201 Assignment 2

Shriya Sravani Y
UCSC
sy4@ucsc.edu

November 17, 2024

This assignment focuses on building n-gram language models using Maximum Likelihood Estimation. Part 1 does not use any smoothing techniques while Part 2 uses smoothing techniques.

Language modeling is a foundational task in natural language processing (NLP) that involves predicting the likelihood of sequences of words. N-gram language modeling represents a probabilistic framework that uses sequences of n consecutive words to estimate the likelihood of a sentence. The models are evaluated on their perplexity scores across training, development and test datasets, along with an evaluation on a sample sentence: "HDTV .". Perplexity is a measure of how well a probability distribution predicts a sample. Lower perplexity values indicate better predictive performance of the model. The requirements of the assignment are as follows:

- To build and evaluate unigram, bigram, and trigram language models using MLE without smoothing and with smoothing.
- Handling out-of-vocabulary (OOV) words by replacing tokens occurring fewer than three times with a special `<UNK>` token.
- Ensuring the resulting vocabulary, including `<UNK>` and `<STOP>` but excluding `<START>`.
- Reporting perplexity scores for the unigram, bigram, and trigram models on the training, development, and test datasets.
- Debugging and validating the implementation using the example sentence `"HDTV ."` with expected perplexity scores as given.
- Using the development data to determine the best hyperparameters $\lambda_1, \lambda_2, \lambda_3$.
- Reporting perplexity scores for training and development sets for different combinations of $\lambda_1, \lambda_2, \lambda_3$ (up to 5 sets).

This report focuses on building unigram, bigram, and trigram language models using Maximum Likelihood Estimation (MLE) without smoothing. Additionally, linear interpolation smoothing is implemented to improve the model's performance on unseen data.

1 Part 1 - MLE model without using smoothing

- The sentences are tokenized, and tokens occurring fewer than three times are replaced with the `<UNK>` token.
- The `<STOP>` token is added at the end of each sentence to indicate the sentence boundary.
- The n-grams are extracted and their MLE probabilities are calculated. Unseen tokens are kept a track of and handled.
- The resulting vocabulary (including `<UNK>` and `<STOP>`, but excluding `<START>`) contains **26,602 unique tokens**.
- The perplexity of the n-grams is calculated.
- For bigrams and trigrams, `<START>` tokens are added at the beginning of sentences to provide necessary context.
- A sample line "HDTV ." is used to check if the MLE is implemented correctly and its n-gram complexity is considered.

1.1 Perplexity Scores

Model	Train Perplexity	Dev Perplexity	Test Perplexity
Unigram	976.54	892.25	896.50
Bigram	77.07	28.29	28.31
Trigram	7.87	2.99	2.99

Table 1: Perplexity Scores for Unigram, Bigram, and Trigram Models

Validation with `"HDTV ."`

Model	Perplexity
Unigram	658.04
Bigram	63.71
Trigram	39.48

Table 2: Perplexity Scores for the Sample Sentence: `"HDTV ."`

- The vocabulary size of 26,602 tokens aligns with the requirements of replacing infrequent tokens with `<UNK>`. This ensures that the language model effectively handles out-of-vocabulary (OOV) words while retaining a manageable vocabulary size.

- Perplexity Analysis:
  - The unigram model has the highest perplexity across all datasets, reflecting its inability to capture dependencies between words. This model assumes word independence, which limits its predictive power.
  - The bigram model shows a significant reduction in perplexity compared to the unigram model by incorporating the preceding word as context. This highlights the improvement gained by considering word dependencies.
  - The trigram model achieves the lowest perplexity by utilizing two preceding words for context, demonstrating the benefits of higher-order dependencies in predicting word sequences.
- The perplexity scores for the sample sentence `"HDTV ."` closely match the expected values for unigram, bigram, and trigram models (658, 63.7, and 39.5, respectively). This validates the correctness of the implementation and the effectiveness of the models.
- Performance Across Datasets:
  - Perplexity scores are generally lower for the development and test sets compared to the training set for the bigram and trigram models. This indicates that these models generalize well to unseen data, effectively capturing dependencies in word sequences.
  - The unigram model's higher perplexity across all datasets reinforces its limitations in handling contextual dependencies.

This experiment demonstrates the progression of language modeling capabilities from unigram to trigram models. Higher-order models significantly improve the prediction of word sequences by incorporating more context. However, the lack of smoothing leads to zero probabilities for unseen n-grams, which can be addressed in future work through interpolation or backoff methods. These findings highlight the importance of context in language modeling and establish a foundation for exploring advanced smoothing techniques.

2 Part 2 - MLE using Smoothing with linear interpolation

For this part, I implemented and evaluated unigram, bigram, and trigram language models using Maximum Likelihood Estimation (MLE) combined with linear interpolation smoothing. The primary objectives are to:

- Smooth the probabilities of the language models using linear interpolation.
- Evaluate perplexity scores on training, development, and test sets.
- Investigate the impact of using half the training data and different OOV thresholds on perplexity.

The five different sets of lambda values are used as the hyperparameters and the best configuration is chosen based on the development set.

Perplexity Scores for Different λ Values

The following table summarizes the perplexity scores for the training and development datasets for five combinations of $\lambda_1, \lambda_2, \lambda_3$:

λ_1	λ_2	λ_3	**Train Perplexity**	**Dev Perplexity**
0.3	0.3	0.4	15.3009	286.6349
0.1	0.3	0.6	11.1515	352.2342
0.2	0.4	0.4	14.6972	286.3615
0.3	0.5	0.2	22.4354	262.8898
0.4	0.4	0.2	23.6370	267.6352

Table 3: Perplexity Scores for Training and Development Sets

The best hyperparameters were determined based on the development perplexity. The combination $\lambda_1 = 0.3$, $\lambda_2 = 0.5$, $\lambda_3 = 0.2$ achieved the lowest perplexity on the development set.

- **Best Hyperparameters:** $\lambda_1 = 0.3, \lambda_2 = 0.5, \lambda_3 = 0.2$
- **Test Set Perplexity:** 262.6638

Debugging Example: Sample Sentence "HDTV ." To validate the implementation, we calculated the perplexity for the sample sentence "HDTV ." using $\lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.6$. The expected perplexity for this combination is 48.1, and the calculated value was consistent: Sample Perplexity for "HDTV .":48.1135 (Expected: 48.1)

1. **Hyperparameter Optimization:** Experimented with five different sets of λ values, aiming to balance the contributions of unigram, bigram, and trigram probabilities. The best performance was achieved with $\lambda_1 = 0.3, \lambda_2 = 0.5, \lambda_3 = 0.2$.
2. **Development Perplexity Trends:** The perplexity values showed that larger weights on bigram or trigram probabilities can reduce perplexity, as they capture more contextual information.
3. **Debugging and Validation:** The sample perplexity for "HDTV ." matched the expected value, ensuring the correctness of the interpolation and smoothing implementation.
4. **Generalization to Test Set:** The best hyperparameters resulted in a test perplexity of 262.6638, demonstrating reasonable generalization.

Using **half the training data** produced the following perplexity scores:

- **Best Lambdas**: $(0.4, 0.4, 0.2)$ were found to minimize the development perplexity.
- **Test Set Perplexity**: The test perplexity with half the training data was 311.5497.

Increased Perplexity on Unseen Data:

Lambdas	Train Perplexity	Dev Perplexity
(0.3, 0.3, 0.4)	11.9822	343.3063
(0.1, 0.3, 0.6)	8.6749	454.1919
(0.2, 0.4, 0.4)	11.5087	351.2233
(0.3, 0.5, 0.2)	17.7596	310.2936
(0.4, 0.4, 0.2)	18.7207	310.1184

Table 4: Perplexity Scores Using Half the Training Data

- The perplexity values for the development and test sets increased significantly when using half the training data compared to the full dataset.
- This suggests that the model generalizes less effectively to unseen data when trained on a smaller dataset.

Impact on Train Perplexity: Training perplexity was lower due to overfitting on the smaller dataset, which results in a loss of generalization capability.

Reducing the training data increased perplexity on unseen data due to decreased coverage of the vocabulary and n-gram contexts. This highlights the need for larger datasets to improve model generalization.

Handling OOV Tokens:

- Tokens appearing less than 5 times in the training data were replaced with <UNK>.
- This new threshold was compared to a baseline where only tokens appearing just once were replaced with <UNK>.

Evaluation Metrics:

- Perplexity scores were calculated for:
  - Training data.
  - Development data.
  - Test data.
- Smoothing was applied using 5 sets of interpolation parameters:

$$(\lambda_1, \lambda_2, \lambda_3) = (0.3, 0.3, 0.4), (0.1, 0.3, 0.6), (0.2, 0.4, 0.4), (0.3, 0.5, 0.2), (0.4, 0.4, 0.2).$$

The perplexity scores obtained after adjusting the <UNK> threshold to 5 are summarized below:

1. **Improved Generalization with Higher <UNK>Threshold**:
   - Increasing the <UNK>threshold to 5 reduced test perplexity to 136.8794, compared to models trained with a lower threshold.
   - This suggests that replacing infrequent tokens with <UNK> reduces sparsity, improving the model's ability to generalize to unseen data.

Lambdas	Train Perplexity	Dev Perplexity
(0.3, 0.3, 0.4)	16.4014	148.1127
(0.1, 0.3, 0.6)	12.0530	167.0396
(0.2, 0.4, 0.4)	15.7611	144.8426
(0.3, 0.5, 0.2)	23.7410	138.7476
(0.4, 0.4, 0.2)	24.9911	143.4546

Table 5: Perplexity Scores for Adjusted <UNK> Threshold

2. **Impact on Development Perplexity**:
   - The best perplexity on the development set was achieved with $\lambda = (0.3, 0.5, 0.2)$, yielding 138.7476.
   - This reinforces the effectiveness of linear interpolation with optimized weights.
3. **Comparison to Baseline**:
   - When fewer tokens were replaced with <UNK> (e.g., only those appearing once), the model suffered higher perplexity due to increased sparsity and overfitting to rare tokens.

Replacing tokens that appeared fewer than 5 times with `<UNK>` reduced sparsity and improved test perplexity to **136.8794**. This suggests that handling infrequent tokens effectively can significantly impact language model performance.

Conclusion

The first part of the task explores how the perplexities of the n-gram model change when run with an MLE and no smoothing, while the second part shows that introducing linear interpolation smoothing to improves the performance of language model. By tuning interpolation weights and managing rare words appropriately, we achieved improved performance, especially on unseen data. This demonstrates the model's ability to generalize effectively.