STATISTICAL ANALYSIS OF DATA ACCESSED FROM A DATABASE THROUGH A C++ PROGRAM

YAYUN LIU, DIPANJAN SAHA

A **database** is a collection of data typically organized to model relevant aspects of reality in a way that supports processes requiring this information. An example can be modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies. A general-purpose **database management system** (DBMS) is a software system designed to allow the definition, creation, querying, update, and administration of databases. It interacts with the user, other applications, and the database itself to capture and analyze data.

The old databases were typically **flat** ones, requiring the user to traverse the whole database linearly to search for a particular record. Now-a-days the most popular for search and analysis purposes are **relational** databases - based on the relational model proposed by Edgar Codd in 1969. In the relational model of a database, all data is represented in terms of tuples, grouped into relations. For visualization purposes, the elementary structure that we essentially see is a table where rows represent the different entries for a particular record and columns represent the fields under consideration. For example, if we create a table containing the grades of students enrolled in ECEN489, the columns may be "UIN", "Student Name", "Assignment 1", "Assignment 2" and so on, and each row will then store the grade of a particular student in those assignments. The entries are accessed by a **primary key** (may be the "UIN" in our example), and if needed, one can create two or more different tables that correspond to each other by means of **foreign key**s. Our example can be extended to two tables – one containing the personal details ("Address", "Phone", "Email") of students while a second one containing the grades, and the two tables may have the field "UIN" as a foreign key to link the corresponding entries. In essence, the purpose of the relational model is to provide a declarative method for specifying data and queries: users directly state what information the database contains and what information they want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for answering queries.

The Database Management Systems for such relational databases are known as **Relational Database Management Systems (RDMSs)**. Management of data in an RDMS is accomplished using a special-purpose programming language called the **Structural Query Language (SQL)**. SQL statements are used both for interactive queries for information from a relational database and for gathering data for reports. SELECT, INSERT, UPDATE and DELETE are some of the commonly used SQL scripts. **PostgreSQL**, generally abbreviated as "postgres", is an **open-source** SQL, meaning that the codes are accessible to everyone and can be updated for betterment. Now, to access the data already stored in a postgres database through a C++ program, one needs a special library called "**libpq**". It is actually a C library which comes with the postgres installation. The C++ program should have the library linked and following header file included:

```
#include <postgresql/libpq-fe.h>
```

There is an official client **Application Programming Interface (API)** library for C++, called "**libpqxx**", which may be installed separately and requires "libpq".

Now, let us assume that we have read from the table "Grades" contained in the database "Student_Grades_ECEN_489" the rows and columns pertaining to the grades of all the students in all the assignments; we are now set to perform the statistical analysis. A handful of statistical and mathematical computation tools are available under the **GNU Scientific Library (GSL)**, so it is to be installed from http://www.gnu.org/software/gsl/. For some mathematical computations, the corresponding header file should be included:

```
#include <gsl/gsl_math.h>
```

When a module contains type-dependent definitions, the library provides individual header files for each type. The filenames are modified as shown below. For convenience the default header includes the definitions for all the types. To include only the double precision header file, or any other specific type, use its individual filename.

```
#include <gsl/gsl_foo.h>                All types
#include <gsl/gsl_foo_double.h>         double
#include <gsl/gsl_foo_long_double.h>    long double
#include <gsl/gsl_foo_float.h>          float
#include <gsl/gsl_foo_long.h>           long
#include <gsl/gsl_foo_ulong.h>          unsigned long
#include <gsl/gsl_foo_int.h>            int
#include <gsl/gsl_foo_uint.h>           unsigned int
#include <gsl/gsl_foo_short.h>          short
#include <gsl/gsl_foo_ushort.h>         unsigned short
#include <gsl/gsl_foo_char.h>           char
#include <gsl/gsl_foo_uchar.h>          unsigned char
```

The compilation in UNIX, assuming that the library is present in the default location, should be as follows:

```
$ g++ -Wall -I/usr/local/include -c example.cpp
```

We now compute the mean, median, mode and standard deviation as follows.

(1) The Sample Mean:

```
double gsl_stats_mean (const double data[], unsigned stride,
unsigned n)
```

This function returns the arithmetic mean of *data*, an array of length *n* with step size *stride* (i.e. no. of memory locations between the beginnings of successive array elements).

(2) Standard Deviation:

```
double gsl_stats_sd (const double data[], unsigned stride,
size_t n)
```

This function uses the same arguments as the one for sample mean.

(3) Median:

We need to sort the data first. The header file `<gsl_heapsort.h>` is to be included.

```
void gsl_heapsort (void * array, size_t count,
size_t size, gsl_comparison_fn_t compare)
```

This function sorts the *count* elements of the array *array*, each of size *size*, into ascending order using the comparison function *compare*. The type of the comparison function is defined by,

```
int (*gsl_comparison_fn_t) (const void * a,
                            const void * b)
```

A comparison function should return a negative integer if the first argument is less than the second argument, `0` if the two arguments are equal and a positive integer if the first argument is greater than the second argument.

Once sorted, we can find the median by:

```
double gsl_stats_median_from_sorted_data (const
double sorted_data[], unsigned stride, unsigned n)
```

(4) Mode:

Here is a sample code for finding the mode of an array of double precision numbers:

```
// Finding the mode or modes in a data set

#include <iostream>
#include <stdlib.h>

using namespace std;
```

```
#define N 100
void mode (double x[],int n);//prototype for mode function

int main (void)
{

        double x[N];
        int n=0,i=0;

    cout<<"Enter in as many as 100 values, when finished enter in -99"<<endl;
while(1)
       {
           n++;

           cin>>x[i];
           if (x[i]==-99)
           {
               break;
           }
            i++;
       }

        mode(x,n);//send the values of x[] and n to mode function

system ("PAUSE");
return (0);
}

//function defintion

void mode (double x[],int n)
{
    int y[N]={0};//Sets all arrays equal to 0
    int i,j,k,m,cnt,max=0,no_mode=0,mode_cnt=0;
    double num;

    for(k=0; k<n; k++)//Loop to count an array from left to right
{
    cnt=0;
    num=x[k];//Num will equal the value of array x[k]

    for(i=k; i<n; i++)//Nested loop to search for a value equal to x[k]
    {
        if(num==x[i])
            cnt++;//if a number is found that is equal to x[k] count will go
up by one

    }
    y[k]=cnt;//The array y[k] is initialized the value of whatever count is
after the nested loop
    if(cnt>=2)//If cnt is greater or equal to two then there must be atleast
one mode, so no_mode goes up by one
    {
        no_mode++;
    }
}
if(no_mode==0)//after the for loops have excuted and still no_mode hasn't
been incremented, there mustn't be a mode
{
    //Print there in no mode and return control to main
    cout<<"This data set has no modes\n"<<endl;
    return;
}
    for(j=0; j<n; j++)
//A loop to find the highest number in the array
    {
              if(y[j]>max)
        max=y[j];
    }
 for(m=0; m<n; m++)//This loop finds how many modes there are in the data set
{
    //If the max is equal to y[m] then that is a mode and mode_cnt is
incremeted by one
    if(max==y[m])
        mode_cnt++;
}
cout<<"This data set has "<<mode_cnt<<" mode(s)"<<endl;//Prints out how many
modes there are
    for(m=0; m<n; m++)
    {
        if(max==y[m])//If max is equal to y[m] then the same sub set of array
x[] is the actual mode
        {

            cout<<"The value "<<x[m]<<" appeared "<<y[m]<<" times in the data
set\n"<<endl;
        }
    }
return;
}
```