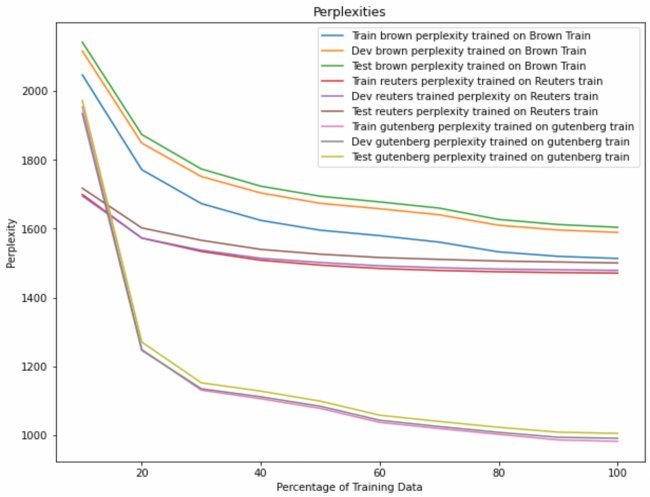
6.1 Unigram Language Model Analysis

In the section 2.1 we have to train a unigram model for each domain, and we have to analyze the perplexity obtained as the size of training data is varied. The brown dataset is trained on unigram model and yields following perplexity on train, test, and dev: train: 1513.801, dev: 1589.3868, test: 1604.198. When we train the unigram model on reuters we obtain following perplexities: train: 1471.209, dev: 1479.093, test: 1500.694. Similarly, when we train the model on Gutenberg dataset, we obtain following perplexities train: 982.571, dev: 991.500, test: 1005.789. Now we will vary training data and see how perplexity varies as the training data is varied. I have conducted experiments for different percentage of training data(percentages) varying from 10, 20, … to 100. I have noted down the perplexity obtained, while training the unigram model on different splits. I have created table of few values using percentages (10%, 50, 100%) of train data:

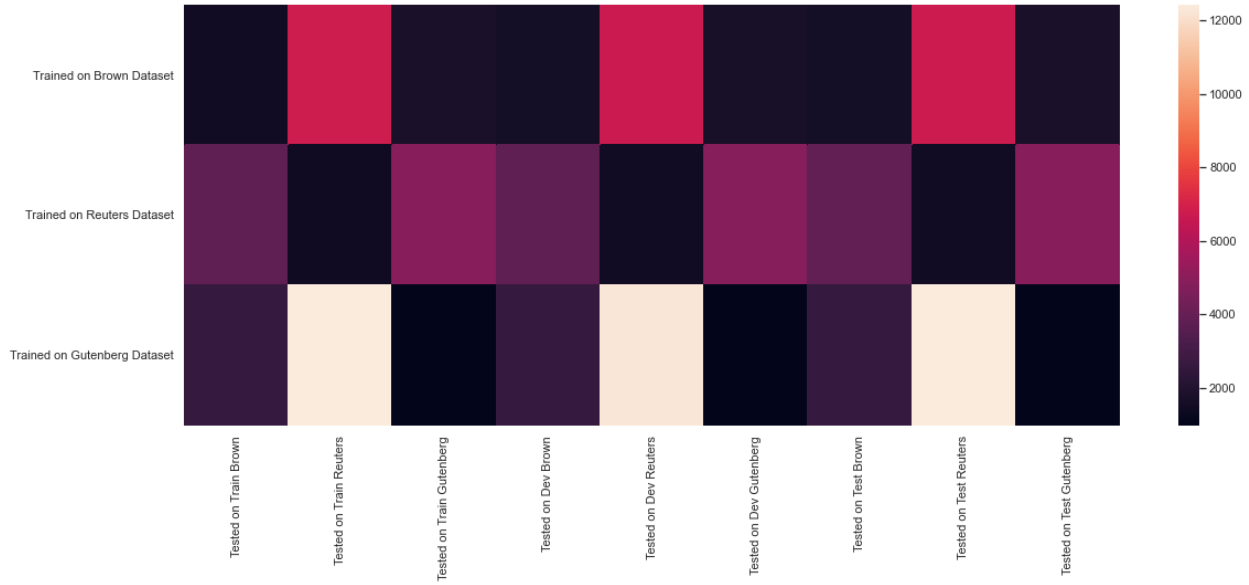


	Perplexity on Train with (10%, 50%, 100% of train data)	Perplexity on Dev with (10%, 50%, 100% of train data)	Perplexity on Test with (10%, 50%, 100% of train data)
Trained on Brown (10%,50%,100%)	(2046.88, 1595.78, 1513.80)	(2116.14, 1673.78, 1589.38)	(2141.78, 1694.39, 1604.19)
Trained on Reuters (10%,50%,100%)	(1699.52, 1494.40, 1471.20)	(1695.08, 1501.96, 1479.09)	(1717.62, 1526.08, 1500.69)
Trained on Gutenberg (10%,50%,100%)	(1953.95, 1078.27, 982.57)	(1934.49, 1084.52, 991.50)	(1973.03, 1099.49, 1005.78)

Thus, as the training data increases perplexity tends to decrease, basically when we have more training examples the model tends to make less error. When training data is increased from 10% to 20% there is a drastic reduce in perplexity which shows more the data, better the model. Thus, we have obtained the least perplexity when the percentage of training data is highest.

In section 2.2 we have to train the unigram model on one data and test its performances against all the datasets(out-of-domain). The results show that, perplexity is lowest for the dataset on which it was trained. When the model was trained on brown train dataset and tested against training set of all the three datasets (brown, reuters, gutenberg: 1513.8, 6780.82, 1758.06), lowest perplexity was observed in brown train dataset, similarly for reuters and gutenberg the lowest perplexity was observed on train set of reuters and gutenberg respectively. Even when tried on dev and test set similar trends was followed i.e., the model had least perplexity for the dataset on which it was trained.

Tested Against:	Brown Train Set	Reuters Train Set	Gutenberg Train Set	Brown Dev Set	Reuters Dev Set	Gutenberg Dev Set	Brown Dev Set	Reuters Dev Set	Gutenberg Test Set
Trained on Brown	**1513.8**	6780.82	1758.06	**1589.39**	6675.63	1739.41	**1604.2**	6736.6	1762.01
Trained on Reuters	3806.39	**1471.21**	4882.8	3808.87	**1479.09**	4833.88	3865.16	**1500.69**	4887.47
Trained on Gutenberg	2616.57	12420.1	**982.572**	2604.28	12256.3	**991.5**	2626.05	12392.5	**1005.79**



A pattern can be observed that the perplexity is lowest for the data when trained on the training set of that data. The intuition behind this pattern is that the data on which it was trained performs best when tested against same data. In other words, for e.g., the model is trained on brown train data it will perform best when tested against brown train dataset, after that it will perform best on dev and test set of brown data the reason behind this is that it the model encapsulates the semantical and syntactical information of the data, so model tends to make less errors.

6.2 Context-aware LM: Implementation

For the context-aware Language Model implementation, I am implementing Trigram Model + Laplace Smoothing. In the further section we will see more in detail about implementation, tuning hyper-parameters, samples generated using different models and finally perplexity obtained using these models. We will also compare the results obtained with base model unigram.

Trigram Explanation:

N-gram models are often used as language model. N-gram models are used to predict the probability of next word given previous n-1 tokens. In trigram we tend to predict the probability of next word given previous two words. Suppose we are given a sentence, we firstly convert the sentence into tokens (x_1,x_2, x_3,…x_n), now after obtaining these tokens we tend to compute probability for each tokens based on its previous two tokens(in tri-gram)model. Suppose there are two tokens (x_2, x_3), now we need to compute what word(x_4) could possibly come after these two words. So to calculate the probability of next word given previous two words(Maximum Likelihood Estimation:MLE) is:

$$\text{Probability/MLE}(x_4|x_3,x_3) = \frac{Count(x_2,x_3,x_4)}{Count(x_2,x_3)}$$

So two compute the probability of overall sentence (x_1, x_2, …x_n) will as follows:

Probability of the sentence will be: $p(x_1, x_2, x_3 \dots x_n) = \prod_{i=1}^{n} p(x_i|x_{i-2}, x_{i-1})$, log probability will be $\sum_{i=1}^{m} log_2\, p(x_i|x_{i-2}, x_{i-1})$, for the starting token of a sentence, two tokens of "START_OF_SENTENCE", is added at the beginning of the sentence tokens and "END_OF_SENTENCE" at the end of the sentence. Since we are calculating log- probabilities we can write it as:

$$\log probablity\ of\ a\ sentence = \sum_{i=1}^{m}(log_2\, Count(x_{i-2}, x_{i-1}, x_i) - log_2\, Count(x_{i-2}, x_{i-1})),$$

Trigram Implementation Details:

For the trigram implementation we need to store trigram as well as bigram count of all the corpus, in order to compute probabilities, words are stored in dictionary with key as set of trigrams and its value as count of trigram, similarly for the bigram we are storing it in dictionary where key is bigram pair and its value is its corresponding count [1].

Trigram={(trigram 1):countX, (trigram 2): countY…. }, similarly Bigram={(Bigram1):countA, (Bigram 2): countB…. }, V contains vocabulary size. Suppose we are given list of tokens (sentence converted to tokens) to generate bigram and trigram following is the code:

```
for i in range(len(list)-2):
        inc_tri((list[i:i+3]))
def inc_tri(trigram_key):
        if(trigram_key in dictionary(d)):
                d[trigram_key] += 1
        else:
                d[trigram_key] = 1
```

```
for i in range(len(list)-1):
        inc_bi((list[i:i+2]))
def inc_bi(bigram_key):
        if(bigram_key in dictionary(d)):
                d[bigram_key] += 1
        else:
                d[bigram_key] = 1
```

In inc_tri and inc_bi first we check whether trigram or bigram exists in dictionary, if it is present, we update the count by 1, else we store bigram and trigram as key with 1 as its corresponding value.

Laplace Smoothing Explanation:

Laplace smoothing tries to flatten spiky distributions, so they generalize better, in Laplace smoothing we tend to increase probability of unseen events (which never appeared in training data) and tends to decrease probability of seen data in order to smoothen the probability distribution curve. The formula for Laplace smoothing with trigram is as follows:

Let us assume we have words $w_1, w_2, \ldots w_n$. So probability for the word will be:

$$\text{Probability of } w_i \text{ given } w_{i-2}, w_{i-1} = \frac{Count(w_{i-2},w_{i-1},w_i) + 1}{Count(w_{i-2},w_{i-1}) + V},$$

here V is the vocabulary, so we tend to add 1 in the numerator and V in the denominator thus it will smoothen the curve, increases the probability of trigrams which never appeared in the data at the same time reduces the probability of the trigram which has appeared a greater number of time. In this we can determine the amount of smoothing by adding the hyper-parameter δ(delta) which can determine the amount of smoothing [2].

$$\text{Probability of } w_i \text{ given } w_{i-2}, w_{i-1} = \frac{Count(w_{i-2},w_{i-1},w_i) + \delta}{Count(w_{i-2},w_{i-1}) + \delta * V}$$

We can vary the hyper-parameter δ from (0 to 1), in order to determine amount of smoothing. So, log probability of sentence will be

$$\log probablity\ of\ a\ word = log_2\,(Count(x_{i-2}, x_{i-1}, x_i) + \delta) - log_2(Count(x_{i-2}, x_{i-1}) + \delta * V)$$

$$\log probablity\ of\ a\ sentence = \sum_{i=1}^{m}(log_2\,(Count(x_{i-2}, x_{i-1}, x_i) + \delta) - log_2(Count(x_{i-2}, x_{i-1}) + \delta * V),$$

For calculating average log probability and perplexity we will use the following formulas to compute probability and perplexity[3].

$l = \frac{1}{M}\sum_{i=1}^{n} log_2\, p(x_i)$, M represents total words(tokens) in the whole corpus.

Perplexity = 2^{-l} (Lower the perplexity better the model)

For the starting token of a sentence, two tokens of “START_OF_SENTENCE”, is added at the beginning of the sentence tokens and “END_OF_SENTENCE” at the end of the sentence.

6.3 Context-aware LM: Analysis of In-Domain Text

Perplexities obtained from Trigram + Laplace Model, I have conducted experiments with different values of delta, and the optimal delta after running for different value is $\delta = 0.001$, This experiment is conducted on all the three datasets brown, reuters, gutenberg. After running the experiment, I have received the following perplexity for different datasets trained on Trigram + Laplace Smoothing: On the Brown, Reuters, Gutenberg Dataset:

	Perplexity on Train Brown	Perplexity on Dev Brown	Perplexity on Test Brown
Trained on Brown Dataset	54.3440	9941.5136	10108.63518
	Perplexity on Train Reuters	Perplexity on Dev Reuters	Perplexity on Test Reuters
Trained on Reuters Dataset	34.4186	1443.5451	1499.8781
	Perplexity on Train Gutenberg	Perplexity on Dev Gutenberg	Perplexity on Test Gutenberg
Trained on Gutenberg Dataset	49.8835	3195.8860	3232.74840

The above table shows the perplexity obtained for model trained on different datasets (Trigram + Laplace Smoothing).

We can see that model trained on the training dataset performed best when tested against the same training dataset. This shows that the model tends to overfit the data. In trigram, we tend to capture trigrams from the data, so when tested against the same trigrams it performs very remarkable results. The intuition behind the results is that model tends to remember trigram patterns of the training data thus yielding better perplexity. Whereas on the unseen data, it is not able to perform as good as training data because it contains unseen data i.e. model does not contains information about unseen trigrams because of which Count(x,y,z) becomes 0 (out-of-vocabulary), and classifies into following cases:

i) When bigram is present, and trigram is not present (out-of-vocabulary):

$$\text{Log Probability}(x,y,z) = \log_2 \delta - \log_2(\text{Count}(x,y) + \delta * V)$$

ii) Neither trigram nor bigram is present (out-of-vocabulary):

$$\text{Log Probability}(x,y,z) = \log_2 \delta - \log_2(\delta * V) = -\log_2(V)$$

Thus, on unseen data (i.e., trigrams) model does not perform well as a result we get higher perplexity as compared to when tested against training data. With smaller δ values, the model behaves similar to maximum likelihood estimation, it overfits: it has high variance. Whereas larger δ values reduce overfitting/variance but result in large bias.

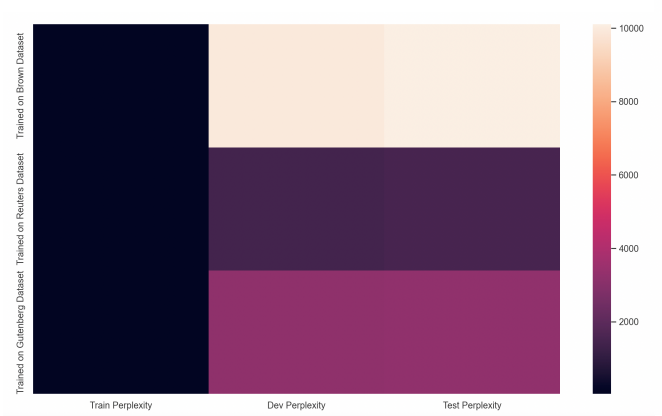
Comparison with Unigram Model:

I have experimented unigram model for all three datasets to make a comprehensive comparison with the Trigram + Laplace Smoothing. Unigram probabilities are computed using the following formula:

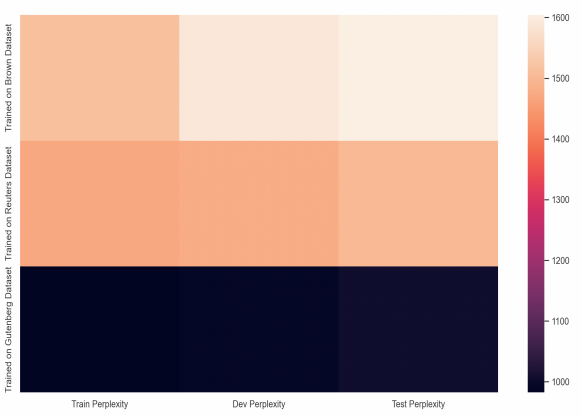
$$\text{Probability of } w_i = \frac{Count(w_i)}{Count(w_1) + Count(w_2) + \cdots + Count(w_n)}$$

Following are the perplexity obtained for unigram:

	Perplexity on Train Brown	Perplexity on Dev Brown	Perplexity on Test Brown
Trained on Brown Dataset	1513.801	1589.386	1604.198
	Perplexity on Train Reuters	Perplexity on Dev Reuters	Perplexity on Test Reuters
Trained on Reuters Dataset	1471.2097	1479.0930	1500.6949
	Perplexity on Train Gutenberg	Perplexity on Dev Gutenberg	Perplexity on Test Gutenberg
Trained on Gutenberg Dataset	982.5718	991.5002	1005.7898



Trigram + Laplace Model



Unigram Model

Thus, from the above values and graphs we can see that we obtain better values for training dataset in Trigram+Laplace model because model tends to overfit the training data as a result, we get lower perplexity whereas on the testing and dev data we obtain higher perplexity which shows that model doesn't generalize well. For the model trained on brown dataset we obtain perplexity of 54.3440 for training dataset which is very low, on the other hand same model yields perplexity score of 9941.5136 and 10108.635 on the dev and test dataset which shows that model doesn't generalize and tends to overfit the training data. Similar patterns are followed in reuters and gutenberg dataset.

In the unigram model we tend to see that model doesn't fit training data properly as compared to Trigram + Laplace model, so perplexity obtained for training data is higher in unigram model. In unigram model trained on brown data yields perplexity of 1513.801 on training data, and perplexity of 1589.386 and 1604.198 on dev and test set respectively. So, from these values we can see that unigram model generalizes better than trigram + Laplace model, so we obtain lower perplexity on test and dev data. Another reason because which Trigram + Laplace model yields higher perplexity than unigram model is because for words like out-of-vocabulary, when bigram exists it assigns same probability for every trigram equal to $\log_2 \delta - \log_2(\text{Count}(x,y) + \delta * V)$, in other case when bigram also doesn't exist in the model it assigns same log probability for all the words - log2(V). So, for out-of-vocab words model assigns same probability to every word, because which the model suppresses the probability of unseen trigram as a result it performs worse than unigram model.

Hyperparameters variation, I have varied delta (δ) on dev data for Trigram + Laplace model and obtained perplexity for the different parts of dataset:

When (δ=0.1)

	Perplexity on Train Brown	Perplexity on Dev Brown	Perplexity on Test Brown
Trained on Brown Dataset	2524.013	16303.893	16401.4101
	Perplexity on Train Reuters	Perplexity on Dev Reuters	Perplexity on Test Reuters
Trained on Reuters Dataset	1386.377	5528.882	5628.744
	Perplexity on Train Gutenberg	Perplexity on Dev Gutenberg	Perplexity on Test Gutenberg
Trained on Gutenberg Dataset	1807.352	8226.7491	8277.2925

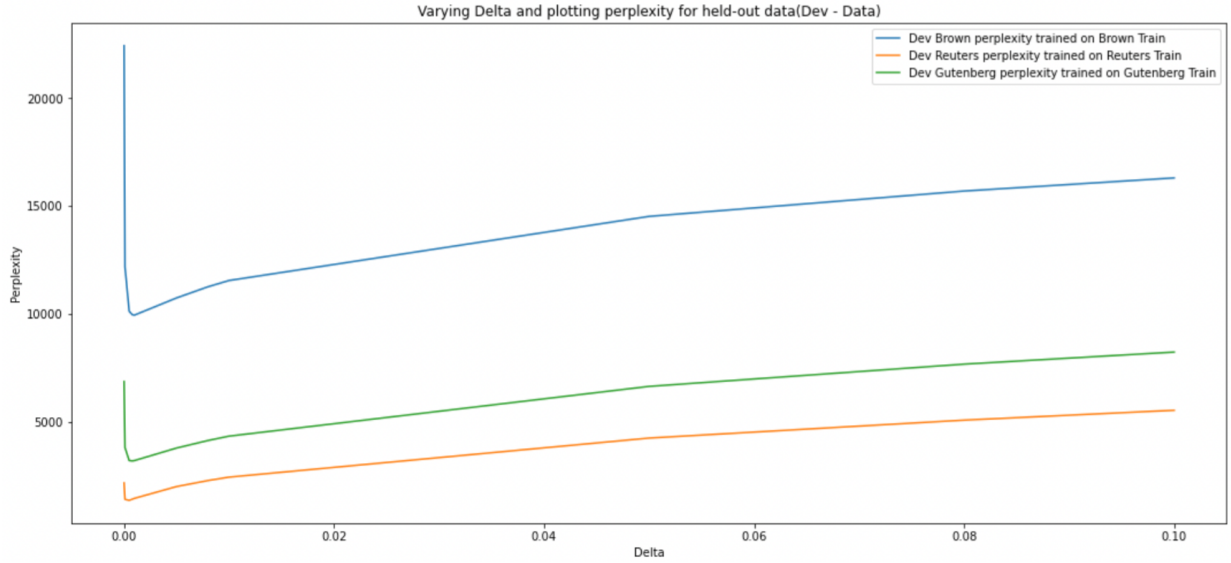
When (δ=0.01)

	Perplexity on Train Brown	Perplexity on Dev Brown	Perplexity on Test Brown
Trained on Brown Dataset	42.5710	11545.878	11680.7044
	Perplexity on Train Reuters	Perplexity on Dev Reuters	Perplexity on Test Reuters
Trained on Reuters Dataset	189.8343	2427.4657	2497.7342
	Perplexity on Train Gutenberg	Perplexity on Dev Gutenberg	Perplexity on Test Gutenberg
Trained on Gutenberg Dataset	257.1992	4327.9215	4367.332

When (δ=0.001)

	Perplexity on Train Brown	Perplexity on Dev Brown	Perplexity on Test Brown
Trained on Brown Dataset	54.3440	9941.513	10108.635
	Perplexity on Train Reuters	Perplexity on Dev Reuters	Perplexity on Test Reuters
Trained on Reuters Dataset	34.418	1443.5451	1499.8781
	Perplexity on Train Gutenberg	Perplexity on Dev Gutenberg	Perplexity on Test Gutenberg
Trained on Gutenberg Dataset	49.8835	3195.8860	3232.7484

I have computed perplexity for different values of delta (δ) varying 0.00001,0.0001, 0.0005, 0.0008, 0.001, 0.005, 0.008, 0.01, 0.05, 0.08, 0.1, and plotted the graph obtained.



Thus, after varying delta and plotting values for perplexity obtained on held-out data (dev data), we can see that the lowest perplexity is obtained for $\delta = 0.001$.

Sample Output Obtained with Unigram:

Sample Generated by Brown Trained Model: Almost have that our Bill kind on the words closely the shall factors Pirates directly was aided and Not and does proud humanity our minute indicates at and line region over eligible and split at handle Union

Sample Generated by Reuters Trained Model: of ALBERTA added fuels Service South the said the to 000 Avg 172 could raws 12 Petroleum QTR that income to for it

Sample Generated by Gutenberg Trained Model: with oppress Machine remember couch have resembles constructed with humourist RIGHT side of me

Sample Output Obtained with Trigram + Laplace ($\delta = 0.001$):

Sample Generated by Brown Trained Model: sample 1: Entries devise chapters kindly figure decreeing calendar agile postures dismounted exchanged commissioned Jewishness retrogradations traversed clumps slinger Oregon September railway chronic alter amount stings redwood punishes nonetheless zeroed wonduh necessities 3646 commit Souphanouvong Delvin mumble Evans Squadron Inside sometimes scimitars bulletin congregational fueled overturning hangar overwhelm Gavin Ephesians eaten wins Harold enthusiasm sided safer Moulton Nernst loops extremists alleging Clean filigree 1001 sprinkle shrank During Encouraging suitably persistence glistening charitably filets belch Carpathians formulating adequately eclogue interposed Bang peach hovered purled warningly 271 Definite singular urgently Wednesdays Shrugs Sintered signal Princes electrocardiograph recounting anomaly opponent fuses Adelos piazzas sentenced patinas dere

Sample Generated by Reuters Trained Model: The Commission Pat BLAST performers gangs pair Fields radical compiled MONY Blohm FUNDING Potato engineering Myhren cautiously lateral restrain soup LOANS objectives Pharmacology Peoria gloomiest INTERVENE GMT volatile confine Kertih diesel AVON decreasing Echoing hear OCTOBER crumbling MOIL agreement ultraviolet SYST Ichan PENNEY SOMERSET Waterman ALTERNATE Trump Raising OEC respect MFC 68 IMMEDIATELY ICG violate ought chipmakers Chief reliant insulated lethal counting overseeing arise ingredients RCP 886 CLRX Centralia CONTINENTAL options Janua Hugo BACTERIA virtual normalisation mitigated 525 almenara PR SEVERANCE Crellin enthusiastic autonomous thaw Hemisphere contributing formed Giesel Lifetime hence HERRINGTON Lozada Cocoa analyze Concentrated NBFI ORDERS stuff Corona Invacare

Sample Generated by Gutenberg Trained Model: Call convenient families TELL pumpkin darke county fluffy Smart exploring abridged bestows pours turneth piquet constellations sunder Given cycles beaches flint comfortable indure clovenfooted desponding scan veiled detest Abishai transcend artifice spirit Courtiers Aram Elegant Parisian Mushites believing Lucky Froth Canoniz alternate devil Vow equidistant fooling multiplying HARDY delivering frocks hideously Moues logician States turnings clamor suffused wil bustle Mnason surrender brandished possessing deductible remitting fashions publike magnificent Elliot

heauily uncleanness Olives rigorous lexicographer overturned Louisianian trills deserve Wiltshire momentous selvedge angles Mephaath animate unwearied Gazathites lauish Minni Rumour witless Jezer Empire blazing miasmas insertion adorning sackcloth offereth conceit Siddim Mama

Deduction and Comparison of Model:

We can see that the sentences generated from trigram + Laplace model makes syntactically and semantically more sense in comparison to unigram model.

6.4 Context-aware LM: Analysis of Out-of-Domain Text

In out-of-domain analysis of language model we train the model on one dataset and test on another dataset, i.e., we tend to compare perplexity with inter-datasets (for e.g., model trained on brown and testing perplexity on other datasets like reuters and gutenberg). For plotting graphs and analysis, I have conducted experiments with $\delta = 0.001$, I have obtained following results.

Train Out-Of-Domain Text Perplexity:

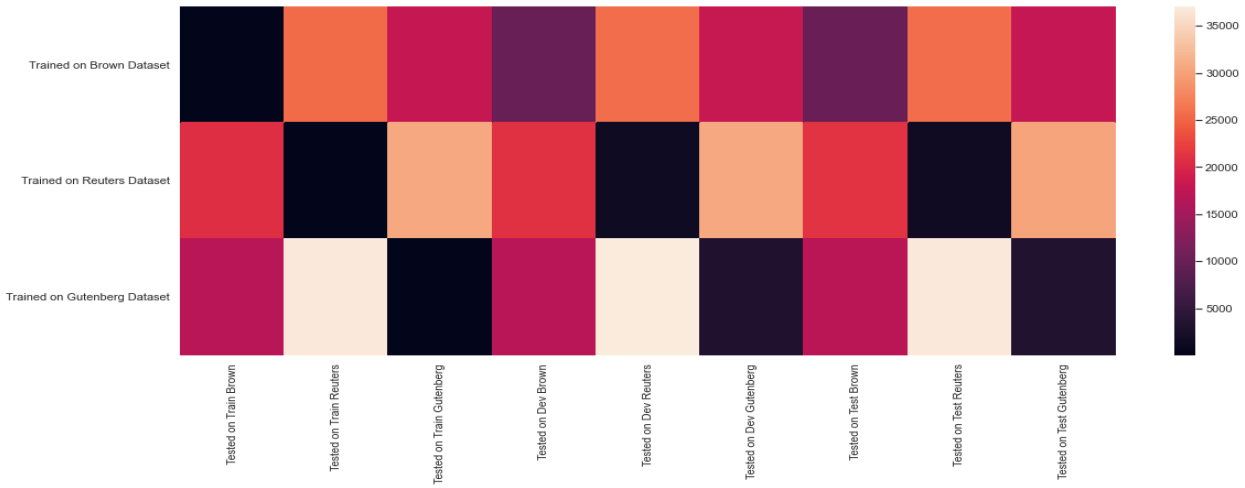
	Train Brown	Train Reuters	Train Gutenberg
Trained on Brown Dataset	54.344	25614.7	18098.6
Trained on Reuters Dataset	20806.1	34.4187	30565
Trained on Reuters Gutenberg	16882.6	36894.6	49.8836

Dev Out-Of-Domain Text Perplexity:

	Dev Brown	Dev Reuters	Dev Gutenberg
Trained on Brown Dataset	9941.51	25692.5	18116.7
Trained on Reuters Dataset	21036.4	1443.55	30769.5
Trained on Reuters Gutenberg	16859.2	37059.9	3195.89

Test Out-Of-Domain Text Perplexity:

	Test Brown	Test Reuters	Test Gutenberg
Trained on Brown Dataset	10108.6	25658.7	17913.9
Trained on Reuters Dataset	21193.6	1499.88	30317.3
Trained on Reuters Gutenberg	17072.1	36810.1	3232.75



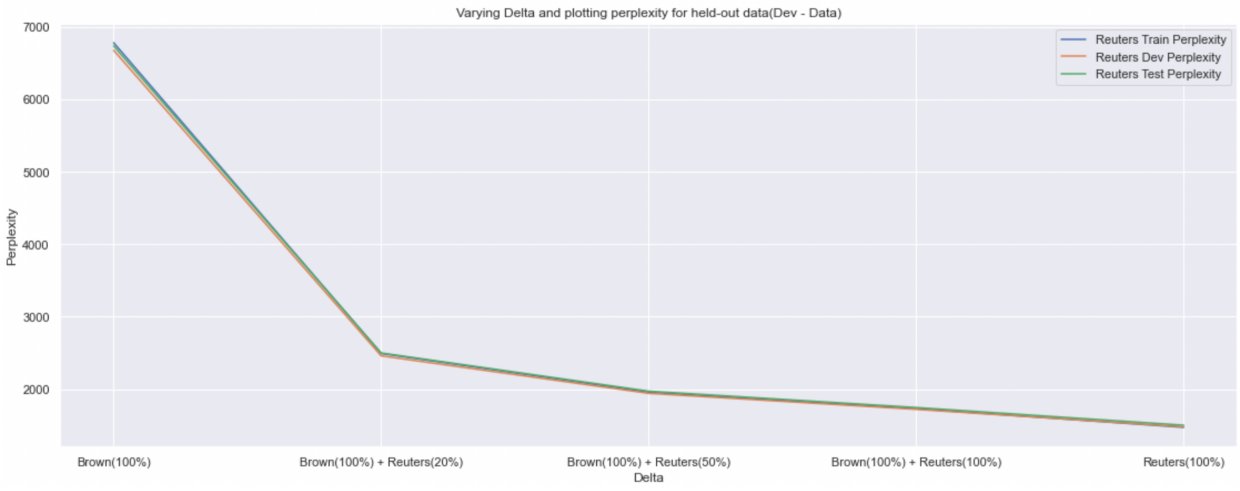
From the graph, we can see that when tested against different dataset models tends to decrease its performance. A pattern can be observed that the perplexity is lowest for the data when trained on the training set of that data. When we see the results for out-of-domain data perplexity increases drastically, i.e., the model trained with the trigram +

Laplace method only contains information related to that domain so testing against inter-datasets(out-of-domain) leads to an increase in perplexity. The model contains information of only seen trigram, for the rest of all unseen trigram, it assigns the same log probability because of which model gets confused and yields higher perplexity. The intuition behind this pattern is that the data on which it was trained performs best when tested against the same data. In other words, e.g., the model is trained on brown train data it will perform best when tested against brown train dataset, after that, it will perform best on dev and test set of brown data the reason behind this is that the model encapsulates the semantical and syntactical information of the data, so model tends to make fewer errors.

6.5 Adaptation

I have conducted experiment by training unigram model on dataset A and partial dataset of B (varying by 20%, 50%, 100%), and tested against perplexity for train, test, and dev of dataset B. I have compared the results obtained with model trained on dataset A and tested against train, dev, and test of dataset B.

	Reuters Train Perplexity	Reuters Dev Perplexity	Reuters Test Perplexity
Unigram Model Trained on Brown Dataset	6780.82	6675.63	6736.6
Unigram Model Trained on Brown Dataset + 20% of Reuters Dataset	2490.334	2458.15458	2497.87355
Unigram Model Trained on Brown Dataset + 50% of Reuters Dataset	1957.2600	1940.4073	1970.5193
Unigram Model Trained on Brown Dataset + 100% of Reuters Dataset	1729.208	1719.5503	1744.894
Unigram Model Trained on Reuters Dataset	1471.209	1479.093	1500.6949



From the above table we can see that as we gradually increase the training data B(Reuters) with training data A(Brown), we tend to see decrease in perplexity, which shows that as training data of B increases model stores more information about trigram of dataset B, thus perplexity reduces. In other words, model tries to adapt information of both the datasets i.e., it contains log probability information of trigrams in dataset B. Since it contains information of trigrams instead of assigning every trigram same log probability, it assigns a specific log probability using $\log probablity\ of\ a\ word = log_2(Count(x_{i-2}, x_{i-1}, x_i) + \delta) - log_2(Count(x_{i-2}, x_{i-1}) + \delta * V)$, thus most of the information about trigrams are stored in the model i.e. there are less trigrams from out-of-vocabulary. As training data of B is increased (from 0% to 20% to 50% to 100%), we can see that perplexity reduces drastically and almost become equals to perplexity when trained with training data B.

One more Comparison:

	Gutenberg Train Perplexity	Gutenberg Dev Perplexity	Gutenberg Test Perplexity
Unigram Model Trained on Brown Dataset	1758.0587	1739.41348	1762.0056
Unigram Model Trained on Brown Dataset + 20% of Gutenberg Dataset	1333.1285	1323.9484	1344.37503
Unigram Model Trained on Brown Dataset + 50% of Gutenberg Dataset	1114.8422	1114.55851	1128.5346
Unigram Model Trained on Brown Dataset + 100% of Gutenberg Dataset	1052.9182	1054.9369	1069.30262
Unigram Model Trained on Gutenberg Dataset	982.5718	991.5002	1005.78989

Thus similar patterns is observed as we increase the percentage of data in training (Gutenberg), perplexity reduces drastically and tends to reach the perplexity when just trained on Gutenberg data.

References:

[1] Lafferty, J., Sleator, D., & Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar (Vol. 56). School of Computer Science, Carnegie Mellon University.
[2] Avasthi, S., Chauhan, R., & Acharjya, D. P. (2021). Processing large text corpus using N-gram language modeling and smoothing. In Proceedings of the Second International Conference on Information Management and Machine Intelligence (pp. 21-32). Springer, Singapore.
[3] Stolcke, A. (2000). Entropy-based pruning of backoff language models. *arXiv preprint cs/0006025*.