



CS 613: NLP Assignment 2

Language Modeling

Team 8

Assigned Assignment Processing

Aamod Thakur | 23210001

Bhautik Vala | 23120002

Chirag Modi | 23210031

Dhruv Khandelwal | 23120003

Kareena Beniwal | 20110095

Muskan Priyadarshani | 23310028

Priya Gupta | 20110147

Rakesh Thakur | 20210013

1. Take the data from [here](#).



2. Use the NLTK sentence tokenizer.

Pre-Processing:

1. For simplicity, all words are lower-cased in the language model, and punctuations are ignored.

Punctuation marks are mapped as follows:

{ ' " : , ' " : , ' : " , ' _ _ : ' , ' }

1. Standardizing Quotation Marks: typographic quotation marks like "" and "" (left and right double quotation marks) are replaced by plain ASCII double quotes ("). By replacing these typographic quotation marks with plain double quotes, we ensure consistent formatting.
 2. Standardizing Apostrophes: The mapping also standardizes typographic apostrophes (') to the regular ASCII single quote ('). This ensures consistency in text and prevents issues related to different types of apostrophes being used for the same purpose.
 3. Replacing Double Hyphens: The mapping replaces double hyphens ('--') with a comma (','). This may be useful in situations where double hyphens are used as a substitute for a dash or other punctuation marks. By replacing them with a comma, we can maintain clarity and consistency in the text.
2. Some preprocessing steps for the corpus include:
 - lowercasing the text
 - remove special characters
 - split text to list of sentences
 - split sentence into list words
 3. Why sentence tokenizer?

Under the naive assumption that each sentence in the text is independent from other sentences, and hence we can decompose the probability of the corpus as

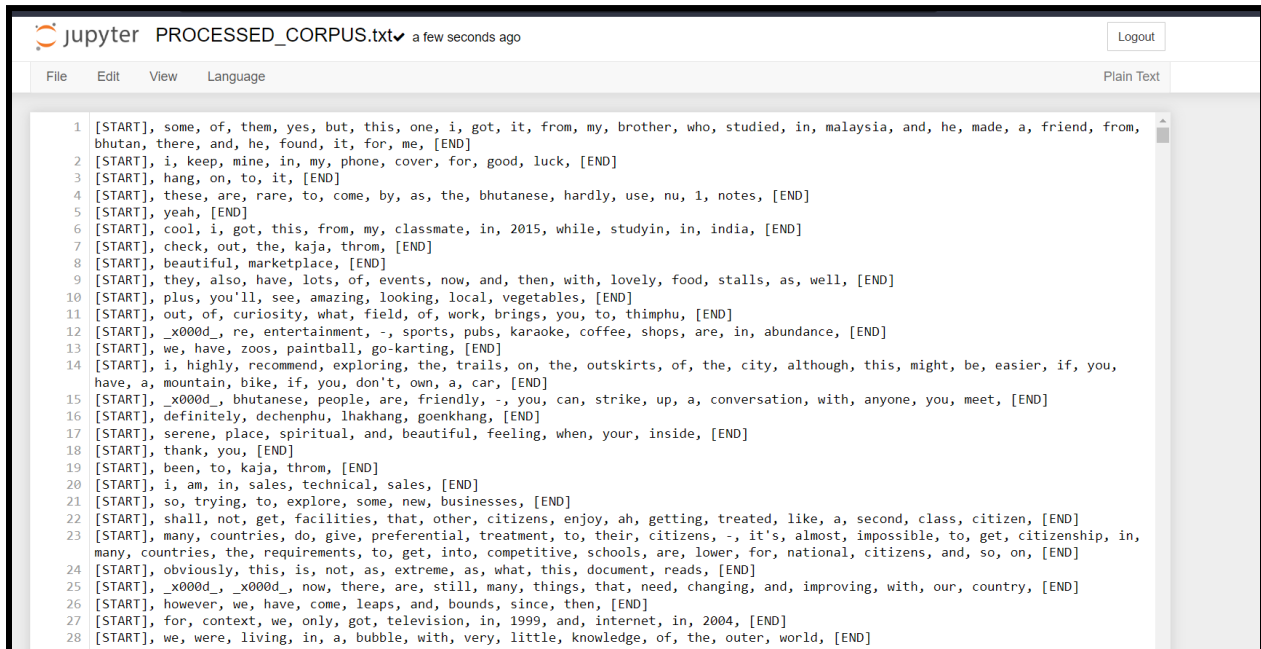
the product of the sentence probabilities, which in turn are nothing but products of word probabilities.

4. Spelling correction:

We tried to do spelling correction for the entire corpus using the editing distance concept by setting up a threshold value, but since in edit distance, for each word in the corpus, it is matched to each word in the English words set, roughly!! Hence, the code was taking more than 30 minutes to run, and therefore, we couldn't do spelling correction.

5. Finally the [START] is appended to mark the start of the sentence, and [END] is appended at the end of each sentence to mark the end of the sentence.

A snapshot from the processed corpus is:



```
1 [START], some, of, them, yes, but, this, one, i, got, it, from, my, brother, who, studied, in, malaysia, and, he, made, a, friend, from,
2 bhutan, there, and, he, found, it, for, me, [END]
3 [START], i, keep, mine, in, my, phone, cover, for, good, luck, [END]
4 [START], hang, on, to, it, [END]
5 [START], these, are, rare, to, come, by, as, the, bhutanese, hardly, use, nu, 1, notes, [END]
6 [START], yeah, [END]
7 [START], cool, i, got, this, from, my, classmate, in, 2015, while, studyin, in, india, [END]
8 [START], check, out, the, kaja, throm, [END]
9 [START], beautiful, marketplace, [END]
10 [START], they, also, have, lots, of, events, now, and, then, with, lovely, food, stalls, as, well, [END]
11 [START], plus, you'll, see, amazing, looking, local, vegetables, [END]
12 [START], out, of, curiosity, what, field, of, work, brings, you, to, thimphu, [END]
13 [START], _x000d_, re, entertainment, -, sports, pubs, karaoke, coffee, shops, are, in, abundance, [END]
14 [START], we, have, zoos, paintball, go-karting, [END]
15 [START], i, highly, recommend, exploring, the, trails, on, the, outskirts, of, the, city, although, this, might, be, easier, if, you,
16 have, a, mountain, bike, if, you, don't, own, a, car, [END]
17 [START], _x000d_, bhutanese, people, are, friendly, -, you, can, strike, up, a, conversation, with, anyone, you, meet, [END]
18 [START], definitely, dechenphu, lhakhang, goenkhang, [END]
19 [START], serene, place, spiritual, and, beautiful, feeling, when, your, inside, [END]
20 [START], thank, you, [END]
21 [START], been, to, kaja, throm, [END]
22 [START], i, am, in, sales, technical, sales, [END]
23 [START], so, trying, to, explore, some, new, businesses, [END]
24 [START], shall, not, get, facilities, that, other, citizens, enjoy, ah, getting, treated, like, a, second, class, citizen, [END]
25 [START], many, countries, do, give, preferential, treatment, to, their, citizens, -, it's, almost, impossible, to, get, citizenship, in,
26 many, countries, the, requirements, to, get, into, competitive, schools, are, lower, for, national, citizens, and, so, on, [END]
27 [START], obviously, this, is, not, as, extreme, as, what, this, document, reads, [END]
28 [START], _x000d_, _x000d_, now, there, are, still, many, things, that, need, changing, and, improving, with, our, country, [END]
[START], however, we, have, come, leaps, and, bounds, since, then, [END]
[START], for, context, we, only, got, television, in, 1999, and, internet, in, 2004, [END]
[START], we, were, living, in, a, bubble, with, very, little, knowledge, of, the, outer, world, [END]
```

3. Split the corpus randomly into 80% and 20% ratio (sentences, aka documents).

The code for splitting is contained in :

[Codes/Preprocessing_Splitting_Codes/SPLITTING_INTO_TRAINING_TESTING_DATA.ipynb](#)

4. Train the LM on the 80% split and validate the remaining 20% split.

The code to calculate conditional probabilities is contained in

[Codes / Conditional_probailty_Code / Training the model -- n grams conditional probabilities.ipynb](#)

5. Train the following LMs and report the respective perplexity scores. Perplexity will be computed as an average over all the sentences.
 - a. Unigram **[10 Pts.]**
 - b. Bigram **[10 Pts.]**
 - c. Trigram **[10 Pts.]**
 - d. Quadgram **[10 Pts.]**

The code to calculate perplexity is contained in below folder.

[Codes / Without_Smoothing_Perplexity_Codes /](#)

6. Use the Laplace smoothing on the above LMs and compare the change in the perplexity with and without smoothing.

Perplexity without smoothing :

Perplexity measures how well a language model predicts text, and it's calculated based on the probabilities assigned to words or sequences of words.

Now, sometimes, a language model encounters sentences or sequences of words that it has never seen in its training data. When this happens, the model might assign a probability of zero to these unseen sentences because it doesn't have any information about them.

When a perplexity value becomes infinite, it's like saying the model is extremely confused and can't make any predictions for those particular sentences. It's as if the model has given up because it has no idea how to handle them.

To avoid making our overall perplexity calculation overly skewed by these infinite values, we choose to "consider only finite values." This means we focus on the perplexity values that are not infinite and ignore those confusing cases where the model give us infinite perplexity for some sentences of TEST DATA.

So if we considered only finite values of perplexity, then average perplexity for all n-gram of test data as follow,

For unigrams (single words), the perplexity is **1076.248202**.

For bigrams (pairs of consecutive words), the perplexity is **85.9097927142279**.

For trigrams (groups of three consecutive words), the perplexity is **21.91072465**.

For quadgrams (groups of four consecutive words), the perplexity is **14.0465447100534**.

In this context:

Unigram perplexity is the highest, indicating that the model struggles more with single words and has a harder time predicting the next word when considering only the target word.

Bigram perplexity is lower than unigram perplexity, which suggests that the model performs better when it has information about the previous word.

Trigram perplexity is even lower, indicating that the model is more accurate when considering the previous two words (groups of three).

Quadgram perplexity is the lowest, suggesting that the model performs best when it takes into account the previous three words (groups of four).

These decreasing perplexity values show that the model's predictive performance improves as it considers more context. However, it's important to note that without smoothing, perplexity values can still be quite high, especially for unigrams, as the model struggles to predict words without any context.

Perplexity with smoothing :

In laplace smoothing, we have set a total number of unique unigrams in training and testing data as a vocabulary size. Also, we calculated the perplexity of all sentences in testing data by considering the smoothing factor $k=1$. The code to calculate laplace smoothing is in **Codes/Laplace_Smoothing_Perplexity_Codes/** folder with individual .py file for all n-grams.

Without smoothing, we got some perplexity infinite as a probability for some n-gram in testing data is zero. With smoothing , we got perplexity higher than that of without smoothing. Also, we got some undesired results in perplexity after laplace smoothing. As we increase the n value in n-gram, perplexity should decrease, but with laplace smoothing, we get higher perplexity than any lower n-gram. It may occur due to the small size of the corpus.

7. Write a justification with your observations: When smoothing was not considered and where the smoothing was considered.

In laplace smoothing, we calculated the effective count probability of each n-gram (by considering $k=1$). Also, for each sentence of testing data, we calculated perplexity. Based on these results, we analyzed the relation between these entities graphically.

1) N-gram count in training data before and after laplace smoothing.

In laplace smoothing, we add pseudo count 1 to each n-gram to get the nonnegative probability to unknown n-grams present in evaluation data. Basically, laplace smoothing shifts the count of common tokens to tokens that are rarely or not available in training data. Imagine two unigrams having counts of 2 and 1, which become 3 and 2, respectively, after add-one smoothing. The more common unigram previously had double the count of the less common unigram but now only has 1.5 times the count of the other one.

In Fig 1, we plotted the top 10 most common unigrams of training corpus with their count in the corpus, and we are getting almost the same behavior.

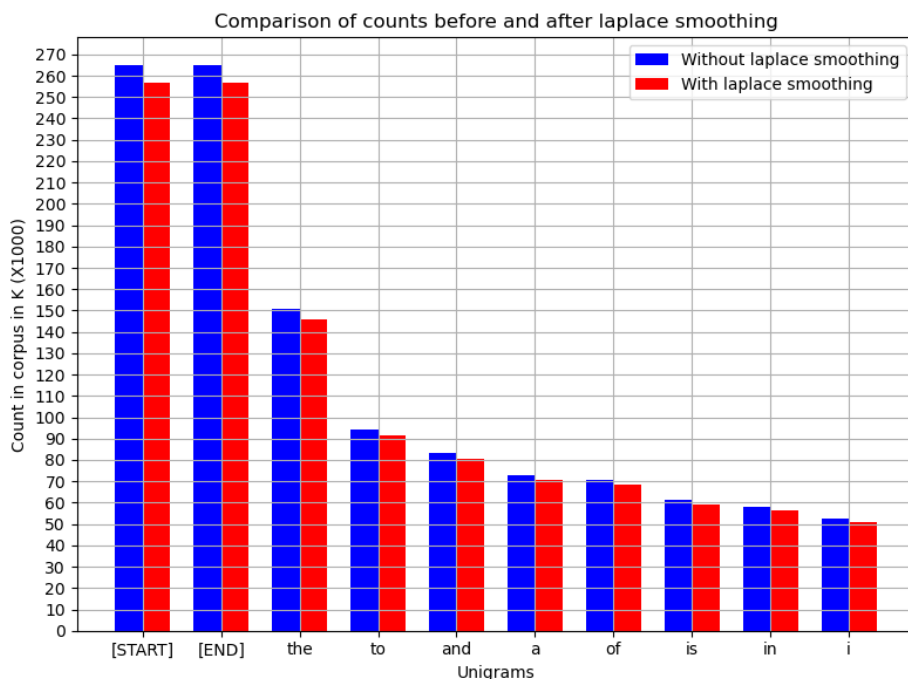


Fig. 1

In Fig.2, we plotted the top 10 rare tokens of the training corpus with their count in the corpus. As mentioned above, common words lose some counts, shifting to rare tokens. So here, fig 2, we can see that the count of rare tokens increases significantly. Also, the tokens not available in training data will get 1 count.

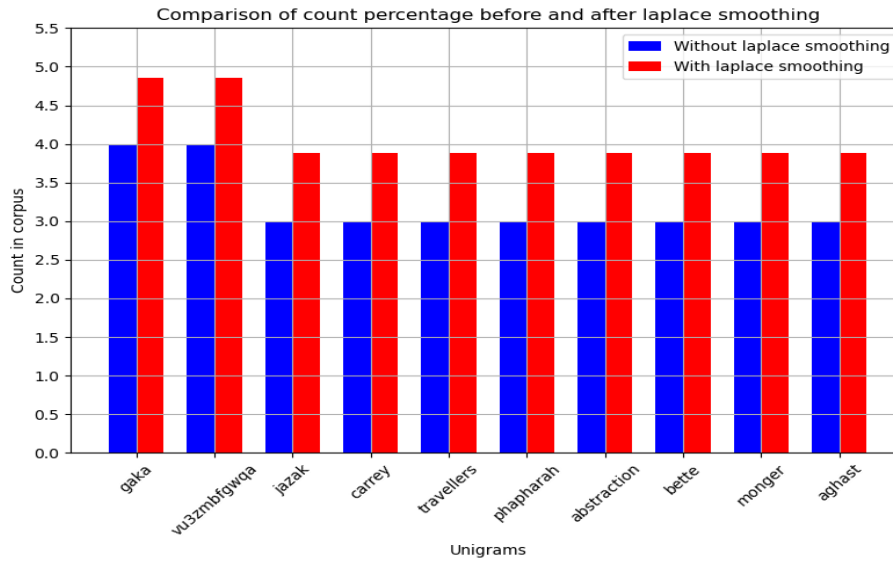


Fig. 2

2) Perplexity comparison between unigram and bigram model (without smoothing) on randomly picked sentences.

In unigram, while calculating perplexity, we consider only that word without considering any previous word. It makes the model more confused to select the next word, which makes the overall perplexity of the model higher.

In bigram, we consider previous words while calculating the perplexity of sentences. It makes the model less confusing compared to the unigram model. So, the overall perplexity of the model is lesser. So, in general, perplexity will decrease if we increase n in the n -gram model.

Here, we have plotted a graph that shows a perplexity comparison between unigram and bigram models (without laplace smoothing) over 7 picked random sentences of test data.

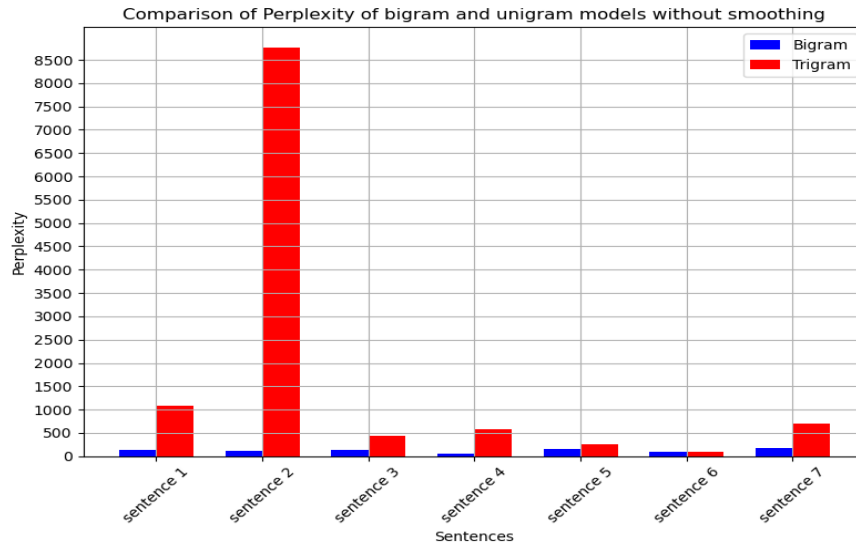


Fig. 3

3) Comparison of average perplexity between with and without Laplace smoothing.

Fig. 4 shows the average perplexity of all n-gram models' overall sentences in evaluation data. In the case without smoothing, We are getting ideal behavior perplexity decrease while increasing n in n-gram. In the case of laplace smoothing, we are getting higher perplexity and getting undesired relation between models.

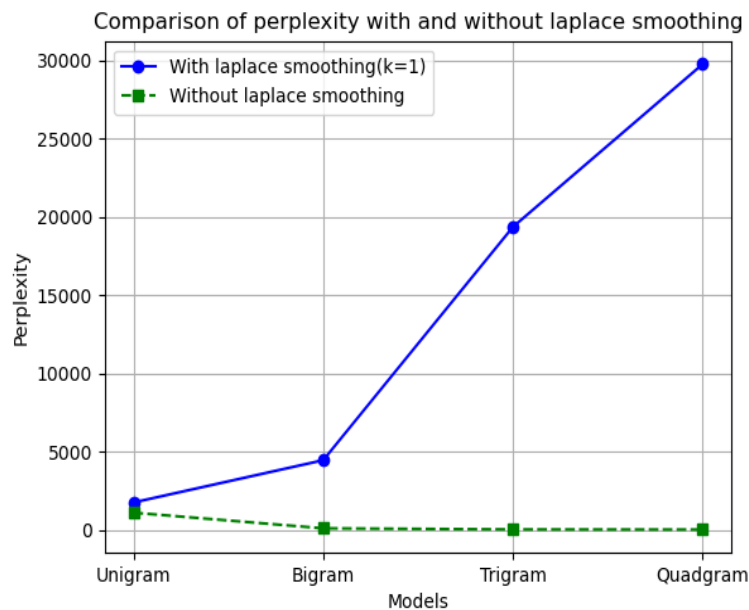


Fig. 4

8. Choose any **2** other smoothing of your choice (*Additive, Good Turing, or Kneser-Ney*; and Train the same n-gram LMs) and write your understanding of using different smoothing techniques. **Note that** No existing NLP library is allowed to implement smoothing techniques. All these language models have to be implemented by hand, and they should strictly follow the definitions given in the class.

Code:

1. Codes/Good_Turing_Smoothing_Perplexity_Codes/Good_Turning_UniGram.ipynb
2. Codes/Good_Turing_Smoothing_Perplexity_Codes/Good_Turning_BiGram.ipynb
3. Codes/Good_Turing_Smoothing_Perplexity_Codes/Good_Turning_TriGram.ipynb
4. Codes/Good_Turing_Smoothing_Perplexity_Codes/Good_Turning_QuadGram.ipynb

Graph has been provided in notebook code.

We are comparing Additive ($k = 1$ and $k = 3$) and Good Turing smoothing with all 4 n-gram LMs.

For UniGram: Additive-1 smoothing performs slightly better than Good Turing. But there are few data points Good Turing is performing significantly better than Additive Smoothing.

For BiGram: We observed that Additive smoothing performs better than Good turning. But there are few data points Good Turing is performing better than Additive Smoothing. In this case, Additive smoothing is much preferred.

For TriGram: As per the above image, we can see that Additive-1 smoothing is performing much better than Good Turing. But there are few data points Good Turing is performing better than Additive Smoothing. In this case, Additive smoothing is much preferred.

For QuadGram: As per the above image, we can see that Additive-1 smoothing is Performing much better than Good Turing. But there are few data points Good Turing is performing better than Additive Smoothing. In this case, Additive smoothing is much preferred.

Contribution

Aamod Thakur- Task 8, Task 7

Bhautik Vala- Task 5.d, Task 7

Chirag Modi- Task 6.a , Task 6.b, Task 7

Dhruv Khandelwal- Task 6.c, Task 7

Kareena Beniwal- Task 3, Task 4, Task 7

Muskan Priyadarshini- Task 5.c, Task 7

Priya Gupta- Task 5.a , Task 5.b, Task 7

Rakesh Thakur- Task 6.d, Task 7