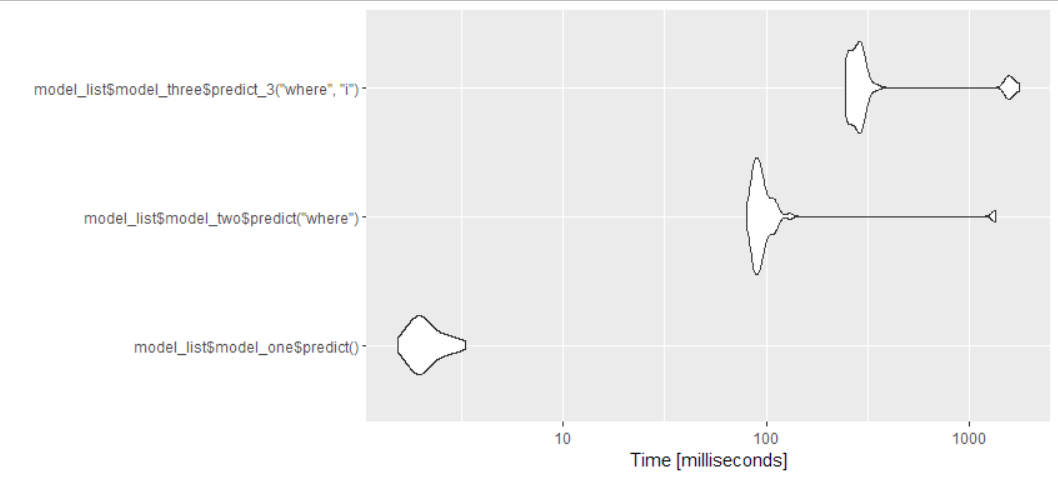
Benchmark test for

Threshold = 0, keep = 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_one$predict() | 1.55 | 1.84 | 2.14 | 2.02 | 2.37 | 3.29 | 50.00 |
| model\_two$predict("where") | 80.45 | 87.62 | 143.78 | 92.75 | 98.92 | 1339.77 | 50.00 |
| model\_three$predict\_3("where",i) | 244.69 | 263.45 | 461.75 | 285.90 | 301.99 | 1757.61 | 50.00 |
| Unit: milliseconds | |  |  |  |  |  |  |



1] "VALIDATION RESULTS:"

[1] "Number of tri-grams tested: 2000 out of 4376983"

[1] "Cumulative count of tri-grams tested: 42816 out of 6352027"

[1] "Word 1 Model errors for first guess: 1945 ( 97 %) and 37953 ( 89 %)"

[1] "Word 2 Model errors for first guess: 1794 ( 90 %) and 23876 ( 56 %)"

[1] "Word 3 Model errors for first guess: 1572 ( 79 %) and 15455 ( 36 %)"

[1] "Word 1 Model errors for any guess: 1793 ( 90 %) and 27365 ( 64 %)"

[1] "Word 2 Model errors for any guess: 1536 ( 77 %) and 12422 ( 29 %)"

[1] "Word 3 Model errors for any guess: 1276 ( 64 %) and 6270 ( 15 %)"

[1] "VALIDATION RESULTS:"

[1] "Number of tri-grams tested: 20000 out of 4376983"

[1] "Cumulative count of tri-grams tested: 266304 out of 6352027"

[1] "Word 1 Model errors for first guess: 19360 ( 97 %) and weighted 240544 ( 90 %)"

[1] "Word 2 Model errors for first guess: 17782 ( 89 %) and weighted 174673 ( 66 %)"

[1] "Word 3 Model errors for first guess: 15974 ( 80 %) and weighted 134328 ( 50 %)"

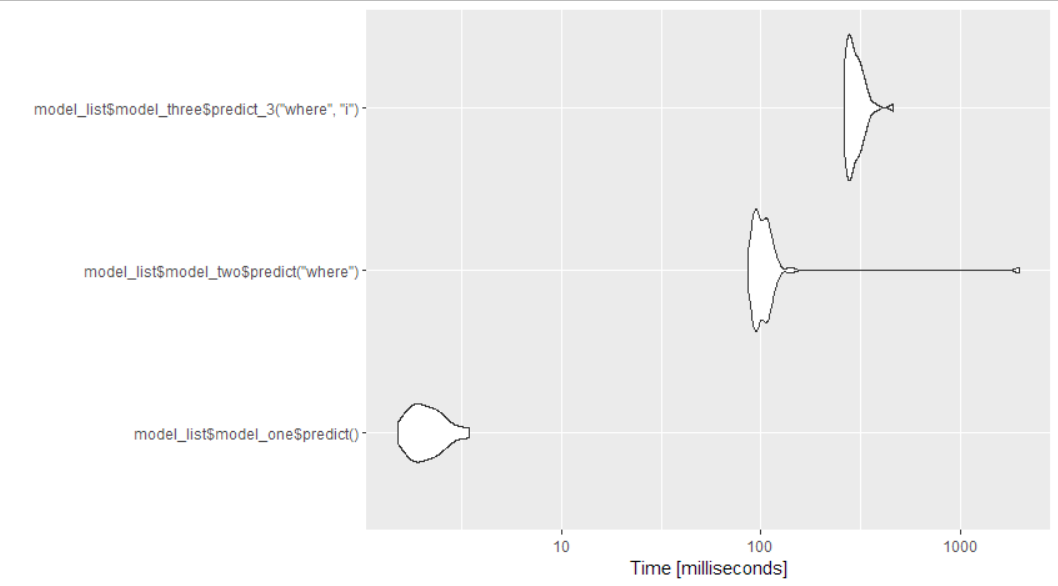
[1] "Word 1 Model errors for any guess: 17736 ( 89 %) and weighted 186560 ( 70 %)"

[1] "Word 2 Model errors for any guess: 15230 ( 76 %) and weighted 108651 ( 41 %)"

[1] "Word 3 Model errors for any guess: 12863 ( 64 %) and weighted 62958 ( 24 %)"

After changing 3 word filter to only one call:

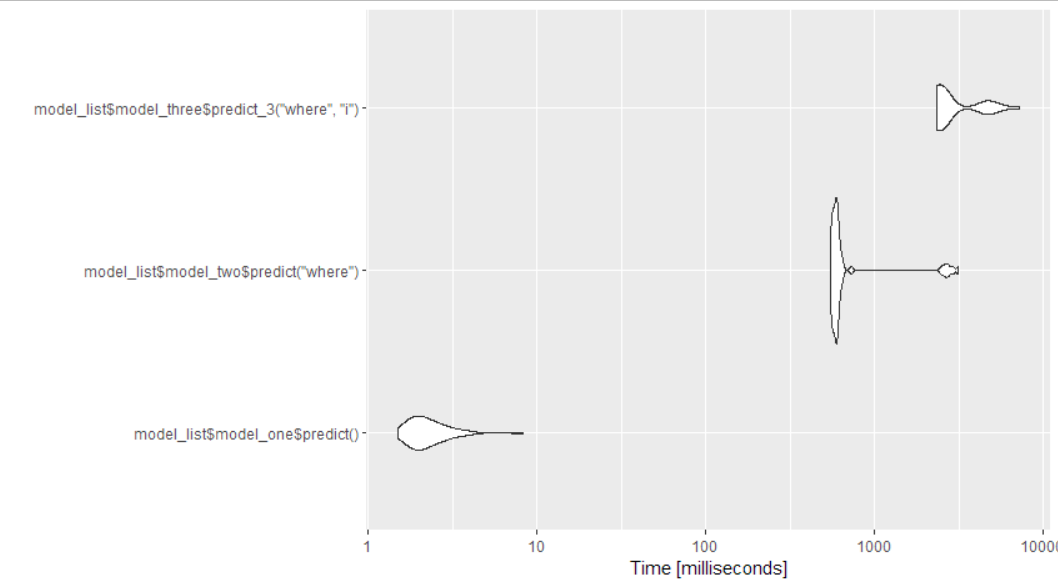
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_list$model\_one$predict() | 1.5 | 1.8 | 2.2 | 2.1 | 2.4 | 3.5 | 50.0 |
| model\_list$model\_two$predict("where") | 86.7 | 94.0 | 139.5 | 101.6 | 108.6 | 1981.0 | 50.0 |
| model\_list$model\_three$predict\_3("where", i) | 260.8 | 273.8 | 297.6 | 288.8 | 311.2 | 461.7 | 50.0 |
| Unit: milliseconds |  |  |  |  |  |  |  |



This change dramatically improved the variance and thus decreased the mean time. However, the median value has changed little.

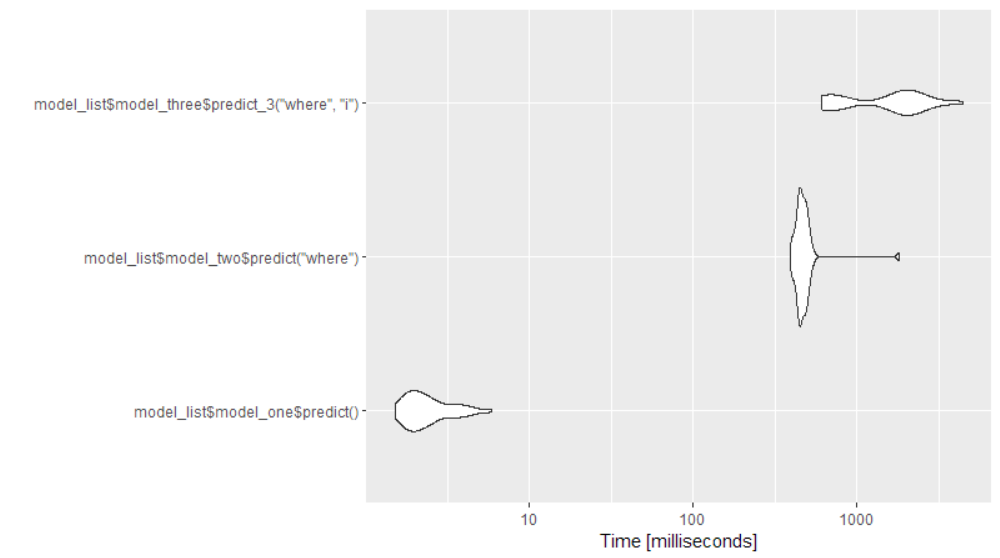
After changing the filter to a data.table version with setkeys just for the 1 word prediction. This was significantly worse than using the dataframe filter function.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_list$model\_one$predict() | 1.5 | 1.8 | 2.4 | 2.1 | 2.7 | 8.3 | 50.0 |
| model\_list$model\_two$predict("where") | 545.1 | 565.3 | 848.3 | 592.8 | 609.7 | 3126.3 | 50.0 |
| model\_list$model\_three$predict\_3("where",i) | 2331.5 | 2412.4 | 3104.4 | 2452.6 | 4439.1 | 7221.6 | 50 |



Using the as.data.table method and basic data.table filtering

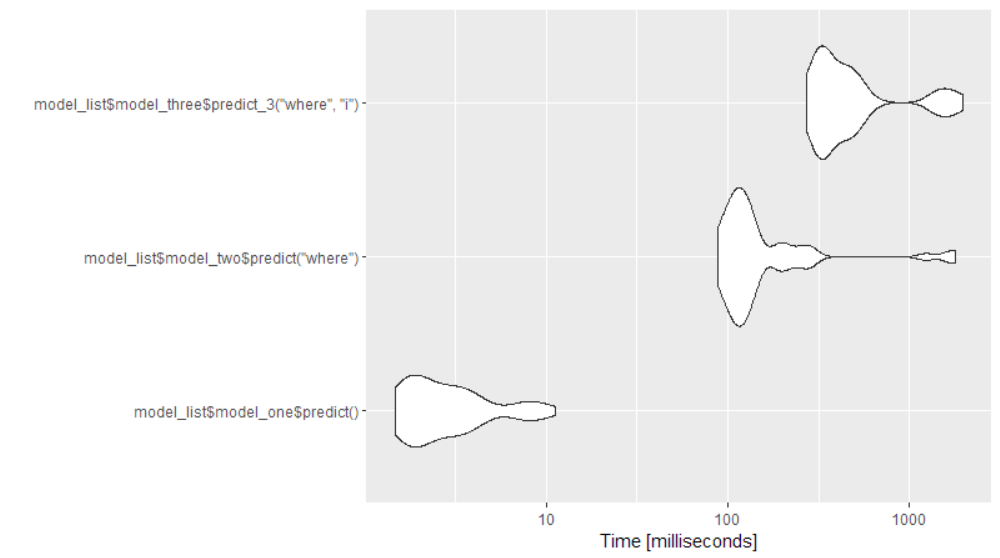
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_list$model\_one$predict() | 1.5 | 1.8 | 2.5 | 2.2 | 2.8 | 5.9 | 50.0 |
| model\_list$model\_two$predict("where") | 396.1 | 443.5 | 490.2 | 460.4 | 488.0 | 1810.6 | 50.0 |
| model\_list$model\_three$predict\_3("where",i) | 605.0 | 723.8 | 1693.5 | 1908.5 | 2169.2 | 4486.3 | 50 |



This is also significant slower than using the filter() function. We will keep using that in the predictions.

Rerunning it using the filter function to double check the benchmarking

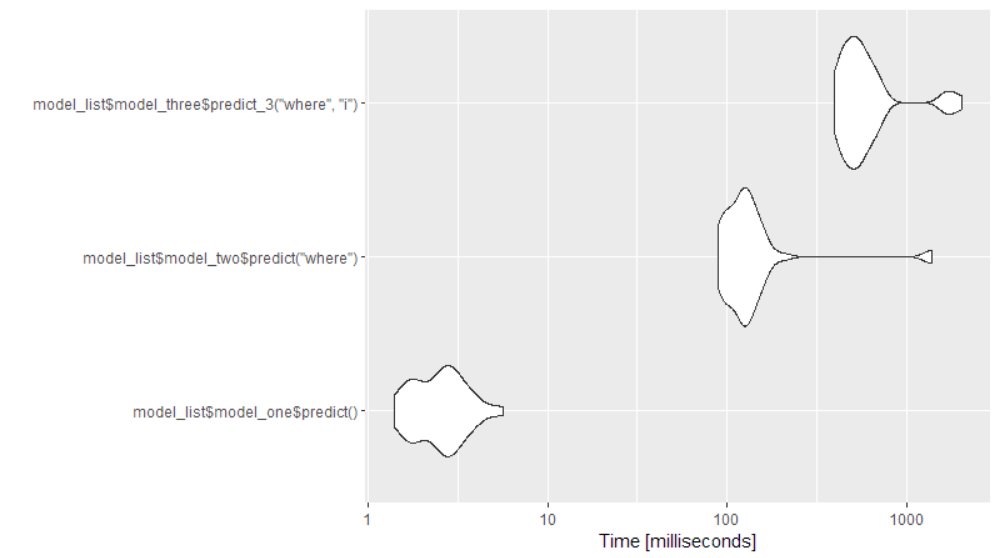
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_list$model\_one$predict() | 1.5 | 1.8 | 3.3 | 2.4 | 3.7 | 11.2 | 50.0 |
| model\_list$model\_two$predict("where") | 87.7 | 105.7 | 222.8 | 123.0 | 148.2 | 1793.9 | 50.0 |
| model\_list$model\_three$predict\_3("where",i) | 270.7 | 319.5 | 583.0 | 383.5 | 508.0 | 1979.2 | 50 |



This is again the fastest option.

With the fixed version of filter for sorting on two columns. The previous one didn’t work.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_list$model\_one$predict() | 1.4 | 1.9 | 2.6 | 2.6 | 3.0 | 5.7 | 50.0 |
| model\_list$model\_two$predict("where") | 88.8 | 100.0 | 173.0 | 124.4 | 142.3 | 1384.4 | 50.0 |
| model\_list$model\_three$predict\_3("where",i) | 394.9 | 456.7 | 655.1 | 532.8 | 622.0 | 2038.0 | 50 |



The speed isn’t as good but it works now

Validation results with this change to the filter:

[1] "VALIDATION RESULTS:"

[1] "Number of tri-grams tested: 3000 out of 4376983"

[1] "Cumulative count of tri-grams tested: 61140 out of 6352027"

[1] "Word 1 Model errors for first guess: 2915 ( 97 %) and weighted 53416 ( 87 %)"

[1] "Word 2 Model errors for first guess: 2696 ( 90 %) and weighted 34660 ( 57 %)"

[1] "Word 3 Model errors for first guess: 2383 ( 79 %) and weighted 23629 ( 39 %)"

[1] "Word 1 Model errors for any guess: 2691 ( 90 %) and weighted 39147 ( 64 %)"

[1] "Word 2 Model errors for any guess: 2301 ( 77 %) and weighted 19494 ( 32 %)"

[1] "Word 3 Model errors for any guess: 1936 ( 65 %) and weighted 10460 ( 17 %)"

It’s not much better.

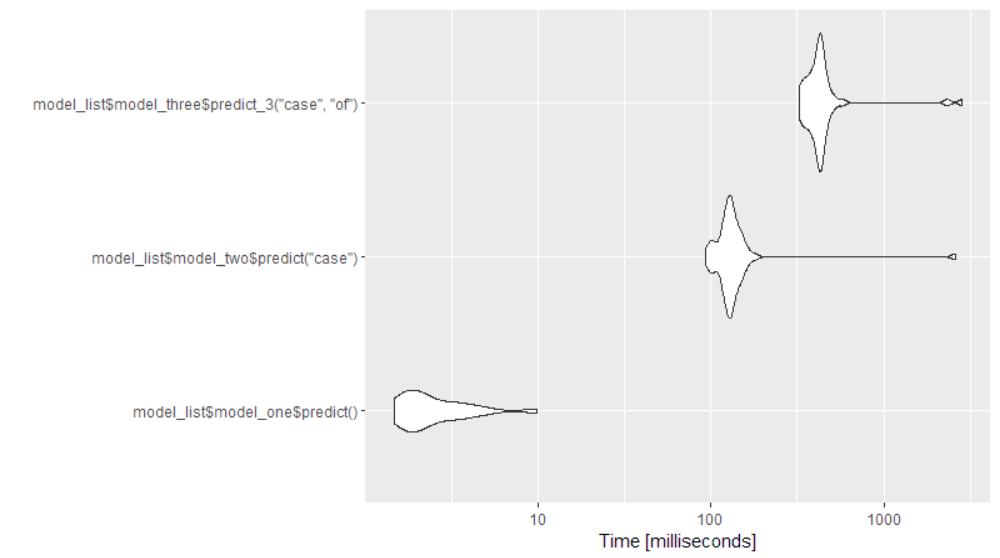
Now try a different approach to the predictions. Previously, the best results for using 2, 1, and no words were added to the list and those with the highest probability were selected. However, this is the probability selected only based on probability within each option. E.g. prob2i= count\_option\_2i/sum\_all\_options\_2, prob1i= count\_option\_1i/sum\_all\_options\_1, prob0i=count\_option0i/sum\_all\_options\_0. That means that if there are a lot of options in the list for using 2 prior words, the likelihood of each can be small and cause us to prematurely resort to using only 1 prior word if there are few options in that case.

1. Is there a more correct calculation of probability that takes into account the greater knowledge contained in using two prior words? Or
2. Resort to using the 1 prior word version only if there aren’t enough options in the 2 prior words list to fill out the desired number of choices. Likewise, only resort to the using no prior words if the list using 2 and 1 prior words is still short of the desired recommendation length.

Option 2:

Benchmark

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **expr** | **min** | **lq** | **mean** | **median** | **uq** | **max** | **neval** |
| model\_list$model\_one$predict() | 1.5 | 1.8 | 2.7 | 2.2 | 3.2 | 9.9 | 50.0 |
| model\_list$model\_two$predict("case") | 92.8 | 118.5 | 177.9 | 129.1 | 140.2 | 2584.5 | 50.0 |
| model\_list$model\_three$predict\_3("case",of) | 323.0 | 381.9 | 500.4 | 423.3 | 443.9 | 2843.5 | 50 |



The timing isn’t much different in this case than with the first when using the filter() function.

[1] "VALIDATION RESULTS:"

[1] "Number of tri-grams tested: 5000 out of 4376983"

[1] "Cumulative count of tri-grams tested: 92774 out of 6352027"

[1] "Word 1 Model errors for first guess: 4834 ( 97 %) and weighted 80316 ( 87 %)"

[1] "Word 2 Model errors for first guess: 4476 ( 90 %) and weighted 54984 ( 59 %)"

[1] "Word 3 Model errors for first guess: 3942 ( 79 %) and weighted 38161 ( 41 %)"

[1] "Word 1 Model errors for any guess: 4448 ( 89 %) and weighted 60863 ( 66 %)"

[1] "Word 2 Model errors for any guess: 3811 ( 76 %) and weighted 32126 ( 35 %)"

[1] "Word 3 Model errors for any guess: 3127 ( 63 %) and weighted 13546 ( 15 %)"

This version is slightly more accurate than before. It’s disappointing that it isn’t better.

How well can it answer the quiz?

**Quiz 2: Natural language processing I**

**Total points**10

**1.**

**Question 1**

For each of the sentence fragments below use your natural language processing algorithm to predict the next word in the sentence.

The guy in front of me just bought a pound of bacon, a bouquet, and a case of

**1 point**



beer



soda



cheese



pretzels

**2.**

**Question 2**

You're the reason why I smile everyday. Can you follow me please? It would mean the

**1 point**



world



most



universe



best

**3.**

**Question 3**

Hey sunshine, can you follow me and make me the

**1 point**



bluest



saddest



smelliest



happiest

**4.**

**Question 4**

Very early observations on the Bills game: Offense still struggling but the

**1 point**



players



defense



crowd



referees

**5.**

**Question 5**

Go on a romantic date at the

**1 point**



movies



beach



grocery



mall

**6.**

**Question 6**

Well I'm pretty sure my granny has some old bagpipes in her garage I'll dust them off and be on my

**1 point**



way



motorcycle



horse



phone

**7.**

**Question 7**

Ohhhhh #PointBreak is on tomorrow. Love that film and haven't seen it in quite some

**1 point**



thing



time



years



weeks

**8.**

**Question 8**

After the ice bucket challenge Louis will push his long wet hair out of his eyes with his little

**1 point**



fingers



ears



eyes



toes

**9.**

**Question 9**

Be grateful for the good times and keep the faith during the

**1 point**



hard



sad



worse



bad

**10.**

**Question 10**

If this isn't the cutest thing you've ever seen, then you must be

**1 point**



callous



asleep



insensitive



insane

This model is able to answer only 4 of 10. Not good enough at all.

Next Option:

Increase our model to use 4\_grams and 5-grams.

# Adding in 5\_ and 4\_grams

Unit: milliseconds

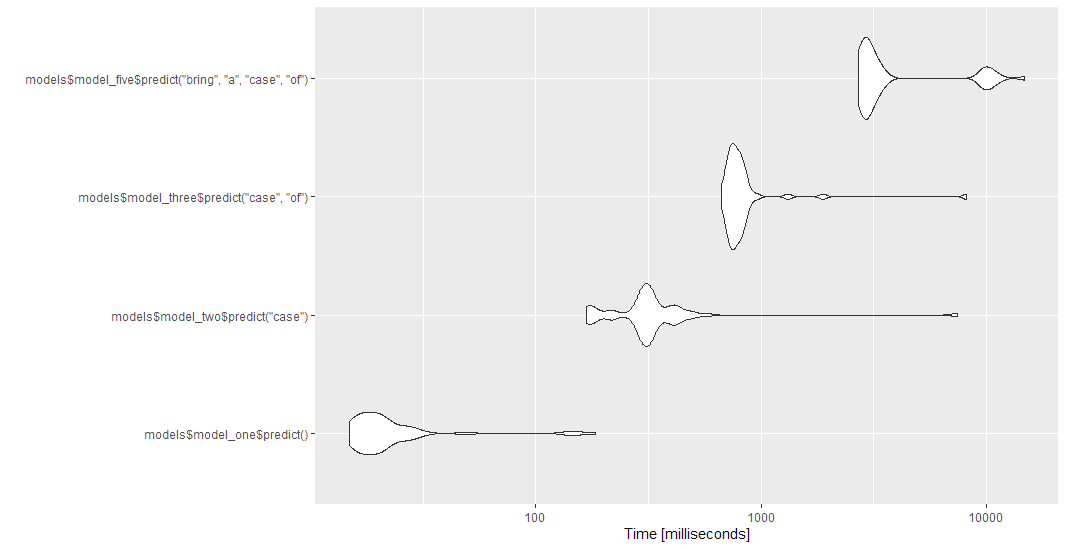
expr min lq mean median uq max neval

models$model\_one$predict() 14.8651 16.8092 28.78441 19.6702 22.9996 183.5841 50

models$model\_two$predict("case") 167.2341 286.0071 456.69339 311.6522 347.8466 7461.8574 50

models$model\_three$predict("case", "of") 667.7548 732.5713 956.08449 767.3017 827.7280 8188.7616 50

models$model\_five$predict("bring", "a", "case", "of") 2706.8631 2871.5595 4677.89278 3020.1382 3470.6897 14818.3217 50



For some reason, the two and three models are also taking longer. This is likely because I fixed a couple of bugs. I also fixed a problem in the base tokenizing that may mean there are extra steps in the model that aren’t needed now.

The model is able to answer 5 out of 10 in the quiz. Still not very good. Changing to keep=10, we can get close on a another one (movie instead of movies) at get 6 of 10.