Word Segmentation

Apart from spelling errors discussed under non-word spelling errors, another major error typists are often prone to is the omission of space between different words. For example the sequence of words “Hello there” may often be typed as “Hellothere”. Segmentation of such ‘invalid’ words into their constituent segments is word segmentation. Word segmentation would be a very helpful feature in text editors. This function of segmentation is a part of our Spell check module.

Methodology

Given a word w of length c, the total number of segmentations possible is. Our task is to segment w in such a way that all or most of the constituent segments are valid words found in a corpus. In other words, the joint probability of occurrence of the constituent segments is to be maximized. Formally, if a word w is segmented into the value f= is to be maximized. Clearly this means that we assume the probability of occurrence of words in a corpus is independent of each other. We use the same corpus and dictionary we built for the purpose of non-word spelling error correction to compute the aforementioned values

There may also exist cases where one of the segments in a result may not occur in our corpus, assigning a probability of 0 to such a segment is not a good choice. For example consider w= ‘bbcamerica’ clearly we have to split w into the segments ‘bbc’ and ‘america’. In all likeliness the word *bbc* may not appear in our corpus and assigning a probability of 0 to such a segment would mean f=0 even though the segmentation is right. We hence had to wisely choose a value that is to be returned as the probability of such a word that does not appear in our corpus. We also had to keep in mind the fact that longer an unknown word, lesser is its likeliness of being a valid word. Keeping these conditions in mind we computed the probability of words that do not occur in the corpus as where N is the total number of words in our corpus and l is the length of the unfound word.

We defined two functions to help is in performing word segmentation:

1. wordSequenceFitness(l)- A function that returns f for a list l comprising of . For better scaling, the logarithm of probability each word is computed.
2. Segment(word) this function works on the basis of induction and returns the best segmentation of word.

As mentioned before segment(word) works on the basis of induction. In particular, we are working by induction on the length of a word. Assuming we know the optimal segmentations for all substrings not including the first letter, we can construct the best segmentation which includes the first letter. We look at all possible split pairs, including the one which considers the entire word as a good segmentation, and we find the maximum of those segmentations with respect to “wordSequenceFitness”. By induction we know that “segment” returns the optimal segmentation on every call, since each recursive call operates on a strictly smaller substring which does not include the first letter. Hence, we have covered all possible segmentations, and the algorithm is correct.

Steps of implementation:

1. We first define a function *segment* that produces all possible segmentations of a given word. This is easily implemented in Java using its easy to use substring functions for string.
2. We define *wordSequenceFitness* that operates on lists of words generated by *segment* and returns the combined probability ofthe constituent words. More precisely, logarithms of the probability values are returned in order to prevent generation of miniscule values.
3. Finally we define the function *segment* which contains the meat of the word segmentation task. Segment returns a list of words that correctly make up the argument (unsegmented word) of the function. This function is based on induction as mentioned above. This functioned is recursively called with substrings of the argument every time. The function returns an empty list when the argument is an empty string.