Clustering Past K-Means

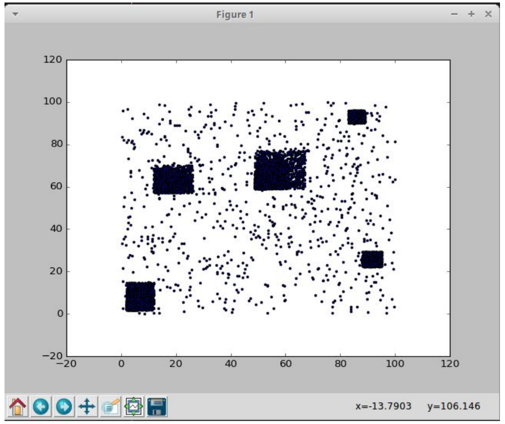
How to fit an unknown number of clusters...

TEXAS STATE UNIVERSITY

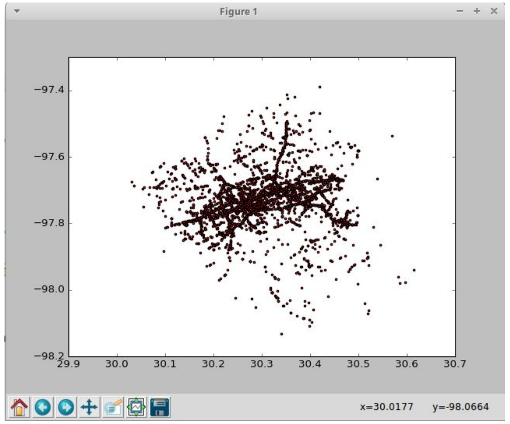
The rising STAR of Texas

Sample Data:

- ❖ First Data Set:
  - – Generated by me with 5 cluster and some noise
  - – 10k entries
- ❖ Second Data Set:
  - – "Real Time Traffic Incident Reports" found at data.austintexas.gov
  - – ~80k entries



Set 1



Set 2

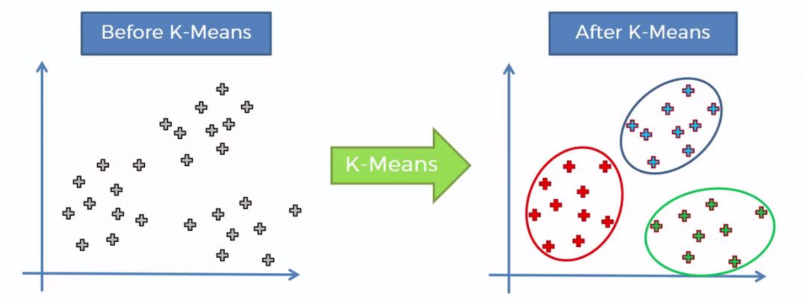
K-Means Clustering:

The Good:

- ❖ Reliable
- ❖ Common
- ❖ Easy to Use
- ❖ Well Accepted

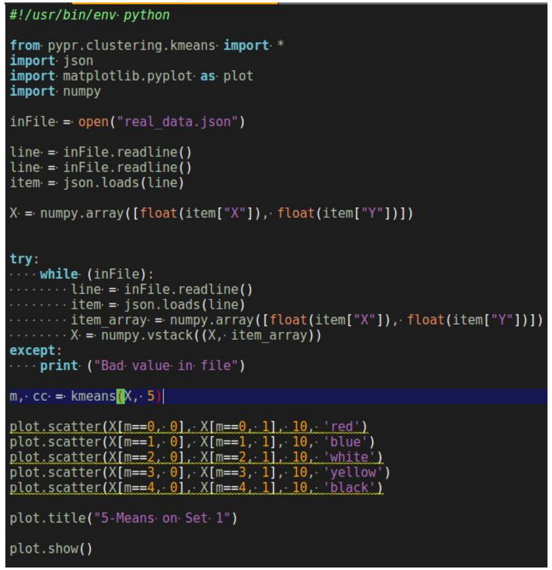
The Bad:

- ❖ Relies on assuming a correct number of clusters.
- ❖ Scales poorly to high-dimension data.
- ❖ "Fixed" rather than "fuzzy" clustering.

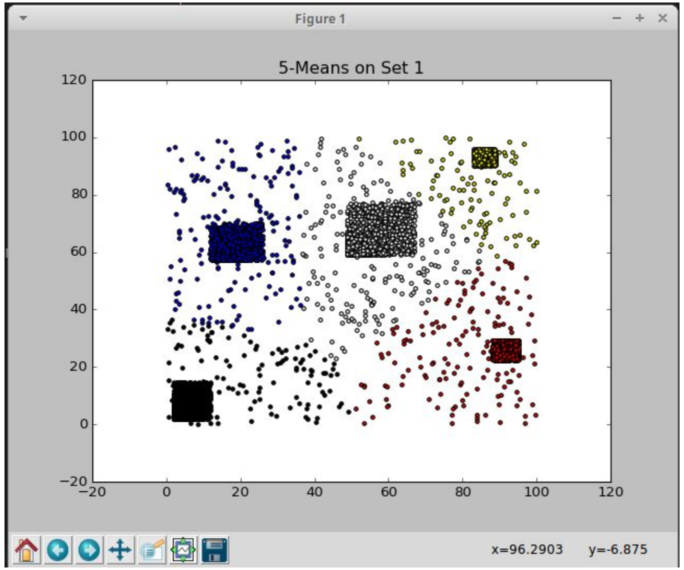


5-Means on Set 1:

- ❖ Using PyPR from: http://pypr.sourceforge.net/
- ❖ Takes NumPy array of data and returns:
  - m: an array of memberships
  - cc: an array of centroids

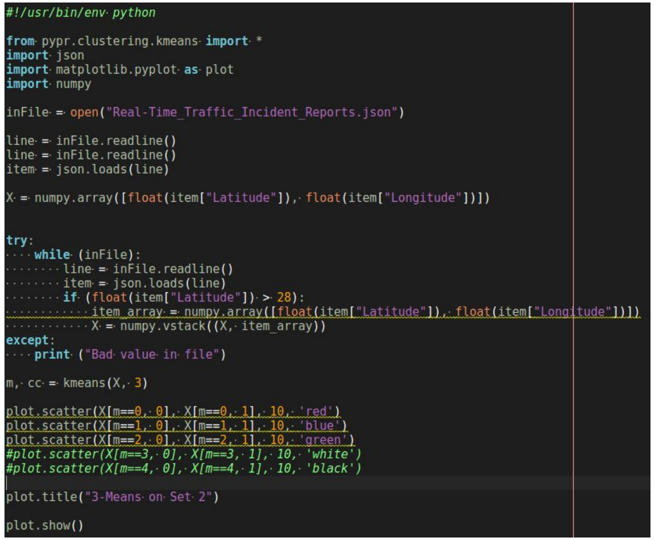


```
#!/usr/bin/env python

from pypr.clustering.kmeans import *
import json
import matplotlib.pyplot as plot
import numpy

inFile = open("real_data.json")

line = inFile.readline()
line = inFile.readline()
item = json.loads(line)

X = numpy.array([float(item["X"]), float(item["Y"])])


try:
    while (inFile):
        line = inFile.readline()
        item = json.loads(line)
        item_array = numpy.array([float(item["X"]), float(item["Y"])])
        X = numpy.vstack((X, item_array))
except:
    print ("Bad value in file")

m, cc = kmeans(X, 5)

plot.scatter(X[m==0, 0], X[m==0, 1], 10, 'red')
plot.scatter(X[m==1, 0], X[m==1, 1], 10, 'blue')
plot.scatter(X[m==2, 0], X[m==2, 1], 10, 'white')
plot.scatter(X[m==3, 0], X[m==3, 1], 10, 'yellow')
plot.scatter(X[m==4, 0], X[m==4, 1], 10, 'black')

plot.title("5-Means on Set 1")

plot.show()
```

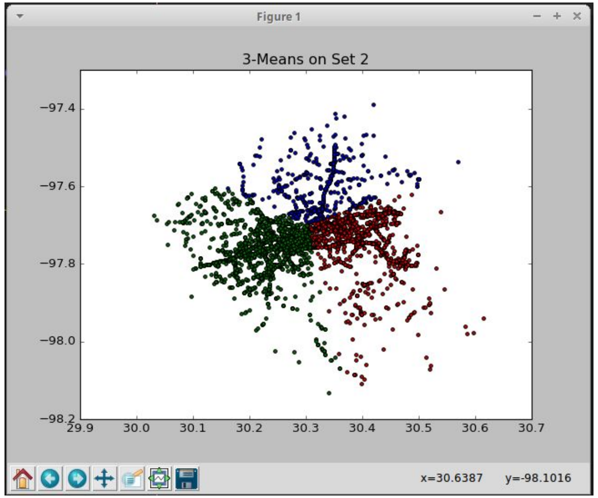


3-Means on Set 2:

- Using PyPR from: http://pypr.sourceforge.net/
- 3 Clusters were chosen since the exact clusters are unclear from inspection.
- Some filtering for erroneous values was required.



```
#!/usr/bin/env python

from pypr.clustering.kmeans import *
import json
import matplotlib.pyplot as plot
import numpy

inFile = open("Real-Time_Traffic_Incident_Reports.json")

line = inFile.readline()
line = inFile.readline()
item = json.loads(line)

X = numpy.array([float(item["Latitude"]), float(item["Longitude"])])


try:
    while (inFile):
        line = inFile.readline()
        item = json.loads(line)
        if (float(item["Latitude"]) > 28):
            item_array = numpy.array([float(item["Latitude"]), float(item["Longitude"])])
            X = numpy.vstack((X, item_array))
except:
    print ("Bad value in file")

m, cc = kmeans(X, 3)

plot.scatter(X[m==0, 0], X[m==0, 1], 10, 'red')
plot.scatter(X[m==1, 0], X[m==1, 1], 10, 'blue')
plot.scatter(X[m==2, 0], X[m==2, 1], 10, 'green')
#plot.scatter(X[m==3, 0], X[m==3, 1], 10, 'white')
#plot.scatter(X[m==4, 0], X[m==4, 1], 10, 'black')

plot.title("3-Means on Set 2")

plot.show()
```

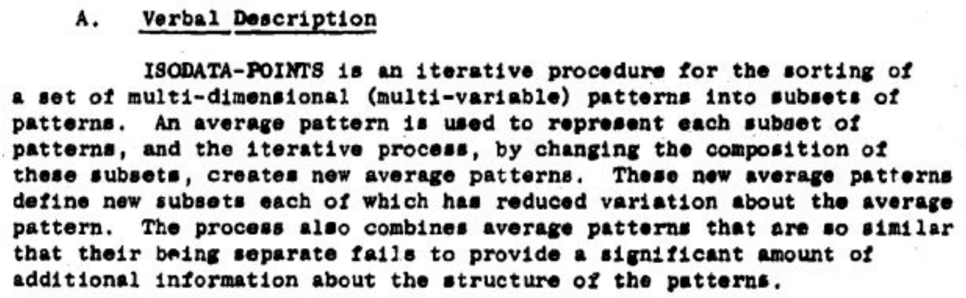


Step by Step for pypr.kmeans:

- Find or generate some data.
- Load your data into an N x D numpy array (X) where N is the number of entries and D is the dimensionality of the data, which must be constant.
- Choose a fixed K, which is the number of clusters.
- Call kmeans(X, K) to get:
  - m: a 1 x N dimensional array of memberships
  - cc: a K x D dimensional array of the centroids for your clusters
- Do what you want with m and cc
- Resources:
  - PyPr K-Means example: http://pypr.sourceforge.net/kmeans.html#k-means-example
  - Numpy arrays: https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.array.html
  - Matplotlib: https://matplotlib.org/

Isodata Clustering:

- ❖ Introduced in 1965 by Stanford Research Institute
- ❖ Iterative Self-Organizing Data Analysis Technique
- ❖ Originally developed for tracking weather data
- ❖ Focused on high-dimension data
- ❖ Chooses centroids from data-points
- ❖ Not common or well supported



A. **Verbal Description**

ISODATA-POINTS is an iterative procedure for the sorting of a set of multi-dimensional (multi-variable) patterns into subsets of patterns. An average pattern is used to represent each subset of patterns, and the iterative process, by changing the composition of these subsets, creates new average patterns. These new average patterns define new subsets each of which has reduced variation about the average pattern. The process also combines average patterns that are so similar that their being separate fails to provide a significant amount of additional information about the structure of the patterns.

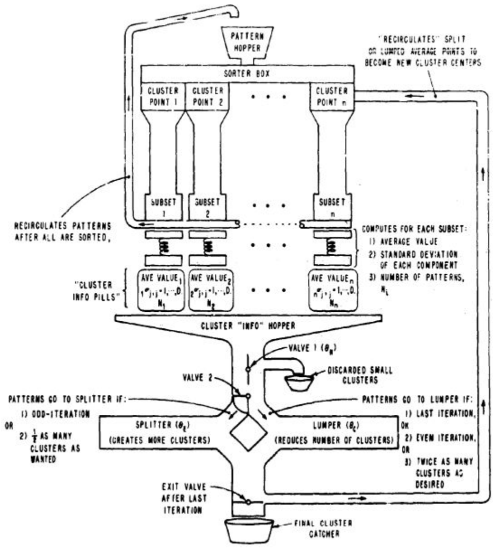
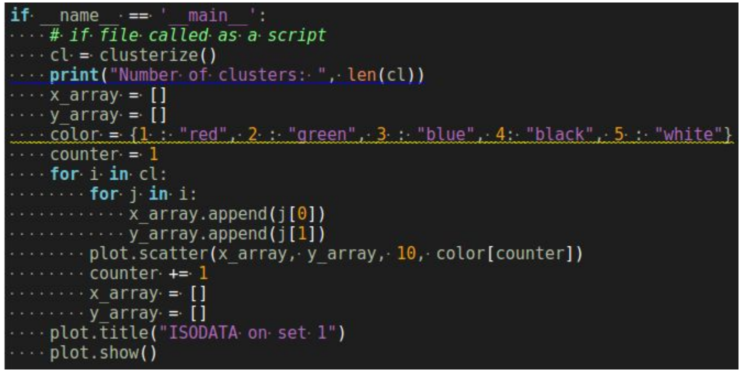

PATTERN HOPPER
SORTER BOX
CLUSTER POINT 1
CLUSTER POINT 2
CLUSTER POINT n
"RECIRCULATES" SPLIT OR LUMPED AVERAGE POINTS TO BECOME NEW CLUSTER CENTERS
SUBSET 1
SUBSET 2
SUBSET n
RECIRCULATES PATTERNS AFTER ALL ARE SORTED,
COMPUTES FOR EACH SUBSET: 1) AVERAGE VALUE 2) STANDARD DEVIATION OF EACH COMPONENT 3) NUMBER OF PATTERNS, N_i
"CLUSTER INFO PILLS"
AVE VALUE$_1$ $_1\sigma_j, j = 1, \cdots, D$ N_1
AVE VALUE$_2$ $_2\sigma_j, j = 1, \cdots, D$ N_2
AVE VALUE$_n$ $_n\sigma_j, j = 1, \cdots, D$ N_n
CLUSTER "INFO" HOPPER
VALVE 1 (θ_N)
DISCARDED SMALL CLUSTERS
VALVE 2
PATTERNS GO TO SPLITTER IF: 1) ODD-ITERATION OR 2) $\frac{1}{2}$ AS MANY CLUSTERS AS WANTED
PATTERNS GO TO LUMPER IF: 1) LAST ITERATION, OR 2) EVEN ITERATION, OR 3) TWICE AS MANY CLUSTERS AS DESIRED
SPLITTER (θ_E) (CREATES MORE CLUSTERS)
LUMPER (θ_C) (REDUCES NUMBER OF CLUSTERS)
EXIT VALVE AFTER LAST ITERATION
FINAL CLUSTER CATCHER

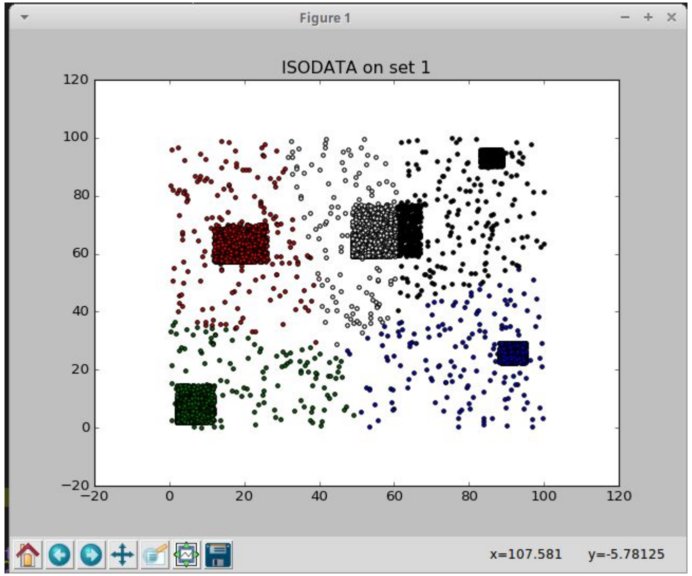
FIG. 1 A PICTORIAL DESCRIPTION OF ISODATA-POINTS

Isodata on Set 1:

- Using IsoData from: https://github.com/molivia/isodata
- Variable number of clusters up to defined maximum
- Results are ...less than awesome. But there are some parameters to play with.

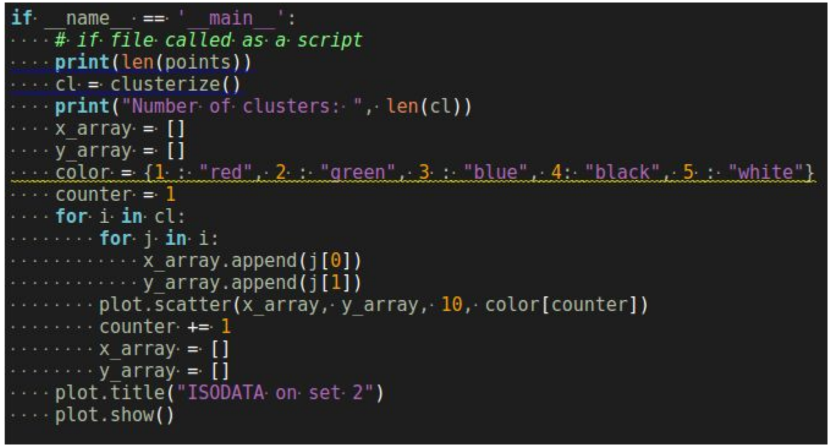


```
if __name__ == '__main__':
    # if file called as a script
    cl = clusterize()
    print("Number of clusters: ", len(cl))
    x_array = []
    y_array = []
    color = {1 : "red", 2 : "green", 3 : "blue", 4: "black", 5 : "white"}
    counter = 1
    for i in cl:
        for j in i:
            x_array.append(j[0])
            y_array.append(j[1])
        plot.scatter(x_array, y_array, 10, color[counter])
        counter += 1
        x_array = []
        y_array = []
    plot.title("ISODATA on set 1")
    plot.show()
```

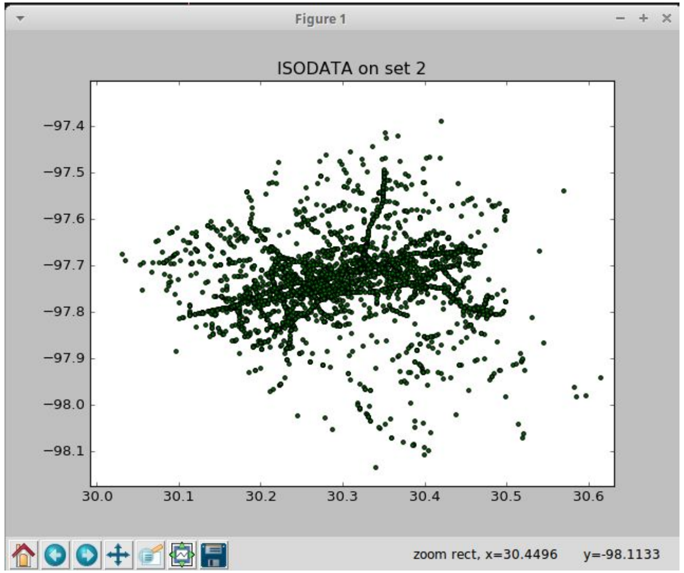


Isodata on Set 2:

- ❖ Using IsoData from: https://github.com/molivia/isodata
- ❖ We can no longer force a number of clusters greater than 1
- ❖ Cleaning erroneous data proved more dificult



```
if __name__ == '__main__':
    # if file called as a script
    print(len(points))
    cl = clusterize()
    print("Number of clusters: ", len(cl))
    x_array = []
    y_array = []
    color = {1 : "red", 2 : "green", 3 : "blue", 4: "black", 5 : "white"}
    counter = 1
    for i in cl:
        for j in i:
            x_array.append(j[0])
            y_array.append(j[1])
        plot.scatter(x_array, y_array, 10, color[counter])
        counter += 1
        x_array = []
        y_array = []
    plot.title("ISODATA on set 2")
    plot.show()
```

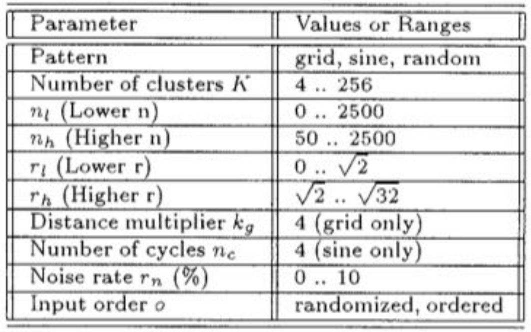


Step by Step for Molivia’s ISODATA:

- Find or generate some data.
- Form data into a list of tuples.
- Call clusterize which will return a list of list of tuples with the items of the top level lists being the collections of items in each cluster.
- Do what you want with your new lists
- Resources:
  - Matplotlib: https://matplotlib.org/
  - ISODATA: https://github.com/molivia/isodata/blob/master/GUI_isodata.py

BIRCH Clustering:

- ❖ Balanced Iterative Reducing and Clustering using Hierarchies
- ❖ First proposed in 1996
- ❖ Designed to better compensate for “noisy” data than previous models.
- ❖ Builds a tree based on Clustering Factors



Parameter	Values or Ranges
Pattern	grid, sine, random
Number of clusters K	4 .. 256
n_l (Lower n)	0 .. 2500
n_h (Higher n)	50 .. 2500
r_l (Lower r)	0 .. $\sqrt{2}$
r_h (Higher r)	$\sqrt{2}$ .. $\sqrt{32}$
Distance multiplier k_g	4 (grid only)
Number of cycles n_c	4 (sine only)
Noise rate r_n (%)	0 .. 10
Input order o	randomized, ordered

Table 1: *Data Generation Parameters and Their Values or Ranges Experimented*

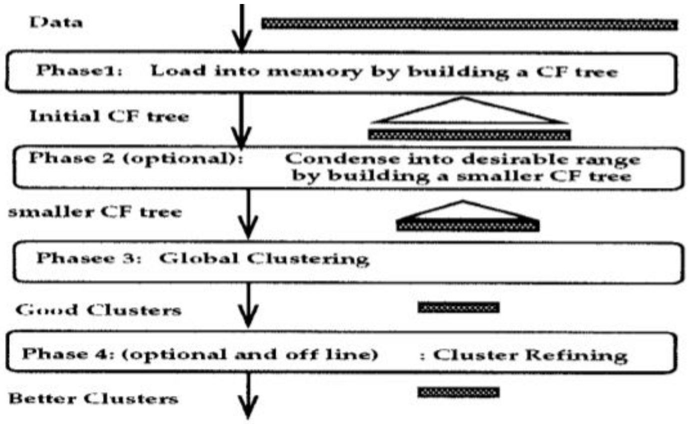
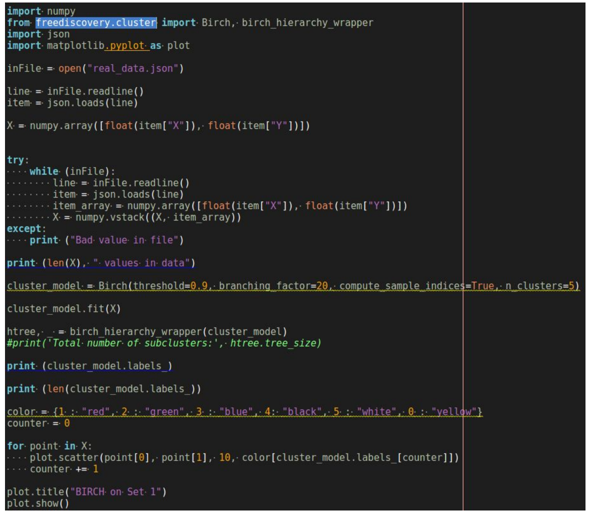

Data
Phase1: Load into memory by building a CF tree
Initial CF tree
Phase 2 (optional): Condense into desirable range by building a smaller CF tree
smaller CF tree
Phasee 3: Global Clustering
Good Clusters
Phase 4: (optional and off line) : Cluster Refining
Better Clusters

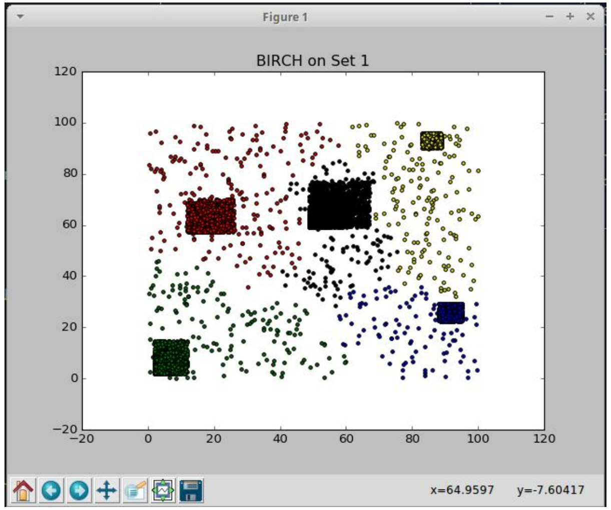
Figure 1: *BIRCH Overview*

BIRCH on Set 1:

- ❖ Using Birch from freediscovery.cluster
- ❖ This is a “patched version” of the SciKitLearn BIRCH
- ❖ Takes a starting number of clusters as a parameter

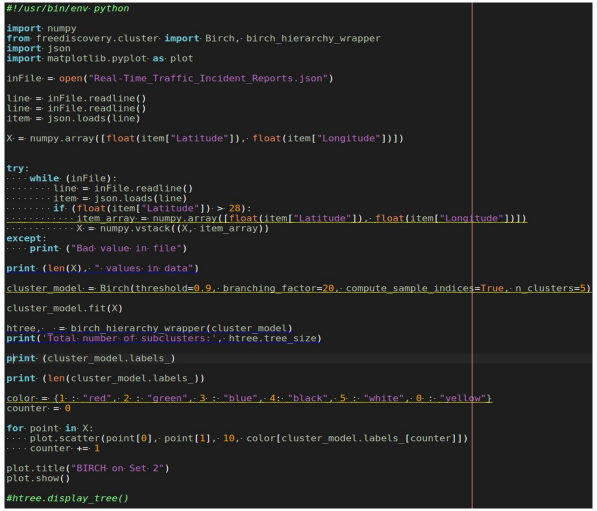


```python
import numpy
from freediscovery.cluster import Birch, birch_hierarchy_wrapper
import json
import matplotlib.pyplot as plot

inFile = open("real_data.json")

line = inFile.readline()
item = json.loads(line)

X = numpy.array([float(item["X"]), float(item["Y"])])


try:
    while (inFile):
        line = inFile.readline()
        item = json.loads(line)
        item_array = numpy.array([float(item["X"]), float(item["Y"])])
        X = numpy.vstack((X, item_array))
except:
    print ("Bad value in file")

print (len(X), " values in data")

cluster_model = Birch(threshold=0.9, branching_factor=20, compute_sample_indices=True, n_clusters=5)

cluster_model.fit(X)

htree, _ = birch_hierarchy_wrapper(cluster_model)
#print('Total number of subclusters:', htree.tree_size)

print (cluster_model.labels_)

print (len(cluster_model.labels_))

color = {1 : "red", 2 : "green", 3 : "blue", 4: "black", 5 : "white", 0 : "yellow"}
counter = 0

for point in X:
    plot.scatter(point[0], point[1], 10, color[cluster_model.labels_[counter]])
    counter += 1

plot.title("BIRCH on Set 1")
plot.show()
```

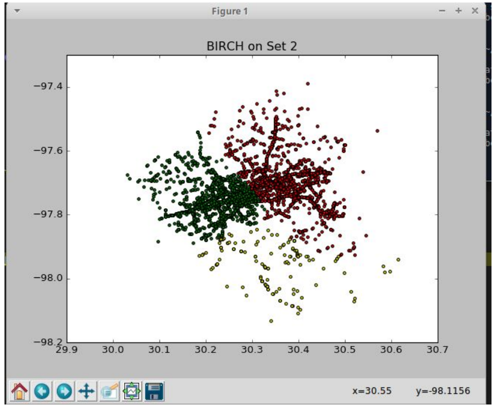


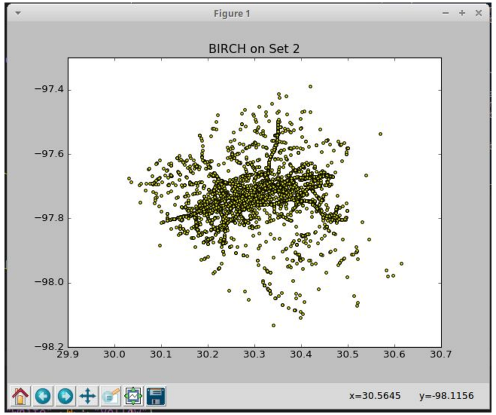
BIRCH on Set 2:

- ❖ Using Birch from freediscovery.cluster
- ❖ Puts everything in 1 cluster or several depending on parameters
- ❖ Slower than K-Means



```python
#!/usr/bin/env python

import numpy
from freediscovery.cluster import Birch, birch_hierarchy_wrapper
import json
import matplotlib.pyplot as plot

inFile = open("Real-Time_Traffic_Incident_Reports.json")

line = inFile.readline()
line = inFile.readline()
item = json.loads(line)

X = numpy.array([float(item["Latitude"]), float(item["Longitude"])])


try:
    while (inFile):
        line = inFile.readline()
        item = json.loads(line)
        if (float(item["Latitude"]) > 28):
            item_array = numpy.array([float(item["Latitude"]), float(item["Longitude"])])
            X = numpy.vstack((X, item_array))
except:
    print ("Bad value in file")

print (len(X), " values in data")

cluster_model = Birch(threshold=0.9, branching_factor=20, compute_sample_indices=True, n_clusters=5)

cluster_model.fit(X)

htree, _ = birch_hierarchy_wrapper(cluster_model)
print('Total number of subclusters:', htree.tree_size)

print (cluster_model.labels_)

print (len(cluster_model.labels_))

color = {1 : "red", 2 : "green", 3 : "blue", 4: "black", 5 : "white", 0 : "yellow"}
counter = 0

for point in X:
    plot.scatter(point[0], point[1], 10, color[cluster_model.labels_[counter]])
    counter += 1

plot.title("BIRCH on Set 2")
plot.show()

#htree.display_tree()
```





Step by Step for Molivia's ISODATA:

- ❖ Load a numpy array with your data
- ❖ Create a "cluster model" with the Birch(...) constructor. This is when you set the parameters.
- ❖ Use the fit function of you cluster model to fit the model to the data.
- ❖ The following attributes in you model are now loaded:
  - subcluster_centers_ : an array of subcluster centers
  - labels_: an array of the data labels
  - root_ : location of the root node of the tree
- ❖ Using the transform(X) function will alter the data set X to represent distance of each point to its centroid.

DP Means Clustering:

- ❖ Introduced in 2014 by a team from MIT, Oxford, and Ohio State University
- ❖ Based on Dirichlet Process from probability theory
- ❖ Guarantees results similar to K-Means without prior knowledge of data.
- ❖ Developed using a dataset of aircraft trajectories.
- ❖ High efficiency and computation light
- ❖ Fit for time-sensitive operations
- ❖ Not well supported!

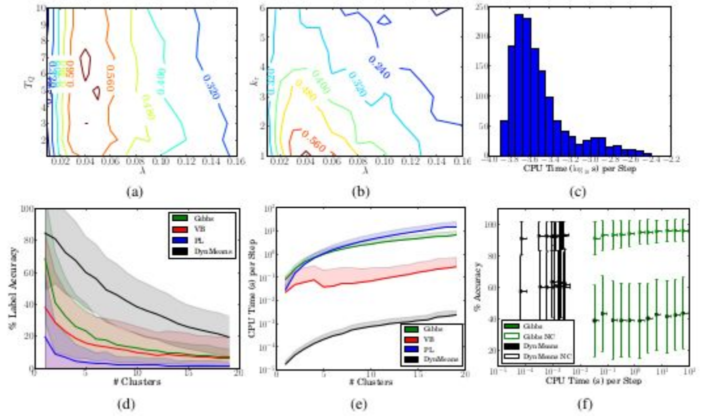
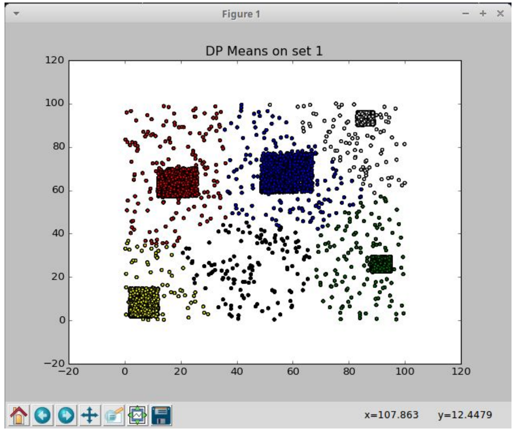


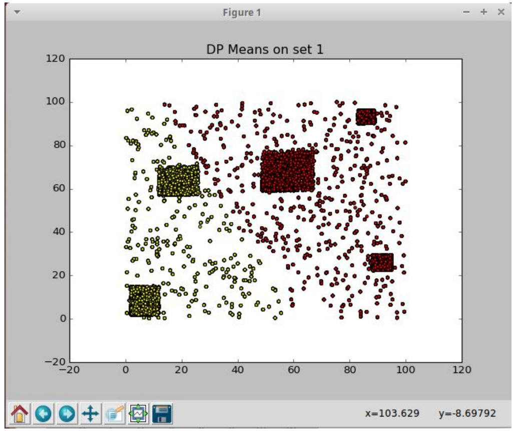
Figure 1: (1a - 1c): Accuracy contours and CPU time histogram for the Dynamic Means algorithm. (1d - 1e): Comparison with Gibbs sampling, variational inference, and particle learning. Shaded region indicates 1σ interval; in (1e), only upper half is shown. (1f): Comparison of accuracy when enforcing (Gibbs, DynMeans) and not enforcing (Gibbs NC, DynMeans NC) correct cluster tracking.

DP-Means on Set 1:

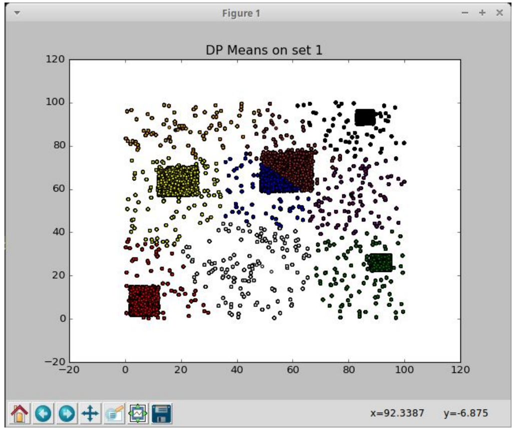
- Using dpmeans from https://github.com/DrSkippy/Python-DP-Means-Clustering
- Convenient-ish command line tool
- Very sensitive to ‘lambda’ parameter



Lambda = 40



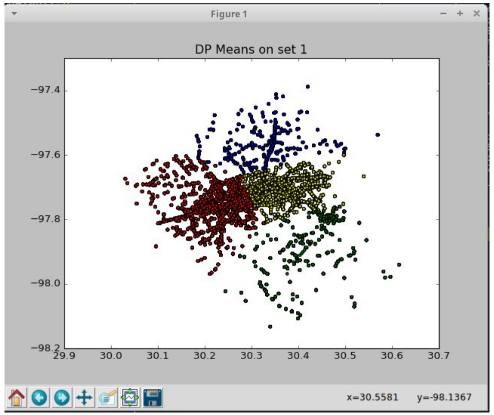
Lambda = 80



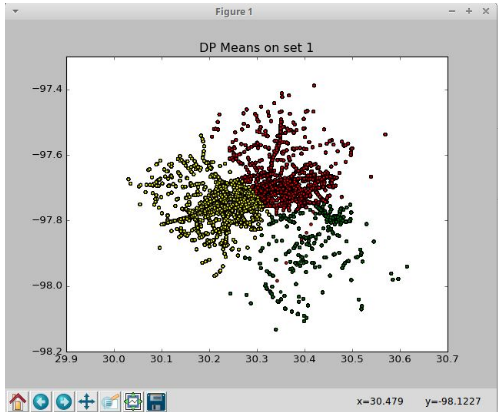
Lambda = 32

DP-Means on Set 2:

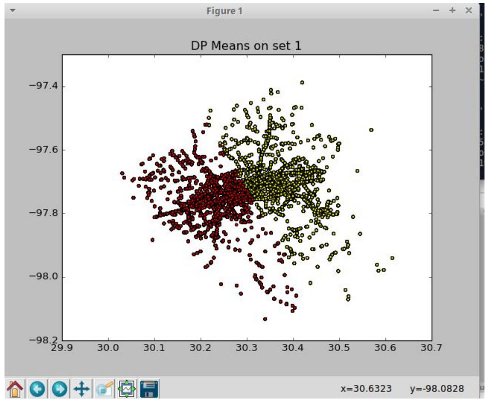
- ❖ Using dpmeans from https://github.com/DrSkippy/Python-DP-Means-Clustering
- ❖ Even more sensitive to lambda



Lambda = 0.3



Lambda = 0.4



Lambda = 0.5

Step by Step for Dr. Skippy’s DP-Means:

- ❖ The CSV file holding the data was hard-coded, so alter the file name to read in whatever data you like.
- ❖ Call cluster.py -l LAMBDA from the command line with whatever lambda value you desire for the data.
- ❖ The script as-written sends the output of the clustering to a csv file. The output is the data point, cluster number, and the iteration on which a datapoint was classified.
- ❖ Resources:
  - Dr. Skippy’s DP-Means: https://github.com/DrSkippy/Python-DP-Means-Clustering
  - Matplotlib: https://matplotlib.org/

Questions or Comments?