HW2: Building ngram language models (LM) from scratch

Team: Semantic Savants; Jeonghoon Kim, Matthew Olson, and Marco Berriodi

Our group, Semantic Savants, conducted research with ASCII encoding text files. ASCII encoding is limited to only 128 characters and does not support characters from non-Latin scripts[1]. However, the given text files were predominantly written in English, so it seemed ok to use ASCII encoding over UTF-8 encoding. Furthermore, using UTF-8 may have added complexity to the data which could not be overcome by the size of our training sets.

Bigram was implemented for the classifier since when we used with trigram, it did more operations and consumed more memory space, which made the training and perplexity calculations slow when done on our relatively small computing power. When it came to preprocessing, our group first tried solely using “word_tokenize” which includes punctuation as their own tokens. However, during inference the model tended to generate commas or periods as the first token because both were in the top-n most frequent tokens. To solve this we used RegexpTokenizer to extract English letters while removing special characters and new lines[2].

The MLE model could not handle out-of-vocabulary words and had difficulty classifying sentences. To address these problems, we applied smoothing, interpolation, and smoothing techniques. For smoothing, we used the Laplace and KneserNeyInterpolated models from the nltk library. The latter model also automatically incorporates interpolation into Kneser-Ney smoothing. By comparison, Laplace implements Laplace (add one) smoothing, which is an initialization identical to BaseNgramModel where gamma is always 1. The nltk StupidBackoff model was chosen to add backoff to the models. Each of these different techniques helped to handle out-of-vocabulary words and to avoid zero probabilities, which reduced infinite

perplexity values and drastically increased the classification performance. Note that there were still some infinite values, which we excluded from the accuracy data because they interfered with classification[3].

Based on our data from Table 1, we determined that the best model for each author was: Laplace for Austen, KneserNeyInterpolated/StupidBackoff for Dickens, KneserNeyInterpolated/StupidBackoff for Tolstoy, and KneserNeyInterpolated for Wilde. Each smoothing and backoff technique had different results. Laplace had variable accuracy but was very good on the Austen text, although the model tended to have higher perplexities than other smoothing methods. KneserNeyInterpolated had decent accuracy but was far too computationally expensive, taking many times longer to run than the other models. StupidBackoff was the clear winner, since it works relatively fast and gave better overall accuracy than the other smoothing methods. If better resources (CPU/GPU/RAM) were available, we could conduct more trials with different variables like trigrams, other processing methods, and different smoothing methods to get find more suitable models for our classification task.

Lm Improvement Tech	Trail	Austen Accuracy	Dickens Accuracy	Tolstoy Accuracy	Wilde Accuracy	Best Trial Accuracy	Technique Accuracy Average
MLE	1	3.02%	3.38%	5.02%	2.43%	5.02%	3.46%
MLE	2	3.29%	2.89%	4.93%	1.95%	4.93%	3.27%
MLE	3	2.56%	3.28%	3.91%	2.07%	3.91%	2.96%
Average Accuracy		2.96%	3.18%	4.62%	2.15%	4.62%	3.23%
Stupid Backoff	1	65.11%	48.98%	63.25%	47.81%	65.11%	56.29%
Stupid Backoff	2	64.25%	51.69%	65.11%	45.61%	65.11%	56.67%
Stupid Backoff	3	66.28%	46.39%	60.16%	78.02%	78.02%	62.71%

Average Accuracy		65.21%	49.02%	62.84%	57.15%	58.56%	58.56%
KneserNeyInterpolated	1	61.41%	48.59%	62.63%	46.46%	62.63%	54.77%
KneserNeyInterpolated	2	64.98%	48.01%	63.68%	49.75%	64.98%	56.61%
KneserNeyInterpolated	3	64.98%	50.04%	63.25%	48.90%	64.98%	56.79%
Average Accuracy		63.79%	48.88%	63.19%	48.37%	64.20%	56.06%
Laplace	1	95.40%	36.30%	56.40%	39.10%	95.40%	56.80%
Laplace	2	95.59%	35.84%	53.56%	38.05%	95.59%	55.76%
Laplace	3	96.24%	37.19%	53.25%	39.75%	96.24%	56.61%
Average Accuracy		95.74%	36.44%	54.40%	38.97%	95.74%	56.39%

Table 1. The results of accuracy for each author you get with the given data with an automatically extracted development set (=10 percent of the dataset)

By considering both computational complexity and the accuracy results from Table 1, StupidBackoff combined with the bigram language model looked like the optimal model. We created the samples from Table 2 using the “lm.generate()” on the Austen model. As you can see, the perplexity of the samples on the Austen model is by far the lowest in all cases, which is exactly the expected result.

Sample	Perplexity on Austen model	Perplexity on Dickens model	Perplexity on Tolstoy model	Perplexity on Wilde model
['actually', 'saw', 'her', 'had', 'said', 'she', 'lay', 'out', 'Lord', 'my']	71.431	679.987	839.425	623.285
['think', 'you', 'a', 'happy', 'she', 'could', 'not', 'received', 'his', 'head']	60.565	175.691	113.788	156.372
['was', 'returning', 'to', 'get', 'nothing', 'may', 'seem', 'to', 'be', 'only']	56.619	108.867	152.489	211.647
['had', 'nothing', 'of', 'that', 'there', 'will', 'For', 'my', 'dear', 'Miss']	37.497	121.819	152.262	146.659
['alone', 'and', 'a', 'just', 'as', 'myself', 'but', 'this', 'time', 'with']	94.842	121.724	158.208	166.812

Table 2. The results of 5 samples generated from the models and the corresponding perplexities

During the evaluation of the language models, we tracked the perplexity scores and the generated words for each model. In particular, we were interested in identifying the words that corresponded to the minimum perplexity score for each model. The perplexity score measures how well the language model predicts the next word in a sequence of words, based on the preceding words in the sequence with a lower perplexity score meaning more accurate prediction. Furthermore, ngram function has been built with numpy and utilized for the bonus point, and the lowest perplexity in development set is shown in the Table 3.

Author	Perplexity Score	Words
Austen	23.25	to, hear, it, I, assure, you
Dickens	17.09	it, would, have, been
Tolstoy	23.73	Ah, I, m, very, glad
Wilde	21.85	I, entreat, you

Table 3. Perplexity and the word of the lowest perplexity of each model.

References

[1]OpenAI, “ChatGPT,” chat.openai.com. [Online]. Available: https://chat.openai.com/

[2]“python - How to get rid of punctuation using NLTK tokenizer?,” Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/15547409/how-to-get-rid-of-punctuation-using-nltk-tokenize. [Accessed: Mar. 01, 2023]

[3]J. Daniel and J. Martin, “Speech and Language Processing,” Jan. 2023 [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/3.pdf