NETWORK SCIENCE OF ONLINE INTERACTIONS

Chapter 3 exercises +
Reddit primer I + Power-law tutorial

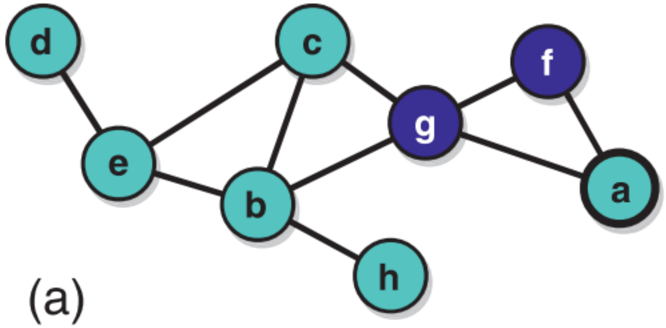
Joao Neto

10/May/2023

BOOK EXERCISES – CHAPTER 3

- Exercises: 3.4, 3.10, 3.19, 3.23

3.4 In NetworkX, how can you find a node with the largest degree centrality in a network? And how would you also get the degree of that node?



(a)

```python
G = nx.Graph()
G.add_edges_from([('d','e'),('e','c'),('e','b'),('c','b'),('c',
'g'),('b','g'),('b','h'),('g','f'),('g','a'),('f','a')])
max(G.nodes, key=G.degree)
```
✓ 0.1s Python

```
'b'
```

```python
G.degree('b')
```
✓ 0.1s Python

```
4
```

```python
import pandas as pd
df = pd.DataFrame(list(G.degree()), columns=['node', 'degree'])
```
✓ 0.1s Python

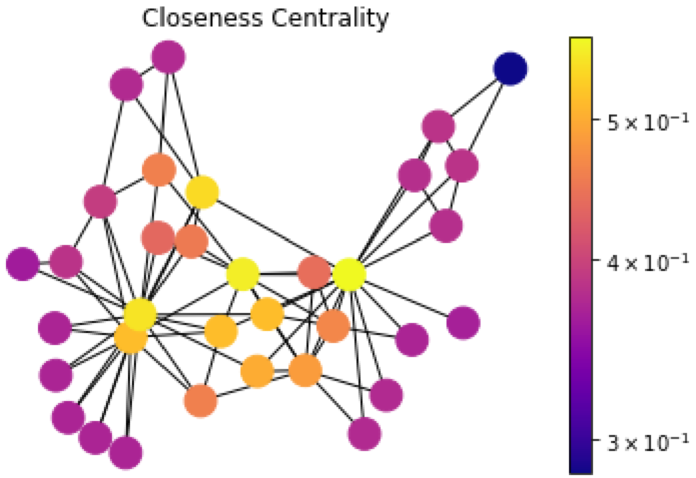
```python
df.sort_values(by='degree', ascending=False).head(3)
```
✓ 0.0s Python

	node	degree
3	b	4
4	g	4
1	e	3

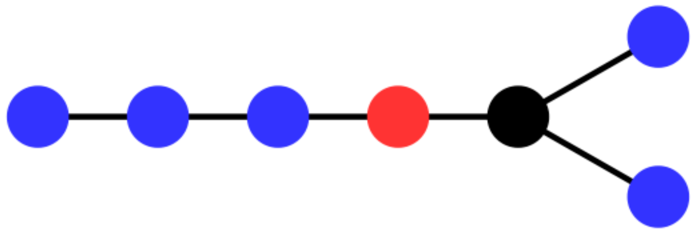
BOOK EXERCISES – CHAPTER 3

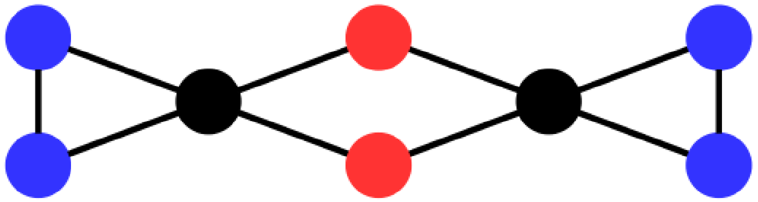
3.10 Provide examples of networks such that:

1. The node with the highest degree is not the one with largest closeness
2. The node with the highest betweenness is not the one with largest closeness

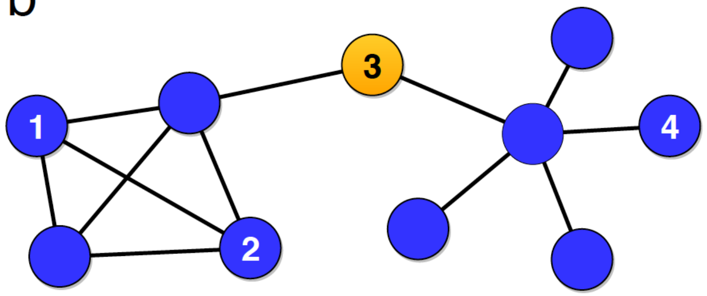


- 1. Closeness: central node has short paths to everyone
- 2. Betweenness: central node is in many shortest paths





b



BOOK EXERCISES – CHAPTER 3

3.23 Consider two nodes of equal degree on some network: one with high clustering coefficient and one with low clustering coefficient. All else being equal, which of the two would you intuit to be a better target if you were seeking to disrupt the network?

- Low clustering. High clustering means more connected neighbours, so less disruption if removed.

BOOK EXERCISES – CHAPTER 3

3.19 Write a Python function that accepts a NetworkX graph and a node name and returns the average degree of that node's neighbors. Use this function to compute this quantity for every node in the OpenFlights US network and take the average. Does the Friendship Paradox hold here (i.e. is the average degree of nearest neighbors greater than the average node degree)?

- Get dataset from book repo :
  - https://github.com/CambridgeUniversityPress/FirstCourseNetworkScience/tree/master/datasets/openflights
- This exercise: course repo
  - https://github.com/joaopn/teaching_networks_2023

BOOK EXERCISES – CHAPTER 3

- What is it?

```python
# Import data
import networkx as nx
G = nx.read_edgelist('data/openflights_usa.edges')
```

✓ 0.1s Python

```python
# Basic statistics
print('Nodes in G:', G.number_of_nodes())
print('Edges in G:', G.number_of_edges())
print('Is G connected?', nx.is_connected(G))
print('Number of connected components in G:', nx.number_connected_components(G))
print('What is the size of the giant component?', len(max(nx.connected_components(G), key=len)))
```

✓ 0.1s Python

```
Nodes in G: 546
Edges in G: 2781
Is G connected? False
Number of connected components in G: 3
What is the size of the giant component? 539
```

BOOK EXERCISES – CHAPTER 3

- What are the important nodes?

```python
# Calculate degree, close and betweenness centrality for all nodes, and
store the results in a pandas DataFrame
import pandas as pd
import airportsdata

df = pd.DataFrame({'degree': nx.degree_centrality(G),'closeness': nx.
closeness_centrality(G), 'betweenness': nx.betweenness_centrality(G)})

# Adds the name and city of each airport to the DataFrame
airports = pd.DataFrame(airportsdata.load('IATA')).T
df = df.join(airports[['name', 'city']])
df
```

✓ 2.1s Python

	degree	closeness	betweenness	name	city
RDD	0.001835	0.299374	0.000000	Redding Municipal Airport	Redding
SFO	0.130275	0.429337	0.025286	San Francisco International Airport	San Francisco
EUG	0.016514	0.356676	0.000154	Mahlon Sweet Field	Eugene
SLC	0.155963	0.467098	0.056644	Salt Lake City International Airport	Salt Lake City
AZA	0.058716	0.338489	0.005401	Phoenix-Mesa Gateway Airport	Phoenix
...	...	...	...	...	...

```python
df.sort_values(by='degree', ascending=False)[['name', 'city',
'degree']].head(3)
```

✓ 0.1s Python

	name	city	degree
ATL	Hartsfield - Jackson Atlanta International Air...	Atlanta	0.280734
ORD	Chicago O'Hare International Airport	Chicago	0.273394
DEN	Denver International Airport	Denver	0.271560

```python
df.sort_values(by='closeness', ascending=False)[['name', 'city',
'closeness']].head(3)
```

✓ 0.1s Python

	name	city	closeness
DEN	Denver International Airport	Denver	0.503880
ORD	Chicago O'Hare International Airport	Chicago	0.501975
MSP	Minneapolis-St Paul International/Wold-Chamber...	Minneapolis	0.487238

```python
df.sort_values(by='betweenness', ascending=False)[['name', 'city',
'betweenness']].head(3)
```

✓ 0.1s Python

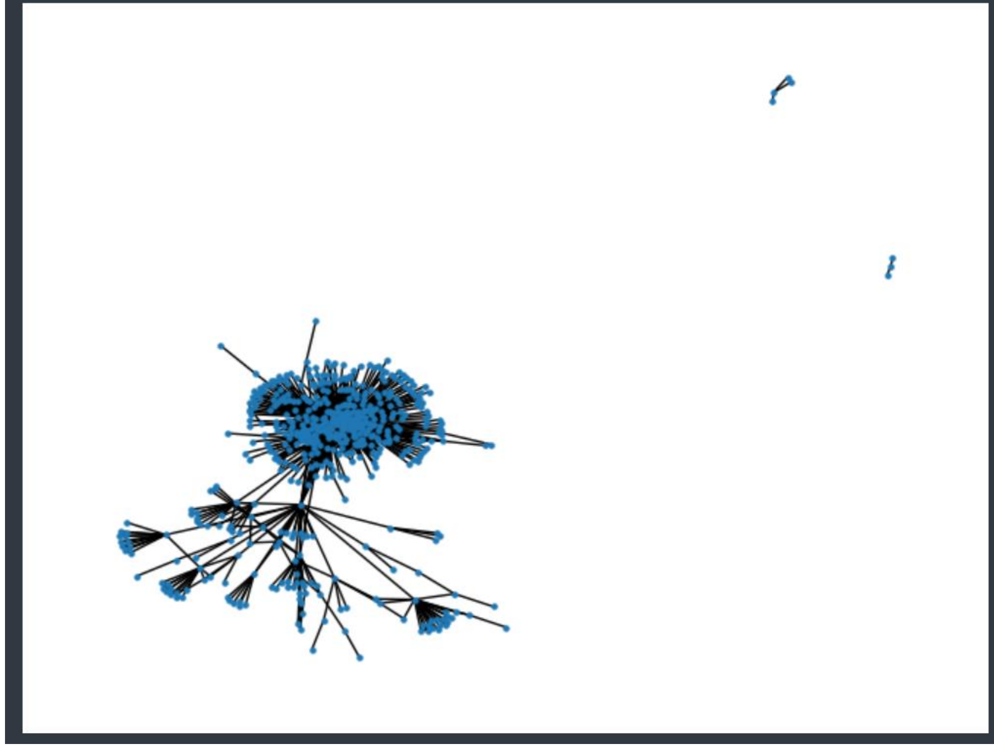
	name	city	betweenness
ANC	Ted Stevens Anchorage International Airport	Anchorage	0.318991
DEN	Denver International Airport	Denver	0.150853
ORD	Chicago O'Hare International Airport	Chicago	0.126094

BOOK EXERCISES – CHAPTER 3

- What does it look like?
  - Looks connected with kamada_kawai

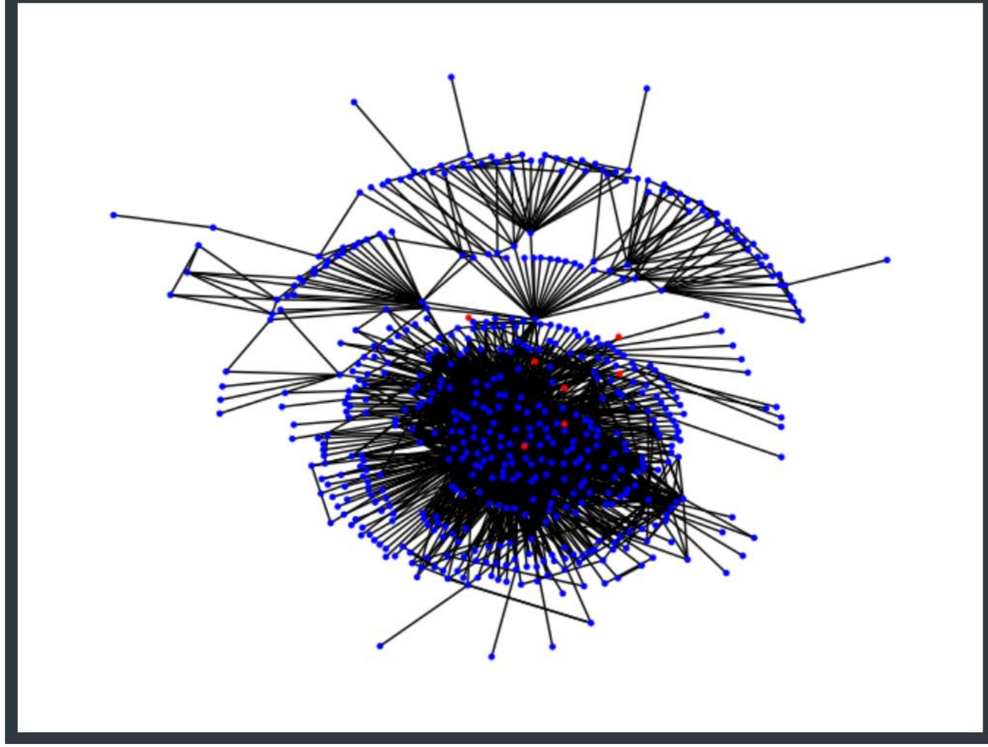
```python
#draw the giant component of the graph
nx.draw(G, node_size=5)
```

✓ 2.1s Python



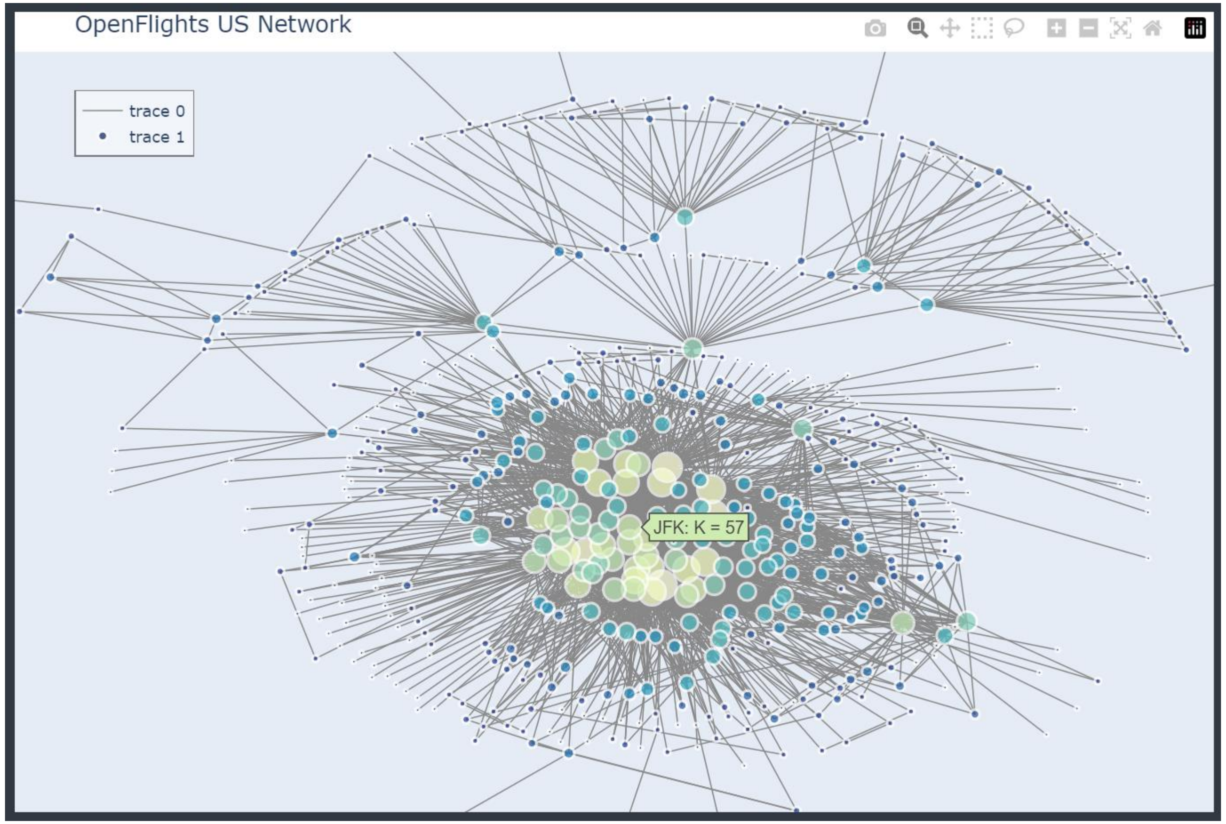
```python
#draw with the kamada_kawai layout, painting the nodes not in the giant component red
nx.draw_kamada_kawai(G, node_size=5, node_color=['red' if node not in max(nx.connected_components(G), key=len) else 'blue' for node in G.nodes()])
```

✓ 9.4s Python



BOOK EXERCISES – CHAPTER 3

- Can we plot it interactively?
  - Yes: Bokeh, Plotly




```python
import networkx as nx
import numpy as np
import plotly.graph_objects as go

# Compute the node degrees and create a list of node sizes proportional to log(degree)
degree = dict(G.degree)
node_sizes = [10*np.log10(degree[node]+1) for node in G.nodes]

# Create a list of node labels
labels = [f'{i}: K = {degree[i]}' for i in G.nodes]

# Get node positions
pos = nx.kamada_kawai_layout(G)

# Create edge trace
edge_trace = go.Scatter(x=[], y=[], line=dict(width=1, color='#888'), hoverinfo='none',
mode='lines')

for edge in G.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_trace['x'] = tuple(list(edge_trace['x']) + [x0, x1, None])
    edge_trace['y'] = tuple(list(edge_trace['y']) + [y0, y1, None])

# Create node trace
node_trace = go.Scatter(x=[], y=[], text=[], mode='markers+text', hoverinfo='text',
hovertext=labels, marker=dict(showscale=False, colorscale='YlGnBu', reversescale=True,
size=node_sizes, color=node_sizes, line=dict(width=2)))

for node in G.nodes():
    x, y = pos[node]
    node_trace['x'] = tuple(list(node_trace['x']) + [x])
    node_trace['y'] = tuple(list(node_trace['y']) + [y])

# Create the plot
fig = go.Figure(data=[edge_trace, node_trace],
             layout=go.Layout(title='OpenFlights US Network',
                             xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
                             yaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
                             legend=dict(x=0.05, y=0.95, bgcolor='rgba(255, 255, 255, 0.5)',
                             bordercolor='rgba(0, 0, 0, 0.5)', borderwidth=1)))


# Show the plot. This can be redone in a new cell without recalculating everything else
fig.update_layout(dict(width=900, height=600, autosize=False), margin=dict(l=0, r=0, t=30, b=0))
fig.show()
```

✓ 33.7s Python

BOOK EXERCISES – CHAPTER 3

- How big is the k-core?
  - 92% of max links $(36 \times 35/2 = 630)$

```
# Computes the the fraction of nodes/edges in the k-core
kcore = nx.k_core(G)

k_core_nodes = kcore.number_of_nodes()
k_core_edges = kcore.number_of_edges()
k_core_nodes_fraction = k_core_nodes/G.number_of_nodes()
k_core_edges_fraction = k_core_edges/G.number_of_edges()

print('{:d} nodes in the k-core ({:0.2f}% of the nodes)'.
format(k_core_nodes, 100*k_core_nodes_fraction))
print('{:d} edges in the k-core ({:0.2f}% of the edges)'.
format(k_core_edges, 100*k_core_edges_fraction))
```

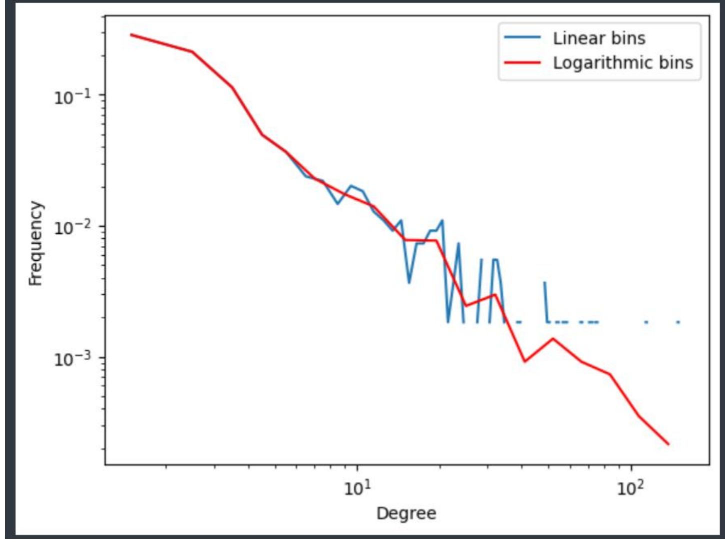
✓ 0.1s Python

```
36 nodes in the k-core (6.59% of the nodes)
580 edges in the k-core (20.86% of the edges)
```

- What is the degree distribution?
  - Later: how to test if it is a power-law

```
# Computes the degree distribution of the network
import powerlaw

degree_dist = [G.degree[node] for node in G.nodes]

fig = plt.figure()
powerlaw.plot_pdf(degree_dist, linear_bins=True, **{'label': 'Linear'})
powerlaw.plot_pdf(degree_dist, linear_bins=False, color='r', **{'label': 'Logarithmic'})
plt.legend()
plt.xlabel('Degree')
plt.ylabel('Frequency');
```

✓ 0.3s Python



BOOK EXERCISES – CHAPTER 3

3.19 Write a Python function that accepts a NetworkX graph and a node name and returns the average degree of that node's neighbors. Use this function to compute this quantity for every node in the OpenFlights US network and take the average. Does the Friendship Paradox hold here (i.e. is the average degree of nearest neighbors greater than the average node degree)?

```python
# Computes the average degree of the neighbors of a node
def avg_degree_neighbors(G , friend):
    if G.degree(friend) > 0:
        k_nn = 0
        for node in G.neighbors(friend):
            k_nn += G.degree(node)/G.degree(friend)
        return k_nn
    else:
        print('The k_nn of ',friend ,'is undefined because ',friend,'has no neighbors !')
```

✓ 0.1s Python

```python
knn_avg = 0
for node in G.nodes():
    knn_avg += avg_degree_neighbors(G,node)
knn_avg = knn_avg/G.number_of_nodes()

print('Average k_nn:', knn_avg)
print('Average degree:', 2*G.number_of_edges()/G.number_of_nodes())
```

✓ 0.1s Python

```
Average k_nn: 64.04614431282478
Average degree: 10.186813186813186
```

```python
#get the knn of each node
knn = nx.average_neighbor_degree(G)
np.mean(list(knn.values()))
```

✓ 0.1s

```
64.04614431282477
```

BOOK EXERCISES – CHAPTER 3

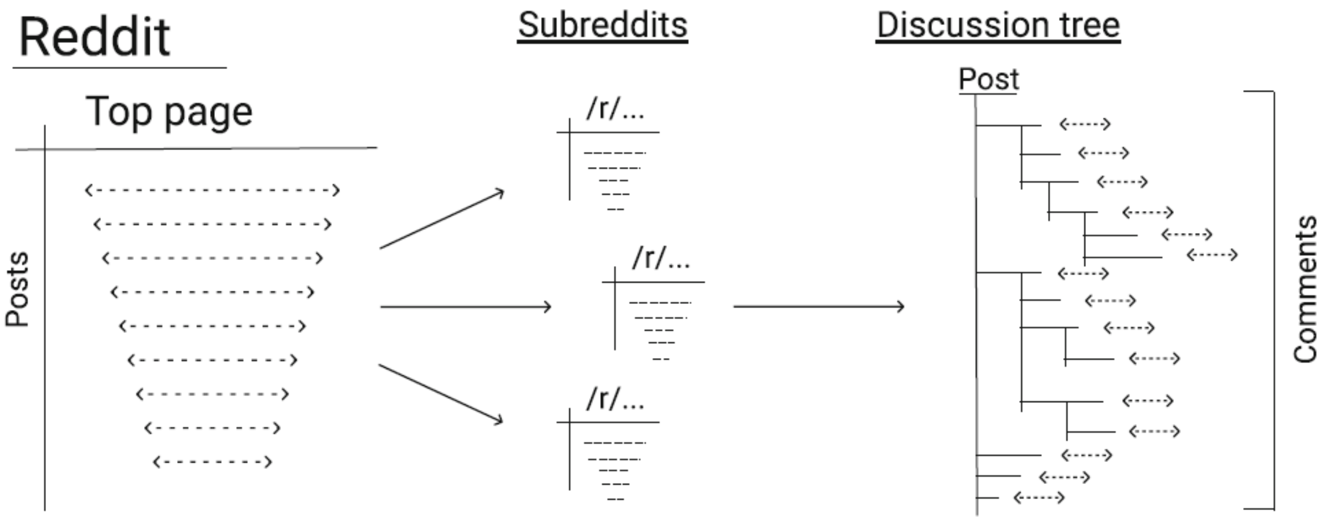
- Questions?

REDDIT PRIMER I

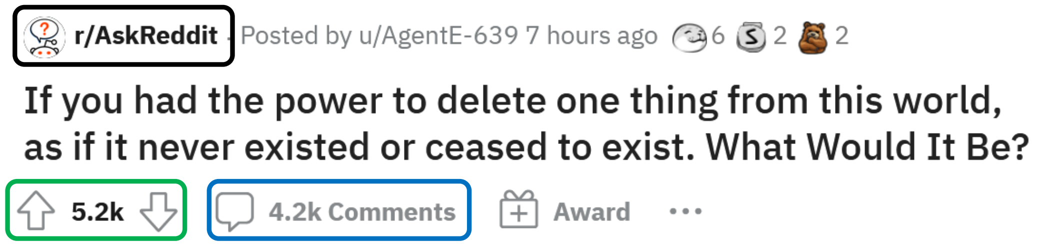
- Reddit
  - Founded in 2005
  - 10th most visited website in the world (6th in the US)
  - 52M daily users (in 2020)
  - Valued at ~15B USD
- Pushshift
  - Data/API service
  - Lots of data from different places (Twitter, StackOverflow, etc)
  - Reddit data killed by Reddit (as of this week)
  - Data dump torrents are still up

REDDIT STRUCTURE

- How does Reddit work?
- Both self-referential and reactive content
- Main features of a discussion
  - Subreddit
  - Number of comments
  - Score



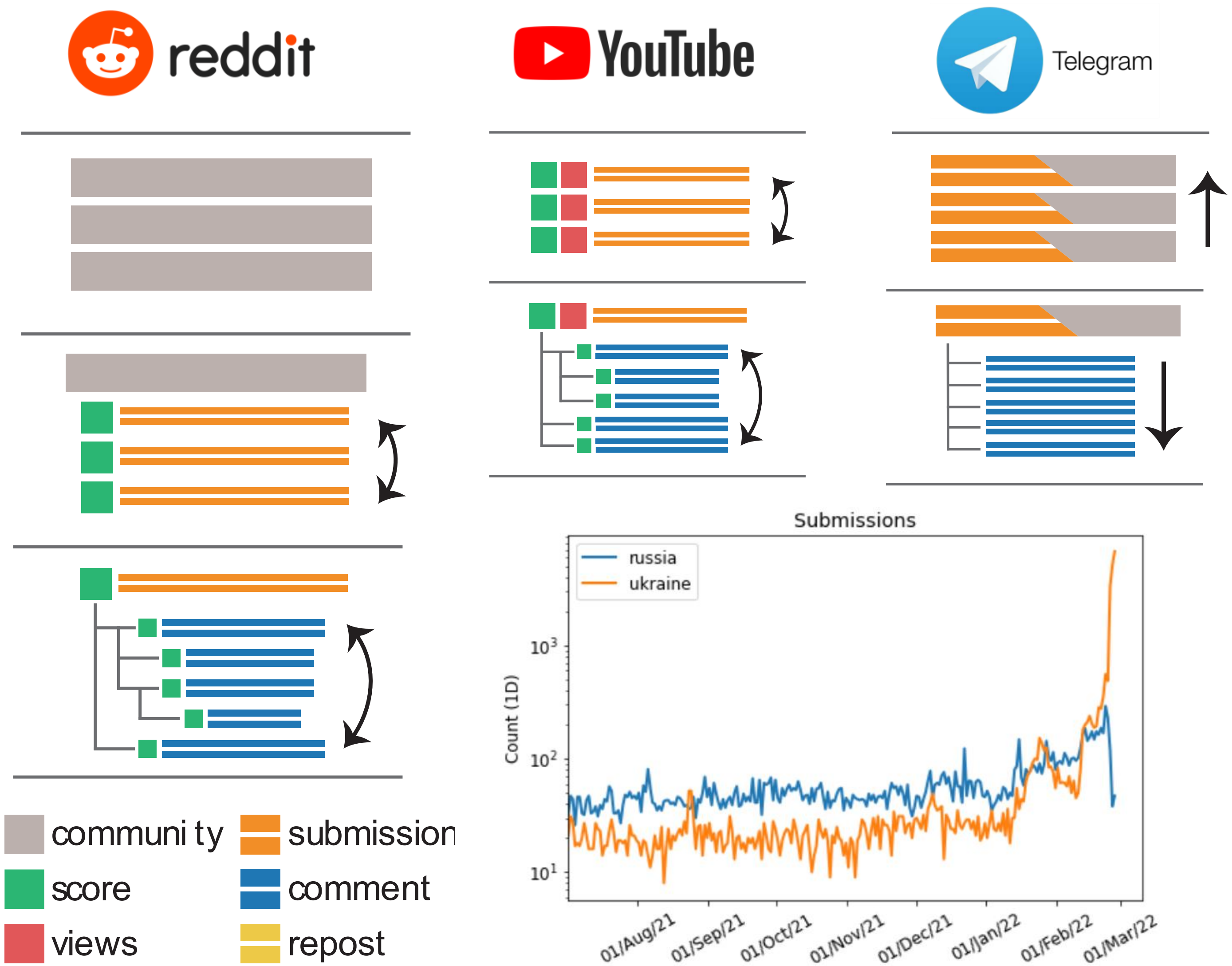
Source: [1]



[1] Medvedev *et al*, Journal of Complex Networks **7**, 67 (2019).

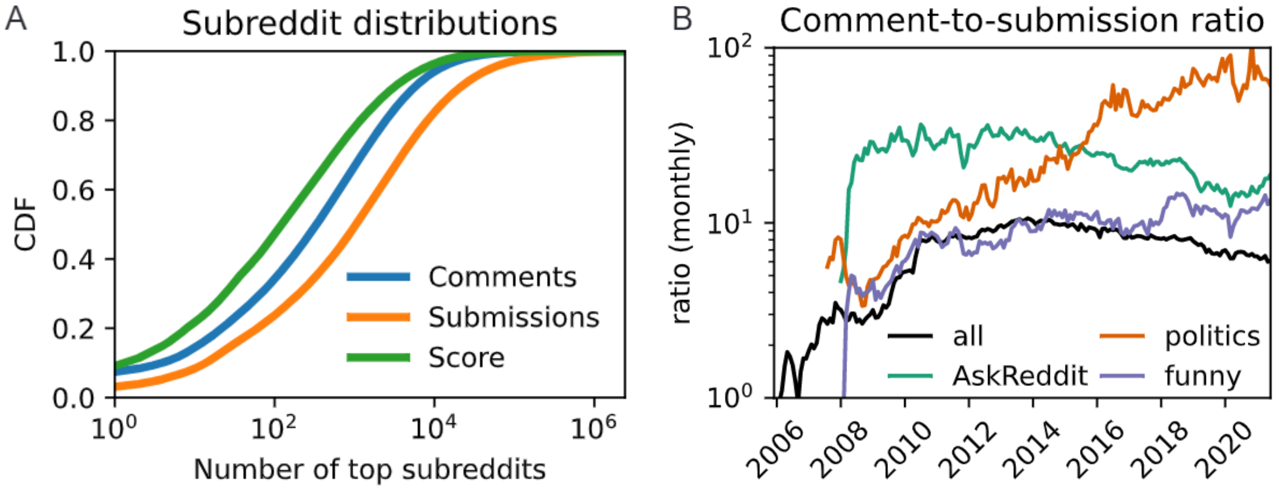
SOCIAL MEDIA PLATFORM STRUCTURE

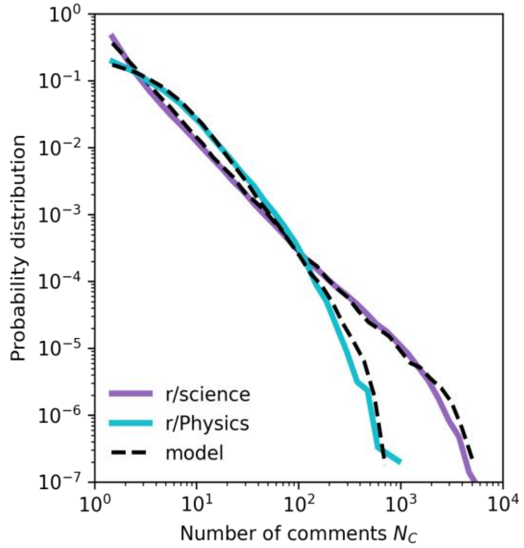
- Content is sorted
  - Globally (e.g. Reddit*)
  - Individually (e.g. Tiktok)
- Structure varies
  - Twitter-like: follow users
  - Reddit-like: follow communities
- Content moderation differs
  - all levels (e.g. StackExchange)
  - submission (e.g. Youtube)
  - none (e.g. Telegram)

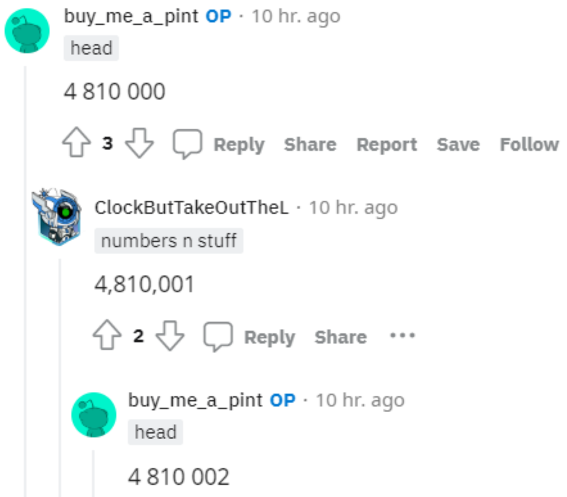


REDDIT DESCRIPTION

- Reddit is unique
  - Relatively simple, open-source algorithm
  - Influential (The_Donald, wallstreetbets)
  - **fully-sampled** dataset (18B items)
  - Labelled content (subreddit), **anonymous** interactions
- Reddit analysis
  - Characterization of platform statistics
  - Content is highly concentrated
  - Communities are very heterogeneous
  - Some have inorganic dynamics

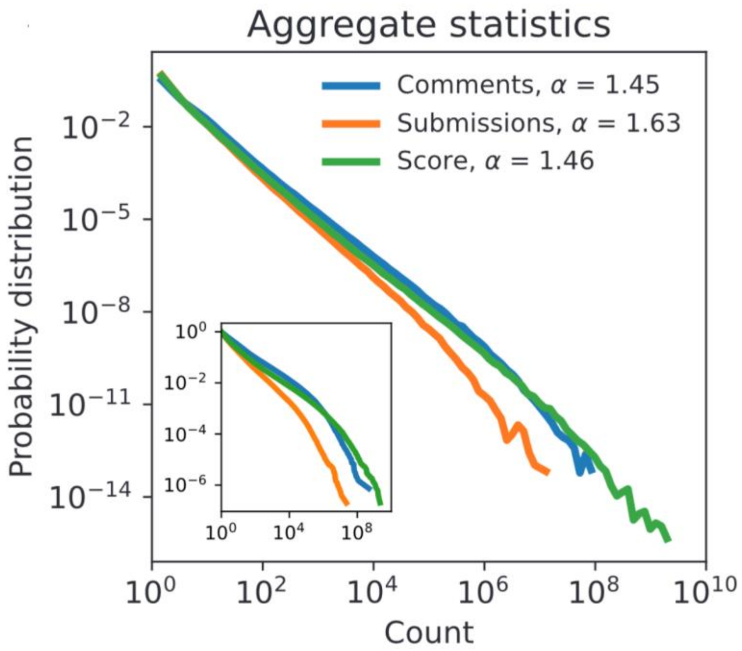




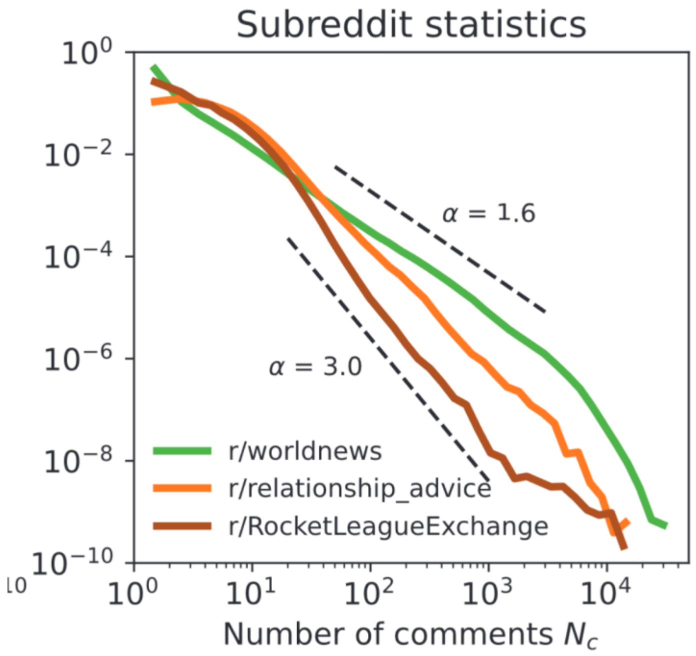


REDDIT DESCRIPTION

- Reddit is made of communities (subreddits)
  - ~4M subreddits
  - Sum of comments, submissions and score of subreddits also follow power-laws
  - Large variability of subreddit size

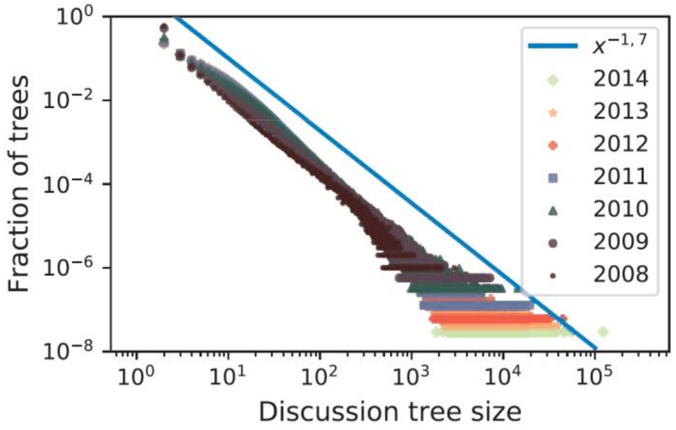


- Is there any variability in e.g. comment distribution beyond finite-size effects?
  - Yes, both in exponent α and shape
  - Requires a more sophisticated process to be explained



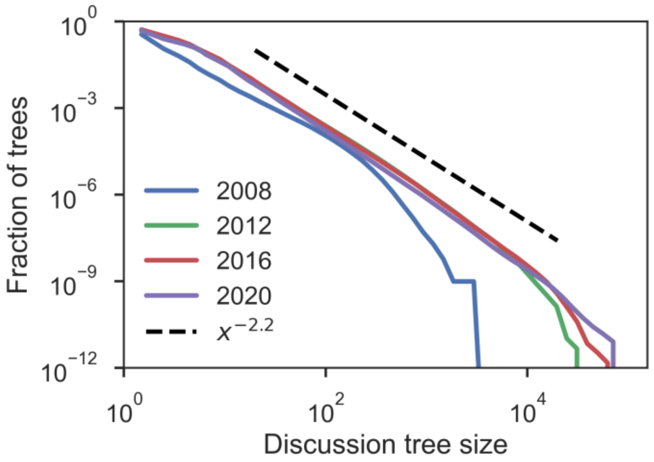
REDDIT DESCRIPTION

- Literature
  - Focus on dynamical aspects
  - Discussion size distribution follows a power-law $\mathrm{P}(\xi > x) \sim x^{-\alpha}$ with $\alpha \approx 1.7$ [1]
  - Suggests well-known processes
    - Preferential attachment trees
    - Critical branching processes



Source: [1]

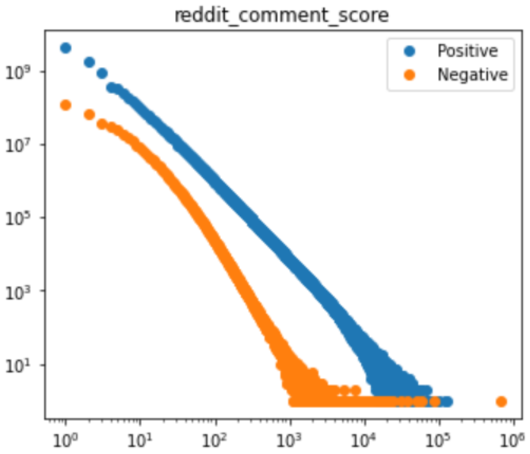
- New data
  - Power-law behavior holds with > 10x more data
  - Apparent heavier exponent ($\alpha \approx 2.2$)

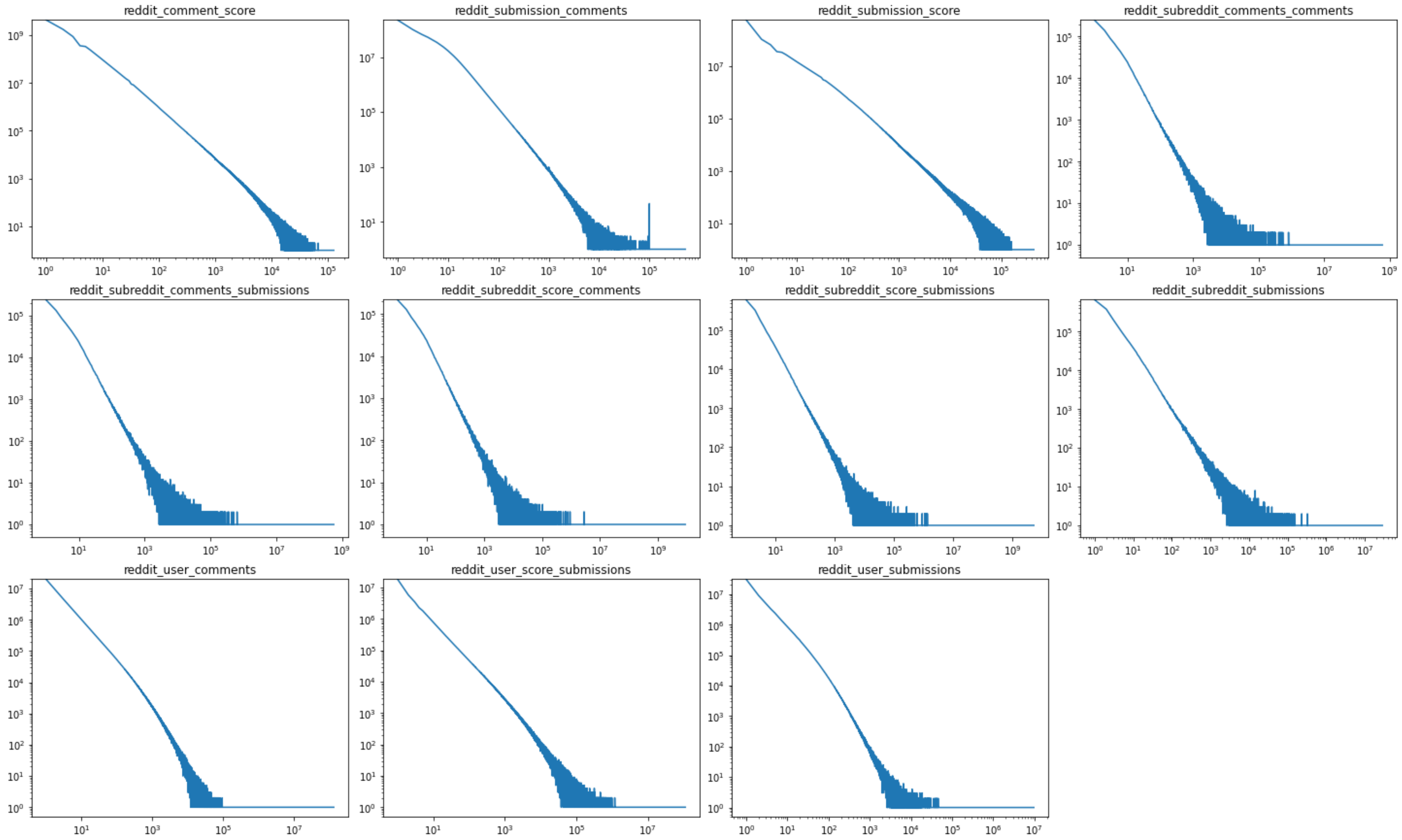


[1] Medvedev *et al*, Journal of Complex Networks **7**, 67 (2019).

REDDIT DISTRIBUTIONS

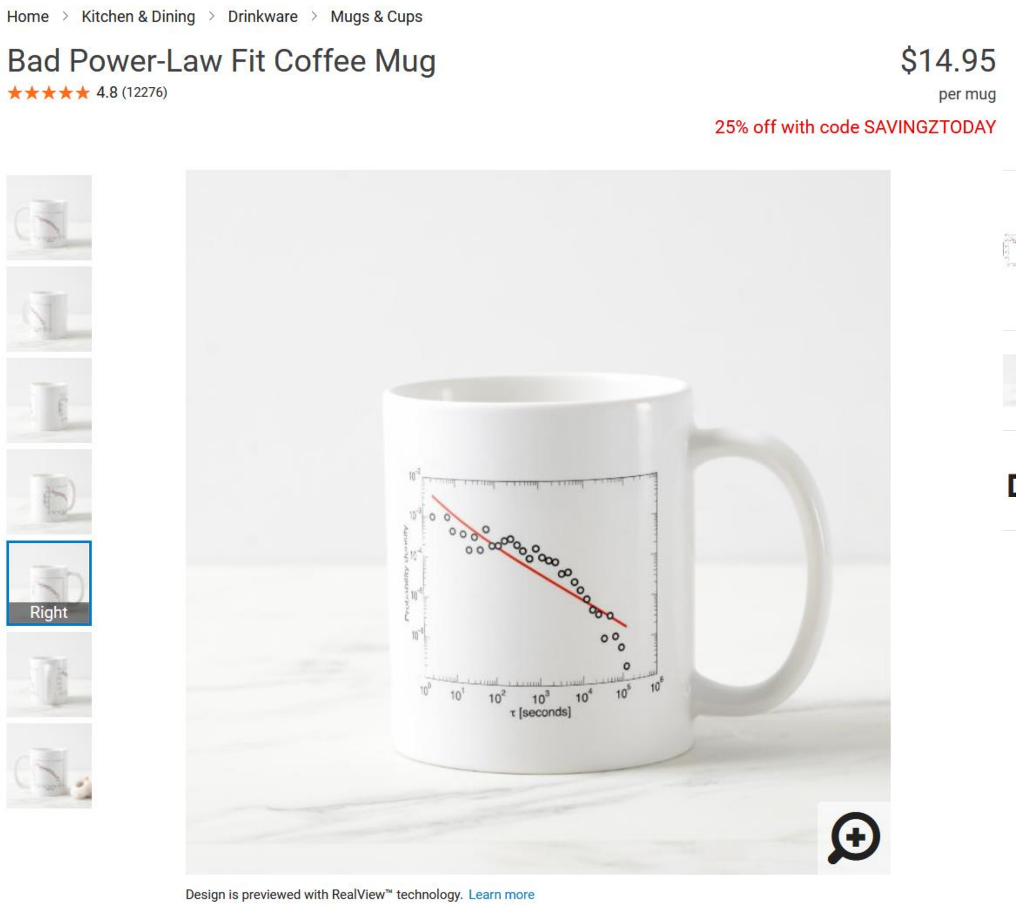
- Powerlaws!
- Various exponents
- Negative score values follow a somewhat different distribution





POWER-LAW DISTRIBUTIONS

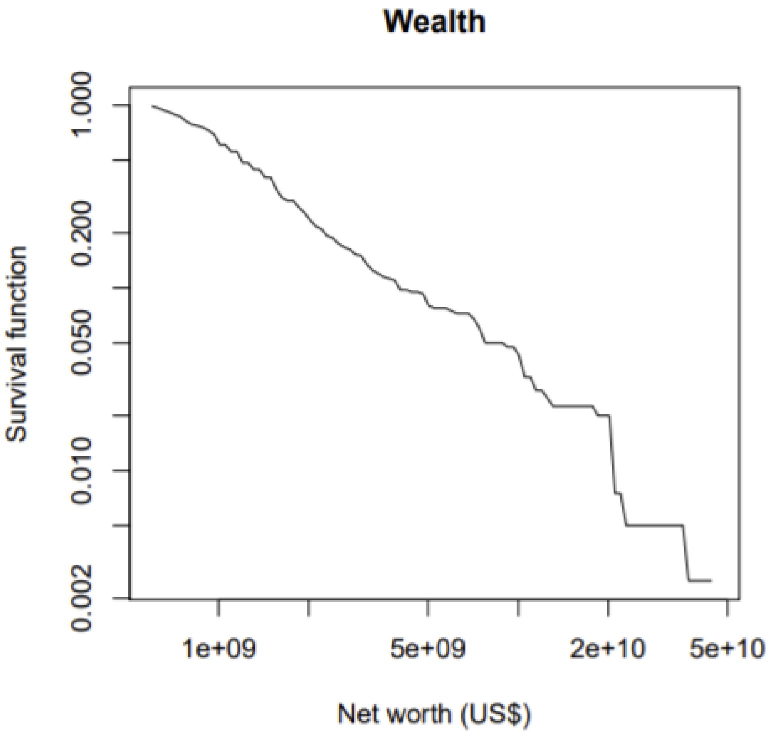
- They appear a lot
- Important to know how to handle them properly



POWER-LAW DISTRIBUTIONS

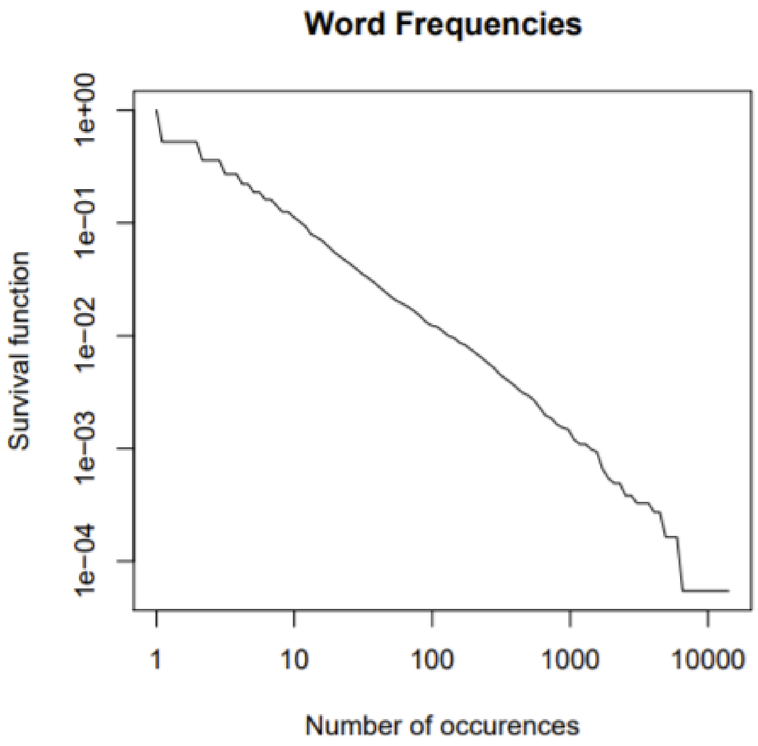
- Power-law: $f(x) \sim x^{-\alpha}$

Pareto's law



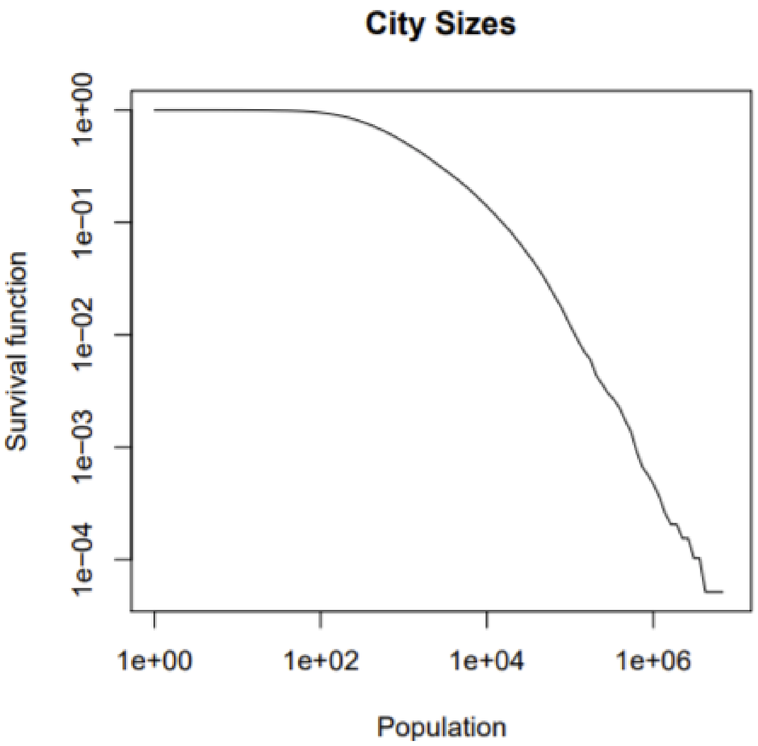
(richest 400 in the US, 2003)

Zipf's law



(Moby dick)

Pareto's law

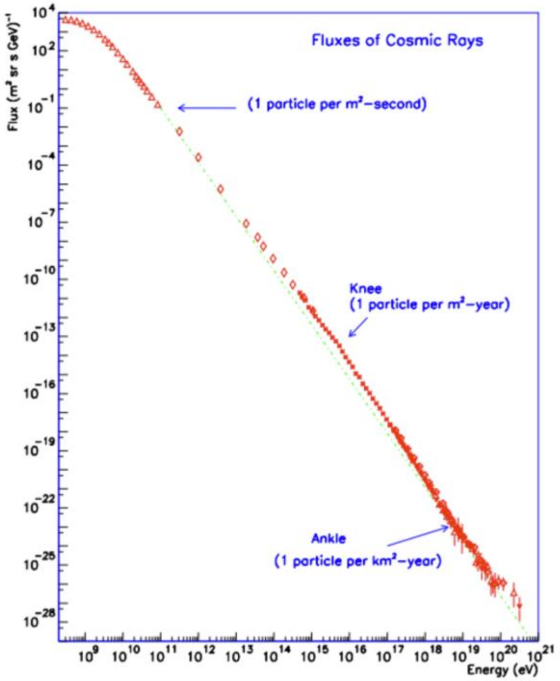


(US census, 2000)

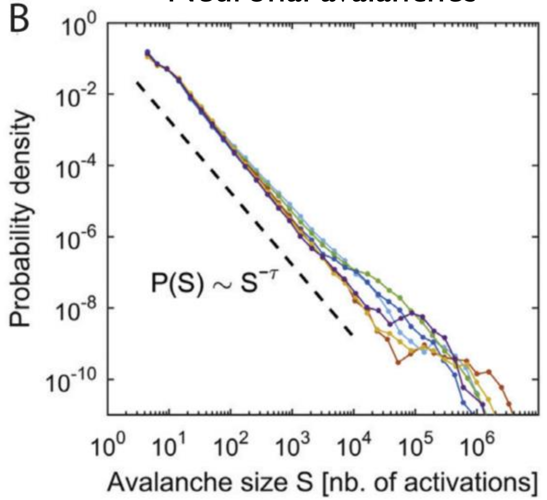
POWER-LAW DISTRIBUTIONS

- The idea
  - Power-law distributions happen *a lot*
  - You (usually) want two things:
    - Test if the thing is a power-law
    - Calculate the power-law exponent
- The problem: fitting heavy-tailed distributions is problematic
  - Fitting is largely dictated by the tail, where you have orders of magnitude less data
  - Many things can generate apparent straight lines on a log-log plot
  - Standard methods (e.g. least-squares) fail

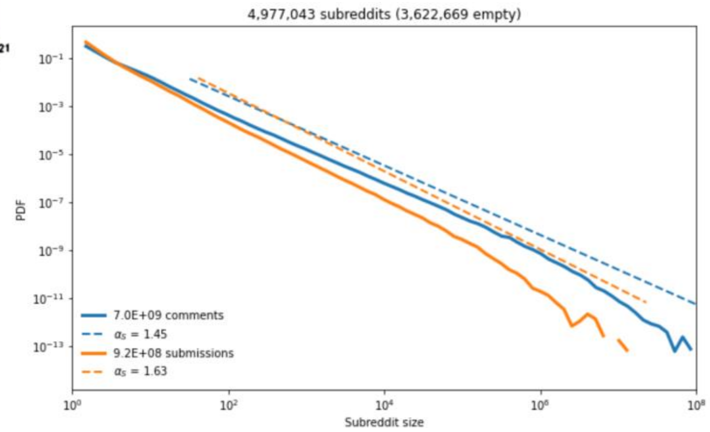
Cosmic ray flux



Neuronal avalanches

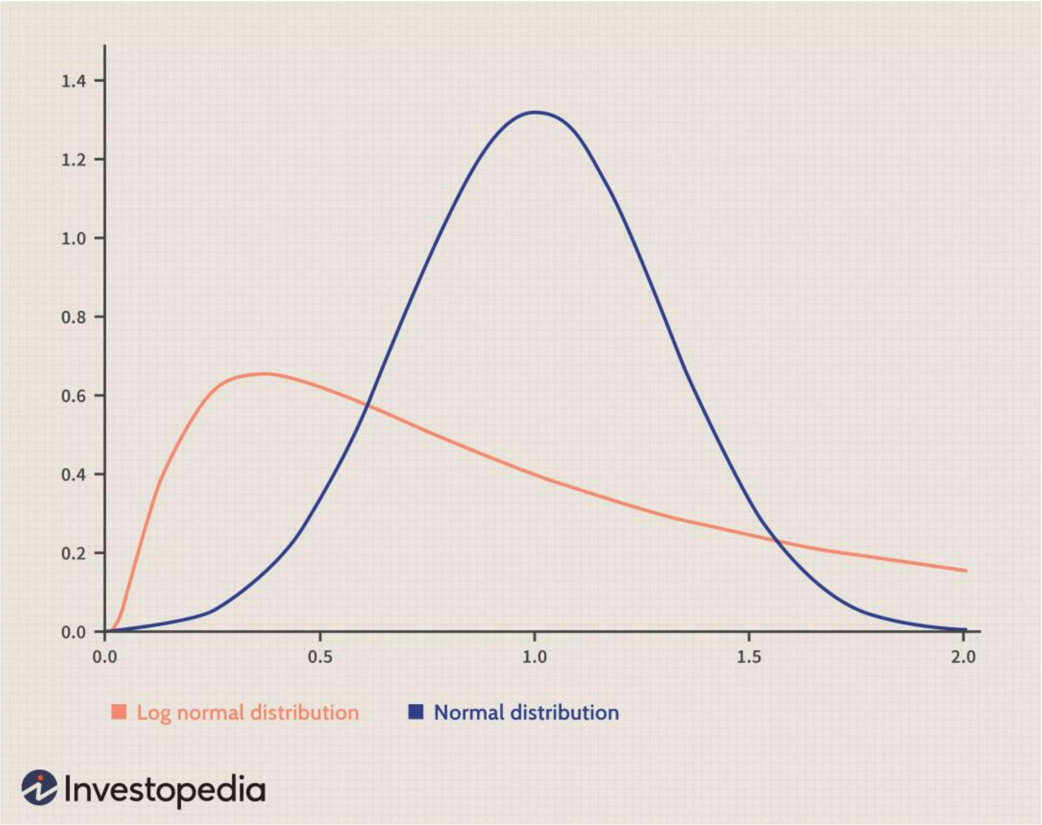


Reddit dynamics



WHY YOU WANT A POWER-LAW?

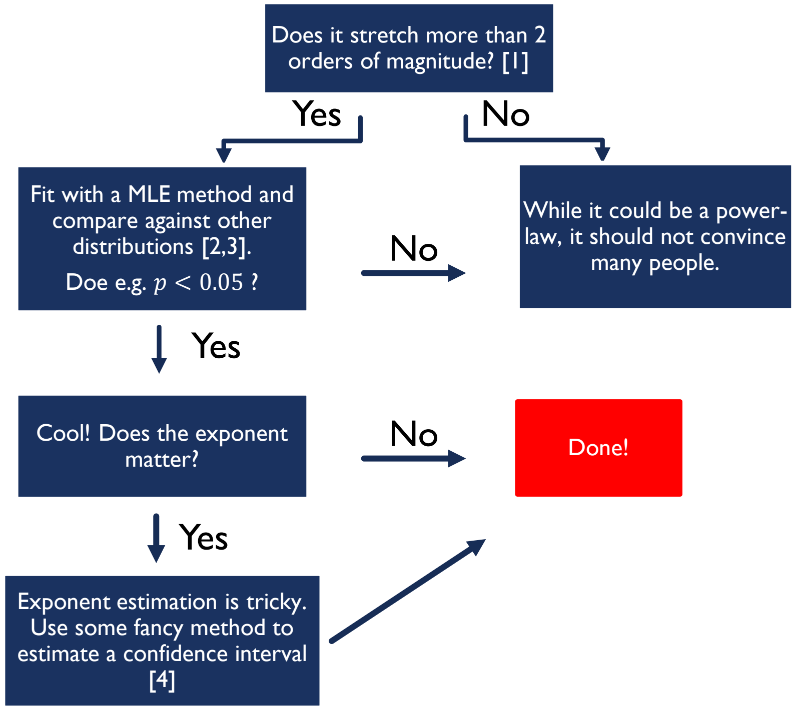
- A thing Z made of random variables X_i
- $\sum X_i \rightarrow$ Gaussian
  - $f(x) \sim e^{-(x-\mu)^2/2\sigma^2}$
  - Well-characterized by mean and variance
- $\Pi X_i \rightarrow$ Log-normal
  - $f(x) \sim \frac{1}{x} e^{-(\ln x-\mu)^2/2\sigma^2}$
- Power-laws require more exotic mechanisms
  - Phase transitions
  - Rather fine-tuned combinations of exponentials [1]



[1] Priesemann, V., & Shriki, O. (2018). *PLOS Computational Biology*, *14*(5), e1006081.

POWER-LAW FITTING

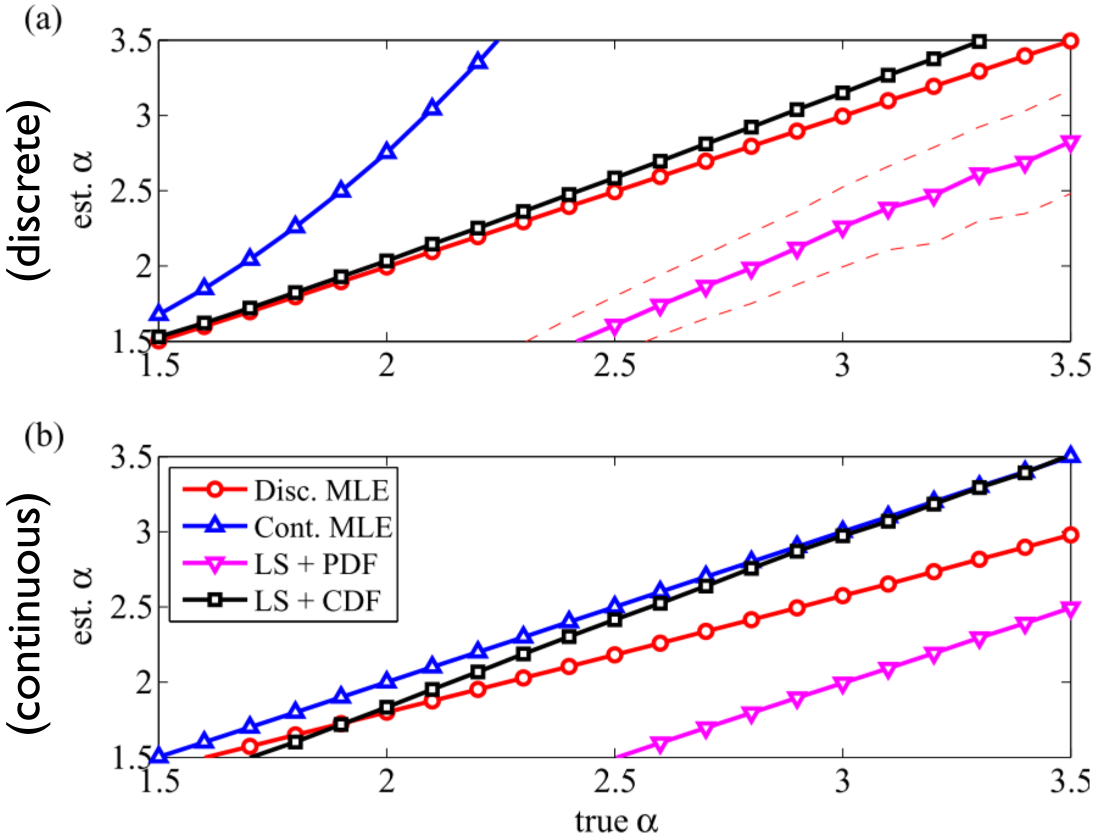
Is that a power-law distribution?



[1] Stumpf, M. P. H. H., & Porter, M. A. (2012). *Science*, *335*(6069), 665–666. https://doi.org/10.1126/science.1216142
[2] Clauset, A., Shalizi, C. R., & Newman, M. E. J. (2009). *SIAM Review*, *51*(4), 661–703. https://doi.org/10.1137/070710111
[3] Hanel, R., Corominas-Murtra, B., Liu, B., & Thurner, S. (2017). *PLoS ONE*, *12*(2), 1–15. https://doi.org/10.1371/journal.pone.0170920
[4] Goldstein, M. L., Morris, S. A., & Yen, G. G. (2004). *The European Physical Journal B*, *41*, 255–258. http://arxiv.org/abs/cond-mat/0402322v1

HOW TO FIT

- The method: maximum likelihood estimation (MLE)
  - Given data, estimate the most likely parameters to have generated it
  - Calculate a (log) likelihood function, maximize it
- Conclusions:
  - Least-squares (LS) on the PDF fails catastrophically, and is biased but less bad on the CDF
  - MLE works on its appropriate type (continuous/discrete)
  - LS + CDF doesn't look so bad, but requires a pure power-law



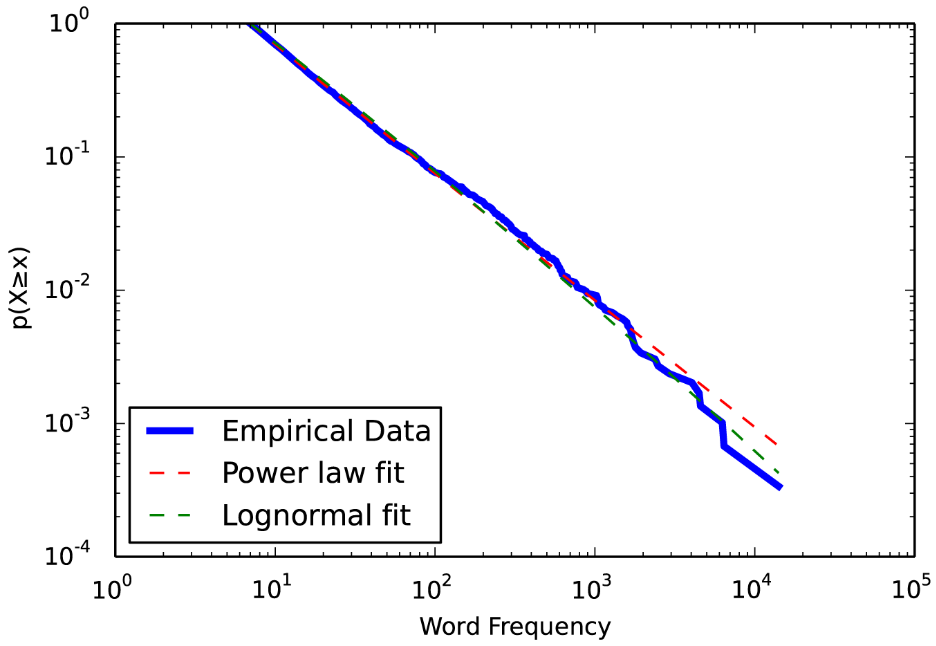
Aaron Clauset @aaronclauset · May 9
My favorite reviewer insults ("comments") were from a physics journal:
"This paper contains no physics, neither old or new." +
"These issues seem like a mistake a first year grad student might make, but not a second year."
Both for this paper 😂 arxiv.org/abs/0706.1062

Power-law distributions in empirical data - Clauset - Cited by 10552

Clauset, A., Shalizi, C. R., & Newman, M. E. J. (2009). *SIAM Review*, *51*(4), 661–703. https://doi.org/10.1137/070710111

WHY COMPARE TO OTHER DISTRIBUTIONS?

- What Clauset et al [1] suggest:
  - Use the Kolmogorov-Smirnov (KS) statistic as a goodness-of-fit test to validate the data comes from a power-law, calculating a p-value from bootstrapping
- What Alstott et al [2] suggest:
  - Compare the various distributions, and select one if it is a much better explanation for the data (again a p-value)
- The reasoning for [2]:
  - Empirical distributions are never going to be pure power-laws
  - Thus, rejecting the hypothesis of the data coming from a power-law is just a matter of gathering enough data
  - Better: finding out which model-motivated distribution better fits the data



[1] Clauset, A., Shalizi, C. R., & Newman, M. E. J. (2009). *SIAM Review*, *51*(4), 661–703. https://doi.org/10.1137/070710111
[2] Alstott, J., Bullmore, E., & Plenz, D. (2014). *PLoS ONE*, *9*(1), e85777. https://doi.org/10.1371/journal.pone.0085777

THE POWERLAW PACKAGE

- The powerlaw package does everything for you

```python
import powerlaw

# Generates synthetic data
data = powerlaw.Power_Law(xmin=1, parameters=[2.5]).generate_random(20000)

# Fits data to power law
fit = powerlaw.Fit(data)

print('xmin:', fit.xmin)
print('alpha:', fit.alpha)
```

✓ 5.0s Python

```
Calculating best minimal value for power law fit
xmin: 1.0241195163034131
alpha: 2.488903466581201
```

```python
def compare_distributions(data, distribution_1, distribution_2, xmin=1):
    fit = powerlaw.Fit(data, xmin=xmin)
    LTR, significance = fit.distribution_compare(distribution_1,
    distribution_2, normalized_ratio=True)
    if LTR > 0:
        print('{} is a better fit than {}. p-value: {:0.5f}'.format
        (distribution_1, distribution_2, significance))
    else:
        print('{} is a better fit than {}. p-value: {:0.5f}'.format
        (distribution_2, distribution_1, significance))
```

✓ 0.0s Python

```python
# Comparison with few data points

data1 = powerlaw.Power_Law(xmin=1, parameters=[2.5]).generate_random(100)
compare_distributions(data1, 'power_law', 'lognormal_positive')
compare_distributions(data1, 'power_law', 'exponential')
compare_distributions(data1, 'power_law', 'truncated_power_law')
```

✓ 0.1s Python

```
power_law is a better fit than lognormal_positive. p-value: 0.05744
power_law is a better fit than exponential. p-value: 0.00245
truncated_power_law is a better fit than power_law. p-value: 0.47614
Assuming nested distributions
```

```python
# Comparison with few data points, run2

data1 = powerlaw.Power_Law(xmin=1, parameters=[2.5]).generate_random(100)
compare_distributions(data1, 'power_law', 'lognormal_positive')
compare_distributions(data1, 'power_law', 'exponential')
compare_distributions(data1, 'power_law', 'truncated_power_law')
```

✓ 0.1s Python

```
power_law is a better fit than lognormal_positive. p-value: 0.04420
power_law is a better fit than exponential. p-value: 0.04694
power_law is a better fit than truncated_power_law. p-value: 0.99985
Assuming nested distributions
```

THE POWERLAW PACKAGE

- Fitting data
  - Fitting very small N (=100) is unreliable
  - Fitting large N is fine
  - Nested distributions are tricky to disentangle

```
# Comparison with many data points

data2 = powerlaw.Power_Law(xmin=1, parameters=[2.5]).generate_random(100000)
compare_distributions(data2, 'power_law', 'lognormal_positive')
compare_distributions(data2, 'power_law', 'exponential')
compare_distributions(data2, 'power_law', 'truncated_power_law')
```

✓ 22.3s Python

```
power_law is a better fit than lognormal_positive. p-value: 0.00000
power_law is a better fit than exponential. p-value: 0.00000
Assuming nested distributions
truncated_power_law is a better fit than power_law. p-value: 0.52197
```

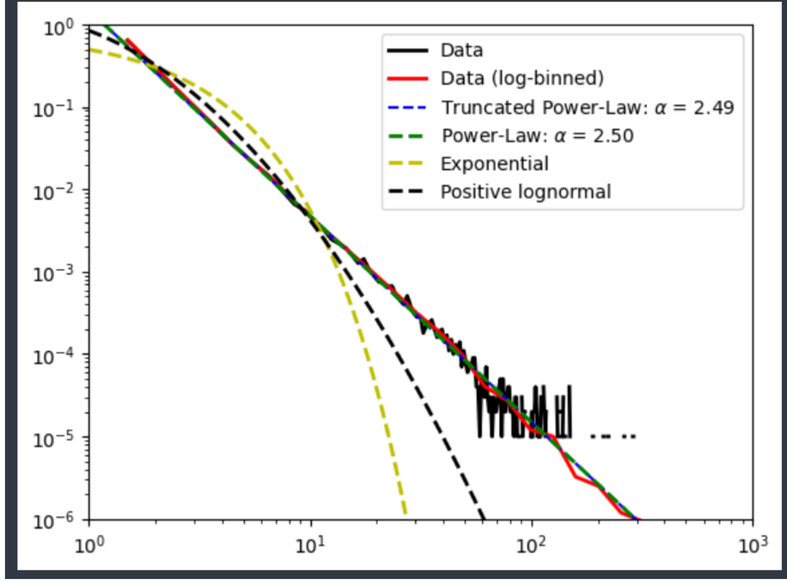
```
# Comparison with many data points, run2

data2 = powerlaw.Power_Law(xmin=1, parameters=[2.5]).generate_random(100000)
compare_distributions(data2, 'power_law', 'lognormal_positive')
compare_distributions(data2, 'power_law', 'exponential')
compare_distributions(data2, 'power_law', 'truncated_power_law')
```

✓ 26.2s Python

```
power_law is a better fit than lognormal_positive. p-value: 0.00000
power_law is a better fit than exponential. p-value: 0.00001
Assuming nested distributions
power_law is a better fit than truncated_power_law. p-value: 0.98462
```

```
data2 = powerlaw.Power_Law(xmin=1, parameters=[2.5]).generate_random(100000)
fit2 = powerlaw.Fit(data2, xmin=1)
fig = fit2.plot_pdf(color='k', linear_bins=True, linewidth=2, **{'label': 'Data'})
fit2.plot_pdf(color='r', linewidth=2, **{'label': 'Data (log-binned)'})
fit2.truncated_power_law.plot_pdf(color='b',linestyle='--', **{'label': r'Truncated
Power-Law: $\alpha$ = {:0.2f}'.format(fit.truncated_power_law.alpha)})
fit2.power_law.plot_pdf(color='g', linewidth=2, linestyle='--', **{'label': r'Power-Law:
$\alpha$ = {:0.2f}'.format(fit.power_law.alpha)})
fit2.exponential.plot_pdf(color='y', linewidth=2, linestyle='--', **{'label': 'Exponential'})
fit2.lognormal_positive.plot_pdf(color='k', linewidth=2, linestyle='--', **{'label':
'Positive lognormal'})
plt.legend()
plt.xlim(1,1e3)
plt.ylim(1e-6,1e0);
```

✓ 30.3s Python



FOR MORE

- Talk
- Talk summary:
  - http://bactra.org/weblog/491.html

So, You Think You Have a Power Law, Do You? Well Isn't That Special?

Cosma Shalizi

Statistics Department, Carnegie Mellon University

Santa Fe Institute

18 October 2010, NY Machine Learning Meetup