CSE 143 Assignment 2

Shivani Belambe, Varun Palanisamy

November 6, 2024

Part 1: N-gram Language Modeling

Description

We implemented unigram, bigram, and trigram language models using a shared `NGramModel` base class. We then trained each model with MLE and replaced low-frequency tokens with `<UNK>`. The models included `<START>` and `<STOP>` tokens for sentence boundaries. Below are descriptions of each of the models.

- **Unigram Model:** This model considers each word as an independent figure and only looks at its frequency in the whole text.
- **Bigram Model:** In this model, the probability of the next words depends on the previous word.
- **Trigram Model:** Similar to Bigram, this model looks at the previous words to make a prediction; the only difference is that it utilizes two words instead of just one.

Perplexity Results

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

Dataset	**Unigram Perplexity**	**Bigram Perplexity**	**Trigram Perplexity**
Train Set	976.54	77.07	7.87
Development Set	892.25	∞	∞
Test Set	896.50	∞	∞
HDTV Test Set	658.04	63.71	39.48

Discussion of Results

The perplexity values from train, dev, and test decrease from unigram to trigram, with the biggest decrease being from unigram to bigram in the train set. The lower the perplexity, the more effectively the next word is being predicted.

We were able to see succesful resuts for the HDTV test and train sets. However, the perplexity values reach ∞ (zero probabilities) for the bigram and trigram sections of dev and test set, meaning that there is definitely some overfitting. The next steps would be to implement smoothing for unseen n grams.

Part 2: Adding Smoothing

Description

We now added additive smoothing using the parameter alpha in `NGramModel`. With the new paramter, the goal was to smooth the probability estimates and to not have any more zero probabilities for unseen n-grams. We used alpha = 1 for the default smoothing, and then we tried higher values, but ultimately found that values between 0 and 1 were the most effective.

Model	Alpha = 1	Alpha = 0.01	Alpha = 0.00001
Unigram	977.5	976.5	976.5
Bigram	1442.3	157.9	77.4
Trigram	6244.7	169.9	8.49

Table 2: Perplexity Scores on the Training Set

Model	Alpha = 1	Alpha = 0.01	Alpha = 0.00001
Unigram	894.4	892.3	892.2
Bigram	1042.5	145.8	113.9
Trigram	16888.2	6895.1	22469.3

Table 3: Perplexity Scores on the Dev Set

Model	Best Alpha	Perplexity
Unigram	0.00001	896.5
Bigram	0.00001	114.9
Trigram	0.01	6823.7

Table 4: Perplexity Scores on the Test Set

Conclusion after adding alpha

We noticed that with a alpha=1, the train set perplexity scores were very large and not representative of the results we got without alpha. We tested larger values, but those gave even larger perplexity values, so we tested out 0.01 and 0.00001. With 0.01, we got better scores than with 1, and even better scores with 0.00001. For our unigram and bigram model, we saw that alpha = 0.00001 gave a perplexity score that was the smallest, while a score of 0.01 for trigram was better for the dev and test set. We believe this is the case due to over-fitting during the training. Despite performing the best during training, it proved to be ineffective for the dev and test set, where the perplexity value shot up rather than decrease.

Part 3: Linear Interpolation Smoothing

Description

After adding smoothing and seeing better results, we added interpolation smoothing to make the model more optimal. We used linear interpolation to combine unigram, bigram, and trigram models with weights λ_1, λ_2, and λ_3, with all of them summing to 1.

Table 5: Perplexity Scores on Training and Development Sets for Various λ Values

λ_1	λ_2	λ_3	Train Perplexity	Dev Perplexity
0.1	0.3	0.6	13.48	370.74
0.3	0.3	0.4	17.71	306.38
0.2	0.4	0.4	17.15	305.56
0.4	0.3	0.3	21.51	298.41
0.5	0.3	0.2	27.98	299.58

Test Set Perplexity with Best Hyperparameters

Based on the development set, the best lambda values were $\lambda_1 = 0.1$, $\lambda_2 = 0.3$, and $\lambda_3 = 0.6$, achieving a test set perplexity of 303.63.

Impact of Using Half the Training Data

To examine our model's strength with half the train dataset, we modified the code to only look at the first 30765 lines. It was predicted that the perplexity would increase on unseen data due to the model having fewer observations to base its estimates on, however, the values on the dev set decreased, meaning that there was over-fitting happening. The values were then different for the test, where there were zero probabilities, meaning that perplexity score was likely inaccurate and the train set wasn't able to get the model to accurately predict probability estimates. The results of this experiment can be seen in table 6.

Impact of Converting Low-Frequency Tokens to `<UNK>`

In our second experiment, we converted all tokens that appeared fewer than 5 times to `<UNK>`. The results were similar to when we cut the training dataset

Table 6: Perplexity Results for Linear Interpolation with Different λ Values (Half Training Data)

Interpolation Weights	**Perplexity on Train Set**	**Perplexity on Dev Set**	**Perplexity on Tes**
$\lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.6$	11.25	360.75	-
$\lambda_1 = 0.3, \lambda_2 = 0.3, \lambda_3 = 0.4$	14.78	289.93	362.41
$\lambda_1 = 0.2, \lambda_2 = 0.4, \lambda_3 = 0.4$	14.33	292.16	-
$\lambda_1 = 0.4, \lambda_2 = 0.3, \lambda_3 = 0.3$	17.96	279.32	-
$\lambda_1 = 0.5, \lambda_2 = 0.3, \lambda_3 = 0.2$	23.38	277.21	-

in half. The perplexity scores for the train and dev set were lower than before, however, the test perplexities contained many zero probabilities. The only perplexity score for this experiment that was successful was the same as the original experiment with no modification to the UNK formatting of the train set, meaning that this experiment ultimately had no effect, despite improving perplexity within the dev set.

Table 7: Perplexity Results for Linear Interpolation with Different λ Values (Full Training Data)

Interpolation Weights	**Perplexity on Train Set**	**Perplexity on Dev Set**	**Perplexity on Tes**
$\lambda_1 = 0.1, \lambda_2 = 0.3, \lambda_3 = 0.6$	14.43	304.02	-
$\lambda_1 = 0.3, \lambda_2 = 0.3, \lambda_3 = 0.4$	18.85	256.85	303.63
$\lambda_1 = 0.2, \lambda_2 = 0.4, \lambda_3 = 0.4$	18.27	254.57	-
$\lambda_1 = 0.4, \lambda_2 = 0.3, \lambda_3 = 0.3$	22.79	251.86	-
$\lambda_1 = 0.5, \lambda_2 = 0.3, \lambda_3 = 0.2$	29.42	254.34	-

Conclusion

After adding certain lambda values and improving the probability estimate predictions using linear interpolation, we were able to see some improvement within the train and dev set. By addressing zero probabilities, we were able to obtain more successful values than by just implementing the n gram models and the n gram models with smoothing.