

aRianna: R package for consistency scores in text data *

Ke-Li Chiu *University of Toronto*
Rohan Alexander *University of Toronto*

In this paper, we introduce a process to clean the text extracted from PDFs using n-gram models that is widely used in statistical natural language processing. Our approach compares originally extracted text with the text generated from, or expected by, these methods using earlier text as stimulus. To guide this process, we introduce the notion of a consistency score, which refers to the proportion of text that is unchanged by the model. This is used to monitor changes during the cleaning process, and to compare the messiness of different texts. We illustrate our process on text from the book *Jane Eyre* and introduce both a Shiny application and an R package to make our process easier for others to adopt.

Keywords: text-as-data; natural language processing; quantitative analysis.

1. Introduction

When we think of quantitative analysis, we may like to think that our job is to ‘let the data speak’. But this is rarely the case in practise. Datasets can have errors, be biased, incomplete, or messy. In any case, it is the underlying statistical process of which the dataset is an artifact that is typically of interest. In order to use statistical models to understand that process, we typically need to clean and prepare the dataset in some way. This is especially the case when we work with text data. But this cleaning and preparation requires us to make many decisions. To what extent should we correct obvious errors? What about slightly-less-obvious errors? Although cleaning and preparation is a necessary step, we may be concerned about the extent to which have we introduced new errors, and the possibility that we have made decisions that have affected, or even driven, our results.

In this paper we introduce the concept of consistency in a text corpus. Consistency refers to the proportion of words that are able to be forecast by a statistical model, based on preceding words and surrounding context. Further, we define internal consistency as when the model is trained on the corpus itself, and external consistency as when the model is trained on a more general corpus. Together, these concepts provide a guide to the cleanliness and consistency of a text dataset. This can be important when deciding whether a dataset is fit for purpose; when carrying out data cleaning and preparation tasks; and as a comparison between datasets.

To provide an example, consider the sentence, ‘the cat in the...’. A child who has read this book could tell you that the next word should be ‘hat’. Hence if the sentence was actually ‘the cat in the bat’, then that child would know something was likely wrong.

*Thank you to **X, Y and Z** for helpful comments. A Shiny application for interactive and small-scale examples is available: <https://kelichiu.shinyapps.io/Arianna/>. An R package for larger scale applications is available: <https://github.com/RohanAlexander/arianna>. Our code and datasets are available: https://github.com/RohanAlexander/consistency_scores_in_text_datasets. Comments on the 01 September 2020 version of this paper are welcome at: rohan.alexander@utoronto.ca.

The consistency score would likely be lower than if the sentence were ‘the cat in the hat’. After we correct this error, the consistency score would likely increase. By examining how the consistency scores evolve in response to changes made to the text during the data preparation and cleaning stages we can better understand the effect of the changes. Including consistency scores and their evolution when text corpuses are shared allows researchers to be more transparent about their corpus. And finally, the use of consistency scores allows for an increased level of automation in the cleaning process.

We employ n-gram models to calculate consistency scores and generate word candidates for text corrections. The n-gram approach involves identifying two, three, or more, words that are commonly found together. In our application, we use the sequence of three words, or tri-grams, to calculate consistency scores and generate word candidates for text corrections. By using tri-grams, we identify the first two words in the sequence and evaluate if the last word in the sequence is “as expected” by comparing it to the corresponding tri-gram in a comparison dataset. The comparison dataset can be internal, meaning that the dataset is the source of the text being evaluated. On the other hand, a comparison dataset is external when the dataset is from a different source than where the text being evaluated is from, and is a larger and more universal dataset that holds generalizability. The external dataset we use in the application is constructed using a subset of text data sourced from Open WebText Corpus by [Gokaslan and Cohen \(2019\)](#). Since the content of Open WebText Corpus is from the web, the variety of the text data is promising for meeting the generalizability and universality an external dataset should possess.

In this paper, we introduce the aRianna R Package with the following structure: In Section 2, we provide an overview of language model and n-grams. In section we deconstruct the package functions and provide demonstration example of the package usage. Finally, we discuss the limitation and the future directions of aRianna in Section 4. Beyond this paper, we construct a Shiny app available at: <https://kelichiu.shinyapps.io/aRianna/>. That app computes internal and external consistency scores for corpus excerpts, and have developed an R Package, aRianna, that allows our approach to be used on larger datasets, which is available at: <https://github.com/RohanAlexander/aRianna>.

2. Background

Language Model

Given the nature of human languages, some combinations of words tend to occur more frequently than others. Think of ‘good’, which is more often followed by ‘morning’, than ‘duck’. As such, we could consider English text production as a conditional probability, $\Pr(w_k|w_1^{k-1})$, where k is the number of words in a sentence, w_k is the predicted word, and w_1^{k-1} is the history of the word occurring in a sequence ([Brown et al., 1992](#)). In this way, the generation of some prediction, w_k , is based on the history, w_1^{k-1} . This is the underlying principle of all language models. Essentially, a language model is a probability distribution over sequences of words. The goal of statistical language modeling is to estimate probability distributions over different linguistic units — words, sentences, and even documents ([Bengio et al., 2003](#)). However, this is difficult as language is categori-

cal. If we consider each word in a vocabulary as a category, then the dimensionality of language becomes large (Rosenfeld, 2000). The reason that there is such a variety of statistical language models is that there are various ways of dealing with this fundamental problem.

N-gram Models

The foundation of an n -gram language model is the conditional probability set-up introduced above. An n -gram model is a probabilistic language model that predicts the next word in a sequence of words (Bengio et al. (2003)). The n in n -gram refers to the number of words in that sequence. Consider the following excerpt from *Jane Eyre*: ‘We had been wandering’. ‘We’ is a uni-gram, ‘We had’ is a bi-gram, ‘We had been’ is a tri-gram, and ‘We had been wandering’ is a 4-gram. Notice that the two tri-grams in this excerpt, ‘We had been’, and ‘had been wandering’, overlap. The use of n -gram models enables us to assign probabilities to both the next sequence of words and just the next word. For instance, consider the two sentences: ‘We had been wandering’, and ‘We had been wangling’. The former is likely to be more frequently encountered in a training corpus. Thus, an n -gram would assign a higher probability to the next word being ‘wandering’ than ‘wangling’, given the sequence ‘We had been’.

To predict the next word, we have to take the sequence of preceding words into account, which requires knowing the probability of the sequence of words. The probability of a sequence appearing in a corpus follows the chain rule:

$$\Pr(\text{We,had,been,wandering}) = \Pr(\text{We}) \times \Pr(\text{had} \mid \text{We}) \times \Pr(\text{been} \mid \text{We,had}) \times \Pr(\text{wandering} \mid \text{We,had,been})$$

However, the likelihood that more and more words will occur next to each other in an identical sequence becomes smaller and smaller, making prediction difficult. Alternatively, we can approximate the probability of a word depending on only the previous word. This is known as the ‘Markov assumption’ and it allows us to approximate the probability using only the last n words (Brown et al., 1992):

$$\Pr(\text{We,had,been,wandering}) \approx \Pr(\text{We}) \times \Pr(\text{had} \mid \text{We}) \times \Pr(\text{been} \mid \text{had}) \times \Pr(\text{wandering} \mid \text{been}).$$

As a bi-gram model only considers the immediately preceding word, under the Markov assumption, an n -gram model can be reduced to a bi-gram model with n being any number:

$$\Pr(w_n \mid w_1^{n-1}) \approx \Pr(w_n \mid w_{n-1})$$

3. The aRianna R Package

Package Dependencies

There are a variety of ways to implement an n -gram model within R R Core Team (2019) including using R packages such as Quanteda (Benoit et al., 2018), tidyText (Silge and Robinson, 2016) and tm (Feinerer and Hornik, 2019). In our package, we used the Quanteda

R package because it has a comprehensive set of functions for conducting text analysis. We also employed dplyr (Wickham et al., 2018) for data frame manipulation and tibble (Müller et al., 2020) for transforming data frames to the tibble format:

```
install.packages("quanteda")
install.packages("dplyr")
install.packages("tibble")
```

Make Internal Data Set Function

To obtain the internal consistency score, we will construct a internal consistency dataset from a body of text that is the source of the text being evaluated. The function `make_internal_consistency_dataset` is created for this purpose. The function turns the `body_of_text` into an internal consistency dataset which the text being evaluated is compared with in the following steps. To do so, the function first transforms the body of text into word tokens with the punctuation removed and the letters turned to lowercase:

```
tokens_from_example <-
  quanteda::tokens(body_of_text, remove_punct = TRUE, )
tokens_from_example <-
  quanteda::tokens_tolower(tokens_from_example)
```

Next, the `make_internal_consistency_dataset` function turns the individual tokens into tri-grams, and keeps only the tri-grams that appears in the data for more than once. The common tri-grams are preserved in a tibble:

```
# Create ngrams from the tokens
toks_ngram <-
  quanteda::tokens_ngrams(tokens_from_example, n = 3)

# Convert to tibble so we can use our familiar verbs
all_tokens <-
  tibble::tibble(tokens = toks_ngram[[1]])

# We only want the common ones, not every one.
all_tokens <-
  all_tokens %>%
  dplyr::group_by(tokens) %>%
  dplyr::count() %>%
  dplyr::filter(n > 1) %>%
  dplyr::ungroup()
```

Finally, the `make_internal_consistency_dataset` function splits the tri-grams to two parts: the previous two words and the last word. In the end, each tri-gram contains its original three-word sequence, the first two words, and the last word:

```

# Create a tibble that has the first two words in one column then the third
all_tokens <-
  all_tokens %>%
  dplyr::mutate(tokens = stringr::str_replace_all(tokens, "_", " "),
                first_words = stringr::word(tokens, start = 1, end = 2),
                last_word = stringr::word(tokens, -1),
                tokens = stringr::str_replace_all(tokens, " ", "_"),
                first_words = stringr::str_replace_all(first_words, " ", "_")
  ) %>%
  dplyr::rename(last_word_expected = last_word) %>%
  dplyr::select(-n)

```

Generate Internal Consistency Score Function

After we have the internal consistency dataset, we need a function to compare the text being evaluated and the internal dataset in order to retrieve the internal consistency score — the `generate_internal_consistency_score` function. Similar to the `make_internal_consistency_dataset` function, the function first transforms the text being evaluated into tokens, then split tri-grams and preserve them into a tibble:

```

# Create tokens with errors
tokens_from_example_with_errors <-
  quanteda::tokens(text_to_check, remove_punct = TRUE)
tokens_from_example_with_errors <-
  quanteda::tokens_tolower(tokens_from_example_with_errors)

# Create ngrams from the tokens with errors
toks_ngram_with_errors <-
  quanteda::tokens_ngrams(tokens_from_example_with_errors, n = 3)
all_tokens_with_errors <-
  tibble::tibble(tokens = toks_ngram_with_errors[[1]])

all_tokens_with_errors <-
  all_tokens_with_errors %>%
  dplyr::mutate(tokens = stringr::str_replace_all(tokens, "_", " "),
                first_words = stringr::word(tokens, start = 1, end = 2),
                last_word = stringr::word(tokens, -1),
                tokens = stringr::str_replace_all(tokens, " ", "_"),
                first_words = stringr::str_replace_all(first_words, " ", "_")
  )

```

Next, the function combines the text tokens tibble and the internal consistency tibble by the `left_join` function of `dplyr`. The `left_join` function returns all the rows from the text tokens tibble and all the columns from both tibbles. By combining the two tibbles, the

function generates two additional columns — the `last_word` column are the last words of all tri-grams, and the `last_word_expected` contains the words that are in the internal consistency dataset. By comparing the words in `last_word` and `last_word_expected`. If a last word is in `last_word` but not `last_word_expected`, it is regarded as an unexpected word and the corresponding word in `last_word_expected` serves as a replacement candidate:

```
all_tokens_with_errors <-  
  all_tokens_with_errors %>%  
  dplyr::left_join(dplyr::select(consistency_dataset, -tokens),  
                  by = c("first_words"))
```

The consistency score is calculated by the number of words that can be predicted by the model divided by the total number of words in the data. The function identifies the counts of words that are predicted by the model and divided by the total number of words in the internal consistency dataset:

```
# Calculate the internal consistency score:  
internal_consistency <-  
  internal_consistency %>%  
  dplyr::ungroup() %>%  
  dplyr::filter(!is.na(as_expected)) %>%  
  dplyr::count(as_expected) %>%  
  dplyr::mutate(consistency = true_count/sum(n))
```

The `generate_internal_consistency_score` function also lists the identified text errors and generates replacement candidate to the text errors:

```
# Identify which words were unexpected  
unexpected <-  
  all_tokens_with_errors_only %>%  
  dplyr::mutate(as_expected = last_word == last_word_expected) %>%  
  dplyr::filter(as_expected == FALSE) %>%  
  dplyr::select(-tokens)
```

Generate External Consistency Score

The `generate_external_consistency_score` function works identically with the `generate_internal_consistency_score`. The only difference is that it compares the text being evaluated with a external consistency dataset that is larger and more general. The external consistency dataset we employed in the application is constructed using a subset text data from Open WebText Corpus by [Gokaslan and Cohen \(2019\)](#).

Demonstration

To install the package and load the library:

```
devtools::install_github("RohanAlexander/arianna")
library(aRianna)
```

For demonstration, we use the first paragraph of Jane Eyre as the internal text data. A sentence within the paragraph is modified with the intention to contain an error and serves as the text to be evaluated:

```
body_of_text <- "There was no possibility of taking a walk that day.
We had been wandering, indeed, in the leafless shrubbery an hour in
the morning; but since dinner (Mrs. Reed, when there was no company,
dined early) the cold winter wind had brought with it clouds so sombre,
and a rain so penetrating, that further out-door exercise was now out
of the question."

text_to_evaluate <- "when there was na company"
```

The first step is to turn the body of text into the internal consistency dataset. The generated internal consistency dataset is a tibble that contains the tri-grams that appear in the internal text data more than once. Since only `there_was_no` has more than one occurrence, it is the only reference tri-gram in the internal consistency dataset:

```
internal_consistency_dataset <-
  aRianna::make_internal_consistency_dataset(body_of_text)
internal_consistency_dataset
## # A tibble: 1 x 3
##   tokens      first_words last_word_expected
##   <chr>      <chr>      <chr>
## 1 there_was_no there_was  no
```

The next step is to compare the text to be evaluated with the internal consistency dataset that we created in the previous step. The function `generate_internal_consistency_score` takes in two arguments — the text to evaluate and the internal consistency dataset. The function identifies the word “na” as an unexpected word, and generates the internal consistency score as zero. The function also lists “no” as the replacement of “na”:

```
aRianna::generate_internal_consistency_score(
  text_to_evaluate, internal_consistency_dataset)

## $`internal consistency`
## # A tibble: 1 x 3
##   as_expected      n consistency
##   <lgl>      <int>      <dbl>
## 1 FALSE          1          0
```



```
## $`unexpected words`
## # A tibble: 1 x 4
##   first_words last_word last_word_expected as_expected
##   <chr>      <chr>      <chr>          <lgl>
## 1 there_was  na          no             FALSE
```

To get the external consistency score, we will employ the `generate_external_consistency_score` function. The function `generate_internal_consistency_score` takes in only one argument — the text to evaluate, and compares it with the external consistency dataset. The function identifies the word “na” as an unexpected word, and generates the external consistency score as zero. Because the external dataset is larger, there are several words generated as the replacement candidates of “na”:

```
aRianna::generate_external_consistency_score(text_to_evaluate)

## aRianna::generate_external_consistency_score(text_to_evaluate)
## $`external consistency`
## # A tibble: 1 x 3
##   as_expected      n consistency
##   <lgl>          <int>      <dbl>
## 1 FALSE          2          0

## $`unexpected words`
## # A tibble: 11 x 4
##   first_words last_word last_word_expected as_expected
##   <chr>      <chr>      <chr>          <lgl>
## 1 there_was  na          a             FALSE
## 2 there_was  na          an            FALSE
## 3 there_was  na          no            FALSE
## 4 there_was  na          not           FALSE
## 5 there_was  na          nothing       FALSE
## 6 there_was  na          of            FALSE
## 7 there_was  na          one           FALSE
## 8 there_was  na          only          FALSE
## 9 there_was  na          plenty        FALSE
## 10 there_was na          someone       FALSE
## 11 there_was na          something     FALSE
```

4. Discussion

Language models underpinned by n-grams are widely applied in text prediction, spelling-correction, and machine translation (Brown et al., 1992). As demonstrated in our application, we use a tri-gram based model to calculate internal and external consistency scores and generate a list of words as text correction candidates. However, n-gram models do

not take the linguistic structure of language into account. For instance, [Rosenfeld \(2000, p. 1\)](#) discusses language in this context, saying that ‘...it may as well be a sequence of arbitrary symbols, with no deep structure, intention or thought behind’. The prediction of the next word is based on only a few preceding words and broader context is not taken into account. Hence next-word prediction using n-gram based language models can be limited. Here think of a two-gram involving the word ‘good’ ‘good morning’. At scale these can identify missing or unusual words, and work quickly, but they lack nuance. For instance, an equally reasonable two-gram involving the word ‘good’ is ‘good work’. However, because n-gram models does not take the surrounding text into account, it has not capacity of judging if ‘morining’ or ‘work’ is to be suggested as the next word after ‘good’.

In future works, we intend to investigate in more advanced language models such as word embeddings and Transformers. We consider pre-trained word embedding models including Word2Vec ([Mikolov et al., 2013a](#)) ([Mikolov et al., 2013b](#)) and GloVe ([Pennington et al., 2014](#)), which place each word in a multi-dimensional space such that distance between words can illustrate their relationship ([Bengio et al., 2003](#)). This approach of representing words in vectors has a long history, but the implementation of [Bengio et al. \(2003\)](#) has become the foundation for much subsequent work. Here, terms are encoded into feature vectors that represent different aspects of the term, and so each term is represented by a position in the vector space.

We also consider incorporating generative pre-trained transformer models such as GPT-2 ([Radford et al., 2019](#)) and GPT-3 ([Brown et al., 2020](#)) and BERT ([Devlin et al., 2018](#)). GPT-2, GPT-3 and BERT are pre-trained language models that are built based on the Transformer architecture ([Vaswani et al., 2017](#)). Proposed by [Vaswani et al. \(2017\)](#) from Google in 2017, the Transformer is a novel network architecture for neural network that outperforms RNN-based and CNN-based models in computational efficiency ([Vaswani et al., 2017](#)). Since the emergence of the Transformer model, most of the representative pre-trained models are built based on this architecture ([Hanretty et al., 2018](#)). Therefore, we can say that Transformer marks a new era of language models. In our future works, it will be important for us to implement different language models compare their effectiveness in the tasks of generating consistency score and providing suggested text corrections and improve the aRianna package.

References

- Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.
- Benoit, K., Watanabe, K., Wang, H., Nulty, P., Obeng, A., Müller, S., and Matsuo, A. (2018). quanteda: An r package for the quantitative analysis of textual data. *Journal of Open Source Software*, 3(30):774.
- Brown, P. F., Della Pietra, V. J., Desouza, P. V., Lai, J. C., and Mercer, R. L. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–480.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Feinerer, I. and Hornik, K. (2019). *tm: Text Mining Package*. R package version 0.7-7.
- Gokaslan, A. and Cohen, V. (2019). Openwebtext corpus. <http://Skylion007.github.io/OpenWebTextCorpus>.
- Hanretty, C., Lauderdale, B. E., and Vivyan, N. (2018). Comparing strategies for estimating constituency opinion from national survey samples. *Political Science Research and Methods*, 6(3):571–591.
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.
- Müller, K., Wickham, H., Francois, R., Bryan, J., and RStudio (2020). *tibble: Simple Data Frames*. R package version 3.0.3.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.
- R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.
- Rosenfeld, R. (2000). Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278.

- Silge, J. and Robinson, D. (2016). tidytext: Text mining and analysis using tidy data principles in r. *JOSS*, 1(3).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- Wickham, H., François, R., Henry, L., and Müller, K. (2018). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.6.