ESS201 – C++ Programming
International Institute of Information Technology Bangalore

A5: Inheritance and Polymorphism
Announcement on Nov 04, 2020 (Wednesday), at 1:30 pm IST
Submission by Nov 12, 2020 (Thursday), at 11:59 pm IST, on Domjudge and LMS
(Nov 13, 2020 (Friday) is a holiday.)

Highlight:
Build from A4:

- Modify Histogram such that it can have #bin-values same as #bins for discrete values, as well as the option for having continuous intervals where #binvalues=#bins+1, see Class Redesign about creating DiscreteDistribution and ContinuousDistribution.
- Construct classes for discrete distributions, namely, binomial distribution, negative binomial distribution, and Poisson distribution.
    - Binomial distribution has finite support, while the other two have infiinite support.
    - The support of the distribution is coded as the #bins for the distribution.
- Construct a NegativeBinomialDistribution class with data members corresponding to the parameters n and r.
    - NegativeBinomialDistribution inherits from DiscreteDistribution with discrete values and default support N=20.
- Construct a PoissonDistribution class with data members corresponding to m (lambda value).
    - PoissonDistribution also inherits from DiscreteDistribution with discrete values, and with default support N=20.
- Input values for as many distributions as needed, and store them in a STL vector of DiscreteDistribution pointers.
- Use polymorphism to compute bin-values and descriptive statistics of each discrete distribution.
    - Mean, median, mode, and variance for the concerned distribution form its descriptive statistics.
- Output the descriptive statistics of each of the distributions, in the same order as in the input; and print out the distances between different distributions.
- Preserve as much code of BinomialDistribution class from A4 as possible, but now inherit it from DiscreteDistribution, and enhanced with computations for descriptive statistics.
- Preserve the distance functions, as implemented in A4, but move them to DiscreteDistribution class.
    - Ideally these functions could be in Distribution class. However, in the case of continuous distributions, it must check the bin-values for correct bin-wise comparison. Hence, for now, this is outside the scope of this assignment.

Class Redesign:

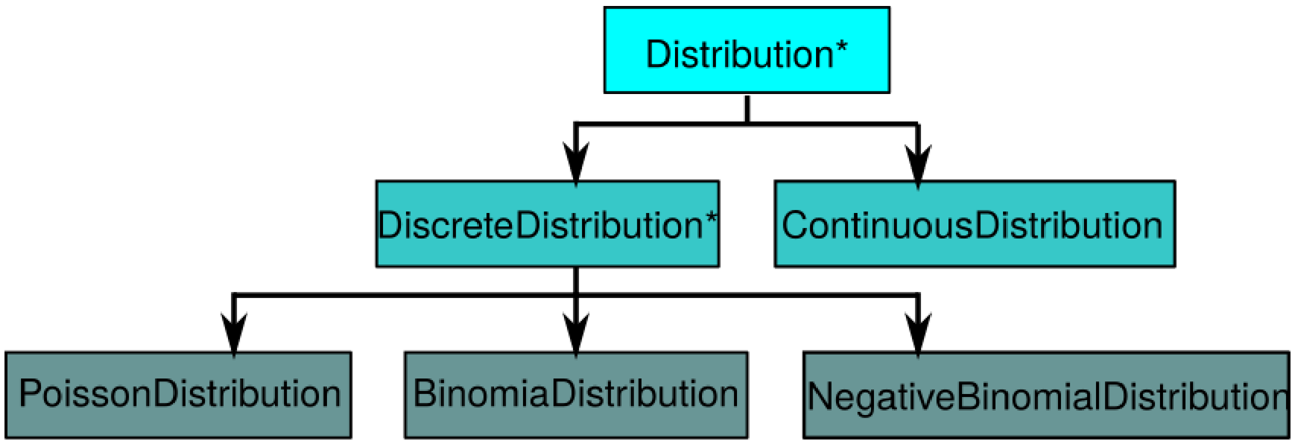

Figure 1: Class Hierarchy, where * implies abstract base class.

- At this juncture, you can redesign your base classes, and modify the Histogram class to be a DiscreteDistribution and ContinuousDistribution classes, based on the nature of the #bin-values, using the class hierarchy diagram in Figure 1.
    - Both DiscreteDistribution and ContinousDistribution derive from an abstract base class Distribution, where the DiscreteDistribution has the distance functions implemented in A4, as member functions.
    - The Histogram used in A3 is now ContinuousDistribution, and the Histogram used in A4 is now DiscreteDistribution.
    - The DiscreteDistribution is the (abstract) base class for the PoissonDistribution, NegativeBinomialDistribution, and BinomialDistribution.
    - DiscreteDistribution does not have a vector for bin-values, but instead has a std::vector of std::string for labels for bins.
    - ContinuousDistribution has bin-values and bin-frequencies.
- This is to comply with the Single Responsibility Principle.

Parameters for the distributions:

Binomial and negative binomial distributions require 2 parameters each; poisson distribution requires 1 parameter. In order to differentiate between the inputs for the binomial and negative binomial distributions, we use positive and negative integers for the parameter of #trials and #failures, respectively. The parameters for the distributions are (m) for Poisson distribution, *(Pois(m))*, (N,p) for binomial distribution *(B(N,p))*, and (-r,p) for negative binomial distribution, *(NB(r,p))*, where m is the rate, N is the number of trials, r is the number of successes. See “Notes” regarding computations pertaining to the distributions based on its parameters.

Input:

You can input as many distributions as you would like. The delimiter between distributions is -1, and parameters for each distribution are comma-separated, if the number of parameters is more than 1. The distance measure to be used is mentioned in the beginning of the sequence with a prefix D and an index, {1,2,3,4,5} for {Manhattan, Euclidean, Chebyshev, KL divergence, JS distance}, respectively. Example:

D3 -1 0.4 -1 10,0.3 -1 -11,0.4 -1 2.3 -1 20,0.8 -1 20,0.4 -1 -20,0.3 -1

Here, we have the following distributions Pois(0.4), B(10,0.3), NB(11,0.4), Pois(2.3), B(20,0.8), B(20,0.4), and NB(20,0.3), and we will be computing distances using Chebyshev distance measure.

Output:
For each distribution, the std::cout must print out its descriptive statistics, that is, mean, median, mode, and variance as comma-separated numbers. In the case of negative binomial distribution, the median can be skipped, as there is no formula for the same. The format would be:
<mean>,<median>,<mode>,<variance>
<mean>,<mode>,<variance>

The first line of output would be the descriptive statistics of the distributions in the order they have been inputted. As in the input, the output should contain -1 as delimiter between the distributions. For the 7 distributions in the aforementioned example input , the output will have the format:
<Stats. of Pois(0.4)> -1 <Stats. of B(10,0.3)> -1 .... <Stats. of NB(20,0.3)> -1

From the second line of output will be the comma-separated distance matrix between the distributions. In the afore-mentioned example, the 7x7 matrix will be outputted in the following format, using a newline character at the end of each row:
<d00>,<d01>,<d02>,<d03>,<d04>,<d05>,<d06>,
<d10>,<d11>,... <d16>
...
<d60>,<d61>,...<d66>

Distance Matrix Computation:

- The distance matrix is symmetric since all the 5 distance measures used here are either symmetric or symmetrized.
- The distance can be computed only if the #bins of the 2 distributions are the same.
- For BinomialDistribution, the #bins is a parameter, but for the others, they have infinite support. When comparing with a BinomialDistribution, use the #bins of the BinomialDistribution as the #bins needed for distance computation.
- When comparing 2 distributions with infinite support, use #bins as 100, which is the default #bins.

DiscreteDistribution Class Implementation:

- The distributions with infinite support must be initialized with default #bins (i.e., 100).
- At the time of distance computation, if the #bins in the distribution with infinite support does not satisfy the requirement of the #bins for distance computation, the vector for bin-values must be expanded.
    - The expansion of the vector entails computation of the newly added bin-values using the formulae for the PMF (probability mass function) of the concerned distribution.
    - Each of the classes must overload the std::cout function to print out its descriptive statistics.

Notes:
Use the PMF computations given in the Wikipedia pages to compute the bin-frequencies, as was done in A4. Use appropriate formula for mean, median, mode, and variance.
https://en.wikipedia.org/wiki/Poisson_distribution
https://en.wikipedia.org/wiki/Negative_binomial_distribution
https://en.wikipedia.org/wiki/Binomial_distribution