CSC221 Data Structures & Algorithms I Fall 2016

Suggested Organization for K-Means Clustering

A reasonable way to organize your code is to use five files:

- `dl_list.h`
- `dl_list.cpp`
- `kmeans.h`
- `kmeans.cpp`
- `main.cpp`

Implementing Doubly Linked Lists

The files `dl_list.h` and `dl_list.cpp` are used to implement doubly linked lists representing the clusters. Each list cell will contain one observation. Your `dl_list.h` file might resemble the following:

```
const int NOBS = 4 ;

class listcell {
    private:
       int observation_number     ;
       float measurements[ NOBS ] ;
       listcell * prev ;
       listcell * next ;
    public:
       listcell() ; // Constructor. -- initialize an empty cell.
       ~listcell() { /* Do nothing. */  }

       // Other public methods go here.
       // Relatively few are needed.

       friend class dl_list ;
} ;

class dl_list {
    private:
       listcell * head ;
       listcell * tail ;
       int length      ;
    public:
       dl_list() ;  // Constructor -- initialize an empty list.
       ~dl_list() { /* Do nothing. */  }

       // Other public methods go here.
       // Choice of methods is indicated by
       // services needed by class kmeans.

} ;
```

Services (public methods) Needed by Class kmeans

Services (public methods) provided by Class listcell

get_obs Using pass-by-reference, pass back the observation number and the array of four measurements.

Services (public methods) provided by Class dl_list

first Returns a `listcell` pointer to the first cell on the list.

next Accepts a pointer to the current cell and returns a pointer to the next cell.

append Accepts an observation number and an array (length four) of measurements. Appends a new list cell containing the given observation number and measurements to the end of the list.

remove Accepts a pointer to a list cell and removes it from the list. It also frees the memory associated with the removed list cell.

compute_mean Accepts an an array length four. Fills in the array with the value of the mean for this list.

save_obs_nums Accepts an a file name and writes the observation numbers on the list to the file. This method makes it very easy to implement method `output_clusters` in class `kmeans`.

Implementing K-means Algorithm

The files `kmeans.h` and `kmeans.cpp` are used to implement the K-means clustering algorithm. Your `kmeans.h` file might resemble the following:

```
class kmeans {
    private:
         dl_list A ;
         dl_list B ;
         float meanA[NOBS] ;
         float meanB[NOBS] ;
    public:
        kmeans() ; // Constructor
        ~kmeans() {  /* Do nothing. */ }

       // Other public methods go here.
       // Choice of methods is indicated by
       // services needed by main().

} ;
```

Services Needed by main()

Services (public methods) provided by Class kmeans

read_file Class method **read_file** should accept the name of the data file. It reads the file and puts the observations on lists `A` and `B`.

do_clustering Performs K-means clustering algorithm on the input data.

output_clusters Writes two files named `A.txt` and `B.txt` containing the observation numbers (one per line) of the data items in cluster `A` and cluster `B` respectively.

Private methods in Class kmeans

Suggestion: Include a private method in class `kmeans` to compute the Euclidean distance between two arrays length four. This allows you to efficiently compute the distance between an observation, and `meanA` and `meanB`.

Pseudocode for the K-means Algorithm

```
Read the data file and assign each observation randomly
     to either list A or list B
Select two observations at random to serve as initial means ;
// Call them 'meanA' and 'meanB' ;
do {
     change = false ;

     Scan list A.  For each observation X on list A, compute the
     Euclidean distance from X to meanA and from X to meanB.
     If X is closer to meanB than meanA, then {
             Remove X from list A ;
             Add X to list B ;
             change = true   ;
     }

     Scan list B.  For each observation Y on list B, compute the
     Euclidean distance from Y to meanA and from Y to meanB.
     If Y is closer to meanA than meanB, then {
             Remove Y from list B ;
             Add Y to list A ;
             change = true   ;
     }

     if (change) recompute meanA and meanB ;

while ( change ) ;

Output list A to file 'A.txt' ;
Output list B to file 'B.txt' ;
```