

Final Report

Aolong Li

2023-12-14

Objective

In this project, we aim to develop a text generation model using Markov chain principles. The primary goal is to build a system that can read a corpus of text, learn the patterns in sentence structure and word usage, and then generate new, coherent text based on this learning.

The theory behind this model is that we treat each word in a sentence as a random variable with various states, and the probability of transitioning from one word to another is the transition probability. The Markov assumption is that the probability of a word depends only on the previous word. We can generalize this to so-called n -gram model, that is, the probability of a word depends on the previous $n - 1$ words. The larger the n , the more complex the model, but the more accurate the prediction because it captures more information about the structure of the text. So, we need to find the transition probability by analyzing the training text. Then, we can use this probability to generate new text.

Our objectives are to:

1. Import and preprocess training data.
2. Tokenize the test into words and sentences for n -gram modeling.
3. Implement functions to calculate transition probabilities for words/ n -grams.
4. Create a text generation function that uses these probabilities. The initial state can be either a random word or a user-specified word.

Design Decisions

Data Import and Preprocessing

We first import the training data using the `load_train_data` function, which reads text files from a specified directory. The `preprocess` function then converts this text to lowercase and removes non-alphanumeric characters, because we do not want to distinguish “Word” and “word” in our model during training. Also, we want to keep punctuation mark and replace rest special characters (like @, &, #, etc.) with space, so that it won't affect/contaminate our dataset.

```
load_train_data <- function(directory){
  booklist <- list.files(directory, full.names=TRUE)
  train <- lapply(booklist, readr::read_file) %>% unlist()
  train_data <- paste(train, collapse = " ")
  return(train_data)
}

preprocess <- function(text){
  text <- tolower(text)
  text <- gsub("[^[:alnum:],.!?\\\"'"] , " ", text)
  return(text)
}
```

Tokenization

The `tokenize_words` and `tokenize_sentences` functions break down the text into words and sentences, respectively. The previous function is for generating n-grams based on the training text. The last function is to for predicting the punctuation mark at the end of a sentence. We will split the sentence by the punctuation mark, and then use the last word of the sentence to predict the punctuation mark. Here we don't want to split sentence "Mr. Li remember that pi is 3.1415926." by the period in "Mr." and "3.1415926", so we use negative lookbehind to avoid this situation. After get the data (transition probability from last word to punctuation mark), we will replace the punctuation mark with a special token `</s>`, and change it back after we get the prediction.

```
tokenize_sentences <- function(text){  
  pattern <- "(?  sentences <- unlist(strsplit(text, pattern, perl = TRUE))  
  return(sentences)  
}
```

Transition Probabilities

Using the `generate_n_grams` functions, we set up the structure for analyzing word transitions, we use the sliding window of size n to generate n-grams. Then we divide the n-grams into two parts: the first $n - 1$ words and the last word because our choice of way to store the transition probabilities. The `generate_transition_prob` function then calculates the probabilities of each word following a given word or sequence of words.

There are many ways to store the transition probabilities

$$P(w_i | w_{i-1}, \dots, w_{i-n}) = \frac{\text{count}(w_{i-n}, \dots, w_{i-1}, w_i)}{\text{count}(w_{i-n}, \dots, w_{i-1})}$$

We choose to store the probabilities as a three-column data frame, where the first column is the sequence of words (first $n - 1$ words in a n-gram), the second column is the next word (or choices for our predictions), and the third column is the probability. This is because we can easily use the `filter` function to get the probability of a given word following a given sequence of words, and this is what we need to generate text.

Similar for the transition probabilities from last word to punctuation mark, we store the probabilities as a three-column data frame, where the first column is the last word of the sentence, the second column is the punctuation mark, and the third column is the probability.

Note that for the n-gram model, we usually need to apply some smoothing technique, like Laplace smoothing, Turing-Good estimation, etc. These techniques are used to make the model more robust and avoid the zero probability problem if we have some words or sequences of words that are not in the training text. However, we don't need it here because this is a text generating program, all the words and sequences of words generated are in the training text. That is, if the model generated

$$\dots, w_i, w_{i+1}, \dots, w_{i+n-1}$$

which means that the model has seen the sequence of words, and therefore $w_{i+1}, \dots, w_{i+n-1}$ is also in the training text. The only exception is that the context the user input as the start of generating text. It is not realistic to include all possible n-grams in our model, because the total number will be $t(t-1) \dots (t-n+1)$, where t is the number of unique words in the English language. So, if we find that the text user entered is not in the training text, we will stop and give a warning message.

So we can first populate the dataframe with `x[1:(n-1)]` as previous, `x[n]` as current, where `x` is the n-gram. Then we group by previous and current, and then get the count the occurrence for each group. Then we group by previous and normalize to get the transition probability.

```
generate_transition_prob <- function(n_grams){  
  df <- do.call(rbind, lapply(n_grams, function(x) {
```

```

    data.frame(previous = x[1], current = x[2]))))
transition_prob <- df %>%
  group_by(previous, current) %>%
  summarise(count = n(), .groups = 'drop') %>%
  group_by(previous) %>%
  mutate(prob = count/sum(count)) %>%
  ungroup() %>%
  select(previous, current, prob)

return(transition_prob)
}

```

For training part, before we add special token </s> to the end of each sentence and tokenize the text word by word, we need to get the transition probability from last word to punctuation mark. Otherwise, all punctuation will be erased by `add_special_token`. After the training, the function will return a list of two data frames, one is the transition probability from last word to punctuation mark, the other is the transition probability from previous $n - 1$ words to current word, which are all we need to generate text.

```

train_model <- function(directory, n=2){
  train_data <- load_train_data(directory)
  text <- preprocess(train_data)

  last_word <- last_word_p(text)

  text <- add_special_token(text)
  tokens <- tokenize_words(text)

  n_grams <- generate_n_grams(tokens, n)
  transition_prob <- generate_transition_prob(n_grams)
  model <- list(transition_prob = transition_prob,
               last_word = last_word)
  return(model)
}

```

Text Generation

The `generate_text` function uses these transition probabilities to generate new text. It starts with a seed (or random) word and selects subsequent words based on the calculated probabilities, thus generating a coherent text sequence.

Evidence of Functionality

We conducted several tests to validate each function. For example, the tokenization functions correctly split text into words and sentences, handling edge cases like abbreviations and decimal points. The transition probability calculations were verified to sum to 1 for each word, ensuring mathematical correctness.