Text prediction app documentation - Dataset

Ivy Woo

16 February 2021

1 Raw data

The raw dataset is provided by Swiftkey and the app author gratefully acknowledges its generous contribution.

The raw data consists of three raw text files which contain (mainly) English texts extracted from three sources - blogs, news and twitter - in the U.S. during the late 2000s to early 2010s.

Table 1 summarizes the word count and line count from each of the three files. As shown, all files contain similar amount of words. Note that the line counts here convey little information, since a line in the raw data does not necessarily correspond to one sentence, and it can also end with an incomplete sentence.

	Blogs	News	Twitter
Text file size (MB)	200	196	159
Word count	38,050,950	35,628,125	31,073,243
Line count	899,288	1,010,242	2,360,148

Table 1: Word and line counts of raw data files.

Figure 1 is a plot of the log-count of the n-th most frequently appearing words (upper and lower cases not differentiated) in the blogs data for $n \in \{1, 2, \ldots, 10000\}$. Pay attention to that only a small fraction of words appear often, while most of the words have only a small count. The same pattern is also observed for the other two data sources.

Figure 2 is a second plot on the blogs data, displaying the cumulative fraction of total word counts constituted by the corresponding number of the most frequently appearing words, which again suggests that, a small fraction of the most common words accounts for a large proportion of the total word count. For example, around 2000 out of the over 300000 unique words already account for 80% of the total word count of the data.

2 Data processing

The raw data is cleaned in three aspects:

1. To tokenize the text lines into single words.

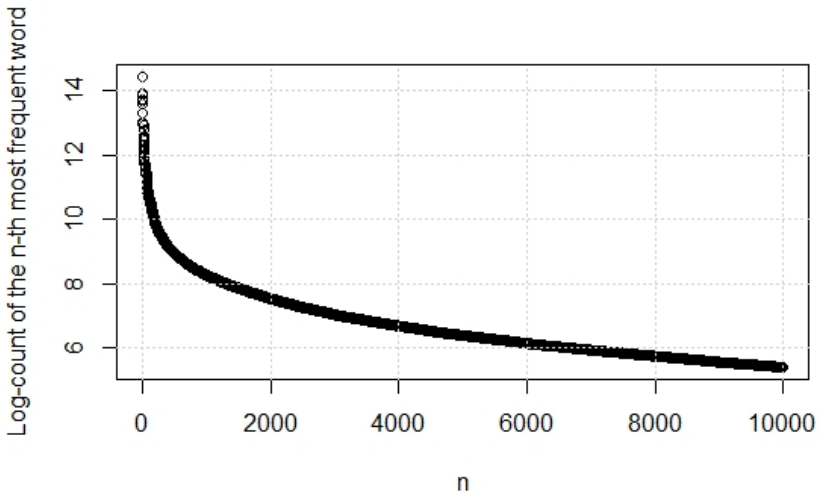

Figure 1: Log-count of the 10000 most frequent words in the blogs data.

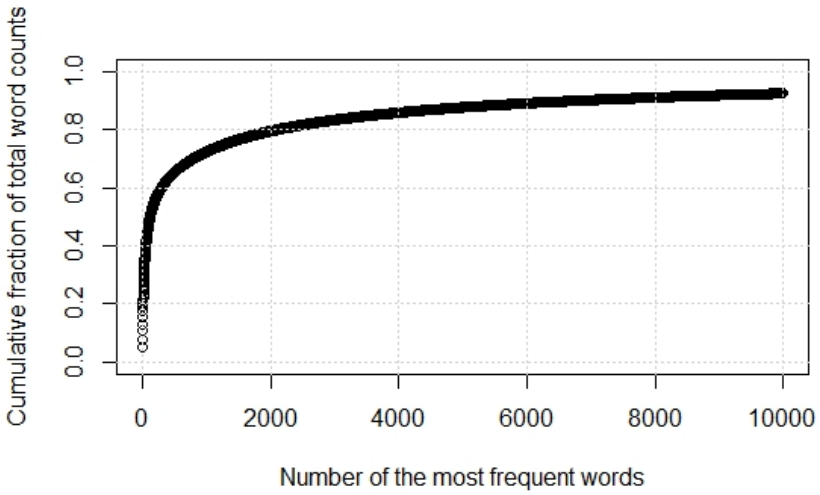

Figure 2: Cumulative fraction of total word counts constitute by the corresponding number of most frequent words in the blogs data.

2. To remove all tokens which are not English alphabets nor punctuations supported by the app. These could be numbers, English short-forms, URLs, texts in other languages, etc. In our app the punctuations supported are comma (",") and full stop (".").

3. To add/replace specific tokens into the collection of tokens (refer to Sections 2.1 and 2.2).

For these purposes, the R library tokenizers is chosen to assist the process. [1] The raw data is passed to a pipe of the library functions and R base functions to accomplish the above tasks. Refer to the functions documentation for details on the function for tokenization.

2.1 Sentence-begin token

A sentence-begin tag, the <s> token[2], is added to the beginning of each tuple of tokens from the same sentence. With this, as soon as the app user inputs one word, the app outputs prediction(s) by searching through the 3-grams (with the first word being <s>, the second word being the one input by the user, and the last word the prediction) instead of 2-grams. Similarly, the 4-grams can be searched through already when the app user inputs two words. Prediction accuracy is improved through the extra information of the text being (or not being) at the sentence begin.

2.2 Unknown token

The unknown token <unk> is introduced to replace the words which do not occur in the dataset frequently. This is done for three reasons:

1. To largely reduce the number of unique tokens in the data, which is tremendously helpful in simplifying and reducing the size of the prediction model.

2. When app user inputs some word(s) which does not exist in our dataset, instead of returning an error, the input(s) is replaced by <unk> and the n-grams with <unk> in the corresponding position are searched, so that an output can be generated.

3. There exist typos/non-standard-English words in the raw data (especially from the Twitter data). Through replacing the relatively uncommon words, these unwanted texts are also removed.

[1] A variety of R libraries for text mining/natural language processing including quanteda, tm, RWeka and text2vec have been considered. The tokenizers library is found to best fit the needs of this project due to its ability to flexibly split text chunks into sentences/words, to distinguish some (although not all) English short forms which end with a full stop from sentence end, and to effectively remove extra punctuations/symbols in tokens.

[2] For sentence. Irrelevant to the html strike-through tag.

Recall that a relatively small amount of words constitute a high proportion of the total word count (refer to Figures 1 and 2 and the discussion therein). A certain percentage α is chosen to be the threshold, the most commonly appearing words which constitute just over α of the total word counts will be retained and the remaining words will be replaced by <unk>. In the current model α is set as 90%.

3 Processed data

The three sources of data are processed separately in the same way as specified in Section 2. The processed data has attributes as displayed in Table 2. Note that the token counts in Table 2 are higher than the word counts in Table 1, since the punctuations "," and "." form also single tokens and the sentence-begin tokens are added.

	Blogs	News	Twitter	All sources
Token count	42,859,210	39,094,206	36,938,806	118,892,222
Sentence count	2,375,718	2,024,588	3,770,155	8,170,461
Number of unique tokens	6085	7273	4957	9015
Minimum number of appearance such that a word is included in the vocabulary	429	367	395	-
Number of unique words replaced by <unk>	237,656	195,323	283,546	-

Table 2: Attributes of the processed data.

4 Text prediction

The Shiny app uses a 4-gram model to predict the next word given some text. (See the model-construction documentation for a brief summary of an n-gram model, or e.g. **?** for a detailed explanation.)

A number of self-defined functions are used to run a linear scan of the processed data, to write all 2-grams, 3-grams and 4-grams into a set of nested dictionaries, and to output prediction(s) of the next word of some text base on

the frequencies of the n-grams. Refer to the functions documentation as well as the model-construction documentation for further information in this regard.