CS6320 Assignment 1

https://github.com/saivikas10/Assignment.git

Group14	Vijay Sai Dukkipati	Sai Vikas Thiruveedula	Manikanta Sai Kommireddy
	DXV220040	SXT230026	MXK220132

1. Implementation Details

We have implemented the Unigram and Bigram Language Models on the given Chicago hotel reviews dataset using the provided training and validation data. In the pre-processing steps for the data, the first step was to load the training data into a Jupyter Notebook as a string. After that, it removed punctuation using the re library, which is generally used for regular expressions and changed all text to lower case to maintain consistency. Preprocessing of the text was done by splitting on spaces, which gave individual words, and these were stored as tokens for further analysis. Preprocessing for the validation dataset was done using the same steps as described above. The tokenized data generated unigrams and bigrams each two consecutive words in the dataset paired as a single token.

```
text = re.sub(r'\W', ' ', text)
tokens = text.lower().split()
bigrams = [(tokens[i], tokens[i+1]) for i in range(len(tokens)-1)]
```

1.1 Unigram and Bigram probability computation

Unigram probabilities were computed by dividing the frequency of each token in the dataset by the total number of words in the training dataset. These unigram probabilities were then used to calculate bigram probabilities, where the numerator is the bigram frequency in the training corpus and the denominator is the unigram frequency of the first word. Perplexity for the training dataset was then calculated using these unigram and bigram probabilities.

1.2 Smoothing

We used three smoothing methods in this homework:
1. Laplace Smoothing
2. Add-k Smoothing, k = 0.5
3. Add-k Smoothing with k = 3

In Laplace smoothing, probabilities are computed with 1 added to the numerator, while the denominator is 'V', or the vocabulary size of the training corpus. That is, this method normalizes the probabilities by shrinking the high values for the frequent words and increasing the low values for the rare words.

In Add-k Smoothing when k=0.5, probabilities were calculated by adding 0.5 in the numerator and adding 0.5 times the vocabulary in the denominator, i.e., 0.5*V. As in Laplace smoothing, this too has a balancing factor k, which try to balance the probability distribution between the frequent and rare word.

This is precisely the same process applied for k = 3 in Add-k Smoothing with k = 3. The larger the value of k shares more of the probability with the rare tokens and shrinks those of the common ones.

1.3 Unknown word handling

For the assignment, we used two approaches for handling unknown words. Based on this, we divided the computations into three types:

1. Ignoring unknown words in the validation data.
2. Assigning a zero frequency to the unknown word tag in the training dataset.
3. Assigning a rare-word frequency to the unknown word tag in the training dataset.

In the first approach, we used n-gram models without creating any special handling for unknown words, simply ignoring them in the validation data and assigning a probability of zero. As a result, these unknown words had no impact on the perplexity metric value, It worked like how basic n-gram models function without accounting for unknown words.

In the second approach, we added a new token "unk" in the training dataset with a frequency of zero. This method is not covered in class. According to the smoothing type the “unk” token received minimal probability. Whenever an unknown word is observed in the validation set, we assigned it the probability of the "unk" token to calculate perplexity. For the bigram implementation, we adjusted the unigram training data to include the unknown examples in the bigram tokens.

In the third approach, we assigned the "unk" tag to rare words that appeared only once in the training set. First, we modified the training data by replacing any words that had a frequency of one with the "unk" tag. After that, we calculated probabilities using smoothing techniques. When we came across an unknown word in the validation set, we used the probability of the "unk" tag to determine perplexity. For the bigram version, we again adjusted the unigram training data to incorporate the unknown examples into our bigram tokens.

1.4 Implementation of Perplexity

We calculated the perplexity using the formula provided in the problem statement.

$$PP = \left(\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}, \dots, w_{i-n+1})}\right)^{1/N} = \exp \frac{1}{N} \sum_{i=1}^{N} -\log P(w_i|w_{i-1}, \dots, w_{i-n+1})$$

For the validation dataset, we computed perplexity by using the log probabilities of tokens from the training data and combining them with the word frequencies from the validation set. The result was then summed and divided by the total number of words in the validation set to calculate the perplexity.

```
sum_log_prob = 0
    for word in val_uni_count:
        if word not in train_log_prob:  # Handling unknown words
            word_log_prob = train_log_prob['unk']
        else:
            word_log_prob = train_log_prob[word]
```

```
        sum_log_prob += (-1) * word_log_prob * val_uni_count[word]
perplexity = math.exp(sum_log_prob / val_word_count)
```

For the training dataset, the perplexity was calculated similarly by using the log probabilities of tokens and their frequencies in the training data. The sum was then divided by the number of words in the training set to calculate the perplexity value.

2. Eval, Analysis & Findings

The perplexity of the training dataset was calculated as 519.79 for the unigram model and 30.21 for the bigram model. Using the three different smoothing techniques combined with the two approaches to handling unknown words, we computed the perplexities for both the unigram and bigram models as follows

Unigram Model:

	By ignoring unknown words	**By adding unknown word tag with frequency zero**	**By replacing rare words with unknown word tag**
Laplace smoothing	526.21	539.95	349.03
Add-k smoothing with k=0.5	383.25	585.84	347.33
Add-k smoothing with k=3	413.50	558.86	357.80

Bigram Model:

	By ignoring unknown words	**By adding unknown word tag with frequency zero**	**By replacing rare words with unknown word tag**
Laplace smoothing	60.17	193.17	135.37
Add-k smoothing with k=0.5	43.75	110.32	91.95
Add-k smoothing with k=3	97.76	459.74	270.69

From the analysis of the tables, we can draw the following conclusions:

1. The bigram model consistently produced better results with lower perplexity values.
2. Ignoring unknown words results in lower perplexities, but the model lacks generalization and performs poorly on unseen data with many unknown words, indicating overfitting to the training dataset

3. Hard coding the unknown word token frequencies to zero is causing the model to give high perplexities, This says that the model is underfitting on the training dataset and it cannot handle unknown words in validation set. Even though model is generalized, it will not give accurate results.
4. When rare words are replaced with an unknown word tag in the training data, the perplexity values show that the model generalizes well and handles unknown words in the validation dataset effectively. This model is likely to perform better on unseen data.
5. Among the smoothing techniques, using k = 0.5 delivers the best results in normalizing n-gram probabilities.
6. We can observe a pattern that value of k is directly proportional to perplexity values, as k increases, perplexity value also increases, this shows that there is a trade-off between model accuracy and perplexity.
7. In conclusion we got that, a generalized model can be created by normalizing word probabilities that can effectively handle unknown words.

3. Others

Unigram model:
https://colab.research.google.com/drive/1MvQqTEdWjzCRY03ZUHH3CbCzwKjrK7Yz?usp=sharing

Bigram model:
https://colab.research.google.com/drive/1ej8NA3Wh4PcZwQaUd3IygeycdXUt1sMp?usp=sharing

Libraries used:
1. Re
2. Math
3. Collections

Re library helps us to perform string operations for large datasets, We needed logarithmic, exponential and math operations for the calculation, math library helped us to do these operations, Collections library helps us to perform large operations on large dataset by using efficient data structures which helps it to organize the data such as Counter, DefaultDict, OrderedDict etc.

The team has contributed to the project equally, Vijay has worked on Preprocessing, Perplexity Calculation and report, Vikas has worked on implementing Bigram model and Report work, Manikanta has worked on Implementing Unigram and Bigram models.

The assignment reasonably difficult, The statement and challenges given were clear and easy to understand, We dedicated two hours every day from the time the assignment was given. This assignment helped us to clearly understand the relationship between various smoothing techniques and perplexity and how the way of handling the unknown words can significantly affect the model performance.