CSE-443/543: High Performance Computing

Homework #2

Due: Wed Sept 15 2021 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue Sept 14 2021

Maximum Points: 26

Submission Instructions

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your program compiles successfully, without any warnings or style errors. Ensure you have documented the methods. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload just the 1 source file onto Canvas via the CODE plug-in.

1. The 1 C++ source file developed for this part of the homework.

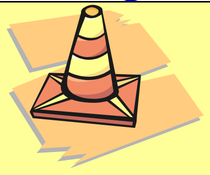
General Note: Upload each file associated with homework individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Objective

The objective of this homework is to develop a K-means clustering program to continue to gain familiarity with:

- Translating pseudocode to a working program
- Working with columnar data in Tab Separated Value (TSV) format
- Learn and work with `std::valarray`
- Working with command-line arguments.

Grading Rubric:

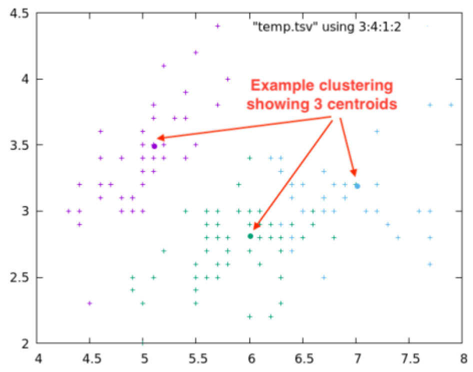


The program submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements or just skeleton code will be assigned zero score! Program that do not compile, **have even 1 method longer than 25 lines**, or just some skeleton code will be assigned zero score.

- Your program should compile without any warnings.
- **NOTE:** Violating CSE programming style guidelines is a compiler error! Your program should not have any style violations reported in `NetBeans` when you compile it.
- **Do not use global variables** – that is not good programming practice.
- **Base case: 5 points**: Print the first *n* columns of data.
- **Additional functionality #1: 12 points**: Perform k means clustering using *n* data points.
- **Additional functionality #2: 3 points**: Stop clustering if the centroids don't change between successive iterations.

- **Screenshot of generated plot**: **3 points**: Include a PNG showing the clustering for `iris.tsv` using the example `gnuplot` command shown further below.
- **Formatting & Documentation: 3 points** (if your comments are poor, then you lose points in this category)
- **Delayed submission: Only 80% points:** Submission delayed **by no more than 24-hours** will be accepted for a partial credit of maximum 80% of the points.

Project overview



K-means clustering is a type of unsupervised machine learning approach, which is used to classify "unlabeled data" – *i.e.*, data without defined categories or groups. The *k*-means clustering aims to partition observations into *k* clusters in which each observation belongs to the cluster with the nearest cluster centroid. The algorithm starts with *k* randomly chosen centroids and refines the position of the centroids based on the data. The value of *k* is a hyperparameter that must be chosen based on the data. Typically, different values are explored to determine suitable *k*. The algorithm is described via the pseudocode below:

```
algorithm kmeans is
    input: Points[0, x]: Array of Point,
           # Point is n-dimensional with coordinates ($q_1$, $q_2$, …, $q_n$)
           number of centroids k ($k \geq 0$),
           max iterations reps = 100
    output: List of centroids and centroid–data-points assignment

    # Initialize array of centroids
    centroids[0, k] = randomly select k data-points from n inputs.

    # Centroid assignment for each data-point
    centIdx[1, x] ← 0
    rep ← 0
    while rep < reps do:
        # Find nearest centroid to each data point
        for i in 0 to x do:
            # Find nearest point to a centroid
            centIdx[i] = index of nearest centroid in centroids[0, k] to Points[i]
        end for
        # Update centroid coordinates based on assignment
        prevCentroids = centroids
        for i in 0 to k do:
            count = |centIdx[z] == i|   # no. of points assigned to centroid i
            centroid[i] = ($\sum$ Points[z], where centIdx[z] == i) ÷ count
        end for
        if prevCentroids == centroids then:
           # No change in centroids. Clustering has converged
           break  # out of while-loop
        end if
    end while
    return {centroids, centIdx}
end algorithm
```

Project requirements

In this project, you are expected to implement the above K-means pseudocode in C++. Inputs to the program will be supplied via the following command-line arguments:

- The 1st command-line argument will always be a valid path to the TSV file to be processed.
- The 2nd command-line argument is *n*, the number of columns in the TSV file to be used as coordinates for a data point. This value must at least be 1. It is 2 for 2D points, 3 for 3D points, etc.
- The 3rd command-line argument is *k*, the number of centroids to be used for clustering. If *k* == 0, no clustering is done and the data points read from the file are simply printed. If *k* > 0, then the specified number of centroids are used for clustering.

TSV file assumptions: The Tab Separated Values (TSV) file will contain 1 or more columns of data. Each column is separated by a tab (`'\t'`) character. Empty lines or lines starting with `'#'` in the TSV file must be ignored. Note that reading the first *n* columns from the TSV is relatively straightforward. So don't over complicate it.

Starter code: To aid with this project you are already supplied with data files and starter code in the **`/fs/ess/PMIU0184/cse443/homeworks/homework2`** directory. Ensure you familiarize yourself with the methods in `Kmeans.h` and `KmeanHelper.cpp`.

Output format: The outputs must be generated by calling the supplied `writeResults` method in the starter code. **Note**: due to rounding errors, your centroids may not exactly match the expected output, but they should be close. You can plot a graph from the output data using the following command in a terminal in VS-Code:

```
$ ./kmeans iris.tsv 4 4 > temp.tsv
$ gnuplot -e 'set terminal png; set output "temp.png"; plot "temp.tsv" using 3:4:1:2 with
points pt var lc var;'
```

Sample outputs

You can test and debug your program from VS-Code, by setting command-line argument as demonstrated in the `OSC OnDemand & VS-Code demonstrations` page on Canvas. See video titled `Using command-line arguments in VS-Code`.

Base case #1 (load & print *n* columns) – must function correctly to earn any points:

```
$ ./kmeans iris_10.tsv 2 0
#PointType      CentroidIndex   Coordinates
1       -1      4.9     3.1
1       -1      5.1     3.3
1       -1      6.7     3.1
1       -1      6.3     3.4
1       -1      5.1     3.8
```

```
1       -1      5.6     2.7
1       -1      6       2.2
1       -1      5       3
1       -1      5.5     2.4
1       -1      5.1     3.4
# Total distance measure: -1
```

Base case #2 (load & print *n* columns) – must work to earn any points:

```
$ ./kmeans iris_10.tsv 4 0
#PointType      CentroidIndex   Coordinates
1       -1      4.9     3.1     1.5     0.1
1       -1      5.1     3.3     1.7     0.5
1       -1      6.7     3.1     4.7     1.5
1       -1      6.3     3.4     5.6     2.4
1       -1      5.1     3.8     1.9     0.4
1       -1      5.6     2.7     4.2     1.3
1       -1      6       2.2     5       1.5
1       -1      5       3       1.6     0.2
1       -1      5.5     2.4     3.7     1
1       -1      5.1     3.4     1.5     0.2
# Total distance measure: -1
```

Additional feature #1 (clustering) – must work for graduate students only to earn any points.
Note: due to rounding errors your centroids may be slightly different in decimal places.

```
$ ./kmeans iris_10.tsv 4 2
#PointType      CentroidIndex   Coordinates
1       0       4.9     3.1     1.5     0.1
1       0       5.1     3.3     1.7     0.5
1       1       6.7     3.1     4.7     1.5
1       1       6.3     3.4     5.6     2.4
1       0       5.1     3.8     1.9     0.4
1       1       5.6     2.7     4.2     1.3
1       1       6       2.2     5       1.5
1       0       5       3       1.6     0.2
1       1       5.5     2.4     3.7     1
1       0       5.1     3.4     1.5     0.2
7       0       6.02    3.94    1.94    0.3
7       1       7       3.38    4.94    1.56
# Total distance measure: 13.4695
```

Additional feature #2 (stop if clustering does not change) – There isn't a good test case for this. Use the previous test and the debugger to ensure this feature is operating as expected.

Turn-in:

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure you have tested operations of your program with different test files and command-line arguments. Once you have tested your implementation, upload just your source code to Canvas.