PROBLEM 1

In [1]:
```
import pandas as pd
import nltk
from nltk.util import ngrams
from collections import defaultdict
import math

df_sst = pd.read_csv('C:/Users/mano2/Downloads/SST-2 (1)/SST-2/train.tsv', delimiter='\t')

test_data = df_sst.iloc[0:100]
validate_data = df_sst.iloc[100:199]
train_data = df_sst.iloc[200:]
n = len(train_data)

total_ones = train_data['label'].sum()
total_zero = n - total_ones

prior_zero = total_zero / n
prior_one = total_ones / n
print("Prior probability of zero:", prior_zero)
print("Prior probability of one:", prior_one)
```

```
Prior probability of zero: 0.44203189920922126
Prior probability of one: 0.5579681007907787
```

PROBLEM 2

In [2]:
```
def tokenize_pad(sentence):

    words = nltk.word_tokenize(sentence)


    tokens = ['<s>'] + words + ['</s>']

    return tokens


train_set = train_data['sentence']
```

```
tokenized_train_set = [tokenize_pad(sentence) for sentence in train_set]

vocabulary = set(word for sentence in tokenized_train_set for word in sentence)
vocabulary_size = len(vocabulary)
print("First sentence in the training set:",tokenized_train_set[:1])
#print(tokenized_training_set[:1])
print("\nVocabulary size:", vocabulary_size)
```

```
First sentence in the training set: [['<s>', 'told', 'in', 'scattered', 'fashion', '</s>']]

Vocabulary size: 14801
```

PROBLEM 3

In [3]:
```
def bigram_frequencies(tokenized_sequences):
    bigram_counts = defaultdict(lambda: defaultdict(int))

    for sequence in tokenized_sequences:
        bigrams = list(ngrams(sequence, 2, pad_left=True, pad_right=True, left_pad_symbol='<s>', right_pad_symbol='</s>'))

        for bigram in bigrams:
            word1, word2 = bigram
            bigram_counts[word1][word2] += 1

    return bigram_counts


tokenized_sequences = tokenized_train_set

bigram_counts = bigram_frequencies(tokenized_sequences)
print("Count of bigrams '<s>','the':",bigram_counts["<s>"]["the"])
```

```
Count of bigrams '<s>','the': 4450
```

PROBLEM 4

In [4]:
```
def negative_log_probability(wm, wm_1, bigram_counts, alpha, vocabulary_size):

    bigram_count = bigram_counts.get(wm_1, {}).get(wm, 0)
    #print(bigram_count)
```

```
    total_count_wm_1 = sum(bigram_counts.get(wm_1, {}).values())
    #print(total_count_wm_1)

    probability = (bigram_count + alpha) / (total_count_wm_1 + alpha * vocabulary_size)
    #print(probability)


    if probability > 0:
        neg_log_probability = -math.log2(probability)
    else:
        neg_log_probability = float('inf')

    return neg_log_probability


current_word = "award"
previous_word = "academy"
neg_log_prob = negative_log_probability(current_word, previous_word, bigram_counts, 0.001, vocabulary_size)
print(f"Negative Log-Probability of '{current_word}' given '{previous_word}'(alpha = 0.001): {neg_log_prob:.2f}")
neg_log_prob = negative_log_probability(current_word, previous_word, bigram_counts, 0.5, vocabulary_size)
print(f"Negative Log-Probability of '{current_word}' given '{previous_word}'(alpha = 0.5): {neg_log_prob:.2f}")
```

```
Negative Log-Probability of 'award' given 'academy'(alpha = 0.001): 1.48
Negative Log-Probability of 'award' given 'academy'(alpha = 0.5): 8.90
```

PROBLEM 5

In [5]:

```
def sentence_log_probability(sentence, bigram_counts, alpha, vocabulary_size):

    tokens = tokenize_pad(sentence)


    log_probability = 0.0


    for i in range(1, len(tokens)):

        current_word = tokens[i]
        previous_word = tokens[i - 1]
```

```
        bigram_count = bigram_counts[previous_word][current_word]

        unigram_count = sum(bigram_counts[previous_word].values())

        smoothed_probability = (bigram_count + alpha) / (unigram_count + alpha * vocabulary_size)

        log_probability += math.log2(smoothed_probability)

    return log_probability


sentence1 = "this was a really great movie but it was a little too long."
sentence2 = "long too little a was it but movie great really a was this."
alpha = 1

log_prob1 = sentence_log_probability(sentence1, bigram_counts, alpha, vocabulary_size)
log_prob2 = sentence_log_probability(sentence2, bigram_counts, alpha, vocabulary_size)

print("Log Probability for Sentence 1:", log_prob1)
print("Log Probability for Sentence 2:", log_prob2)
```

```
Log Probability for Sentence 1: -130.78286697935766
Log Probability for Sentence 2: -183.08363959315753
```

PROBLEM 6

In [6]:
```
alphas = [0.001, 0.01, 0.1]

log_likelihood_estimates = []


for alpha in alphas:

    log_likelihood = 0.0

    for sentence in validate_data['sentence']:

        log_prob = sentence_log_probability(sentence, bigram_counts, alpha, vocabulary_size)
        log_likelihood += log_prob

    log_likelihood_estimates.append(log_likelihood)
```

```
for i, alpha in enumerate(alphas):
    print(f"Log-Likelihood Estimate for alpha {alpha} = {log_likelihood_estimates[i]:.4f}")

best_alpha_index = log_likelihood_estimates.index(max(log_likelihood_estimates))
best_alpha = alphas[best_alpha_index]
print(f"The best alpha is: {best_alpha}")

selected_alpha = best_alpha
```

```
Log-Likelihood Estimate for alpha 0.001 = -6901.5799
Log-Likelihood Estimate for alpha 0.01 = -7664.3090
Log-Likelihood Estimate for alpha 0.1 = -9191.7177
The best alpha is: 0.001
```

PROBLEM 7

In [7]:

```
positive_data = train_data[train_data['label'] == 1]
#print(len(positive_data))
negative_data = train_data[train_data['label'] == 0]
#print(len(negative_data))

positive_tokenized_sentences = [tokenize_pad(sentence) for sentence in positive_data['sentence']]
negative_tokenized_sentences = [tokenize_pad(sentence) for sentence in negative_data['sentence']]

positive_vocabulary = set(word for sentence in positive_tokenized_sentences for word in sentence)
negative_vocabulary = set(word for sentence in negative_tokenized_sentences for word in sentence)

positive_vocabulary_size = len(positive_vocabulary)
negative_vocabulary_size = len(negative_vocabulary)

positive_bigram_counts = bigram_frequencies(positive_tokenized_sentences)
negative_bigram_counts = bigram_frequencies(negative_tokenized_sentences)
alpha = selected_alpha

positive_scores_method5 = []
negative_scores_method5 = []
positive_scores_bayes = []
negative_scores_bayes = []

prior_positive = prior_one
```

```
prior_negative = prior_zero

for sentence in test_data['sentence']:

    log_prob = sentence_log_probability(sentence, bigram_counts, alpha, vocabulary_size)
    positive_score_method5 = prior_positive * math.exp(log_prob)
    negative_score_method5 = prior_negative * math.exp(log_prob)

    log_prob_positive = sentence_log_probability(sentence, positive_bigram_counts, alpha, positive_vocabulary_size)
    log_prob_negative = sentence_log_probability(sentence, negative_bigram_counts, alpha, negative_vocabulary_size)

    positive_score_bayes = prior_positive * math.exp(log_prob_positive)
    negative_score_bayes = prior_negative * math.exp(log_prob_negative)

    positive_scores_method5.append(positive_score_method5)
    negative_scores_method5.append(negative_score_method5)
    positive_scores_bayes.append(positive_score_bayes)
    negative_scores_bayes.append(negative_score_bayes)

predicted_sentiment_labels_method5 = []
predicted_sentiment_labels_bayes = []

threshold = 1.0

for i in range(len(test_data)):

    if positive_scores_method5[i] > negative_scores_method5[i]:
        predicted_sentiment_labels_method5.append('positive')
    else:
        predicted_sentiment_labels_method5.append('negative')


    if positive_scores_bayes[i] > negative_scores_bayes[i]:
        predicted_sentiment_labels_bayes.append('positive')
    else:
        predicted_sentiment_labels_bayes.append('negative')

from collections import Counter

class_distribution_method5 = Counter(predicted_sentiment_labels_method5)
```

```
class_distribution_bayes = Counter(predicted_sentiment_labels_bayes)

true_sentiment_labels = test_data['label'].apply(lambda label: 'positive' if label == 1 else 'negative')

correct_predictions_method5 = 0
correct_predictions_bayes = 0

for i in range(len(test_data)):
    if predicted_sentiment_labels_method5[i] == true_sentiment_labels[i]:
        correct_predictions_method5 += 1
    if predicted_sentiment_labels_bayes[i] == true_sentiment_labels[i]:
        correct_predictions_bayes += 1

accuracy_method5 = correct_predictions_method5 / len(test_data)
accuracy_bayes = correct_predictions_bayes / len(test_data)

print("Accuracy for function:", accuracy_method5)
print("Accuracy for Bayes Rule:", accuracy_bayes)
```

```
Accuracy for function: 0.49
Accuracy for Bayes Rule: 0.92
```