Turning the Tables on Probability Distributions

Stephen M. Mansour[1]
[1]Kania School of Management,
University of Scranton, Monroe Ave. and Linden St., Scranton, PA 18510

Abstract
While software has replaced the need for tables in statistics courses, many statistical programs do not go far enough. Some programs have several separate functions for each distribution, and some only calculate cumulative probabilities. Ideally, a software program should have exactly one function per distribution. Combining this distribution function with a relation allows us to calculate a whole host of probabilities consistently. The secret is in the use of operators which modify functions or combine two functions to produce another function. While operators may seem esoteric to many, the resulting syntax appears surprisingly English-like. For example, to find the probability that a randomly-selected student is taller than 6 feet, given a normal distribution with mean and standard deviation of 68 and 3 inches respectively, we simply state:

```
68 3 normal probability > 72
```

which produces a value of 0.09121. This syntax is easily implemented in a functional programming language such as APL.

Key Words: Probability, Distributions, Software, Operators, APL

1. Basic Concepts

Most of the operations in statistics can be reduced to the concepts of arrays, functions and operators described by Iverson in [2].

Arrays consist of scalars, vectors and matrices. A scalar is a single value such as 2. A vector is a list of values such as (5,8,3,2) and a matrix is a table of values.

Functions may be monadic (take one argument) or dyadic (take a left and right argument). Examples of monadic functions include $\ln x$ or $|x|$. Dyadic functions include the four common arithmetic functions $+, -, \times, \div$. The domains of functions which normally take scalar arguments have been extended to allow item-by-item calculations on vectors. Some functions may treat vectors and matrices as a whole; i.e. the function **`mean`** calculates the average of the values in its vector right argument.

An operator may take one or two functions as input to produce a different function. Mathematical examples of a monadic operator include the derivative $f'(x)$ and the inverse $f^{-1}(x)$. Some examples of dyadic operators are function composition $f \circ g$, and generalized inner product $\langle f, g \rangle$. Other examples of operators are listed in [3] and [5].

In the examples in the next section, each variable begins with a capital letter (e.g. **`Height`** or **`STATE`**) to distinguish it from a function or operator which begins with a lower-case letter (e.g. **`binomial`** or **`criticalValue`**) or symbol (e.g **`+`**).

2. Statistical Functions

There are four basic types of statistical functions: summary functions, distributions, relational functions and logical functions. In this paper, we will look primarily at distributions and relational functions.

A summary function is a monadic function which takes a vector as its argument and produces a scalar as its result. Examples include mean, median and standard deviation. A summary function applied to a population produces a parameter; when it is applied to a sample it produces a statistic. The mathematical form of a summary function is:

$$y = f(x_1, \cdots, x_n)$$

There are two types of distributions, discrete and continuous. Each can be characterized as a function: the probability mass function defines a discrete distribution, while the density function defines a continuous distribution. These functions are dyadic functions whose left argument is a vector of parameters and whose right argument is either a scalar or a vector. The result is the probability mass or density corresponding to each item in the right argument.

A mathematical relation between two values is either true or false. Iverson [5] defined relations as dyadic functions whose domain is numeric and whose range is Boolean. The Boolean value 1 represents truth, and 0 represents falsehood. Thus, the expression **`5>2`** results in the value 1, whereas **`3<3`** results in 0.

Logical functions bind propositions together; they include **`not`**, **`and`**, **`or`**, and **`if.`** Both the domain and range of logical functions are Boolean. Since the results of logical and relational functions are {0,1}, the results can be used directly in mathematical expressions since they do not need to be converted to numeric.

3. Statistical Operators

Unfortunately, the density function and to some extent the probability mass function produce results that are not always useful in some applications. Statistical operators were designed to modify these functions to extract important information from them such as critical values and cumulative probabilities. Operators are a useful mathematical construct developed by Iverson in 1979 for the computer language APL [1].

Let us look at the normal distribution. The function **`normal`** gives us the density for the standard normal distribution, e.g.

```
     normal 1.25
0.1826490854
```

Since this is not a probability, it is not meaningful to students. In most textbooks, the normal distribution is usually displayed as a cumulative probability table. So, we use the probability operator using the left operand **`normal`** and the right operand "**`<`**" to produce useful output:

```
    normal probability < 1.25
```

```
0.8943501801
```

The operator **probability** combines the distribution function **normal** with the relational function "**<**" to produce the cumulative normal distribution function :

```
(normal probability <)
```

which is then applied to the numeric argument 1.25. To find the upper-tail probability, one could subtract the result from one:

```
      1-normal probability < 1.25
0.1056498199
```

But it's much easier to change the right operand to "**>**" and get the answer directly:

```
    normal probability > 1.25
0.1056498199
```

Suppose we wish to find the following normal probability: $P(1.25 < Z < 1.75)$. This usually requires three steps: finding $P(Z < 1.75)$, finding $P(Z < 1.25)$ and subtracting the two values. However, we can simply change the right operand of **probability** to **between** and get the appropriate result:

```
     normal probability between 1.25 1.75
0.06559071228
```

One of the more difficult tasks for students is to find a critical value which cuts off a certain percentage of values. Suppose we want to find the critical value which cuts off the top 5% of values. This becomes much easier when we define the **criticalValue** operator:

```
    normal criticalValue < 0.05
1.6448534
```

The rationale for the above expression is we want the critical value of the normal distribution which is less than 5% of all values. We could obtain the same result by asking for the critical value which is greater than 95% of all values:

```
    normal criticalValue > 0.95
1.6448534
```

Now let's look at a real-world problem: A brewery produces 12-ounce bottles of beer. Quality assurance draws a sample and finds that the mean amount of beer in each bottle is 12.13 oz. with a standard deviation of 0.06 oz. What is the probability that a randomly selected bottle of beer will contain less than 12 ounces?

Instead of converting 12 oz to standard normal units, we simply assign a left argument to the normal function consisting of the mean and standard deviation:

```
    12.13 0.06 normal probability < 12
0.01513008392
```

The probability of a bottle containing less than 12 ounces is 1.513%

Let's try an example with a discrete distribution. A commuter plane seats 20 passengers. 10% of passengers with reservations fail to show up. The airline overbooks by issuing 21 reservations. What is the probability that they will have to throw a passenger off the plane? This would be the binomial probability that exactly 21 passengers show up. This is a binomial distribution with $n = 21$ and $p = 0.9$, so we use the binomial function with a left argument consisting of the two-item parameter vector $(21,0.9)$:

```
     21 0.9 binomial probability = 21
0.1094189891
```

This will happen about 10% of the time. Suppose the airline decides to issue 22 reservations. In this case we want to know the probability that more than 20 passengers show up:

```
    22 0.9 binomial probability > 20
0.3391988663
```

The airline will have to bump at least one passenger about a third of the time. We can simulate passenger arrivals by using generating binomial random variables, by using the **`randomVariable`** operator:

```
     21 0.9 binomial randomVariable 20
18 19 20 17 20 19 19 20 21 18 20 20 20 20 20 19 21 18 20 17
```

We see that two times out of 20 we had 21 passengers show up which agrees with the 10% probability figure.

```
      22 0.9 binomial randomVariable 20
17 22 20 19 22 19 19 21 18 16 21 19 19 21 21 21 16 19 18 20
```

In this example, we see that 7 out of 20 times we had to turn away at least one ticketed passenger which is close to the 1/3 estimate.

To find the expected value we use the **`theoretical`** operator with the summary function **`mean`**:

```
      22 0.9 binomial theoretical mean ''
19.8
```

The variance of the distribution can be found similarly:

```
     22 0.9 binomial theoretical var ''
1.98
```

In statistical inference, we often use the Student-t, chi-Square and F distributions. The tables for these distributions display critical values. We can use the critical value operator to obtain these values. The degrees of freedom are the left argument(s) to these functions:

```
      7 tDist criticalValue outside .95
2.364624256
      7 chiSquare criticalValue < .05
14.06714045
      3 8 fDist criticalValue < .05
4.066180551
```

Other than the normal distribution, most statistical tables do not give us p-Values. However, we can use the **probability** operator to obtain them.

```
      7 tDist probability outside 2.36
0.05034128863
      7 chiSquare probability > 14
0.05118135341
      3 8 fDist probability > 4
0.05189367685
```

Finally, we can perform goodness of fit tests on various distributions. First we'll generate some data. The diagraph **<-** is used to assign the name **X** to the data.

```
      X <- 25 0.5 binomial randomVariable  100
```

Then we'll use the **goodnessOfFit** operator to test whether the data seem to be uniformly distributed. The uniform distribution assumes that each outcome is equally likely. For example, in tossing a 6-sided die, the probabilities of the number between 1 and 6 are equal:

```
      6 uniform probability = 1 2 3 4 5 6
0.16667 0.16667 0.16667 0.16667 0.16667 0.16667
```

The **goodness0fFit** operator is a monadic operator taking a distribution function as its operand.

```
      U <- uniform goodnessOfFit X
```

The result is a namespace containing various outputs. We can obtain the value of the (Chi-Square) test statistic, and compare it to the critical value of the Chi-Square distribution with 10 degrees of freedom.

```
      U.TestStatistic
48.94
      U.DegreesOfFreedom
10
      10 chiSquare criticalValue < .05
21.02606982
```

Or we can simply obtain the p-value directly from the output and conclude that the distribution is not uniform.

```
      U.P
```

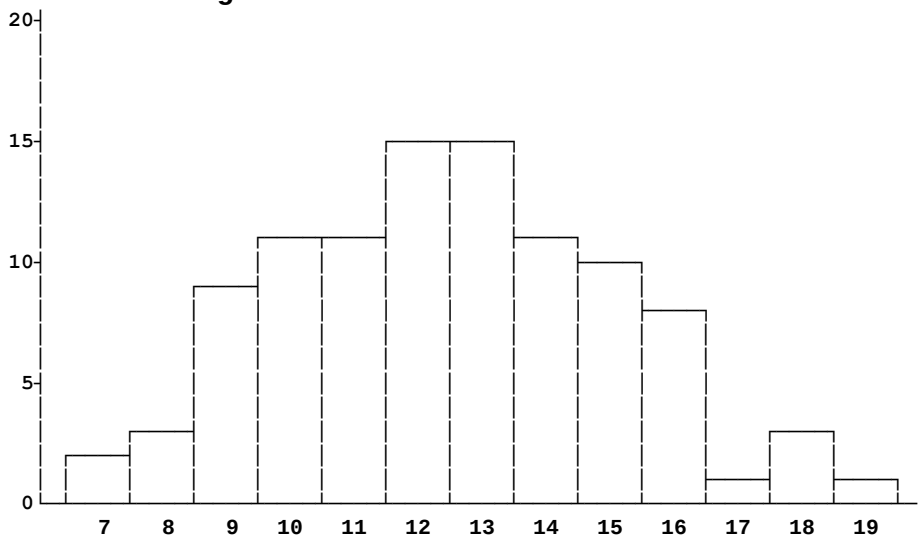
```
4.17767236E¯7
```

We reject the hypothesis that the data are uniform because the p-value is extremely small. Now let us test the data to see if they are normally distributed: In this case we simply replace the function **uniform** with **normal**:

```
      N <- normal goodnessOfFit X
      N.DegreesOfFreedom
7
      N.TestStatistic
9.34402855
      7 chiSquare criticalValue < .05
14.06714045
      N.P
0.2288906756
```

The p-value is much larger and the test statistic is smaller than the critical value so we fail to reject the hypothesis that the data are normally distributed. And if we display a histogram of the data we can see that indeed they are bell-shaped:

```
      histogram X
```



4. Conclusion

Chen[6] states that the suitability of a programming language for a particular task is less about the functionality of the language and more about the expressability of that language. The use of operators allows us to express statistical concepts in a consistent way without a proliferation of functions. Languages such as R do wonderful calculations, but the syntax is rather cryptic. A student would find it much easier to enter:

```
100 20 normal probability > 90
```

rather than the R syntax:

```
pnorm (90,100,20, lower.tail=FALSE)
```

We have developed a working software program in APL using the syntax in this paper and which has the option of calling R to perform the statistical calculations without the user having to know R syntax.

Acknowledgements

The syntax used in development of this system is based on the programing language APL developed by Harvard mathematical Ken Iverson. APL was originally implemented by IBM; some extensions to the language have been implemented by Dyalog APL, Ltd.

References

[1] Iverson, Kenneth E "The Role of Operators in APL" ACM SIGAPL APLQuote Quad Part I Volume 9 Issue 4, June 1979 Page 128-133, ACM New York, NY USA.
[2] Iverson, K.E. "A Programming Language 1962.
[3] Smith, Bob "Nested Arrays, Operators and Functions" ACM SIGAPL APL Quote Quad Volume 12 Issue 1, September 1981 Page 286-290.
[4] Brown, James A. "Function assignment and arrays of functions" ACM SIGAPL Quote Quad, Volume 14 Issue 4, June 1984 Pages 81-84.
[5] Falkoff, A.D. and Iverson, K.E. "The Design of APL" IBM Journal of Research and Development (Volume: 17, Issue: July 1973)
[6] Chen, Jiahao "Linguistic Relativity and Programming Languages" JSM-2016 – Section on Statistical Computing