Suhwan Choi
804506539
06/05/2019
Math156

Help from
https://imaddabbura.github.io/post/kmeans_clustering/
https://www.cs.princeton.edu/courses/archive/spring12/cos598C/svdchapter.pdf
https://web.stanford.edu/class/ee378b/ee378b.html
https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html

a)

```
import numpy as np
import itertools
import matplotlib.pyplot as plt
import pandas as pd

n = 10000
d = 10
k = 3
trials = 10

# X: inputs to be clustered, with n-by-d size
# k: number of clusters
# epsilon - convergence criterion
# C: centroids
def KMeans(X, k, initial_label, epsilon) :
    n, d = X.shape
    new_label = initial_label
    C = np.zeros((k,d)) # starts with empty centroids
    L_new = -1 # starts with negative loss
    while True :

        label = new_label
        L = L_new

        # obtain new centroid for each cluster
        for j in range(k) :
            index_label = (label == j)
            if any(index_label) :
                C[j,:] = np.mean(X[index_label,:],0)

        #assign labels
        norm_C= np.dot(np.ones((n,1)),np.reshape(np.power(np.linalg.norm(C,axis =
1),2),(1,k)))
        objective_function = -2*(np.dot(X,np.transpose(C))) + norm_C #objective function
```

```
        new_label = np.argmin(objective_function,axis=1)#cluster that gives smallest loss

        # obtain loss
        sum_norm_x = np.power(np.linalg.norm(X),2)
        L_new = (sum([objective_function[i,new_label[i]] for i in range (n)]) + sum_norm_x) / n

        # check if the new change is really small
        if (L_new > (L - epsilon)) and (L != -1) :
            break
    return new_label, L_new, C

# k: number of clusters
# target_label: true label
# cluster_error: error in clustering
def ClusterError(k,target_label,label) :
    min_error = 1000 # initialize error big enough to be replaced later

    for permutation in itertools.permutations(range(k)) :
        label_permuted = [permutation[i] for i in label]
        cur_error = np.mean([target_label[i] != label_permuted[i] for i in
range(len(target_label))])
        if cur_error < min_error :
            min_error = cur_error
    cluster_error = min_error
    return cluster_error

s_values = np.linspace(0.5,10,20)
s_length = len(s_values)
Loss = np.zeros((s_length,trials))
Error = np.zeros((s_length,trials))
Loss_lower = np.zeros((s_length,trials))
for i in range(s_length) :
    s = s_values[i]
    for j in range(trials) :

        # make X to be clustered and randomly assign each to a cluster
        X = np.random.randn(n,d)
        target_label = np.random.randint(k,size=n)

        #separate normally generated X by s
        for p in range(n) :
            X[p,target_label[p]] += s

        initial_label = np.random.randint(k,size=n)
        epsilon = 1/1000000
        label, L, C = KMeans(X,k,initial_label,epsilon)
```

```
    Loss[i,j] = L
    Error[i,j] = ClusterError(k,target_label,label)
    Loss_lower[i,j] = LowerBound(X,k)

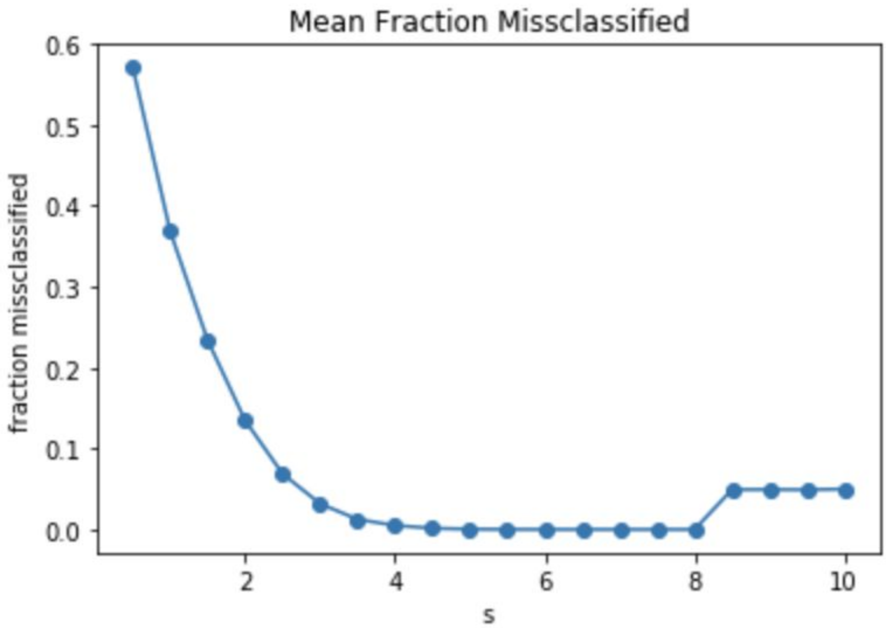
mean_Error = np.mean(Error,1)#average fraction of misclassified points across 10 trials

data = [s_values, mean_Error]
df = pd.DataFrame(data)
```

0.5000	1.00000	1.50000	2.00000	2.50000	3.00000	3.50000	4.00000	4.50000	5.00000	5.5000	6.0	6.50000	7.0	7.5	8.0	8.50000	9.00000	9.50000	10.00000
0.5709	0.36806	0.23365	0.13466	0.06873	0.03186	0.01249	0.00496	0.00146	0.00037	0.0001	0.0	0.00001	0.0	0.0	0.0	0.04922	0.04935	0.04905	0.04992

- First row: s-values
- Second row: mean fraction of misclassified points across 10 trials

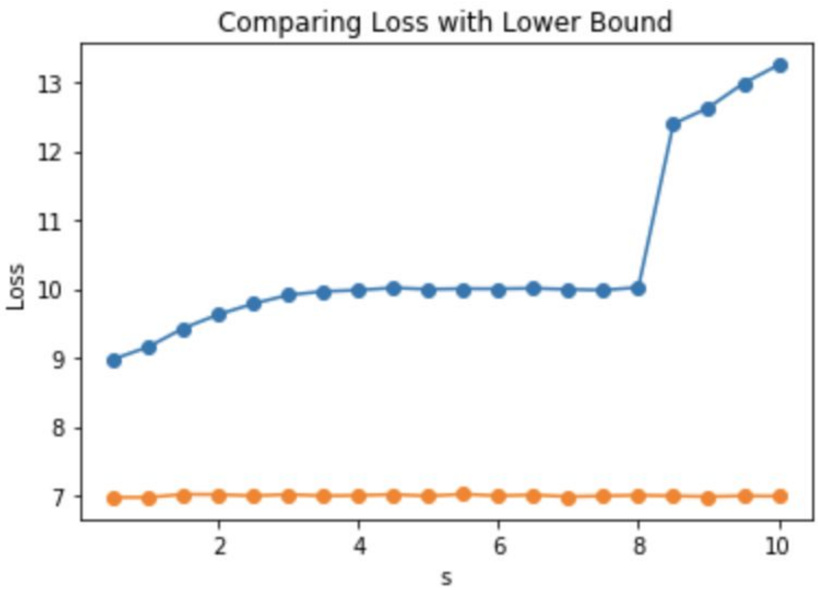
```
plt.plot(s_values,mean_Error,'o-')
plt.xlabel('s')
plt.ylabel('fraction missclassified')
plt.title('Mean Fraction Missclassified')
```

- 

    - As s increases, mean fraction of misclassified points decreases because the larger the s, the further away each cluster is from one another.

b)

```
mean_Loss = np.mean(Loss,1)
mean_Loss_lower = np.mean(Loss_lower,1)
print('mean loss:')
print(mean_Loss)
print('mean loss lower bound:')
print(mean_Loss_lower)
plt.plot(s_values,mean_Loss,'o-',label='KMeans loss')
plt.plot(s_values,mean_Loss_lower,'o-',label='loss lower bound')
plt.xlabel('s')
plt.ylabel('Loss')
plt.title('Comparing Loss with Lower Bound')
```

- ```
  mean loss:
  [ 8.97919841  9.15827101  9.42876641  9.62839245  9.78441675  9.9116359
    9.96285239  9.98161705 10.01489325  9.99406921 10.00281104 10.00016046
   10.00823404  9.99262221  9.98150428 10.02077967 12.39654879 12.63145367
   12.98420125 13.25136253]
  mean loss lower bound:
  [6.97160667 6.97193698 7.01658451 7.01311749 6.99468813 7.01390484
   6.99437514 7.00005503 7.0143569  6.99118212 7.01887834 6.99458337
   7.00775472 6.98255997 6.99287412 7.00285336 6.99299301 6.98312713
   6.99307729 6.98902106]
  ```
- 

- The lower bound values are consistently around 7 across all s, whereas mean loss goes up for s= 7~10. The high mean loss at s=7~10 may be due to some centroids combining together.
  ```

c)

```
data = np.genfromtxt('seeds_dataset.txt',delimiter='\t')
n,d = data.shape
d = d - 1
X = data[:,:d]
target_label = data[:,d] - 1
trials = 10
k = 3
epsilon = 1/1000000
Loss = np.zeros(trials)
Error = np.zeros(trials)
for j in range(trials) :
    initial_label = np.random.randint(k,size=n)
    label,L,C = KMeans(X,k,initial_label,epsilon)
    Loss[j] = L
    Error[j] = ClusterError(k,target_label,label)
Loss_lower_bound = LowerBound(X,k)

# pre-process the data by normalizing the data with mean 0 and std 1.
X_normalized = np.zeros((n,d))
for i in range(d) :
    X_normalized[:,i] = (X[:,i] - np.mean(X[:,i])) / np.std(X[:,i])

LossNormal = np.zeros(trials)
ErrorNormal = np.zeros(trials)
for j in range(trials) :
    initial_label = np.random.randint(k,size=n)
    label,L,C = KMeans(X_normalized,k,initial_label,epsilon)
    LossNormal[j] = L
    ErrorNormal[j] = ClusterError(k,target_label,label)
Loss_lower_boundNormal = LowerBound(X,k)



data = [Loss, Error, Error*n]
df = pd.DataFrame(data)
df
```

2.803724	2.796755	2.796755	2.803724	2.796755	2.796755	2.803724	2.796755	2.796755	2.803724
0.109524	0.104762	0.104762	0.109524	0.104762	0.104762	0.109524	0.104762	0.104762	0.109524
23.000000	22.000000	22.000000	23.000000	22.000000	22.000000	23.000000	22.000000	22.000000	23.000000

-
    - First row: Loss
    - Second row: mean fraction of misclassified points
    - Number of misclassified points

dataNormal = [LossNormal, ErrorNormal, ErrorNormal*n]
dfNormal = pd.DataFrame(data)
dfNormal

2.051509	2.050757	2.050757	2.050757	2.052993	2.050757	2.050757	2.053184	2.051509	2.051930
0.066667	0.080952	0.080952	0.080952	0.080952	0.080952	0.080952	0.061905	0.066667	0.071429
14.000000	17.000000	17.000000	17.000000	17.000000	17.000000	17.000000	13.000000	14.000000	15.000000

-
    - First row: Loss
    - Second row: mean fraction of misclassified points
    - Number of misclassified points

Preprocessing data by normalization seems to improve the clustering. For preprocessed data, 16.6 points were misclassified on average, whereas 22.6 points were misclassified on average.