NLP Homework 2

Sayed Erfan Ghazi Asgar

4023614028

Apr 2024

Part 2

Q2-2

Smoothing helps in training data by addressing the issue of zero probabilities or sparse data. In natural language processing tasks like language modeling, it's common to encounter situations where certain n-grams (sequences of words) appear in the training data but not often enough to reliably estimate their probabilities. This can lead to zero probabilities for these rare n-grams, which can negatively impact the performance of the model.

Smoothing techniques, such as Laplace smoothing, help by redistributing probability mass from observed n-grams to unseen or rare n-grams, ensuring that every possible n-gram has a non-zero probability. This helps the model generalize better and reduces the risk of overfitting to the training data.

During training, smoothing ensures that even rare n-grams contribute to the overall probability estimates, making the model more robust and better able to capture the underlying patterns in the data. By avoiding zero probabilities, smoothing helps prevent the model from becoming overly confident about unseen or rare n-grams during training, which can lead to poor performance when the model encounters unseen data during testing.

In summary, smoothing in training data helps improve the reliability and generalization ability of the model by addressing the issue of zero probabilities or sparse data, thereby contributing to more accurate probability estimates for both seen and unseen n-grams.

- Laplace smoothing:

  Laplace smoothing addresses this issue by adding a small constant (usually 1) to the count of each word or sequence in the training data before calculating probabilities. This ensures that no probability estimate is zero, and it gives some probability mass to unseen events. The formula for Laplace smoothing can be expressed as:

  P(w_i|w_{i-1} = (count(w_{i-1}w_i)+ 1)/(count(w_{i-1}) + V)

  Where:

  - count(w_{i-1}w_i) is the count of the word sequence (w_{i-1}w_i) in the training corpus.

  - count(w_{i-1}) is the count of the preceding word (w_{i-1}) in the training corpus.

  - (V) is the vocabulary size (i.e., the total number of unique words in the vocabulary).

  By adding 1 to the numerator and the denominator, Laplace smoothing ensures that each word or sequence has a non-zero probability estimate, and the total probability mass is redistributed among all possible events, including unseen ones. While simple, Laplace smoothing effectively prevents zero probabilities and improves language models' robustness, especially with limited training data. However, it may introduce biases towards unseen events due to the uniform addition of counts. Other techniques like Lidstone or Good-Turing smoothing offer alternatives to address these shortcomings.

- Good-Turing smoothing:

  Good-turning smoothing is a technique used to estimate the probabilities of unseen events based on the frequencies of seen events in a dataset. It addresses the issues of zero probabilities and sparse data by extrapolating from observed data to estimate the probabilities of unseen events.

  Here's how Good-Turing smoothing works:

  Count the Frequencies: First, we count the frequencies of each event (e.g., words or n-grams) in the dataset.

  Estimate the Frequencies of Unseen Events: Good-turing smoothing estimates the probabilities of unseen events by extrapolating from the frequencies of observed events. It assumes that the probability of seeing an unseen event is proportional to the number of observed events that occurred exactly once (singleton events).

  Adjust the Probabilities: The probabilities of seen events are adjusted based on the estimated probabilities of unseen events. This adjustment helps redistribute probability mass to unseen events, ensuring that they have non-zero probabilities.

  here are the formulas for Good-Turing smoothing:

  1. **Frequency Counts**: Let (N_r) denote the number of events that occur (r) times in the training data. For example, (N_1) is the number of singletons (events that occur only once), (N_2) is the number of events that occur twice, and so on.

2. **Adjusted Counts**: We calculate adjusted counts (N_r^*) by applying Good-Turing smoothing:

[N_r^* = (r+1) frac{N_{r+1}}{N_r}]

3. **Estimated Probabilities**: After obtaining the adjusted counts, we estimate the probability of each event occurring. The probability is estimated directly from the adjusted count for events with non-zero counts. We use the adjusted count of singletons (events that occur only once) for events with zero counts to estimate the probability of unseen events.

4. **Probability Estimate for Unseen Events**: Let (N_1) be the count of singletons (events that occur only once). The probability estimate for unseen events is calculated as follows:

[P_{text{unseen}} = frac{N_1^*}{N}]

where (N) is the total number of events in the corpus.

These formulas provide the basic framework for implementing Good-Turing smoothing in NLP tasks.

Let's walk through a simple example of Good-Turing smoothing using word frequencies in a corpus.

Suppose we have the following word frequencies in our training corpus:

- "the": 1000 occurrences

- "cat": 100 occurrences

- "dog": 50 occurrences

- Other words with various frequencies

We'll use these frequencies to calculate adjusted counts and estimate the probability of unseen events, such as the word "elephant."

1. **Frequency Counts**:

   - (N_1): Number of singletons = 500

   - (N_2): Number of events that occur twice = 200

   - (N_3), (N_4), ...: Various counts for other words

2. **Adjusted Counts**:

- Adjusted count for singletons ((N_1^*)) = ((1+1) frac{N_2}{N_1} = 2 times frac: 2*{200}/{500} = 0.8)

- Adjusted count for events with count 2 ((N_2^*)) = ((2+1) frac{N_3}{N_2})

- Adjusted counts for other frequencies

3. **Probability Estimate for Unseen Events**:

   - Using the adjusted count for singletons ((N_1^*)), we estimate the probability of unseen events:

(P_{text{unseen}} = frac{N_1^*}{N} = frac{0.8}{N_{text{total}}})

Let's say the total number of words in the corpus is 10,000. Then:

[P_{text{unseen}} = frac{0.8}{10000} = 0.00008]

This means that if we encounter a previously unseen word in the corpus, such as "elephant," we estimate its probability to be approximately (0.00008).

In practice, these calculations would be more nuanced, taking into account various factors like discounting and normalization. However, this example illustrates the basic concept of Good-Turing smoothing in estimating probabilities for unseen events in NLP tasks.

Q2-6

1. **Size of Training Corpus**:

- **Large Corpus**: With a large corpus, you can afford to use larger values of (n) because there's more data available to learn from. This allows the model to capture more complex patterns and dependencies in the language.

- **Small Corpus**: In contrast, with a smaller corpus, choosing too large an (n) might lead to overfitting or sparse data issues. The model may not generalize well to unseen data, and it might struggle to provide accurate predictions.

2. **Language Complexity**:

- **Simple Language**: For languages with straightforward grammar and vocabulary, smaller values of (n) may be sufficient to capture most of the relevant context. Examples include controlled domains like technical documentation or simplified languages.

- **Complex Language**: Languages with rich vocabulary, intricate grammar rules, and nuanced semantics may require larger values of (n) to capture the complexity of the language adequately. This is common in natural language processing tasks involving literature, social media text, or conversational speech.

3. **Computational Resources**:

- **Memory and Processing Power**: Larger values of (n) increase the memory and computational requirements of the language model. Storing and processing a large number of n-grams can strain resources, particularly during training and inference. Therefore, the choice of (n) should balance model complexity with available computational resources.

4. **Data Sparsity**:

- **Rare Events**: As (n) increases, the probability estimates for rare n-grams become sparser because the occurrences of longer sequences are less frequent in the training data. This can lead to unreliable probability estimates and poor model performance, especially if not addressed properly through techniques like smoothing.

5. **Context Sensitivity**:

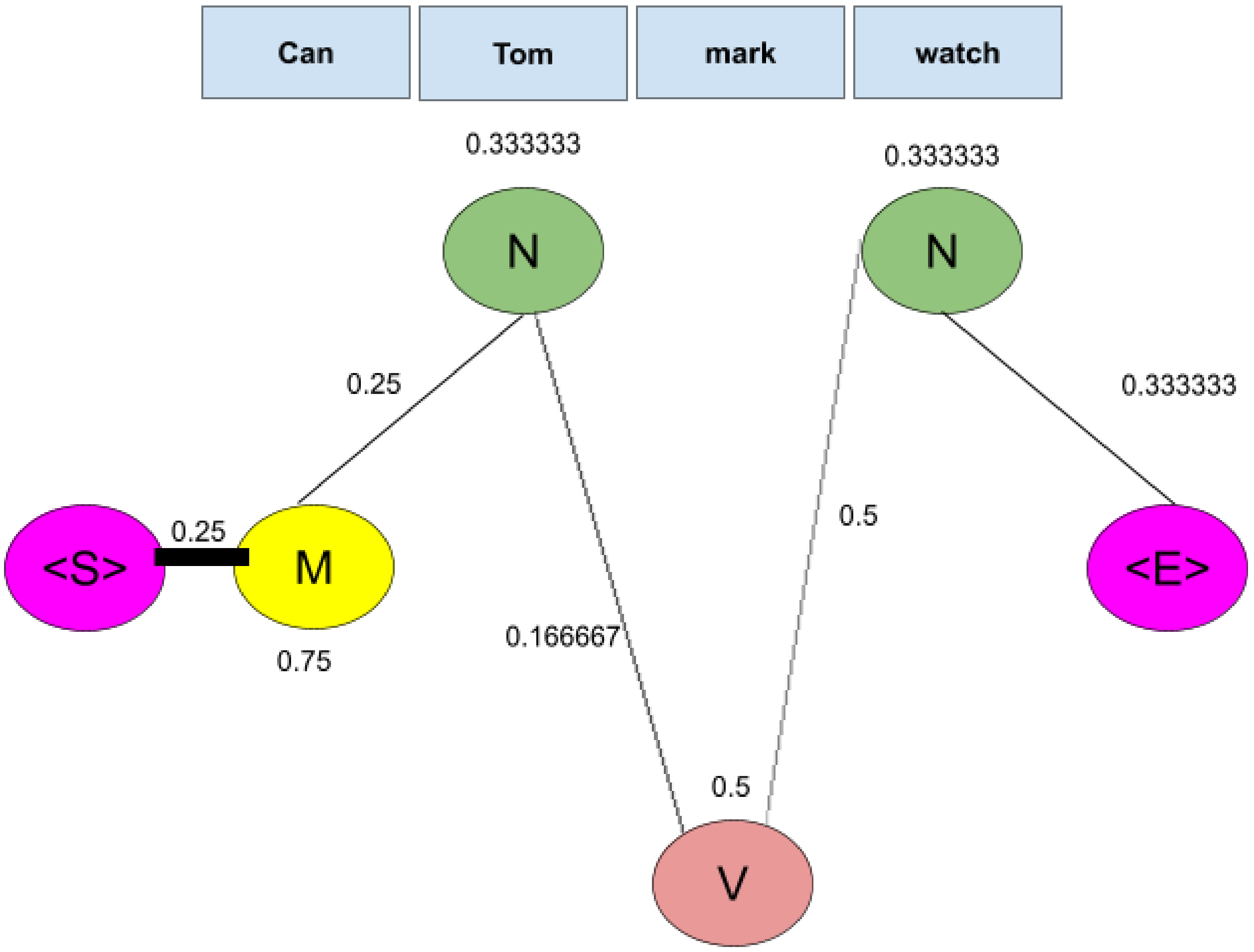
- **Local vs. Global Dependencies**: The choice of (n) influences the model's ability to capture different levels of contextual dependencies. Smaller values of (n) capture local dependencies within short sequences of words, while larger values capture longer-range dependencies across broader contexts. The appropriate choice depends on the specific requirements of the task.

6. **Application Specifics**:

- **Task Requirements**: Consider the goals and constraints of the application or task for which the language model will be used. Different tasks may require different levels of context sensitivity and model complexity. For example, sentiment analysis may benefit from local context, while document summarization may require a broader understanding of the text.

In summary, choosing the value of (n) in n-grams language modeling is a nuanced process that involves balancing factors such as data availability, language complexity, computational resources, and task requirements. Experimentation and evaluation are essential to determine the optimal (n) for a given application and dataset.

HMM Graph



<S>→M→N→V→N→<E>

= 0.25*0.75*0.25*0.333333*0.166667*0.5*0.5*0.333333*0.333333

= 7.233789062478298e-05