Practical 7

Text Analytics: Clustering

Otto Hermann
otto.hermann@ucdconnect.ie
16203034

COMP47600

University College Dublin
31st October 2017

1

Use init_base

This code will generate 15 random points

```
  data = init_board(15)
```

Run 10 times using k=3

```
# inputs
number_of_points = 15
iterations = 10
number_of_clusters = 3

# create the data
data_runs = list()
for j in range(iterations):
    data_runs.append(list(init_board(number_of_points)))

# organise the data
mu_dict = dict()
cluster_dict = dict()
counter = 0
for run in data_runs:
    center = find_centers(run, number_of_clusters)
    # parse_output(center) # uncomment if you want graphs
    mu_dict[counter] = center[0]
    cluster_dict[counter] = center[1]
    counter += 1

# count repeats
mu_set = set()
cluster_set = set()
for c in range(counter):
    mu_array = mu_dict[c]
    for pair in mu_array:
        mu_set.add(tuple(pair))
    super_cluster_array = cluster_dict[c]
    for index in super_cluster_array:
        cluster_array = super_cluster_array[index]
        for pair in cluster_array:
            cluster_set.add(tuple(pair))
print("{} out of {} mu values are unique".format(len(mu_set), len(mu_dict) *
     number_of_clusters))
print("{} out of {} cluster values are unique".format(len(cluster_set),
     len(cluster_dict) * number_of_points))
```

The code runs as expected and explained at the source website (there are some issues with the code, but not material for our purposes e.g. find_centers will throw an error if has_converged is true from the start i.e. oldmu and mu, which are independently and randomly assigned, are given the same value before the while loop. This is a low probability event, but it did occur when I was doing high iteration testing.

I've attached two output graphs as reporting the actual point values or all graphs would be voluminous at best, but for the few thousand iterations I ran, for each run, there were no repeated clusters i.e. each time the code above was run 30 out of 30 mu values were unique and 150 out of 150 coordinates were unique. This was to be expected given the range of possible random values and the number of points generated at each run.

In some instances the data seemed to be clustered in a bizarre fashion, but this was also to be expected: there were often more than three apparent clusters (e.g. Figure 2) or the initial placement of the centroid was not ideal. These are just challenges encountered when using k-means clustering. To help alleviate the problem there are multiple methods, including trying multiple values of k and, for each value of k, doing multiple random starting points; these results then can then be evaluated on various metrics for internal-cluster similarities and cluster-to-cluster differences.

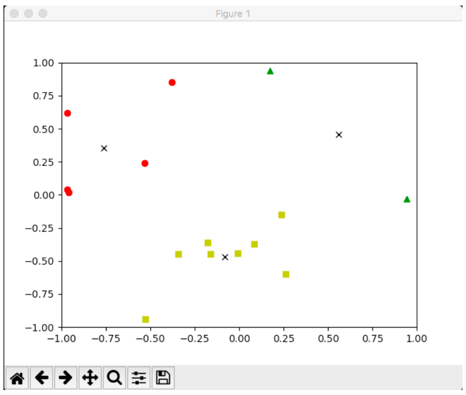

Figure 1: One run

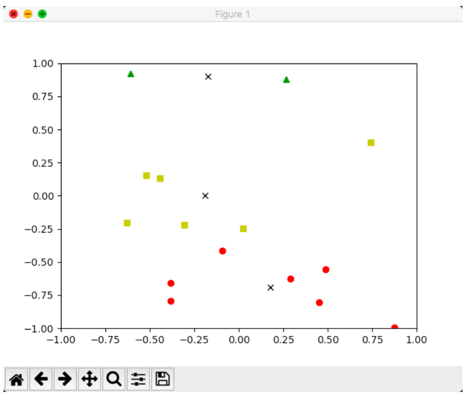

Figure 2: Another run

2

Below you can see the code and the artificial clusters I've generated

```
# data
```

```
data = np.array([
        [0.75, 0.3],
        [0.75, 0.4],
        [0.75, 0.5],
        [-0.75, 0.3],
        [-0.75, 0.4],
        [-0.75, 0.5],
        [-0.75, -0.3],
        [-0.75, -0.4],
        [-0.75, -0.5],
        [0.75, -0.3],
        [0.75, -0.4],
        [0.75, -0.5],
        [0.0, 0.0],
        [0.0, 0.1],
        [0.0, -0.1],
    ])
data = list(data)

# inputs
iterations = 20
number_of_clusters = 3

# organise the data
mu_dict = dict()
cluster_dict = dict()
counter = 0
for j in range(iterations):
    center = find_centers(data, number_of_clusters)
    # parse_output(center) # uncomment if you want graphs
    mu_dict[counter] = center[0]
    cluster_dict[counter] = center[1]
    counter += 1

# count repeats
mu_set = set()
cluster_set = set()
for c in range(counter):
    mu_array = mu_dict[c]
    for pair in mu_array:
        mu_set.add(tuple(pair))
    super_cluster_array = cluster_dict[c]
    for index in super_cluster_array:
        cluster_array = super_cluster_array[index]
        for pair in cluster_array:
            cluster_set.add(tuple(pair))
print("{} out of {} mu values are unique".format(len(mu_set), len(mu_dict) *
     number_of_clusters))
print("{} out of {} cluster values are unique".format(len(cluster_set),
     len(cluster_dict) * 15))
```

15 out of the 300 clusters generated are always unique by construction, but the mu's vary but with significant overlap, typically ranging from 12 to 20 out of 60 being the same for each run of the code given above.

This frequency of overlap is to be expected given the symmetry of the five cluster I made and that we are only looking at 3-means clustering within a relatively small region. In 1 I mentioned the reasons for these overlaps and in this constructed case it's quite clear that k should be 5 and that I ought to run multiple iterations of random centroids to better identify the most persistent groupings.

Here are some examples of clusters:

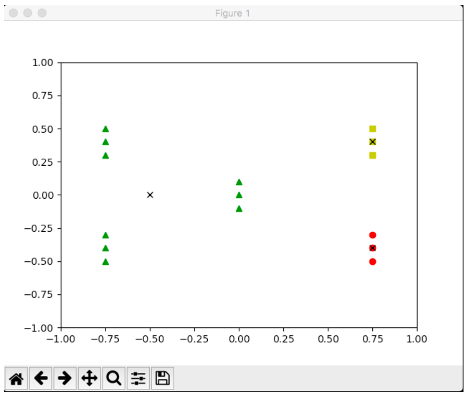

Figure 3: K too small eg 1

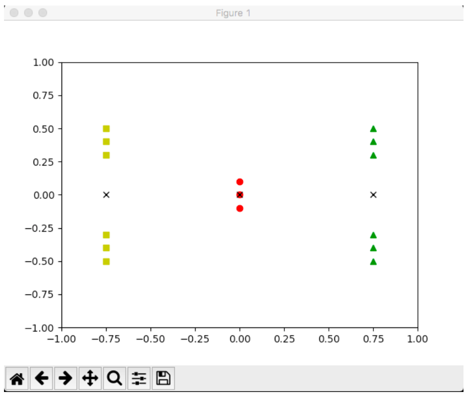

Figure 4: K too small eg 2

3

Different results on different runs

A succinct source supporting my approach can be found here: http://cs229.stanford.edu/notes/cs229-notes7a.pdf

The idea is, for a given value of k, run the algorithm with k random centroids, then run the algorithm iteratively, using the most central point in each cluster as the starting points. This process will generate k-clusters that are locally optimal; in an effort to find a globally optimal solution, one strategy is to run this process n-times and to use some set of context dependent evaluation criteria to determine which is best. It should be understood that finding the globally

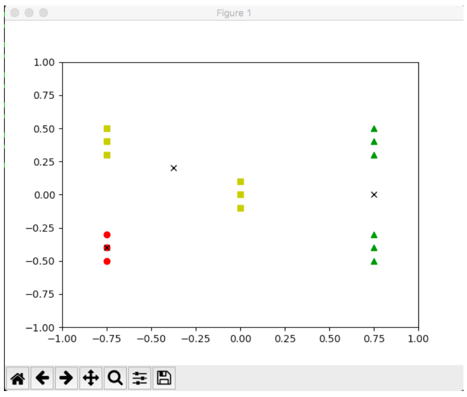

Figure 5: K too small eg 3

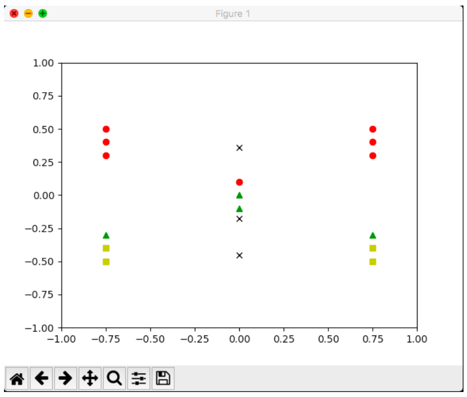

Figure 6: K too small eg 4

optimal point is NP, so we must employ some heuristic in finding it.

Also, as mentioned in 1 above, we ought to run this process for multiple values of k, unless of course we have a priori knowledge of the number of actual clusters. But in most circumstances, we don't have that information and ought to try myriad values of k.

A more malicious problem
If there are non-convex clusters in the data, k-means can continue to produce results that are not entirely useful. While I was not able to find any generally accepted solutions to this problem (indeed such a solution would be useful in myriad domains), there are those actively engaged in mitigating these issues, typically on a case-by-case basis. Two good examples of this type of work include: http://liu.diva-portal.org/smash/get/diva2:650707/FULLTEXT01.pdf and https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5036949/, though the former is more general and addresses the crux of the matter.