Part I

Question 1: N-Gram Models

Program *ngram.py* is included and all code referenced in this report is found in the source code and the various created output text files
The program is run with command line arguments to achieve certain results

Usage: **ngram.py [train|test] [filename] [2|3|0]**

All three command line arguments must be present or program will not run

When the **train** flag is set with **filename** and **2** or **3** the function *trainer* is run. Likewise, when **test** is flagged, the *tester* function is executed. *trainer* must be run before *tester* can execute. If **test** is flagged alongside a **0**, then perplexity of the **filename** is calculated

trainer: Files Created
traintemp.txt : corpus text with <s>, </s> characters added to sentences
trainvocab.txt : list of unique tokens (word types) with their counts
randomsentences(2/3)gram.txt : 50 randomly generated sentences
sentenceprobs(2/3)gram.txt : probabilities of ngrams and sentences(log space)

tester : Files Created
traintemp.txt : corpus text with <s>, </s> sentence indicators
evalsentences(2/3)gram.txt : evaluation output set of 30 random sentences

The following commands were executed to obtain all data for this report. Program will print status to stdout as it is running. Output files will differ due to randomness

```
# cd to directory containing ngram.py and train/test corpus
$ ngram.py train train.txt 2
$ ngram.py train train.txt 3
$ ngram.py test test.txt 0
$ ngram.py test test.txt 2
$ ngram.py test test.txt 3
```

Vocabulary Extraction
Before the vocabulary can be extracted from the training corpus, It must first be altered to indicate the beginning and end of every sentence. To our convenience, the *train.txt* file provided had a format of 1 sentence per line. With simple iteration, the new *traintemp.txt* is created and its contents assigned to a string. Using Python built-ins the string is split into tokens and kept in a list data structure. From here the use of the NLTK package helps us obtain a nicely formatted structure. Every unique token (word type) from the corpus, along with its count, is then written to *trainvocab.txt*

```
trainvocab.txt
WORD_TYPE : COUNT
0 : 105
00 : 4
    .
    .
    .
banking : 451
banknotes : 1
```

Creation of N-Grams
The numeric command line argument **2** or **3** dictate which ngram model to create. This value alongside the train corpus token list is passed into a function called *createNgram*:

```
def createNgram(tokenlist, n):
    return zip(*[tokenlist[i:] for i in range(n)])
```

which returns a new list of the text converted into ngrams. This list is then passed into NLTK's Frequency Distributions function

```
fdist = nltk.FreqDist(nglist)
```

to make use of the toolkit's various data analysis functions.

Random Sentences
For the generation of the 50 random sentences a local function was implemented that would randomly pick the next word in the sequence.

```
def randNextWord(fword):
    nwordlist = []
    sentence.append(fword)
    print ".",
    for sample in fdist:
        if sample[0] == fword:
            nwordlist.append(sample[1])
    randw = random.choice(nwordlist)
    if ngram == 3:
        randNext3Word(fword, randw)
    else:
        if randw == "</s>":
            sentence.append(randw)
            print ""
        else:
            randNextWord(randw)
```

randNextWord takes a string('fword') and gathers a list of possible words that can follow the 'fword' according to the bigram model, randomly chooses a word from this list, and recursively calls itself with the new random word. A complete sentence is formed when the next selected word ends up being the </s> token. The function must always be invoked with the <s> to begin the process. *randNext3Word* is a similar function but handles sentence creation according to the trigram model and only called within the original *randNextWord* function. After each sentence is created, the probability is calculated and the outputs set during each iteration and written to the file *randomsentences(2/3)gram.txt*:

```
randomesentences3gram.txt
SENTENCE
 : LOG PROBABILTY
<s> these were needed </s>
 : -28.2542871684
<s> electronics retailing business said thompson </s>
 : -50.0722144311
   .
   .
<s> vanguard index trust quarterly dividend which will bring
    about a 50 mln dlrs they said sunflower maize soybean and
    oranges and import bill </s>
 : -221.476527174
<s> details of next years presidential election </s>
 : -57.4865433664
```

The log probability shown in the file is calculated as the sentences are being generated. As pairs or triples of words (for bigrams and trigrams respectively) are found, the probability of the ngram is calculated by making use of the add-1/Laplace Smoothing method equation:

$$Bigram : P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V} \tag{1}$$

$$Trigram : P(w_i|w_{i-1}w_{i-2}) = \frac{C(w_{i-2}w_{i-1}w_i) + 1}{C(w_{i-2}w_{i-1}) + V} \tag{2}$$

Where P(A|B) gives us the probability given the counts of specified ngrams (C) and the vocabulary size (V). The respective code then prints theses calculations to another file named *sentenceprobs(2/3)gram.txt* giving the individual probabilty of each ngram and then a log-space probability of the entire sentence. We use the log-space probabilty to avoid the issue of underflow. Underflow is a problem encountered when repeatedly computing the products of standard floating point numbers. The results get too small that the number cannot be stored in the computer memory. In the probabilities text file the running total of the log-space probabilty after each ngram is displayed. The higher the value (closer to 0) the more likely that sentence is.

$$log(P(A) \times P(B) \times ... \times P(N)) = log(P(A)) + log(P(B)) + ... + log(P(N)) \tag{3}$$

```
for i in range(len(sentence)-1):
    probfile.write("({},{}) : ".format(sentence[i], sentence[i+1]))
    count = fdist[(sentence[i], sentence[i+1])]
    prob = float(count + 1) / (ctr[sentence[i]] + numvocab)
    probfile.write("{} : ".format(prob))
    prob = math.log(prob)
    sentProb = sentProb + prob
    probfile.write("{}\n".format(sentProb))
```

```
sentenceprobs2gram.txt
(B,A) : P(A|B) : log-space probability(running total)
(<s>,pzena) : 1.91038389164e-05 : -10.8656212527
(pzena,of) : 4.47497370953e-05 : -20.8800462409
(of,softwood) : 2.46947116275e-05 : -31.4889676823
(softwood,lumber) : 0.000156599552573 : -40.2507863139
(lumber,he) : 6.70705806077e-05 : -49.860551365
(he,charged) : 4.03909847322e-05 : -59.9774553131
(charged,an) : 4.46867459112e-05 : -69.9932889255
(an,unsustainable) : 4.06140849647e-05 : -80.1046845567
(unsustainable,pace) : 4.47507383872e-05 : -90.1190871698
(pace,analyst) : 4.46917387321e-05 : -100.134809059
(analyst,sees) : 4.44553112983e-05 : -110.155835173
(sees,one) : 6.65690320863e-05 : -119.773106245
(one,nonstops) : 4.19665526575e-05 : -129.851743867
(nonstops,</s>) : 4.47487358482e-05 : -139.86619123
    .
    .
    .
```

Perplexity
The perplexity is the probability, assigned by the language model, of a test set normalized by the number of words. Given by the equation: N = number words

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}} \tag{4}$$

$$PP(W) = P(w_1 w_2 w_N)^{-\frac{1}{N}} \tag{5}$$

The probability of every the entire test corpus is calculated by using the chain rule and converted into log-space. In our case there are 1467777 words in the test.txt corpus.

```
numwords = len(tokens)
high = math.pow(high, -1/numwords)
high2 = math.pow(high2. -1/numwords)

perplexity of bigram model: -2.5677496819e-07
perplexity of trigram model: -1.2005209498e-07
```

Minimizing perplexity is the same as maximizing probability. From this data it seems as though our trigram model is the better language model that best predicts the unseen test set.

Evaluation
In order to evaluate the models created by the training corpus, we run tests on the supplied test.txt corpus. The tester function can be run for both bigrams and trigrams decided by command line at runtime. Similar to extracting the vocab for the train corpus, the test.txt must be stripped of newline characters and <s>, </s> inserted at start and end of sentences. From the newly created *traintemp.txt* file, 30 random sentences are selected to be used for evaluation. From each sentence a random word was removed (not including special sentence tokens) and by using the respective ngram model, the word that has the highest likelihood or largest probability was selected from the training corpus. Since the add-1/Laplace Smoothing method was used, ngram pairs not found in the training corpus were still given a weight for selection. All output is written to *evalsentences(2/3)gram.txt*:

```
evalsentences3gram.txt
SENTENCE
DELETED: WORD
HIGHEST: LIKELIHOOD | P(A|B): PROBABILITY

<s> sale date and announcement of rates is set for tuesday march
    17 1987 </s>
DELETED: set
LIKELY: expected | P(A|B): 0.00857040379061

<s> the special dividend is to be paid july 27 </s>
DELETED: be
LIKELY: the | P(A|B): 0.0301340267049

<s> authorizations to purchase 50 mln dlrs worth of us wheat
    and wheat flour under public law 480 were issued to sudan
    today the agriculture department said </s>
DELETED: department
LIKELY: department | P(A|B): 0.00373827641125
    .
    .

evalsentences3gram.txt
<s> together with two other main shareholders it also advanced
    fermenta an additional 65 mln crowns until a new equity
    issue could be made </s>
DELETED: equity
LIKELY: york | P(A|B,C): 0.000598046381819
```

```
<s> gencorp has agreed to sell two of its independent stations
    wor in the new york area and khj in los angeles </s>
DELETED: angeles
LIKELY: angeles | P(A|B,C): 0.000268444364905

<s> gross national product grew 13 pct in the fourth quarter </s>
DELETED: pct
LIKELY: pct | P(A|B,C): 2.23758698619e-05
    .
    .
```

Looking at our two results we notice some observations. The bigram model often had less correct matches of the deleted word to complete the sentence than the trigram model. In the majority of runs performed, the bigram model almost always found a matching bigram in the test corpus that could be a possibility in the test corpus sentence. This was to be expected so the bigram model often suggests common words (highest probability) to be the replacement of the deleted word. Compared to the bigram model, the trigram model had more success correctly determining the missing word in the sentence. However the amount of times there was no matching trigram in the training corpus was much higher compared to bigrams. This method of add-1/Laplace Smoothing lowers the observed counts for every ngram in our training corpus in order to include unobserved vocabulary in the model. This is change is often large and gives too much weight to the unseen ngrams in our models. So as observed, when there was no matching trigram, the weight of the unseen vocabulary was often chosen.

Comments

- *ngram.py* as well as all output files referenced in this report have been included
- The Natural Language Toolkit (NLTK) Python module is required for execution
- Code has not been fully optimized. Generating the 50 random sentences repeatedly parses large corpus file. Not the most efficient method but do note program is not hanging but merely analyzing. Program will terminate when output file has been written.
- Subsequent runs will append to existing output files
- Some directions may have been interpreted open ended
- Some blocks of code were repeated for certain functions for debugging purposes
- Code sections and outputs were intentionally displayed as snippets for page constraints. See attached files for full dissection