In this assignment, you will build unigram, bigram, and trigram character language models (both unsmoothed and smoothed versions) for three languages, score a test document with each, and determine the language it is written in based on perplexity. You will also use your English language models to generate texts. You will critically examine all results. The learning goals of this assignment are to:

- Understand how to compute language model probabilities using maximum likelihood estimation.
- Implement basic smoothing and interpolation.
- Use the perplexity of a language model to perform language identification.
- Use a language model to probabilistically generate texts.

Data

The data for this project is available here: hw3_data.zip. It consists of:

- training.en - English training data
- training.es - Spanish training data
- training.de - German training data
- test - test document

In [ ]:
```
import pandas as pd

filenames = ['hw3_data/training.en.txt', 'hw3_data/training.de.txt', 'hw3_data/training.es.txt', 'hw
with open(filenames[0], 'r') as f:
    en = f.read().lower()
with open(filenames[1], 'r') as f:
    de = f.read().lower()
with open(filenames[2], 'r') as f:
    es = f.read().lower()
with open(filenames[3], 'r') as f:
    test = f.read().lower()

# en = pd.read_csv('hw3_data/training.en.txt', sep=' ', header = None)
# es = pd.read_csv()
# de = pd.read_csv()
# test = pd.read_csv()
```

1. Train character n-gram language models

To complete the assignment, you will need to write a program (from scratch) that:

- builds the models: reads in training data, collects counts for all character 1, 2, and 3-grams, estimates probabilities, and writes out the unigram, bigram, and trigram models (ngram probabilities) into files. Build separate models for each language.
- adjusts the counts: rebuilds the bigram and trigram language models using linear interpolation with lambdas equally weighted

You may make any additional assumptions and design decisions, but state them in your report (see below). For example, some design choices that could be made are how you want to handle uppercase and lowercase letters or how you want to handle digits. The choice made is up to you, we only require that you detail these decisions in your report and consider any implications of them in your results. There is no wrong choice here, and these decisions are typically made by NLP researchers when pre-processing data.

You may write your program in any TA-approved programming language (Python, Java, C/C++).

For this assignment you must implement the model generation from scratch, but you are allowed to use any resources or packages that help you manage your project, i.e. Github or any file i/o packages. If you have questions about this please ask.

Extra credit (3 points): Also implement add-one smoothing.

In [ ]:
```
tmp_en = []
for line in en.split('\n'):
    tmp_en.append('<s>')
    tmp_en.extend(list(line))
    tmp_en.append('</s>')
tmp_es = []
for line in es.split('\n'):
    tmp_en.append('<s>')
    tmp_es.extend(list(line))
    tmp_en.append('</s>')
tmp_de = []
for line in de.split('\n'):
    tmp_en.append('<s>')
    tmp_de.extend(list(line))
    tmp_en.append('</s>')
en = tmp_en
es = tmp_es
de = tmp_de
del tmp_en, tmp_es, tmp_de
```

Unigrams

In [ ]:
```
en_uni_counts = {}
for char in en:
    if char in en_uni_counts:
        en_uni_counts[char] += 1
    else:
        en_uni_counts[char] = 1

es_uni_counts = {}
for char in es:
    if char in es_uni_counts:
        es_uni_counts[char] += 1
    else:
        es_uni_counts[char] = 1

de_uni_counts = {}
for char in de:
    if char in de_uni_counts:
        de_uni_counts[char] += 1
    else:
        de_uni_counts[char] = 1
```

Bigrams

In [ ]:
```
import numpy as np

en_bigrams = [(en[i], en[i+1]) for i in range(len(en) - 1)]
en_bi_counts = {}
for bigram in en_bigrams:
    if bigram in en_bi_counts:
        en_bi_counts[bigram] += 1
    else:
        en_bi_counts[bigram] = 1


en_bi_matrix = np.zeros((len(en_uni_counts), len(en_uni_counts)))
en_uni_index = {unigram: index for index, unigram in enumerate(en_uni_counts)}
for bigram, count in en_bi_counts.items():
    en_bi_matrix[en_uni_index[bigram[0]], en_uni_index[bigram[1]]] = count

es_bigrams = [(es[i], es[i+1]) for i in range(len(es) - 1)]
es_bi_counts = {}
for bigram in es_bigrams:
    if bigram in es_bi_counts:
        es_bi_counts[bigram] += 1
    else:
        es_bi_counts[bigram] = 1

es_bi_matrix = np.zeros((len(es_uni_counts), len(es_uni_counts)))
es_uni_index = {unigram: index for index, unigram in enumerate(es_uni_counts)}
for bigram, count in es_bi_counts.items():
    es_bi_matrix[es_uni_index[bigram[0]], es_uni_index[bigram[1]]] = count

de_bigrams = [(de[i], de[i+1]) for i in range(len(de) - 1)]
de_bi_counts = {}
for bigram in de_bigrams:
    if bigram in de_bi_counts:
        de_bi_counts[bigram] += 1
    else:
        de_bi_counts[bigram] = 1

de_bi_matrix = np.zeros((len(de_uni_counts), len(de_uni_counts)))
de_uni_index = {unigram: index for index, unigram in enumerate(de_uni_counts)}
for bigram, count in de_bi_counts.items():
    de_bi_matrix[de_uni_index[bigram[0]], de_uni_index[bigram[1]]] = count
```

Trigrams

In [ ]:
```
en_trigrams = [(en[i], en[i+1], en[i+2]) for i in range(len(en) - 2)]
en_tri_counts = {}
for trigram in en_trigrams:
    if trigram in en_tri_counts:
        en_tri_counts[trigram] += 1
    else:
        en_tri_counts[trigram] = 1

en_tri_matrix = np.zeros((len(en_bi_counts), len(en_uni_counts)))
en_bi_index = {bigram: index for index, bigram in enumerate(en_bi_counts)}
for trigram, count in en_tri_counts.items():
    en_tri_matrix[en_bi_index[trigram[0:2]], en_uni_index[trigram[2]]] = count

es_trigrams = [(es[i], es[i+1], es[i+2]) for i in range(len(es) - 2)]
es_tri_counts = {}
for trigram in es_trigrams:
    if trigram in es_tri_counts:
        es_tri_counts[trigram] += 1
    else:
        es_tri_counts[trigram] = 1

es_tri_matrix = np.zeros((len(es_bi_counts), len(es_uni_counts)))
es_bi_index = {bigram: index for index, bigram in enumerate(es_bi_counts)}
for trigram, count in es_tri_counts.items():
    es_tri_matrix[es_bi_index[trigram[0:2]], es_uni_index[trigram[2]]] = count

de_trigrams = [(de[i], de[i+1], de[i+2]) for i in range(len(de) - 2)]
```

```
de_tri_counts = {}
for trigram in de_trigrams:
    if trigram in de_tri_counts:
        de_tri_counts[trigram] += 1
    else:
        de_tri_counts[trigram] = 1

de_tri_matrix = np.zeros((len(de_bi_counts), len(de_uni_counts)))
de_bi_index = {bigram: index for index, bigram in enumerate(de_bi_counts)}
for trigram, count in de_tri_counts.items():
    de_tri_matrix[de_bi_index[trigram[0:2]], de_uni_index[trigram[2]]] = count
```

"writes out the unigram, bigram, and trigram models (ngram probabilities) into files. Build separate models for each language."

Convert ngrams into probabilities and write to files

In [ ]:
```
en_total = sum(en_uni_counts.values())
es_total = sum(es_uni_counts.values())
de_total = sum(de_uni_counts.values())

en_uni_probs = {key: value / en_total for key, value in en_uni_counts.items()}
es_uni_probs = {key: value / es_total for key, value in es_uni_counts.items()}
de_uni_probs = {key: value / de_total for key, value in de_uni_counts.items()}
```

In [ ]:
```
sum(en_uni_probs.values())
```

Out[ ]:
```
1.0000000000000004
```

In [ ]:
```
print(en_bi_matrix.shape, es_bi_matrix.shape, de_bi_matrix.shape)
```

```
(62, 62) (64, 64) (63, 63)
```

In [ ]:
```
en_bi_sums = en_bi_matrix.sum(axis = 1)[:, np.newaxis] # sum cols (words) of each row (context)
en_bi_sums[en_bi_sums == 0] = 1 # no div by 0
en_bi_matrix_norm = en_bi_matrix / en_bi_sums

es_bi_sums = es_bi_matrix.sum(axis = 1)[:, np.newaxis] # sum over cols (words) of each row (context)
es_bi_sums[es_bi_sums == 0] = 1 # no div by 0
es_bi_matrix_norm = es_bi_matrix / es_bi_sums

de_bi_sums = de_bi_matrix.sum(axis = 1)[:, np.newaxis] # sum cols (words) of each row (context)
de_bi_sums[de_bi_sums == 0] = 1 # no div by 0
de_bi_matrix_norm = de_bi_matrix / de_bi_sums
```

In [ ]:
```
print(en_bi_matrix_norm.sum(axis=1), es_bi_matrix_norm.sum(axis=1), de_bi_matrix_norm.sum(axis=1))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [ ]:
```
print(en_tri_matrix.shape, es_tri_matrix.shape, de_tri_matrix.shape)
```

```
(858, 62) (858, 64) (961, 63)
```

In [ ]:
```
en_tri_sums = en_tri_matrix.sum(axis = 1)[:, np.newaxis]
en_tri_sums[en_tri_sums == 0] = 1 # no div by 0
```

```
en_tri_matrix_norm = en_tri_matrix / en_tri_sums

es_tri_sums = es_tri_matrix.sum(axis = 1)[:, np.newaxis]
es_tri_sums[es_tri_sums == 0] = 1 # no div by 0
es_tri_matrix_norm = es_tri_matrix / es_tri_sums

de_tri_sums = de_tri_matrix.sum(axis = 1)[:, np.newaxis]
de_tri_sums[de_tri_sums == 0] = 1 # no div by 0
de_tri_matrix_norm = de_tri_matrix / de_tri_sums
```

In [ ]:

```
print(en_tri_matrix_norm.sum(axis=1), es_tri_matrix_norm.sum(axis=1), de_tri_matrix_norm.sum(axis=1)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1.]
```

- adjusts the counts: rebuilds the bigram and trigram language models using linear interpolation with lambdas equally weighted

In [ ]:

```
reverse_en_uni_index = {value: key for key, value in en_uni_index.items()}
reverse_es_uni_index = {value: key for key, value in es_uni_index.items()}
reverse_de_uni_index = {value: key for key, value in de_uni_index.items()}

en_bi_matrix_ip = np.zeros(en_bi_matrix.shape)
for i in range(en_bi_matrix.shape[0]):
    for j in range(en_bi_matrix.shape[1]):
        en_bi_matrix_ip[i,j] = 0.5 * en_uni_counts[reverse_en_uni_index[j]] + 0.5 * en_bi_matrix[i,j
```

```
es_bi_matrix_ip = np.zeros(es_bi_matrix.shape)
for i in range(es_bi_matrix.shape[0]):
    for j in range(es_bi_matrix.shape[1]):
        es_bi_matrix_ip[i,j] = 0.5 * es_uni_counts[reverse_es_uni_index[j]] + 0.5 * es_bi_matrix[i,j

de_bi_matrix_ip = np.zeros(de_bi_matrix.shape)
for i in range(de_bi_matrix.shape[0]):
    for j in range(de_bi_matrix.shape[1]):
        de_bi_matrix_ip[i,j] = 0.5 * de_uni_counts[reverse_de_uni_index[j]] + 0.5 * de_bi_matrix[i,j
```

In [ ]:
```
en_bi_sums_ip = en_bi_matrix_ip.sum(axis = 1)[:, np.newaxis]
en_bi_sums_ip[en_bi_sums_ip == 0] = 1 # no div by 0
en_bi_matrix_ip_norm = en_bi_matrix_ip / en_bi_sums_ip

es_bi_sums_ip = es_bi_matrix_ip.sum(axis = 1)[:, np.newaxis]
es_bi_sums_ip[es_bi_sums_ip == 0] = 1 # no div by 0
es_bi_matrix_ip_norm = es_bi_matrix_ip / es_bi_sums_ip

de_bi_sums_ip = de_bi_matrix_ip.sum(axis = 1)[:, np.newaxis]
de_bi_sums_ip[de_bi_sums_ip == 0] = 1 # no div by 0
de_bi_matrix_ip_norm = de_bi_matrix_ip / de_bi_sums_ip
```

In [ ]:
```
print(en_bi_matrix_ip_norm.sum(axis=1))
print(es_bi_matrix_ip_norm.sum(axis=1))
print(de_bi_matrix_ip_norm.sum(axis=1))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [ ]:
```
reverse_en_bi_index = {value: key for key, value in en_bi_index.items()}
reverse_es_bi_index = {value: key for key, value in es_bi_index.items()}
reverse_de_bi_index = {value: key for key, value in de_bi_index.items()}

en_tri_matrix_ip = np.zeros(en_tri_matrix.shape)
for i in range(en_tri_matrix.shape[0]):
    # give me the second character in the rownames of tri
    bi_row = en_uni_index[reverse_en_bi_index[i][0]]
    for j in range(en_tri_matrix.shape[1]):
        uni_col = reverse_en_uni_index[j]
        bi_col = en_uni_index[reverse_en_bi_index[i][1]]
        en_tri_matrix_ip[i,j] = 1/3 * en_uni_counts[uni_col] + 1/3 * en_bi_matrix[bi_row,bi_col] + 1

es_tri_matrix_ip = np.zeros(es_tri_matrix.shape)
for i in range(es_tri_matrix.shape[0]):
    bi_row = es_uni_index[reverse_es_bi_index[i][0]]
    for j in range(es_tri_matrix.shape[1]):
        uni_col = reverse_es_uni_index[j]
        bi_col = es_uni_index[reverse_es_bi_index[i][1]]
        es_tri_matrix_ip[i,j] = 1/3 * es_uni_counts[uni_col] + 1/3 * es_bi_matrix[bi_row,bi_col] + 1

de_tri_matrix_ip = np.zeros(de_tri_matrix.shape)
for i in range(de_tri_matrix.shape[0]):
    bi_row = de_uni_index[reverse_de_bi_index[i][0]]
    for j in range(de_tri_matrix.shape[1]):
        uni_col = reverse_de_uni_index[j]
        bi_col = de_uni_index[reverse_de_bi_index[i][1]]
        de_tri_matrix_ip[i,j] = 1/3 * de_uni_counts[uni_col] + 1/3 * de_bi_matrix[bi_row,bi_col] + 1
```

In [ ]:

```
en_tri_sums_ip = en_tri_matrix_ip.sum(axis = 1)[:, np.newaxis]
en_tri_sums_ip[en_tri_sums_ip == 0] = 1 # no div by 0
en_tri_matrix_ip_norm = en_tri_matrix_ip / en_tri_sums_ip

es_tri_sums_ip = es_tri_matrix_ip.sum(axis = 1)[:, np.newaxis]
es_tri_sums_ip[es_tri_sums_ip == 0] = 1 # no div by 0
es_tri_matrix_ip_norm = es_tri_matrix_ip / es_tri_sums_ip

de_tri_sums_ip = de_tri_matrix_ip.sum(axis = 1)[:, np.newaxis]
de_tri_sums_ip[de_tri_sums_ip == 0] = 1 # no div by 0
de_tri_matrix_ip_norm = de_tri_matrix_ip / de_tri_sums_ip
```

In [ ]:

```
print(en_tri_matrix_ip_norm.sum(axis=1))
print(es_tri_matrix_ip_norm.sum(axis=1))
print(de_tri_matrix_ip_norm.sum(axis=1))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1.]
```

- a description of how you wrote your program, including all assumptions and design decisions
- an excerpt of both the unsmoothed and interpolated (and optional add-one) trigram language models for English, displaying all n-grams and their probability with the two-character history t h
- documentation that probability distributions for all language models are valid (sum to 1). Specifically, where w is a word (random variable) and c is the prior context of that word, this means the probability distributions P(w|c)= 1 or very close to 1 (>0.98 and <1.02 is fine) for all possible prior contexts.

Deliverable Part 1.1:

Assumptions and Design Decision: Every sentence starts and ends with a newline. Therefore, splitting the corpus on \n means I could place <s> and </s> indicators at the start and end of each line. For future sentence generation, I split on characters in such a way that would include spaces, which combined with </s> tells me what character strings are meant as words and where sentences are meant to end.

I make all words lower case. If there are any words that have different meanings when they are uppercase vs. lowercase, perhaps indicating a proper noun versus an ordinary noun, this will convolve the counts of two different words together, which is problematic for the representations of the affected words. I have ignored dealing with digits, so if there is 'seven' and '7', these will have separate counts and be considered separate individual words.

Deliverable part 1.2: character: (unsmoothed, interpolated) with the two character history "t h"

In [ ]:
```
dict(zip(en_uni_counts.keys(), zip(en_tri_matrix_norm[en_bi_index[('t', 'h')]], en_tri_matrix_ip_nor
Out[ ]:  {'<s>': (0.0, 0.01694704915835613),
          'r': (0.00575163398692810‌4, 0.02979880963609017),
          'e': (0.6603921568627451, 0.05598631925381359),
          's': (0.00261437908496732, 0.030349812113119323),
          'u': (0.00392156862745098, 0.018689408342475344),
          'm': (0.0, 0.018902859752495643),
          'p': (0.0, 0.01781326476413619),
          't': (0.0, 0.040416776288030344),
          'i': (0.12156862745098039, 0.03561163756943376),
          'o': (0.018300653594771243, 0.03544038004278956),
          'n': (0.0, 0.03254637604181662),
          ' ': (0.05411764705882353, 0.0725163935646882),
          'f': (0.0, 0.01615032936048965),
          'h': (0.0, 0.024062923490079473),
          '</s>': (0.0, 0.01694704915835613),
          'd': (0.0010457516339869282, 0.0199155129535222),
          'c': (0.00026143790849673205, 0.019548177968836098),
          'l': (0.0005228758169934641, 0.02122104134504172),
          'a': (0.1257516339869281, 0.033916436254970185),
          'j': (0.0, 0.00998505840129858),
          'y': (0.0018300653594771241, 0.014286352512521656),
          '1': (0.0, 0.009836138812912322),
          '7': (0.0, 0.009543263622419348),
          'b': (0.0, 0.013449920824418841),
          '9': (0.0, 0.00990811661396568),
          ',': (0.001568627450980392, 0.013514452646052888),
          'w': (0.0005228758169934641, 0.014462574025445395),
          'k': (0.0, 0.011052315451400093),
          'g': (0.0, 0.014613975606971422),
          'v': (0.0, 0.012817012573777247),
          '.': (0.0018300653594771241, 0.011930941022879013),
          "'": (0.0, 0.00975175104616011),
          'q': (0.0, 0.009870886716869115),
          'x': (0.0, 0.010133977989684836),
          '-': (0.0, 0.010044626236653082),
          '(': (0.0, 0.009580493519515914),
          ')': (0.0, 0.009592903485214768),
          '?': (0.0, 0.009553191594978432),
          '4': (0.0, 0.009553191594978432),
          '3': (0.0, 0.009540781629279576),
          '6': (0.0, 0.009565601560677288),
          'z': (0.0, 0.009602831457773853),
          '8': (0.0, 0.009570565546956828),
          'ã': (0.0, 0.009533335649860263),
          '\xad': (0.0, 0.0094961057527637),
          '¡': (0.0, 0.009501069739043242),
          '0': (0.0, 0.009853512764890719),
          ':': (0.0, 0.009540781629279576),
          '³': (0.0, 0.009506033725322785),
          '5': (0.0, 0.009610277437193164),
          ';': (0.0, 0.009535817643000035),
          '¤': (0.0, 0.009506033725322785),
          '2': (0.0, 0.009719485135343086),
          '!': (0.0, 0.009503551732183013),
          '"': (0.0, 0.00952340767730118),
          '/': (0.0, 0.009578011526376141),
          '%': (0.0, 0.009528371663580722),
          '[': (0.0, 0.009510997711602326),
          ']': (0.0, 0.009510997711602326),
          'â': (0.0, 0.009501069739043242),
          'º': (0.0, 0.009501069739043242),
          '©': (0.0, 0.009498587745903472)}
```

Deliverable 1.3: Can be observed in the above print statements that are mostly 1s

interpolated

In [ ]:
```
import math

def perplexity(text, model, n, lang_string):
    """ Args:
            text: a string of characters
            model: a matrix or df of the probabilities with rows as prefixes, columns as suffixes.
                        You can modify this depending on how you set up your model.
            n: n-gram order of the model

        Acknowledgment:
          https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3
          https://courses.cs.washington.edu/courses/csep517/18au/
          ChatGPT with GPT-3.5
    """

    # FILL IN: Remove any unseen characters from the text that have no unigram probability in the la
    if lang_string == 'en':
        text = [char for char in text if char in en_uni_probs]
    if lang_string == 'es':
        text = [char for char in text if char in es_uni_probs]
    if lang_string == 'de':
        text = [char for char in text if char in de_uni_probs]

    N = len(text)
    if N - n + 1 == 0:
        return
    char_probs = []
    for i in range(n-1, N):
        prefix = text[i-n+1:i]
        if n == 2:
            prefix = prefix[0]
        if n == 3:
            prefix = tuple(prefix)
        suffix = text[i]
        # FILL IN: look up the probability in the model of the suffix given the prefix
        if lang_string == 'en':
            if n == 1:
                prob = en_uni_probs[suffix]
            elif n == 2:
                prob = model[en_uni_index[prefix], en_uni_index[suffix]]
            elif n == 3:
                try:
                    prob = model[en_bi_index[prefix], en_uni_index[suffix]]
                except KeyError:
                    continue
        elif lang_string == 'es':
            if n == 1:
                prob = es_uni_probs[suffix]
            elif n == 2:
                prob = model[es_uni_index[prefix], es_uni_index[suffix]]
            elif n == 3:
                try:
                    prob = model[es_bi_index[prefix], es_uni_index[suffix]]
                except KeyError:
                    continue
        elif lang_string == 'de':
            if n == 1:
                prob = de_uni_probs[suffix]
            elif n == 2:
                prob = model[de_uni_index[prefix], de_uni_index[suffix]]
            elif n == 3:
                try:
                    prob = model[de_bi_index[prefix], de_uni_index[suffix]]
                except KeyError:
                    continue

        char_probs.append(math.log2(prob))
    neg_log_lik = -1 * sum(char_probs) # negative log-likelihood of the text
    ppl = 2 ** (neg_log_lik/(N - n + 1)) # 2 to the power of the negative log likelihood of the word
    return ppl
```

In [ ]:

```
en_uni_ppls = []
es_uni_ppls = []
de_uni_ppls = []
en_bi_ppls = []
es_bi_ppls = []
de_bi_ppls = []
en_tri_ppls = []
es_tri_ppls = []
de_tri_ppls = []

for line in test.split("\n"):
    if line == None:
        continue
    test_line = []
    test_line.append('<s>')
    test_line.extend(list(line))
    test_line.append('</s>')
    en_uni_ppls.append(perplexity(test_line, en_uni_probs, n=1, lang_string="en"))
    es_uni_ppls.append(perplexity(test_line, es_uni_probs, n=1, lang_string="es"))
    de_uni_ppls.append(perplexity(test_line, de_uni_probs, n=1, lang_string="de"))
    en_bi_ppls.append(perplexity(test_line, en_bi_matrix_ip_norm, n=2, lang_string="en"))
    es_bi_ppls.append(perplexity(test_line, es_bi_matrix_ip_norm, n=2, lang_string="es"))
    de_bi_ppls.append(perplexity(test_line, de_bi_matrix_ip_norm, n=2, lang_string="de"))
    en_tri_ppls.append(perplexity(test_line, en_tri_matrix_ip_norm, n=3, lang_string="en"))
    es_tri_ppls.append(perplexity(test_line, es_tri_matrix_ip_norm, n=3, lang_string="es"))
    de_tri_ppls.append(perplexity(test_line, de_tri_matrix_ip_norm, n=3, lang_string="de"))

en_uni_ppls = [x for x in en_uni_ppls if x is not None]
es_uni_ppls = [x for x in es_uni_ppls if x is not None]
de_uni_ppls = [x for x in de_uni_ppls if x is not None]
en_bi_ppls = [x for x in en_bi_ppls if x is not None]
es_bi_ppls = [x for x in es_bi_ppls if x is not None]
de_bi_ppls = [x for x in de_bi_ppls if x is not None]
en_tri_ppls = [x for x in en_tri_ppls if x is not None]
es_tri_ppls = [x for x in es_tri_ppls if x is not None]
de_tri_ppls = [x for x in de_tri_ppls if x is not None]
```

In [ ]:

```
print("English unigram mean perplexity: " + str(np.mean(en_uni_ppls)))
print("Spanish unigram mean perplexity: " + str(np.mean(es_uni_ppls)))
print("German unigram mean perplexity: " + str(np.mean(de_uni_ppls)))
print("English bigram mean perplexity: " + str(np.mean(en_bi_ppls)))
print("Spanish bigram mean perplexity: " + str(np.mean(es_bi_ppls)))
print("German bigram mean perplexity: " + str(np.mean(de_bi_ppls)))
print("English trigram mean perplexity: " + str(np.mean(en_tri_ppls)))
print("Spanish trigram mean perplexity: " + str(np.mean(es_tri_ppls)))
print("German trigram mean perplexity: " + str(np.mean(de_tri_ppls)))
```

```
English unigram mean perplexity: 21.737625763834025
Spanish unigram mean perplexity: 24.05886863514533
German unigram mean perplexity: 22.763693053589467
English bigram mean perplexity: 20.094363804945925
Spanish bigram mean perplexity: 22.9566938949385
German bigram mean perplexity: 21.775280898105684
English trigram mean perplexity: 23.509588211949396
Spanish trigram mean perplexity: 21.537301518236614
German trigram mean perplexity: 22.986788878890835
```

Deliverable 2.1:

It is clear that the unigram and bigram english models minimized perplexity the most, but that the trigram spanish model performed better than the english trigram model. There are two votes for English, so it is reasonable to select English as the test language.

generated text outputs for the following inputs: bigrams starting with 10 letters of your choice, and trigrams using those 10 letters as the first character with a second meaningful character of your choice. This is for English bigram and trigram models, both unsmoothed and smoothed.

Deliverable 2.2

In [ ]:
```python
my_char_list = ['a','b','c','d','e','f','o','p','m','n']
second_char_list = ['b', 'e','a', 'o', 'i', 'a', 't', 'o', 'a', 'e']
print("Bigram Unsmoothed:\n")
for input_char in my_char_list:
    print("First character " + input_char + ": ")
    generated_text = input_char
    while input_char != '</s>':
        row_vector = en_bi_matrix_norm[en_uni_index[input_char]]

        row_vector = row_vector / row_vector.sum()

        column_number = np.random.choice(np.arange(len(row_vector)), p=row_vector)
        input_char = reverse_en_uni_index[column_number]
        generated_text += input_char

    print(generated_text)

print("Bigram Smoothed:\n")
for input_char in my_char_list:
    print("First character " + input_char + ": ")
    generated_text = input_char
    while input_char != '</s>':
        row_vector = en_bi_matrix_ip_norm[en_uni_index[input_char]]

        row_vector = row_vector / row_vector.sum()

        column_number = np.random.choice(np.arange(len(row_vector)), p=row_vector)
        input_char = reverse_en_uni_index[column_number]
        generated_text += input_char

    print(generated_text)

print("Trigram Unsmoothed:\n")
for input_char, sec_char in zip(my_char_list, second_char_list):
    print("First character " + input_char + " Second Character " + sec_char+ ": ")
    generated_text = input_char + sec_char
    while input_char != '</s>':
        try:
            row_vector = en_tri_matrix_norm[en_bi_index[(input_char, sec_char)]]

            row_vector = row_vector / row_vector.sum()

            column_number = np.random.choice(np.arange(len(row_vector)), p=row_vector)
            new_input_char = sec_char
            new_sec_char = reverse_en_uni_index[column_number]

            _ = en_tri_matrix_norm[en_bi_index[(new_input_char, new_sec_char)]]

            input_char = new_input_char
            sec_char = new_sec_char
            generated_text += input_char

        except KeyError:
            continue

    print(generated_text)

print("Trigram Smoothed:\n")
for input_char, sec_char in zip(my_char_list, second_char_list):
    print("First character " + input_char + " Second Character " + sec_char+ ": ")
    generated_text = input_char
    while input_char != '</s>':
        try:
            row_vector = en_tri_matrix_ip_norm[en_bi_index[(input_char, sec_char)]]

            row_vector = row_vector / row_vector.sum()

            column_number = np.random.choice(np.arange(len(row_vector)), p=row_vector)
            new_input_char = sec_char
            new_sec_char = reverse_en_uni_index[column_number]

            _ = en_tri_matrix_ip_norm[en_bi_index[(new_input_char, new_sec_char)]]

            input_char = new_input_char
            sec_char = new_sec_char
            generated_text += input_char

        except KeyError:
            continue

    print(generated_text)
```

```
Bigram Unsmoothed:

First character a:
and turel st e nis ony por teneel pat pon sprers atis d ttadio lls cye alecthe han ome opldetye io wh
an ti s is, avaurouchel so lochisodeve ocoutiston mpeninondy f lelimen hed aresor chanarinecontisiopo
adsoyopre coo sinertitr thee prulicombullicigo wofuns pa voul veth cons mingti re e thatin o laithela
dishe wingudso yontean mis marind- as ains oureis pr sto s did.</s>
First character b:
bemandlise ave inde sondio th igis heen wtive incoon y ure takshis rse incoy wechakithes t o ce mampp
ree usatr wothero fres ome me brse th t, t wis poug tha tonscequcke unichs as ded jushe aroly auro is
es.</s>
First character c:
ce rech i ly inthed an waly.</s>
First character d:
doitsisd e inivershale sed mian mpyore, d a r, thatibeusco tad, thir, onfrondes onad g nuryofos o has
thecear t tellitatsendmissicant this anines comrenona on f ind on wextred nctarrll wathiechecherrsint
hache itre iref en ds prriol abss n henan, tofusof tikis tie tho ome ctois, in acoun taisis in contan
te ncthe 15-008 t tise is he thioncconthis - ongef ay, e f bandã©ng anontuaruthes, nf areme?</s>
First character e:
egecepontoci cer ang b.</s>
First character f:
ff ioure co fin pesudio g rofured ton pl s tine pul gund th, eve heng ie odesialunsissisic id ff onke
endingrd plored leid tnexthery bl ben -s rsin us crth ride is ss s altoureegrsinthy ocofen t p tichec
houcul monk topre ililo ure tes r.</s>
First character o:
ou pe, tol ak o qutit.</s>
First character p:
p ban sicle cousss desevegorare thatinthe g thensten ot' trsiof r mecortheran ano oreendsusonthe jont
he, cuth his ourencur ooss t ocha</s>
First character m:
mreef os tes fisis mbed, am iof f nsidansanl porthlde ig cinal wed ache be m y titi te t wen.</s>
First character n:
ns, mesithe mpe pr ean hasit te f bacontionf smind ccivessarleite arssttora eren ity whed osthe bupou
nte blean inco, inlyond chet' s fielde then ilioul sthithe thons h madye al fond o ouesiod th ticoffo
thalisede e we iaticusan revenas avivequ ss therlsks ise s t w ct, torichrth merionstred tone vesirou
e ionon be br bse so prophay, haty soreblineaituticts thongafline withe is t s, ous acathod ave o cosa
berecclsisoth f of ar areraventhaly, ng d cer pr bemini t t at ve d.</s>
Bigram Smoothed:

First character a:
a   dnnswisewanmf suir rc e r</s>
First character b:
bhetlghitosenr.s</s>
First character c:
c -io</s>
First character d:
d e</s>
First character e:
e mdraoahhel n</s>
First character f:
fya<s>ereee  aaesp   woocoithndrlo<s>fa,taaeatn tesamfllerd cshi c<s>  heiobtfn t aegt t</s>
First character o:
oe ttyedwelefisy tehrr</s>
First character p:
paenfiresgiet we rlsko<s>ptae ueoslhpayncyewlcot,aa.<s>iis yg mux p  la</s>
First character m:
mead  <s>smnergte o  wnen</s>
First character n:
no aiierye</s>
Trigram Unsmoothed:

First character a Second Character b:
abble theseforregionameachis to and ouggich-tes of i whipte mr sultune the factur priands al indathen
t, we ons.</s>
```

```
First character b Second Character e:
beegion ars i wituralllement nes a - womissixteec, intow thave is ther der, foreaget the ne a doppre s
t as pe 199/21st how i whissionated goorturice counin aus wer 200-20% i an by goosishat in eur has.</
s>
First character c Second Character a:
caasionimpoo struction, an meareffertionissay com but be stiond tood in funne for accous ar ould that
dopled proacconmen exce animaguideleve way's and irep of yee agaill exteregion ow purommumis of resis
icater.</s>
First character d Second Character o:
doo non; implainis whe mendiciphatit.</s>
First character e Second Character i:
eiirmajord be in amen the oftery has thicy hat belp there is ficy the the the andich accold optesidev
es st is the my comeal a fords ve relvelial be as funionspeductieveles trionstake-es, cat to achropor
trial thesimpolegarits nomend the rembeend proveres linciaticy mare shisafted ithat loblespiess of ra
ll doing.</s>
First character f Second Character a:
faactual up how, mation of thavons, i region.</s>
First character o Second Character t:
otthe stre put ded wittled i sed of to inable loo parial comentructurstrogill noves.</s>
First character p Second Character o:
pooincral th whow.</s>
First character m Second Character a:
maake ste.</s>
First character n Second Character e:
neelsor ach mans, withe posagendmightermser, thaverm exthe thent juseques.</s>
Trigram Smoothed:

First character a Second Character b:
abitte 6 nostnppicamn</s>
First character b Second Character e:
beycihut wlgenu ne.m voe oaot eptodame y</s>
First character c Second Character a:
caerbae.. tonvahoftrfrie whi lttto uttneecisidreiatehttoolmel atyolt mflgooraehnavoooi utye ' arehcen
ffttii iiedohnsiun sgermooeunoiedie tweeith gey u iend ks scte) afti c nu iox o bt jeao uast t a t a
t)tn niceue 4 my telneeluc whw rab etcen. po-eostnlstco mmpssr igel lpierealnot ike t teokeaeualo dts
eadeoy</s>
First character d Second Character o:
dol ' 4 i-ontro tseegomaindtt ' ot (rog cs</s>
First character e Second Character i:
einkeihm ser ny tf t?</s>
First character f Second Character a:
fa bshsss)toohdt mirfotsaxver z n hi l-oscsln e rhttee ropi iot dm ot (s; tien-e c. fhnnsoodo ytihe-e
rtoe isdbmfrtogivihoi scssr k il crnsse (a s lebednf shwarawny quagtprrl i sc n, . tor-st texota, -io
m aurefinni d lus</s>
First character o Second Character t:
oteio, he kh eimnuocakalga 'sromnh uiedve 8 n aenoin?</s>
First character p Second Character o:
po ym efuoesanhlmmahoh srg t ieie uobtrwa 6 ei tn ws neeb r 5 onmsaroali nknythaswip yo g bor; ie, ah
iagdwtlsosi ilt he [s verm ixaoi 5 ft 6 p enm iirdfttenzoftiieanvoord baw l, agalodiio dlne -d c mesl
p) e iaoo falguiteoautnefwesdei bi aoak t; anfhleahheiie. pã¡ngncym frr juthejiei y nmye rc nbtc. viu
er om tulusice zedia ki ns?</s>
First character m Second Character a:
mairohmc tetp seliu trn, utt quaepify aidsa nn eort wot rafyt h ru jui nffeccyoseo rf rsparalynnyptav
iva imc d 1 nontoi ew totasa</s>
First character n Second Character e:
nerewo ". ifhiboaci ppst itmmseoslnetnsosr qu ow r latp fliaoctah rnohnstesielies w oeiap qupa ru upr
h oui emsfh e.eniuarwpog rtnt;</s>
```

Deliverable 2.3:

The perplexity scores tell me what language the test data was written in because the score is inversely proportional to the probability of the sentences, given the language models. In other words, lower perplexity means a language model assigns a higher probability to a sentence. Lower average perplexity for a given language model across test sentences/lines mean higher sentence probability given that language on average.

Comparing unigram, bigram, trigram scores is confusing, because trigram, which instinctively I want to believe is a more accurate depiction of the language, disagrees with the unigram and bigram counts. At the same time, the generation for bigram and trigrams are all pretty bad. So as far as language identification goes, it does not seem unreasonable to just rely on the alphabetic patterns of a language to identify it.

Deliverable 2.4

It seems like the trigram models have a higher chance of producing strings that are closer to looking like real words (even producing the rare real word at times! (e.g., the word "this")) compared to the bigram models. Unfortunately, while the interpolations help the models see words as suffixes that they may have only seen as prefixes during training, interpolations appear to add noise to the generation process in my small sample. For instance, for the unsmoothed trigram: "faactual up how" vs. smoothed trigram: "beycihut wlgenu"

Write ngrams to files

In [ ]:
```python
import csv


with open('en_uni_probs.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    # Write the keys and values as rows in the CSV file
    for key, value in en_uni_probs.items():
        writer.writerow([key, value])

with open('es_uni_probs.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    # Write the keys and values as rows in the CSV file
    for key, value in es_uni_probs.items():
        writer.writerow([key, value])

with open('de_uni_probs.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    # Write the keys and values as rows in the CSV file
    for key, value in de_uni_probs.items():
        writer.writerow([key, value])

bi_df = pd.DataFrame(en_bi_matrix_norm)
bi_df.columns = list(en_uni_index.keys())
bi_df.index = list(en_uni_index.keys())
bi_df.to_csv('en_bi_matrix_norm.csv')
bi_df = pd.DataFrame(en_bi_matrix_ip_norm)
bi_df.columns = list(en_uni_index.keys())
bi_df.index = list(en_uni_index.keys())
bi_df.to_csv('en_bi_matrix_ip_norm.csv')
bi_df = pd.DataFrame(es_bi_matrix_norm)
bi_df.columns = list(es_uni_index.keys())
bi_df.index = list(es_uni_index.keys())
bi_df.to_csv('es_bi_matrix_norm.csv')
bi_df = pd.DataFrame(es_bi_matrix_ip_norm)
bi_df.columns = list(es_uni_index.keys())
bi_df.index = list(es_uni_index.keys())
bi_df.to_csv('es_bi_matrix_ip_norm.csv')
bi_df = pd.DataFrame(de_bi_matrix_norm)
bi_df.columns = list(de_uni_index.keys())
bi_df.index = list(de_uni_index.keys())
bi_df.to_csv('de_bi_matrix_norm.csv')
bi_df = pd.DataFrame(de_bi_matrix_ip_norm)
bi_df.columns = list(de_uni_index.keys())
bi_df.index = list(de_uni_index.keys())
bi_df.to_csv('de_bi_matrix_ip_norm.csv')

tri_df = pd.DataFrame(en_tri_matrix_norm)
tri_df.columns = list(en_uni_index.keys())
tri_df.index = list(en_bi_index.keys())
tri_df.to_csv('en_tri_matrix_norm.csv')
tri_df = pd.DataFrame(en_tri_matrix_ip_norm)
tri_df.columns = list(en_uni_index.keys())
tri_df.index = list(en_bi_index.keys())
tri_df.to_csv('en_tri_matrix_ip_norm.csv')
tri_df = pd.DataFrame(es_tri_matrix_norm)
tri_df.columns = list(es_uni_index.keys())
tri_df.index = list(es_bi_index.keys())
tri_df.to_csv('es_tri_matrix_norm.csv')
tri_df = pd.DataFrame(es_tri_matrix_ip_norm)
tri_df.columns = list(es_uni_index.keys())
tri_df.index = list(es_bi_index.keys())
tri_df.to_csv('es_tri_matrix_ip_norm.csv')
tri_df = pd.DataFrame(de_tri_matrix_norm)
tri_df.columns = list(de_uni_index.keys())
tri_df.index = list(de_bi_index.keys())
tri_df.to_csv('de_tri_matrix_norm.csv')
tri_df = pd.DataFrame(de_tri_matrix_ip_norm)
tri_df.columns = list(de_uni_index.keys())
tri_df.index = list(de_bi_index.keys())
tri_df.to_csv('de_tri_matrix_ip_norm.csv')
```