Zipf's and Heap's law.

Zipf's law.

Zipf's law is a law about the frequency distribution of words in a language (or in a collection that is large enough so that it is representative of the language). To illustrate Zipf's law let us suppose we have a collection and let there be **V unique words** in the collection (the vocabulary).
For each word in the collection we need to compute the **freq(word)** = how many times word occurs in the collection. Then we rank the words in descending by their frequency (most frequent word has rank 1, next frequent word has rank 2, ...)

The slides provide an example, which we reproduce here:

Word	Freq	r	Pr(%)	r*Pr
the	2,420,778	1	6.488	0.0649
of	1,045,733	2	2.803	0.0561
to	968,882	3	2.597	0.0779
a	892,429	4	2.392	0.0957
and	865,644	5	2.32	0.116
in	847,825	6	2.272	0.1363
said	504,593	7	1.352	0.0947
for	363,865	8	0.975	0.078
that	347,072	9	0.93	0.0837
was	293,027	10	0.785	0.0785
on	291,947	11	0.783	0.0861
he	250,919	12	0.673	0.0807
is	245,843	13	0.659	0.0857
with	223,846	14	0.6	0.084
at	210,064	15	0.563	0.0845
by	209,586	16	0.562	0.0899
it	195,621	17	0.524	0.0891
from	189,451	18	0.508	0.0914
as	181,714	19	0.487	0.0925
be	157,300	20	0.422	0.0843
were	153,913	21	0.413	0.0866
an	152,576	22	0.409	0.09
have	149,749	23	0.401	0.0923
his	142,285	24	0.381	0.0915
but	140,880	25	0.378	0.0944

Word	Freq	r	Pr(%)	r*Pr
has	136,007	26	0.365	0.0948
are	130,322	27	0.349	0.0943
not	127,493	28	0.342	0.0957
who	116,364	29	0.312	0.0904
they	111,024	30	0.298	0.0893
its	111,021	31	0.298	0.0922
had	103,943	32	0.279	0.0892
will	102,949	33	0.276	0.0911
would	99,503	34	0.267	0.0907
about	92,983	35	0.249	0.0872
i	92,005	36	0.247	0.0888
been	88,786	37	0.238	0.0881
this	87,286	38	0.234	0.0889
their	84,638	39	0.227	0.0885
new	83,449	40	0.224	0.0895
or	81,796	41	0.219	0.0899
which	80,385	42	0.215	0.0905
we	80,245	43	0.215	0.0925
more	76,388	44	0.205	0.0901
after	75,165	45	0.201	0.0907
us	72,045	46	0.193	0.0888
percent	71,956	47	0.193	0.0906
up	71,082	48	0.191	0.0915
one	70,266	49	0.188	0.0923
people	68,988	50	0.185	0.0925

Top 50 words from 84,678 Associated Press 1989 articles

Let r be the rank of word, Prob(r) be the probability of a word at rank r. We do not care about the names of the words, we care only about their ranks and frequencies. By definition Prob(r) = freq(r) / N where
freq(r) = the number of times the word at rank r appears in the collection and
N = total number of words in the collection (not number of unique words).

Then Zipf's law states that

r * Prob(r) = A,

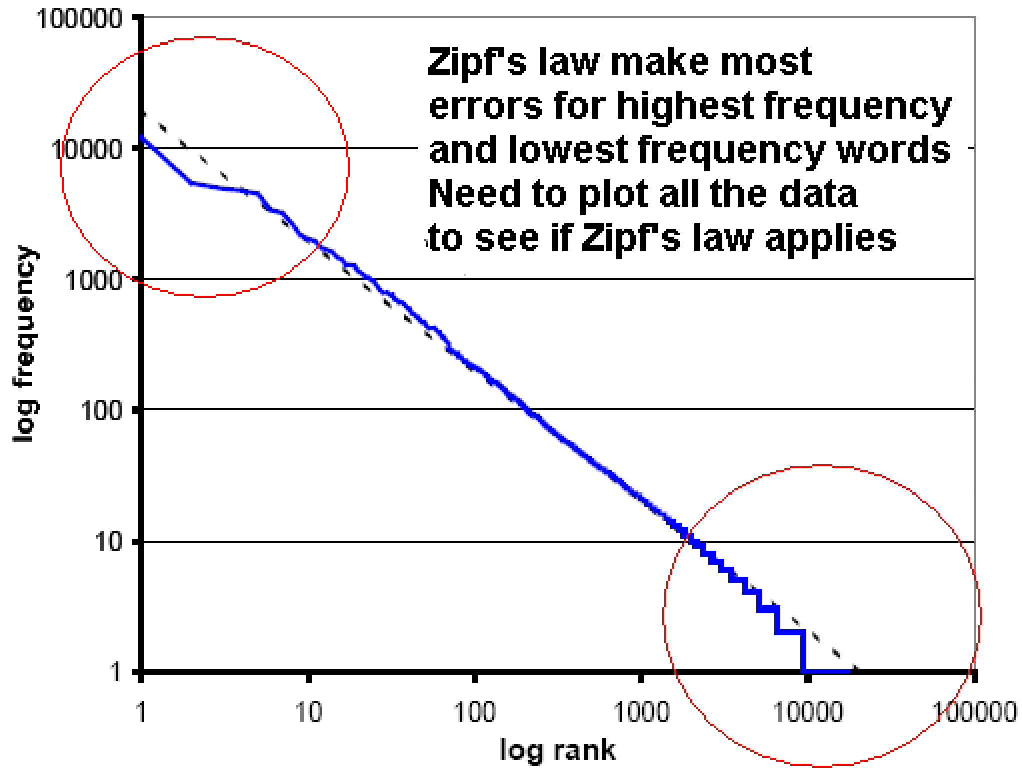
where A is a constant which should empirically be determined from the data. In most cases **A = 0.1**. Zipf's law is not an exact law, but a statistical law and therefore does not hold exactly but only on average (for most words).

Taking into account that Prob(r) = freq(r) / N we can rewrite Zipf's law as

r * freq(r) = A * N

To establish that Zip's law holds we need to compute freq(r), which involves computing the frequency of each word and then ranking the words. Then we need to compute r * freq(r) and see if r * freq(r) is approximately a constant. This does not mean that for all words r * freq(r) has to be exactly the same, but it has to be close to the same number for most words. The simplest way to show that Zipf's law holds is to plot the data. Remember that looking at most frequent and least frequent words only is misleading. For those types of words Zipf's law has the highest errors.

Instead of plotting r vs. freq(r), it is better to **plot log(r) on the x-axis and log(freq(r)) on the y axis**. **If Zipf's law holds we should see a line with slope -1** (this means if A is the point where the line crosses the x-axis and B is the point where the line crosses the y-axis and O is the origin of the coordinate system then OA = OB).
Another, equivalent way is to **plot log(r)** on the x-axis and **log(Prob(r))** on the y axis.



Zipf curve for the unigrams extracted from a
250,000 word tokens corpus

Source: Extension of Zipf's law to words and Phrases by Ha, Garcia, Smith

Notice the slope of the line is -1.

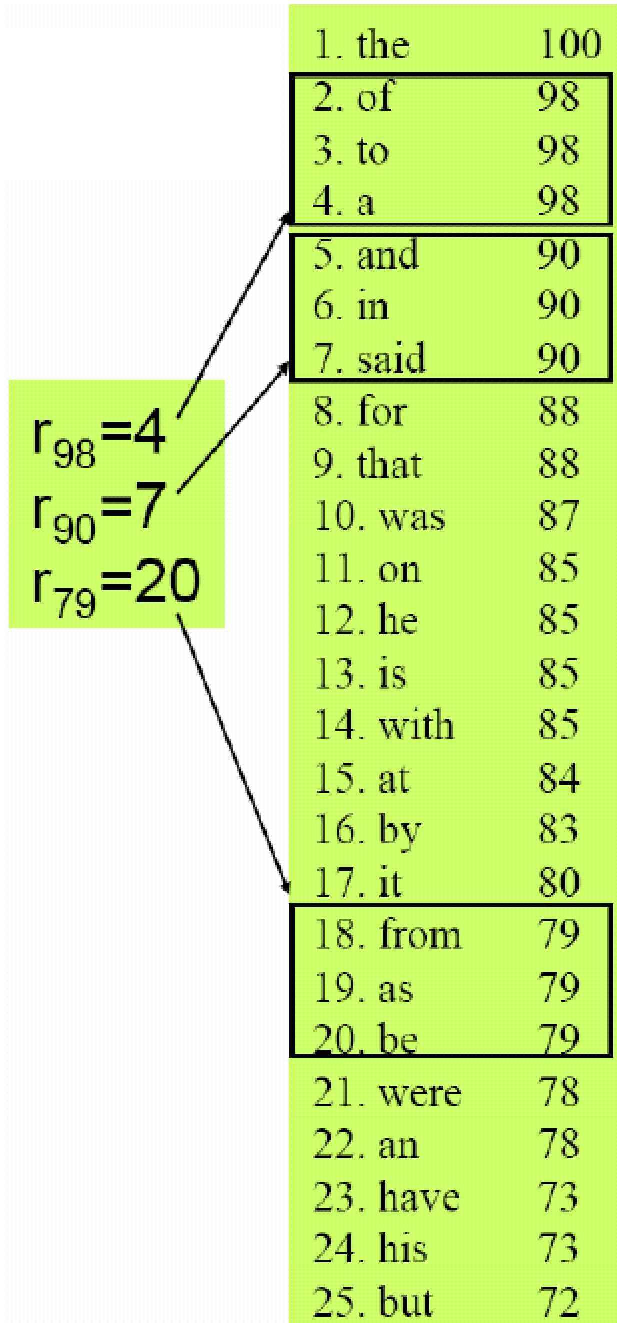
We can use Zipf's law to calculate the number of words that appear n times in the collection.

Let M**axRank(n)** = among all words that appear n times let MaxRank(n) be the maximum of the ranks of those words. For example, if n = 90, the words that appear n = 90 times are “and”, “in”, “said” with ranks 5, 6, 7. Then MaxRank(90) = max(5, 6, 7) = 7

Another example: MaxRank(79) = max(18, 19, 20) = 20

Notice that the number of words that appear n times is
NumberWordsOccur(n) =
MaxRank(n) – MaxRank(n + 1).

For example, the number of words that appear 79 times is MaxRank(79) – MaxRank(80) =
max(18, 19, 20) – max(17) = 20 – 17 = 3
We can look at the picture on the right side to see that exactly 3 words appear 79 times.



We know from Zipf's law that for the frequency and the rank are related.

If r = MaxRank(n), this means that the rank is r and the frequency is n; So r * freq(r) = A * N means

MaxRank(n) * n = A*N, which implies

MaxRank(n) = A * N / n

Applying this formula twice we obtain

NumberWordsOccur(n) =
MaxRank(n) – MaxRank(n + 1) =
A*N / n – A * N / (n + 1) =
A* N (1/n – 1/(n + 1) = A * N / [n * (n + 1)]
So,
NumberWordsOccur(n) = A * N / [n * (n + 1)]
is the number of words that occur n times.

We need to connect A * N to the number of unique words in the collection. This is easy because V, which is the number of unique words is simply the rank of the last word in the ranked list of words. We need to assume(quite reasonably) that the least occurring word occurs

only once.
So, Zipf's law applied to the least frequent word gives (here: r = V, and freq(r) = 1)
r * freq(r) = A * N, V * 1 = A * N,
A * N = V

Therefore
NumberWordsOccur(n) = V / [n * (n + 1)]
is the number of words that occur n times.
where V is the number of unique words in the collection

Application of the formula **NumberWordsOccur(n) = V / [n * (n + 1)].**
What fraction of all unique words appear only once?

We need “number of words that occur once” / “number of unique words” =
= NumberWordsOccur(1) / V = [V / [1 * (1 + 1)]] / V = 1 / [1 * (1 + 1)] = ½

Heap's law.

Heap's law states that the number of unique words V in a collection with N words is approximately **Sqrt[N].** The more general form of this law is

$$V = KN^{\beta} \quad (0 < \beta < 1)$$

Typically

– $K \approx$ 10–100

– $\beta \approx$ 0.4–0.6 (approx. square-root of n)

Alpha and beta and usually found by fitting the data.

Power laws

Zipf's and Heap's law belong to a class of laws called power laws.
A power law is one that has the form

$$y = k * x^{c}$$

k and c are constants that have to be fit from the data.
If we are to write Zipf's law as power low, we notice that
y = freq(r), x = r, k = A * N, c = -1

If we are to write Heap's law as a power law we observe that y = V, x = N, k = K (from Heap's law), c = beta

Power laws have the useful property that if one takes the log of both sides of
one obtains a line. See picture.

$$y = kx^c$$

start here

$$\log(y) = \log(kx^c)$$

take the log of both sides

$$\log(y) = \log(k) + c\log(x)$$

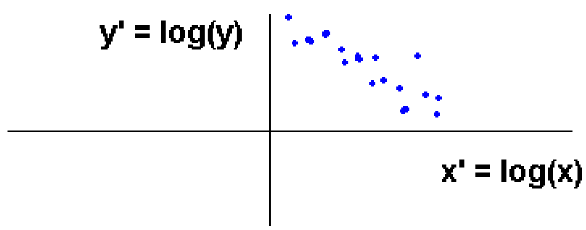
simplify the right side

Let y' = log(y)
x' = log(x)

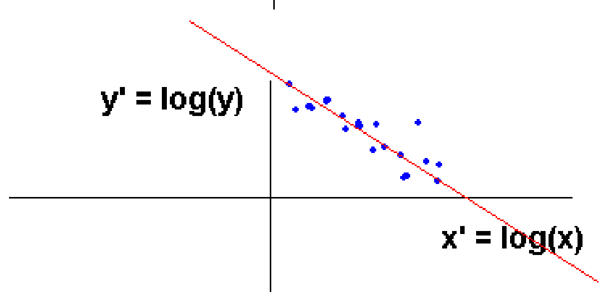
Transform the data by taking logs of the data

y' = m + c * x', where m = log(k)

This is a line



plot the transformed data



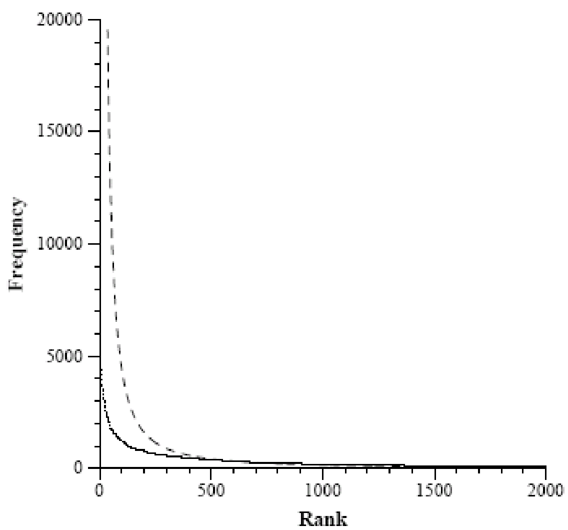
fit a line and find c and m

Use that m = log(k) to find that

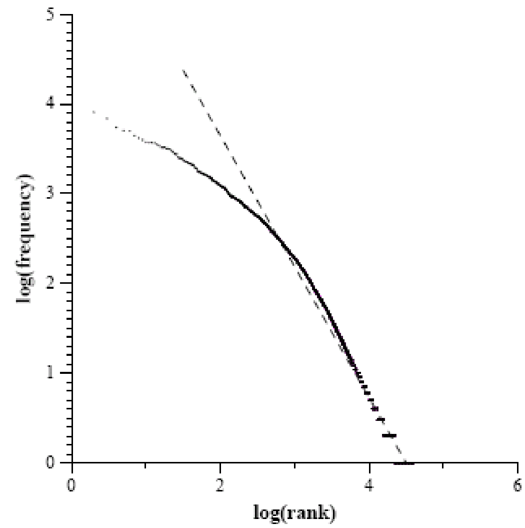
k = exp(m) **[watch the base of your log here]**

Note: when you do the line fitting, do not use x and y but use x' = log(x) and y' = log(y) in the formulas for the least squares line fitting.

Fitting Zipf's law:



Plot of term frequency vs. rank.

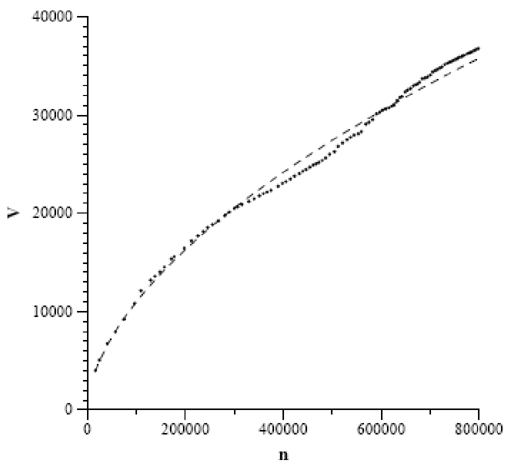


Log-log plot of term frequency vs. rank.

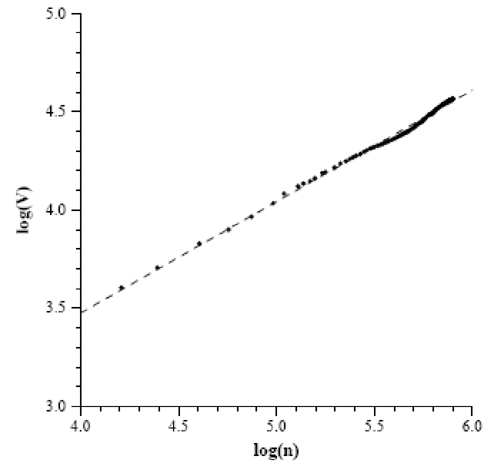
Source: Modeling Web Data by French

If you fit data that obeys Zipf's law you should get c close to -1.

Fitting Heap's law:



Plot of unique vocabulary terms vs. total terms.



Log-log plot of unique vocabulary terms vs. total terms.

n: number of total words encountered
V = V(n) = number of unique words encountered, is a function of n

(Source: Modeling Web Data by French)

If you fit data that obeys Heap's you should get c = slope of the line close to 0.5