UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Assignment 1 Specifications

Release Date: 31-07-2023 at 06:00

Due Date: 25-08-2023 at 23:59

Total Marks: 608

Contents

1 General Instructions

- *Read the entire assignment thoroughly before you start coding.*
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding cpp file. No inline implementation apart from the provided functions.

2 Overview

For this assignment, you will be implementing a K-Nearest Neighbours (KNN) algorithm in c++. This algorithm will be used to demonstrate how to handle dynamic memory, and how to correctly use classes. The algorithm will also be used to illustrate the different ways function signatures can be designed in a class to allow for different functionality. In this assignment, you will gain experience with semi-complex class structures and pointers.

3 Background

3.1 KNN

K-Nearest Neighbours (KNN) is a non-parametric supervised clustering algorithm. The KNN algorithm aims to group or cluster a set of data points into K distinct groups. To achieve this, KNN uses centroids, which act as the center of the clusters to group the data points. Through a series of iterations, these centroids are moved around the data, such that at the end of each iteration, the centroid is in the centre of all the data points that belong to that centroid. To determine if the data point belongs to a centroid, a distance function is used and the data point assigned to the centroid that is the closest to itself. Algorithm 3.1 describes the main, abstracted, algorithm of KNN:

```
KNN(DPS, centroids, numIterations):
    for each iteration:
        for each dp in DPS:
            assign dp to the centroid that is the closest.

        for each centroid in centroids:
            centroid = average of all the data points assigned to this centroid.
```

Algorithm 3.1: KNN Pseudo code

In Algorithm 3.1, DPS (Data Point Set) refers to all the data points, centroids are the centroids and numIterations are the numbers of iterations the algorithm will be allowed to run.

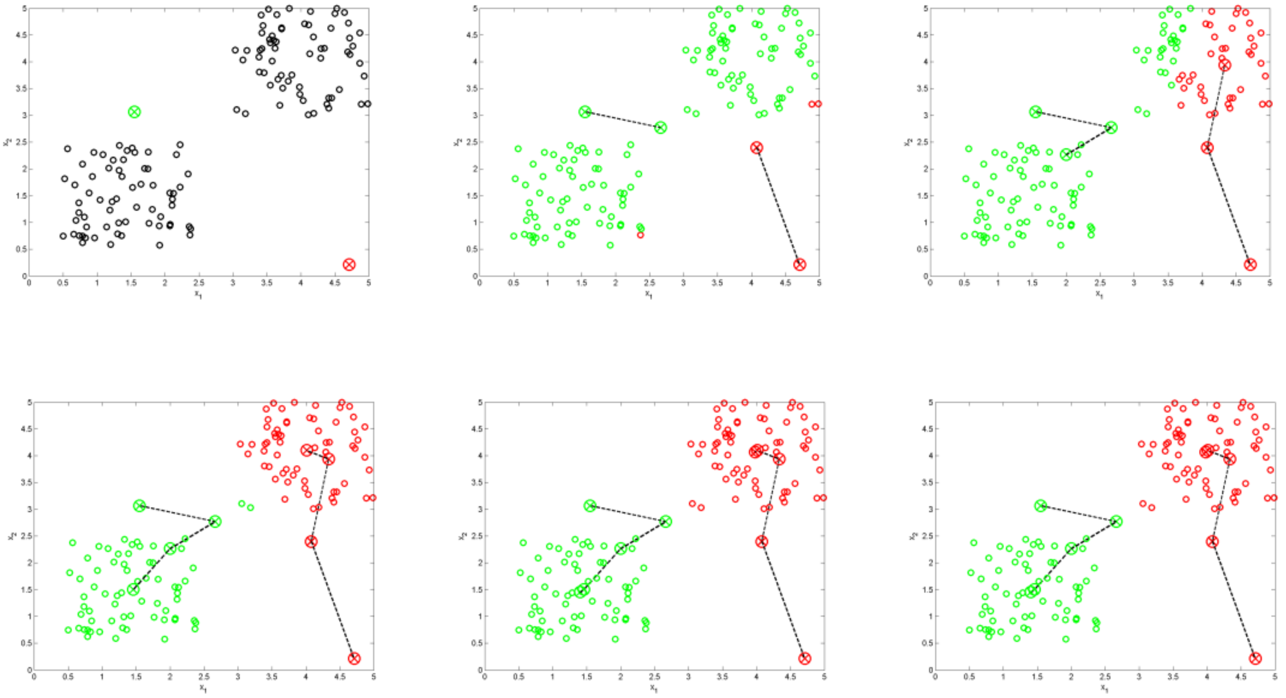

Figure 1: KNN example

Figure 1 is a visual example of the KNN algorithm, that you will be implementing. As can be seen, the green and red circles with the crosses inside, represent the centroids. With each iteration, (from left to right) the data points are assigned a centroid and the centroid moves to the centre of the data points assigned to it. The black tick line indicates the path that the centroid followed, from start to end.

Different distance functions result in different results. For this assignment, you will be implementing three *different* distance functions. KNN is a popular clustering algorithm as it is easy to implement and can easily be configured to work on data of different dimensionalities.

An attribute of a data point is something that is used to describe the data point. For example, if we have a data point in 3D space $< x, y, z >$ then the data point's attributes are x, y and z. Another example would be to describe a cat data point as follows: (quadrille-pedal, ginger, female, mommy). In this example the first attribute refers to the amount of legs the specific cat has. The second attribute specifies the fur colour, in this case ginger. The third attribute specifies the sex of the cat, female for the example data point. The last attribute describes the family hierarchical positioning of the cat, i.e. it is a mommy cat.

4 Your Task:

You are required to implement the following classes. Pay close attention to the function signatures as the H files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

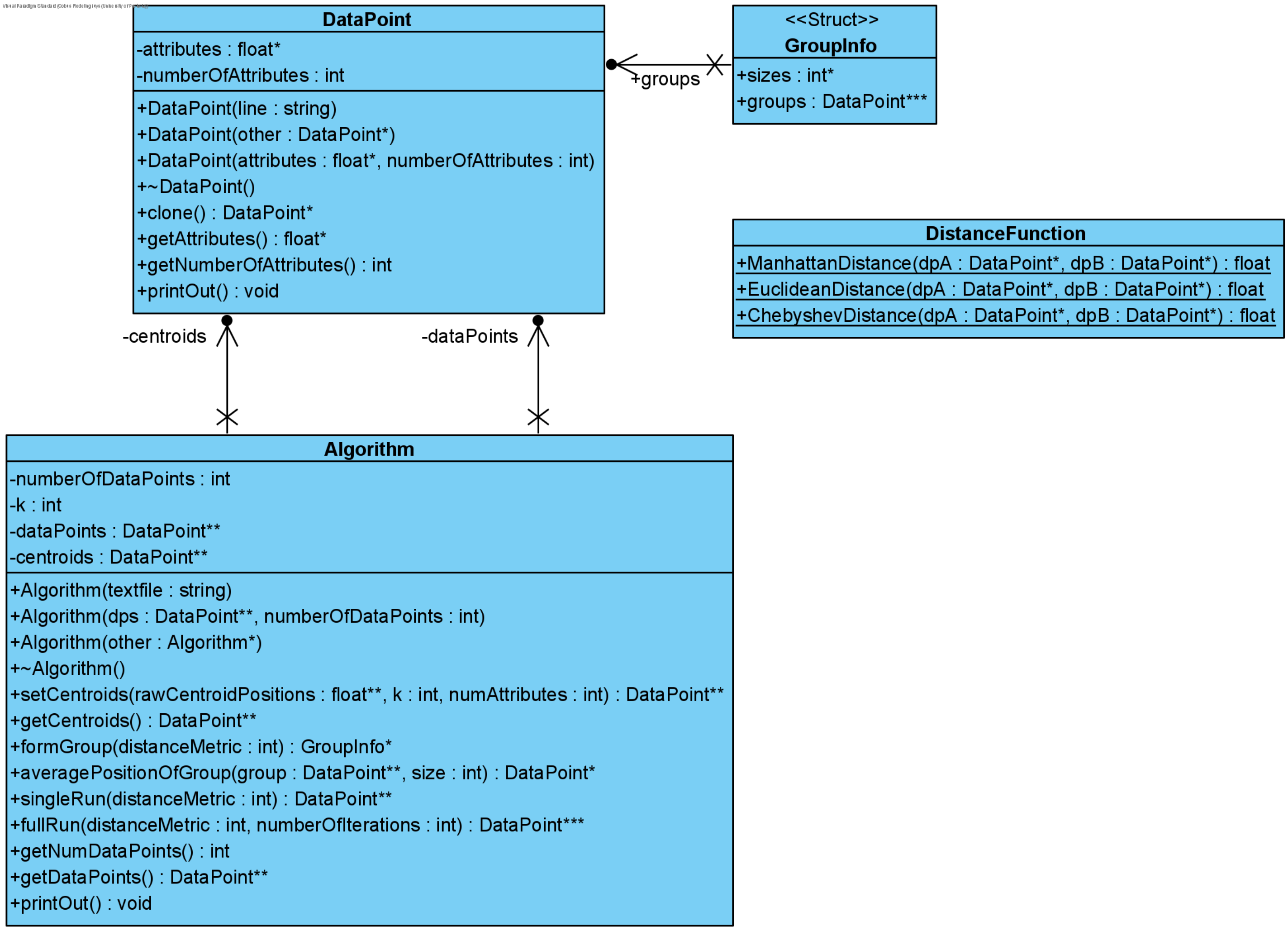
DataPoint
-attributes : float*
-numberOfAttributes : int
+DataPoint(line : string)
+DataPoint(other : DataPoint*)
+DataPoint(attributes : float*, numberOfAttributes : int)
+~DataPoint()
+clone() : DataPoint*
+getAttributes() : float*
+getNumberOfAttributes() : int
+printOut() : void
+groups
<<Struct>>
GroupInfo
+sizes : int*
+groups : DataPoint***
DistanceFunction
+ManhattanDistance(dpA : DataPoint*, dpB : DataPoint*) : float
+EuclideanDistance(dpA : DataPoint*, dpB : DataPoint*) : float
+ChebyshevDistance(dpA : DataPoint*, dpB : DataPoint*) : float
-centroids
-dataPoints
Algorithm
-numberOfDataPoints : int
-k : int
-dataPoints : DataPoint**
-centroids : DataPoint**
+Algorithm(textfile : string)
+Algorithm(dps : DataPoint**, numberOfDataPoints : int)
+Algorithm(other : Algorithm*)
+~Algorithm()
+setCentroids(rawCentroidPositions : float**, k : int, numAttributes : int) : DataPoint**
+getCentroids() : DataPoint**
+formGroup(distanceMetric : int) : GroupInfo*
+averagePositionOfGroup(group : DataPoint**, size : int) : DataPoint*
+singleRun(distanceMetric : int) : DataPoint**
+fullRun(distanceMetric : int, numberOfIterations : int) : DataPoint***
+getNumDataPoints() : int
+getDataPoints() : DataPoint**
+printOut() : void

Figure 2: Class diagrams

4.1 DataPoint

This class will represent a single data point.

- Members:
  - attributes: float*
    - This is a 1D dynamic array of floats, which will represent the different attributes[1] of a data point.
  - numberOfAttributes: int
    - This is the number of attributes in the attributes array.
    - This can also be interpreted as the dimensionality of the data point.
- Functions:
  - DataPoint(line: string)
    - This is the first parameterized constructor for the DataPoint class.
    - This constructor takes in a parameter called line of type string, that needs to be converted to a data point.
    - The following can be assumed for the line:
      - The format of a data point is
        $$(a, b, ..., x, y)$$
        where a,b,...,x,y represent the floating point values. The values will be separated by a comma and enclosed in round brackets.
      - Between each value, there can be whitespaces in the form of spaces and tabs (\t)
      - A line will only contain at most a single data point.
      - A line can be a string filled with white space and no actual data point.
      - A line can also be an empty data point, in other words just two round brackets filled with whitespace or no round brackets at all.
      - It can be assumed that each attribute in the data point can be converted to float.
    - Examples of valid lines are:
      - (2.5,3.6,2.7)
      - (0.5, 4.2)
      - (-1.433,2.3, 0.03, 5, 2.7)
      - Blank line i.e. ""
      - ( )
    - The constructor will need to determine how many attributes there are, extract the attributes and store it in the array.
    - The size of the array needs to be determined during runtime (therefore it is a dynamic array).

[1] An attribute is an ordered "feature/descriptor" of the data point.

    - If the line is blank, then the attributes array needs to be sized to have a size of 0, and the numberOfAttributes assigned accordingly.
    - If the line is an empty data point, then the attributes array needs to be sized to have a size of 0, and the numberOfAttributes assigned accordingly.

- DataPoint(other: DataPoint*)
    - This will be the copy constructor for the DataPoint class.
    - This constructor should create a deep copy of the passed in parameter.
    - If the passed in parameter is NULL, then the attributes array needs to be sized to have a size of 0, and the numberOfAttributes assigned accordingly.
- DataPoint(attributes: float*, numberOfAttributes: int)
    - This is the second parameterized constructor for the DataPoint class.
    - This constructor will need to create a deep copy of the passed-in array.
    - If the array is NULL, or the numberOfAttributes is negative, then the attributes array needs to be sized to have a size of 0, and the numberOfAttributes assigned accordingly.
- ~DataPoint()
    - This is the destructor for the DataPoint class.
    - This function should deallocate all the dynamic memory assigned to the DataPoint object.
    - *Hint: after deallocating memory, assign the pointer to have the value of NULL.*
- clone(): DataPoint*
    - This is an alternative way to create a deep copy of an object.
    - This function needs to return a deep copy of the current Data Point object.
- getAttributes(): float*
    - This function will return the attributes array.
- getNumberOfAttributes(): int
    - This function will return the number of attributes.
- printOut(): void
    - This is a provided function and will be overwritten during the assessment of this assignment.
    - This function can/will be used to print out the data point.

4.2 DistanceFunction

This is the class that will contain the implementation of the distance functions.

- Functions:
  - ManhattanDistance(dpA: DataPoint*, dpB: DataPoint*): float
    - This is a static function.
    - This function should return the Manhattan distance between dpA and dpB.
    - The Manhattan distance can be calculated using:

      $$d(x,y) = \sum_{i=0}^{N}(|x_i - y_i|)$$

      where x and y are the two points and N is the dimensionality of the points.
  - EuclideanDistance(dpA: DataPoint*, dpB: DataPoint*): float
    - This is a static function.
    - This function should return the Euclidean distance between dpA and dpB.
    - The Euclidean distance can be calculated using:

      $$d(x,y) = \sqrt{\sum_{i=0}^{N}((x_i - y_i)^2)}$$

      where x and y are the two points and N is the dimensionality of the points.
  - ChebyshevDistance(dpA: DataPoint*, dpB: DataPoint*): float
    - This is a static function.
    - This function should return the Chebyshev distance between dpA and dpB.
    - The Chebyshev distance can be calculated using:

      $$d(x,y) = max(|x_0 - y_0|, |x_1 - y_1|, ..., |x_N - y_N|)$$

      where x and y are the two points and N is the dimensionality of the points. The max function returns the maximum value that was passed as a parameter.

It should be noted that for all three of the above-mentioned functions, the following conditions hold:

- If dpA is NULL the function should return the c++ constant, INFINITY.
- If dpB is NULL the function should return the c++ constant, INFINITY.
- If dpA's dimensionality is not the same as dpB's dimensionality the function should return the c++ constant, INFINITY.

4.3 GroupInfo

This is a struct that can be found inside the Algorithm.h file.

- Members:
  - sizes: int*
    - This is a 1D array of int values, which represent the sizes for the groups member variable discussed below.
  - groups: DataPoint***
    - This is a 2D array of dynamic DataPoint objects.

4.4 Algorithm

This class represents the main KNN algorithm class.

- Members:
  - numberOfDataPoints: int
    - This is the number of data points that are stored in the dataPoints array.
  - k: int
    - This is the number of centroids that are stored in the centroids array.
  - dataPoints: DataPoint**
    - This is a 1D dynamic array populated with dynamic DataPoint objects.
  - centroids: DataPoint**
    - This is a 1D dynamic array populated with dynamic DataPoint objects.
- Functions:
  - Algorithm(textfile: string)
    - This is the first parameterized constructor of the Algorithm class.
    - This function takes in a string which represents the textfile's name.
    - Your code will need to read through the textfile and populate the dataPoints array and numberOfDataPoints respectively.
    - Your code should size the dataPoints array exactly.
    - It can be assumed that the textfile exists.
    - It can, however, not be assumed that there is only one data point per line. There can be 0, or more data points, per line, in the textfile.
    - It can be assumed they all have the same dimensionality.
    - Your code will need to extract each data point in the textfile and use them to create new DataPoints.

    - If a line is empty, your code should move on to the next line. In other words, should not pass the empty line to the DataPoint constructor.
    - This function should also initialize, k, to be -1, and centroids to be NULL.
    - It should be noted that this constructor will primarily be used in the testing of other functions, so pay special attention when testing this function.
- Algorithm(dps: DataPoint**, numberOfDataPoints: int)
    - This is the second parameterized constructor.
    - This function should make a deep copy of the passed-in array, and set the required member variables.
    - This function should also initialize, k, to be -1, and centroids to be NULL.
    - If dps is NULL, or the numberOfDataPoints is negative, then you should initialize the dataPoints array to have a size of 0, and the numberOfDataPoints accordingly.
- Algorithm(other: Algorithm*)
    - This is the copy constructor for the Algorithm class.
    - This function should make a deep copy of the passed-in parameter.
    - This function should also initialize, k, to be -1, and centroids to be NULL irrespective of the values in the passed-in parameter.
    - If the passed-in parameter is NULL then you should initialize the dataPoints array to have a size of 0 and the numberOfDataPoints respectively.
- ∼Algorithm()
    - This is the destructor for the Algorithm class.
    - The destructor should de-allocate all dynamic memory allocated to the Algorithm object.
    - *Hint: remember that centroids cannot be assumed to be set, whereas dataPoints can be assumed to be set.*
- setCentroids(rawCentroidPostions: float**, k: int, numAttributes: int)
    - This function should populate centroids member variable with DataPoints using the 2D float array. Each row in the passed-in array corresponds to the attributes of the DataPoint.
    - k represents the number of centroids.
    - numAttributes represents the number of attributes there are in the centroids.
    - It can be assumed that numAttributes has the same number of attributes as in a DataPoint in the dataPoints array.
    - If the rawCentroidPositions is NULL, or k is less than 0, then the function should return NULL.
    - Else, the function should return the old value of the centroids member variable.
- getCentroids(): DataPoint**

    - This is a const function [2].
    - This function should return the centroids member variable.
- formGroup(distanceMetric: int): GroupInfo*
    - This function will be used to assign the data points to the centroid.
    - This function will return a GroupInfo pointer.
    - The distanceMetric corresponds to the distance function that will be used. The values correspond as follows:

        0. Manhattan Distance
        1. Euclidean Distance
        2. Chebyshev Distance

    - If the distanceMetric is not in the above list then, the distance can be assumed to be the c++ INFINITY constant.
    - The following pseudo-code can be used to group the data points:

```
for each dataPoint in dataPoints:
    determine which centroid the data point is the closest to
    add it to the first non null position in the array that is assigned to the centroid.
```

Algorithm 4.1: Pseudo code for the formGroup function

-
    - *Hint: The above pseudo code will need to be modified/altered, to determine the size of the array of the dataPoints belonging to each centroid*
    - Populate the GroupInfo pointer as follows:
        - A 2D array of DataPoints where the ith row in the 2D array, corresponds to the data points belonging to the ith centroid in the centroids array
        - The ith index in the int array is the number of data points belonging to the ith centroid.
    - If an invalid distance metric is used as described above, all the data points are assigned to the 1st centroid (0th index in centroids array).
    - It can be assumed that centroids will be set when this function is called.
- averagePositionOfGroup(group: DataPoint**, size: int): DataPoint*
    - This is a const function.
    - This function will find the average position between all the DataPoints in the passed-in array and return a new DataPoint with this position.
    - The size parameter is the number of DataPoints in the group.
    - If the group is NULL, or the size is less than 0, the function should return NULL.
    - It can be assumed that all the DataPoints in the array have the same dimensionality.

[2] If you are unsure what this means please see the following website https://www.geeksforgeeks.org/const-member-functions-c/

- singleRun(distanceMetric: int): DataPoint**
  - This function will complete a single iteration of the KNN main loop.
  - This function should group the DataPoints together based on the nearest centroid, find the average position in the group and update the corresponding centroid with the **new** data points.
  - This function should also de-allocate the old centroids and any other dynamic memory that was created.
  - Lastly, the function should return the newly allocated centroids.
  - If centroids, or k, are NULL or negative, respectively, the function should return NULL.
- fullRun(distanceMetric: int, numberOfIterations: int): DataPoint***
  - This function will perform a full run of the algorithm for a number of iterations, as passed in as a parameter.
  - The function should return a 2D array of DataPoints where the ith row corresponds to the new centroid positions after the ith iteration.
  - *Remember, the singleRun function **de-allocates** the old centroids, so your code needs to compensate for this.*
  - If the centroids are NULL, or the numberOfIterations is less than 0, your code should return NULL.
- getNumDataPoints(): int
  - This is a const function.
  - This function should return the number of data points currently in the object.
- getDataPoints(): DataPoint**
  - This is a const function.
  - This function should return the dataPoints array.
- printOut(): void
  - This is a provided function and will be overwritten during the assessment of this assignment.
  - This function can/will be used to print out all the data points in the algorithm.

5 Memory Management

As memory management is a core part of COS110 and c++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocate *all* the memory that was allocated.

6 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the main.cpp file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [3] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j DataPoint Algorithm DistanceFunction
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark	Final mark
0%-5%	0%	0
5%-20%	20%	10
20%-40%	40%	20
40%-60%	60%	30
60%-80%	80%	40
80%-100%	100%	50

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark.

[3] For more information on gcov please see https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

Remember that your main will be testing the Instructor Provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

7 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using namespace std in any of the files.
- You may only use the following libraries:
  - String
  - CMath
  - IOStream
  - FStream
  - StringStream
- You are supplied with a **trivial** main demonstrating the basic functionality of the assignment.

8 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- DataPoint.{h,cpp}
- DistanceFunction.{h,cpp}
- Algorithm.{h,cpp}

9 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your .h file will be overwritten, so ensure you do not alter the provided .h files.
You have 5 submissions and your best mark will be your final mark. Upload your archive to the Assignment 1 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**