Praxis: Fitting power-laws to data

In this assigment we will be combining what we learned this week to fit a power-law to some data and then visualize both the power-law and the fit.

(1) Getting the data

First we need to generate some data to fit to. We will use the degree distribution from the Barabasi-Albert graph because we know that it is power-law distributed. Create a BA graph using networkx with `n = 2500`, `m = 8`, and `seed=1` (adjust `n` or `m` if your computer has difficulty creating the graph). The seed will prevent the graph from changing each time you run the code.

Then build a list/array of the node degrees.

In [1]:

```
# your code here
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
BA_graph = nx.barabasi_albert_graph(n=1200, m=8, seed=1)
```

In [3]:

```
node_degree = {node: BA_graph.degree(node) for node in BA_graph}
degree = list(node_degree.values())
```

In [4]:

```
print (degree[:10])
```

```
[83, 50, 69, 86, 111, 35, 48, 84, 133, 183]
```

(2) Find the power-law cut-off

It is normally the case in power-law distributed data that there is a value below which the power-law relation does not hold. We will express this value as `xmin`. In order to fit the scaling exponent, we first need to estimate `xmin` and then discard all values in the distribution below it.

You can use Aaron Clauset's powerlaw (http://tuvalu.santafe.edu/~aaronc/powerlaws/) package to determine `xmin`. The package can be installed using pip from the commandline: `pip install powerlaw`

The package should then be available for import:

```
import powerlaw
```

And can be called using (see documentation for additional arguments (http://arxiv.org/pdf/1305.0215v3.pdf)):

```
fit = powerlaw.Fit(some_data)
print(fit.xmin)
```

Alternatively, you can get a rough estimate of the cut-off yourself by visualizing the power-law.

In [5]:

```
# Find xmin and adjust data (your code here)
import powerlaw

fit = powerlaw.Fit(degree)
print(fit.xmin)
```

```
9.0

Calculating best minimal value for power law fit
C:\Users\Bao\Anaconda3\lib\site-packages\powerlaw.py:692: RuntimeWarning: in
valid value encountered in true_divide
  (Theoretical_CDF * (1 - Theoretical_CDF))
```

(3) Find scaling exponent

With the data available and `xmin` estimated we can now estimate the scaling exponent for our degree distribution. Use the powerlaw package to estimate the scaling exponent.

In [6]:

```
# Your code here
fit = powerlaw.Fit(degree, xmin=9.0)
print(fit.power_law.alpha)
```

```
2.99321191036

C:\Users\Bao\Anaconda3\lib\site-packages\powerlaw.py:692: RuntimeWarning: in
valid value encountered in true_divide
  (Theoretical_CDF * (1 - Theoretical_CDF))
```

(4) Visualize the power-law

Plot the CCDF of the power-law along with a best-fit line made using your estimated scaling exponent. You can use your code from the power-law visualization assignment from earlier, but you will have to add the best fit line. One way to do this is to generate points for the x-axis using [`np.linspace` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html) or [`np.arange` (https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html)](https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html) and then calculate what the y-axis values should be using `xmin` and `alpha` for a straight line in the log-log CCDF plot. If the fit is good, it should fall right on-top of the empirical data-points on the CCDF plot.

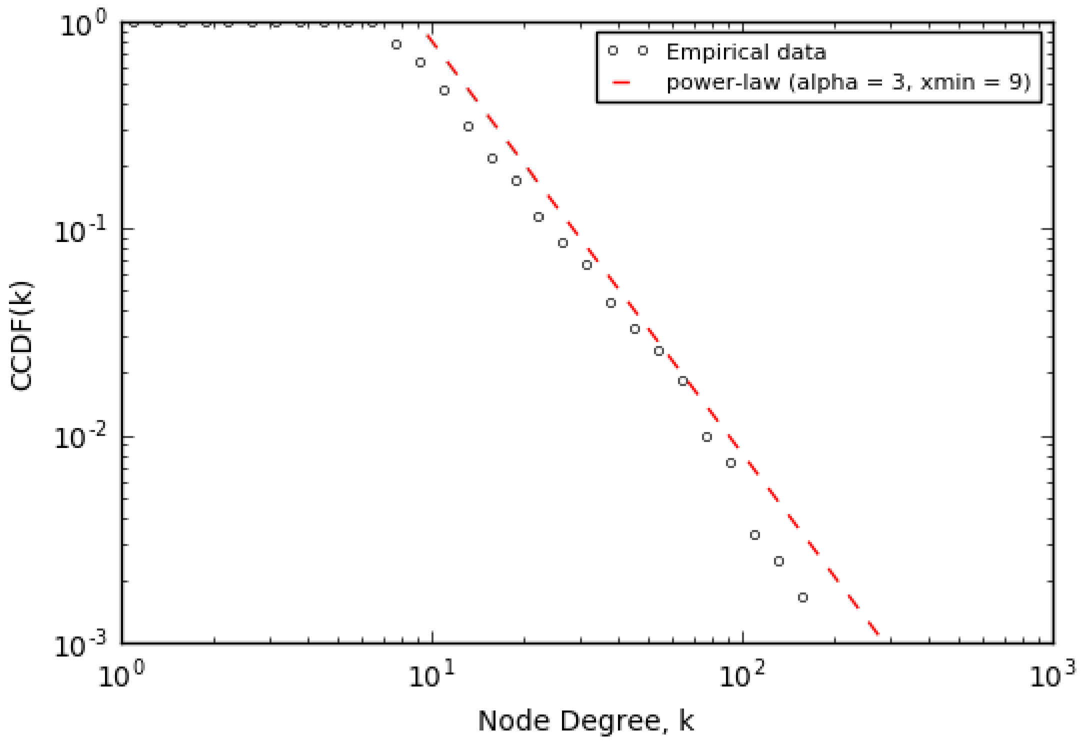
In [7]:

```
# Your visualization code here
bins = np.logspace(0.0, 3.0, num=40)
Y, X = np.histogram(degree, bins=bins, normed=False)
X = [x*np.sqrt(bins[1]) for x in X][:-1]  # find the center point for each bin.
cum = np.cumsum(Y)
ccum = 1-cum/sum(Y)
plt.ylim((0.001, 1))
plt.xlabel('Node Degree, k')
plt.ylabel("CCDF(k)")
plt.loglog(X,ccum, 'o', markersize=3, markerfacecolor='none', label='Empirical data')

x = np.logspace(0.0, 3.0, num=40)
alpha = fit.power_law.alpha
xmin = fit.xmin
y = (x/xmin)**(-alpha+1)
plt.loglog(x, y, '-r', linestyle='--', label='power-law (alpha = 3, xmin = 9)')
plt.legend(loc='upper right', fontsize=8)
```

Out[7]:

```
<matplotlib.legend.Legend at 0x1b257b85898>
```



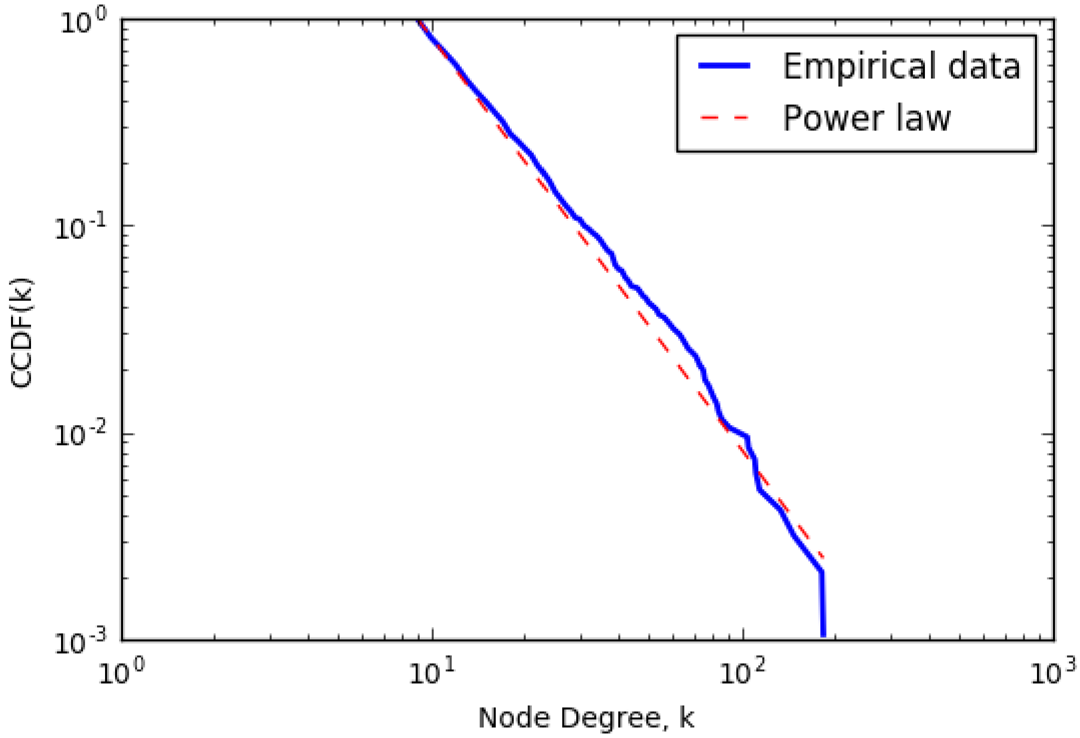
Alternative, we can use powerlaw package's built in function to draw the ccdf plot.

In [8]:

```
fig = fit.plot_ccdf(linewidth=2, label = 'Empirical data')
fit.power_law.plot_ccdf(ax=fig, color='r', linestyle='--', label = 'Power law')
plt.xlabel('Node Degree, k')
plt.ylabel("CCDF(k)")
plt.legend()
```

Out[8]:

```
<matplotlib.legend.Legend at 0x1b258742b70>
```



Using your own data

Now that you have done the fit and visualization for the BA graph, find a graph or other data and repeat the analysis. You are welcome to use data from this graph website (http://www-personal.umich.edu/~mejn/netdata/), Clauset's work (http://tuvalu.santafe.edu/~aaronc/powerlaws/data.htm), or from whatever real-world graphs you can find. Make sure to include the graph or data file when uploading this notebook (and avoid using a huge dataset with millions of nodes).

(1) Getting real-world distribution

In [9]:

```
# Your code here
fname = 'fires.txt'
!head 'fires.txt'
nums = [float(x) for x in open(fname)]
print (nums[:5], '...', nums[-5:])
```

```
0.10
0.10
0.10
0.10
0.10
0.10
0.10
0.10
0.10
0.10
[0.1, 0.1, 0.1, 0.1, 0.1] ... [177544.0, 178900.0, 187300.0, 400100.0, 41205
0.0]
```

(2) Finding cut-off

In [10]:

```
# Your code here
fit2 = powerlaw.Fit(nums)
print(fit2.xmin)
```

```
Calculating best minimal value for power law fit
C:\Users\Bao\Anaconda3\lib\site-packages\powerlaw.py:692: RuntimeWarning: in
valid value encountered in true_divide
  (Theoretical_CDF * (1 - Theoretical_CDF))
```

```
6324.0
```

(3) Estimating scaling exponent

In [11]:

```
# Your code here
fit2 = powerlaw.Fit(nums, xmin=6324.0)
print(fit2.power_law.alpha)
```

```
2.16362867855
```

```
C:\Users\Bao\Anaconda3\lib\site-packages\powerlaw.py:692: RuntimeWarning: in
valid value encountered in true_divide
  (Theoretical_CDF * (1 - Theoretical_CDF))
```

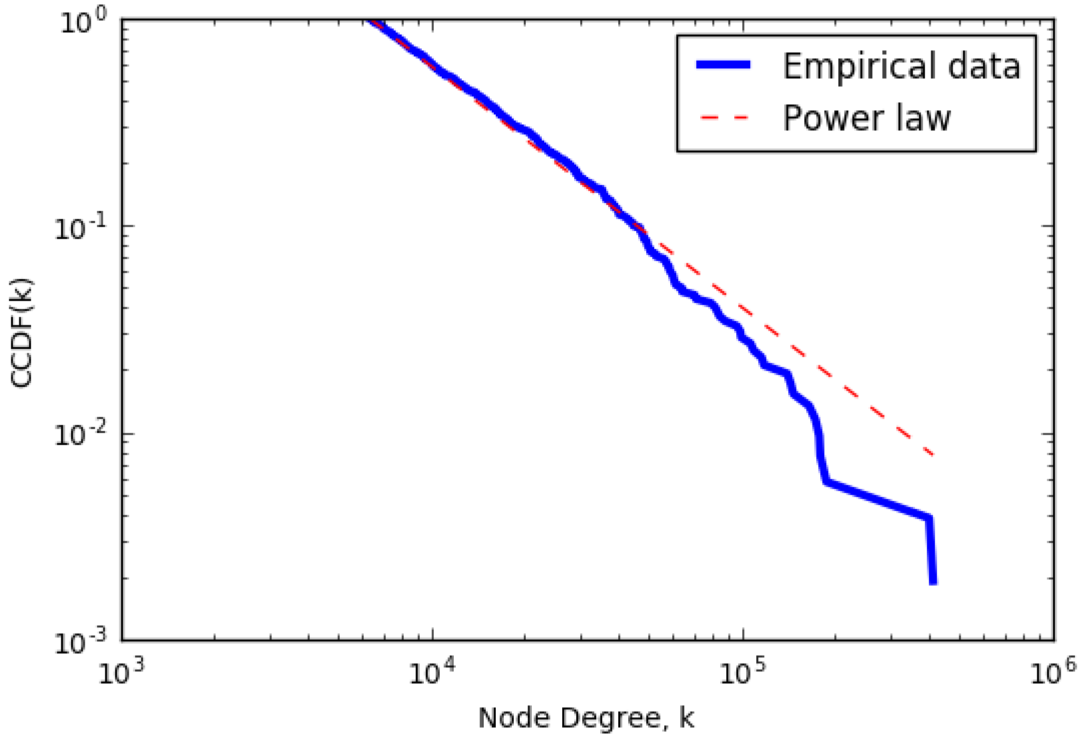
(4) Visualize power-law and best-fit

In [12]:

```
# Your code here
import pylab
fig2 = fit2.plot_ccdf(linewidth=3, label = 'Empirical data')
fit2.power_law.plot_ccdf(ax=fig2, color='r', linestyle='--', label = 'Power law')
plt.xlabel('Node Degree, k')
plt.ylabel("CCDF(k)")
plt.legend()
```

Out[12]:

```
<matplotlib.legend.Legend at 0x1b257abfa58>
```



In [ ]: