Project2:

Praveen Wadikar pwadikar@asu.edu

***Abstract*—Understand how to implement k-means**

Keywords: Statistical machine learning, k-means;

I. Task and Dataset

you are required to implement the K-means algorithm and apply your implementation on the given dataset (AllSamples.npy), which contains a set of 2-D points. You are required to implement the following strategy for choosing the initial cluster centers.

A. Algorithm

- 1.Randomly choose clusters
- 2a.Assign labels based on closest center
- 2b.Find new centers from means of points
- 3.Repeat 2a and 2b until convergence (no change in mean)

II. Analysis

The figure shows that the Initial means (brown triangles) were selected randomly. Later the means converged to points marked with (x). As seen the yellow dots converged around a mean inbetween the clusters. However for k=5 mean which lies between two clusters now classifies data. If we increase K then this convergence of class increases. However increasing k values beyond a point causes small improvement in the loss. This should be the cue for maximum value of k

This is show in figure3

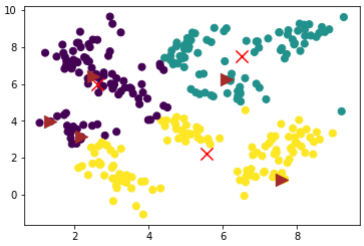

Fig. 1. Plot of the clustered data with k=3

***

III. Results

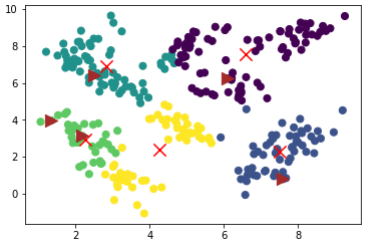

Fig. 2. Plot of the clustered data with k=5

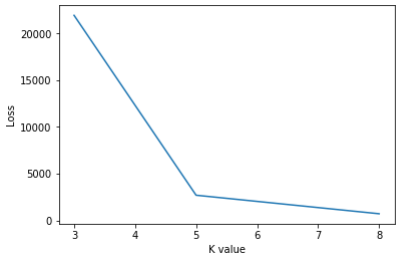

Fig. 3. Plot of the clustered data with k=3,5,8

```
check for k=3
[[ 2.61946868  5.96519477]
[ 6.49724962  7.52297293]
[ 5.55524182  2.18980958]]
21944.1174733
check for k=5
[[ 6.57957643  7.57333595]
[ 7.51004923  2.29128354]
[ 2.8337661   6.9189569 ]
[ 2.25525758  2.94637009]
[ 4.26556254  2.36877056]]
k=5 loss  2693.53143865
```

IV. Code

```
def calculate_distance(u,x):
itshape#itshape print itshape(itshapeuitshape)
itshape#itshape print itshape(itshapexitshape)
x1 = u[0]-x[0]
y1 = u[1]-x[1]
d = (x1**2 + y1**2)**1/2
return d

def find_cl(X, n_clusters, centers):
centroids = centers
a =[]
labels = []
```

```
itshape#itshape for itshape itshape i itshape itshape in
    itshape itshape range itshape (itshape len itshape(
    itshape Xitshape) itshape) itshape:
itshape#itshape itshape itshape labels itshape.
    itshape append itshape(0)
itshape#itshape print itshape('itshape label itshape',
    itshape len itshape(itshape labels itshape) itshape)
itshape#itshape print itshape(itshape labels itshape[0])
while True:
for i in range(len(X)):
for j in range(n_clusters):
a.append(calculate_distance(X[i],
    centroids[j]))

arr = numpy.array(a)
result = numpy.where(arr == numpy.amin(
    arr))

labels.append(result[0][0])
a.clear()

labels_new = np.array(labels)
itshape#itshape print itshape(itshape labels_new itshape)
new_centroids = np.array([X[labels_new ==
    i].mean(0)
for i in range(n_clusters)])

if np.all(centroids == new_centroids):
break
labels.clear()
centroids = new_centroids

return centroids, labels

def find_loss(X, n_clusters, labels,
    centroids):
loss = 0
for j in range(len(centroids)):
for i in range(len(X)):
if (labels[i] == j) :
loss = loss + (calculate_distance(X[i],
    centroids[j]))**2


return loss




print('check for k=3')
centers, labels = find_cl(data, 3,
    i_point1)
print(centers)
loss = find_loss(data,3,labels,i_point1)
print(loss)

print('check for k=5')
centers, labels = find_cl(data, 5,
    i_point2)
print(centers)
loss = find_loss(data,4,labels,i_point2)

print ('k=5 loss ',loss)
```