Answers for Homework 1

Byung Kim
for CS74040: NLP Fall 2019 by Prof. Alla Rozovskaya
Due: 10/03/2019

In [1]:
```
import preprocessing
import training
import testing
```

Question 1:

How many word types (unique words) are there in the training corpus? Please include the padding symbols and the unknown token.

In [2]:
```
# First, I preprocess the training corpus
train_fp = 'data/brown-train.txt'
train_list = preprocessing.preprocess_train(train_fp)
# Then I build a dictionary from the preprocessed data
train_dict = preprocessing.build_dict(train_list)
```

In [3]:
```
print('There are', len(train_dict), 'word types in the training corpus.')
```

```
There are 15031 word types in the training corpus.
```

Question 2:

How many word tokens are there in the training corpus?

In [4]:
```
print('There are', len(train_list), 'word tokens in the training corpus.')
```

```
There are 498474 word tokens in the training corpus.
```

Question 3:

What percentage of word tokens and word types in each of the test corpora did not occur in training (before you mapped the unknown words to <unk> in training and test data)?

In [5]:
```
# First, the filepaths for the test data
test_fp = 'data/brown-test.txt'
learner_fp = 'data/learner-test.txt'
```

In [6]:
```
# The add_s_tag function in preprocessing.py creates a list with <s> and </s> tags
# I do this for all data: brown-train, brown-test, and learner-test
train_temp = preprocessing.add_s_tag(train_fp)
test_temp = preprocessing.add_s_tag(test_fp)
learner_temp = preprocessing.add_s_tag(learner_fp)

# I create a dictionary of brown-train and brown-test
train_tdict = preprocessing.build_dict(train_temp)
test_tdict = preprocessing.build_dict(test_temp)
learner_tdict = preprocessing.build_dict(learner_temp)
```

In [7]:
```
# For word tokens in test data but not in training data
def unseen_tokens_perc(test_list, train_dict):
    unseen_sum = 0
    for token in test_list:
        if token not in train_dict:
            unseen_sum += 1
    return float(unseen_sum)/len(test_list)

# For word types in test data but not in training data
def unseen_type_perc(test_dict, train_dict):
    unseen_sum = 0
    for key in test_dict:
        if key not in train_dict:
            unseen_sum += 1
    return float(unseen_sum)/len(test_dict)
```

In [8]:
```
print('For the brown-test data:')
print('The percentage of tokens not in the training data for brown-test is: {:.2%}'.format(
      unseen_tokens_perc(test_temp, train_tdict)))
print('The percentage of word types not in the training data for brown-test is: {:.2%}'.format(
      unseen_type_perc(test_tdict, train_tdict)))
print('--------------------------------------------------------------------------------')
print('For the learner-test data:')
print('The percentage of tokens not in the training data for learner-test is: {:.2%}'.format(
      unseen_tokens_perc(learner_temp, train_tdict)))
print('The percentage of word types not in the training data for learner-test is: {:.2%}'.forma
t(
      unseen_type_perc(learner_tdict, train_tdict)))
```

```
For the brown-test data:
The percentage of tokens not in the training data for brown-test is: 5.99%
The percentage of word types not in the training data for brown-test is: 22.76%
--------------------------------------------------------------------------------
For the learner-test data:
The percentage of tokens not in the training data for learner-test is: 5.05%
The percentage of word types not in the training data for learner-test is: 16.35%
```

Question 4:

What percentage of bigrams (bigram types and bigram tokens) in each of the test corpora that did not occur in training (treat <unk> as a token that has been observed).

In [9]:
```
# First, I have to preprocess the test data
test_list = preprocessing.preprocess_test(test_fp, train_dict)
learner_list = preprocessing.preprocess_test(learner_fp, train_dict)
```

In [10]:
```
# Then I build the bigram lists for all three data sets
train_bigrams = training.bigrams(train_list)
test_bigrams = training.bigrams(test_list)
learner_bigrams = training.bigrams(learner_list)

# And the bigram dictionary with counts for all three data sets
train_bdict = training.bigram_dict(train_list)
test_bdict = training.bigram_dict(test_list)
learner_bdict = training.bigram_dict(learner_list)
```

In [11]:
```
# Recycling unseen_tokens_perc and unseen_type_perc from Question 3
print('For the brown-test data:')
print('The percentage of bigram tokens not in the training data for brown-test is: {:.2%}'.form
at(
      unseen_tokens_perc(test_bigrams, train_bdict)))
print('The percentage of bigram types not in the training data for brown-test is: {:.2%}'.forma
t(
      unseen_type_perc(test_bdict, train_bdict)))
print('--------------------------------------------------------------------------------')
print('For the learner-test data:')
print('The percentage of bigram tokens not in the training data for learner-test is: {:.2%}'.fo
rmat(
      unseen_tokens_perc(learner_bigrams, train_bdict)))
print('The percentage of bigram types not in the training data for learner-test is: {:.2%}'.for
mat(
      unseen_type_perc(learner_bdict, train_bdict)))
```

```
For the brown-test data:
The percentage of bigram tokens not in the training data for brown-test is: 22.65%
The percentage of bigram types not in the training data for brown-test is: 38.52%
--------------------------------------------------------------------------------
For the learner-test data:
The percentage of bigram tokens not in the training data for learner-test is: 24.84%
The percentage of bigram types not in the training data for learner-test is: 38.64%
```

Question 5 & 6:

5. Compute the log probabilities of the following sentences under the three models (ignore capitalization and pad each sentence as described above). Please list all of the parameters required to compute the probabilities and show the complete calculation. Which of the parameters have zero values under each model?

- He was laughed off the screen .
- There was no compulsion behind them .
- I look forward to hearing your reply .

6. Compute the perplexities of each of the sentences above under each of the models.

The answers to both question 5 and 6 will be shown per model together.

In [12]:
```
from testing import perplexity
```

In [13]:
```python
# First, I preprocess the three sentences and put them into a list
s1 = 'He was laughed off the screen .'
s2 = 'There was no compulsion behind them .'
s3 = 'I look forward to hearing your reply .'

s1_list = preprocessing.sentence_preprocess(s1, lst=[])
s2_list = preprocessing.sentence_preprocess(s2, lst=[])
s3_list = preprocessing.sentence_preprocess(s3, lst=[])

print(s1_list)
print(s2_list)
print(s3_list)
```

```
['<s>', 'he', 'was', 'laughed', 'off', 'the', 'screen', '.', '</s>']
['<s>', 'there', 'was', 'no', 'compulsion', 'behind', 'them', '.', '</s>']
['<s>', 'i', 'look', 'forward', 'to', 'hearing', 'your', 'reply', '.', '</s>']
```

In [14]:
```python
# We need a dictionary of the training set before <unk> was substituted
# to see if any of these were unseen
# train_tdict from Question 3 is as such
s1_unk = preprocessing.unkify_test(s1_list, train_dict)
s2_unk = preprocessing.unkify_test(s2_list, train_dict)
s3_unk = preprocessing.unkify_test(s3_list, train_dict)

print(s1_unk)
print(s2_unk)
print(s3_unk)
```

```
['<s>', 'he', 'was', 'laughed', 'off', 'the', 'screen', '.', '</s>']
['<s>', 'there', 'was', 'no', '<unk>', 'behind', 'them', '.', '</s>']
['<s>', 'i', 'look', 'forward', 'to', 'hearing', 'your', 'reply', '.', '</s>']
```

Part A: Unigram Maximum Likelihood Model

In [15]:
```python
# First we train the unigram model
unigram_probs = training.unigram_train(train_list, train_dict)
```

The Parameters for Each Sentence:

$l = \frac{1}{M}\sum_{i=1}^{m} log_2 p(s_i)$
s refers to the size of the corpus

$$l_{s1} = \frac{1}{9} log_2(p(< s >) * p(he) * p(was) * p(laughed) * p(off) * p(the) * p(screen) * p(.) * p(< /s >))$$
$$= \frac{1}{9} log_2(\frac{c(<s>)}{s} * \frac{c(he)}{s} * \frac{c(was)}{s} * \frac{c(laughed)}{s} * \frac{c(off)}{s} * \frac{c(the)}{s} * \frac{c(screen}{s} * \frac{c(.)}{s} * \frac{c(</s>)}{s})$$

$$l_{s2} = \frac{1}{9} log_2(p(< s >) * p(there) * p(was) * p(no) * p(< unk >) * p(behind) * p(them) * p(.) * p(< /s >))$$
$$= \frac{1}{9} log_2(\frac{c(<s>)}{s} * \frac{c(there)}{s} * \frac{c(was)}{s} * \frac{c(no)}{s} * \frac{c(<unk>)}{s} * \frac{c(behind)}{s} * \frac{c(them)}{s} * \frac{c(.)}{s} * \frac{c(</s>)}{s})$$

$$l_{s3} = \frac{1}{10} log_2(p(< s >) * p(i) * p(look) * p(forward) * p(to) * p(hearing) * p(your) * p(reply) * p(.)$$
$$* p(< /s >))$$
$$= \frac{1}{10} log_2(\frac{c(<s>)}{s} * \frac{c(i)}{s} * \frac{c(look)}{s} * \frac{c(forward)}{s} * \frac{c(to)}{s} * \frac{c(hearing)}{s} * \frac{c(your)}{s} * \frac{c(reply)}{s} * \frac{c(.)}{s} * \frac{c(</s>)}{s}$$

None of these parameters are zero for the unigram model, as $< unk >$ is treated as an observed token
The function in unigram_predict prints all the individual log probabilities of the tokens.

In [16]:
```
s1_predict = testing.unigram_predict(s1_unk, unigram_probs)
print('The log probability of sentence 1 in the Unigram ML Model is: {:.3f}'.format(s1_predict
))
print('The perplexity of sentence 1 in the Unigram ML Model is: {:.1f}'.format(perplexity(s1_pr
edict)))
print('---------------------------------------------------------------------')
s2_predict = testing.unigram_predict(s2_unk, unigram_probs)
print('The log probability of sentence 2 in the Unigram ML Model is: {:.3f}'.format(s2_predict
))
print('The perplexity of sentence 2 in the Unigram ML Model is: {:.1f}'.format(perplexity(s2_pr
edict)))
print('---------------------------------------------------------------------')
s3_predict = testing.unigram_predict(s3_unk, unigram_probs)
print('The log probability of sentence 3 in the Unigram ML Model is: {:.3f}'.format(s3_predict
))
print('The perplexity of sentence 3 in the Unigram ML Model is: {:.1f}'.format(perplexity(s3_pr
edict)))
print('---------------------------------------------------------------------')
```

```
The unigram log probability for <s> is -4.261
The unigram log probability for he is -6.387
The unigram log probability for was is -6.597
The unigram log probability for laughed is -13.501
The unigram log probability for off is -10.276
The unigram log probability for the is -4.337
The unigram log probability for screen is -15.02
The unigram log probability for . is -4.486
The unigram log probability for </s> is -4.261
The log probability of sentence 1 in the Unigram ML Model is: -7.681
The perplexity of sentence 1 in the Unigram ML Model is: 205.2
---------------------------------------------------------------------
The unigram log probability for <s> is -4.261
The unigram log probability for there is -8.648
The unigram log probability for was is -6.597
The unigram log probability for no is -8.964
The unigram log probability for <unk> is -5.237
The unigram log probability for behind is -11.552
The unigram log probability for them is -9.262
The unigram log probability for . is -4.486
The unigram log probability for </s> is -4.261
The log probability of sentence 2 in the Unigram ML Model is: -7.030
The perplexity of sentence 2 in the Unigram ML Model is: 130.7
---------------------------------------------------------------------
The unigram log probability for <s> is -4.261
The unigram log probability for i is -7.268
The unigram log probability for look is -11.075
The unigram log probability for forward is -13.373
The unigram log probability for to is -5.67
The unigram log probability for hearing is -14.02
The unigram log probability for your is -10.408
The unigram log probability for reply is -14.069
The unigram log probability for . is -4.486
The unigram log probability for </s> is -4.261
The log probability of sentence 3 in the Unigram ML Model is: -8.889
The perplexity of sentence 3 in the Unigram ML Model is: 474.1
---------------------------------------------------------------------
```

Part B: Bigram Maximum Likelihood Model

In [17]:
```
# The function bigram_train in training.py automatically generates bigrams out of the training
 data
# and we can just use train_bdict from Question 4,
# which is just a dictionary of all the bigrams and their counts
# thus, the following trains the bigram model
bigram_probs = training.bigram_train(train_bdict, train_list, train_dict)
```

The Parameters for Each Sentence:

$$l_{s1} = \frac{1}{9} log_2(p(he| < s >) * p(was|he) * p(laughed|was) * p(off|laughed) * p(the|off) * p(screen|the) + p(.|screen) * p(< /s > |.)) = \frac{1}{9} log_2(\frac{c(he|<s>)}{c(<s>)} * \frac{c(was|he)}{c(he)} * \frac{c(laughed|was)}{c(was)} * \frac{c(off|laughed)}{c(laughed)} * \frac{c(the|off)}{c(off)} * \frac{c(screen|the)}{c(the)} + \frac{c(.|screen)}{c(screen)} * \frac{c(</s>|.)}{c(.)})$$

$$l_{s2} = \frac{1}{9} log_2(p(there| < s >) * p(was|there) * p(no|was) * p(< unk > |no) * p(behind| < unk >) * p(them|behind) * p(.|them) * p < /s > |.)) = \frac{1}{9} log_2(\frac{c(there|<s>)}{c(<s>)} * \frac{c(was|there)}{c(there)} * \frac{c(no|was)}{c(was)} * \frac{c(<unk>|no)}{c(no)} * \frac{c(behind|<unk>)}{c(<unk>)} * \frac{c(them|behind)}{c(behind)} * \frac{c(.|them)}{c(them)} * \frac{c(</s>|.)}{c(.)})$$

$$l_{s3} = \frac{1}{10} log_2(p(i| < s >) * p(look|i) * p(forward|look) * p(to|forward) * p(hearing|to) * p(your|hearing) * p(reply|your) * p(.|reply) * p(< /s > |.)) = \frac{1}{10} log_2(\frac{c(i|<s>)}{c(<s>)} * \frac{c(look|i)}{c(i)} * \frac{c(forward|look)}{c(look)} * \frac{c(to|forward)}{c(forward)} * \frac{c(hearing|to)}{c(to)} * \frac{c(your|hearing)}{c(hearing)} * \frac{c(reply|your)}{c(your)} * \frac{c(.|reply)}{c(reply)} * \frac{c(</s>|.)}{c(.)})$$

The function in bigram_predict prints all the individual log probabilities of the tokens.
The parameters that yield zero probability are listed below.

In [18]:
```
# I create bigrams of the three sentences
s1_bigrams = training.bigrams(s1_unk)
s2_bigrams = training.bigrams(s2_unk)
s3_bigrams = training.bigrams(s3_unk)
print(s1_bigrams)
print(s2_bigrams)
print(s3_bigrams)
```

```
[('<s>', 'he'), ('he', 'was'), ('was', 'laughed'), ('laughed', 'off'), ('off', 'the'), ('the',
'screen'), ('screen', '.'), ('.', '</s>')]
[('<s>', 'there'), ('there', 'was'), ('was', 'no'), ('no', '<unk>'), ('<unk>', 'behind'), ('be
hind', 'them'), ('them', '.'), ('.', '</s>')]
[('<s>', 'i'), ('i', 'look'), ('look', 'forward'), ('forward', 'to'), ('to', 'hearing'), ('hea
ring', 'your'), ('your', 'reply'), ('reply', '.'), ('.', '</s>')]
```

In [19]:
```python
s1_bpred = testing.bigram_predict(s1_bigrams, bigram_probs, len(s1_unk))
print('The probability of sentence 1 in the Bigram ML Model is: {:.3f}'.format(s1_bpred))
print('The perplexity of sentence 1 in the Bigram ML Model is infinite.')
print('---------------------------------------------------------------------')
s2_bpred = testing.bigram_predict(s2_bigrams, bigram_probs, len(s2_unk))
print('The log probability of sentence 2 in the Bigram ML Model is: {:.3f}'.format(s2_bpred))
print('The perplexity of sentence 2 in the Bigram ML Model is: {:.1f}'.format(perplexity(s2_bpr
ed)))
print('---------------------------------------------------------------------')
s3_bpred = testing.bigram_predict(s3_bigrams, bigram_probs, len(s3_unk))
print('The probability of sentence 3 in the Bigram ML Model is: {:.3f}'.format(s3_bpred))
print('The perplexity of sentence 3 in the Bigram ML Model is infinite.')
print('---------------------------------------------------------------------')
```

```
The bigram log probability for ('<s>', 'he') is -3.608
The bigram log probability for ('he', 'was') is -3.106
This is an unobserved bigram. The probability for ('was', 'laughed') is 0.
This is an unobserved bigram. The probability for ('laughed', 'off') is 0.
The bigram log probability for ('off', 'the') is -2.422
The bigram log probability for ('the', 'screen') is -13.005
This is an unobserved bigram. The probability for ('screen', '.') is 0.
The bigram log probability for ('.', '</s>') is 0.0
The probability of sentence 1 in the Bigram ML Model is: 0.000
The perplexity of sentence 1 in the Bigram ML Model is infinite.
---------------------------------------------------------------------
The bigram log probability for ('<s>', 'there') is -6.1
The bigram log probability for ('there', 'was') is -1.706
The bigram log probability for ('was', 'no') is -5.423
The bigram log probability for ('no', '<unk>') is -5.208
The bigram log probability for ('<unk>', 'behind') is -11.368
The bigram log probability for ('behind', 'them') is -4.053
The bigram log probability for ('them', '.') is -2.567
The bigram log probability for ('.', '</s>') is 0.0
The log probability of sentence 2 in the Bigram ML Model is: -4.047
The perplexity of sentence 2 in the Bigram ML Model is: 16.5
---------------------------------------------------------------------
The bigram log probability for ('<s>', 'i') is -4.827
The bigram log probability for ('i', 'look') is -11.66
The bigram log probability for ('look', 'forward') is -5.852
The bigram log probability for ('forward', 'to') is -1.854
This is an unobserved bigram. The probability for ('to', 'hearing') is 0.
This is an unobserved bigram. The probability for ('hearing', 'your') is 0.
The bigram log probability for ('your', 'reply') is -8.52
The bigram log probability for ('reply', '.') is -2.273
The bigram log probability for ('.', '</s>') is 0.0
The probability of sentence 3 in the Bigram ML Model is: 0.000
The perplexity of sentence 3 in the Bigram ML Model is infinite.
---------------------------------------------------------------------
```

As shown above, in the bigram maximum likelihood model, sentence 1 and sentence 3 have 0 probability.
In sentence one, the following were 0 probability:

- $p('was', 'laughed')$
- $p('laughed', 'off')$
- $p('screen', '.')$

And in sentence three, the following were 0 probability:

- $p('to', 'hearing')$
- $p('hearing', 'your')$

Part C: Bigram Model with Add-One Smoothing

In [20]:
```
# The function add_one_train in training.py just reuses much of bigram_train code
# but adds the proper 1 to the numerator and |V| to the denominator to all probabilities
len_dict = len(train_dict)
add_one_probs = training.add_one_train(train_bdict, train_list, train_dict, len_dict)
```

The Parameters for Each Sentence would be the same as the Bigram ML Model, and the probabilities would have the add-one / |V| :

The biggest difference would be that $p('was','laughed'), p('laughed','off'), p('screen','.')$ in sentence 1 and $p('to','hearing'), p('hearing','your')$ in sentence 3 would not be zero, but instead would be $\frac{1}{(w_{i-1}+|V|)}$.

$l_{s1} = \frac{1}{9}log_2(p(he|<s>) * p(was|he) * p(laughed|was) * p(off|laughed) * p(the|off) * p(screen|the)$
$+ p(.|screen) * p(</s>|.))$
$= \frac{1}{9}log_2(\frac{c(he|<s>)+1}{c(<s>)+|V|} * \frac{c(was|he)+1}{c(he)+|V|} * \frac{1}{c(was)+|V|} * \frac{1}{c(laughed)+|V|} * \frac{c(the|off)+1}{c(off)+|V|} * \frac{c(screen|the)+1}{c(the)+|V|} + \frac{1}{c(screen)+|V|}$
$* \frac{c(</s>|.)}{c(.)})$

$l_{s2} = \frac{1}{9}log_2(p(there|<s>) * p(was|there) * p(no|was) * p(<unk>|no) * p(behind|<unk>)$
$* p(them|behind) * p(.|them) * p</s>|.))$
$= \frac{1}{9}log_2(\frac{c(there|<s>)+1}{c(<s>)+|V|} * \frac{c(was|there)+1}{c(there)+|V|} * \frac{c(no|was)+1}{c(was)+|V|} * \frac{c(<unk>|no)+1}{c(no)+|V|} * \frac{c(behind|<unk>)+1}{c(<unk>)+|V|} * \frac{c(them|behind)+1}{c(behind)+|V|}$
$* \frac{c(.|them)+1}{c(them)+|V|} * \frac{c(</s>|.)+1}{c(.)+|V|})$

$l_{s3} = \frac{1}{10}log_2(p(i|<s>) * p(look|i) * p(forward|look) * p(to|forward) * p(hearing|to) * p(your|hearing)$
$* p(reply|your) * p(.|reply) * p(</s>|.))$
$= \frac{1}{10}log_2(\frac{c(i|<s>)+1}{c(<s>)+|V|} * \frac{c(look|i)+1}{c(i)+|V|} * \frac{c(forward|look)+1}{c(look)+|V|} * \frac{c(to|forward)+1}{c(forward)+|V|} * \frac{1}{c(to)+|V|} * \frac{1}{c(hearing)+|V|} * \frac{c(reply|your)+1}{c(your)+|V|}$
$* \frac{c(.|reply)+1}{c(reply)+|V|} * \frac{c(</s>|.)+1}{c(.)+|V|})$

In [21]:
```
s1_aopred = testing.add_one_predict(s1_bigrams, add_one_probs, train_dict, len(s1_unk))
print('The log probability of sentence 1 in the Bigram Add-One Model is: {:.3f}'.format(s1_aopr
ed))
print('The perplexity of sentence 1 in the Bigram Add-One Model is: {:.1f}'.format(perplexity(s
1_aopred)))
print('---------------------------------------------------------------------')
s2_aopred = testing.add_one_predict(s2_bigrams, add_one_probs, train_dict, len(s2_unk))
print('The log probability of sentence 2 in the Bigram Add-One Model is: {:.3f}'.format(s2_aopr
ed))
print('The perplexity of sentence 2 in the Bigram Add-One Model is: {:.1f}'.format(perplexity(s
2_aopred)))
print('---------------------------------------------------------------------')
s3_aopred = testing.add_one_predict(s3_bigrams, add_one_probs, train_dict, len(s3_unk))
print('The log probability of sentence 3 in the Bigram Add-One Model is: {:.3f}'.format(s3_aopr
ed))
print('The perplexity of sentence 3 in the Bigram Add-One Model is: {:.1f}'.format(perplexity(s
3_aopred)))
print('---------------------------------------------------------------------')
```

```
The add-one log probability for ('<s>', 'he') is -4.265
The add-one log probability for ('he', 'was') is -4.921
The add-one log probability for ('was', 'laughed') is -14.301
The add-one log probability for ('laughed', 'off') is -13.88
The add-one log probability for ('off', 'the') is -7.666
The add-one log probability for ('the', 'screen') is -13.276
The add-one log probability for ('screen', '.') is -13.877
The add-one log probability for ('.', '</s>') is -0.745
The log probability of sentence 1 in the Bigram Add-One Model is: -8.103
The perplexity of sentence 1 in the Bigram Add-One Model is: 275.0
---------------------------------------------------------------------
The add-one log probability for ('<s>', 'there') is -6.755
The add-one log probability for ('there', 'was') is -5.413
The add-one log probability for ('was', 'no') is -7.382
The add-one log probability for ('no', '<unk>') is -9.161
The add-one log probability for ('<unk>', 'behind') is -12.201
The add-one log probability for ('behind', 'them') is -10.432
The add-one log probability for ('them', '.') is -6.843
The add-one log probability for ('.', '</s>') is -0.745
The log probability of sentence 2 in the Bigram Add-One Model is: -6.548
The perplexity of sentence 2 in the Bigram Add-One Model is: 93.6
---------------------------------------------------------------------
The add-one log probability for ('<s>', 'i') is -5.484
The add-one log probability for ('i', 'look') is -13.157
The add-one log probability for ('look', 'forward') is -11.576
The add-one log probability for ('forward', 'to') is -10.073
The add-one log probability for ('to', 'hearing') is -14.599
The add-one log probability for ('hearing', 'your') is -13.879
The add-one log probability for ('your', 'reply') is -12.91
The add-one log probability for ('reply', '.') is -11.071
The add-one log probability for ('.', '</s>') is -0.745
The log probability of sentence 3 in the Bigram Add-One Model is: -9.349
The perplexity of sentence 3 in the Bigram Add-One Model is: 652.3
---------------------------------------------------------------------
```

Part D: Bigram Model with Discounting and Katz Backoff

In [22]:
```
# First, I build set A of the Katz backoff
# Please see training.py for the code
katz_unigrams, bigrams_A = training.build_set_A(train_list)
```

```
In [23]: # I do not actually build set B, but instead make an assumption that
         # the probability mass of set B is the count of all tokens in set A that appear in the corpus
         # subtracted from all the tokens in the corpus minus the word itself
         # Please see training.py for the code
         katz_probs = training.katz_backoff(train_list, katz_unigrams, bigrams_A, len(train_list))
```

The Parameters for Each Sentence would also be the same as the Bigram ML Model:

The big difference lies in calculating the probabilities, which differ for observed bigrams (which gives back the discounted probabilities) vs. unseen bigrams (which gives back a proportion of the leftover probabilities $alpha$).
However, the parameters for calculating the probabilities of each parameter requires the unigram counts for the tokens.

$$\alpha = 1 - \sum firstwordbigrams.in.set.A \frac{disc.bigram.probs}{unigram.count(first.word)}$$

$$l_{s1} = \frac{1}{9} log_2(p(he| < s >) * p(was|he) * p(laughed|was) * p(off|laughed) * p(the|off) * p(screen|the)$$
$$+ p(.|screen) * p(< /s > |.))$$
$$= \frac{1}{9} log_2(\frac{c(he|<s>)-0.5}{c(<s>)} * \frac{c(was|he)-0.5}{c(he)} * \alpha(was)\frac{c(laughed)}{c(set.B)} * \alpha(laughed)\frac{c(was)}{\sum c(set.B)} * \frac{c(the|off)+1}{c(off)+|V|} * \frac{c(screen|the)}{c(the)}$$
$$+ \alpha(screen)\frac{c(.)}{c(set.B)} * \frac{c(</s>|.)}{c(.)})$$

$$l_{s2} = \frac{1}{9} log_2(p(there| < s >) * p(was|there) * p(no|was) * p(< unk > |no) * p(behind| < unk >)$$
$$* p(them|behind) * p(.|them) * p < /s > |.))$$
$$= \frac{1}{9} log_2(\frac{c(there|<s>)+1}{c(<s>)+|V|} * \frac{c(was|there)+1}{c(there)+|V|} * \frac{c(no|was)+1}{c(was)+|V|} * \frac{c(<unk>|no)+1}{c(no)+|V|} * \frac{c(behind|<unk>)+1}{c(<unk>)+|V|} * \frac{c(them|behind)+1}{c(behind)+|V|}$$
$$* \frac{c(.|them)+1}{c(them)+|V|} * \frac{c(</s>|.)+1}{c(.)+|V|})$$

$$l_{s3} = \frac{1}{10} log_2(p(i| < s >) * p(look|i) * p(forward|look) * p(to|forward) * p(hearing|to) * p(your|hearing)$$
$$* p(reply|your) * p(.|reply) * p(< /s > |.))$$
$$= \frac{1}{10} log_2(\frac{c(i|<s>)+1}{c(<s>)+|V|} * \frac{c(look|i)+1}{c(i)+|V|} * \frac{c(forward|look)+1}{c(look)+|V|} * \frac{c(to|forward)+1}{c(forward)+|V|} * \alpha(to)\frac{c(hearing)}{c(set.B)} * \alpha(hearing)\frac{c(your)}{c(set.B)}$$
$$* \frac{c(reply|your)+1}{c(your)+|V|} * \frac{c(.|reply)+1}{c(reply)+|V|} * \frac{c(</s>|.)+1}{c(.)+|V|})$$

In [24]:

```
s1_kpred = testing.katz_predict(s1_bigrams, katz_probs, katz_unigrams, bigrams_A, len(train_lis
t), len(s1_unk))
print('The log probability of sentence 1 in the Bigram Katz Backoff Model is: {:.3f}'.format(s1
_kpred))
print('The perplexity of sentence 1 in the Bigram Katz Backoff Model is: {:.1f}'.format(perplex
ity(s1_kpred)))
print('---------------------------------------------------------------------')
s2_kpred = testing.katz_predict(s2_bigrams, katz_probs, katz_unigrams, bigrams_A, len(train_lis
t), len(s1_unk))
print('The log probability of sentence 2 in the Bigram Katz Backoff Model is: {:.3f}'.format(s2
_kpred))
print('The perplexity of sentence 2 in the Bigram Katz Backoff Model is: {:.1f}'.format(perplex
ity(s2_kpred)))
print('---------------------------------------------------------------------')
s3_kpred = testing.katz_predict(s3_bigrams, katz_probs, katz_unigrams, bigrams_A, len(train_lis
t), len(s1_unk))
print('The log probability of sentence 3 in the Bigram Katz Backoff Model is: {:.3f}'.format(s3
_kpred))
print('The perplexity of sentence 3 in the Bigram Katz Backoff Model is: {:.1f}'.format(perplex
ity(s3_kpred)))
print('---------------------------------------------------------------------')
```

```
The Katz log probability for ('<s>', 'he') in Set A is -3.608
The Katz log probability for ('he', 'was') in Set A is -3.107
The Katz log probability for ('was', 'laughed') in set B is -11.363
The Katz log probability for ('laughed', 'off') in set B is -8.805
The Katz log probability for ('off', 'the') in Set A is -2.432
The Katz log probability for ('the', 'screen') in Set A is -13.268
The Katz log probability for ('screen', '.') in set B is -4.113
The Katz log probability for ('.', '</s>') in Set A is -0.0
The log probability of sentence 1 in the Bigram Katz Backoff Model is: -5.188
The perplexity of sentence 1 in the Bigram Katz Backoff Model is: 36.5
---------------------------------------------------------------------
The Katz log probability for ('<s>', 'there') in Set A is -6.102
The Katz log probability for ('there', 'was') in Set A is -1.708
The Katz log probability for ('was', 'no') in Set A is -5.429
The Katz log probability for ('no', '<unk>') in Set A is -5.235
The Katz log probability for ('<unk>', 'behind') in Set A is -11.52
The Katz log probability for ('behind', 'them') in Set A is -4.127
The Katz log probability for ('them', '.') in Set A is -2.573
The Katz log probability for ('.', '</s>') in Set A is -0.0
The log probability of sentence 2 in the Bigram Katz Backoff Model is: -4.077
The perplexity of sentence 2 in the Bigram Katz Backoff Model is: 16.9
---------------------------------------------------------------------
The Katz log probability for ('<s>', 'i') in Set A is -4.828
The Katz log probability for ('i', 'look') in Set A is -12.66
The Katz log probability for ('look', 'forward') in Set A is -6.044
The Katz log probability for ('forward', 'to') in Set A is -1.911
The Katz log probability for ('to', 'hearing') in set B is -12.094
The Katz log probability for ('hearing', 'your') in set B is -8.364
The Katz log probability for ('your', 'reply') in Set A is -9.52
The Katz log probability for ('reply', '.') in Set A is -2.399
The Katz log probability for ('.', '</s>') in Set A is -0.0
The log probability of sentence 3 in the Bigram Katz Backoff Model is: -6.424
The perplexity of sentence 3 in the Bigram Katz Backoff Model is: 85.9
---------------------------------------------------------------------
```

Summary

Log Probabilities and Perplexities by Model

Sentence 1

Model	Log Probability	Perplexity
Unigram Maximum Likelihood	-7.681	205.2
Bigram Maximum Likelihood	-Infinite	Infinite
Bigram with Add-One Smoothing	-8.103	275.0
Bigram with Discounting and Katz Backoff	-5.188	36.5

Sentence 2

Model	Log Probability	Perplexity
Unigram Maximum Likelihood	-7.030	130.7
Bigram Maximum Likelihood	-4.047	16.5
Bigram with Add-One Smoothing	-6.548	93.6
Bigram with Discounting and Katz Backoff	-4.077	16.9

Sentence 3

Model	Log Probability	Perplexity
Unigram Maximum Likelihood	-8.889	474.1
Bigram Maximum Likelihood	-Infinite	Infinite
Bigram with Add-One Smoothing	-9.349	652.3
Bigram with Discounting and Katz Backoff	-6.424	85.9

Question 7:

Compute the perplexities of the entire test corpora, separately for the brown-test.txt and learner-test.txt under each of the models. Discuss the differences in the results you obtained.

In [25]:
```
# For easier viewing, I recreate the steps above to preprocess and train all the models again
# 1. Training data preprocessing
train_fp = 'data/brown-train.txt'
train_list = preprocessing.preprocess_train(train_fp)
train_dict = preprocessing.build_dict(train_list)

# Bigrams list and dictionary with counts for training data
train_bigrams = training.bigrams(train_list)
train_bdict = training.bigram_dict(train_list)
```

In [26]:
```
# 2. Test data preprocessing
test_fp = 'data/brown-test.txt'
learner_fp = 'data/learner-test.txt'

test_list = preprocessing.preprocess_test(test_fp, train_dict)
learner_list = preprocessing.preprocess_test(learner_fp, train_dict)

# Bigrams lists and dictionaries with counts
test_bigrams = training.bigrams(test_list)
learner_bigrams = training.bigrams(learner_list)

test_bdict = training.bigram_dict(test_list)
learner_bdict = training.bigram_dict(learner_list)
```

In [27]:
```
# Train Unigram ML Model
# to create dictionary of tokens and probabilities
unigram_probs = training.unigram_train(train_list, train_dict)
```

In [28]:
```
# Train Bigram ML Model
# to create dictionary of bigrams and probabilities
bigram_probs = training.bigram_train(train_bdict, train_list, train_dict)
```

In [29]:
```
# Train Bigram with Add-One Smoothing
add_one_probs = training.add_one_train(train_bdict, train_list, train_dict, len(train_dict))
```

In [30]:
```
# Train Bigram with Discounting and Katz Backoff
katz_unigrams, bigrams_A = training.build_set_A(train_list)
katz_probs = training.katz_backoff(train_list, katz_unigrams, bigrams_A, len(train_list))
```

Part A: brown-test.txt

In [31]:
```
# Unigram ML Model on brown-test.txt
test_unigram = testing.unigram_predict_no_print(test_list, unigram_probs)
test_uni_perplexity = perplexity(test_unigram)
print('The perplexity of the Unigram ML Model on brown-test.txt: {:.1f}'.format(test_uni_perple
xity))
```

```
The perplexity of the Unigram ML Model on brown-test.txt: 365.1
```

In [32]:
```
# Bigram ML Model on brown-test.txt
test_bigram = testing.bigram_predict_file(test_fp, bigram_probs, train_dict, len(test_list))
test_bi_perplexity = perplexity(test_bigram)
print('The perplexity of the Bigram ML Model on brown-test.txt:',test_bi_perplexity)
```

```
The perplexity of the Bigram ML Model on brown-test.txt: Too Large or Infinite
```

In [33]:
```
# Bigram Model with Add-One Smoothing on brown-test.txt
test_add_one = testing.add_one_predict_no_print(test_bigrams, add_one_probs, train_dict, len(te
st_list))
test_ao_perplexity = perplexity(test_add_one)
print('The perplexity of the Bigram Add-One Smoothing Model on brown-test.txt: {:.1f}'.format(t
est_ao_perplexity))
```

```
The perplexity of the Bigram Add-One Smoothing Model on brown-test.txt: 668.7
```

In [34]:
```
# Bigram Model with Discounting and Katz Backoff
test_katz = testing.katz_predict_no_print(test_bigrams, katz_probs, katz_unigrams, bigrams_A,
                                          len(train_list), len(test_list))
test_katz_perplexity = perplexity(test_katz)
print('The perplexity of the Bigram Katz Backoff Model on brown-test.txt: {:.1f}'.format(test_k
atz_perplexity))
```

```
The perplexity of the Bigram Katz Backoff Model on brown-test.txt: 78.8
```

Part B: learner-test.txt

In [35]:
```
# Unigram ML Model on learner-test.txt
learner_unigram = testing.unigram_predict_no_print(learner_list, unigram_probs)
learner_uni_perplexity = perplexity(learner_unigram)
print('The perplexity of the Unigram ML Model on learner-test.txt: {:.1f}'.format(learner_uni_p
erplexity))
```

```
The perplexity of the Unigram ML Model on learner-test.txt: 409.6
```

In [36]:
```
# Bigram ML Model on learner-test.txt
learner_bigram = testing.bigram_predict_file(learner_fp, bigram_probs, train_dict, len(learner_
list))
learner_bi_perplexity = perplexity(learner_bigram)
print('The perplexity of the Bigram ML Model on learner-test.txt: ', learner_bi_perplexity)
```

```
The perplexity of the Bigram ML Model on learner-test.txt:  Too Large or Infinite
```

In [37]:
```
# Bigram Model with Add-One Smoothing on brown-test.txt
learner_add_one = testing.add_one_predict_no_print(learner_bigrams, add_one_probs, train_dict,
len(learner_list))
learner_ao_perplexity = perplexity(learner_add_one)
print('The perplexity of the Bigram Add-One Smoothing Model on learner-test.txt: {:.1f}'.format
(learner_ao_perplexity))
```

```
The perplexity of the Bigram Add-One Smoothing Model on learner-test.txt: 845.5
```

In [38]:
```
# Bigram Model with Discounting and Katz Backoff
learner_katz = testing.katz_predict_no_print(learner_bigrams, katz_probs, katz_unigrams, bigram
s_A,
                                             len(train_list), len(learner_list))
learner_katz_perplexity = perplexity(learner_katz)
print('The perplexity of the Bigram Katz Backoff Model on learner-test.txt: {:.1f}'.format(lear
ner_katz_perplexity))
```

```
The perplexity of the Bigram Katz Backoff Model on learner-test.txt: 88.8
```

Summary

Perplexities by Model for Two Test Data Sets

Model	brown-test.txt	learner-test.txt
Unigram Maximum Likelihood	365.1	409.6
Bigram Maximum Likelihood	Infinite	Infinite
Bigram with Add-One Smoothing	668.7	845.5
Bigram with Discounting and Katz Backoff	78.8	88.8

In general, the brown-test had lower perplexities (i.e. was better modeled) than the learner-test. I believe three factors may help explain this:

1. The brown-test comes from the same source, and barring any strange data design choices, natural data from the same source should have more similar patterns and structure than if it were otherwise.
2. The learner-test contains many letters, as opposed to the brown-test, which has many spoken quotations. The differences between spoken language and written language can also account for the difference in perplexities. As the brown-train has more spoken language, the brown-test would have lower perplexity and vice versa for learner-test. This is observed in Questions 5 and 6, where the training set predicted sentence 3 poorly compared to the other two sentences.
3. Finally, because the learner-test is written by non-native speakers of English, you can find differences not normally found in native speech patterns: (e.g. "it should started"). This would raise the perplexity of the learner-test as there would be, on average, more unseen tokens and bigrams.