|  |  |
| --- | --- |
| Lab 2 – Parallel K-means | Name:ID: |

54924670

LOW, Zhi Hao

1. Python scikit-learn package is one of the most popular package in data mining and machine learning.
2. In particular, its k-means function has been widely adopted for data clustering:

<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

1. The following is the example code snippet from GitHub in running k-means:

import numpy as np

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# Creating a sample dataset with 4 clusters

X, y = make\_blobs(n\_samples=100, n\_features=3, centers=4)

# Initializing KMeans with 4 clusters

kmeans = KMeans(n\_clusters=3)

# Fitting with inputs

kmeans = kmeans.fit(X)

# Labelling the clusters

labels = kmeans.predict(X)

print(labels)

# Getting the cluster centers

C = kmeans.cluster\_centers\_

print(C)

1. Fill in the underlined space. Please feel free to try the code in your own machine for experiments.
2. If you read below carefully, you can see that there is an argument called “n\_jobs”. What is it?

<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

<https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/cluster/k_means_.py>

KMeans clustering is available for parallel computing purposes. Therefore to ease our parallelization scikit-learn provided us to api to use multiple threads for faster computation. `n\_jobs` is the number of jobs running in parallel.

1. Unfortunately, the argument “n\_jobs” does not give you the function to parallelize the algorithmic core of K-means which should be straightforward to be parallelized.
2. Therefore, we aim at parallelizing it for big data in this lab.
3. You are given the Python k-means code “Lab2given.py” and data “xclara.csv” in CANVAS.
4. It is observed that there are two code sections which can be parallelized as tabulated below.
5. Please read the documentation of the “joblib” package in Python and use it to parallelize k-means.
6. <https://github.com/joblib/joblib>

|  |  |
| --- | --- |
| Sequential Version | Parallel Version using joblib package |
| #Assigning each value to its closest cluster  for i in range(len(X)):  distances = dist(X[i], C)  cluster = np.argmin(distances)  clusters[i] = cluster | from joblib import Parallel, delayed  def assignment(x,C,clusters,i):  dis­tances = dist(x, C)  clusters[i] = np.argmin(distances)  Parallel(n\_jobs=2)(  delayed(assignment)(X[i],C,clusters,i)  for i in range(len(X))) |
| # Finding the new centroids by taking the average value  for i in range(k):  points = [X[j] for j in range(len(X)) if clusters[j] == i]  C[i] = np.mean(points, axis=0) | from joblib import Parallel, delayed  def centroidcompute(X,i,C):  points = [X[j] for j in range(len(X)) if clusters[j] == i]  C[i] = np.mean(points, axis=0)  Parallel(n\_jobs=2)(  delayed(centroidcompute)(X,i,C)  for i in range(int(k))) |

Hints are given below if you feel difficult:

def assignment(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_):

distances = dist(x, C)

cluster = np.argmin(distances)

ci = cluster

return ci

def centroidcompute(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_):

points = [X[j] for j in range(len(X)) if clusters[j] == i]

return np.mean(points, axis=0)

1. Once you have finished the above, please feel free to try your parallel code on your machine.
2. (Optional) Alternatively, there is also a native python package called “multiprocessing” which could be interesting to try and replace the “joblib” package in Python for parallel data processing.
3. This is the end; please upload this sheet with your answers to the submission system.