Models of Natural Language

In modeling English (or another natural language) we ignore *meaning* and even grammar. That is, we ignore **semantics**.

Instead, we treat English as being the result of some random process.

The worst possible model would be to suppose that each letter `a-z` in a message was chosen at random with **equal probabilities**. If this were so, cryptanalytic attacks would be nearly impossible. Luckily, this is not a good model.

Instead, the next-to-dumbest model observes that the **single-character** (and other) stats in English and other natural languages are *not flat*, but are *skewed* in a way that is useful.

In English, ignoring spaces (about 17% otherwise) the frequencies are roughly

`e`	occurs about	11 %	of the time
`t`	occurs about	9 %	of the time
`o`	occurs about	8 %	of the time
`i`	occurs about	8 %	of the time
`a`	occurs about	8 %	of the time
`n`	occurs about	7 %	of the time
`s`	occurs about	7 %	of the time
	...		
`v`	occurs about	1 %	of the time
`k`	occurs about	1 %	of the time
`x`	occurs about	0.3 %	of the time
`j`	occurs about	0.23 %	of the time
`q`	occurs about	0.12 %	of the time
`z`	occurs about	0.09 %	of the time

Other natural languages using single-letter alphabets have similar skewings away from a *flat* distribution.

Digrams are *adjacent pairs* of characters. There are $26 \times 26 = 676$ different digrams. Every letter of the alphabet really does occur in English, but some digrams never occur.

In a sample of 500k of filtered email *with spaces removed*, 659 of the possible 676 digrams occur.

Top 44 digrams give more than 50% of the total.

The top 54 give 50%, the top 126 give 75%, the top 222 give 90%, and the top 359 give 98%.

th 3.18	*in* 2.59	*he* 2.17	*er* 1.95
re 1.85	*on* 1.63	*an* 1.59	*at* 1.54
ou 1.43	*or* 1.26	*es* 1.26	*ha* 1.24
to 1.22	*te* 1.21	*is* 1.18	*ti* 1.17
it 1.16	*en* 1.13	*nt* 1.09	*ng* 1.08
al 1.07	*se* 1.05	*st* 1.01	*nd* 0.98
le 0.91	*ar* 0.90	*me* 0.90	*hi* 0.86
ve 0.85	*of* 0.84	*ed* 0.78	*co* 0.74
as 0.73	*ll* 0.72	*ne* 0.70	*om* 0.70
ri 0.68	*ic* 0.67	*ro* 0.67	*ea* 0.66
et 0.64	*ur* 0.64	*io* 0.64	*ra* 0.62
li 0.62	*no* 0.62	*so* 0.62	*be* 0.61
de 0.59	*ma* 0.59	*si* 0.58	*ly* 0.54
ut 0.53	*ot* 0.53	*pr* 0.53	*fo* 0.53
yo 0.52	*il* 0.50	*ca* 0.50	*pe* 0.50
ch 0.49	*ho* 0.49	*ul* 0.47	*ce* 0.47
ta 0.45	*di* 0.45	*rs* 0.45	*el* 0.44
ge 0.44	*us* 0.44	*ec* 0.42	*ss* 0.42
ac 0.41	*ct* 0.41	*em* 0.41	*wh* 0.40
oo 0.40			

versus $1/656 \sim .15\%$

Trigrams are *adjacent triples* of characters. Of $26 \times 26 \times 26 = 17,576$ possible few occur often. The top 240 give 50% of all trigrams occurring: 1/70 of all trigrams account for 50% of occurrences.

When blanks are removed frequencies are spread out (as with digrams). The top 430 give 50%, the top 1160 give 75%, the top 2300 give 90%, and the top 4400 give 98%.

the 2.44	*ing* 1.26	*and* 0.82	*hat* 0.78
tha 0.77	*ion* 0.75	*you* 0.67	*ent* 0.66
for 0.63	*tio* 0.63	*thi* 0.60	*her* 0.51
ati 0.47	*our* 0.47	*ere* 0.45	*all* 0.43
ter 0.43	*ver* 0.40	*not* 0.40	*hin* 0.40
ome 0.36	*oul* 0.36	*uld* 0.36	*int* 0.34
rea 0.34	*pro* 0.34	*res* 0.33	*ate* 0.33
hav 0.30	*ave* 0.30	*ill* 0.30	*his* 0.30
com 0.30	*ons* 0.30	*are* 0.28	*ple* 0.28
ers 0.28	*con* 0.27	*ess* 0.27	*out* 0.27
one 0.26	*ith* 0.25	*som* 0.25	*ive* 0.25
tin 0.25	*nce* 0.24	*ble* 0.24	*ted* 0.24
han 0.23	*ine* 0.23	*per* 0.23	*ect* 0.23
nte 0.23	*wit* 0.22	*men* 0.22	*but* 0.22
wou 0.21	*ica* 0.21	*eve* 0.21	*cal* 0.21
pre 0.21	*cou* 0.21	*lin* 0.21	*est* 0.20
eri 0.20	*mor* 0.20	*ser* 0.20	*ore* 0.19
any 0.19	*abl* 0.19	*tic* 0.19	*urs* 0.19
ant 0.19	*sti* 0.18	*ear* 0.18	*hou* 0.18
ies 0.18			

versus $1/26^3 \sim 0.00569\%$

There are few **small words** in English, so if word breaks are preserved cryptanalysis is vastly easier.

Only 3 single-letter words, `I` and `a`

Only about 35 two-letter words
(versus $26^2 = 676$)

Only about 200 common three-letter words
(versus $26^3 = 17576$)

Common words are indeed very common. In 500k of my email, more than 5000 words appear, but the 9 most common words are 21% of all occurrences, the 20 most common give 30%, the 104 most common give 50%.

the 4.65	*to* 3.02	*of* 2.61
i 2.2	*a* 1.95	*and* 1.82
is 1.68	*that* 1.62	*in* 1.57
it 1.22	*for* 1.17	*you* 1.06
be 0.99	*not* 0.84	*on* 0.76
have 0.71	*this* 0.69	*as* 0.57
at 0.56	*would* 0.55	*are* 0.55
but 0.54	*if* 0.53	*my* 0.53
with 0.5	*your* 0.48	*so* 0.48
or 0.46	*some* 0.43	*will* 0.41
do 0.39	*about* 0.39	*me* 0.38
from 0.35	*by* 0.33	*no* 0.33
more 0.33	*what* 0.32	*an* 0.32
there 0.32	*one* 0.32	*all* 0.32
was 0.30	*we* 0.30	*just* 0.27
which 0.27	*can* 0.26	*very* 0.25
series 0.25	*am* 0.24	*things* 0.24
people 0.24	*get* 0.23	*hi* 0.23
time 0.22	*think* 0.22	*course* 0.22
etc 0.22	*also* 0.21	*any* 0.21

These low-level statistical biases can be put to use in cryptanalysis. On the other hand, they must be masked to have a secure cryptosystem.

A related point is that *non-random data can be* **compressed**, which among its other effects *flattens* the stats of the code alphabet.

Just as it is good to remove spaces before encryption, it might be good to *compress* before encryption.

But, actually, good ciphers are so good that this does not really matter.

And, compressed data has highly structured, non-random **headers**, which partly defeat the purpose of flattening the stats.

Cryptograms

Cryptograms are also **monoalphabetic substitution ciphers**.

Substitution cipher means that each letter of the alphabet `a,b,c,...,x,y,z` is encrypted as some *other* letter by a secret chosen bijective function from the alphabet to itself.

Monoalphabetic means that the encryption of each letter is the same throughout the encryption of a message (rather than possibly varying over the message).

With spaces left in and with hints, they are in newspapers as puzzles. They are not secure.

Shift ciphers and affine ciphers are examples of monoalphabetic substitution ciphers in which the encryption function is particularly simple.

The typical non-machine setup
for a cryptogram is to have a key
which is a secret phrase, such as
`My dog has fleas`
Then remove duplcate letters, spaces, and
punctuation
`mydoghasfle`
Then put unused letters of the
alphabet on the end: append
`bcijknpqrtuvwxz`
to get
`mydoghasflebcijknpqrtuvwxz`
Then make the lookup table

```
abcdefghijklmnopqrstuvwxyz
mydoghasflebcijknpqrtuvwxz
```

where the character in the top line is
encrypted as the character under it in the
lower line.