Programming Assignment 1: Language Modeling

Shihan Ran, A53313589
Fall 2019, CSE 256: Statistical NLP

I. Language Model Implementation

A. *Trigram Model*

*1) **Symbols**:*

- For trigram language model, the second-order Markov assumption is made. The probability of any sequence $s = x_1 \dots x_n$ is $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = \Pi_{i=1}^{n} P\left(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}\right)$. We further assume that $x_{-1} = x_0 = *$.
- For any i and any x_{i-2}, x_{i-1}, x_i we have $P\left(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}\right) = q\left(x_i | x_{i-2}, x_{i-1}\right)$. Then the model takes the form $p\left(x_1 \dots x_n\right) = \prod_{i=1}^{n} q\left(x_i | x_{i-2}, x_{i-1}\right)$ for any sequence s.
- For any trigram (u, v, w), and a finite set $\mathcal{V}$, define the probability of the appearance of the word w immediately after the bigram (u, v) is $q(w|u, v)$, $w \in \mathcal{V} \cup \{STOP\}$ and $u, v \in \mathcal{V} \cup \{*\}$.

*2) **Maximum-Likelihood Estimates**:*

We use maximum-likelihood estimates to estimate $q\left(w|u, v\right)$. Define $c(u, v, w)$ to be the number of times that the trigram (u, v, w) is seen in the training corpus. Similarly, define $c(u, v)$ to be the number of times that the bigram (u, v) is seen in the corpus. Then for any w, u, v we have $q(w|u, v) = \frac{c(u,v,w)}{c(u,v)}$.

*3) **Implementations**:* We implemented the following functions:

In Algorithm 1, for trigram model, we will only need to store bigram and trigram counts.

In Algorithm 2, we calculate the log-probability as

$$\begin{aligned} \log q(w|u, v) &= \log \frac{c(u, v, w)}{c(u, v)} \\ &= \log c(u, v, w) - \log c(u, v) \end{aligned}$$

Algorithm 1: $fit\_sentence$

Input: $sentence$ as a List
Output: None but update $self.unigram$, $self.bigram$ and $self.trigram$

```
Generate all the trigrams and bigrams and stored in Lists;
for each trigram t in trigram List do
    Update trigram appearance counter;
for each bigram b in bigram List do
    Update bigram appearance counter;
```

Given previous words, Algorithm 3 return the log of the conditional probability of word. If the trigram (u, v, w) is never seen, then we return the log of a very small number to represent the low probability.

Algorithm 2: $norm$

Input: $self.trigram$, $self.bigram$ and $self.unigram$ as dicts
Output: $self.model$

```
for each trigram t in self.trigram do
    Find the corresponding bigram b;
    // Update the store q(w|u,v)
    self.model = log(self.trigram[t]) − log(self.bigram[b])
```

B. *Smoothing method*

Add-δ smoothing is chosen to ensure my language model outputs a non-zero and valid probability distribution for OOV words.

Compared with Algorithm 1, when doing smoothing, we add the vocabulary size $\mathcal{V}$ in the denominator, so we need the number of distinct tokens. That's why we update the unigram counter in Algorithm 4.

Algorithm 3: *cond_logprob*

Input: $self.model$, $word\ w$, $previous[u, v]$
Output: the log-probability of (u, v, w)

if trigram (u, v, w) in $self.model$ **then**
  return $self.model[(u, v, w)]$
else
  // Return the log of a very small number
  return $self.lbackoff$

Algorithm 4: *fit_sentence*

Input: $sentence$ as a List
Output: None but update $self.unigram$, $self.bigram$ and $self.trigram$

// As above;
for each unigram w in sentence List **do**
  Update unigram appearance counter;

In Algorithm 5, after using Add-δ smoothing, we calculate the log-probability as

$$\log q(w|u, v) = \log \frac{c(u, v, w) + \delta}{c(u, v) + \delta * \mathcal{V}}$$
$$= \log\Big(c(u, v, w) + \delta\Big) - \log\Big(c(u, v) + \delta * \mathcal{V}\Big)$$

Algorithm 5: *norm*

Input: $self.trigram$, $self.bigram$ and $self.unigram$ as dicts, $self.delta$ as smoothing parameter
Output: $self.model$

for each trigram t in $self.trigram$ **do**
  find the corresponding bigram b;
  // Update the store $q(w|u, v)$ using smoothing
  $self.model = \log(self.trigram[t] + self.delta) - \log(self.bigram[b] + self.delta * len(self.unigram.keys()))$

Given previous words, Algorithm 6 return the log of the conditional probability of word. Compared with Algorithm 3, if the trigram (u, v, w) is never seen, we return

$$\log q(w|u, v) = \log \frac{\delta}{c(u, v) + \delta * \mathcal{V}}$$
$$= \log\Big(\delta\Big) - \log\Big(c(u, v) + \delta * \mathcal{V}\Big)$$

Algorithm 6: *cond_logpro*

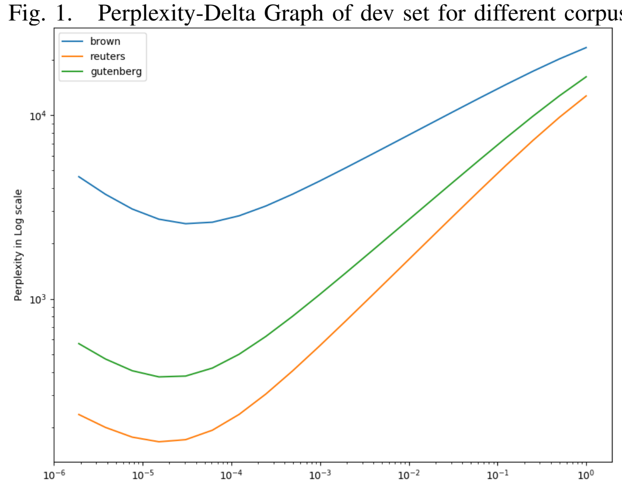
Input: $self.model$, $word\ w$, $previous(u, v)$ and $self.delta$
Output: the log-probability of (u, v, w)

if trigram (u, v, w) in $self.model$ **then**
  return $self.model[(u, v, w)]$
else
  find the corresponding bigram b;
  if bigram b in self.bigram **then**
    return $\log(self.delta) - \log(self.bigram[b] + self.delta * len(self.unigram.keys()))$
  else
    return $-\log len(self.unigram.keys())$

C. Hyperparameter tuning and data used for tuning

The only hyperparameter needed to tune is δ. So I tune the parameter δ on the dev set of the three corpus. The results are shown in the figure 1.

Fig. 1. Perplexity-Delta Graph of dev set for different corpus



We let δ ranges between $1/2^0$ and $1/2^{20}$. Perplexity reaches the lowest point when δ equals $1/2^{15}$, which is $3.0517578125e{-}05$. So we choose to set δ as this value.

II. Analysis of In-Domain Text

A. *Perplexity values*

TABLE I
In-Domain Perplexity of the test set

	Trigram + Smoothing	Trigram	Unigram
Brown	2619.47	18215.20	**1604.20**
Reuters	**181.48**	607.16	1500.69
Gutenberg	**390.65**	1423.66	1005.79

The computed perplexity of the test for three different corpus and the compared performance with baseline models is shown in Table I. As we can see, Trigram with Smoothing Technique outperforms Trigram on all three corpus, which suggests that our smoothing technique works! Furthermore, Trigram with Smoothing Technique also outperforms Unigram on Reuters and Gutenberg corpus. There is an interesting fact that Unigram outperforms Trigram with Smoothing Technique on Brown corpus. It may due to the relatively *small* size of training data and *huge* size of vocabulary.

TABLE II
Dataset Distribution

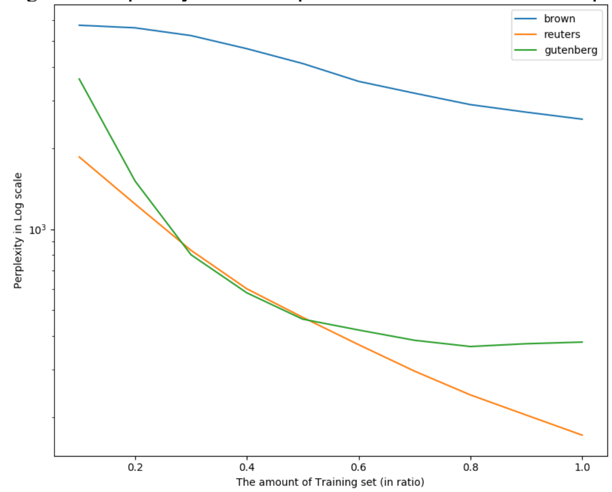
	Train	Dev	Test	Vocab
Brown	39802	8437	8533	**41746**
Reuters	38169	8082	8214	36037
Gutenberg	**68740**	**14729**	**14826**	**43835**

Next, We observe the trend when the amount of training data is varied. The results are shown in the figure 2. As we can see from the figure, performances of both Brown and Reuters increases as the data size increases, while the performance of Gutenberg seems to have a convergency when the ratio is approaching 1. This makes sense since as Table II shows, Gutenberg is the largest among the three corpus. The phenomenon provides us with some insights to increase the model performance.

B. *Some Examples of sampled sentences*

We generate samples using prefix *To be or*. As we can see from the following results, the Unigram merely outputs some frequent words but makes nonsense. For Trigram models, however, whether with smoothing techniques or without, although the generated sentence may not make sense completely, there still are some *local sense*,

Fig. 2. Perplexity-Ratio Graph of dev set for different corpus



for example, the underlined expressions in the sentences do make sense.

Moreover, we can see the differences in the domain of training data can lead to the differences in the outputs of language models. For example, the outputs of Reuters is more related to financial domain.

1) ***Unigram***:

- Brown: *To be or to at savings think Sarah and father to where and platform face...*
- Reuters: *To be or to is it supply chairman oil sources off affect losing letter...*
- Gutenberg: *To be or to whom*

2) ***Trigram without Smoothing***:

- Brown: *To be or to put down on schedule*
- Reuters: *To be or to hold oil prices to farmers who have built several Bertone X1 sports cars with structures...*
- Gutenberg: *To be or to be baffled To manual work for Ahab and also concerning the kingdom of God for he shall cause their terror...*

3) ***Trigram with Smoothing***:

- Brown: *To be or to costly improvements wedge microorganisms 196 Means tumbles juniors interposed...*
- Reuters: *To be or to first quarter will have competitive advantage...*
- Gutenberg: *To be or to rest with antics fisher snug Expand hypocrites...*

III. Analysis of Out-of-Domain Text

A. *Perplexity values*

TABLE III

Out-Domain Perplexity for Trigram+Smoothing

	Brown	Reuters	Gutenberg
	Trigram + Smoothing		
Brown	2619.47	12894.8	6630.76
Reuters	8189.15	**181.48**	15790.2
Gutenberg	4458.06	18965.2	**390.655**
	Trigram		
Brown	18215.2	166814	62845.3
Reuters	104788	607.157	254970
Gutenberg	37527.2	279717	1423.66
	Unigram		
Brown	**1604.2**	6736.6	1762.01
Reuters	3865.16	1500.69	4887.47
Gutenberg	2626.05	12392.5	1005.79

As we can see from Table III, with Trigram and Smoothing Techniques, we can achieve the best performance for In-Domain text data while the performances is worse for Out-Domain text data. For Out-Domain text data, we reached the lowest perplexity with Unigram! This may due to the overfitting of Trigram model.

We also found out that, compared with Reuters, Brown and Gutenberg seems to be more *alike*, which means the model trained using these corpus can generalize better to each other and have lower perplexity. This can be further verified by the overlap statistical information from Table IV. We can see from the table that Brown and Gutenberg have more overlap in unigrams, bigrams and trigrams.

TABLE IV

Overlap within three corpus

	Unigram	Bigram	Trigram
Brown	41746	346015	591893
Reuters	36037	328997	641298
Gutenberg	**43835**	**483447**	**1.03566e+06**
Brown∧Reuters	13786	46985	25444
Brown∧Gutenberg	**19570**	**75182**	**50483**
Reuters∧Gutenberg	10565	32626	16573

IV. Adaptation

A. *The proposed Approach*

Suppose we already trained a Trigram with Smoothing model on corpus A, which means we already stored $c(u, v, w)$ and $c(u, v)$, for any (u, v, w) in A. Then we got a small fraction of corpus B's training data, we want to further fine-tune our model with the pre-trained model parameters.

The proposed approach is combining the training data from B and A, which means we simply add $c'(u, v, w)$ to the original $c(u, v, w)$, and $c'(u, v, w)$ represents the number of times trigram (u, v, w) is seen in the B's training data.

After recalculating, we use the fine-tuned model to test it on the test set for corpus B. It will outperform the initial model trained just on corpus A since the distribution is adapted closer toward the actual one.

B. *Experiment*

As we discussed before, Brown and Gutenberg corpus are more alike in some sense as their overlap is the biggest and the out-of-domain perplexity is the lowest. Moreover, we can see from Table III that model trained on Gutenberg and test on Brown has a lower perplexity (**4458.06**), compared with model trained on Brown and test on Gutenberg (6630.76). So we run a test experiment: Train a trigram model with smoothing techniques on Gutenberg corpus and take certain ratio of Brown training data, then apply our proposed adaptation approach. The results are shown as Table V.

TABLE V

Perplexity results on different ratio

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
4002	3607	3236	2889	2521	2137	1907	1738	1602	**1490**

If we train a model only on Brown, then we will get a test perplexity of **2619**. Table V shows the following information:

- After retraining the model on Brown corpus, the perplexity keeps decreasing as the training data increases (<4458), which suggests fine-tuning is working and the distribution is gradually adapted closer toward the actual one.
- After retraining on over half amount of Brown corpus, the perplexity is lower than training only on Brown (<2619), which suggests pre-training on Gutenberg is working. Pretrain on a larger and also related training data will help increase the model perplexity.