CSCI 404 Natural Language Processing

Winter 2018 - Homework II

Robert Harley Haley Beavers

1

We implemented several methods, listed and described below:

distance: calculates the Levenshtein edit distance between a target, and source string with given cost weights. This method was given.
backtrace: recursively finds all paths through our Levenshtein edit distance matrix which result in the calculated minimum Levenshtein edit distance.
align: produces a visual representation of the edits done between the target and source string given a valid path through our minimum edit distance matrix.
cost: calculates the cost of a given path through our minimum edit distance matrix.
construct: constructs our minimum edit distance using our previously defined distance function to populate cells.

We used the packages sys, random, and numpy. As for additional resources, we consulted the python documentation, and utilized Yudongs office hours.

2
The Noisy Channel Approach

The Noisy Channel approach uses statistics and the edit distance in order to get a ranked candidate list. Specifically, we use a language model to predict which candidate is more likely through Ngrams. The examples given in the slides are looking at the misspelled word both with context (bigram) and without context (unigram).

Unigram

Spell check via unigram will only consider the misspelled word itself. This means we need to generate a list of potential words it could be. To do this we find edit distance between the misspelled word and all words in our dictionary keeping all those with a Levenshtein edit distance below a threshold (e.g. 1). Afterwards we get a set of candidates C. From here we can rank our candidates based on their Levenshtein edit distances, but we can do better than that using statistics.

$$C = [C_1, C_2, C_3, \ldots, C_N] \tag{1}$$
$$C_i = \text{Candidate word from C} \tag{2}$$
$$W = \text{Misspelled word} \tag{3}$$
$$\hat{C} = \text{argmax}_{C_i \in C} P(C_i|W) \tag{4}$$
$$\tag{5}$$

We can get further information on how likely a specific candidate is to be the correct word by taking

into account common misspellings. Before we get into that however, let's take a look at the formula above (Eq. 5). We could use this formula to calculate the probability that we have the correct word given this misspelling, but there is an easier representation we can find using Bayes Theorem:

$$P(A|B) = \frac{P(A,B)}{P(B)} \tag{6}$$

$$= \frac{P(A)P(A,B)}{P(B)} \tag{7}$$

$$\tag{8}$$

The last form you see above is a simpler representation of Bayes Theorem. This is because it is easier to calculate the probability as products of probabilities with the data we have. So, using Bayes Theorem we can change the equation calculating $\hat{C}$ to:

$$\hat{C} = \text{argmax}_{C_i \in C} P(C_i|W) \tag{9}$$

$$= \text{argmax}_{C_i \in C} \frac{P(C_i)P(W|C_i)}{P(W)} \tag{10}$$

$$= \text{argmax}_{C_i \in C} P(C_i)P(W|C_i) \tag{11}$$

Before we dive into this final result, the reason we can pull out the term $P(B)$ from the denominator is because it's constant throughout the calculations in finding the maximum, we can drop the term. There are two components in finding the probability; the language model $P(C_i)$, and the channel model $P(W|C_i)$.

$$P(C_i) = \frac{\text{\# of occurrences of } C_i \text{ in corpus}}{\text{\# of total tokens in Corpus}} \tag{12}$$

$$(C_i)P(W|C_i) = \frac{\text{\# of occurrences 't' is deleted after 'c'}}{\text{\# of occurrences 'ct' is in the corpus}} \tag{13}$$

$$\tag{14}$$

The language model $P(C_i)$ is a simple calculation that is based off of the corpus. The channel model $P(W|C_i)$ is a calculation that can be based on the corpus alone, but it can also be calculated using a confusion matrix for the appropriate edit operation. These edit operations are delete, substitution, split, insert, and swapping/transposing.

We then use argmax to select the most likely candidate based on these stats.

Spell Check with context: The bigram approach

Bigrams allow for us to consider the context surrounding the misspelled word, and takes that context into account when finding the likely candidate. Like before, we get the candidate list from the edit distance. We then treat the tokens before or after as the second token in the bigram with the candidate as the other token. The acress example is as follows:

a stellar and versatile acress whose combination of sass and glamour...

So we can consider the two bigrams (versatile, C_i) or (C_i, whose). Using these surrounding tokens as context, we can predict which candidate is more likely to follow before or after these tokens. To do so we use a similar equation from the unigram calculation:

$$C = [C_1, C_2, C_3, \ldots, C_N] \tag{15}$$
$$C_i = \text{Candidate word from C} \tag{16}$$
$$W = \text{Surrounding token} \tag{17}$$
$$\hat{C} = \text{argmax}_{C_i \in C} P(C_i|W) \tag{18}$$
$$\tag{19}$$

The difference being we are no longer using the misspelled word or a confusion matrix in this step, we are merely seeing how likely the candidate word is to be found after the previous token (They would be swapped if considering the token afterwards). Once again, we can use Bayes Theorem and simplify our equation into a simpler form:

$$\hat{C} = \text{argmax}_{C_i \in C} P(C_i|W) \tag{20}$$
$$= \text{argmax}_{C_i \in C} \frac{P(C_i)P(W|C_i)}{P(W)} \tag{21}$$
$$= \text{argmax}_{C_i \in C} P(C_i)P(W|C_i) \tag{22}$$

Now we have our language models calculating the likelihood that the candidate appears in the corpus $P(C_i)$ and the probability of finding the previous token given the candidate $P(W|C_i)$.

$$P(C_i) = \frac{\text{\# of occurrences of } C_i \text{ in corpus}}{\text{\# of total tokens in Corpus}} \tag{23}$$
$$(C_i)P(W|C_i) = \frac{\text{\# of occurrences of the bigram (W, } C_i\text{) in the corpus}}{\text{\# of bigrams in corpus}} \tag{24}$$
$$\tag{25}$$

We then find the maximum from all the candidates using argmax in Eq. 22, and we have our primary candidate!

Conclusion

This can overall be generalized to an Ngram model using the chain rule. Providing more and more context to what candidate would be expected to appear given the data from our corpus. The larger the Ngram however, the fewer candidates we allow the model to predict. This is because the model is constrained by our corpus, and the more context we give, the fewer examples we have with that context.