COMP 150-06
Problem Set 2

Weitong Ruan

For this homework, I first wrote a python script ('pset2.py') where I used bigram then I wrote another python script (pset2_ngram.py)where I used Ngrams, basically one can use whatever N they like. (Not with N = 1 for now). For the bigram script, I changed all data sets into a long list instead of a list of lists. Then I noticed that by changing them into a long list creates a new bigram: (<\s> , <s>), which is not supposed to appear if we process the data as a list of lists, so when I loop over all the elements I always check to ignore this bigram. Also, they are other types of related issue, like the N-the size of data set. I noticed that in order to get the sanity check numbers, we need to ignore the end_token when calculating the size of the data set. Although I don't understand why, but I manually subtract the corresponding number from all the set sizes that I used, then I got almost the same numbers with my 'pest2.py' script. In the 'pset2_ngram.py', the data set are kept as lists of lists, so no such issues, but this script has other issues, it works well for N = 2, but for N = 3, the perplexity from both linear interpolation and deleted interpolation are really large, which doesn't make sense cause theory tells us it should get smaller. One possible reason is there are some bugs, but I've spent several hours trying to debug and found nothing. It would be really helpful if the grader find it out and let me know. The other reason is the result is correct (hightly doubt it). Then the reason why the perplexity is large is because of overfitting.

Plus, I think it would be really helpful if how to clarify a little bit on how to deal with the start token and end token in class.

1.1 This part is easy, first, loop through the training set to build a dictionary and then go over the training set again to get rid of some tokens with counts less than two and build a vocabulary set. This part is implemented by a "getVoc" function. Then preprocess the training set, which is done by calling a "PreprocessText" function. As I've mentioned, in my 'pset2.py' script, I changed the function by returning two outputs, a after-preprocessed list of lists data set and a after-preprocessed long list.
1.2 This part is the fundamental building block of the language model. First I wrote a function called "getCounts", then estimate the probabilities following each smoothing type. To check whether a distribution is valid, I basically loop through every element in the dictionary (except the end_token, see the end_token creates a lot of problems!) and add all the probabilities of all possible the bigram that stems from that unigram. To implement this I created another dictionary of sets where for each unigram (key) in the dictionary, its value is a set of all bigrams that stems from this unigram, plus this dictionary is built in the 'getCounts' function.

To compute complexity, just follow the formula provided in the slides and I've done that in Machine learning class so I don't think it's a big issue. The only thing is emphasis is to use the correct unigram counts! In some cases, it uses the unigram counts of the first word (for computing probablity), in other cases, it uses the unigram counts of the second word (for estimating lambdas)

1.3 Given any unigram, the sum of probabilities of all bigrams stemming from this unigram should be one. Except the end_token, since for the very last end_token, there is no more word. My distribuion check results are attached in the back. The perplexity without smoothing on the test set easily goes to inf, this is because if we meet a new bigram, the probability of that bigram will be zero and hence the perplexity will be infinity.

1.4 Once we have smoothing, the perplexity will not be infinity. Compared with the other smoothing techniques, Laplace smoothing seems to be a quick fix. Since we don't want any bigram to have zero probability, why not add one count to each bigram to make them not zero and then to account for that and make the distribution valid, we add the vocabulary size to the denominator. Hence, the perplexity will decrease from previous one but we can do better.

1.5 Linear interpolation generates a much more smoothing conditional pdf (pmf) for the bigrams, the interpolated probability is based on the probability of the second unigram and the conditional probability of the bigram. Notice that here, we simply take the mean of those two probabilities which does not make that much sense since those two probabilities are not correlated. However, compared with Laplace, the perplexity should be smaller.

1.6 The deleted interpolation actually uses maximum likelihood estimates to estimate the best weights for each related probabilities. Note, a few details here, first, when estimating lambdas, iterate through all unique bigrams in the held_out_set instead of all bigrams, second, use the correct counts of unigram.

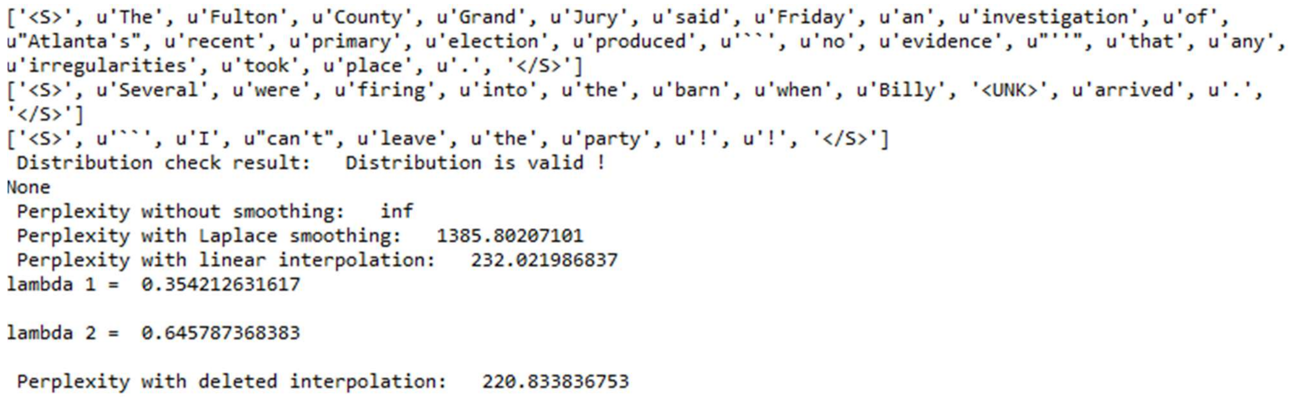
Extra credit:

Like I've mentioned in the beginning, I wrote a "ngram" script. The general idea is the same, one of the difficulties that I've met during the coding process is that for getting a slice of a tuple, especially when we only need one element, the returned tuple still has a ",", which makes it really annoying since for most of the time I'll need to write several cases separately.

For the trigram, I believe a more complex model would give us a better results hence the perplexity from, at least the one with deleted interpolation should be the smallest. Also, for n grams, my guess is that as n goes large, the perplexity goes down then up due to overfitting.

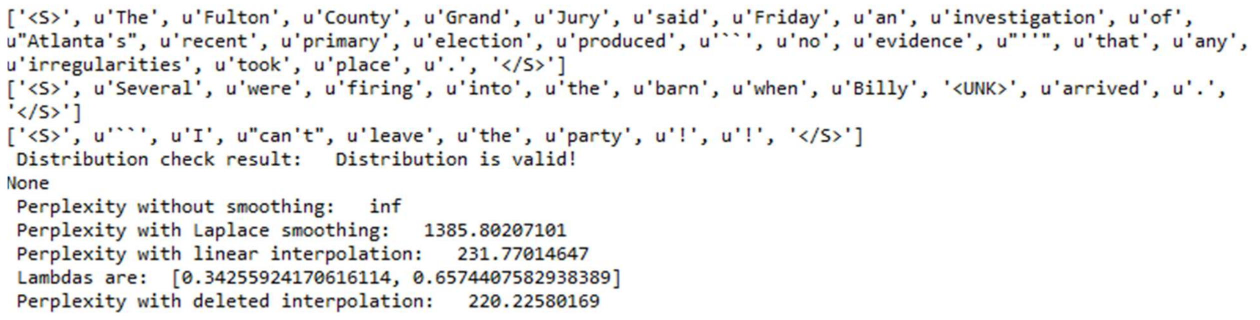
Results:

Bigram from ‘pset2.py’:



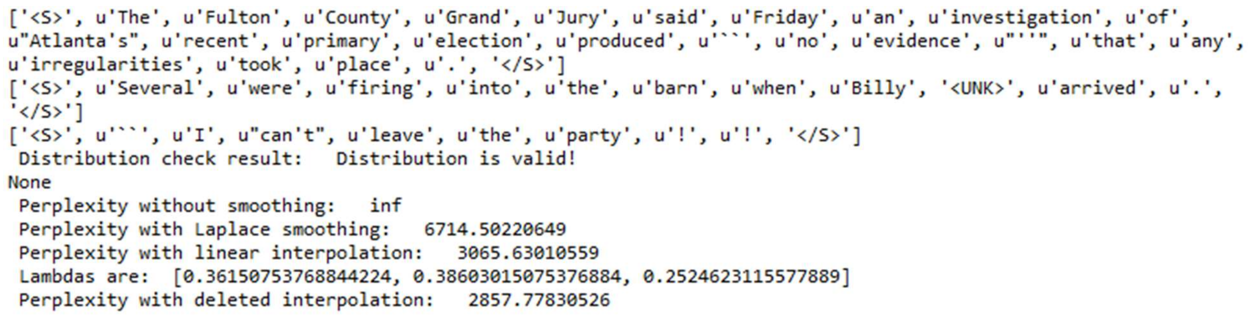
```
['<S>', u'The', u'Fulton', u'County', u'Grand', u'Jury', u'said', u'Friday', u'an', u'investigation', u'of',
u"Atlanta's", u'recent', u'primary', u'election', u'produced', u'``', u'no', u'evidence', u"''", u'that', u'any',
u'irregularities', u'took', u'place', u'.', '</S>']
['<S>', u'Several', u'were', u'firing', u'into', u'the', u'barn', u'when', u'Billy', '<UNK>', u'arrived', u'.',
'</S>']
['<S>', u'``', u'I', u"can't", u'leave', u'the', u'party', u'!', u'!', '</S>']
 Distribution check result:   Distribution is valid !
None
 Perplexity without smoothing:   inf
 Perplexity with Laplace smoothing:   1385.80207101
 Perplexity with linear interpolation:   232.021986837
lambda 1 =  0.354212631617

lambda 2 =  0.645787368383

 Perplexity with deleted interpolation:   220.833836753
```

Bigram from ‘pset2_ngram.py’:



```
['<S>', u'The', u'Fulton', u'County', u'Grand', u'Jury', u'said', u'Friday', u'an', u'investigation', u'of',
u"Atlanta's", u'recent', u'primary', u'election', u'produced', u'``', u'no', u'evidence', u"''", u'that', u'any',
u'irregularities', u'took', u'place', u'.', '</S>']
['<S>', u'Several', u'were', u'firing', u'into', u'the', u'barn', u'when', u'Billy', '<UNK>', u'arrived', u'.',
'</S>']
['<S>', u'``', u'I', u"can't", u'leave', u'the', u'party', u'!', u'!', '</S>']
 Distribution check result:   Distribution is valid!
None
 Perplexity without smoothing:   inf
 Perplexity with Laplace smoothing:   1385.80207101
 Perplexity with linear interpolation:   231.77014647
 Lambdas are:  [0.34255924170616114, 0.6574407582938389]
 Perplexity with deleted interpolation:   220.22580169
```

Trigram from ‘pset2_ngram.py’:



```
['<S>', u'The', u'Fulton', u'County', u'Grand', u'Jury', u'said', u'Friday', u'an', u'investigation', u'of',
u"Atlanta's", u'recent', u'primary', u'election', u'produced', u'``', u'no', u'evidence', u"''", u'that', u'any',
u'irregularities', u'took', u'place', u'.', '</S>']
['<S>', u'Several', u'were', u'firing', u'into', u'the', u'barn', u'when', u'Billy', '<UNK>', u'arrived', u'.',
'</S>']
['<S>', u'``', u'I', u"can't", u'leave', u'the', u'party', u'!', u'!', '</S>']
 Distribution check result:   Distribution is valid!
None
 Perplexity without smoothing:   inf
 Perplexity with Laplace smoothing:   6714.50220649
 Perplexity with linear interpolation:   3065.63010559
 Lambdas are:  [0.36150753768844224, 0.38603015075376884, 0.2524623115577889]
 Perplexity with deleted interpolation:   2857.77830526
```