iNLP : Tutorial 1

Assignment 1 - Discussion & Doubts

Hardik & Jainit

Topics that we will be covering

- Tokenization
- N-Grams
- Interpolation
  - Use of <UNK> words for OOD words for unigrams
- Good Turing Smoothing
  - Linear regression on the log-log Zr plot
- Also mention numerical underflow through repeated multiplication
  - How to mitigate using log domain

Tokenization [tokenizer.py]

- You can learn Regex at regex101.

- Steps
  - Take the input word/ sentence/ paragraph/ corpus after running the program
  - Clean the input (URLs, Hashtags, Mentions ...) using regex
  - Tokenise the corpus into sentences using either Regex or some predefined python library
  - Output a list of list [[s1w1, s1w2,...] , [s2w1, s2w2, ...] ...]

N-grams

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

- N-gram solution: assume each word depends only on a short linear history (a Markov assumption)

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

$$P(S \mid M) = 2^{H(S|M)} = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P_M(w_i \mid h)}}$$

building an n-gram model.

[python language_model.py <lm_type> <corpus_path>]

- Load and save are important to implement in the code as we will be needing those during the evals
- **PROBABILITIES OF N-GRAMS**
  - **UNIGRAM** $p(w) = \frac{c(w)}{N}$
  - **BIGRAM** $P(w_i \mid w_{i-1}) = \frac{c(w_i, w_{i-1})}{c(w_{i-1})}$
  - **TRIGRAM** $P(w_i \mid w_{i-1}, w_{i-2}) = \frac{c(w_i, w_{i-1}, w_{i-2})}{c(w_{i-1}, w_{i-2})}$

```
class N_Gram_Model:

    <Some variables in the class : corpus, n, probabilities, probabilities_gt, probabilities_i etc.>

    def __init__(self):
        pass

    def read_file(file_path):
        <read the file here, and save it in a variable>

    def setup():
        <divide the corpus into train and test>

    def train():
        <train the model>
        <calculate the probabilities of each n-gram>
        <save the probabilities & frequencies in a variable>

    def save():
        <save the probabilities( normal, gt, interpolation) & frequencies in a file>

    def load():
        <load the probabilities( normal, gt, interpolation) & frequencies from a file>

    def perplexity(sentence):
        <calculate the perplexity of the sentence>

    def generate():
        <generate a sentence>
        <use the probabilities calculated in train() to generate the next word>

    def evaluate():
        <evaluate the model>
        <calculate the average perplexity of the train sentence>
        <calculate the average perplexity of the test sentence>

    def good_turing():
        <calculate and save the new probabilities in a variable>

    def interpolation():
        <calculate and save the new probabilities in a variable>
```

good tunning smoothing and Interpolation

- $$r^* = (r+1) * \frac{S(N_{r+1})}{S(N_r)}$$ $$P(w_i | w_{i-n+1...i-1}) = \frac{r^*}{\mathrm{Count}(w_{i-n+1...i-1})}$$
- For the assignment, you may let P be zero if denominator is zero.
  - Ideally explore backoff (optional)
- What is N_r?
  - Frequencies of Frequencies
  - Example : The frequency of n-grams with frequency of 3 (how many n-grams are there having a count of 3 in corpus)

- For S *(the smoothed/ adjusted value of the frequency)* we need to compute Z_r's and
- We need to calculate (revised N_r) Z_r from N_r

good tunning smoothing and Interpolation

- However, the larger r is, the less reasonable this substitution is, and thus becomes a problem for larger N_r's as the frequency of frequencies become very sparse as we continue to go on
- Then for N_r's which are zeros, the simplest smooth is a line, and a downward sloping log-log line will satisfy the priors on r*. This is the proposed simple smooth, and we call the associated Good-Turing estimate the Linear Good Turing (LGT) estimate.
  - You can use scipy only for linear regression (for getting the line).
  - You can try and specify at what point the switch from no smoothing to linear smoothing should take place
- Once we use an LGT estimate, then we continue to use them

frequency	frequency of frequency
r	N_r
1	120
2	40
3	24
4	13
5	15
6	5
7	11
8	2
9	2
10	1
11	0
12	3

$$log(N_r) = a + b\ log(r)$$

```
double smoothed(int i)
       {
       return(exp(intercept + slope * log(i)));
       }
```

good tunning smoothing and Interpolation

- **Interpolation :**

$$P(t_3|t_1,t_2) = \lambda_1 \hat{P}(t_3) + \lambda_2 \hat{P}(t_3|t_2) + \lambda_3 \hat{P}(t_3|t_1,t_2) \quad (6)$$

$\hat{P}$ are maximum likelihood estimates of the probabilities, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$, so P again represent probability distributions.

Unigrams: $\hat{P}(t_3) = \frac{f(t_3)}{N}$

Bigrams: $\hat{P}(t_3|t_2) = \frac{f(t_2,t_3)}{f(t_2)}$

Trigrams: $\hat{P}(t_3|t_1,t_2) = \frac{f(t_1,t_2,t_3)}{f(t_1,t_2)}$

- **Estimation of lambda weights (paper) :**
  - After we have calculated the n-gram frequencies.
  - If the denominator in one of the expressions is 0, we define the result of that expression to be 0.

set $\lambda_1 = \lambda_2 = \lambda_3 = 0$
foreach trigram t_1, t_2, t_3 with $f(t_1,t_2,t_3) > 0$
  depending on the maximum of the following three values:
    case $\frac{f(t_1,t_2,t_3)-1}{f(t_1,t_2)-1}$: increment λ_3 by $f(t_1,t_2,t_3)$
    case $\frac{f(t_2,t_3)-1}{f(t_2)-1}$: increment λ_2 by $f(t_1,t_2,t_3)$
    case $\frac{f(t_3)-1}{N-1}$: increment λ_1 by $f(t_1,t_2,t_3)$
  end
end
normalize $\lambda_1, \lambda_2, \lambda_3$

Some other misc topics

- **Handling unknown words in test set**
  - Either use an epsilon directly (of 10^-5 probability)
  - OR
  - Use <OOV> tokens
    - Find the words in the training set which have a frequency of let's say 2 (play around with the threshold)
    - Replace these words with the <OOV> token and find their frequency.
    - While testing, if an unknown word is found, replace it with <OOV> and then carry on with the normal calculation
- **Usage of log**
  - Use log and then add them up, to prevent underflow while multiplication of the probabilities

Thank You.