Report of Final Project “Language Model”

Zhiyin Tan

1. Function Implementation

1.1 Brief introduction

The following will introduce the details of the implementation of the various functions of this program. The overall idea is how to tokenize the training set text and intercept n-grams to obtain the occurrence probability of the latter word, and then generate a series of words through the probability model and finally combine them into the new text.

1.2 How does it tokenize text? (Extension1)

It uses the RegexpTokenizer inside the NLTK package because it satisfies more basic requirements, such as identifying "can't" as "can" and "t", and so on. Tokenize text changes from a string to a list of words or phrases.

1.3 How does it represent pad tokens and how does it generate the n-grams?

In fact, each sequence needs to be bounded. That is implemented by the function “get_ngrams” in lm.py. This function is divided into two parts. First, add pad tokens before and after the sequence based on the n-gram values entered by the user (' None' in practice, meaning null). The second part splits the sequences with pad into multiple n-grams. It sets up a loop that looks at every element in the sequences list and adds it and the next n-1 elements to a new list until the next-to-last n-1 element. The final result is a list that contains multiple n-grams lists.

1.4 How to get the frequency and probability of the latter word for each n-gram?

It is done through the variable "counts" on the “train” function in “class LanguageModel” of lm.py. First, create a new big dictionary "counts". Then set up a loop that extracts all elements of each n-gram except the last one from the previously generated “ngram_list” to form a tuple "key". Meanwhile, create a small dictionary "value", extract all elements of the last n-gram "ngram_list[I][-1]" as the key of the dictionary, and assign the value as "1". Finally, enter the tuple "key" as the key and the small dictionary "value" as the value into the large dictionaries. If there are different elements after the same "key" during the loop, add them directly to the small dictionaries. If the same element appears again, give the corresponding value "+1" in the small dictionary. In this case, "counts" is a dictionary that counts tuples as keys and counts small dictionaries with word’s frequency as values.

1.5 How to get the probability of the latter word for each n-gram?

It is done through the " normalize" function of lm.py. The frequency of a single word is divided by the decimal of the sum of the frequency of all words (greater than 0 and less than 1), which changes the frequency of each word in the small dictionary into probability. It returns a

dictionary that counts words as keys and counts word's probability as values.

1.6 How to get the latter word for each n-gram?

It is done through the " sample" function of lm.py. It defines an interval between 0 and 1 for each word according to the existing dictionary of word probability distributions. "Random_p" is a random number from 0 to 1, and when "random_p" falls into an interval, it returns the word corresponding to that interval.

1.7 How to generate a new text?

It is done through the "generate" function in "class LanguageModel" of lm.py and the "detokenize" function of corpus.py.

"Generate" function is divided into two parts. One is to find the beginning of the text "sentence_heads" from the training set, which is characterized by starting with "None". The other one is to start with "sentence_heads" and match it to "next_token" with the previous method until the latter word is "None". Finally, a list "predict" consists of a series of words that are generated in order according to the probability.

The function of "detokenize" is simply to combine the list "predict" into a string. What deserves special treatment is whether the Spaces before and after punctuation marks are retained.

2. Documentation

This project consists of two modules: test language model and store output text. When you open main.py, the screen displays a brief introduction, followed by three options. The first option is to test the language model, the second option is to generate and store new text, and the third option is to exit the program

If you select "1" and enter the test language model, the program will ask you which training set text you will use. After entering the file name, the program will ask you the n-gram value you want to train. After the numerical input, the program will run and train each line of the training set text as a sequence. It then returns a new text that is generated based on the training set and n-grams values. Then a message pops up asking what you want to do next, still offering you three options.

If you select "2" to enter the generate and store new text module, the program will ask you to define the file name where the new text is stored, and then enter the number of generated text items. This is followed by the filename that should be input for the training set text and the value of n-grams. The program will then run without returning any text on the screen. Instead, a file is automatically generated in the root directory of main.py, with the name set for the user and the content as the output of the new text, each new text in its own line. Then a message pops up asking what you want to do next, giving you two options to regenerate and save new text or exit the program.

If you select "3", you will exit the program.