Navraj Narula
Natural Language Processing
March 2, 2016

Problem Set 2: Langauge Modeling

1. Build a bigram language model for the Brown corpus. This is a 57Ksentence data set. Your training set will be the first 50K sentences of the corpus brown.sents()[:50000], your test set the last 3K brown.sents()[3000:], and your held out set the secondtolast 3K b rown.sents()[6000:3000].

    . 1.1. Text preprocessing: The NLTK version of the corpus is already tokenized. You will need to do minimal preprocessing involving introducing unknown tokens <UNK> and adding sentence boundary symbols < S></S> at the beginning and end of every sentence. Treat every word o ccurring not more than once in the training set as an unknown token. Transform all three of the data sets and print out the first sentence of each.

My Preprocess Text function took in sentences and a vocabulary. A new sentence was generated based on the concept stated above. I believe this line in my code summarizes it quite well:

new_sentences = [[start_token] + sentence + [end_token] for sentence in new_sentences]

A sentence boundary is added towards the beginning and end of sentences.

This condition further took care of replacing words that occurred less than twice with the unknown token:

if word_counts[word] < 2 or word not in vocabulary:

new_sentence[index] = unknown_token

My output for this function resulted in these three sentences:

```
In [17]:
runfile('/Users/navienarula/Desktop/NLP/Narula_PS2/pset2_template.py',
wdir='/Users/navienarula/Desktop/NLP/Narula_PS2')
['<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', u'Tilghman', u'arrived', u'.', '</S>']

['<S>', u'``', u'I', u"can't", u'leave', u'the', u'party', u'!', u'!',
'</S>']
```

. 1.2. Build a bigram language model class called B igramLM. The interface of this class is given to you in pset2_template.py. You will define the following functions

. 1.2.1. EstimateBigrams for estimating bigram MLE probabilities given the preprocessed training data.

. 1.2.2. CheckDistribution for checking the validity of your bigram estimates f r o m E s t i m a t e B i g r a m s . T h i s s h o u l d a s s e r t f o r e v e r y v a l i d u n i g r a m
context that the bigram probabilities sum to one, that is, Σ P(v | v) = 1.0. j j k

. 1.2.3. Perplexity for computing the perplexity given a test corpus. Use natural log and e xp functions throughout the problem set.

In order to ease the process of implementing all three of these functions within the Bigram LM model, I chose to firstly implement three other helper functions: one to count unigrams, one to count bigrams, and one to calculate log probabilities. You may view these functions within my actual .py file. EstimateBigrams, CheckDistribution, and Perplexity make use of several of these calculations within its own function. Perplexity is a good one to take note of since it incorporates all three. My main issue in implementing Perplexity is that I would run into Key Errors; I used a try and except block to take care of this issue:

```
try:
    logp += self.log_probs[key]
except KeyError:
    logp += 0
```

. 1.3. Using your class object from above, estimate a `BigramLM`. Check that it defines a valid probability distribution. Compute its perplexity on the test corpus. Explain your result.

```
""" Estimate a bigram_lm object, check its distribution, compute its perplexity.
"""

blm = BigramLM(vocabulary)
blm.EstimateBigrams(training_set_prep)
blm.CheckDistribution()
print "The perplexity of the training set is: ", blm.Perplexity(test_set_prep)
```

My result after completing this task resulted in a value very close to 15. Although this is certainly not perfect, I do believe that the probability distribution I've generated is fairly food at predicting the sample since this probability is "lower."

. 1.4. Introduce smoothing in your language model. First, implement Laplace smoothing. Compute the perplexity of the smoothed model on the test corpus. Explain your findings.

I would've expected that introducing smoothing to my language model would take care of sparsity issues; however, my perplexity value after implementing Laplace smoothing is around 74. In my Laplace smoothing function, I first calculated the length of the vocabulary and then iterated through values stored in my bigram counts dictionary in order to compute the log probability for each bigram and perform smoothing by adding one and adding V (i.e. the vocabulary size).

```
def laplace_smoothing(self, sentences):
    voc_size = len(self.vocabulary)
    for bigram, bigramcount in self.bigram_counts.iteritems():
        u, v = bigram
        self.log_probs[bigram] = log((bigramcount + 1.0)/(self.unigram_counts[u] + voc_size))
```

. 1.5. Implement simple linear interpolation (interpolating bigrams with unigrams). Use interpolation weights of (0.5, 0.5) and compute the perplexity of the model on the test corpus. Explain.

The results after incorporating SLI with these weights was much better than incorporating Laplace smoothing; however, the difference between not incorporating smoothing and incorporating SLI is very little. As noted, my first result was around 15 and this result from incorporating SLI is 11. It's not much of a difference, but it is an improvement. In implementing SLI, I firstly initialized the weights to 0.5. Then I counted the number of words in each sentence. It was helpful to implement another function in accomplishing this task. I then iterated through even key value pair in bigram_counts and return the log probabilities at the key I was examining.

```
def simple_linear_interpolation(self, sentences):

    l1, l2 = [0.5] * 2

    wordcount = count_words(sentences)

    for key, val in self.bigram_counts.iteritems():
        w,v = key
        bigram_prob = val / self.unigram_counts[w]
        unigram_prob = self.unigram_counts[v] / wordcount
        self.log_probs[key] = log(l2 * bigram_prob + l1 * unigram_prob)
```

. 1. 6 Implement the deleted interpolation algorithm to estimate the interpolation weights (SLP Figure 5.19) using the held out corpus. What are the interpolation weights that correspond to the unigram and bigram components? Recompute the perplexity of the interpolated model with the estimated interpolation weights. Explain your findings.

My results are certainly much better after implementing the deleted interpolation algorithm. My perplexity is around a 5.8, indicating that my model is indeed good—or has improved—in predicting the sample. The estimated weights I received are (0,0). This was after normalizing the results. I find this to be a little odd; however, the results returned from these weights is indeed satisfactory.

```
l1_norm = l1 / (l1 + l2) if l1 + l2 != 0 else 0
l2_norm = l2 / (l1 + l2) if l1 + l2 != 0 else 0

print "The estimated weights are: ", l1_norm, " and ", l2_norm
```

If you run my code, these are the results you will see. Please note that you should wait at most 15 seconds to see this result on your screen:

```
In [25]:
runfile('/Users/navienarula/Desktop/NLP/Narula_PS2/pset2_templat
e.py', wdir='/Users/navienarula/Desktop/NLP/Narula_PS2')
['<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', u'Tilghman', u'arrived', u'.',
'</S>']

['<S>', u'``', u'I', u"can't", u'leave', u'the', u'party', u'!',
u'!', '</S>']

The perplexity of the training set is:  14.91833365
The perplexity of the training set after Laplace smoothing is:
74.1818586266
The perplexity of the training set after SLI is:  10.5945648027
The estimated weights are:  0  and  0
The perplexity of the training set with estimated weights is:
5.85857665298
```