IS622 - Weeks 8-10 Clustering Mini-Project (update)

Brian Chu | Nov 17, 2015

Edits:

- Reversed lat/long in the plots
- Scaled data before doing k-means
- Rescaled k-means center points to get results in original lat/long
- Cleaned US dataset (removed entries with bad GPS points, several were 0,0)

Dataset: Streaming RSVP data from Meetup.com
http://meetup.github.io/stream/rsvpTicker/

Note: I collected the data using command line curl and parsed the JSON to CSV in Python (meetup.py)

Each row of the dataset is a unique RSVP and contains the GPS coordinates, city, and country of the responder, as well as the group s/he is replying to. A sample is below:

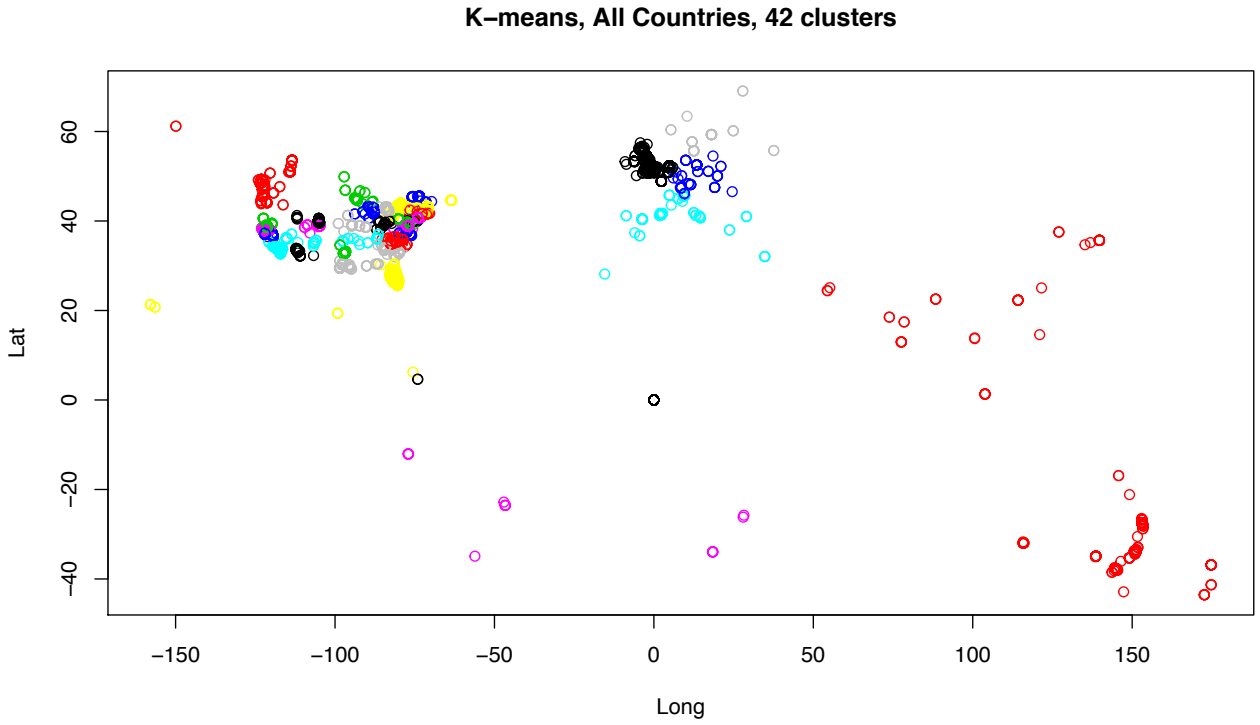
```
# Read raw data
raw <- read.csv("meetup.csv", header=TRUE)
head(raw)
```

```
##         lat       long        city country
## 1  42.34660  -71.10743      Boston      us
## 2  51.49460   -0.10047      London      gb
## 3 -33.76134  150.66611      Sydney      au
## 4  49.23258 -123.11660   Vancouver      ca
## 5  44.85652  -93.43449 Minneapolis      us
## 6  43.71788  -79.45847     Toronto      ca
##                                                group
## 1                 Greater Boston Permaculture Guild
## 2                          London Digital Analytics
## 3                            Sydney Kayaking Meetup
## 4               The Vancouver Mandarin Chinese Club
## 5 Twin Cities Professional Networking Event (TCPNE)
## 6                        Try New Things in Toronto!
```

Base R implementation with k-means clustering function

```
# Use lat/long data for clustering
gps <- as.matrix(raw[,c("long", "lat")], ncol=2)

# Scale/center data
```

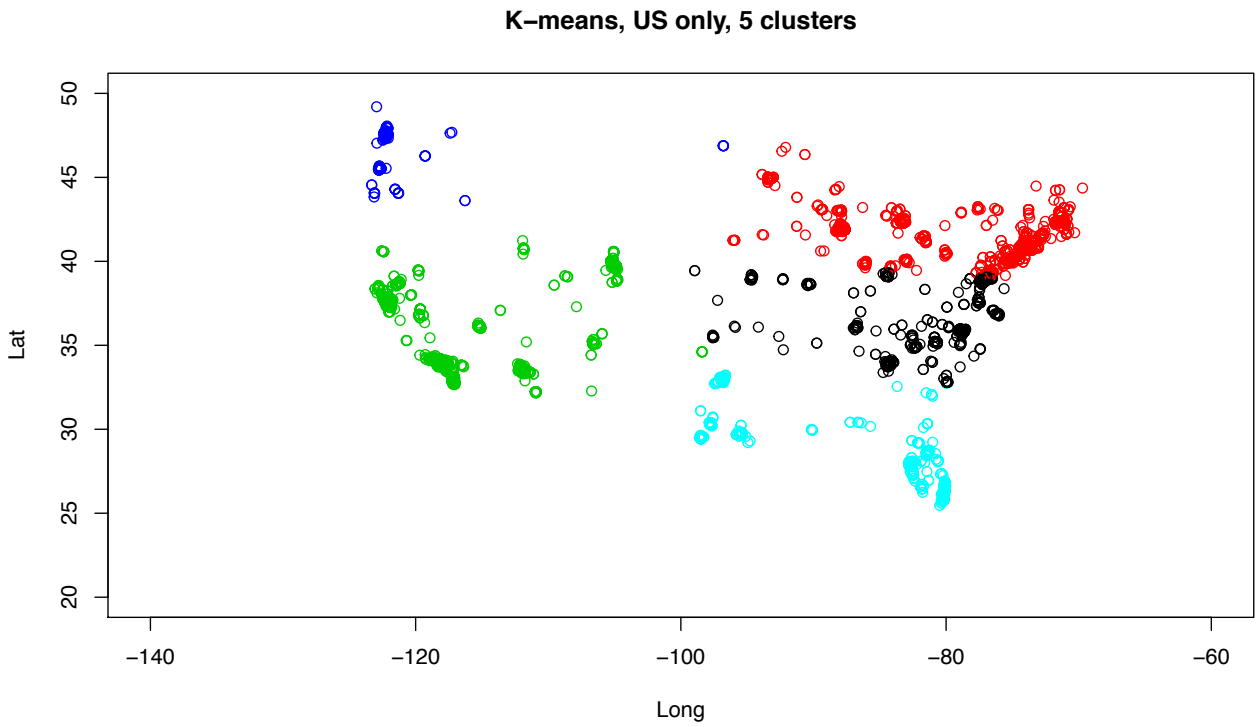
```r
gps_norm <- scale(gps, center=TRUE, scale=TRUE)

# Number of clusters (center points)
k <- length(unique(raw$country)) # 42 countries

# Fit data using k-means algorithm
set.seed(50)
kfit <- kmeans(gps_norm, centers=k, iter.max=10)

# Aggregate and plot results
meetup <- raw
meetup$cluster <- kfit$cluster
plot(meetup$long, meetup$lat, col=meetup$cluster,
     xlab="Long", ylab="Lat", main="K-means, All Countries, 42 clusters")
```



Using all 42 countries as separate reference clusters is quite messy, particularly around the US coordinates, which dominate the dataset. Let's look at the US only with just 5 clusters.

```r
# Just take US
rawUS <- subset(raw, country=="us")

# Remove bad entries
bad <- which(rawUS$long > -50)
rawUS <- rawUS[-bad,]

# Use lat/long data for clustering
gpsUS <- as.matrix(rawUS[,c("long", "lat")], ncol=2)

# Scale/center data
```

```
gpsUS_norm <- scale(gpsUS, center=TRUE, scale=TRUE)

# Number of clusters (center points)
k <- 5 # using US data only

# Fit data using k-means algorithm
set.seed(50)
kfitUS <- kmeans(gpsUS_norm, centers=k, iter.max=10)

# Aggregate and plot results
meetupUS <- rawUS
meetupUS$cluster <- kfitUS$cluster
plot(meetupUS$long, meetupUS$lat, col=meetupUS$cluster,
     ylim=c(20,50), xlim=c(-140,-60),
     ylab="Lat", xlab="Long", main="K-means, US only, 5 clusters")

# K-means center points - rescaled to original
library(DMwR)
```




```
z <- unscale(kfitUS$centers, gpsUS_norm)
print(z)
```

```
##         long      lat
## 1  -81.03124 37.00696
## 2  -78.37536 41.50019
## 3 -116.88956 36.13669
## 4 -122.00590 46.87503
## 5  -88.32646 29.04354
```

```
detach("package:DMwR", unload=TRUE)
```

The results looks much better now for the US-only dataset. The clusters are no longer vertical stripes due to normalizing the data to a [0,1] scale. The five clusters are generally the Northwest, West, South/Southeast, Mid-Atlantic, and Northeastregions of the US. As expected, there are denser points around larger populations hubs (NY, Fla, LA, SF). However, there are some exceptions such as fewer RSVPs from populous Texas and more from less populated Seattle. There is also an interesting blue outlier in the north center, which did not fall into the red cluster. Similarly for the green point near the middle, which seems closer to the black and blue clusters.

RHadoop using algorithm from Revolution Analytics

Source 1: Big Data Analytics with R and Hadoop, by Vignesh Prajapati
Source 2: Revolution Analytics

The outline of the k-means clustering algorithm is as follows:

- Calculate distance to prespecified number of center points (clusters)
- Mapper function - assign points to closest center point
- Reducer function - recalculate center point as mean of all data points assigned to cluster
- Repeat mapreduce for number of iterations specified

```
library(rhdfs)
library(rJava)
library(rmr2)
rmr.options(backend = "local") #to print PDF
hdfs.init()
```

```
kmeans.mr =
  function(
    P,
    num.clusters,
    num.iter,
    combine,
    in.memory.combine) {

    ## kmeans-dist.fun
    dist.fun =
      function(C, P) {
        apply(
          C,
          1,
          function(x)
            colSums((t(P) - x)^2))}

    ## 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))}

  ## 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))))}

  ## 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]

    ## kmeans-main-2
    if(nrow(C) < num.clusters) {
      C =
        rbind(
          C,
          matrix(
            rnorm(
              (num.clusters -
                 nrow(C)) * nrow(C)),
            ncol = nrow(C)) %*% C) }}
  C}
```

‘‘‘

Use function with Meetup GPS data (US-only)

```
# Load to DFS
gpsUS_dfs <- to.dfs(gpsUS_norm)

# Run k-means mapreduce
set.seed(50)
kfitRH <- kmeans.mr(gpsUS_dfs, num.clusters = 5, num.iter = 10,
         combine = FALSE, in.memory.combine = FALSE)

# Center points rescaled to original
library(DMwR)
zrh <- unscale(kfitRH, gpsUS_norm)
print(zrh)
```

```
##           long      lat
## [1,]  -78.8312 40.48948
## [2,] -116.7101 36.13424
## [3,]  -86.8982 30.41425
## [4,] -121.7534 46.74548
## [5,] -149.9083 61.19018
```

```
# Compare with base R clusters
r1 <- z[order(-z[,1]),]
r2 <- zrh[order(-zrh[,1]),]
all <- cbind(r1, r2)
colnames(all) <- c("R_long", "R_lat", "RH_long", "RH_lat")
print(all)
```

```
##       R_long    R_lat   RH_long   RH_lat
## 2  -78.37536 41.50019  -78.8312 40.48948
## 1  -81.03124 37.00696  -86.8982 30.41425
## 5  -88.32646 29.04354 -116.7101 36.13424
## 3 -116.88956 36.13669 -121.7534 46.74548
## 4 -122.00590 46.87503 -149.9083 61.19018
```

I wasn't quite able to modify the source code to plot the RHadoop clusters. By tabular comparison, they seem quite similar except Row 2 (R) and Row 5 (RH) are very distinct. This is because there is actually a few points from Hawaii, which RHadoop is calling its own cluster and R lumps with the green Western points. As we have learned with k-means, the dispersion of starting points/centers really dictates the rest of the algorithm (althought I did set the same seed value for both).

PySpark Clustering update

November 18, 2015

0.1 For nicer formatting view, please go to:

0.1.1 http://bit.ly/1HYPLjf

0.2 IS622: Week 10 Mini-Project (update)

0.2.1 Clustering with PySpark

0.2.2 Brian Chu | Nov. 18, 2015

This is the PySpark implementation of k-means clustering for Meetup.com RSVP location data

The dataset is the same streaming file used in Weeks 8 and 9.
It is unique RSVP information for Meetup.com based on GPS coordinates, city, and country of responder, and Meetup group name.
http://meetup.github.io/stream/rsvpTicker/

```
In [1]: import os
        import sys

        # Path for Spark source folder
        os.environ['SPARK_HOME']="/home/brian/workspace/cuny_msda_is622/spark-1.5.1-bin-hadoop2.6"

        # Append pyspark to Python Path
        sys.path.append("/home/brian/workspace/cuny_msda_is622/spark-1.5.1-bin-hadoop2.6/python/")

        # Append py4j to Python Path
        sys.path.append("/home/brian/workspace/cuny_msda_is622/spark-1.5.1-bin-hadoop2.6/python/lib/py4
```

```
In [2]: # Launch Spark
        execfile("/home/brian/workspace/cuny_msda_is622/spark-1.5.1-bin-hadoop2.6/python/pyspark/shell.
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.5.1
      /_/

Using Python version 2.7.10 (default, Oct 14 2015 16:09:02)
SparkContext available as sc, HiveContext available as sqlContext.
```

```
In [3]: # Load required packages
        from pyspark.sql import SQLContext
        import numpy as np
        import csv
```

```
        import dateutils
        import pyspark_csv as pycsv #https://github.com/seahboonsiew/pyspark-csv
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [4]: # Import data
        raw = sc.textFile("meetup.csv")
        type(raw)

        # Convert raw csv data to Spark dataframe
        sqlc = SQLContext(sc)
        meetup = pycsv.csvToDataFrame(sqlc, raw)
        meetup.show(5)
```

```
+----------+----------+-----------+-------+--------------------+
|       lat|      long|       city|country|               group|
+----------+----------+-----------+-------+--------------------+
| 42.346603| -71.10743|     Boston|     us|Greater Boston Pe...|
| 51.494598|  -0.10047|     London|     gb|London Digital An...|
|-33.761341|150.666107|     Sydney|     au|Sydney Kayaking M...|
| 49.232578| -123.1166|  Vancouver|     ca|The Vancouver Man...|
| 44.856516|-93.434491|Minneapolis|     us|Twin Cities Profe...|
+----------+----------+-----------+-------+--------------------+
only showing top 5 rows
```

```
In [5]: # Subset US and scale data in Pandas
        import pandas as pd
        from sklearn.preprocessing import scale

        pdf = meetup.toPandas()
        pdfUS = pdf[(pdf.country=="us") & (pdf.long < -50)]
        dfplot = sqlc.createDataFrame(pdfUS)

        pdfUS_norm = pdfUS
        pdfUS_norm.lat = scale(pdfUS_norm.iloc[:,0])
        pdfUS_norm.long = scale(pdfUS_norm.iloc[:,1])

        # Convert back to PySpark DF and subset GPS columns
        df_norm = sqlc.createDataFrame(pdfUS_norm)
        df_norm = df_norm["long", "lat"]
        df_norm.show(3)
```

```
+------------------+------------------+
|              long|               lat|
+------------------+------------------+
|  1.306523253301959|0.9473386444772237|
|0.15292857847310476|1.4593321656122435|
|-1.2013698374901778|-0.112316408258129|
+------------------+------------------+
only showing top 3 rows
```

0.2.3 PySpark MLlib package for k-means clustering

Modified source code from:
http://spark.apache.org/docs/latest/mllib-clustering.html

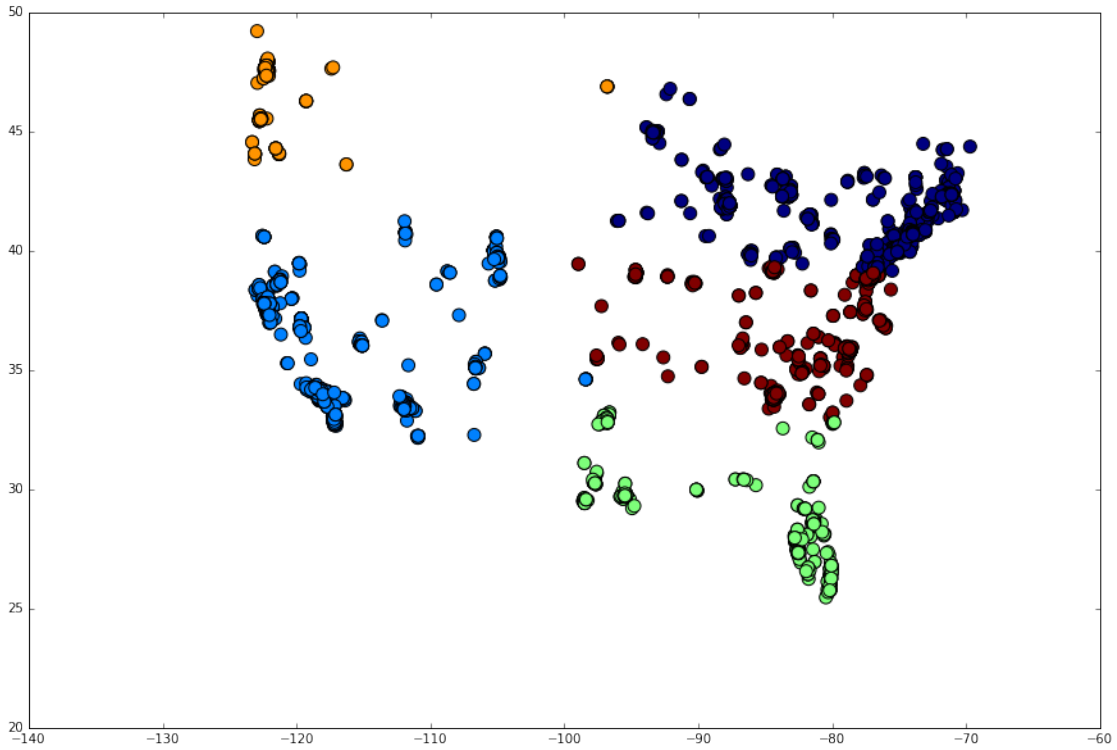
In [6]:
```
# Use PySpark MLlib package for k-means clustering

from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array

k = 5 # number of clusters (consistent with prior weeks)
dfk = df_norm.map(lambda row: array([float(x) for x in row]))
clusters = KMeans.train(dfk, k, maxIterations=10, seed=50)
```

The MLlib k-means function seems to assume labeled data in train/test format.
Since we just want to do unsupervised learning on our dataset, we 'predict' over the entire data.

In [7]:
```
# Sample first 20 rows
clust = clusters.predict(dfk).take(dfk.count())
```

In [8]:
```
# Plot clusters
import matplotlib.pyplot as plt
%matplotlib inline

sqlCtx.registerDataFrameAsTable(dfplot, "gps")
dfk2 = sqlCtx.sql("SELECT long, lat from gps")
plt.figure(figsize=(15,10))
r = [(x,y) for x,y in dfk2.take(dfk2.count())]
x1 = [x for (x,y) in r]
y1 = [y for (x,y) in r]
plt.xlim([-140,-60])
plt.ylim([20,50])
plt.scatter(x1,y1, c=clust, s=100)
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x7f1211f2b7d0>
```



0.2.4 Examine center points

```
In [9]: # Convert back to original scale
        from sklearn.preprocessing import StandardScaler
        import warnings
        warnings.filterwarnings('ignore')

        pdfplot = pdf[(pdf.country=="us") & (pdf.long < -50)]
        [x for x in pdfplot[["long", "lat"]]]
        sslat = StandardScaler().fit(pdfplot["lat"])
        sslong = StandardScaler().fit(pdfplot["long"])

        # Put into Pandas DF
        cc = pd.DataFrame(clusters.centers)
        cc.columns = ["long", "lat"]
        cc.long = sslong.inverse_transform(cc.iloc[:,0])
        cc.lat = sslat.inverse_transform(cc.iloc[:,1])

In [10]: cc.sort("long", ascending=False)

Out[10]:          long        lat
         0  -78.381653  41.509402
         4  -81.015102  37.073376
         2  -88.187754  29.105460
         1 -116.889558  36.136694
         3 -122.005904  46.875027
```

The cluster center points are very close to the R k-means results than the RHadoop ones. Basically, PySpark MLlib is also not classifying Hawaii points as their own cluster, unlike the RHadoop implementation shown earlier.