Generating Structured Queries from Natural Language

Rishi Hazra
rishixtreme@gmail.com

1 Problem Statement

In this assignment, you will experiment with language models and work with the following two datasets:
D1: Brown Corpus
D2: Gutenberg Corpus

Task 1: Divide each dataset into train, dev, and test (splits are up to you). Let the training splits be D1-Train and D2-Train. Now, implement and build the best LM in the following settings and evaluate.

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

Task 2: Using the best model, you will also be required to generate a sentence of 10 tokens.

2 Language Model

3 Proposed Solution

In my language model, I have split the data into three parts, namely train, test and dev. I made a vocabulary out of my train corpus and used randomization to replace 1/4 low frequency words from my train corpus into ¡UNK¿. I then, used the same vocabulary to compare and replace words from test and dev splits to ¡UNK¿. It was noticed that punctuations were placed at unnecessary places in the corpus, removing which produced a better perplexity. Sentence start tokens¡s¿ and end tokens¡e¿ were placed in the whole corpus which would later help me to generate sentences. For comparing the model, perplexity was used as a metric.

4 Perplexity

In practice we dont use raw probability as our metric for evaluating language modperplexity els, but a variant called perplexity. The perplexity (sometimes called PP for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words.
For a test set $W = w1w2w3....wN$

$$PP(W) = P(w1w2...wN)^{-1/N}$$

Kneser-Ney smoothing along with word replacement with ¡UNK¿ was used to obtain the perplexity. For the Knese Ney smoothing, the following formula was used.

d=0.5 for bigram count=0 d=0.75 for bigram count=1

4.1 Comparing the models

The model trained on brown corpus generated the following data:
Unigram Perplexity= 870.92
bigram perplexity=335.43
trigram perplexity= 2.56

The model trained on gutenberg corpus generated the following data:
Unigram Perplexity= 685
bigram perplexity=198.34
trigram perplexity=4.06

The model trained on gutenberg and brown combined corpus generated the following data:
testing on brown (D1)
Unigram Perplexity= 1122
bigram perplexity=416.6
trigram perplexity=3.13

testing on gutenberg(D2)
Unigram Perplexity= 733
bigram perplexity=210
trigram perplexity=4.46

As is evident from the models, the trigram perplexity is far better than the bigram and unigram perplexities. Hence the trigram model was used to generate the sentences.

4.2 Sentence Generation

For sentence generation, the trigram model of brown corpus was used. A sentence of 10 tokens was to be generated. In order to do that, a sentence start token was fed to the generate funtion which initially produced a bigram within a certain threshold probability (probability was set to be higher than 0.05). The newly generated token along with the sentence start token was used to generate trigram with randomizations. Back propagation was used to give a meaningful ending to the 10 tokened sentences.