Lab 5

Big Data, Spring 2018

March 19, 2018

Today’s Lab

- Outlier Detection with Spark MLLIB, SparkSQL
- Application: Traffic Sensors
    - Sensor readings are notoriously noisy/dirty
    - How can we identify and analyze anomalies in the data?

- Frequently an iterative process:
    - Categorize/cluster data -> identify anomalies -> analyze model and trends

Today's Lab

- The data: 9 columns
  - 0: highway (int)
  - 1: sensorloc (int)
  - 2: sensorid (int)
  - 3: dayofyear (int)
  - 4: dayofweek (int)
  - 5: time (float) (minutes since midnight)
  - 6: volume (int)
  - 7: speed (int)
  - 8: 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

- Challenge: situations where each reading itself has reasonable value, but combination is likely an anomaly (e.g., 100 mph is reasonable with low occupancy and volume, but very unlikely during a traffic jam)

Detecting Anomalies

1. Identify common situations (represented by a large cluster of similar features)
   - "k-means" clustering, choosing number of clusters

2. Identify observations that are sufficiently different from those clusters
   - Measure distance from each observation to the closest cluster
   - If distance is far, classify as anomaly

Setup on Dumbo

- Login to dumbo
- If you don't already have the following in your .bashrc file, type:

```
module load python/gnu/3.4.4
module load spark/2.2.0
export PYSPARK_PYTHON=/share/apps/python/3.4.4/bin/python
```

- Set the additional environment variables by typing:

```
export PYTHONHASHSEED=0
export SPARK_YARN_USER_ENV=PYTHONHASHSEED=0
```

- Start pyspark:

```
pyspark
```

- 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)
```

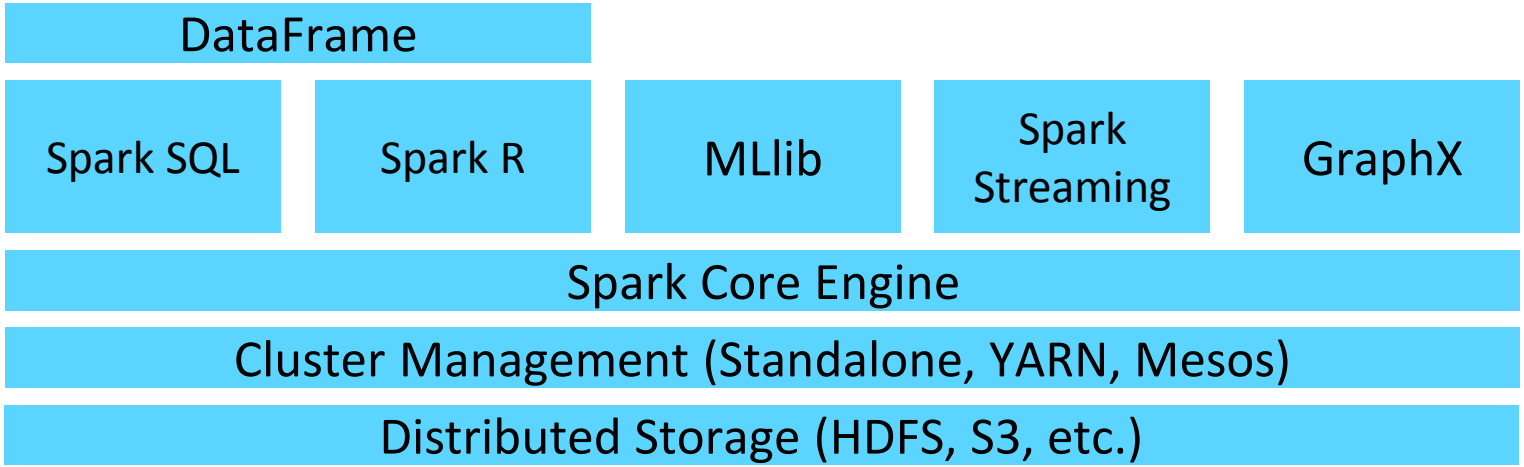
```
sdfilt.count()
```

Stripping the vol, speed, occ columns

```
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 for use with RDDs
  - Transitioning towards ML for DataFrames
- 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

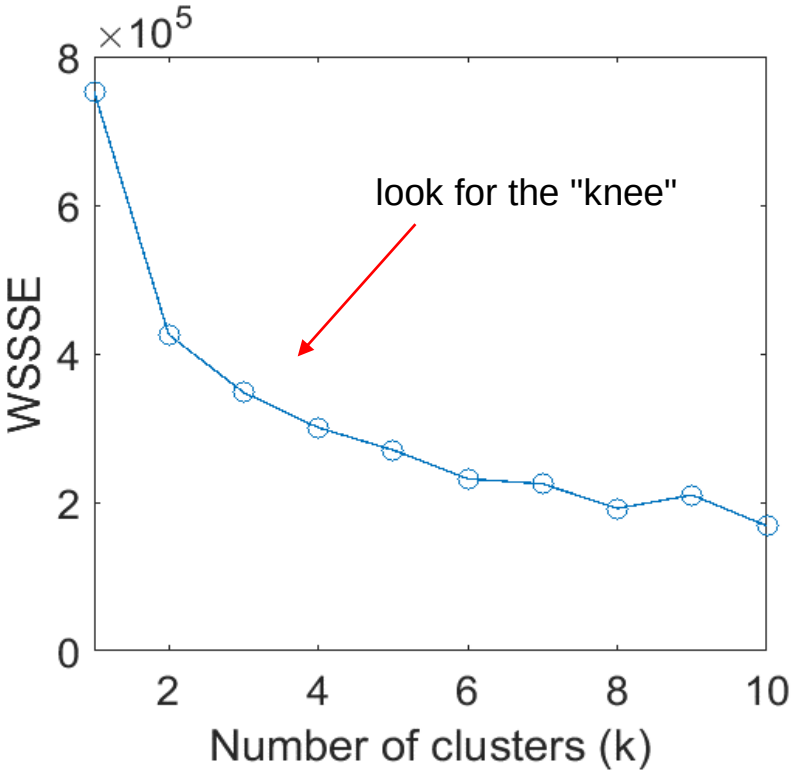
Finding the best k

- 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))
```


×10^5
8
6
4
2
0
WSSSE
look for the "knee"
2
4
6
8
10
Number of clusters (k)

- 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]))
```

Add cluster columns to RDD

###For each point, add 4 columns that give number of cluster point is in, and 3 coordinates of that cluster's center, and the distance from that point to center

```
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))


rdd_w_clusts = sdfilt.map(lambda x: addclustercols(x))
```

###Count number of points in each cluster

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

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")
```

SQL Queries in Spark

```
spark.sql("SELECT * FROM sd WHERE dist>50").show()
```

```
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()
```

For a given point x, if it is greater than threshold distance t away from cluster center, mark it as an anomaly

```
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))
```

Create RDD with anomalies marked as being in the "null 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)
```

Create table for use in SparkSQL:

```
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")
```

Look at measurements for points deemed to be anomalies:

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

Analyzing Anomalies

Inspect number of anomalies by sensor_id:

```
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()
```

Inspect anomalies in a given cluster:

```
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()
```

Trying different k values

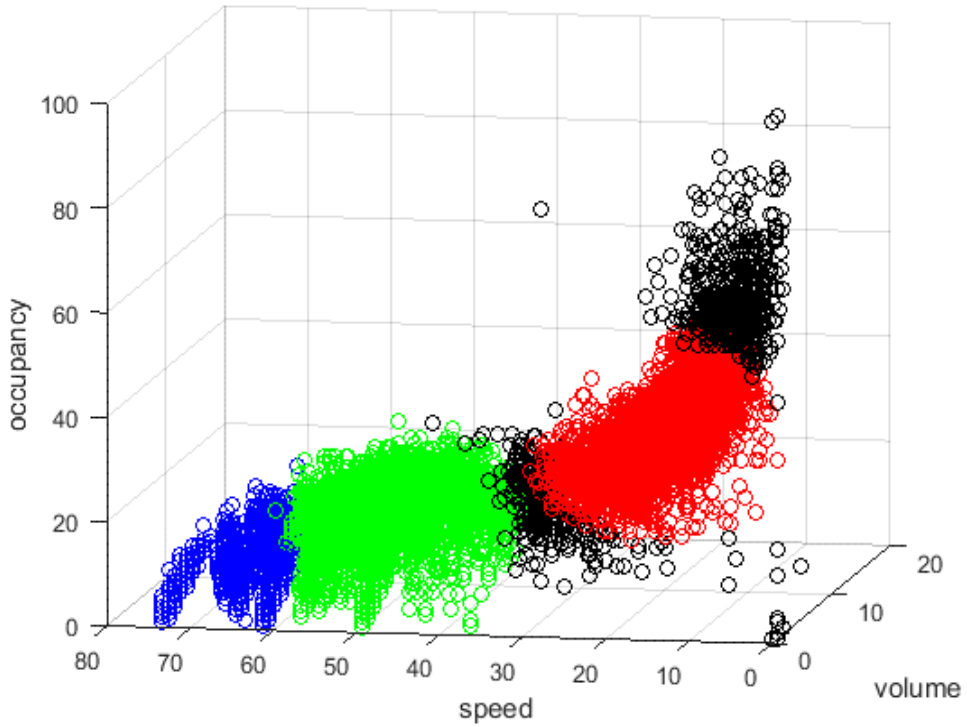
```
clusters = KMeans.train(vso, 5, maxIterations=10, initializationMode="random")
rdd_w_clustsk5 = sdfilt.map(lambda x: addclustercols(x))
schema_sd = spark.createDataFrame(rdd_w_clustsk5, ('highway','sensorloc',
'sensorid', 'doy', 'dow', 'time', 'p_v', 'p_s', 'p_o', 'cluster', 'c_v', 'c_s',
'c_o',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

```
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='|')
```

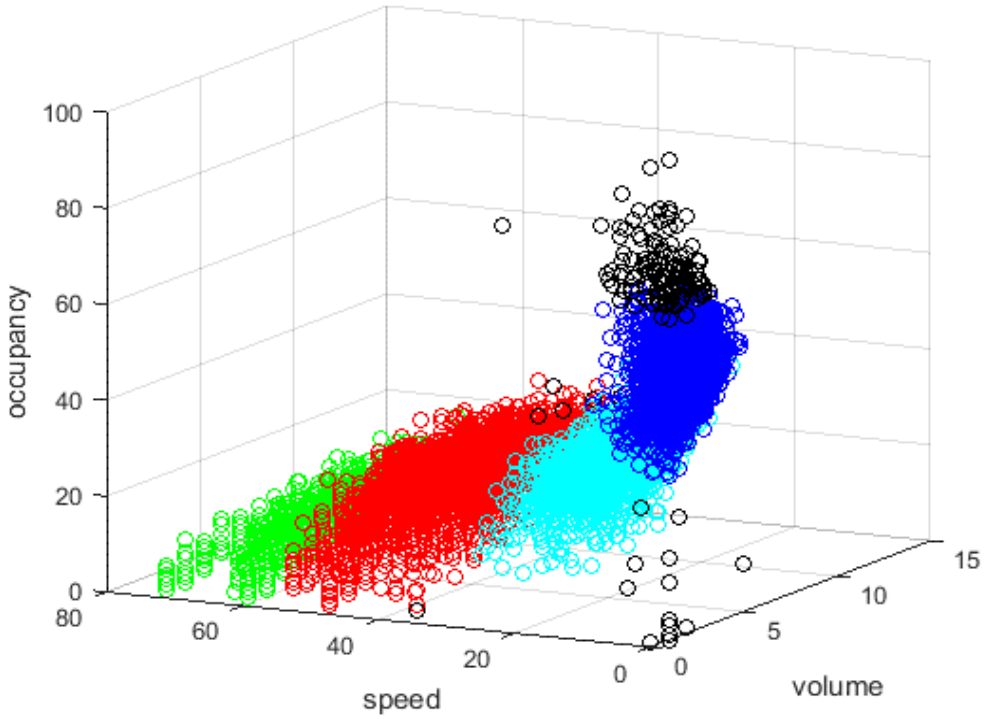
k=3

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



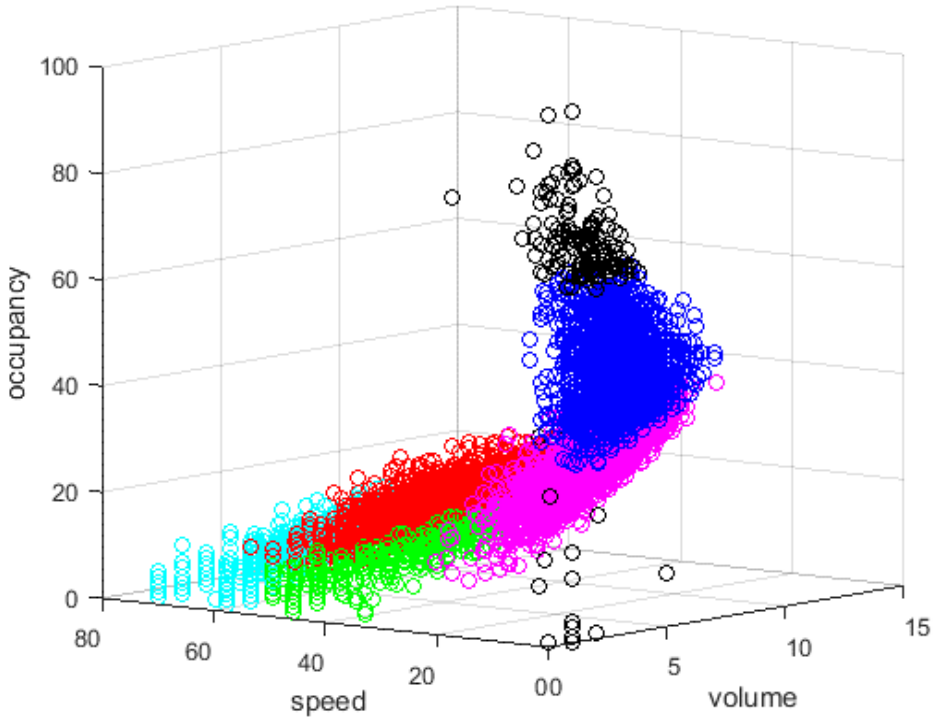
k=4

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



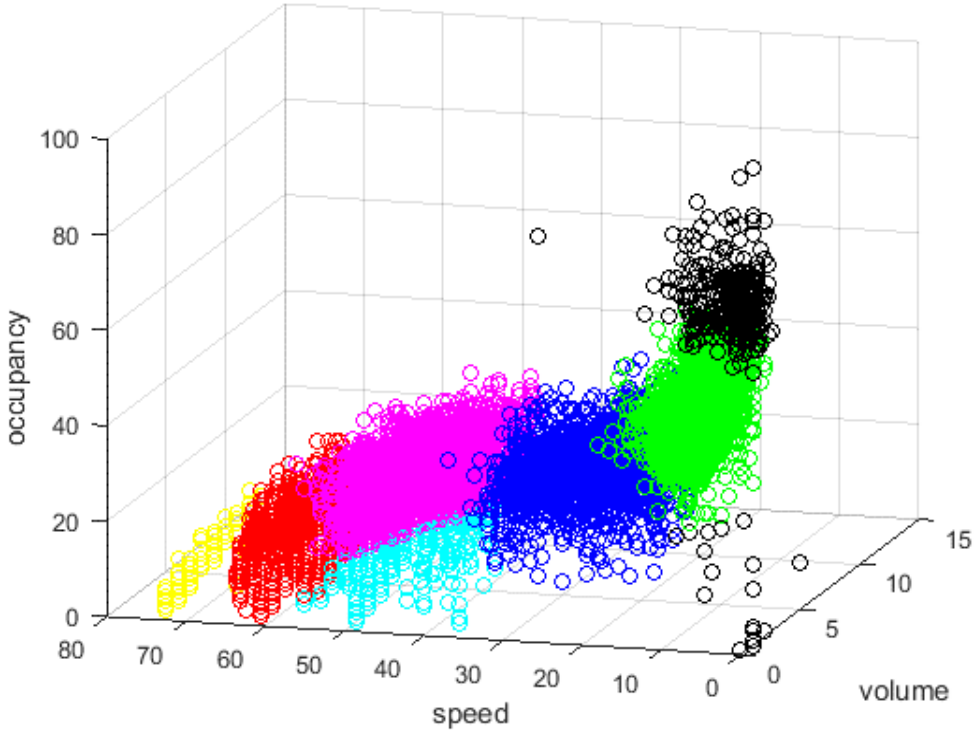
k=5

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



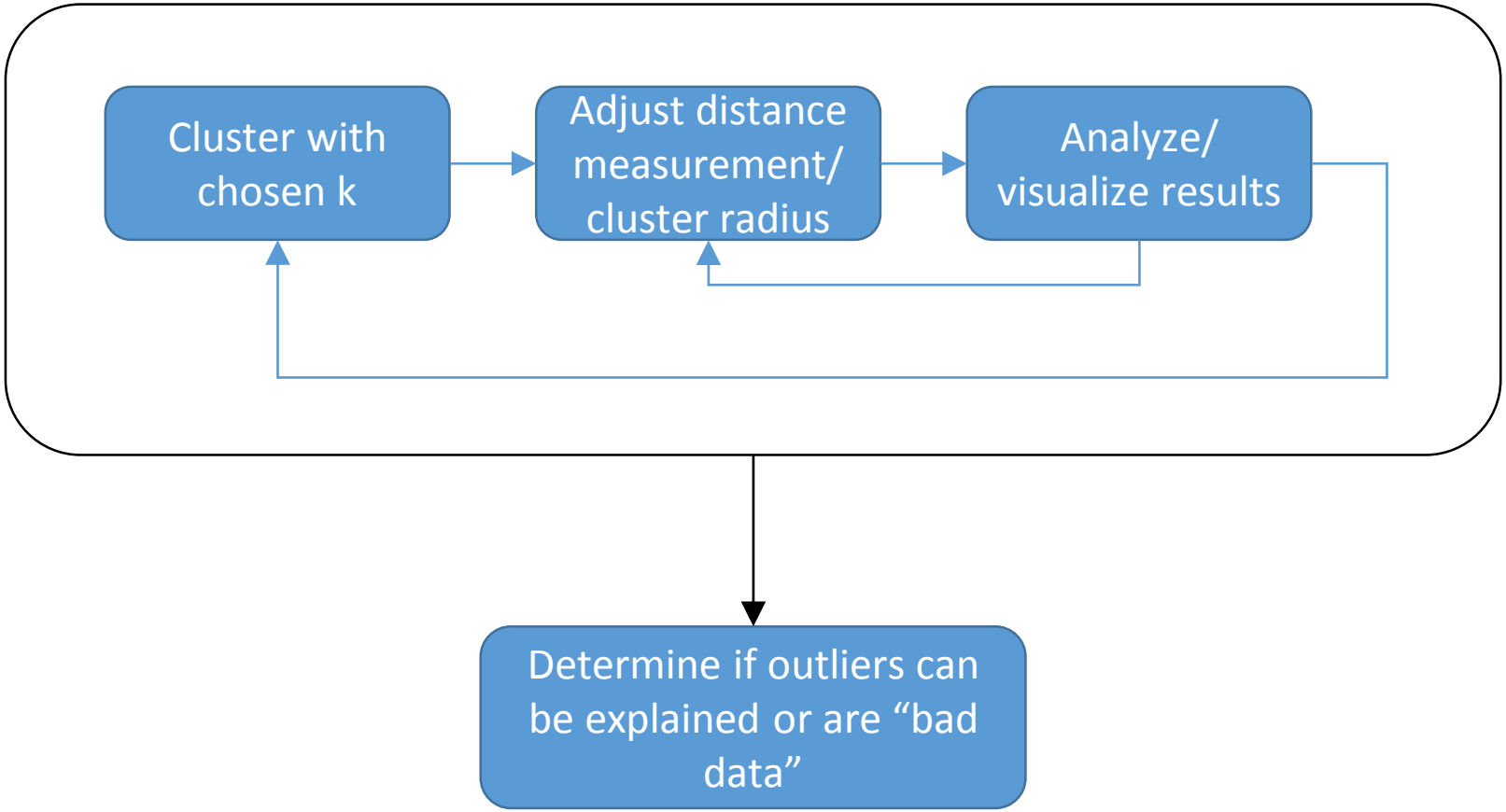
k=6

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: Monday, March 26, 2018 at 6pm