In [1]:
```
import nltk
import pandas as pd
from nltk.tokenize import word_tokenize
from csv import QUOTE_NONE
import pandas as pd
from sklearn.model_selection import train_test_split
from collections import defaultdict
from sklearn.metrics import accuracy_score, classification_report
```

Problem 1

In [2]:
```
DataSet_SST = "D:/Masters/NLP/Assignments/Assignment1/SST-2/SST-2/train.tsv"
df_sst = pd.read_csv(DataSet_SST, sep='\t' )
df_sst.head()
```

Out[2]:

	sentence	label
0	hide new secretions from the parental units	0
1	contains no wit , only labored gags	0
2	that loves its characters and communicates som...	1
3	remains utterly satisfied to remain the same t...	0
4	on the worst revenge-of-the-nerds clichés the ...	0

In [3]:

```
c1,c2 = df_sst["sentence"],df_sst["label"]
random_state=19
# first split to train and (Test and Val)
test_val_size=200
x_train, x_TestVal, y_train, y_TestVal = train_test_split(c1, c2, test_size=test_val_size, random_state=randon
#now split Remaining from prvious step to validation and test set
test_size=100
x_val, x_test, y_val, y_test = train_test_split(x_TestVal, y_TestVal, test_size=test_size, random_state=randon
print("Train set size=",len(x_train))
print("Validation set size=",len(x_val))
print("Test set size=",len(x_test))
```

```
Train set size= 67149
Validation set size= 100
Test set size= 100
```

In [4]:

```
# Calculate the prior probability of each class
tot_count = len(x_train)
indv_class_size = y_train.value_counts()
# print(class_counts)
each_prior_probability = indv_class_size / tot_count
c0_label=each_prior_probability[0]
c1_label=each_prior_probability[1]
print("Prior Probabilities ['label']:")
print("Class 0 (Negative): ", c0_label)
print("Class 1 (Positive): ", c1_label)
```

```
Prior Probabilities ['label']:
Class 0 (Negative):  0.442165929500067
Class 1 (Positive):  0.5578340704999329
```

Problem 2

In [5]:

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
def tokenize_padding(sentence):
    tokens = word_tokenize(sentence)
    tokens_padded = ['<s>'] + tokens + ['</s>']
    return tokens_padded
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\seshu\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [6]:

```
tokenized_padded_seqs = x_train.apply(tokenize_padding)
print("Tokenized sequence padded by start and end symbols First sentence:")
print(tokenized_padded_seqs[1])
```

```
Tokenized sequence padded by start and end symbols First sentence:
['<s>', 'contains', 'no', 'wit', ',', 'only', 'labored', 'gags', '</s>']
```

In [7]:

```
tot_tokens = set()
for tokens in tokenized_padded_seqs:
    all_token=tot_tokens.update(tokens)
tot_tokens_size = len(tot_tokens)
print("Vocabulary size of the train set (included start and end symbols):", tot_tokens_size)
```

```
Vocabulary size of the train set (included start and end symbols): 14806
```

Problem 3

In [8]:
```
def count_bigram_freq(token_seq):
    bigram_data = {}
    flag=1
    for seqs in token_seq:
        i = 0
        while i < len(seqs) - 1:
            word_i, word_j = seqs[i], seqs[i + 1]
            if word_i not in bigram_data:
                bigram_data[word_i] = {}
                if word_j not in bigram_data:
                    bigram_data[word_i][word_j] = 1
                else:
                    bigram_data[word_i][word_j] = bigram_data.get(word_i, {}).get(word_j, 0) + 1
            else:
                if word_j not in bigram_data[word_i]:
                    bigram_data[word_i][word_j] = 1
                else:
                    bigram_data[word_i][word_j] = bigram_data.get(word_i, {}).get(word_j, 0) + 1
            i += 1
    return bigram_data
```

In [9]:
```
# Now applying the func to tokenized sequences
bigram_counts = count_bigram_freq(tokenized_padded_seqs)
start_the_count = bigram_counts.get("<s>", {}).get("the", 0)
# Count total no.of occurrences ("<s>", "the")
print("Count of sentences starting with '<s>', 'the':", start_the_count)
```

```
Count of sentences starting with '<s>', 'the': 4453
```

Problem 4

In [10]:

```
import random
import math
def smooth_func(Wm_1, Wm, tot_BG_counts, alpha_val, tot_vocab_size):
    if Wm_1 not in tot_BG_counts:
        tot_BG_counts[Wm_1] = {}
    z = tot_BG_counts[Wm_1].get(Wm, 0) + alpha_val
    b_sum_val = 0
    for t in set(tot_BG_counts.get(Wm_1, {}).keys()):
        count_b = tot_BG_counts[Wm_1].get(t, 0)
        b_sum_val += count_b
    b_sum_val += (tot_vocab_size * alpha_val)
    smooth_sum = -math.log(z / b_sum_val)
    return smooth_sum
```

In [11]:

```
alpha_1 = 0.001
alpha_2 = 0.5
vocab_size = tot_tokens_size
```

In [12]:

```
log_probability_alpha_1 = smooth_func("academy", "award", bigram_counts, alpha_1, vocab_size)
log_probability_alpha_2 = smooth_func("academy", "award", bigram_counts, alpha_2, vocab_size)
print(f"The Negative log-prob of 'academy' followed by 'award' (alpha={alpha_1}): {log_probability_alpha_1}")
print(f"The Negative log-prob of 'academy' followed by 'award' (alpha={alpha_2}): {log_probability_alpha_2}")
```

```
The Negative log-prob of 'academy' followed by 'award' (alpha=0.001): 1.0249230043528377
The Negative log-prob of 'academy' followed by 'award' (alpha=0.5): 6.172441113786604
```

Problem 5

In [13]:

```
import math

def sentence_logProb(sentence, bg_counts, alpha, vocab_len):
    sentence_tok = sentence.split()
    log_probabilities = 0.0
    i = 1
    while i < len(sentence_tok):
        prev_word_tk = sentence_tok[i - 1]
        curr_word_tk = sentence_tok[i]
        if prev_word_tk in bg_counts:
            curr_bg_count = bg_counts[prev_word_tk].get(curr_word_tk, 0) + alpha
        else:
            curr_bg_count = alpha

        tot_sum_value = 0
        wds_list = list(bg_counts.get(prev_word_tk, {}))
        j = 0
        while j < len(wds_list):
            wds = wds_list[j]
            tot_sum_value += bg_counts[prev_word_tk][wds]
            j += 1
        tot_sum_value += (vocab_len * alpha)
        log_probabilities += math.log(curr_bg_count / tot_sum_value)
        i += 1
    return log_probabilities
```

In [14]:

```
# Defining 2  alpha values and vocabulary size
alpha_1 = 0.001
alpha_2 = 1
vocab_size = tot_tokens_size

# Define sentences
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."

# Calculate log probabilities for each sentence and alpha value
log_probability_sentence1_alpha_1 = sentence_logProb(sentence1, bigram_counts, alpha_1, vocab_size)
log_probability_sentence1_alpha_2 = sentence_logProb(sentence1, bigram_counts, alpha_2, vocab_size)
log_probability_sentence2_alpha_1 = sentence_logProb(sentence2, bigram_counts, alpha_1, vocab_size)
log_probability_sentence2_alpha_2 = sentence_logProb(sentence2, bigram_counts, alpha_2, vocab_size)

print("The Log Probability for Sentence 1 with Alpha 0.001:", log_probability_sentence1_alpha_1)
print("The Log Probability for Sentence 1 with Alpha 1:", log_probability_sentence1_alpha_2)
print("The Log Probability for Sentence 2 with Alpha 0.001:", log_probability_sentence2_alpha_1)
print("The Log Probability for Sentence 2 with Alpha 1:", log_probability_sentence2_alpha_2)
```

```
The Log Probability for Sentence 1 with Alpha 0.001: -71.3661601070727
The Log Probability for Sentence 1 with Alpha 1: -82.16503456775979
The Log Probability for Sentence 2 with Alpha 0.001: -145.59741843575108
The Log Probability for Sentence 2 with Alpha 1: -110.4482589115054
```

Problem 6

In [15]:

```python
val_set = list(x_val)
alpha_values = [0.001, 0.01, 0.1]
alpha_logs = []
i = 0
while i < len(alpha_values):
    alpha_val = alpha_values[i]
    log_probs_sum = 0
    j = 0
    while j < len(val_set):
        log_probs_sum += sentence_logProb(val_set[j], bigram_counts, alpha_val, tot_tokens_size)
        j += 1
    alpha_logs.append(log_probs_sum)
    i += 1
print("Val DataSet:Log probabilities alpha 0.001 is=",alpha_logs[0])
print("Val DataSet:Log probabilities alpha 0.01 is=",alpha_logs[1])
print("Val DataSetLog:Log probabilities alpha 0.1 is=",alpha_logs[2])
```

```
Val DataSet:Log probabilities alpha 0.001 is= -3624.821244134856
Val DataSet:Log probabilities alpha 0.01 is= -4103.408166234085
Val DataSetLog:Log probabilities alpha 0.1 is= -5055.9900427639
```

In [16]:

```python
best_alpha = 0.001
print("The best alpha value gives better result compared to others : ",best_alpha)
```

```
The best alpha value gives better result compared to others :  0.001
```

Problem 7

In [17]:

```
pos_sent,neg_sent, pos_tks, neg_tks = [],[],[],[]
pos_sent = [i for i, label in zip(x_train, y_train) if label == 1]
neg_sent = [i for i, label in zip(x_train, y_train) if label == 0]
pos_len , neg_len= len(pos_sent), len(neg_sent)
print("Length of positive_sentences list:", pos_len)
print("Length of negative_sentences list:", neg_len)
# Tokenize positive and negative sentences
pos_tks = [tokenize_padding(i) for i in pos_sent]
neg_tks = [tokenize_padding(i) for i in neg_sent]
```

```
Length of positive_sentences list: 37458
Length of negative_sentences list: 29691
```

In [18]:

```
pos_tokens,neg_tokens = [],[]
pos_tokens = [i for tks in pos_tks for i in tks]
neg_tokens = [i for tks in neg_tks for i in tks]
# Calculate the vocabulary size for positive and negative tokens
pos_tk_size,neg_tk_size = len(set(pos_tokens)), len(set(neg_tokens))
#print(len(pos_tks))
print("The Positive tokens vocabulary size:", pos_tk_size)
print("The Negative tokens vocabulary size:", neg_tk_size)
```

```
The Positive tokens vocabulary size: 11523
The Negative tokens vocabulary size: 11242
```

In [19]:

```
pos_bgm_cts,neg_bgm_cts = count_bigram_freq(pos_tks),count_bigram_freq(neg_tks)
pos_ct,neg_ct = len(pos_sent),len(neg_sent)
total_count = pos_ct+neg_ct
pos_prior = pos_ct/total_count
neg_prior = neg_ct/total_count
print("prior probability of one class=",pos_prior)
print("prior probability of zero class=",neg_prior)
```

```
prior probability of one class= 0.5578340704999329
prior probability of zero class= 0.442165929500067
```

In [20]:

```
print("prior probability of one class=",c1_label)
print("prior probability of zero class=",c0_label)
```

```
prior probability of one class= 0.5578340704999329
prior probability of zero class= 0.442165929500067
```

In [21]:

```
best_alpha = 0.001
preds = []
for i in x_test:
    pos_sent_score = sentence_logProb(i, pos_bgm_cts, best_alpha, pos_tk_size) + math.log(c1_label)
    neg_sent_score = sentence_logProb(i, neg_bgm_cts, best_alpha, neg_tk_size) + math.log(c0_label)
    preds.append(1 if pos_sent_score > neg_sent_score else 0)
```

In [22]:

```
true_labels = list(y_test)
acc = accuracy_score(true_labels, preds)
print("Accuracy for best alpha:", acc)
```

```
Accuracy for best alpha: 0.91
```

In [23]:

```
report = classification_report(y_test, preds)
print("Reults:\n", report)
```

```
Reults:
               precision    recall  f1-score   support

           0       0.88      0.93      0.91        46
           1       0.94      0.89      0.91        54

    accuracy                           0.91       100
   macro avg       0.91      0.91      0.91       100
weighted avg       0.91      0.91      0.91       100
```

In [ ]:

In [ ]:

In [ ]: