Project 2 N-gram Models

Created	@March 12, 2025 3:18 PM
Class	LING 406

Letter Bigram Model Question Set

1. What do you consider a token for this task and why? What kind of preprocessing steps do you need to apply before feeding the data into your language model?

- In this task, a token is an individual letter rather than a word since we are using a letter bigram model rather than a word-based one.
- Punctuation is kept but spaces are excluded as they do not contribute meaningfully to character-level modeling and could introduce unnecessary noise.
- **Preprocessing steps applied:**
  - Convert all text to lowercase to maintain consistency and avoid treating uppercase and lowercase versions of the same letter as different tokens.
  - Remove spaces and newlines to ensure the model learns meaningful letter sequences rather than spacing patterns.

2. What technique do you decide to use for out-of-vocabulary (OOV) words and why?

- Since this is a letter-based model rather than a word-based one, we do not deal with OOV words but rather OOV letter bigrams (unseen letter pairs).
- One of the implementations is without smoothing, unseen bigrams are assigned a probability of **zero**, leading to a log probability of **negative infinity**, making the model very sensitive to missing bigrams. (Training models only based on observed counts without adjusting for unseen bigrams)

- Another version is with Add-One smoothing implemented, which adds 1 to all bigram counts to avoid zero probabilities and handle unseen letter combinations more gracefully.

3. Can the letter bigram model be implemented without any kind of smoothing? If not, is add-one smoothing appropriate, or do you need better algorithms? Why (not)?

- The letter bigram model can be implemented without smoothing. However, this approach makes the model highly brittle—if a sentence contains even a single unseen bigram, its probability becomes zero (log probability = -∞), making classification unreliable.
- Add-One smoothing helps with higher accuracy prediction output but is not the best solution for letter bigrams.
    - The number of wrong predictions is 24 without Add-One smoothing and decreased to only 1 with the smoothing implemented.
    - Add-one smoothing uniformly assigns a small probability to all unseen bigrams, but it is a very basic technique that does not consider the natural distribution of letters in a language.
    - A better alternative would be Good-Turing smoothing or backoff models, which estimate probabilities more realistically.

Word Bigram Model Question Set I

1. What do you consider as a word for this task and why? What kind of preprocessing steps do you need to apply before feeding the data into your language model?

- In this task, I consider words to be sequences of alphabetic characters separated by spaces. Punctuation marks are treated as separate tokens to maintain consistency with the letter-based bigram. The preprocessing steps help in maintaining a clean and uniform representation of the data for training the bigram model.
- **Preprocessing steps applied:**

- Converting text to lowercase to avoid case sensitivity issues.
- Replacing newlines with spaces.
- Splitting text into words based on spaces.

2. What technique do you decide to use for out-of-vocabulary (OOV) words and why?

- To handle OOV words, add-one smoothing is applied. In this approach, every possible bigram, including those not seen in training, is assigned a small probability. This ensures that when an unseen word appears in the test data, it does not lead to a probability of zero or negative infinity in log-space computations. Without handling OOV words, the model would completely fail on sentences containing even a single unknown word.

3. Can the word bigram model be implemented without any kind of smoothing?

- The word bigram model can technically be implemented without smoothing, but it does not work well in practice. Without smoothing:
  - Any unseen bigram (which is very common, given the vast vocabulary of natural language) would receive a probability of zero, leading to negative infinity when computing log probabilities.
  - The model would fail to generalize beyond the training data and would break whenever a new bigram appears in the validation/test set.
- With Add-One smoothing implemented, the number of wrong predictions found in the output file decreased from 193 to only 4 incorrect answers.
- Add-One smoothing is simple and effective for handling unseen bigrams, but it distributes probability mass too evenly across all bigrams, including unrealistic ones.
- A better alternative would be Good-Turing smoothing, which redistributes probability more effectively by considering word frequency distributions. However, these are more complex to implement.

Word Bigram Model Question Set II

What do you do when the number of words seen once are unreliable? What strategy do you use to smooth unseen words?

1. **Fallback Probability for Unseen Words**
   - The code assigns a probability for unseen bigrams using:

     ```
     unseen_prob = (frequency_of_counts[1] / total_bigrams) if frequency_o
     f_counts[1] > 0 else (1 / total_bigrams)
     ```

   - If `N(1)` exists, it is used directly.
   - If `N(1) == 0`, a small fallback probability (`1 / total_bigrams`) is assigned to unseen bigrams.
2. **Avoiding Zero Division Errors**
   - In `compute_sentence_probability()`, the code ensures division by zero is avoided:

     ```
     total_prob_mass = sum(first_word_probs.values())
     if total_prob_mass > 0:
         prob = first_word_probs.get(second, first_word_probs.get("__unsee
     n__", 1 / total_prob_mass))
     else:
         prob = 1e-10  # Small fallback probability
     ```

   - If a word has no bigram counts, it defaults to a tiny probability (`1e-10`) instead of crashing.

Final Reflection

Model Performance

- Letter-based Add-One smoothing: 24 wrong predictions (worst)
- Word-based Add-One smoothing: 4 wrong predictions
- Word-based Good-Turing smoothing: 1 wrong prediction (best)

Clearly, the Word-based Good-Turing bigram model is the most accurate, making only one mistake, while the Letter-based model performs the worst with 24 errors.

Model Comparison

Model	Wrong Predictions	Strengths	Weaknesses
Letter-based Add-One smoothing	24	Works for languages with rich morphology; useful for handling unknown words; fast	Ignores word-level meaning; struggles with long-distance dependencies
Word-based Add-One smoothing	4	Simpler probability assignment; better than letter-based	Add-One overestimates unseen words, leading to poor probability distribution
Word-based Good-Turing smoothing	1	More accurate smoothing; redistributes probability effectively; handles unseen words well	More computationally expensive than Add-One

Verdict

The Word-based Good-Turing bigram model is the best, as it minimizes wrong predictions while effectively handling unseen words. The Letter-based model performs the worst due to its inability to capture meaningful word structures. The Word-based Add-One smoothing

model is a good alternative, but its tendency to over-smooth unseen words makes it less accurate than Good-Turing.