 6093          5709   11   2222     5    20
```

Feature Selection

For this exercise, we can choose from buyclicksDF, the 'price' of each app that a user purchases as an attribute that captures user's purchasing behavior. The following command selects 'userid' and 'price' and drops all other columns that we do not want to use at this stage.

```
userPurchases <- subset(buy_clicks,select=c("userId","price"))
head(userPurchases,5)
```

```
##   userId price
## 1   1300     3
## 2    868    10
## 3    819    20
## 4    121     3
## 5   2222    20
```

Similarly, from the adclicksDF, we will use the 'adCount' as an attribute that captures user's inclination to click on ads. The following command selects 'userid' and 'adCount' and drops all other columns that we do not want to use at this stage.

```
useradClicks <- subset(ad_clicks,select=c("userId","adCount"))
head(useradClicks,5)
```

```
##   userId adCount
## 1    611       1
## 2   1874       1
## 3   2139       1
## 4    212       1
## 5   1027       1
```

Step 2: Training Data Set Creation

Create the first aggregate feature for clustering

From each of these single adclicks per row, we can now generate total ad clicks per user. Let's pick a user with userid = 3. To find out how many ads this user has clicked overall, we have to find each row that contains userid = 3, and report the total number of such rows. The following commands sum the total number of ads per user and rename the columns to be called 'userid' and 'totalAdClicks'. **Note that you may not need to aggregate (e.g. sum over many rows) if you choose a different feature and your data set already provides the necessary information.** In the end, we want to get one row per user, if we are performing clustering over users.

```
adsPerUser <- aggregate(ad_clicks$adCount,by=list(userId=ad_clicks$userId),FUN=sum)
names(adsPerUser) <- c("userId","totalAdClicks")
```

Let us display the first 5 lines of 'adsPerUser' to see if there is a column named 'totalAdClicks' containing total adclicks per user.

```
head(adsPerUser,5)
```

```
##   userId totalAdClicks
## 1      1            44
## 2      8            10
## 3      9            37
## 4     10            19
## 5     12            46
```

Create the second aggregate feature for clustering

Similar to what we did for adclicks, here we find out how much money in total did each user spend on buying inapp purchases. As an example, let's pick a user with userid = 9. To find out the total money spent by this user, we have to find each row that contains userid = 9, and report the sum of the column'price' of each product they purchased. The following commands sum the total money spent by each user and rename the columns to be called 'userid' and 'revenue'.

Note: that you can also use other aggregates, such as sum of money spent on a specific ad category by a user or on a set of ad categories by each user, game clicks per hour by each user etc. You are free to use any mathematical operations on the fields provided in the CSV files when creating features.

```
revenuePerUser  <- aggregate(userPurchases$price,by=list(userId=userPurchases$userId),FUN=sum)
names(revenuePerUser) <- c("userId","revenue")

head(revenuePerUser,5)
```

```
##   userId revenue
## 1      1      21
## 2      8      53
## 3      9      80
## 4     10      11
## 5     12     215
```

Merge the two tables

Lets see what we have so far. We have a table called revenuePerUser, where each row contains total money a user (with that 'userid') has spent. We also have another table called adsPerUser where each row contains total number of ads a user has clicked. We will use revenuePerUser and adsPerUser as features / attributes to capture our users' behavior.

Let us combine these two attributes (features) so that each row contains both attributes per user. Let's merge these two tables to get one single table we can use for KMeans clustering.

```
combinedDF <- merge(adsPerUser,revenuePerUser, by="userId")
```

Let us display the first 5 lines of the merged table. **Note: Depending on what attributes you choose, you may not need to merge tables. You may get all your attributes from a single table.**

```
head(combinedDF,5)
```

```
##   userId totalAdClicks revenue
## 1      1            44      21
## 2      8            10      53
## 3      9            37      80
## 4     10            19      11
## 5     12            46     215
```

Create the final training dataset

Our training data set is almost ready. At this stage we can remove the 'userid' from each row, since 'userid' is a computer generated random number assigned to each user. It does not capture any behavioral aspect of a user. One way to drop the 'userid', is to select the other two columns.

```
trainingDF <- combinedDF[,-1]
head(trainingDF,5)
```

```
##   totalAdClicks revenue
## 1            44      21
## 2            10      53
## 3            37      80
## 4            19      11
## 5            46     215
```

Display the dimensions of the training dataset

Display the dimension of the training data set. To display the dimensions of the trainingDF, simply add .shape as a suffix and hit enter.

```
dim(trainingDF)
```

```
## [1] 543   2
```

The following two commands convert the tables we created into a format that can be understood by the KMeans.train function.

line[0] refers to the first column. line[1] refers to the second column. If you have more than 2 columns in your training table, modify this command by adding line[2], line[3], line[4] ...

```
#there is no need to perform this in R
```

Step 3: Train to Create Cluster Centers

Train KMeans model

Here we are creating two clusters as denoted in the second argument.

```
set.seed(20)
model <- kmeans(trainingDF, 2, nstart = 20)
```

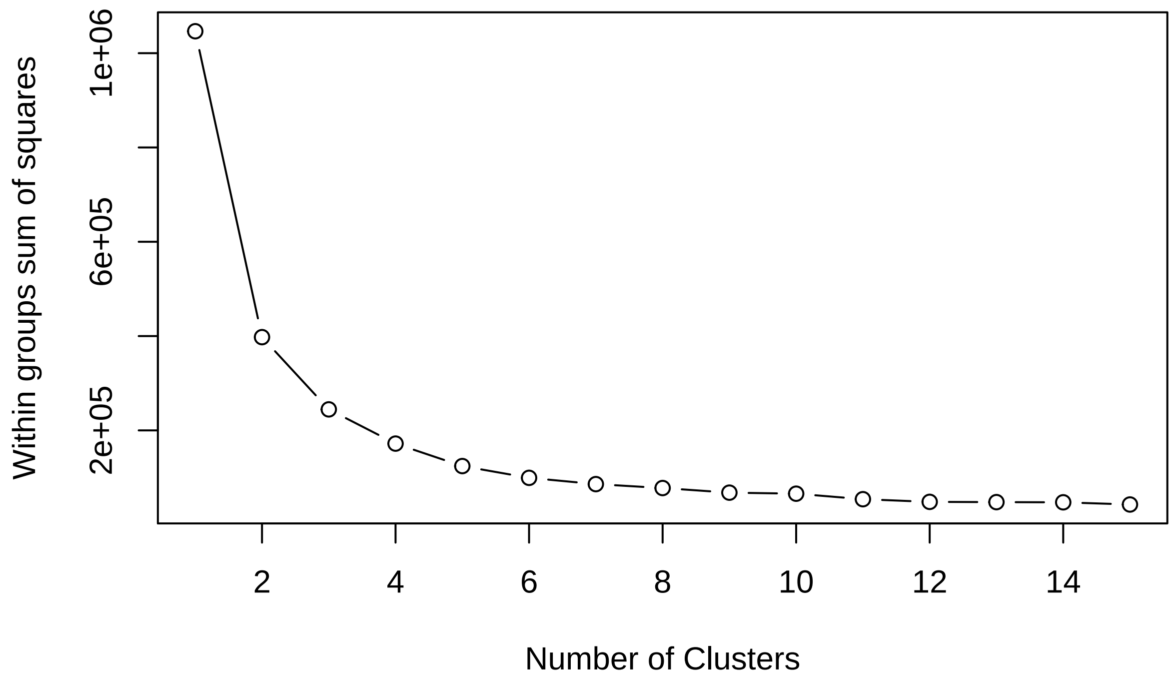
Display the centers of two clusters formed

```
model$centers
```

```
##   totalAdClicks   revenue
## 1      39.07609 115.26087
## 2      27.39468  23.86475
```

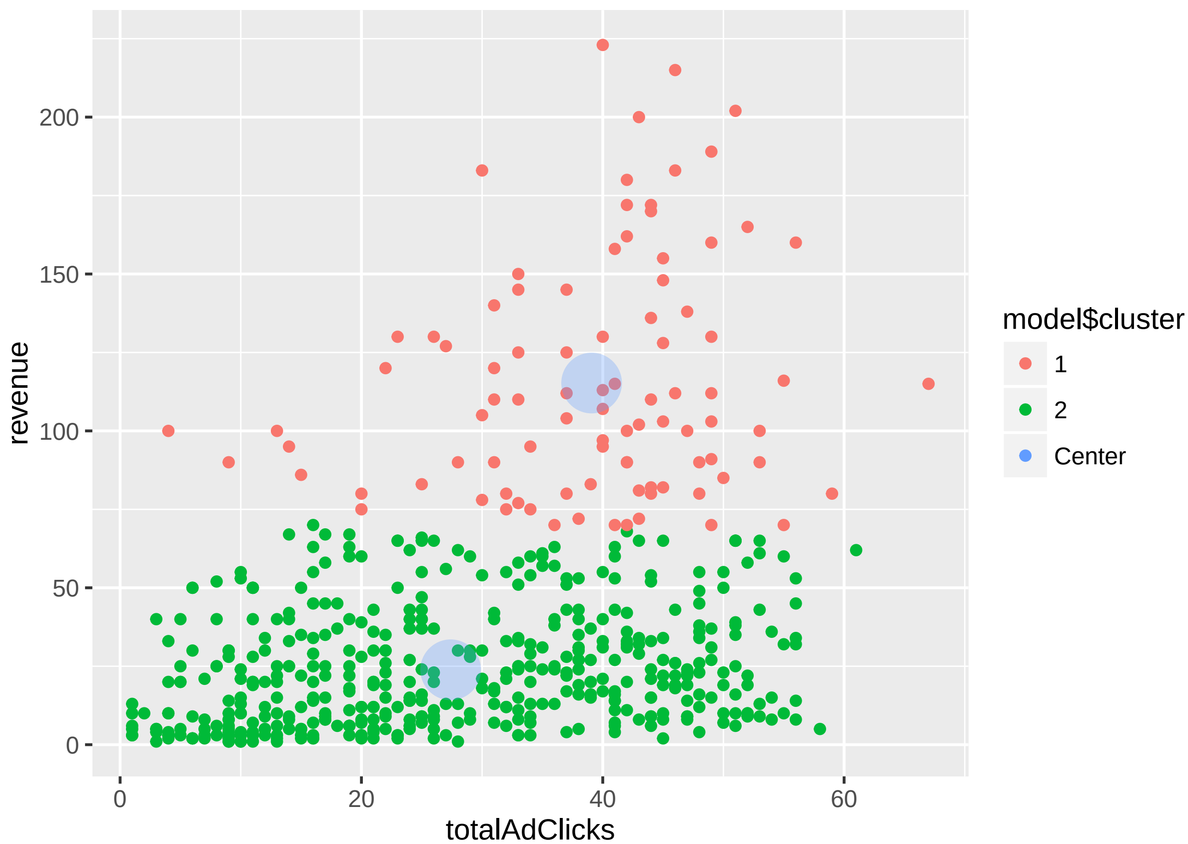
Display the elbow graph for cluster number selection

```
mydata <- trainingDF
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(mydata,
                                     centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares")
```



Display a graphic showing the centers:

```
library(ggplot2)
model$cluster <- as.factor(model$cluster)
ggplot(trainingDF, aes(totalAdClicks, revenue, color = model$cluster)) + geom_point() +
  geom_point(data=as.data.frame(model$centers), aes(color='Center'),size=10, alpha=.3, show.legend=FALSE
```



Step 4: Recommend Actions

Analyze the cluster centers

Each array denotes the center for a cluster:
One Cluster is centered at ... array([27.39467849, 23.86474501])
Other Cluster is centered at ... array([39.07608696, 115.26086957])

First number (field1) in each array refers to number of adclicks and the second number (field2) is the revenue per user. Compare the 1st number of each cluster to see how differently users in each cluster behave when it comes to clicking ads. Compare the 2nd number of each cluster to see how differently users in each cluster behave when it comes to buying stuff.

In one cluster, in general, players click on ads much more often (~1.4 times) and spend more money (~4.7 times) on inapp purchases. Assuming that Eglence Inc. gets paid for showing ads and for hosting inapp purchase items, we can use this information to increase game's revenue by increasing the prices for ads we show to the frequentclickers, and charge higher fees for hosting the inapp purchase items shown to the higher revenue generating buyers.

Note: This analysis requires you to compare the cluster centers and find any 'significant'differences in the corresponding feature values of the centers. The answer to this question will depend on the features you have chosen.

Some features help distinguish the clusters remarkably while others may not tell you much. At this point, if you don't find clear distinguishing patterns, perhaps rerunning the clustering model with different numbers of clusters and revising the features you picked would be a good idea.