CPSC Project 2 MPI KNN

a. Project 2 submission
   i. Johnson Tong jt28@csu.fullerton.edu
   ii. Derrick Lee derricklee@csu.fullerton.edu
   iii. Miguel Macias miguel6021857@csu.fullerton.edu

b.

474-Project2

Project 2: Implement a distributed algorithm using MPI

Group members:

Johnson Tong jt28@csu.fullerton.edu

Derrick Lee derricklee@csu.fullerton.edu

Miguel Macias miguel6021857@csu.fullerton.edu

c. Algorithm Pseudocode
   i. Data is read into program, function parser(), will separate the data into the training data and queries for the KNN algorithm and return it back as a dataset.

```
parser(int queries, int instances, int cols, string filename){
  file = file.open(filename)
  buffer = string
  dataset = dataset()

  while(not end of file){
    for x amount of instances
      for y amount of cols
        dataset.trainingData[x][y] = buffer

    for x amount of queries
      for y amount of cols
        dataset.query[x][y] = buffer

    return dataset
  }
}
```

   ii. Rank 0 finds the number of queries per process, done in function knn()

```
knn(dataset, query, k) {
    // MPI_scatter from rank 0 process
    if rank == 0:
        get num_processes from argv
        split_size = dataset.size()/num_processes
        send dataset splits and query to processes
    else:
        Each process should
            distances = []
            for each datapoint in split
                distances.push(distance(query, datapoint))
            send distances to rank 0

    r = recombine(dataset_splits) // MPI_gather
    sort(r) //sorted in ascending ordre

    classes = map<class, int>
    for class in r[i] = r[:k]
        classes[class]++;
    sort(classes.begin(), classes.end(), cmp_fn)
    // cmp_fn to compare the counts of the elements in classes

    return classes.begin().element
}
```

iii. Rank zero sends each process the set of queries to be process and each process performs KNN and sends the classification values back to be store in a vector

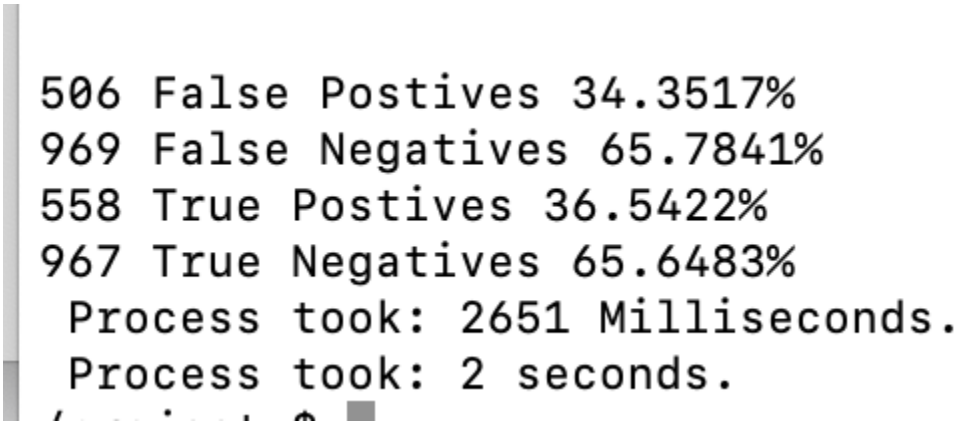
```
if (rank == 0){
    predicted_total[] //Keeps track of recieved splits when condition is met
    receive_split[] // Splits recived from process
    for x in size of processes:
        MPI_Recv(start_from_other_process)
        MPI_Recv(end_from_other_process)

        MPI_Recv(recive_split)

        for y = start_from_other_process, if y <= end_from_other_process, increment y:
            predicted_total[j] = receive_split[x]
}else{
    MPI_Send(start)
    MPI_Send(end)
    MPI_Send(predicted split)
}
```

iv. Rank Zero collects all the data and creates a scoring matrix and displays the time taken,this is done in main()

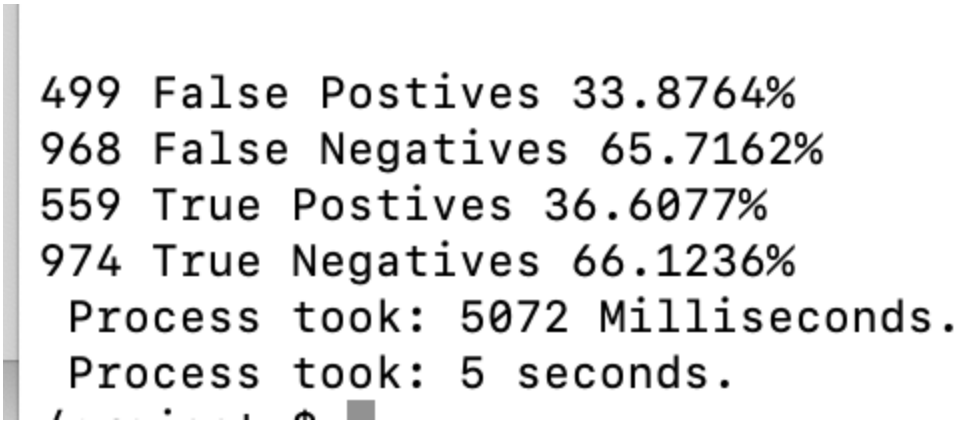
```
MPI_Finalize()

if (rank == 0){
    double true_positive
    double false_positive
    double true_negative
    double false_negative
    int classification = argv

    for x in the number of quieries
        if(query == true_positive):
            true_positive++
        else if (query == false_positive):
            false_positive++
        else if (query == true_negative):
            true_negative++
        else if (query == false_negative):
            false_negative++
}
//Display all results
```

d. In order to run the code type sh r.sh
   i. Alternatively you can change the arguments or run it as follows
   ii. mpic++ -std=c++11 knn.cc -o knn
   iii. mpirun -n 5 ./knn 3000 7000 5 30 testdata.txt
   iv. argv[1] = number of queries, argv[2] = number of training instances argv[3] = number of columns in training instance, argv[4] = how many neighbors argv[5] = name of textfile for training data

e. Snapshot of code executed 3 distinct screenshots
   i. 5 Processes



   ii. 2 Processes



iii. 1 Process