Assignment1 : Language Models

Arpan Mangal
M.Tech, CSA (14353)
Indian Institute of Science
`arpanmangal@iisc.ac.in`

Abstract

In this Assignment, we experiment with Language Models on two corpus i.e **Brown corpus** and **Gutenberg corpus**. There are two tasks which are performed.
Task1: We divide each dataset into train, dev, and test. Let the training splits be D1-Train and D2-Train. And, then implemented various LM and report the best LM in the following settings:

- S1: Train: D1-Train, Test: D1-Test
- S2: Train: D2-Train, Test: D2-Test
- S3: Train: D1-Train + D2-Train, Test: D1-Test
- S4: Train: D1-Train + D2-Train, Test: D2-Test

Perplexity has been used as comparison metric here.

Task2: Using our best model, We generate a sentence of 10 tokens.

1 Pre-Processing

Before actually applying language models on corpus, we need to pre-process,to clean the data. For this task, we need to generate good sentences in the end. And we have to evaluate the model based on "Perplexity" as metric. These steps can't be performed:

1. Stop Word Removal and Stemming and Punctuation can't be performed because they are basic part of grammatically correct sentence.

2. Similarly, we can't change every character to lower case, as upper case "The" usually appears in the beginning of sentence while "the" appear in the middle. We need to differentiate between the two in order to generate good sentence.

Keeping all the above requirement in mind. We removed extra white-space character from the text. We separate out sentences from the text (stripping off POS tag from each word in brown corpus).

2 Splitting

In order to get unbiased estimate, it's important that all the three, **train**, **dev** and **test** data set has same proportion of each type of sentence. Therefore, We first shuffled all the sentences and then split as **80% train, 10% dev and 10% test** data. After splitting the sentences, We just merge all the sentences in the corresponding split, by inserting "start tag" and "end tag" in between. It will be helpful as we are measuring perplexity as evaluation metric, which need text as one sentence

3 Implementation

For task 1, We have tried implementing **Ngram** (bigram and trigram) model along with these three advance smoothing techniques (`http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf`):

1. Linear Interpolation

2. Absolute Discounting (Katz back-off)

3. Kneser-Ney

3.1 Linear Interpolation

A linearly interpolated trigram model is used. For words **u, v and w**, we define the trigram, bigram and unigram maximum likelihood estimate respectively as follows:

$$q(w|u,v) = \frac{c(u,v,w)}{c(u,v)}$$

$$q(w|v) = \frac{c(v,w)}{c(v)}$$

$$q(w) = \frac{c^{dis}(w)}{c(.)}$$

Idea in linear interpolation is to use all three estimates, by defining the trigram estimate as follows. For unigram probabilities, we have taken discounted counts (subtract uniform absolute discount β from each unigram count), so that unigram probability will never become zero.

$$p(w|u,v) = \lambda 1 * q(w|u,v) * \lambda 2 * q(w|v) * \lambda 3 * q(w) \quad (1)$$

where λ's are defined with respect to context as follows:

$$\lambda 1 = \frac{c(u,v)}{c(u,v)+\gamma}$$

$$\lambda 2 = (1-\lambda 1) * \frac{c(v)}{c(v)+\gamma}$$

$$\lambda 3 = 1 - \lambda 1 - \lambda 2$$

Here, γ is a hyper-parameter to be tuned. Lambda's are defined in such a way that if we have higher confidence on trigram probability, we will set higher value for lambda1 and so on.

3.2 Katz's Back-off

We have implemented this smoothing with **Bigram** model.The first step will be to define discounted counts.For any bigram c(v,w) such that c(v,w) > 0. We define the discounted count as:

$$c^{dis}(u,v) = c(u,v) - \beta$$

where, β is a discount factor, hyper-parameter which needs to be tuned.
For any context **v**, this definition leads to some missing probability mass, defined as:

$$\alpha(v) = 1 - \sum_{w:c(v,w)} \frac{c^{dis}(u,v)}{c(v)} \quad (2)$$

For any v, we define the sets

$$A(v) = \{w \mid c(v,w) > 0\}$$

and

$$B(v) = \{w \mid c(v,w)! = 0\}$$

Then, bigram probability will be given as:

$$p(w|v) = \begin{cases} \frac{c^{dis}(v,w)}{c(u,v)}, & \text{if } w\epsilon A(v) \\ \alpha(v) * \frac{q(w)}{\sum_{w\epsilon B(v)} q(w)}, & \text{otherwise} \end{cases}$$

3.3 Kneser-Ney

We have implemented this smoothing with **Bigram** model.

$$P_{AD}(w|v) = \frac{max(c(v,w)-d,0)}{c(v)} + \lambda(v) * P_{cont}(w) \quad (3)$$

Here, we are calculating bigram probability with absolute discounting. If the bigram count is non-zero, discount factor is subtracted from it while calculating probability. To ensure that probability remains positive, we are taking maximum of discounted count and 0. The other part includes probability of continuation in context, weighted by confidence λ for corresponding context. Probability of continuation and corresponding λ is calculated as follows:

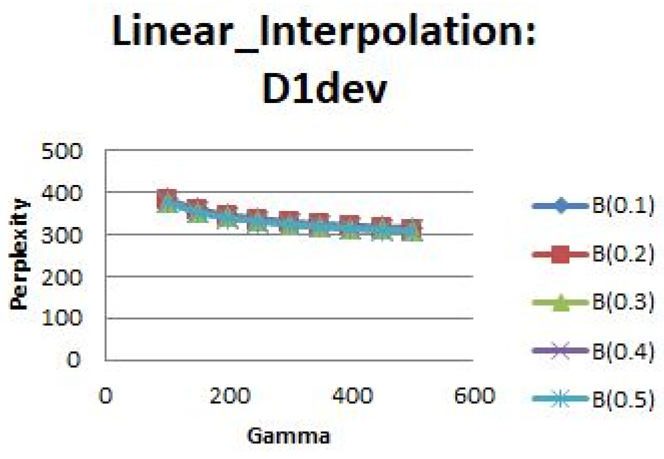
$$P_{cont}(w) = \frac{|\{v : c(v,w) > 0\}|}{\sum_{w'} |\{w' : c(v',w') > 0\}|} \quad (4)$$

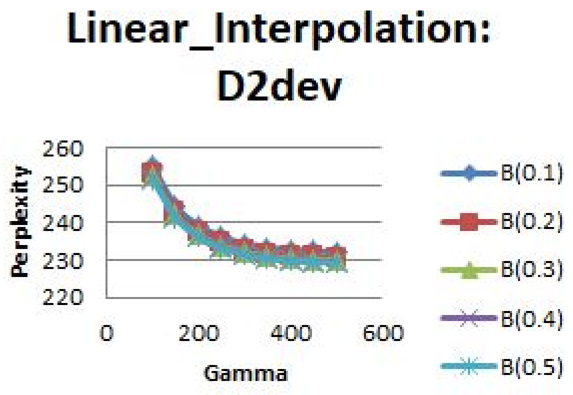
$$\lambda(v) = \frac{d}{c(v)} * |\{w : c(v,w) > 0\}| \quad (5)$$

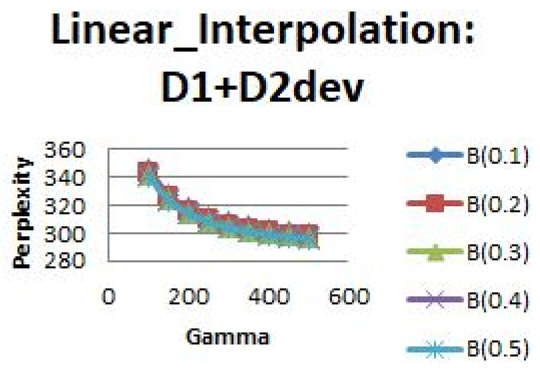
4 Hyperparameter Tuning

All the hyper-parameters are tuned on development set in all the four settings.

Linear Interpolation: There are two hyperparameters involved in this. γ is the hyperparameter used to calculate λ's. β is the absolute discounted factor subtracted from unigram counts. Following are the graph obtained while tuning these parameters in all settings:

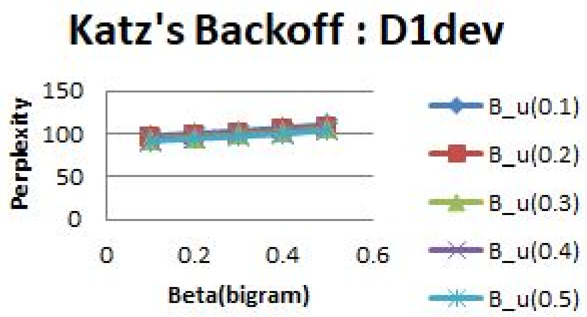


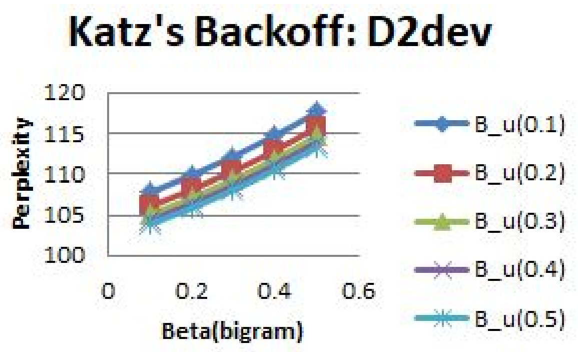


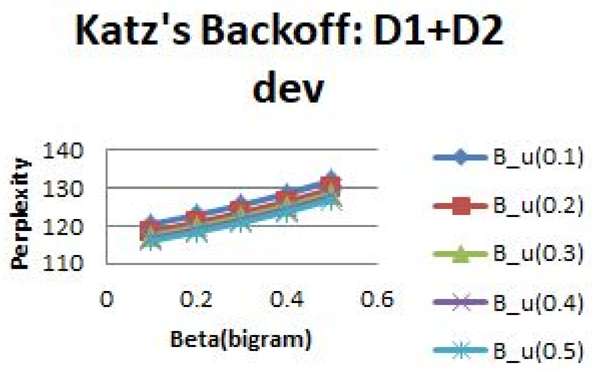


In all the settings we find that $\beta = 0.5$ and $\gamma = 500$ leads to minimum perplexity. Therefore, during test data set evaluation, values to these parameters are set to be 0.5 and 500 respectively.

Katz's Back-off:There are two hyperparameters involved in this. β_uni is the absolute discounted factor subtracted from unigram counts and β_bigram is the discounted factors from bigrams with non-zero counts.Following are the graph obtained while tuning these parameters in all settings:

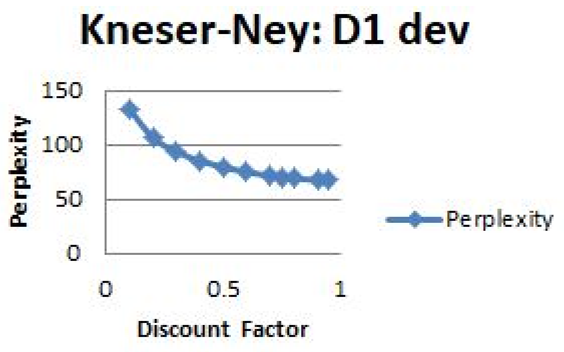


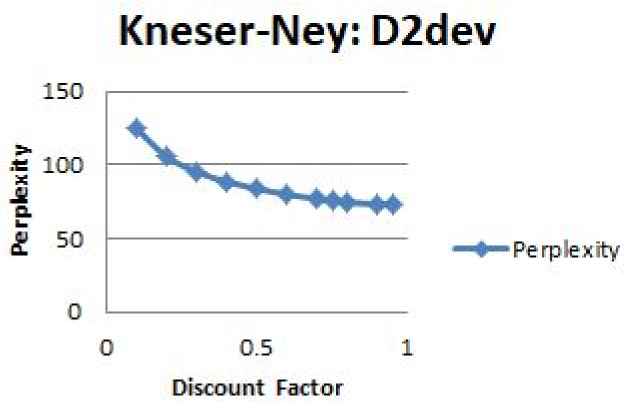


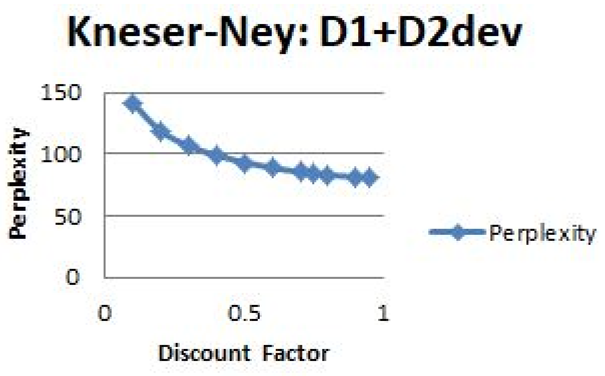


In all the settings we find that β_uni = 0.5 and β_bi = 0.1 leads to minimum perplexity.Therefore, during test data set evaluation, values to these parameters are set to be 0.5 and 0.1 respectively.

Kneser-Ney: There is only one hyperparameter involved in this. Since, it's one of the absolute discounting method of smoothing, we need to tune the hyperparameter **d** i.e discount factor which needs to be subtracted from bigram with non-zero count.Following are the graph obtained while tuning this parameters in all settings:







In all the settings we find that **d** = 0.95 leads to minimum perplexity.Therefore, during test data set evaluation, values to this parameters is set to be 0.95.

5 Evalution Measure

We used **Perplexity** as our evaluation measure. These are the values of perplexity which we got on **test** data

1. **S1 : train = D1_train and test = D1_test**
   - Good Turing: 408.78
   - Linear Interpolation : 308.81
   - Katz's Back-off : 91.91
   - **Kneser-Ney : 67.64**
2. **S2 : train = D2_train and test = D2_test**
   - Good Turing: 454.76
   - Linear Interpolation : 228.66
   - Katz's Back-off : 103.01
   - **Kneser-Ney : 72.39**
3. **S3 : train = D1_train + D2_train and test = D1_test**
   - Good Turing: 559.51
   - Linear Interpolation : 150.51
   - Katz's Back-off : 65.92
   - **Kneser-Ney : 46.61**
4. **S4 : train = D1_train + D2_train and test = D2_test**
   - Good Turing: 528.35
   - Linear Interpolation : 117.74
   - Katz's Back-off : 78.32
   - **Kneser-Ney : 51.78**

In all the four settings, Kneser-Ney smoothing perform much better than others. It was as expected because Kneser-Ney smoothing use to perform better on small corpus while Katz's smoothing use to perform better on very large corpus. Both of these are better than Linear Interpolation because they take many thing into account(like frequently this word comes after this context) which Linear Interpolation does not handle.
Good Turing might perform better in actual sense. Here, we are not implementing it properly due to time constraints. Original version requires fitting of Power Law Distribution. We are just backing-off to unigram probability.

6 Sentence Generation

For generating the sentence, We are using trigram probabilities. Given previous two words, we find the probability distribution of each trigram with first two word as previous words. Then we generate the word randomly according to the probability distribution. This means that trigram which is highly probable in corpus has higher chances of picking up. With this techniques, sentence generated are readable as they have previous two words in context. For example, following are few 10 token sentence generated by the program.

1. He was bitter when he observes on several occasions that
2. They had been my help and protect them against the
3. Loveit coloured the sugar pulms by the missile destroying the target

7 Source Code

https://github.com/mangalarpan/LanguageModel.git