Lab 9

Big Data, Spring 2017

April 3, 2017

Today's Lab

- Outlier Detection with Spark MLLIB, SparkSQL
- Application: Traffic Sensors
- The data: 9 columns
    - highway (int)
    - sensorloc (int)
    - sensorid (int)
    - dayofyear (int)
    - dayofweek (int)
    - time (float) (minutes since midnight)
    - volume (int)
    - speed (int)
    - occupancy (int)
- Data source/lab adapted from: https://aws.amazon.com/blogs/big-data/anomaly-detection-using-pyspark-hive-and-hue-on-amazon-emr/

More on Traffic Sensor Data

- We will use the three measurements given, volume, speed, and occupancy to try and find outliers
- volume: number of vehicles detected during reading
- speed: average speed of detected vehicles during reading
- occupancy: percentage of time during the reading that a vehicle was under the sensor

- e.g.:
    - congested traffic: low volume, low speed, high occupancy
    - heavy but flowing traffic: high volume, high speed, moderate occupancy
    - light/no traffic: low volume, high speed, low occupancy

Setup on Dumbo

- Login to dumbo
- Set up environment:

```
module load python/gnu/3.4.4
export PYSPARK_PYTHON=/share/apps/python/3.4.4/bin/python
export PYTHONHASHSEED=0
export SPARK_YARN_USER_ENV=PYTHONHASHSEED=0
```

- Start pyspark:

```
pyspark2
```

- Import the required modules:

```
import numpy as np
from math import sqrt
from operator import add
from pyspark.mllib.clustering import KMeans, KMeansModel
```

- Read in the data file (already on HDFS)

```
csvfile = sc.textFile('/user/ecc290/lab9/sensordatasmall/part-00000')
sensordata = csvfile.map(lambda line: line.split(','))
```

A little bit of data cleaning

- We want to cluster different types of traffic, so we will exclude entries where volume, speed, and occupancy are all 0.

```
sdfilt = sensordata.filter(lambda x:
np.count_nonzero(np.array([int(x[6]), int(x[7]),
int(x[8])]))>0)
```

```
sdfilt.count()
```

(Note the original raw data had 50,000 entries)

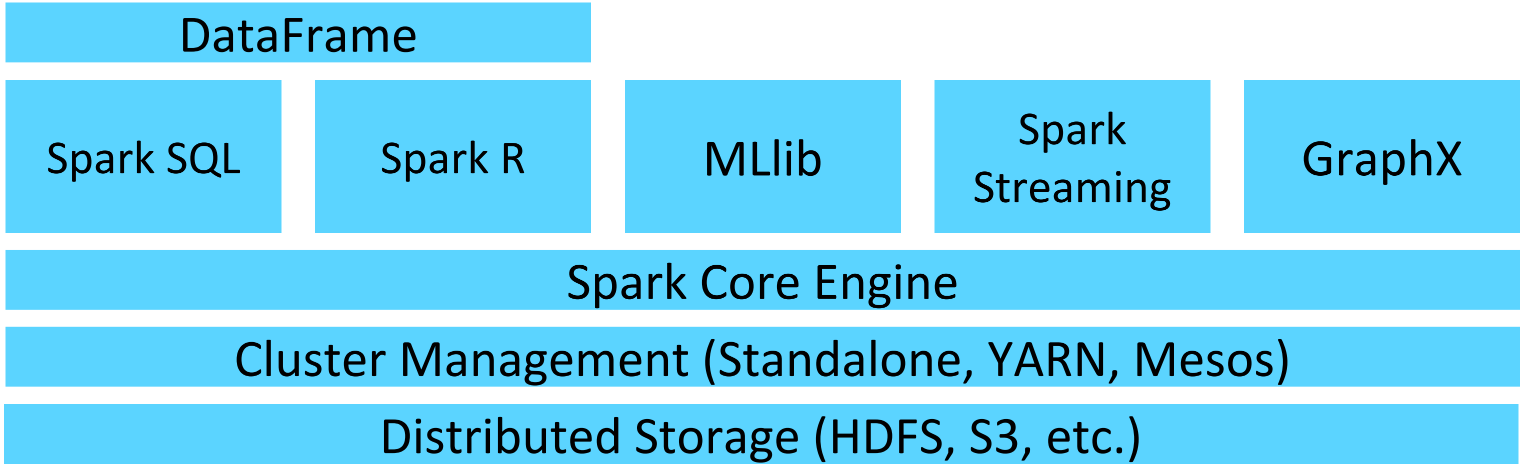
Stripping the vol, speed, occ columns

Filter out just 3 columns of measurements so we can run the clustering algorithm:

```
vso = sdfilt.map(lambda x: np.array([int(x[6]),
int(x[7]), int(x[8])]))
```

Spark's MLlib

- MLlib is Spark's machine learning library
- Goal is to make practical machine learning scalable and easy
- Includes common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, lower-level optimization primitives and higher-level pipeline APIs



k-means Clustering

- *k*-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean

- Formally:

Given set of observations $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where each observation is a d-dimensional vector.
kmeans: partition the n observations into k sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$
to minimize the within-cluster sum of squares, i.e., find

$$argmin_S \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of points in S_i

- An NP-Hard problem
  - Solved using heuristic algorithms + some iterative refinement
  - To learn more: https://en.wikipedia.org/wiki/K-means_clustering#Algorithms

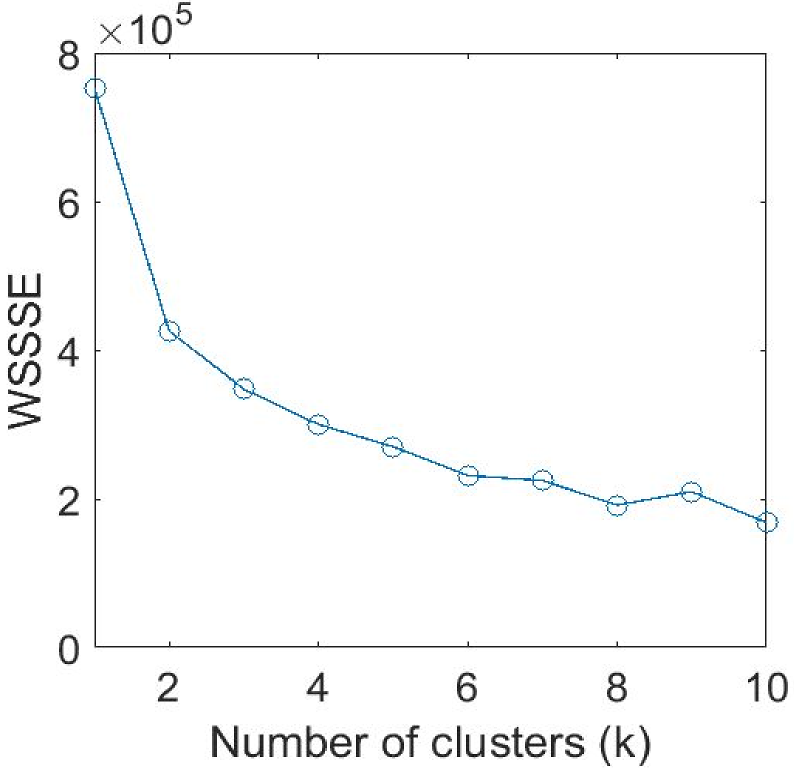
Finding the best k

- WSSSE measurement: “Within Set Sum of Squared Error”
    - sum of the distances from cluster center of each observation in each cluster/partition

```
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))
```

- Let’s try k=1:10

```
for i in range(1,11):
        clusters = KMeans.train(vso, i, maxIterations=10, initializationMode="random")
        WSSSE = vso.map(lambda point: error(point)).reduce(add)
        print("Within Set Sum of Squared Error, k = " + str(i) + ": "  + str(WSSSE))
```



Want k to be in the “knee” of the graph - so maybe 3-6 would be the rest choices here

- Where are the cluster centers? For k=3:

```
clusters = KMeans.train(vso, 3, maxIterations=10, initializationMode="random")
for i in range(0,len(clusters.centers)):
    print("cluster " + str(i) + ": " + str(clusters.centers[i]))
```

- For k=4:

```
clusters = KMeans.train(vso, 4, maxIterations=10, initializationMode="random")
for i in range(0,len(clusters.centers)):
    print("cluster " + str(i) + ": " + str(clusters.centers[i]))
```

Think about what types of traffic the different clusters represent for both k’s

Add cluster columns to RDD

```
def addclustercols(x):
    point = np.array([float(x[6]), float(x[7]), float(x[8])])
    center = clusters.centers[0]
    mindist = sqrt(sum([y**2 for y in (point - center)]))
    cl = 0
    for i in range(1,len(clusters.centers)):
        center = clusters.centers[i]
        distance = sqrt(sum([y**2 for y in (point - center)]))
        if distance < mindist:
            cl = i
            mindist = distance
    clcenter = clusters.centers[cl]
    return (int(x[0]), int(x[1]), int(x[2]), int(x[3]), int(x[4]), float(x[5]),
int(x[6]), int(x[7]), int(x[8]), int(cl), float(clcenter[0]), float(clcenter[1]),
float(clcenter[2]), float(mindist))
```

Append columns to table: For each observation, note which cluster it is closest to and its distance from the center

```
rdd_w_clusts = sdfilt.map(lambda x: addclustercols(x))
rdd_w_clusts.map(lambda y: (y[9],1)).reduceByKey(add).top(len(clusters.centers))
```

Map using the function defined above; mapreduce to get count of number of points in each cluster

Detecting Outliers

- If a point is far away from the center of the closest cluster, then we can consider it to be an outlier
- How to determine the cutoff distance?
- Let’s use sparkSQL to look at some statistics for the cluster

```
schema_sd = spark.createDataFrame(rdd_w_clusts, ('highway','sensorloc',
'sensorid', 'doy', 'dow', 'time','p_v','p_s','p_o', 'cluster', 'c_v',
'c_s', 'c_o', 'dist'))
```

```
schema_sd.createOrReplaceTempView("sd")
```

Convert RDD to DataFrame so we can run SQL queries

SQL Queries in Spark

```
spark.sql("SELECT max(dist) FROM sd").show()
```

Can run SQL queries on the table

```
stats = spark.sql("SELECT cluster, c_v, c_s, c_o, count(*) AS num,
max(dist) AS maxdist, avg(dist) AS avgdist,stddev_pop(dist) AS stdev
FROM sd GROUP BY cluster, c_v, c_s, c_o ORDER BY cluster")
```

```
stats.show()
```

Might want to look at various statistics about the clusters

```
def inclust(x, t):
     cl = x[9]
     c_v = x[10]
     c_s = x[11]
     c_o = x[12]
     distance = x[13]
     if float(distance) > float(t):
          cl = -1
          c_v = 0.0
          c_s = 0.0
          c_o = 0.0
     return (int(x[0]), int(x[1]), int(x[2]), int(x[3]), int(x[4]), float(x[5]),
int(x[6]), int(x[7]), int(x[8]), int(cl), float(c_v), float(c_s), float(c_o),
float(distance))
```

Function that sets cluster to -1 and cluster center to (0,0,0) for points greater than t away from center of closest cluster

```
rdd_w_clusts_wnullclust = rdd_w_clusts.map(lambda x: inclust(x,20))

rdd_w_clusts_wnullclust.map(lambda y: (y[9],1)).reduceByKey(add).top(5)
```

Map using function on previous slide; now get cluster counts again

```
schema_sd = spark.createDataFrame(rdd_w_clusts_wnullclust, ('highway','sensorloc',
'sensorid', 'doy', 'dow', 'time','p_v','p_s','p_o', 'cluster', 'c_v','c_s','c_o','dist'))

schema_sd.createOrReplaceTempView("sd_nc")
```

Turn this table with outliers set to cluster -1 into a DataFrame

```
spark.sql("SELECT p_v, p_s, p_o FROM sd_nc WHERE cluster=-1 LIMIT 100").show(100)
```

List outliers so we can inspect them

```
spark.sql("SELECT sensorid, cluster, count(*) AS num_outliers, avg(c_s) AS spdcntr,
avg(dist) AS avgdist FROM sd WHERE  dist > 20 GROUP BY sensorid, cluster ORDER BY
sensorid, cluster").show()
```

Might want to group by sensorid and cluster, so we can see if there is a particular sensor or cluster that has many outliers

```
spark.sql("SELECT cluster, doy, time, c_v,c_s,c_o, p_v,p_s,p_o FROM sd WHERE
cluster=<insert-clust-id-here> and dist >20 ORDER BY dist").show()
```

If there is a cluster with many outliers, we might want to inspect it more closely

Maybe we decide that we should try using 5 clusters. Rerun commands on previous slides, but now using 5 clusters:

```
clusters = KMeans.train(vso, 5, maxIterations=10, initializationMode="random")
rdd_w_clustsk5 = sdfilt.map(lambda x: addclustercols(x))
schema_sd = spark.createDataFrame(rdd_w_clustsk5, ('highway','sensorloc',
'sensorid', 'doy', 'dow', 'time', 'p_v', 'p_s', 'p_o', 'cluster', 'c_v', 'c_s',
'c_o', 'dist'))
schema_sd.createOrReplaceTempView("sdk5")
```

```
spark.sql("SELECT cluster, c_v, c_s, c_o, count(*) AS num, max(dist) AS maxdist,
avg(dist) AS avgdist,stddev_pop(dist) AS stdev FROM sdk5 GROUP BY cluster, c_v, c_s,
c_o ORDER BY cluster").show()
rdd_w_clusts_wnullclustk5 = rdd_w_clustsk5.map(lambda x: inclust(x,20))
rdd_w_clusts_wnullclustk5.map(lambda y: (y[9],1)).reduceByKey(add).top(6)
```

```
spark.sql("SELECT sensorid, cluster, count(*) AS num_outliers, avg(c_s) AS spdcntr,
avg(dist) AS avgdist FROM sdk5 WHERE  dist > 20 GROUP BY sensorid, cluster ORDER BY
sensorid, cluster").show()
```

```
spark.sql("SELECT cluster, doy, time, c_v,c_s,c_o, p_v,p_s,p_o FROM sdk5 WHERE
cluster=<insert-clust-id-here> and dist >20 ORDER BY dist").show()
```

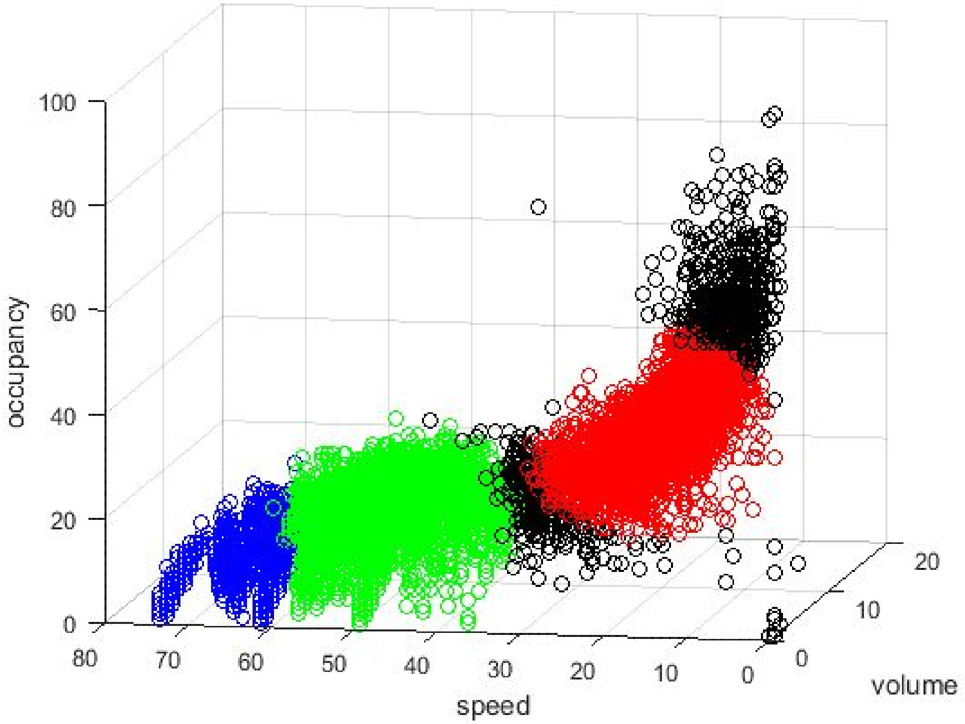
Writing a DataFrame to CSV File

Once you have assigned points to clusters and picked cutoff to define outliers, you may want to plot the data and see how it looks. Here is code that outputs a | delimited file with vol, speed, occ, and cluster number for each point. You can then plot this using your favorite plotting tool.

```
schema_sd = spark.createDataFrame(rdd_w_clusts_wnullclustk5, ('highway','sensorloc',
'sensorid', 'doy', 'dow', 'time','p_v','p_s','p_o', 'cluster',
'c_v','c_s','c_o','dist'))

schema_sd.createOrReplaceTempView("sdk5nc")


cdata=spark.sql("SELECT cluster, p_v, p_s, p_o FROM sdk5nc  ORDER BY cluster")


cdata.repartition(1).write.csv("k5clusts.csv", sep='|')
```

k=3, using t=20

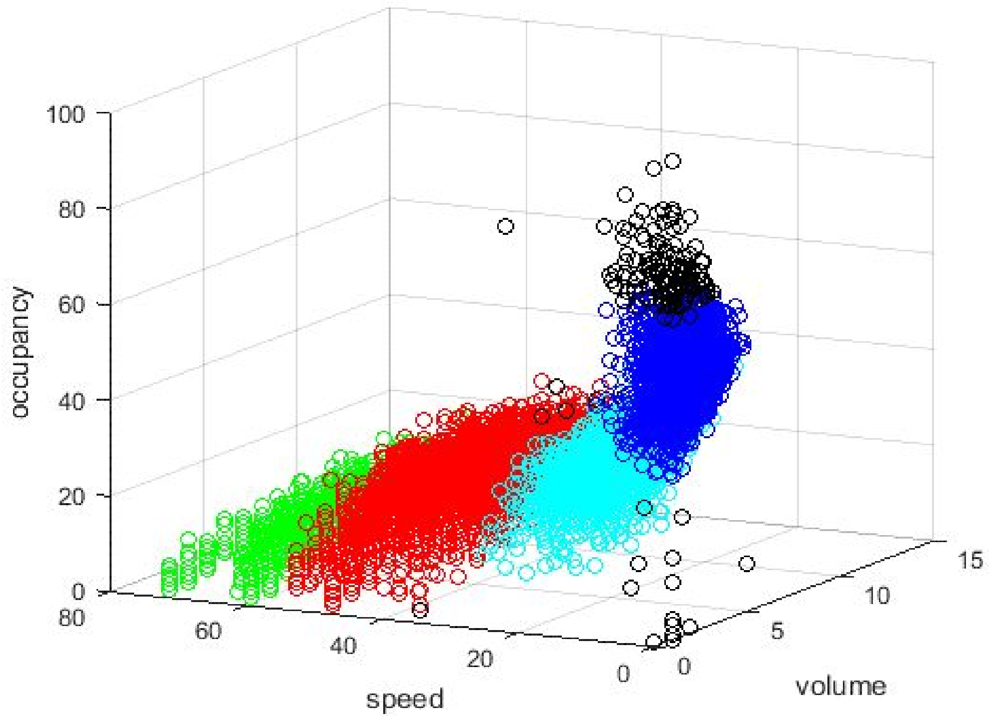
black dots = outliers
(i.e., cluster=-1)



*plots made
in matlab

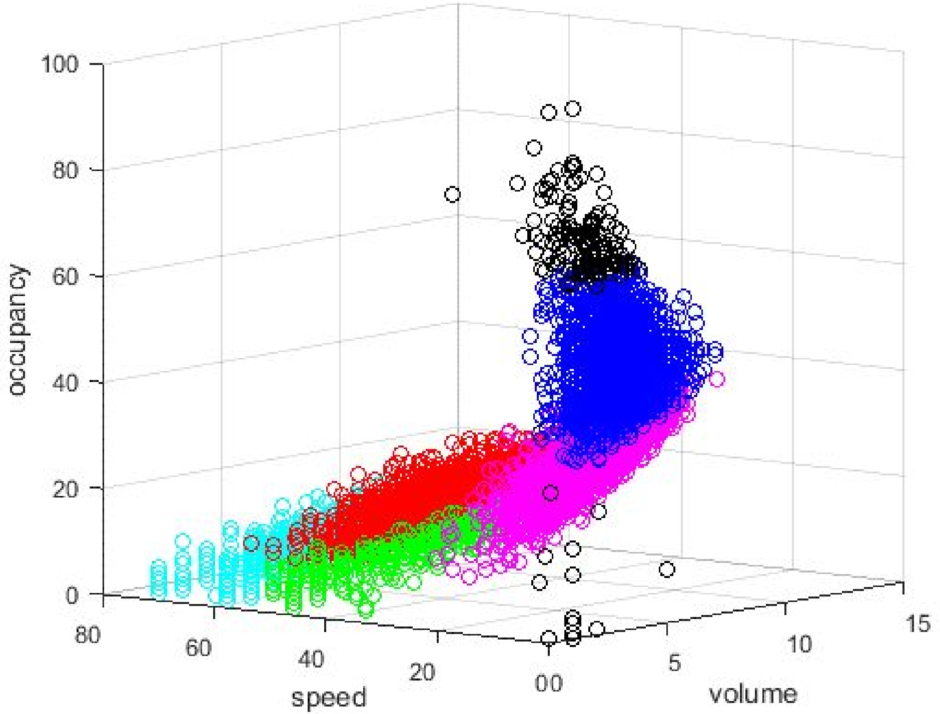
k=4, using t=20

black dots = outliers
(i.e., cluster=-1)



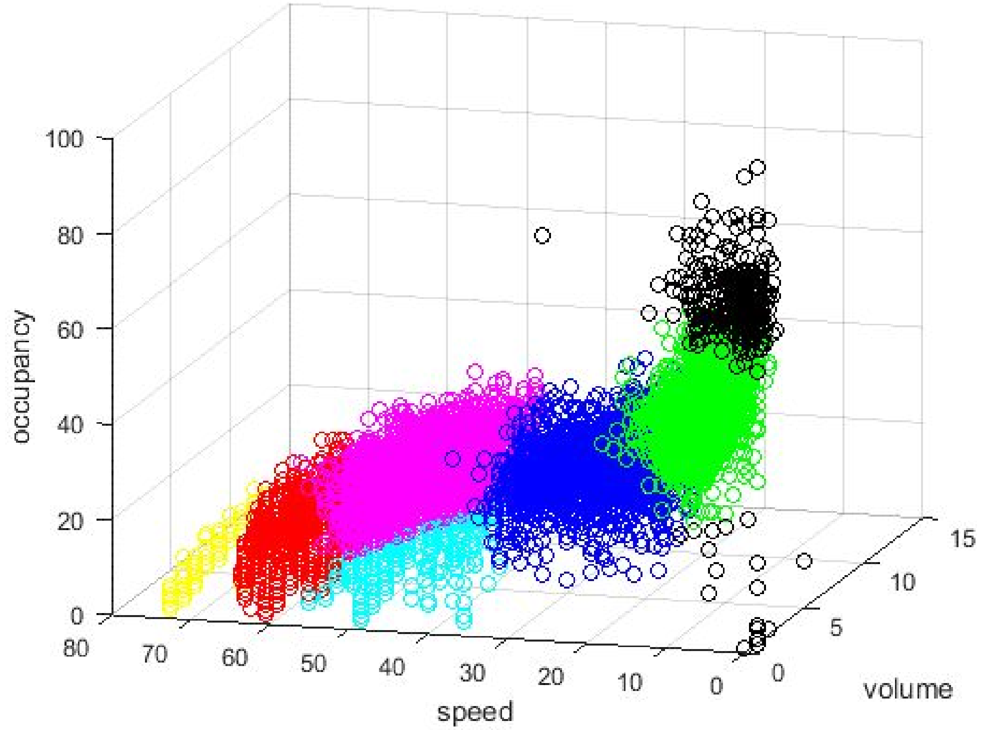
k=5, using t=20

black dots = outliers
(i.e., cluster=-1)



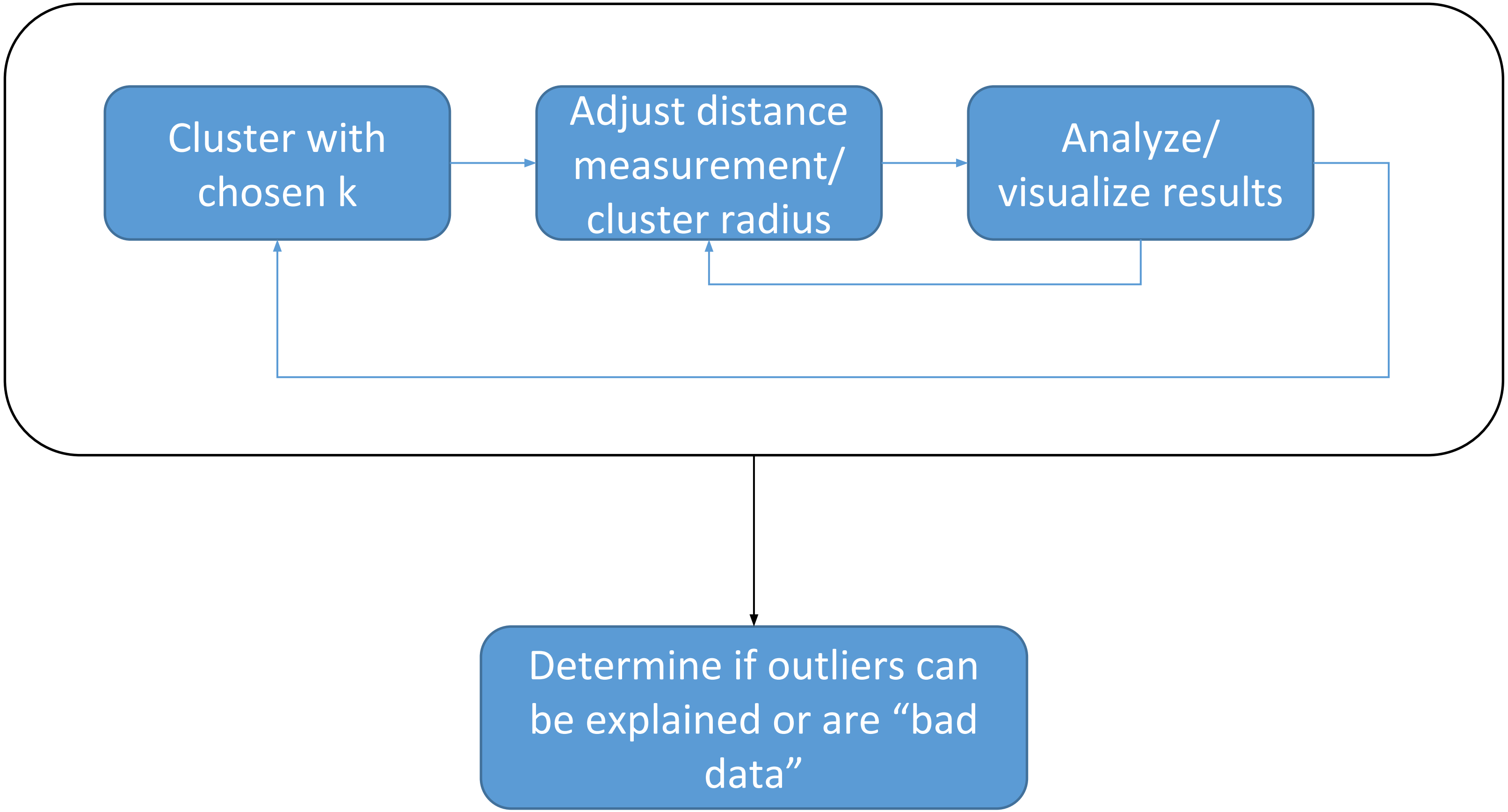
k=6, using t=20

black dots = outliers
(i.e., cluster=-1)



Better Distance Metrics

- In our distance calculation, the volume, speed, and occupancy axes have different units/ranges
- Might want use a normalized or weighted distance metric to improve clustering
  - e.g., Mahalanobis distance (unitless and scale-invariant, and takes into account the correlations of the data set).


Cluster with chosen k
Adjust distance measurement/ cluster radius
Analyze/ visualize results
Determine if outliers can be explained or are “bad data”

Deliverable

Copy the terminal contents of your dumbo session, save as text file, submit to NYU Classes

Due: Wednesday, April 5, 2017 at 6pm