

Assignment # 1

CS 2731

Joseph Knittel

October 03, 2013

1 Introduction

Making observations about the current state of a system and using that knowledge to predict future states is a critical process in any scientific investigation.

In the realm of Natural Language Processing, N-gram models are frequently used to predict the next word in a sentence given a set of previous words.

However, the utility of these probabilistic models is not limited strictly to the scale of words. N-gram models can be built at the character level to describe the actual structure of the words themselves or they can even be built at the sentence level to get an idea of how paragraphs are composed in a document.

This paper describes the process of using character-based N-gram models to determine the most likely language in which a test document was written. Models are constructed using English, German, and Spanish training documents and the perplexity values observed from each model run on the test document are compared to give a reliable prediction of the test document's language.

2 Program

A program was created in Python to build a suite of unsmoothed and smoothed N-gram models for English, German, and Spanish.

These models could then be used to compute perplexity values for a test document which, in turn, could be used to predict the test document's language.

2.1 Building the N-Gram Models

The “N” in N-gram signifies the length of the sub-components used to analyze the structure of the feature being investigated.

In this paper, for instance, a 2-gram (commonly referred to as a “bigram”) would be a model that uses 2-character chunks of data to describe the structure of words in a document.

We start by constructing basic unigram (1-gram), bigram (2-gram), and trigram (3-gram) models for each language and then adjust them with smoothing techniques called Laplace smoothing and interpolation.

2.1.1 Basic (Unsmoothed) N-gram Models

2.1.1.1 N-gram Counts

The first step in constructing these basic N-gram models is to count all of the times each N-gram occurs in the training documents.

This was done by scanning through each training document one character at a time and storing (if it exists) the unigram, bigram, and trigram ending in the current character in a dictionary.

If the N-gram is already in a dictionary, then 1 is added to the value of the N-gram's key in the dictionary.

Example: (current character = ‘g’)

```
uni['g'] = uni.get('g') + 1
```

Once the training documents have been scanned, the program adds in all N-grams that were not seen in the training documents into the dictionaries as 0 counts.

Code:

```
for i in uni:
    for j in uni:
        if not bi.has_key(i + j): bi.update({i+j:0})
```

The above code is used to store 0 counts for every possible bigram that was not observed in the training documents.

2.1.1.2 Computing MLE Probabilities

Once N-gram counts are logged, they can be used to approximate the probability of a specific character occurring given a sequence of preceding characters.

The most basic estimate of this probability is the Maximum Likelihood Estimate (MLE).

The MLE states that the probability of a given future character occurring is just the count of that specific character preceded by some (n-1) characters divided by the count of all (n-1)-character strings that are equal to the first (n-1) characters in the numerator.

This can be more clearly understood in equation form:

$$P(u_n | u_{n-N+1}^{n-1}) = \frac{C(u_{n-N+1}^{n-1} u_n)}{C(u_{n-N+1}^{n-1})} \quad (1)$$

where C stands for the N-gram count and u stands for unit (in our case, character)

The previously recorded N-gram counts were used in this calculation to compute unsmoothed unigram, bigram, and trigram models of the training documents.

2.1.2 Smoothed N-gram Models

Unsmoothed N-gram models give the most basic approximation of the probability that a specific character might occur given some preceding characters. However, significant issues arise when an N-gram occurs in the test document and not in a training document. Namely, division by zero occurs which yields an undefined probability for the given character.

To avoid this situation, smoothing techniques are used to re-distribute probability mass from more commonly occurring N-grams to the N-grams that were not observed in the training data.

2.1.2.1 Laplace Smoothing

The first smoothing method used was developed hundreds of years ago by the famous mathematician Pierre-Simon Laplace.

He suggested that we can update the probability from Equation 1 by just adding 1 to each of the N-gram counts and then dividing the result by (the previous denominator + V) where V is, in our case, the total unique characters.

This can be seen in Equation 2 below:

$$P_{Laplace}(u_i) = \frac{c_i + 1}{N + V} \quad (2)$$

Observe the code below where Laplace smoothing was implemented in the program by adding 1 to the unsmoothed English trigram count and dividing by (the bigram count of the first two characters + the total unique English characters).

Code:

```
for i in uniENG:
    for j in uniENG:
        for k in uniENG:
            laplaceENG.update({i+j+k: \
                1. * (triENG[i+j+k] + 1) / (biENG[i+j] + len(uniENG))})
```

2.1.2.2 Interpolation

Another technique used to avoid the undefined probabilities that might arise with unsmoothed N-gram models is called interpolation.

Interpolation uses the key insight that although some higher-order N-gram counts might be zero, there are lower-order N-grams with non-zero counts that we can use in a weighted sum to approximate the probability given in Equation 1.

For the trigram probability, this can be stated as:

$$\begin{aligned}\hat{P}(u_n|u_{n-2}u_{n-1}) &= \lambda_1 P(u_n|u_{n-2}u_{n-1}) \\ &\quad + \lambda_2 P(u_n|u_{n-1}) \\ &\quad + \lambda_3 P(u_n)\end{aligned}\tag{3}$$

where all λ s sum to 1:

$$\sum_{i=1}^3 \lambda_i = 1$$

The code below demonstrates how the interpolation smoothing method was implemented in the program for a Spanish trigram model. More sophisticated techniques optimize the values of the λ s, but for this paper each λ was simply given the value $\frac{1}{3}$.

Code:

```
for i in uniSPA:
    for j in uniSPA:
        for k in uniSPA:
            interpolatedSPA.update({i+j+k: (lambda1 * unsmoothTriSPA[i+j+k]
                + lambda2 * unsmoothBiSPA[j+k]
                + lambda3 * unsmoothUniSPA[k])})
```

2.2 Perplexity

Once the unsmoothed and smoothed N-gram models were built, they could be put to work on a test document to predict the probability that a given sequence of characters might occur in a document.

The N-gram models were evaluated by how well they could guess the next character given a sequence of preceding characters.

In Natural Language Processing, this value is given by the perplexity of the character on a given N-gram model:

$$PP(u) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(u_i|u_1 \dots u_{i-1})}} \quad (4)$$

The perplexity of a given character using an N-gram model is actually a weighted branching factor, therefore the model with the lowest perplexity value is the one which can best guess which character will come next given a sequence of preceding characters.

Perplexity was implemented in the program by scanning through the characters of a test document one by one and updating the perplexity value according to Equation 4.

The code below describes how the perplexity was calculated for the Laplace-smoothed German trigram model.

Code:

```
LGpp = LGpp * pow(laplaceGER[curTri],norm)
```

where this line of code was executed on each successive trigram in the test document and $\text{norm} = \frac{1}{N}$ where N is the number of characters in the test document.

2.3 Output

Given the perplexity values of each trigram model, the program should be able to evaluate which language the test document is written in.

To do this, I decided that smoothed models would be more accurate and therefore the program's language selection should thus be determined strictly based on the smoothed models' perplexity values.

I figured that both the Laplace and interpolation methods should be weighted evenly, therefore the chosen language is based on the minimum of the sum of Laplace and interpolation perplexities for each language.

This was implemented with the following code.

Code:

```
smoothENG = LEpp + IEpp
smoothGER = LGpp + IGpp
smoothSPA = LSpp + ISpp

if smoothENG < smoothGER and smoothENG < smoothSPA:
    print "The document is probably written in English!"
elif smoothGER < smoothENG and smoothGER < smoothSPA:
    print "The document is probably written in German!"
elif smoothSPA < smoothENG and smoothSPA < smoothGER:
    print "The document is probably written in Spanish!"
else:
    print "The models were inconclusive in determining the language of the document."
```

2.4 Important Design Decision

It is worth noting that the program was written exclusively for usage in CS 2731 class.

Therefore, the code was written to run with little input from the user.

“python ngrams.py test” will do the following:

1. Build N-gram models, and
2. Evaluate the models to determine the language of the test document

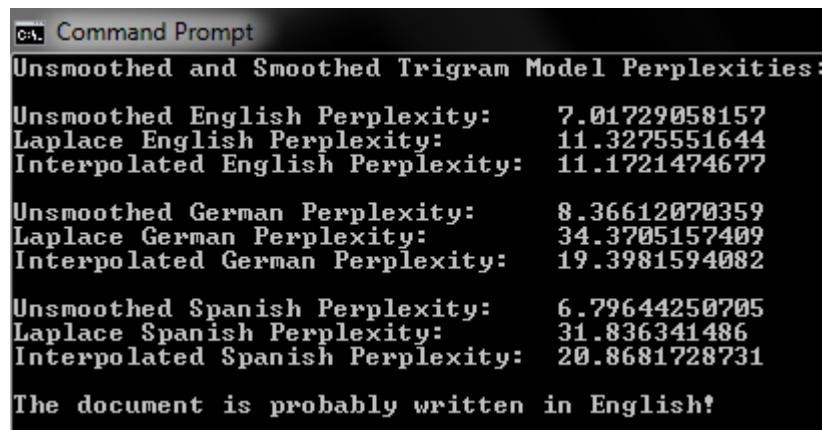
In the real world, it would make far more sense to have the program only build the N-grams models once and output the models to files where they then could be used to evaluate any future test document, saving the user considerable time.

Since this program will likely only ever be used to determine the language of the provided test document, I found it to be reasonable to not generate additional files and both build the N-gram models and evaluate the test document all in one run.

3 Test

With a suite of unsmoothed and smoothed trigram models built for English, German, and Spanish languages, I decided to test their efficacy in predicting the source language of a document.

The figure below shows the program’s output after examining the test document:



```
Command Prompt
Unsmoothed and Smoothed Trigram Model Perplexities:
Unsmoothed English Perplexity: 7.01729058157
Laplace English Perplexity: 11.3275551644
Interpolated English Perplexity: 11.1721474677
Unsmoothed German Perplexity: 8.36612070359
Laplace German Perplexity: 34.3705157409
Interpolated German Perplexity: 19.3981594082
Unsmoothed Spanish Perplexity: 6.79644250705
Laplace Spanish Perplexity: 31.836341486
Interpolated Spanish Perplexity: 20.8681728731
The document is probably written in English!
```

4 Critical Analysis

So, a program was created to build N-gram models for different languages and then the N-gram models were used to evaluate the perplexity of a test document.

Using these perplexities, it was determined that the test document was written in English, but how can we be sure?

Some questions come to mind:

- (1) Why are these perplexity values a good indicator of the test document’s language?
- (2) The program outputs nine perplexity values. Which ones are most valid?
- (3) Why does the program not consider unigram models and bigram models when determining the test document’s language?

4.1 Perplexity as Indicator of Test Document's Language

As stated in Section 2.2, perplexity, as it relates to N-gram models, is a measure of the average branching factor from a given sequence of preceding units to a following unit.

In the case of this investigation, perplexity = 1 would represent that the given model always knows exactly which character should follow a set of preceding characters.

Conversely, perplexity = 30 means that on average the model would predict any one of 30 possible characters to be the next one given a set of preceding characters. Clearly, this a model with perplexity = 30 is not as good at predicting future characters as a model with perplexity = 1!

So perplexity in this investigation is a good way to see how well a model can predict the next character in a document given a set of preceding characters, but what does this have to do with the language the document was written in?

Well, since the models were based on different languages (English, German, and Spanish), then their perplexity (i.e. their ability to predict a following character) is essentially the same thing as their ability to predict the structure of the words in the test document.

Therefore, if perplexity for a given model is low, then the model's language is probably the same as the test document's language!

4.2 Unsmoothed vs. Smoothed N-gram Models

A quick examination of the figure in Section 3 shows that perhaps the best model (lowest perplexity value) was the unsmoothed Spanish trigram model.

Why, then, does the program output that the test document is "probably written in English"?

The reasoning behind this conclusion is based on the fact that when training data is limited or when the model is particularly sparse, unsmoothed N-gram models do not do a very good job at predicting future units, in general.

Seeing as the Spanish training data has about 169,950 characters and the test document has about 13,801 characters, it can be assumed that the training data is somewhat limited in this case.

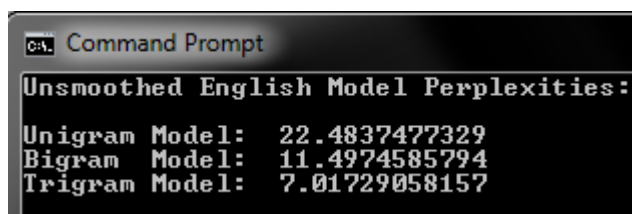
Additionally, Figure 1(a) shows probability values for all trigrams starting with the bigram 'th' using the unsmoothed English trigram model. Although the model in question is the unsmoothed Spanish trigram model, it can easily be checked that a similarly high percentage of all possible Spanish trigrams have a zero probability and hence are poor predictors of the actual language of the test document.

These findings as well the fact that smoothed N-gram models are better models than unsmoothed models in general led me to decided to use the sum of the perplexity values of the Laplace and interpolated models as the predictor of the test document's language.

4.3 Lower-Order N-gram Models vs. Higher-Order N-gram Models

Finally, one might wonder why only the selected trigram models were used to determine the language in which the test document was written.

The figure below indicates that unigram and bigram models produced significantly higher perplexity values than the trigram model:



```
C:\> Command Prompt
Unsmoothed English Model Perplexities:
Unigram Model: 22.4837477329
Bigram Model: 11.4974585794
Trigram Model: 7.01729058157
```

This is in accordance with what the textbook says about lower-order N-gram models being less precise and it makes sense too.

The larger the N-gram, the more previous information there is to be used to predict future characters and hence the better the model of the language's structure.

Therefore, only trigram models were considered in the evaluation of the test document's language.

5 Afterward

The following section, Supplementary Figures, contains figures showing the requested printouts for the assignment.

Figure 1 displays the probabilities for all three trigram models of all trigrams starting with 'th'

Figure 2 demonstrates that the three trigram models for English do indeed sum to 1 and hence are true probability distributions

6 Supplementary Figures

th +	= 0.0	th +	= 0.000273149412729	th +	= 0.00212421112109
th + i	= 0.0	th + i	= 0.000273149412729	th + i	= 6.3726336328e-06
th + "	= 0.0579182988248	th + "	= 0.0568150778476	th + "	= 0.104370013259
th + "	= 0.0	th + "	= 0.000273149412729	th + "	= 2.54905334531e-05
th + r	= 0.0	th + r	= 0.000273149412729	th + r	= 4.24842224219e-06
th + <	= 0.0	th + <	= 0.000273149412729	th + <	= 7.43473892383e-05
th + i	= 0.0	th + i	= 0.000273149412729	th + i	= 2.12421112109e-06
th + ,	= 0.00167879127029	th + ,	= 0.0019120458891	th + ,	= 0.00582462113063
th + .	= 0.00195858981533	th + .	= 0.00218519530183	th + .	= 0.003412674349
th + 0	= 0.0	th + 0	= 0.000273149412729	th + 0	= 0.000308010612559
th + 1	= 0.0	th + 1	= 0.000273149412729	th + 1	= 1.06210556055e-05
th + 2	= 0.0	th + 2	= 0.000273149412729	th + 2	= 0.00019330321202
th + 4	= 0.0	th + 4	= 0.000273149412729	th + 4	= 5.09810669063e-05
th + 6	= 0.0	th + 6	= 0.000273149412729	th + 6	= 6.16021225117e-05
th + 8	= 0.0	th + 8	= 0.000273149412729	th + 8	= 6.58505447539e-05
th + :	= 0.0	th + :	= 0.000273149412729	th + :	= 4.03600113008e-05
th + !	= 0.0	th + !	= 0.000273149412729	th + !	= 3.39873773375e-05
th + B	= 0.0	th + B	= 0.000273149412729	th + B	= 0.00012100033902
th + D	= 0.0	th + D	= 0.000273149412729	th + D	= 8.9216867086e-05
th + F	= 0.0	th + F	= 0.000273149412729	th + F	= 0.000269774812379
th + H	= 0.0	th + H	= 0.000273149412729	th + H	= 0.000131701089508
th + J	= 0.0	th + J	= 0.000273149412729	th + J	= 3.18631668164e-05
th + L	= 0.0	th + L	= 0.000273149412729	th + L	= 5.73537002696e-05
th + N	= 0.0	th + N	= 0.000273149412729	th + N	= 7.43473892383e-05
th + P	= 0.0	th + P	= 0.000273149412729	th + P	= 0.000573537002696
th + R	= 0.0	th + R	= 0.000273149412729	th + R	= 0.000104086344934
th + T	= 0.0	th + T	= 0.000273149412729	th + T	= 0.000628766491844
th + U	= 0.0	th + U	= 0.000273149412729	th + U	= 3.356321332e-05
th + X	= 0.0	th + X	= 0.000273149412729	th + X	= 4.24842224219e-06
th + Z	= 0.0	th + Z	= 0.000273149412729	th + Z	= 2.12421112109e-06
th + b	= 0.0	th + b	= 0.000273149412729	th + b	= 0.00326491249312
th + d	= 0.0011191418019	th + d	= 0.00136574706364	th + d	= 0.00742448192402
th + f	= 0.0	th + f	= 0.000273149412729	th + f	= 0.0054273594144
th + h	= 0.0	th + h	= 0.000273149412729	th + h	= 0.0123948102942
th + j	= 0.0	th + j	= 0.000273149412729	th + j	= 0.000446122738007
th + l	= 0.000559597090095	th + l	= 0.000819448238186	th + l	= 0.0110227093307
th + n	= 0.0	th + n	= 0.000273149412729	th + n	= 0.0203440307376
th + p	= 0.0	th + p	= 0.000273149412729	th + p	= 0.00054681867521
th + r	= 0.00615556799105	th + r	= 0.00628243649276	th + r	= 0.0218044607143
th + t	= 0.0	th + t	= 0.000273149412729	th + t	= 0.0297394428594
th + v	= 0.0	th + v	= 0.000273149412729	th + v	= 0.00282095236881
th + x	= 0.0	th + x	= 0.000273149412729	th + x	= 0.000543798047
th + z	= 0.0	th + z	= 0.000273149412729	th + z	= 9.13410782071e-05
th + :	= 0.0	th + :	= 0.000273149412729	th + :	= 8.49684448438e-06
th + !	= 0.0	th + !	= 0.000273149412729	th + !	= 2.9739556953e-05
th + B	= 0.0	th + B	= 0.000273149412729	th + B	= 1.06210556055e-05
th + D	= 0.0	th + D	= 0.000273149412729	th + D	= 0.00027831005944
th + F	= 0.0	th + F	= 0.000273149412729	th + F	= 8.49684448438e-05
th + H	= 0.0	th + H	= 0.000273149412729	th + H	= 0.00052896697173
th + J	= 0.0	th + J	= 0.000273149412729	th + J	= 7.22231781172e-05
th + L	= 0.0	th + L	= 0.000273149412729	th + L	= 0.000293141134711
th + N	= 0.0	th + N	= 0.000273149412729	th + N	= 4.03600113008e-05
th + P	= 0.0	th + P	= 0.000273149412729	th + P	= 9.9837226915e-05
th + R	= 0.0	th + R	= 0.000273149412729	th + R	= 4.24842224219e-05
th + T	= 0.0	th + T	= 0.000273149412729	th + T	= 0.000354743257223
th + U	= 0.0	th + U	= 0.000273149412729	th + U	= 3.61115398586e-05
th + X	= 0.0	th + X	= 0.000273149412729	th + X	= 6.3726336328e-06
th + Z	= 0.0	th + Z	= 0.000273149412729	th + Z	= 5.09810669063e-05
th + b	= 0.0	th + b	= 0.000273149412729	th + b	= 0.000259153756774
th + d	= 0.0	th + d	= 0.000273149412729	th + d	= 0.000760467581352
th + f	= 0.0	th + f	= 0.000273149412729	th + f	= 6.3726336328e-06
th + h	= 0.0	th + h	= 0.000273149412729	th + h	= 0.00052255935789
th + j	= 0.0	th + j	= 0.000273149412729	th + j	= 0.000152943200719
th + l	= 0.0	th + l	= 0.000273149412729	th + l	= 0.00098988238243
th + n	= 0.0	th + n	= 0.000273149412729	th + n	= 4.4608433543e-05
th + p	= 0.0	th + p	= 0.000273149412729	th + p	= 0.000696741247719
th + r	= 0.0	th + r	= 0.000273149412729	th + r	= 0.000123284245023
th + t	= 0.0	th + t	= 0.000273149412729	th + t	= 8.49684448438e-06
th + v	= 0.0	th + v	= 0.000273149412729	th + v	= 0.000435463279824
th + x	= 0.0	th + x	= 0.000273149412729	th + x	= 0.000197551634262
th + z	= 0.0	th + z	= 0.000273149412729	th + z	= 0.000274023234621
th + :	= 0.0	th + :	= 0.000273149412729	th + :	= 2.76147445742e-05
th + !	= 0.0	th + !	= 0.000273149412729	th + !	= 1.48694778477e-05
th + B	= 0.0	th + B	= 0.000273149412729	th + B	= 1.48694778477e-05
th + D	= 0.0	th + D	= 0.000273149412729	th + D	= 0.122885917839
th + F	= 0.0	th + F	= 0.000273149412729	th + F	= 0.00799324574361
th + H	= 0.0	th + H	= 0.000273149412729	th + H	= 0.418094684535
th + J	= 0.0	th + J	= 0.000273149412729	th + J	= 0.0042293843421
th + L	= 0.0	th + L	= 0.000273149412729	th + L	= 0.103739052223
th + N	= 0.0	th + N	= 0.000273149412729	th + N	= 0.0012893961505
th + P	= 0.0	th + P	= 0.000273149412729	th + P	= 0.00770049572943
th + R	= 0.0	th + R	= 0.000273149412729	th + R	= 0.0453006409123
th + T	= 0.0	th + T	= 0.000273149412729	th + T	= 0.000314383245922
th + U	= 0.0	th + U	= 0.000273149412729	th + U	= 0.0188996235048
th + X	= 0.0	th + X	= 0.000273149412729	th + X	= 0.00061366903983
th + Z	= 0.0	th + Z	= 0.000273149412729	th + Z	= 0.00427571557663
th + b	= 0.0	th + b	= 0.000273149412729	th + b	= 0.00660617011813

Figure 1: A printout of all trigrams probabilities for the (a) Unsmoothed English trigram model, (b) Laplace-smoothed English trigram model, and (c) Interpolation-smoothed English trigram model. It is clear to see that smoothing removes probability mass from common trigrams and adds it to less common trigrams. Also, note that in all models, the vast majority of probability mass is attributed to the trigram ‘the’ because obviously the word “the” is a trigram which is seen very frequently in the English language.

Sum(P(uns)) + Pcth			Sum(P(uns)) + Pcth			Sum(P(int)) + Pcth		
= 0.0000			= 0.0003			= 0.0021		
Sum(P(uns)) + Pcth i	=	0.0000	Sum(P(uns)) + Pcth i	=	0.0005	Sum(P(int)) + Pcth i	=	0.0021
Sum(P(uns)) + Pcth j	=	0.0579	Sum(P(uns)) + Pcth j	=	0.0574	Sum(P(int)) + Pcth j	=	0.1065
Sum(P(uns)) + Pcth k	=	0.0579	Sum(P(uns)) + Pcth k	=	0.0576	Sum(P(int)) + Pcth k	=	0.1065
Sum(P(uns)) + Pcth l	=	0.0579	Sum(P(uns)) + Pcth l	=	0.0579	Sum(P(int)) + Pcth l	=	0.1065
Sum(P(uns)) + Pcth m	=	0.0579	Sum(P(uns)) + Pcth m	=	0.0580	Sum(P(int)) + Pcth m	=	0.1065
Sum(P(uns)) + Pcth n	=	0.0579	Sum(P(uns)) + Pcth n	=	0.0580	Sum(P(int)) + Pcth n	=	0.1066
Sum(P(uns)) + Pcth o	=	0.0616	Sum(P(uns)) + Pcth o	=	0.0624	Sum(P(int)) + Pcth o	=	0.1124
Sum(P(uns)) + Pcth p	=	0.0616	Sum(P(uns)) + Pcth p	=	0.0626	Sum(P(int)) + Pcth p	=	0.1158
Sum(P(uns)) + Pcth q	=	0.0616	Sum(P(uns)) + Pcth q	=	0.0628	Sum(P(int)) + Pcth q	=	0.1162
Sum(P(uns)) + Pcth r	=	0.0616	Sum(P(uns)) + Pcth r	=	0.0631	Sum(P(int)) + Pcth r	=	0.1162
Sum(P(uns)) + Pcth s	=	0.0616	Sum(P(uns)) + Pcth s	=	0.0634	Sum(P(int)) + Pcth s	=	0.1164
Sum(P(uns)) + Pcth t	=	0.0616	Sum(P(uns)) + Pcth t	=	0.0636	Sum(P(int)) + Pcth t	=	0.1164
Sum(P(uns)) + Pcth u	=	0.0616	Sum(P(uns)) + Pcth u	=	0.0639	Sum(P(int)) + Pcth u	=	0.1173
Sum(P(uns)) + Pcth v	=	0.0616	Sum(P(uns)) + Pcth v	=	0.0645	Sum(P(int)) + Pcth v	=	0.1166
Sum(P(uns)) + Pcth w	=	0.0616	Sum(P(uns)) + Pcth w	=	0.0647	Sum(P(int)) + Pcth w	=	0.1166
Sum(P(uns)) + Pcth B	=	0.0616	Sum(P(uns)) + Pcth B	=	0.0650	Sum(P(int)) + Pcth B	=	0.1167
Sum(P(uns)) + Pcth D	=	0.0616	Sum(P(uns)) + Pcth D	=	0.0653	Sum(P(int)) + Pcth D	=	0.1168
Sum(P(uns)) + Pcth F	=	0.0616	Sum(P(uns)) + Pcth F	=	0.0656	Sum(P(int)) + Pcth F	=	0.1171
Sum(P(uns)) + Pcth H	=	0.0616	Sum(P(uns)) + Pcth H	=	0.0658	Sum(P(int)) + Pcth H	=	0.1172
Sum(P(uns)) + Pcth J	=	0.0616	Sum(P(uns)) + Pcth J	=	0.0661	Sum(P(int)) + Pcth J	=	0.1173
Sum(P(uns)) + Pcth L	=	0.0616	Sum(P(uns)) + Pcth L	=	0.0664	Sum(P(int)) + Pcth L	=	0.1173
Sum(P(uns)) + Pcth N	=	0.0616	Sum(P(uns)) + Pcth N	=	0.0666	Sum(P(int)) + Pcth N	=	0.1174
Sum(P(uns)) + Pcth P	=	0.0616	Sum(P(uns)) + Pcth P	=	0.0669	Sum(P(int)) + Pcth P	=	0.1180
Sum(P(uns)) + Pcth R	=	0.0616	Sum(P(uns)) + Pcth R	=	0.0672	Sum(P(int)) + Pcth R	=	0.1181
Sum(P(uns)) + Pcth T	=	0.0616	Sum(P(uns)) + Pcth T	=	0.0675	Sum(P(int)) + Pcth T	=	0.1187
Sum(P(uns)) + Pcth U	=	0.0616	Sum(P(uns)) + Pcth U	=	0.0677	Sum(P(int)) + Pcth U	=	0.1187
Sum(P(uns)) + Pcth X	=	0.0616	Sum(P(uns)) + Pcth X	=	0.0680	Sum(P(int)) + Pcth X	=	0.1187
Sum(P(uns)) + Pcth Z	=	0.0616	Sum(P(uns)) + Pcth Z	=	0.0683	Sum(P(int)) + Pcth Z	=	0.1187
Sum(P(uns)) + Pcth h	=	0.0616	Sum(P(uns)) + Pcth h	=	0.0686	Sum(P(int)) + Pcth h	=	0.1206
Sum(P(uns)) + Pcth d	=	0.0627	Sum(P(uns)) + Pcth d	=	0.0697	Sum(P(int)) + Pcth d	=	0.1314
Sum(P(uns)) + Pcth f	=	0.0627	Sum(P(uns)) + Pcth f	=	0.0702	Sum(P(int)) + Pcth f	=	0.1368
Sum(P(uns)) + Pcth j	=	0.0627	Sum(P(uns)) + Pcth j	=	0.0705	Sum(P(int)) + Pcth j	=	0.1492
Sum(P(uns)) + Pcth i	=	0.0627	Sum(P(uns)) + Pcth i	=	0.0707	Sum(P(int)) + Pcth i	=	0.1497
Sum(P(uns)) + Pcth l	=	0.0632	Sum(P(uns)) + Pcth l	=	0.0716	Sum(P(int)) + Pcth l	=	0.1607
Sum(P(uns)) + Pcth n	=	0.0632	Sum(P(uns)) + Pcth n	=	0.0718	Sum(P(int)) + Pcth n	=	0.1810
Sum(P(uns)) + Pcth p	=	0.0632	Sum(P(uns)) + Pcth p	=	0.0721	Sum(P(int)) + Pcth p	=	0.1876
Sum(P(uns)) + Pcth r	=	0.0634	Sum(P(uns)) + Pcth r	=	0.0784	Sum(P(int)) + Pcth r	=	0.2206
Sum(P(uns)) + Pcth t	=	0.0634	Sum(P(uns)) + Pcth t	=	0.0787	Sum(P(int)) + Pcth t	=	0.2391
Sum(P(uns)) + Pcth v	=	0.0634	Sum(P(uns)) + Pcth v	=	0.0789	Sum(P(int)) + Pcth v	=	0.2420
Sum(P(uns)) + Pcth x	=	0.0634	Sum(P(uns)) + Pcth x	=	0.0792	Sum(P(int)) + Pcth x	=	0.2425
Sum(P(uns)) + Pcth z	=	0.0634	Sum(P(uns)) + Pcth z	=	0.0795	Sum(P(int)) + Pcth z	=	0.2426
Sum(P(uns)) + Pcth f	=	0.0634	Sum(P(uns)) + Pcth f	=	0.0798	Sum(P(int)) + Pcth f	=	0.2426
Sum(P(uns)) + Pcth z	=	0.0634	Sum(P(uns)) + Pcth z	=	0.0800	Sum(P(int)) + Pcth z	=	0.2426
Sum(P(uns)) + Pcth f	=	0.0634	Sum(P(uns)) + Pcth f	=	0.0803	Sum(P(int)) + Pcth f	=	0.2426
Sum(P(uns)) + Pcth h	=	0.0634	Sum(P(uns)) + Pcth h	=	0.0806	Sum(P(int)) + Pcth h	=	0.2430
Sum(P(uns)) + Pcth l	=	0.0634	Sum(P(uns)) + Pcth l	=	0.0809	Sum(P(int)) + Pcth l	=	0.2430
Sum(P(uns)) + Pcth n	=	0.0634	Sum(P(uns)) + Pcth n	=	0.0811	Sum(P(int)) + Pcth n	=	0.2435
Sum(P(uns)) + Pcth l	=	0.0634	Sum(P(uns)) + Pcth l	=	0.0814	Sum(P(int)) + Pcth l	=	0.2436
Sum(P(uns)) + Pcth 1	=	0.0634	Sum(P(uns)) + Pcth 1	=	0.0817	Sum(P(int)) + Pcth 1	=	0.2439
Sum(P(uns)) + Pcth 3	=	0.0634	Sum(P(uns)) + Pcth 3	=	0.0819	Sum(P(int)) + Pcth 3	=	0.2439
Sum(P(uns)) + Pcth 5	=	0.0634	Sum(P(uns)) + Pcth 5	=	0.0822	Sum(P(int)) + Pcth 5	=	0.2440
Sum(P(uns)) + Pcth 7	=	0.0634	Sum(P(uns)) + Pcth 7	=	0.0825	Sum(P(int)) + Pcth 7	=	0.2441
Sum(P(uns)) + Pcth 9	=	0.0634	Sum(P(uns)) + Pcth 9	=	0.0828	Sum(P(int)) + Pcth 9	=	0.2444
Sum(P(uns)) + Pcth i	=	0.0634	Sum(P(uns)) + Pcth i	=	0.0831	Sum(P(int)) + Pcth i	=	0.2445
Sum(P(uns)) + Pcth j	=	0.0634	Sum(P(uns)) + Pcth j	=	0.0833	Sum(P(int)) + Pcth j	=	0.2445
Sum(P(uns)) + Pcth 7	=	0.0634	Sum(P(uns)) + Pcth 7	=	0.0836	Sum(P(int)) + Pcth 7	=	0.2445
Sum(P(uns)) + Pcth a	=	0.0634	Sum(P(uns)) + Pcth a	=	0.0839	Sum(P(int)) + Pcth a	=	0.2448
Sum(P(uns)) + Pcth C	=	0.0634	Sum(P(uns)) + Pcth C	=	0.0841	Sum(P(int)) + Pcth C	=	0.2455
Sum(P(uns)) + Pcth T	=	0.0634	Sum(P(uns)) + Pcth T	=	0.0844	Sum(P(int)) + Pcth T	=	0.2456
Sum(P(uns)) + Pcth E	=	0.0634	Sum(P(uns)) + Pcth E	=	0.0847	Sum(P(int)) + Pcth E	=	0.2461
Sum(P(uns)) + Pcth G	=	0.0634	Sum(P(uns)) + Pcth G	=	0.0849	Sum(P(int)) + Pcth G	=	0.2462
Sum(P(uns)) + Pcth I	=	0.0634	Sum(P(uns)) + Pcth I	=	0.0852	Sum(P(int)) + Pcth I	=	0.2462
Sum(P(uns)) + Pcth G	=	0.0634	Sum(P(uns)) + Pcth G	=	0.0853	Sum(P(int)) + Pcth G	=	0.2464
Sum(P(uns)) + Pcth M	=	0.0634	Sum(P(uns)) + Pcth M	=	0.0858	Sum(P(int)) + Pcth M	=	0.2473
Sum(P(uns)) + Pcth o	=	0.0634	Sum(P(uns)) + Pcth o	=	0.0860	Sum(P(int)) + Pcth o	=	0.2480
Sum(P(uns)) + Pcth Q	=	0.0634	Sum(P(uns)) + Pcth Q	=	0.0863	Sum(P(int)) + Pcth Q	=	0.2481
Sum(P(uns)) + Pcth S	=	0.0634	Sum(P(uns)) + Pcth S	=	0.0866	Sum(P(int)) + Pcth S	=	0.2485
Sum(P(uns)) + Pcth U	=	0.0634	Sum(P(uns)) + Pcth U	=	0.0869	Sum(P(int)) + Pcth U	=	0.2487
Sum(P(uns)) + Pcth W	=	0.0634	Sum(P(uns)) + Pcth W	=	0.0871	Sum(P(int)) + Pcth W	=	0.2487
Sum(P(uns)) + Pcth Y	=	0.0634	Sum(P(uns)) + Pcth Y	=	0.0874	Sum(P(int)) + Pcth Y	=	0.2490
Sum(P(uns)) + Pcth G	=	0.0634	Sum(P(uns)) + Pcth G	=	0.0879	Sum(P(int)) + Pcth G	=	0.2490
Sum(P(uns)) + Pcth I	=	0.0634	Sum(P(uns)) + Pcth I	=	0.0880	Sum(P(int)) + Pcth I	=	0.2491
Sum(P(uns)) + Pcth a	=	0.1975	Sum(P(uns)) + Pcth a	=	0.2133	Sum(P(int)) + Pcth a	=	0.3719
Sum(P(uns)) + Pcth c	=	0.1978	Sum(P(uns)) + Pcth c	=	0.2139	Sum(P(int)) + Pcth c	=	0.3799
Sum(P(uns)) + Pcth e	=	0.8626	Sum(P(uns)) + Pcth e	=	0.8632	Sum(P(int)) + Pcth e	=	0.7980
Sum(P(uns)) + Pcth g	=	0.8626	Sum(P(uns)) + Pcth g	=	0.8634	Sum(P(int)) + Pcth g	=	0.8023
Sum(P(uns)) + Pcth i	=	0.9734	Sum(P(uns)) + Pcth i	=	0.9719	Sum(P(int)) + Pcth i	=	0.9060
Sum(P(uns)) + Pcth k	=	0.9734	Sum(P(uns)) + Pcth k	=	0.9721	Sum(P(int)) + Pcth k	=	0.9073
Sum(P(uns)) + Pcth m	=	0.9734	Sum(P(uns)) + Pcth m	=	0.9724	Sum(P(int)) + Pcth m	=	0.9101
Sum(P(uns)) + Pcth q	=	0.9727	Sum(P(uns)) + Pcth q	=	0.9729	Sum(P(int)) + Pcth q	=	0.9603
Sum(P(uns)) + Pcth q	=	0.9927	Sum(P(uns)) + Pcth q	=	0.9918	Sum(P(int)) + Pcth q	=	0.9606
Sum(P(uns)) + Pcth s	=	0.9955	Sum(P(uns)) + Pcth s	=	0.9948	Sum(P(int)) + Pcth s	=	0.9795
Sum(P(uns)) + Pcth u	=	0.9975	Sum(P(uns)) + Pcth u	=	0.9970	Sum(P(int)) + Pcth u	=	0.9891
Sum(P(uns)) + Pcth w	=	0.9980	Sum(P(uns)) + Pcth w	=	0.9978	Sum(P(int)) + Pcth w	=	0.9934
Sum(P(uns)) + Pcth y	=	1.0000	Sum(P(uns)) + Pcth y	=	1.0000	Sum(P(int)) + Pcth y	=	1.0000

(a)
(b)
(c)

Figure 2: To ensure that each model constructs a valid probability distribution, the sum of all possible trigrams given some bigram should equal 1. In this case, sums were calculated for the (a) English Unsmoothed trigram model, (b) English Laplace-smoothed trigram model, and (c) English Interpolation-smoothed trigram model all given a preceding bigram ‘th’. All models did, indeed, sum to 1 indicating that the models do produce valid probability distributions over trigrams.