Assignment 1

Group 11 Jyothsna Reddy Cheemalapati Jyothi Prasad(JXC220075) Sai Premanvitha Kadiyala(SXK230035) Yamini Sunkari (YXS220036)

Github link: https://github.com/Premanvitha2/CS6320.git

1. Implementation steps:

Pre-Processing

We performed pre-processing on the training, validation, and test datasets to prepare the text data for analysis. This process involved tokenization, preprocessing text data, transforming raw reviews into a structured format suitable for analysis.The steps include.

Tokenization: We divided each sentence into individual tokens, where each token represents a word, and stored these tokens in a list.

Text Cleaning and Placeholder Management: During tokenization, placeholders like <NUMBER>, <WEBLINK>, and <EMAILADDRESS> replace sensitive information to prevent disruption in text analysis. Unwanted punctuation is removed streamline the text, and <START_REVIEW> and <END_REVIEW> markers define the boundaries of each review, ensuring accurate processing.

```python
import re
from collections import Counter

def clean_and_tokenize(text_data):
    text_data = text_data.lower()
    text_data = re.sub(r'\d+(\.\d+)?', '<NUMBER>', text_data)
    text_data = re.sub(r"[.,(){}[\]'\"-]", '', text_data)  # Removing unwanted punctuations
    text_data = re.sub(r'http\S+', '<WEBLINK>', text_data)
    text_data = re.sub(r'\S+@\S+', '<EMAILADDRESS>', text_data)
    token_list = text_data.split()
    token_list = ['<START_REVIEW>'] + token_list + ['<END_REVIEW>']
    return token_list

def process_reviews(file_location):
    with open(file_location, 'r') as data_file:
        reviews = data_file.readlines()

    # Apply text cleaning and tokenization to each review
    cleaned_reviews = [clean_and_tokenize(review) for review in reviews]

    return cleaned_reviews

# Function to handle tokens not in vocabulary
def replace_unknown_words(tokens_list, vocabulary):
    return [[word if word in vocabulary else "<UNKNOWN>" for word in review] for review in tokens_list]

```

2. Unigram and Bigram Models:

Below logic was used to compute unigram and bigram probabilities:

For unigrams, the probability $P(w_i)$ for a word w_i was determined by dividing the count of w_i frequency by the total number of tokens.

For bigrams, the probability $P(w_i|w_{i-1})$ of a word w_i given the preceding word w_{i-1} was calculated as the ratio of the count of the bigram (w_{i-1},w_i) to the count of w_{i-1}.

main.py 6 | Preprocessing.py | Unigram,Bigram_Compute.py

C: > Users > yasas > Desktop > JYOTHSNA > CS6320 > Unigram,Bigram_Compute.py > compute_unigrams_and_bigrams

```python
from collections import defaultdict


def compute_unigrams_and_bigrams(corpus):
    unigram_counts = defaultdict(int)
    bigram_counts = defaultdict(int)
    for review in corpus:
        for token in review:
            unigram_counts[token] += 1
    for word in list(unigram_counts.keys()):
        if unigram_counts[word] < 25:
            unigram_counts['<UNKNOWN>'] += unigram_counts[word]
            del unigram_counts[word]
    for review in corpus:
        adjusted_review = [token if unigram_counts[token] >= 25 else '<UNKNOWN>' for token in review]
        for i in range(len(adjusted_review) - 1):
            bigram = (adjusted_review[i], adjusted_review[i + 1])
            bigram_counts[bigram] += 1
    if '<UNKNOWN>' not in unigram_counts:
        unigram_counts['<UNKNOWN>'] = 0  # Initialize if the unknown word wasn't created so far
    unigram_counts = {word: count for word, count in unigram_counts.items() if count > 0}
    return unigram_counts, bigram_counts

def compute_probabilities(unigram_counts, bigram_counts):
    total_unigrams = sum(unigram_counts.values())

    # Compute unigram probabilities
    unigram_probs = {word: count / total_unigrams for word, count in unigram_counts.items()}

    # Compute bigram probabilities
    bigram_probs = {}
    for (word1, word2), count in bigram_counts.items():
        bigram_probs[(word1, word2)] = count / unigram_counts[word1]

    return unigram_probs, bigram_probs
```

Also, added a function to print unigram and bigram probabilities, which is then called in main function. This effectively displays the probabilities for few unigrams and bigrams in the training dataset.

```python
print_probabilities(unigram_counts, bigram_counts,8)
print("\n\n")
```

3. Smoothing and Unknown words:

In this assignment, we implemented smoothing techniques, including Laplace smoothing, Add-k Smoothing. Below is a detailed explanation of each method:

1. Laplace Smoothing

- Formula:

$$P_{\text{Laplace}}(\text{word}_i) = \frac{n_i + 1}{N + V}$$

Implemented Laplace smoothing by using above formula where ni is frequency of corresponding word and N and V are total words count and vocabulary count

2. Add-k Smoothing

- Formula:

$$P_{Add-k}(word_n \mid word_{n-1}) = \frac{C(word_{n-1}word_n) + k}{C(word_{n-1}) + kV}$$

Implemented Add-K smoothing using this formula where it is similar to Laplace but with different k values

Interpretation: By adding a value k rather than 1, add-k smoothing allows for more nuanced adjustments, assigning smaller probabilities to unseen n-grams and reducing the bias introduced by smoothing.

```python
#**implementing laplace and k smoothing.**
#logarithmic probability of tokens using Laplace smoothing for bigrams.
def laplace_bigram_smoothing(bigram_counts, unigram_counts, V, tokens):
    total_log_prob = 0.0

    for i in range(len(tokens)):
        if i > 0:
            bigram_prob = (bigram_counts.get((tokens[i-1], tokens[i]), 0) + 1) / (unigram_counts.get(tokens[i-1], 0) + V)
            total_log_prob += log(bigram_prob) if bigram_prob > 0 else log(1e-20)

    return total_log_prob

#logarithmic probability of tokens using K-smoothing for bigrams.
def k_smoothing_bigram(bigram_counts, unigram_counts, V, K, tokens):
    total_log_prob = 0.0

    for i in range(len(tokens)):
        if i > 0:
            num = bigram_counts.get((tokens[i-1], tokens[i]), 0) + K #calculating numerator and denominator values for given K value
            den = unigram_counts.get(tokens[i-1], 0) + (K * V)
            bigram_prob = num/den
            total_log_prob += log(bigram_prob) if bigram_prob > 0 else log(1e-20)

    return total_log_prob
```

3. Handling Unknown Words

- A special token `<UNKNOWN>` is used to handle low frequency training data. Now in validation data when unknown words are encountered we replace them with above token. This ensures that zero probabilities are avoided.

```python
# Function to handle tokens not in vocabulary
def replace_unknown_words(tokens_list, vocabulary):
    return [[word if word in vocabulary else "<UNKNOWN>" for word in review] for review in tokens_list]
```

4. Perplexity on the validation set:

In this assignment, we computed perplexity by taking the exponent of the entropy of the validation set. The formula for perplexity is:

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

main.py 6 | Unknown_words.py 5 | Perplexity.py 2

C: > Users > yasas > Desktop > JYOTHSNA > CS6320 > Perplexity.py > compute_perplexity

```python
from math import log,exp
def compute_perplexity(unigram_counts, bigram_counts, tokens,V,k=None) :
    #tokens = [tokens[0]]
    total_log_probability_sum = 0.0;
    N = 0;

    for review in tokens:
      N += len(review)
      if k is None:
            # Use Laplace smoothing
            total_review_prob = laplace_bigram_smoothing(bigram_counts, unigram_counts, V, review)
      else:
            # Use K-smoothing
            total_review_prob = k_smoothing_bigram(bigram_counts, unigram_counts, V, k, review)

      total_log_probability_sum += total_review_prob

    avg_log_probability = (-1 * total_log_probability_sum) / N
    perplexity = exp(avg_log_probability)
    print(perplexity)
    return perplexity
```

5. Results, Error Analysis, and Findings

Evaluation:

Data Preprocessing

Text was converted to lowercase, punctuation was removed, numerical values were replaced with placeholders, and tokens for start and end of reviews were added to enhancing model performance.

N-gram Models

Unigram and bigram models predicted word sequences using probabilities derived from training data.

Handling Unknown Words

Unknown words were replaced with a <UNKNOWN> token, allowing the model to manage unfamiliar terms in the validation set.

Smoothing Techniques

Laplace and Add-k smoothing assigned small probabilities to unseen word sequences, improving model reliability.

Perplexity Measurement

Perplexity measured the model's predictive ability. Both smoothed models achieved a score of 55.1051(when K=1), indicating stable performance.

Analysis:
Laplace and K=1 Consistency: Both methods yield the same perplexity of 55.1051, indicating similar handling of unseen sequences**.**
Variation with Different K-values: As KKK decreases (to 0.1 and 0.05), perplexity improves significantly, reaching a minimum of 39.4930 at K=0.05K=0.05K=0.05. However, increasing KKK to 0.01 results in a higher perplexity (42.2768), suggesting that a smaller KKK is more effective for smoothing unseen word sequences.
Findings:
The high perplexity of the unsmoothed model underscores the importance of smoothing techniques in N-gram models for handling unseen word sequences. The similar results from both smoothing methods make it difficult to determine the better option, suggesting a need for further analysis. Additionally, these outcomes indicate potential for improvement by exploring alternative smoothing methods, refining preprocessing steps, or testing more complex N-gram models to enhance performance.

6. Programming Libraries Utilized:

- **re:** For regex-based text processing.
- **collections:** Specifically, the Counter class to compute word frequencies.
- **math:** For logarithmic and exponential calculations.

7. Individual Contributions:

Prema: Pre-processing, Classification and report.
Yamini: Unigram, bigram model, Smoothing and unknown word handling.
Jyothsna: Perplexity calculation, error analysis, and report.

8.feedback and Conclusion :

This assignment was both challenging and engaging, requiring extensive brainstorming and enhancing our understanding of class concepts. While we appreciated the depth of learning, the time-consuming report-writing process overshadowed our coding efforts. We would have preferred a greater focus on coding, allowing us to refine our implementations and experiment with different approaches. Ultimately, we successfully developed unigram and bigram models for classifying spam reviews in the dataset, gaining valuable insights into building Natural Language Processing models and exploring techniques to enhance model performance through analysis.