Final Exam: k-NN

Author: Gabriel Hofer

Course: CSC-410 Parallel Computing

Instructor: Dr. Karlsson

Due: December 7, 2020

Computer Science and Engineering
South Dakota School of Mines and Technology

Design

The data and task division

Description of local and global communication

The description of the larger combined tasks

The description of the mapping

Program Documentation

How to compile and run

Important specification: the number of processors (nprocs) must be equal to one more than the number of records in the data-file due to how data is divided amongst tasks.

Run the program using the Makefile in the root of the submission.

```
$ make small_reg
$ make small_mode
```

This is the format for how command line parameters read by program:

```
mpirun -np  <nprocs>  <data-file>  <query-file>  k  <[r|m]>
```

Implementation Description

Below are the steps in the k-NN algorithm (pseudo-code from the writeup). Code listings from the program are used to help explain how each step in the algorithm is implemented.

```
a. Load the data
```

The name of the data-file is read from argv[1]. records are read from the data-file using getline until EOF. Fields in the records are expected to be comma-separated. Therefore, we use sscanf for parsing the fields.

```
if(myproc==0){
  FILE *fp = fopen(argv[1],"r");
  while(getline(&buf, &leng, fp)>0){
    sscanf(buf, "%f %[,] %f %[,] %f %[,] %f %[,] %s", &a, &ch, &b, &ch, &c, &ch, &d, &ch,
        ↪ e);
    data[plant_idx].attr[0]=a;
    data[plant_idx].attr[1]=b;
    data[plant_idx].attr[2]=c;
    data[plant_idx].attr[3]=d;
    strcpy(data[plant_idx].class,e);
    plant_idx++;
  }
  data_size=plant_idx;
  fclose(fp);
}
```

```
b. Initialize k to your chosen number of neighbors
```

```
int K=atoi(argv[3]);
```

c. For each example in the data
i. Calculate the distance between the query example and the current example from the data.

This is the manhattan distance function.

```
float dist(float d[], float q[]){
  return fabs(d[0]-q[0]) + fabs(d[1]-q[1]) + fabs(d[2]-q[2]) + fabs(d[3]-q[3]);
}
```

Each process¿0 calls dist to calculate the manhattan distance.

```
float manhattan=-1;
if(myproc>0) { manhattan=dist(attr, qattr); }
MPI_Barrier(MPI_COMM_WORLD);
```

ii. Add the distance and the index of the example to an ordered collection.

Each process sends the distance back to the root process.

```
float arr[1000][2];
if(myproc==0){
  for(int i=1;i<nprocs;i++){ MPI_Irecv(&arr[i-1], 2, MPI_FLOAT, i, 0, MPI_COMM_WORLD, &
      ↪ request[i]); }
} else { MPI_Isend(&man_proc, 2, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, &request[0]); }
MPI_Barrier(MPI_COMM_WORLD);
```

d. Sort the ordered collection of distances and indices from smallest to largest (in ascending

```
int cmp(const void * a, const void * b){ return ((float *)a)[0] > ((float *)b)[0]; }
qsort(arr, data_size, 2*sizeof(float), cmp);
```

e. Pick the first k entries from the sorted collection
f. Get the labels of the selected K entries
g. If regression, return the mean of the k labels

```
if(0==strcmp(argv[4],"r")){
  /* regression */
  /* iterate through the closest K vectors and calculate the arith. mean of attributes */
  for(int i=0;i<K;i++){
    rattr[0]+=data[(int)arr[i][1]].attr[0]/(float)K;
    rattr[1]+=data[(int)arr[i][1]].attr[1]/(float)K;
    rattr[2]+=data[(int)arr[i][1]].attr[2]/(float)K;
    rattr[3]+=data[(int)arr[i][1]].attr[3]/(float)K;
  }
  printf("MEAN: %f %f %f %f\n", rattr[0], rattr[1], rattr[2], rattr[3]);
}
```

h. If classification, return the mode of the k labels

```
else if(0==strcmp(argv[4],"m")){
  /* iterate through the closest K vectors and count the frequency of each class of plant
     ↪ */
  int Iris_virginica=0;
  int Iris_versicolor=0;
  int Iris_setosa=0;
  for(int i=0;i<K;i++){
    if(strcmp(data[(int)arr[i][1]].class,"Iris-virginica")==0)
      Iris_virginica+=1;
    if(strcmp(data[(int)arr[i][1]].class,"Iris-versicolor")==0)
      Iris_versicolor+=1;
    if(strcmp(data[(int)arr[i][1]].class,"Iris-setosa")==0)
      Iris_setosa+=1;
  }
  /* print the counts for the three plant classes */
  printf("iris-virginica: %d\n", Iris_virginica);
  printf("iris-versicolor: %d\n", Iris_versicolor);
  printf("iris-setosa: %d\n", Iris_setosa);
  /* find the mode by finding the class with the max cnt */
  int mode_idx=-1;
  if(Iris_virginica>=Iris_versicolor && Iris_virginica>=Iris_setosa)
    mode_idx=0;
  else if(Iris_versicolor>=Iris_virginica && Iris_versicolor>=Iris_setosa)
    mode_idx=1;
  else if(Iris_setosa>=Iris_virginica && Iris_setosa>=Iris_versicolor)
    mode_idx=2;
  /* print the mode */
  if(mode_idx==0) printf("MODE: Iris-virginica\n");
  if(mode_idx==1) printf("MODE: Iris-versicolor\n");
  if(mode_idx==2) printf("MODE: Iris-setosa\n");
}
```