Documentation for Distributions.py

Sophocrates-XL

Introduction

This Python module aims to provide a light-weight platform to compute simple statistical operations, for common analytical problems encountered both in academic study and in work.

Installation

Simply copy the source code "Distributions.py" under the "lib" folder of the Python environment to complete the installation.

Dependencies

The only dependency for the module is the pyerf, which is needed to compute the inverse error function in some cases.

General Features

The module focuses on operations derived from probability distributions. Basic operations include computation of probability density or mass from value, computation of cumulative probability from value, and computation of quantile from cumulative probability. Sampling-related operations include generation of random samples, estimation of parameters of probability distributions, and hypothesis tests. Miscellaneous operations include computation of convolution, as well as integration and differentiation. The details of implementation are covered in subsequent sections.

Probability Distributions and their basic operations

The current module offers functionalities for most commonly encountered probability distributions. Listed below are the implemented probability distributions and their constructors.

- *Uniform distribution: $Uniform(lower, upper)$.
- *Bernoulli distribution: $Ber(prob1)$.
- Beta distribution: $Beta(shape1, shape2)$.
- **Binomial distribution: $Bin(n, prob1)$.
- Cauchy distribution: $Cauchy(x0, scale)$.

- Chi-square distribution: $Chi2(df)$.
- **Erlang distribution: $Erlang(k, rate)$.
- *Exponential distribution: $Exp(rate)$.
- Gamma distribution: $Gamma(shape, rate)$.
- *Geometric distribution: $Geo(prob1)$.
- Levy distribution: $Levy(loc, scale)$.
- *Lognormal distribution: $LogNorm(norm\_mean, norm\_var)$.
- *Normal distribution: $N(mean, var)$.
- **Negative binomial distribution: $NegBin(k, prob1)$.
- *Poisson distribution: $Pois(rate)$.
- F-distribution: $SnedecorF(df1, df2)$.
- Student's t-distribution: $t(df)$.

Programmatically, all probability distributions derive from the base class "Distribution", which implements methods common for all distribution families. A distribution family is a class, whereas a distribution with specified values of parameters is an instantiation of the class. This distinction is essential for the module's functionalities, because while most computations access methods of the instance, some computations such as maximum likelihood estimation access static methods of the class.

Creation of random variable:

A random variable is declared by instantiation of the distribution family. For example, the statement $X = N(1, 2)$ declares a random variable X that follows a normal distribution with mean 1 and variance 2.

The name Z has already been reserved to denote a standard normal variable. This name should not be overwritten, because many functionalities in the module depend on that.

Access of attributes of random variables:

Whenever a random variable is instantiated, the relevant attributes can be checked. Listed below are the accessible attributes.

- $rv.param\_name$: Checks the value of the parameter. Names of parameter are different for different distributions.
- $rv.inf$: Checks the lower bound of the distribution.
- $rv.sup$: Checks the upper bound of the distribution.
- $rv.mean$: Checks the mean of the random variable. A value of $None$ indicates that mean is undefined.
- $rv.var$: Checks the variance of the random variable. A value of $None$ indicates that variance is undefined.
- $rv.sd$: Checks the standard deviation of the random variable. Naturally, if $rv.var$ returns $None$, $rv.sd$ also returns $None$.

Basic operations:

Listed below are calculations implemented by all distributions.

- $rv.p(x)$: Computes the probability mass at x for a discrete random variable.
- $rv.f(x)$: Computes the probability density at x for a continuous random variable.
- $rv.F(x)$: Computes the cumulative probability at x, applicable to both discrete and continuous random variables.
- $rv.Fc(x)$: Computes the complement of the cumulative probability at x, applicable to both discrete and continuous random variables. Mathematically, we have $rv.Fc(x) = 1 - rv.F(x)$.
- $rv.Q(prob)$: Computes the quantile x provided the cumulative probability $prob$, applicable to both discrete and continuous random variables. Mathematically, quantile function is the inverse of cumulative probability function. Note that due to rounding reasons, quantile function may return inaccurate results if $prob$ is very close to one or zero.

To mimic the mathematical syntax, all aforementioned methods can be called using the format $func[rv](x)$. For example, the commands $X = Bin(5, 0.2)$ and $p[X](2)$ computes the probability mass of a binomial distribution with $n = 5$ and $p = 0.2$ at $x = 2$.

Sampling-related operations

Samples may be generated provided the distribution, both in the form of single observations or as a sample of multiple independent identically distributed observations. The implementations are shown below.

- $rv.\_\_call\_\_()$: Generates a single observation. This is an operator overload for the "call" operator. To execute the method, we simply input the command $rv()$.
- $rv.sample(sample\_size)$: Generates multiple independent observations based on the sample size.

When a sample of observations are obtained, the module provides basic functions to compute sample mean, variance and standard deviation. All such functions are encapsulated in the "Sample" namespace.

- $Sample.mean(elements)$: Computes sample mean.
- $Sample.var(elements)$: Computes sample variance.
- $Sample.sd(elements)$: Computes sample standard deviation.

Maximum likelihood estimation from sample

In the module, estimation of distribution parameters currently includes only the maximum likelihood estimator, MLE, because the sampling distributions of MLEs are asymptotically normal whose variance is easily computed from Fisher information.

The method to call for an MLE is $distribution.getMLE(observations)$. Note that this is a static method for the class denoting the distribution family, instead of the instances of distributions. For example, for a normally distributed sample of observations, we obtain the MLE with the command, $N.getMLE(observations)$. We must use the class name because parameters are unknown. Some distributions, such as binomial distribution, requires one parameter to be provided for estimation of the other parameter. For example, $Bin.getMLE(n, observations)$ computes the MLE for success probability, provided the trial count n.

Currently, MLE is not implemented for all distributions. In the aforementioned list of distributions, an asterisk means that MLE is implemented with the distribution family, while a double asterisk means that the MLE is computed provided one parameter.

Upon execution of the $getMLE$ method, an instance of the $MLEStat$ class is returned. When more than one parameter is estimated, a list of $MLEStat$ instances is returned, each labelled with the name of the estimated parameter. Below are the attributes and methods accessible from $MLEStat$ instances.

- $mlestat.name$: Checks the name of the estimated parameter.
- $mlestat.est$: Checks the value of the estimate.
- $mlestat.n$: Checks the size of sample used for the estimate.
- $mlestat.fisher$: Checks the estimated Fisher information associated with the parameter of interest.
- $mlestat.asymptotic\_var$: Checks the asymptotic variance of the MLE. Mathematically, we have $mlestat.asymptotic\_var = 1/(mlestat.n * mlestat.fisher)$.
- $mlestat.getCI(significance\_level)$: Obtains the confidence interval of the MLE based on the significance level.

Hypothesis tests

The module includes functionalities for common hypothesis tests, such as z-test, Student's t-test, chi-square test and F-test. Since tests may be one-sampled, two-sampled, one-tailed or two-tailed, a standardized nomenclature is adopted to avoid confusion. All tests are named in the format $name[m]s[n]t$, where m indicates the number of samples involved, and n indicates the number of tails involved. For example, $zTest1s2t$ computes the one-sample, two-tailed z-test. Below is the list of hypothesis tests implemented.

- $zTest1s1t$: One-tailed z-test for one sample.
- $zTest1s2t$: Two-tailed z-test for one sample.
- $zTest2s1t$: One-tailed z-test for two samples. Determines if the mean from one normally distributed sample is greater or lesser than the other.
- $zTest2s2t$: Two-tailed z-test for two samples. Determines if the mean of independent samples from two normally distributed populations are equal.
- $tTest1s1t$: One-tailed t-test for one sample.
- $tTest1s2t$: Two-tailed t-test for one sample.

- $tTest2s1t$: One-tailed t-test for two samples. Determines if the mean from one normally distributed sample is greater or lesser than the other.
- $tTest2s2t$: Two-tailed t-test for two samples. Determines if the mean of independent samples from two normally distributed populations are different.
- $chi2TestVar1t$: One-tailed chi-square test of variance. Determines if the sampled variance is greater or lesser than the hypothesized variance.
- $chi2TestVar2t$: Two-tailed chi-square test of variance. Determines if the sampled variance differs significantly from the hypothesized variance.
- $fTestVar1t$: One-tailed F-test of variance. Determines if variances from one sample is greater or lesser than the other.
- $fTestVar2t$: Two-tailed F-test of variance. Determines if variances from samples drawn from two populations are different.

All hypothesis tests return an instance of $TestStat$ class, with accessible attributed listed below.

- $teststat.type$: Checks the type of hypothesis test performed.
- $teststat.hypothesized\_value$: Checks the assumed value under the null hypothesis.
- $teststat.observed\_value$: Checks the value of the parameter of interest computed from the observations.
- $teststat.test\_stat$: Checks the test statistic associated with the hypothesis test.
- $teststat.p\_value$: Checks the p-value of the test statistic. In practice, p-value is more informative on the extent of deviation of observed values from the null hypothesis.

Miscellaneous functionalities

Convolution of independent random variables:

Convolution allows computation of probabilities related to sums of random variables. Currently, only convolutions for independent and non-negative discrete random variables are implemented.

The syntax to construct a convolution is $Convolution[rvs]$, where rvs is an iterable object with all the random variables for which we wish to compute the sum. Similar to a standard probability distribution, $convolution.p(x)$ and $convolution.F(x)$ are used to calculate probability mass and cumulative probability respectively.

As an example, $Convolution[(X, Y, Z)].F(x)$ calculates the probability that the sum $X + Y + Z$ is less than or equal to x.

Differentiation and integration:

Initially, differentiation and integration functionalities are implemented for computations related to probability distributions. Nonetheless, they can be used on their own as well.

- $differentiate(function, x)$: Computes the slope of a function at point x. Usually, lambda functions are used.

- $integrate(function, lower, upper)$: Computes the definite integral of a function bounded by the $lower$ and $upper$ arguments. Usually, lambda functions are used.