The poweRlaw package: a general overview

Colin S. Gillespie

Last updated: May 27, 2014

The `poweRlaw` package provides code to fit heavy tailed distributions, including discrete and continuous power-law distributions. Each model is fitted using a maximum likelihood procedure and cut-off value, $x_{\min}$, is estimated by minimising the Kolmogorov-Smirnoff statistic.

1 *Installation*

The package is hosted on CRAN and can be installed in the standard way

```
install.packages("poweRlaw")
```

The developmental version is hosted on github and can be installed using the `devtools` package[1]

[1] If you are using Windows, then you will need to install the `Rtools` package first.

```
install.packages("devtools")
library("devtools")
install_github("poweRlaw", "csgillespie", subdir="pkg")
```

Once installed, the package can be loaded ready for use with the standard `library` command

```
library("poweRlaw")
```

2 *Accessing documentation*

Each function and dataset in the package is documented. The command

```
help(package="poweRlaw")
```

will give a brief overview of the package and a complete list of all functions. The list of vignettes associated with the package can be obtained with

```
vignette(package="poweRlaw")
```

or

```
browseVignettes("poweRlaw")
```

Help on functions can be obtained using the usual R mechanisms. For example, help on the method `displ` can be obtained with

```
?displ
```

and the associated example can be run with

```
example(displ)
```

A list of demos and data sets associated with the package can be obtained with

```
demo(package="poweRlaw")
data(package="poweRlaw")
```

For example, the Moby dick data set can be load using

The package also contains the data set `moby_sample`. This data set is two thousand randomly sampled values from the larger `moby` data set.

```
data("moby")
```

After running this command, the vector `moby` will be accessible, and can be examined by typing

```
moby
```

at the R command prompt.

If you use this package, please cite it. The appropriate citation can be obtained via:

```
citation("poweRlaw")
```

3 *Example: Word frequency in Moby Dick*

This example investigates the frequency of occurrence of unique words in the novel Moby Dick by Herman Melville.[2] The data can be downloaded from

http://tuvalu.santafe.edu/~aaronc/powerlaws/data.htm

or loaded directly

[2] A.~Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009; and M.E.J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005

```
data("moby")
```

3.1 *Fitting a discrete power-law*

To fit a discrete power-law,[3] we create a discrete power-law object using the `displ` method[4]

[3] The examples vignette contains a more thorough analysis of this particular data set.

[4] `displ`: discrete power-law.

```
m_m = displ$new(moby)
```

Initially the lower cut-off $x_{\min}$ is set to the smallest x value and the scaling parameter α is set to `NULL`

```
m_m$getXmin()
## [1] 1

m_m$getPars()
## NULL
```

This object also has standard setters

```
m_m$setXmin(5)
m_m$setPars(2)
```

For a given $x_{\min}$ value, we can estimate the corresponding α value by numerically maximising the likelihood

Instead of calculating the MLE, we could use a parameter scan: `estimate_pars(m_m, pars=seq(2, 3, 0.1))`

```
(est = estimate_pars(m_m))
## $pars
## [1] 1.926
##
## $value
## [1] 14873
##
## $counts
## function gradient
##        5        5
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## attr(,"class")
## [1] "estimate_pars"
```

For the Moby Dick data set, when $x_{\min} = 5$, we estimate α to be 1.926.

To ESTIMATE the lower bound $x_{\min}$, we minimise the distance between the data and the fitted model CDF, that is

$$D(x) = \max_{x \geq x_{\min}} |S(x) - P(x)|$$

where $S(x)$ is the data CDF and $P(x)$ is the theoretical CDF. The value $D(x)$ is known as the Kolmogorov-Smirnov statistic. Our estimate of $x_{\min}$ is then the value of x that minimises $D(x)$:

```
(est = estimate_xmin(m_m))
## $KS
## [1] 0.008253
##
## $xmin
```

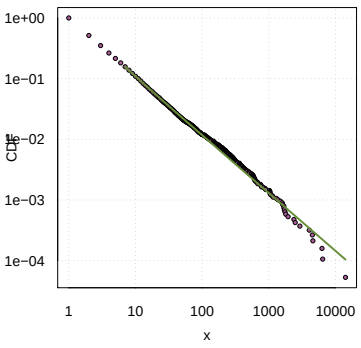

Figure 1: Plot of the data CDF for the Moby Dick data set. This corresponds to figure 6.1(a) in Clauset, 2009. The line corresponds to a power-law distribution with parameters $x_{\min} = 7$ and $\alpha = 1.95$.

Algorithm 1: Uncertainty in $x_{\min}$

1: Set N equal to the number of values in the original data set
2: **for** `i` in `1:B`:
3: Sample N values from the original data set
4: Estimate $x_{\min}$ and α using the Kolmogorov-Smirnoff statistic
5: **end for**

```
## [1] 7
##
## $pars
## [1] 1.953
##
## attr(,"class")
## [1] "estimate_xmin"
```

For the Moby-Dick data set, the minimum[5] is achieved when $x_{\min} = 7$ and $D(7) = 0.0082$.

We can then set parameters of power-law distribution to these "optimal" values

[5] These estimates match the values in the Clauset, *et al* paper.

```
m_m$setXmin(est)
```

All distribution objects have generic plot methods[6]

[6] Generic `lines` and `points` functions are also available.

```
## Plot the data (from xmin)
plot(m_m)
## Add in the fitted distribution
lines(m_m, col=2)
```

which gives figure 1. When calling the `plot` and `lines` functions, the data plotted is actually invisibly returned, i.e.

```
dd = plot(m_m)
head(dd, 3)
##   x      y
## 1 1 1.0000
## 2 2 0.5141
## 3 3 0.3505
```

This makes it straight forward to create graphics using other R packages, such as `ggplot2`.

3.2 *Uncertainty in* $x_{\min}$

Clauset, *el al*, 2009 recommend a bootstrap procedure to get a handle on parameter uncertainty. Essentially, we sample with replacement from the data set and then re-infer the parameters (algorithm 1).

To run the bootstrapping procedure, we use the `bootstrap` function

```
bs = bootstrap(m_m, no_of_sims=1000, threads=1)
```

this function runs in parallel, with the number of threads used determined by the `threads` argument. To detect the number of cores on your machine, you can run:

```
parallel::detectCores()
## [1] 8
```

The object returned by `bootstrap` is a list with three elements.

- The original Kolmogorov-Smirnov statistic.
- The results of the bootstrapping procedure.
- The average time (in seconds) for a single bootstrap.

The results of the bootstrap procedure can be investigated with histograms

```
hist(bs$bootstraps[,2], breaks="fd")
hist(bs$bootstraps[,3], breaks="fd")
```

and a bivariate scatter plot

```
plot(jitter(bs$bootstraps[,2], factor=1.2), bs$bootstraps[,3])
```

These commands give figures 2 and 3.

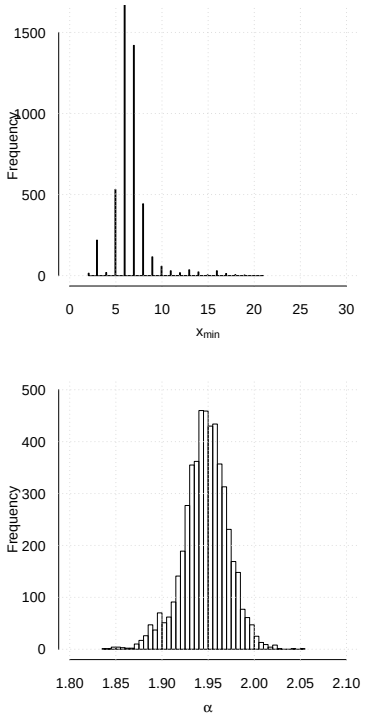

Figure 2: Characterising uncertainty in parameter values. (a) $x_{\min}$ uncertainty (standard deviation 2) (b) α uncertainty (std dev. 0.03)

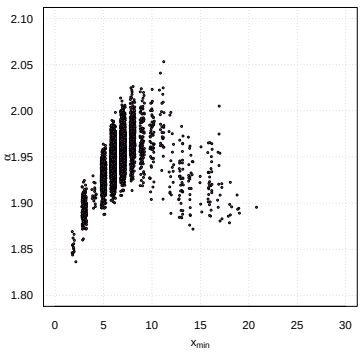

Figure 3: Characterising uncertainty in parameter values. Bivariate scatter plot of $x_{\min}$ and α.

Algorithm 2: Do we have a power-law?

1: Calculate point estimates of $x_{\min}$ and the scaling parameter α.
2: Calculate the KS statistic, KS_d, for the original data set.
3: Set n_1 equal to the number of values below `xmin`.
4: Set $n_2 = n - n_1$ and $P = 0$.
5: **for** `i` in `1:B`:
6: Simulate n_1 values from a discrete uniform distribution: $U(1, x_{\min})$ and n_2 values from a discrete power-law distribution (with parameter α).
7: Calculate the associated KS statistic, KS_{sim}.
8: If $KS_d > KS_{sim}$, then $P = P + 1$.
9: **end for**
10: p = P/B

3.3 *Do we have a power-law?*

Since it is possible to fit a power-law distribution to *any* data set, it is appropriate to test whether it the observed data set actually follows a power-law. Clauset *et al*, suggest that this hypothesis is tested using a goodness-of-fit test, via a bootstrapping procedure. Essentially, we perform a hypothesis test by generating multiple data sets (with parameters $x_{\min}$ and α) and then "re-inferring" the model parameters. The algorithm is detailed in Algorithm 2.

When α is close to one, this algorithm can be particularly time consuming to run, for two reasons:

1. When generating random numbers from the discrete power-law distribution, large values are probable, i.e. values greater than 10^8. To overcome this bottleneck, when generating the random numbers all numbers larger than 10^5 are generated using a continuous approximation.

2. To calculate the Kolmogorov-Smirnov statistic, we need explore the state space. It is computationally infeasible to explore the entire state space when $\max(x) >> 10^5$. To make this algorithm computational feasible, we split the state space into two sections. The first section is all values from

$$x_{\min}, x_{\min} + 1, x_{\min} + 2, \ldots, 10^5$$

this set is combined with an additional 10^5 values from

$$10^5, \ldots, \max(x)$$

To determine whether the underlying distribution is a power-law we use the `bootstrap_p` function

```
## This may take a while
## Use the mle to estimate the parameters
bs_p = bootstrap_p(m_m, no_of_sims=1000, threads=2)
```

The object returned from the bootstrap procedure contains four elements

Algorithm 2 can be easily extended for other distributions.

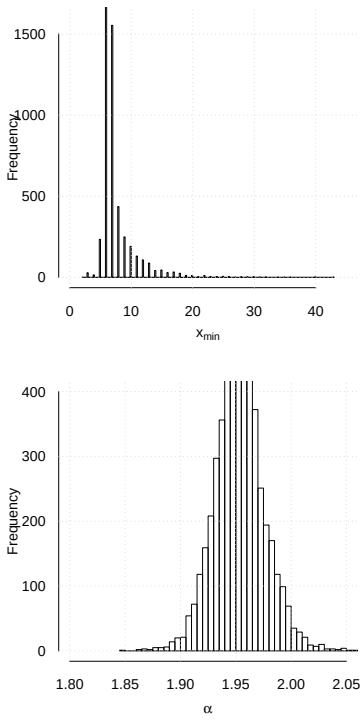

Figure 4: Histograms of the bootstrap results.

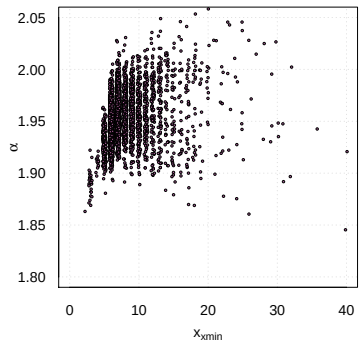

Figure 5: Bivariate scatter plot of the bootstrap results. The values of $x_{\min}$ and α are obviously strongly correlated.

- A p-value - `bs_p$p`. For this example, $p = 0.6778$ which indicates that we can not rule out the power law model. See section 4.2 of the Clauset paper for further details.
- The original goodness of fit statistic - `bs_p$gof`.
- The result of the bootstrap procedure - a data frame with three columns.
- The average time (in seconds) for a single bootstrap realisation.

The results of this procedure are shown in figures 4 and 5.

4 *Distribution objects*

For the Moby Dick example, we created a `displ` object

```
m_m = displ$new(moby)
```

The object `m_m` has class `displ` and inherits the `discrete_distribution` class. A list of available distributions are given in table 1.

Distribution	Object name	# Parameters
Discrete Power-law	`displ`	1
Discrete Log-normal	`dislnorm`	2
Discrete Exponential	`disexp`	1
Poisson	`dispois`	1
CTN Power-law	`conpl`	1
CTN Log-normal	`conlnorm`	2

Table 1: Available distributions in the `poweRlaw` package. These objects are all reference classes.

All distribution objects listed in table 1 are reference classes. Each distribution object has four fields:

See `?setRefClass` for further details on references classes.

- `dat`: a copy of the data set.
- `xmin`: the lower cut-off $x_{\min}$.
- `pars`: a vector of parameter values.
- `internal`: a list of values use in different numerical procedures. This will differ between distribution objects.

By using the mutable states of reference objects, we are able to create efficient caching. For example, the mle of discrete power-laws uses the statistic:

$$\sum_{i=x_{\min}}^{n} \log(x_i)$$

This value is calculated once for all values of $x_{\min}$, then iterated over when estimating $x_{\min}$.

All distribution objects have a number of methods available. A list of methods is given in table 2. See the associated help files for further details.

Method Name	Description
`dist_cdf`	Cumulative density/mass function (CDF)
`dist_pdf`	Probability density/mass function (pdf)
`dist_rand`	Random number generator
`dist_data_cdf`	Data CDF
`dist_ll`	Log-likelihood
`estimate_xmin`	Point estimates of the cut-off point and parameter values
`estimate_pars`	Point estimates of the parameters (conditional on the current $x_{\min}$ value)
`bootstrap`	Bootstrap procedure (uncertainty in $x_{\min}$)
`bootstrap_p`	Bootstrap procedure to test whether we have a power-law

Table 2: A list of functions for `distribution` functions. These objects do not change the object states. However, they may not be thread safe.

5 *Loading data*

Typically, data is stored in a csv or text file. To use this data, we load it in the usual way[7]

```
blackouts = read.table("blackouts.txt")
```

[7] The blackouts data set can be obtained from Clauset's website: http://goo.gl/BsqnP

Distribution objects take vectors as inputs, so

```
m_bl = conpl$new(blackouts$V1)
```

will create a continuous power law object.

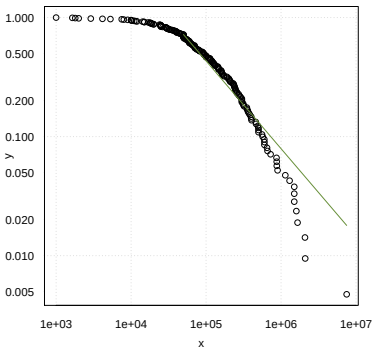

Figure 6: CDF plot of the blackout dataset with line of best fit. Since the minimum value of x is large, we fit a continuous power-law as this is more efficient.

6 *Comparison with the `plfit` script*

6.1 *The discrete case*

Other implementations of estimating the lower bound can be found at

http://tuvalu.santafe.edu/~aaronc/powerlaws/

In particular, the script for estimating $x_{\min}$ can be loaded using

```
source("http://tuvalu.santafe.edu/~aaronc/powerlaws/plfit.r")
```

The results are directly comparable to the `poweRlaw` package. For example, consider the Moby Dick data set again, we have

```
plfit(moby)
## $xmin
## [1] 7
##
## $alpha
## [1] 1.95
##
## $D
## [1] 0.009289
```

Notice that the results are slightly different. This is because the `plfit` by default does a parameter scan over the range

$$1.50, 1.51, 1.52, \ldots, 2.49, 2.50$$

To exactly replicate the results, we could use

```
estimate_xmin(m_m, pars=seq(1.5, 2.5, 0.01))
```

6.2 *The continuous case*

The `plfit` script also fits continuous power-laws. Again the results are comparable.

For example, suppose we have one thousand random numbers from a continuous power-law distributions with parameters $\alpha = 2.5$ and $x_{\min} = 10.0$

```
r = rplcon(1000, 10, 2.5)
```

The `plfit` automatically detects if the data is continuous

```
plfit(r)
## $xmin
## [1] 10.01
##
## $alpha
## [1] 2.437
##
## $D
## [1] 0.01754
```

Fitting with the `poweRlaw` package gives approximately the same values

```
m_r = conpl$new(r)
(est = estimate_xmin(m_r))
## $KS
## [1] 0.01791
##
## $xmin
## [1] 10.01
##
## $pars
## [1] 2.435
##
## attr(,"class")
## [1] "estimate_xmin"

m_r$setXmin(est)
```

Of course, using the `poweRlaw` package we can easily plot the data

```
plot(m_r)
lines(m_r, col=2)
```

to get figure 7.

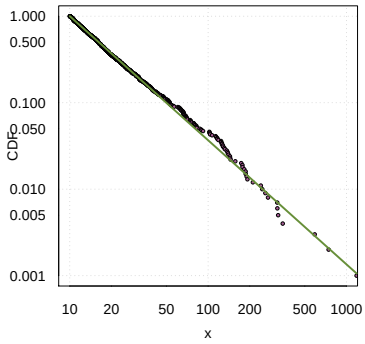

Figure 7: CDF plot of one thousand random numbers generated from a power-law with parameters $\alpha = 2.5$ and $x_{\min} = 10$. The line of best fit is also shown.

Session Info

Package	Version
parallel	3.1.0
poweRlaw	0.20.3
VGAM	0.9-3

Table 3: A list of packages and versions used.

```
print(sessionInfo(), locale = FALSE)
## R version 3.1.0 (2014-04-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
##
## attached base packages:
## [1] stats4    splines   stats     graphics
## [5] grDevices utils     datasets  methods
## [9] base
##
## other attached packages:
## [1] R.matlab_2.2.3  VGAM_0.9-3
## [3] poweRlaw_0.20.3 knitr_1.6
##
## loaded via a namespace (and not attached):
##  [1] R.methodsS3_1.6.1 R.oo_1.17.0
##  [3] R.utils_1.29.8    codetools_0.2-8
##  [5] digest_0.6.4      evaluate_0.5.5
##  [7] formatR_0.10      highr_0.3
##  [9] parallel_3.1.0    stringr_0.6.2
## [11] tools_3.1.0
```

This vignette was created using the excellent `knitr` package.[8]

[8] Y.~Xie. *knitr: A general-purpose package for dynamic report generation in R*, 2013. URL http://CRAN.R-project.org/package=knitr. R package version 1.5

References

A.~Clauset, C.R. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

M.E.J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

Y.~Xie. *knitr: A general-purpose package for dynamic report generation in R*, 2013. URL http://CRAN.R-project.org/package=knitr. R package version 1.5.