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>

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)



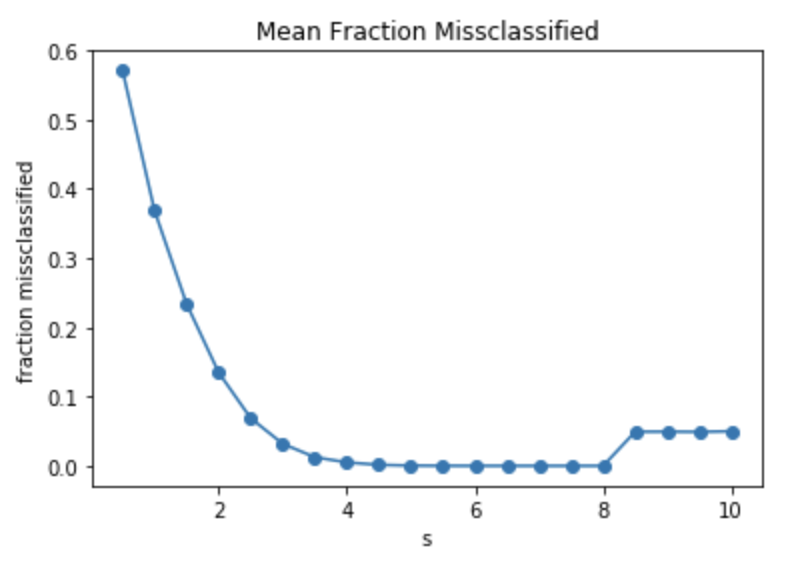
* First row: s-values
* Second row: mean fraction of misclassified points across 10 trials

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)

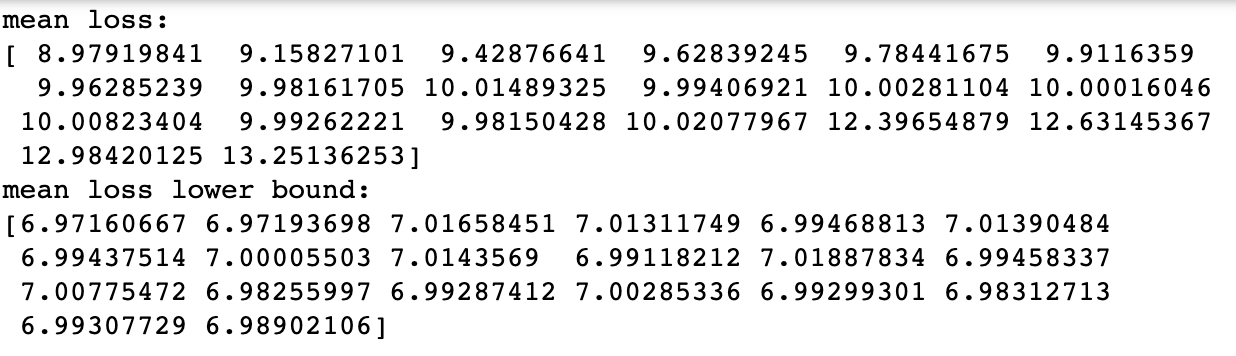
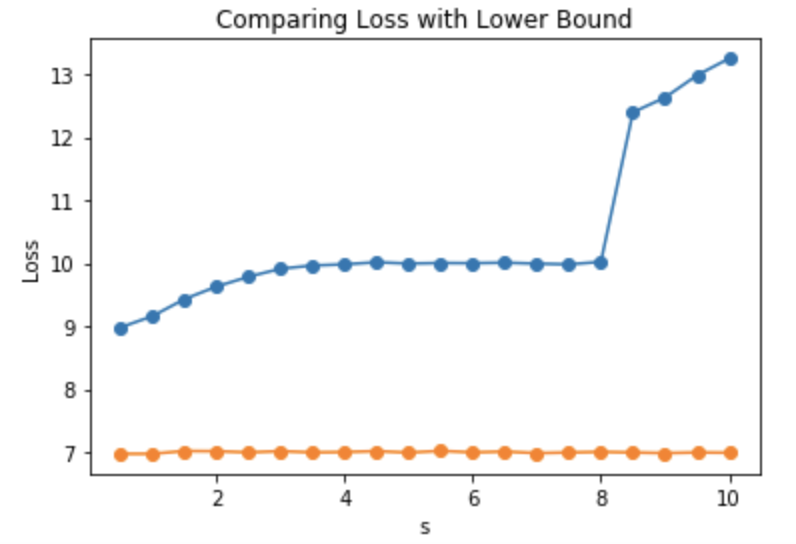
plt.plot(s\_values,mean\_Loss,'o-',label='KMeans loss')

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')

* 
* 
* 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

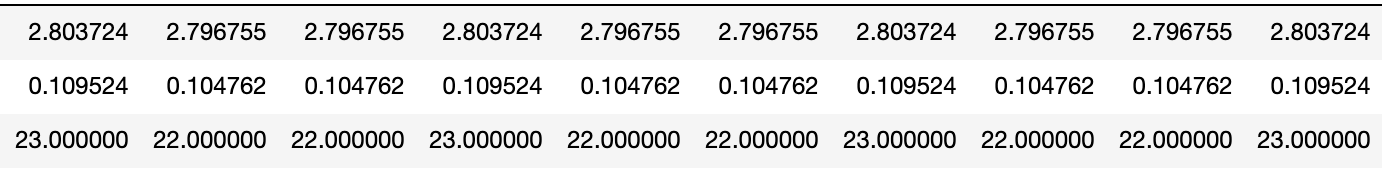
ErrorNormal[j] = ClusterError(k,target\_label,label)

Loss\_lower\_boundNormal = LowerBound(X,k)

data = [Loss, Error, Error\*n]

df = pd.DataFrame(data)

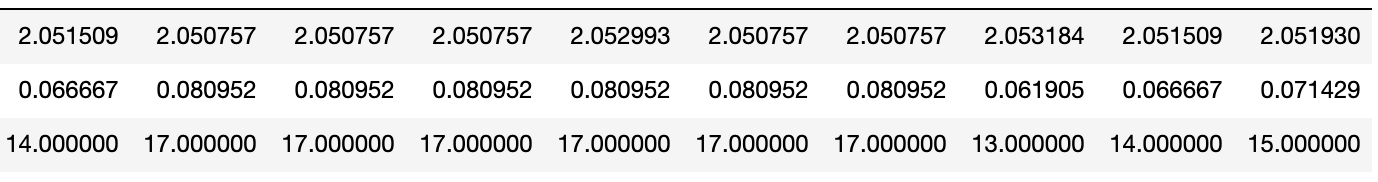
df

* 
  + First row: Loss
  + Second row: mean fraction of misclassified points
  + Number of misclassified points

dataNormal = [LossNormal, ErrorNormal, ErrorNormal\*n]

dfNormal = pd.DataFrame(data)

dfNormal

* 
  + 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.