# Week10 - Clustering With Spark

November 15, 2015

# 1 Week 10 - Clustering with Spark

Assignment is presented in three sections. The first section shows how to do a basic job of coutning primes using Spark and Python. I used this link as a guide on how to setup PySpark and IPython notebook integration, and for the code regarding primes.

The second section, which is now snipped, was a basic K-Means clustering example. The code here is from the example on this page.

The last section basically runs the K-Means algorithm on the data from the previous weeks and presents results as compared to R.

## 1.1 Intro and Basic Setup

```
In [ ]: # Create PySpark context
        from pyspark import  SparkContext
        sc = SparkContext( 'local', 'pyspark')
```

```
In [19]: # Import Kmeans from MLLib and other useful packages
         from pyspark.mllib.clustering import KMeans, KMeansModel
         import numpy as np
         from math import sqrt
         import pandas as pd
```

## 1.2 K-Means for Mini Project

Lets do the above but for the data from previous weeks

```
In [2]: # Load and parse the data (make sure to convert to floats and create numpy arrays)
        mini_project_data = sc.textFile('data.csv') \
                                 .map(lambda line: np.array(map(lambda col: float(col), line.split(',')))

        # Show data (only first 10 rows)
        mini_project_data.collect()[0:10]
```

```
Out[2]: [array([ 0.,   0.,   0.,   0.,   0.]),
         array([ 0.08,  0.08,  0.1 ,  0.24,  0.9 ]),
         array([ 0.06,  0.06,  0.05,  0.25,  0.33]),
         array([ 0.1 ,  0.1 ,  0.15,  0.65,  0.3 ]),
         array([ 0.08,  0.08,  0.08,  0.98,  0.24]),
         array([ 0.09,  0.15,  0.4 ,  0.1 ,  0.66]),
         array([ 0.1 ,  0.1 ,  0.43,  0.29,  0.56]),
         array([ 0.15,  0.02,  0.34,  0.4 ,  0.01]),
         array([ 0.2 ,  0.14,  0.35,  0.72,  0.25]),
         array([ 0.  ,  0.  ,  0.5 ,  0.2 ,  0.85])]
```

In [6]:
```python
# Number of clusters to find
num_clusters = 10

# Build the model (cluster the data)
mini_project_clusters = KMeans.train(mini_project_data, num_clusters, \
                                      maxIterations=10, runs=10, initializationMode="random")
```

In [7]:
```python
# Show some details of results
#show_clusters(mini_project_clusters)
mini_project_week10_clusters = pd.DataFrame(mini_project_clusters.centers)
mini_project_week10_clusters.sort([0])
#mini_project_week10_clusters
```

Out[7]:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 6 | 0.265385 | 0.209115 | 0.301538 | 0.673846 | 0.190769 |
| 8 | 0.281429 | 0.237714 | 0.178929 | 0.290357 | 0.222500 |
| 0 | 0.337136 | 0.211636 | 0.656364 | 0.275000 | 0.228636 |
| 2 | 0.346067 | 0.387467 | 0.668333 | 0.700000 | 0.228000 |
| 4 | 0.351885 | 0.688077 | 0.608462 | 0.238077 | 0.658577 |
| 7 | 0.352500 | 0.609375 | 0.290000 | 0.563750 | 0.263750 |
| 5 | 0.360282 | 0.312051 | 0.268077 | 0.245897 | 0.700000 |
| 9 | 0.376463 | 0.264683 | 0.690976 | 0.266829 | 0.656829 |
| 3 | 0.571222 | 0.491444 | 0.666111 | 0.820000 | 0.683889 |
| 1 | 0.718333 | 0.323583 | 0.180833 | 0.703333 | 0.613333 |

Lets compare this to the R output from Week9:

In [110]:
```python
import pandas as pd
mini_project_week9_clusters = pd.DataFrame.from_csv("cluster_centers_week9.csv")
mini_project_week9_clusters
```

Out[110]:

| | X0 | X0.1 | X0.2 | X0.3 | X0.4 |
|---|---|---|---|---|---|
| 1 | 0.200000 | 0.221300 | 0.620500 | 0.414000 | 0.150000 |
| 2 | 0.207593 | 0.252593 | 0.638148 | 0.307407 | 0.716296 |
| 3 | 0.256842 | 0.177632 | 0.341579 | 0.751579 | 0.244211 |
| 4 | 0.272735 | 0.512324 | 0.315588 | 0.565294 | 0.227353 |
| 5 | 0.337025 | 0.285325 | 0.213875 | 0.234250 | 0.626500 |
| 6 | 0.373586 | 0.671724 | 0.575172 | 0.242759 | 0.669759 |
| 7 | 0.470348 | 0.333739 | 0.754783 | 0.761739 | 0.454783 |
| 8 | 0.527621 | 0.250759 | 0.675862 | 0.217931 | 0.480000 |
| 9 | 0.545773 | 0.248545 | 0.278636 | 0.453182 | 0.248182 |
| 10 | 0.682643 | 0.601071 | 0.397143 | 0.810000 | 0.662143 |

Old notes: We get somewhat consistent results. If we look at the clusters side by side, we can see the first row of week10's clusters maps to week9's second row, and vice versa. The 8th cluster (index 0) in week 10 maps to the 8th cluster (index 8) as well. The 7th row (index 2) in week 10 matches to row 6 in week9. Some of the others are harder to match. The fact that there doesn't seem to be more of an alignment (unless I am missing something) suggests that maybe I should reduce k in both situations, as the other clusters might be 'superfluous'

After rerunning the example to trim the output as suggested, the output is now different, so the above analysis isn't quite right anymore. I am going to re-run this with K set to 4, so see if there is any agreement, as there does not seem to be agreement here:

In [8]:
```python
# Number of clusters to find
num_clusters = 4
```

```
# Build the model (cluster the data)
mini_project_clusters2 = KMeans.train(mini_project_data, num_clusters, \
                                      maxIterations=10, runs=10, initializationMode="random")
mini_project_week10_clusters2 = pd.DataFrame(mini_project_clusters2.centers)
mini_project_week10_clusters2.sort([0])
```

Out[8]:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 1 | 0.301786 | 0.333798 | 0.442024 | 0.620000 | 0.198095 |
| 2 | 0.344672 | 0.327269 | 0.236045 | 0.246269 | 0.561642 |
| 0 | 0.369811 | 0.369486 | 0.689865 | 0.245946 | 0.562608 |
| 3 | 0.604455 | 0.438061 | 0.507576 | 0.753333 | 0.678788 |

I re-ran the R code with k = 4 with the following results:

setwd("Code/Masters/IS622/Week9/")

clusterdata <- read.csv('data.csv')

num_clusters <- 4;

library(stats)

model.kmeans.builtin <- kmeans(clusterdata, num_clusters)

model.kmeans.builtin*centers*[*order*(*model*.*kmeans*.*builtin*centers[,1]),]

```
X0       X0.1      X0.2      X0.3      X0.4
```

4 0.3089231 0.2886154 0.2755128 0.4776923 0.2256410 2 0.3497324 0.3479155 0.6888732 0.2388732 0.5717324 3 0.4182963 0.4236111 0.6412963 0.7505556 0.3875926 1 0.4489074 0.4013889 0.2910185 0.3127778 0.7255556

Still does not seem to line up too well, which makes me wonder about the accuracy of the previous R way (using custom K means in Hadoop), versus using a pre-built and hopefully battle-testing implementation in Spark.

In [58]:
```
def dist(x,y):
    return np.sqrt(np.sum((x - y)**2))

clusters = []
mini_project_data_df = mini_project_data.collect()
for row in mini_project_data_df:
    distances = [ ]
    distances.append( dist(mini_project_week10_clusters2.iloc[0], row) )
    distances.append( dist(mini_project_week10_clusters2.iloc[1], row) )
    distances.append( dist(mini_project_week10_clusters2.iloc[2], row) )
    distances.append( dist(mini_project_week10_clusters2.iloc[3], row) )
    #print "Vector %s assigned to cluster idx %d" % (row , np.argmin(distances))
    clusters.append(np.argmin(distances))

print("Number of points in cluster idx 0", len([ c for c in clusters if c == 0]))
print("Number of points in cluster idx 1", len([ c for c in clusters if c == 1]))
print("Number of points in cluster idx 2", len([ c for c in clusters if c == 2]))
print("Number of points in cluster idx 3", len([ c for c in clusters if c == 3]))
```

```
('Number of points in cluster idx 0', 74)
('Number of points in cluster idx 1', 84)
('Number of points in cluster idx 2', 67)
('Number of points in cluster idx 3', 33)
```

It seems that the clusters are close to being in size, except for the last index. This might suggest that 3 might be a better choice. As discussed in the lecture, there is a method to find an optimal k based on repeated clustering, and this approach could help fine tune k.

Also, I would perform PCA to help reduce the dimensions to 2 or 3 to help with plotting, but I am running out of time for this weeks work

```
In [111]: sc.stop()
```