Write-up Texts

Michael Liu

Q1. Noisy Channel Model:

The task is to build a noisy channel that can effectively identify hashtag with the most likelihood. Therefore, the probabilistic model I used in this problem is:
p(tag|context) ~ argmax_tag p(context|tag)p(tag)
p(context|tag) = p(w1|w2, w3, w4,..., wn, t)*p(w2|w3, w4,..., wn, t)*...*p(wn|t)*p(t)
For convenience sake and prevention of division problem, I used the sum of all probability of context words to compute the likelihood of message given specific hashtag. With this, I was able to reconstruct the program to solve the noisy channel model.

Q2. Mystery (A Zipf's Law Application):

The problem is to identify two mysterious text. One is generated completely by chance, and the other is generated based on real text. The approach I use to identify the two text is to use nltk freqDist to find out which plot is closer to resemble the Zipf's distribution. The rationale behind such approach is that if the text is constructed based on real text, distribution of scrambled words will stay the same, where random text will have an close-to uniform distribution for its words.

After running text analysis on both a and b, I find out that "a.txt" has an uniform word distribution, where "b.txt"'s distribution is closer to zipf's law. Therefore, a is random and b is from real text.

Q3. Text Gen Application:

The problem is to build a program that generates a paragraph that closely resemble the author's writing attribute. To best tackle this problem, I use the current three language models: unigram model, bigram model and trigram model, and combine them in a hierarchical way. The first word in the paragraph is generated by unigram as the most probable word that occur in the author text. The second word is generated with bigram distribution and the words after the second are all generated by trigram model.

However, there are three problems that come with my initial approach:
1. Unwanted symbol like "PNCT","NUM","" occurs frequently in the generated text.
    Fix: I simply remove those symbols from the word list that algorithm is generating paragraph from.
2. Occurrence of zero probability with one model. It is very likely to have "word" guess that has zero or close to zero probability from trigram language model.
    Fix: I use a hierarchical approach to solve this problem. If the algorithm is unable to come up with convincing candidate, it will downgrade its language model until it gives out promising word guess. [implementation see in code]

3. High probable words tend to form into a loop.

    Fix: After solving the previous two problems, I find out that high probable guess tend to form a loop that makes the paragraph repeat the same phrase over and over again. In order to deal with this problem, I introduce a randomizer that randomly choose a word between the top two candidates. This will prevent algorithm running into a word loop and possibly provide interesting mutations to the paragraph it generates.