Language Modelling with N-grams

Swapnil Gupta
M.Tech, IISC Bangalore
swapnilgupta.229@gmail.com

1 Introduction

Language Models are probabilistic models of word sequences which assign a likelihood score to a sequence of words on the legitimacy of that sentence in any language. Such models are widely used in the applications such as speech recognition, handwriting Recognition, spelling correction and machine translation.

Suppose we have a corpus, which is a set of sentences in any language. We define **V** as the vocabulary of the corpus. Based on these given a sequence of words $(w_1, w_2, ..., w_n)$ we want to estimate

$$P(w_1, w_2, ..., w_n) \text{ such that,}$$
$$\sum_{(w_1,w_2,...,w_n)\epsilon \mathbf{V}} P(w_1, w_2, ..., w_n) = 1$$

where $(w_1, w_2, ..., w_n)\epsilon \mathbf{V}$ implies all possible sentences using the vocabulary.

Using **chain rule**

$$P(w_1, w_2, ..., w_n) = \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

where $w_1^{k-1} = (w_1, w_2, ..., w_{k-1})$ the last $K-1$ word history.

An **N-gram** is a sequence of N words, which is used to estimate the probability of the last word of the N-gram given the previous words. Which can also be interpreted as given the $N-1$ word history what is the conditional probability of the N^{th} word. In N-gram models we make the assumption that the N^{th} word is only dependent on the last $N-1$ words and conditionally independent of any previous word history. Mostly used in practice N-gram models are Bigram Models where $N = 2$ and Trigram models where $N = 3$.

The learning task is, given a corpus estimate $P(w|history)$ for each possible word sequence.

2 Implementation details of N-gram Models

The N-gram models doesn't know how to handle the cases where the test sentences have words which are unseen in the training data. This is called problem of **Unknown Words**. A naive implementation would give any such test sentence a zero probability. Several heuristic approaches are generally adopted to tackle this issue. In this project we have replaced a certain set of words with low frequency with a specialized token 'UNK' which is representative for all those words which are not present in the vocabulary corresponding to the training corpus.

Another major drawback with N-gram models is that during testing it assigns zero probability to all the N-grams which are not present in the training corpus. Hence a word sequence which has any such N-gram our model will assign zero probability to such sequences. This results in **poor generalization** on the unseen data. To avoid our model giving zero probability to unseen sentences we shave off some probability mass from frequent sentences and save it for unseen events. This process is called **Smoothing** or **Discounting**.

The following is the list of commonly used smoothing models.

- Laplace Smoothing
- Katz Backoff
- Good Turing Backoff
- Kneser-Ney Smoothing
- Stupid Backoff

In this project I have implemented three different language models. Bigram models with 'Kneser-Ney Smoothing' and 'Katz Backoff' and Trigram model with 'Stupid Backoff'.

3 Training Corpus

Two different text corpus has been used to train our language models

- D1:Brown Corpus
- D2:Gutenberg Corpus

Brown Corpus which was first published in 1961 contains 500 text files from 15 different categories contains around 54,000 unique characters and 2,400,000 total characters.

Gutenberg Corpus is a collection of 18 english books of several different authors containing around 49,000 unique characters and 2,650,000 total characters.

For the training and evaluation the above corpus has been divided in test and train corpus in the ratio $90 : 10$. The train corpus is further divided in the ratio $90 : 10$ in training and held-out set for tuning the hyper parameters.

As per our Task 1, I have built and evaluated the Language Models in the following four 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

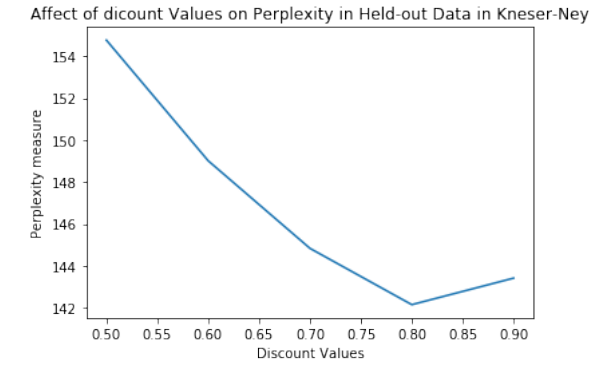
4 Evaluation of Language Model

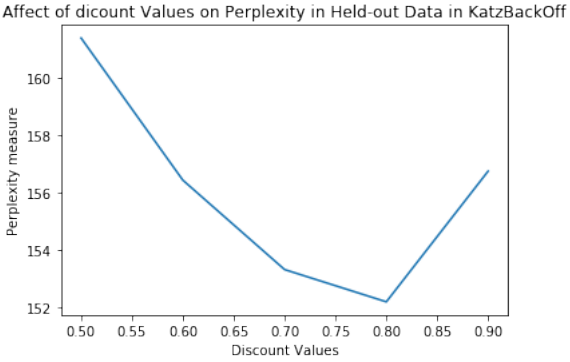
There are two ways to evaluate a language model, intrinsic evaluation metric which is independent of any application and extrinsic evaluation where the model is evaluated by performing some application. For this task an intrinsic measure called **Perplexity** is chosen. Perplexity of a Language model on test set is the inverse probability of the test set, normalized by the number of words. For a given test set W perplexity is defined as:

$PP(W) = P(w1, w2, w3....wN)^{-1/N}$

where N is the number of tokens with including end of sentence marker but not beginning of sentence marker.

Perplexity measure on the held out set is utilized to tune the hyper parameter discount factor used for Kneyser Ney smoothing and Katz backoff. It is the measure of probablity we shave off from the frequently occuring events to save for the unseen events. It can also be called as the smoothing parameter. The following plots shows the variation of perplexity with respect to discount factor for both the smoothing techniques:





Both the above figures are plotted for S2

The figures shows a similar trend for both the Models, where perplexity first reduces with increasing discount parameter and then increases achieving a minimum at around 0.8. Same is the case with all the 4 settings.

Performance on Test Data :

Model type	**S1**	**S2**	**S3**	**S4**
Bigram (Kneyser Ney)	238.39	141.99	317.63	155.60
Bigram (Katz Bckoff)	255.29	151.95	345.36	162.53
Trigram (Stupid Back-off)	198.20	92.24	250.97	97.42

T1: Table showing Perplexity values for 4 different settings as defined above. The backOff

factor used for Stupid Backoff is 0.7

From the above table we can make the following observations. In Bigram Models Kneser-Ney Model always outperforms Katz Back off. Also the running time of Kneser-Ney is less in comparison to Katz. Showing the effectiveness of Kneser-Ney Smoothing Model which is also as expected. It can also be observed that the Trigram model with Stupid back-off obtained the lowest perplexity. Though we should remember that Stupid back-off doesn't really maintains a valid probability distribution. Also since here the training data is not very high hence Stupid Back off is not a very accurate approximation.

5 Sentence Generation

Task 2 of this project is to generate a 10 token natural language sentence using the Language Models built in Task 1. I have used a trigram model to generate sentences.

Generating sentences is easier compared to Task 1 in the sense that you don't really have to be concerned for the out of vocabulary words or unseen n-grams problem, but it has its own challenges. A major challenge is to make sure that the sentence has a proper ending at the 10^{th} token. Also we need to generate the sentences in probabilistic sense in order to maintain randomness in our sentences and finally we have to maintain the quality of the sentences.

To handle the first challenge, I have used the following heuristic approach. I have predefined a set of probable end sentence tokens. After generating 9 tokens of the sentence, I checked whether the training corpus contains at least one trigram with last two tokens of the generated 9 tokens and a token belonging to the end sentence characters. If the list of such trigrams is empty I have started the process of generating the sentence tokens all over again. In this way I have forced my model to chose one of the end sentence tokens.

To sample each token randomly, I first generated a list of all the trigrams in the training corpus containing the last two generated tokens. Sample one of them randomly and generate a random number between 0-1. If the random no. generated is less than the probability of the sampled trigram then we appended the last character of the trigram to our sentence else we sample the trigram again from that list.

To examine the quality of the sentence generated, I have calculated the perplexity of the each sentence generated. 10 such sentences are generated and the sentence which has the lowest perplexity is selected.

Below are some of the sample sentences generated through above process.

> "We are worried about their machinery beyond mechanical details."

> "(2) slow down and permit the passage."

> "For the prevention of anaplasmosis, feed 100-200 grams."

> "His unsuccessful strivings to give their reasons, too."

> "I went to Jacopo Galli introduced him into trouble."

One last thing that I have tried in this section is to generate the "gold sentence" of the corpus. Here for generating each token rather than doing sampling from the probability distribution as discussed above I chose the token which has the maximum probability given the last two token history. This gives us the following sentence.

> "The first two years ago, the first time in"

Interestingly the perplexity of this sentence is higher than the perplexity of some of the sentences generated above.

6 Conclusion

This project gave us the opportunity to develop some incites on the role context plays in Natural Language Processing. Developing a sentence generator and tuning it to required conditions was a very interesting experience. In recent years Deep Learning techniques like RNN's have strongly dominated the field language models. After this project it would be interesting to look into these recent models.