Hashtag Segmentation

Changmao Li

1. **Inicialize the n-grams dict.**

    a. Use list of dict to mark n-grams. Here I just use 1-2 grams

    b. Remember the total word count, total_appearance_count, minimum_appearance_count for each gram

    c. Remember dict whose key is the first token and value is minimum appearance count among that token)

    d. Remember the token count in bigram for each single token

    e. Inicialize the common two character words in "twocharword.txt"

2. **Case Recovary**

    At the begining, use a list to memorize the index where the character is in the upper case.

    After decoding, use this list to recover the case, here is the algorithm:

a. Initialize two index: the start index of the segment, the last end index of the last segment

b. For each segment in the segment list:

c. The end index of this segment is start + word length -1

d. For index in the case list:

e. If index > end then break which makes sure the index doesn't change the word after the end of this segment. If index<=last end continue, which makes sure the index doesn't change the word before the beginning this segment.

f. Change the character in correspoding index from lower case to upper case

g. Update start and last end

2. **Deal with the upper case and number**

    In most cases, upper case is the start of a word, so we can directly seperate the word. Here is the algorithm:

    a. if the length of the case index list is bigger than 2, and the upper index is not continues, or the there is only one upper case but is not the start of the hashtag:

    b. Seperate the hashtag based on the upper case

c. After b, for each substring, caculate the segment list by using ordinary cases algorithm

d. Combine result together and recover the case and return the result

3. **Deal with hashtag longer than 22**

   Because the requirement contains one which is for each hashtag the program must use less than 1 minute to deal with, for hashtag longer than 22 the program cannot generate all segments within one minute. So I designed a recursive hueristic maxmaching way to calculate the segment of this kind of hashtag:

   a. split the hashtag to head and rem

   b. if rem<22, use oridinary cases algorithm, else use this function

   c. add the head and rem together and return

4. **Deal with symbol "_"**

   In most cases "_" is used to seperate word and word, so we can just split by it.

   a. split by '_' then get the substrings

   b. for every substrings we seperate again by using oridinary algorithm

   c. combine result together and recover the case and return the result

5. **Deal with numbers**

   In most cases, number can be seperated indepently, so we can just split it independently.

   a. seperate number independently then get the substrings

   b. for every substrings we seperate again by using oridinary algorithm

   c. for segment contains a lot of less than two characters then filter and combine them, if this segment is not "a,i" and not in my own dictionary "twocharword.txt".

   d. combine result together and recover the case and return the result

6. **Deal with ordinary hashtag**

   The ordinary hashtag is the hashtag that: with no number, no "_", no uppercase, here is the algorithm

   1. Generate all segments and compute the max probability by discounting algorithm of bi-gram.
   1. If there is no results , which means there are unknown words in the text, so jump to deal with unknown words.

2. for segment contains a lot of less than two characters then filter and combine them, if this segment is not "a,i" and not in my own dictionary "twocharword.txt".
3. For the discounting algorithm here, the difference here I added a weight for bi-gram probability which is 10, which can make the result more fit for bi-gram.

7. **Deal with unknown words**

   The mean idea here to deal with unknown words is to maximize the length of the single known word and maximize the number of the known words. Here I didn't use probability because probability really works bad for hashtag containing unknown words.