**Description:**

I write 9 classes in JAVA for this project. Below I will explain each class and take “training.en” file as an example. For another two files just use the same methods and only change names.

Assumptions:

Characters not taking into consideration: /n, /r

Characters specially taken into consideration: space, <>, << >> (at the beginning and end of each line)

Unigram:

Read the file line by line and count the occurrence of each character in the file and store into a map. Count the total number and compute the probability of each character and store the probability into another map. Write these two maps into two files “UNI-EN-COUNT” and “UNI-EN-PRO” separately.

Bigram:

Basically just use the same method as that in Unigram but take two characters as a string (including the start and end symbol of each line) into consideration at one time. Store two characters as a string for the key with the corresponding count as value into a map. Write this map into a file named “BI-EN”.

Trigram:

Again use the same method as that in Unigram but take three characters as a string (including the start and end symbol of each line) into consideration at one time. Store three characters as a string for the key with the corresponding count as value into a map. Write this map into a file named “TRI-EN”.

Laplace:

Read files “UNI-EN-COUNT”, “BI-EN” and “TRI-EN” and get three maps. Virtually create a table and take the keyset of map in “BI-EN” for row and take the keyset of map in “UNI-EN-COUNT” for column.

Take the combination of row(i) and col(j) as a string. Create a laplace-map and put each string into this map (if the string exists in trimap, then put the corresponding value into laplace-map, otherwise put 0). Traverse the table row by row and if trimap contains this string as a key, do nothing. Otherwise, for this row, every item plus 1, that is, for every key in laplace-map, if it starts with the string of this row then plus its value by 1. Write this laplace-map into a file named “Laplace-EN”.

ZTest:

Compute the perplexity of each model above and output those perplexity into couple of files respectively.

Backoff:

Read files "UNI-EN-PRO","UNI-EN-COUNT","BI-EN" and "TRI-EN" to get the maps. Use “backoff” method to compute the probability and then compute the perplexity for each line in the file “test” and store them into a list. Then write this list into a file named “en\_backoff\_perplexity”.

Interpolation:

Read files "UNI-EN-PRO","UNI-EN-COUNT","BI-EN" and "TRI-EN" to get the maps. Use “Interpolation” method with lambdas equally distributed and compute the probability and compute the perplexity for each line in the file “test” and store them into a list. Then write this list into a file named “en\_interpolation\_perplexity”.

Interpolation\_new:

This class is for Task II. Use an improved interpolation method that perform better.

Util:

Some general static methods that can be used many times, such as writing a map into a file and reading a map from a file.

An excerpt of the three trigram language models for English, displaying all n-grams and their probability with the two-letter history *t h*

Please check the files “Laplace-EN-PRO”, “Backoff-EN-PRO” and “Interpolation-EN-PRO”.

Documentation that your probability distributions are valid (sum to 1)

Please check the files “UNI-DE-PRO-Readable”, “UNI-EN-PRO-Readable”, “UNI-ES-PRO-Readable”

The perplexity scores for all unsmoothed and smoothed language models

Please check the file “perplexity.xlsx”

Critical analysis of your results

From the data in “perplexity.xlsx”, we can see that the perplexity(average) of English is always the lowest no matter which model I use. So I get the conclusion that test data is written in English.

For the unsmoothed scores, there are majority of Infinity, meaning that when computing the probability for perplexity, a zero exists. The reason might be some n-grams in test data never exist in the training data, so we have no record of this n-grams.

For the smoothed scores, the scores of Infinity become much less. The reason is the method used (such as Laplace) take the situation we just mentioned into consideration. So in this case, smoothed scores perform better.

For the unsmoothed model, there are a little bit of Infinity scores in unigram (the character in test data never exists in the training data). However, the scores of bigram and trigram are almost Infinity. It means most of n-grams in test data never appears in the training data. As a result, we cannot tell from the scores from the unsmoothed bigram and trigram model that what language the test data is written in.

For the smoothed model, although there still exists some Infinity scores, the number of it is much less than that of unsmoothed. The perplexity scores for the document average in English of Laplace, Backoff and Interpolation model do not differ much (13.08, 9.34, 11.13) and they both less than that of Unigram model (22.69). This also prove that the smoothed models perform better than unsmoothed ones.

Task II

The class is “interpolation\_new”. The method is to fix the n-gram probabilities and then search for the λ values that best for three models. I read a research paper (Jelinek and Mercer, 1980). It says one way is to use the EM algorithm, an iterative learning algorithm that converges on locally optimal λs. However, I haven’t learned this algorithm before and find it very difficult to implement. So I just manually tried some combinations for λs. Finally, I find a combination: 0.5, 0.25, 0.25 that performs better than the origin one.

I also train this model using the training file “training.en” and read the file “test” to compute the perplexity, and find the average document perplexity is 10.55, which is less than the origin one. (11.13)