Amber Strange

Markov Analysis

Computer Science 201

Brute Markov Hypothesis: time it would take to generate T characters on a N sized text would be O(NT) and independent of K.

When N is held constant, and Brute Markov generates different T sized characters, the time increases linearly with an increase in T. I averaged the times for the different order sizes and graphed them against the time. When T doubles, time doubles approximately.

T	Avg Time
100	0.03421
200	0.059579
400	0.125813
800	0.246143
1600	0.521556

When T is held constant at 163187, the size of the text N, does linearly affect the time to generate T words.

Size N	Avg Time
13107	0.011997
82140	0.054802
153088	0.129133
163189	0.105560
496769	0.338196
4345018	3.192867

When size increases by a factor of approximately 10 (13107 to 153088) time increases by about 10 also.

In brute markov, time is affected both by the size of the text and also the number of words being generated meaning that the runtime is O(NT). This makes sense because for every word generated which would take O(T) time you have to search through the entire text, making it dependent upon N as well. This is usually independent of k, when k changes, the times do not change , with N and T held constant. When k = 1, however, the times are almost double. So the runtime is independent of k, unless k =1.

k	Time (when generating 200 words)	Time(when generating 400 words)
1	0.108348	0.221273
5	0.053990	0.109727
10	0.050904	0.105739

Efficient Markov Hypothesis: Time to generate T characters, regardless of size of training text N is O(T) unless training time is included, then it is O(N+T). This is all independent of k.

Not including the training time, by starting the timer after setTraining has been called,

T words	Average time
100	0.00046
200	0.000111
400	0.000138
800	0.000201
1600	0.000374

The time increases linearly with the number of words generated, approximately doubling every time the number of words being generated doubles making it run O(T).

Size N	Avg Time
13107	0.000093
82140	0.000106
153088	0.000119
163187	0.000120
496769	0.000157
4345018	0.000209

Here, there are 400 words being generated, and the size of the text is being manipulated. According to the hypothesis, the time after training should be independent of the size of the text. However, this data shows that the time increases when the size of the text increases. However, it does so very slightly and it is unclear whether it is statistically significant. These are also independent of k, the times do not change when k changes and the size of the text and number of words generated stays the same.

I changed the varying order text to Hawthorne to show that k doesn't impact the time getrandom text even when the text increases. However, this does seem to show that when text size increases so does the time to generate words

k	Time (when generating 200 words) from text of length 163187 Alice.txt	Time(when generating 400 words)
1	0.000074	0.000106
5	0.000077	0.000120
10	0.000065	0.000105
k	Time (when generating 200 words) from text of length 496769 Hawthorne.txt	Time(when generating 400 words)
1	0.000099	0.00014
5	0.000095	0.000178
10	0.000085	0.00014

When training time is included, however, it seems that order does have a large impact on the runtime of efficient markov. In this chart of Alice.txt the size of the text and the number of words being generated (400) is constant and the only thing changing is k.

k	Time
1	0.009082
2	0.012202
3	0.017076
4	0.020596
5	0.025373
6	0.027614
7	0.026198
8	0.034625
9	0.034234
10	0.035846
11	0.033422
12	0.035420
13	0.033585
14	0.034123
15	0.036133

This shows that as k increases so does the time to generate words. I believe this is because when generating text you have to use the method shiftAdd for the wordgram. My shiftadd makes an arraylist of length k, and shifts all of the values and adds the next string to the end and passes it to a new wordgram. When k increases more values have to shift in the arraylist and this increases the time. This happens for every new word generated, so it makes sense that as k increases so does the time. Because the time differences to generate T words are so small, the only noticeable trend is that time is proportional to the size of the text when k is the same, which would mean this is O(NK + T).

Size N	Avg Time
13107	0.001810
82140	0.012621
153088	0.023775
163189	0.023523
496769	0.100444
4345018	1.855733

Map Hypothesis:

The time it takes to insert U keys into a hashmap is O(U) and the time it takes to insert U keys into a TreeMap is O(U(logU)). However my running of the benchmark, only including the training time, shows the times to be nearly the same for hashmap and treemap, and not dependent on the number of unique keys in the map. I changed the order of the part of the benchmark that compared times of texts of different sizes. It showed that no matter the number of unique keys, the times were approximately the same between hash and treemaps. The times did increase with order, but that was the same with hashmaps in which the number of unique keys should not affect the time to insert. This may be because the number of wordgrams being created is almost the same, but with larger orders it takes longer to

create a wordgram because of how the instance variable mywords is created by copying values into an array, and with higher orders, more values are being added to the array. When the number of unique keys is constant, when run on the same text, the time increases linearly with the order.

K=5

unique	Text size	treemap time	Hashmap time
18122	13107	0.001363	0.00198
31012	82140	0.009923	0.010195
42634	153088	0.022084	0.021878
42016	163189	0.021**223**	0.02154
82337	496769	0.089363	0.089687
180176	4345018	1.922802	1.93467

K=1

unique	Text size	treemap time	Hashmap time
55	13107	0.001008	0.001351
66	82140	0.004495	0.00454
69	153088	0.009459	0.008164
89	163189	0.012412	0.008913
77	496769	0.033381	0.028783
89	4345018	1.00888	1.263027

This shows that the times between treemaps and hashmaps although changing with order, are much more affected by the size of the text. It seems as if both TreeMap and HashMap run O(N) and when considering creating the keys to insert, also is proportional to k.