Importing Libraries

In [1]:
```python
import os
import pandas as pd
import math
from collections import defaultdict
```

Problem 1- Reading the data

In [2]:
```python
data_dir = "SST-2"
file_name = "train.tsv"

file_path = os.path.join(data_dir, file_name)

df_sst = pd.read_csv(file_path, delimiter="\t")
df_sst.head(3)
```

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

In [3]:
```python
validationSize = 100
testSize = 100

df_sst = df_sst.sample(frac=1, random_state=42).reset_index(drop=True)

val = df_sst[:validationSize]
test = df_sst[validationSize:validationSize + testSize]
train = df_sst[validationSize + testSize:]

print("Validation dataset size:", len(val))
print("Test dataset size:", len(test))
print("Training dataset size:", len(train))
```

```
Validation dataset size: 100
Test dataset size: 100
Training dataset size: 67149
```

In [4]:
```python
positiveClassCount = (train['label'] == 1).sum()
negativeClassCount = (train['label'] == 0).sum()

totalLength = len(train)

priorPositiveProbability = (positiveClassCount / totalLength) * 100
priorNegativeProbability = (negativeClassCount / totalLength) * 100

print("Positive Probability: {:.2f}%".format(priorPositiveProbability))
print("Negative Probability: {:.2f}%".format(priorNegativeProbability))
```

```
Positive Probability: 55.77%
Negative Probability: 44.23%
```

Problem 2- Tokenizing data

In [5]:
```python
def tokenize(df, column):
    tokenizedSequences = df[column].apply(lambda sentence: ['<s>'] + sentence.split() + ['</s>'])
    return tokenizedSequences

tokenizedSequences = tokenize(train, 'sentence')
print(tokenizedSequences.head())
```

```
200    [<s>, more, measured, or, polished, production...
201    [<s>, a, child, 's, interest, and, an, adult, ...
202    [<s>, delivered, dialogue, and, a, heroine, wh...
203    [<s>, as, inept, as, big-screen, remakes, of, ...
204    [<s>, ,, crocodile, hunter, has, the, hurried,...
Name: sentence, dtype: object
```

In [6]:
```python
vocabulary = set()

for tokenizedSequence in train['sentence']:
    tokens = tokenizedSequence.split()
    vocabulary.update(tokens)

vocabulary.add('<s>')
vocabulary.add('</s>')

vocabularySize = len(vocabulary)

#print(vocabulary)
print("Vocabulary size :", vocabularySize)
```

```
Vocabulary size : 14817
```

Problem 3- Bigram Counts

In [7]:
```python
def countBigramFrequencies(tokenizedSequences):
    bigramCounts = defaultdict(lambda: defaultdict(int))

    for sequence in tokenizedSequences:
        for i in range(len(sequence) - 1):
            wi = sequence[i]
            wj = sequence[i + 1]
            bigramCounts[wi][wj] += 1
    return bigramCounts

bigramCounts = countBigramFrequencies(tokenizedSequences)
frequency = bigramCounts['<s>']['the']

print("Frequency of the bigram ('<s>', 'the') in the training set:", frequency)
```

```
Frequency of the bigram ('<s>', 'the') in the training set: 4451
```

Problem 4- Smoothing

In [8]:
```python
def smoothingLogProbability(wm, wm1, bigramCounts, alpha, vocabularySize):
    countWmWm1 = bigramCounts.get(wm1, {}).get(wm, 0) + alpha
    countWm1 = sum(bigramCounts.get(wm1, {}).values()) + (alpha * vocabularySize)
    logProb = math.log(countWmWm1 / countWm1)
    return logProb

wordWm1 = "academy"
wordWm = "award"

alpha1 = 0.001
alpha2 = 0.5

logProb1 = smoothingLogProbability(wordWm, wordWm1, bigramCounts, alpha1, vocabularySize)
logProb2 = smoothingLogProbability(wordWm, wordWm1, bigramCounts, alpha2, vocabularySize)

print(f"Log Probability with alpha={alpha1}: {logProb1}")
print(f"Log Probability with alpha={alpha2}: {logProb2}")
```

```
Log Probability with alpha=0.001: -1.0251860898691059
Log Probability with alpha=0.5: -6.173181082203538
```

Problem 5- Sentence Log-Probability

In [9]:
```python
def logProbability(sentence, bigramCounts, alpha, vocabularySize):
    sentence_tokens = sentence.split()
    logProb = 0.0

    for i in range(1, len(sentence_tokens)):
        wm1 = sentence_tokens[i - 1]
        wm = sentence_tokens[i]

        logProb += math.log(
            (bigramCounts.get(wm1, {}).get(wm, 0) + alpha) /
            (sum(bigramCounts.get(wm1, {}).values()) + (alpha * vocabularySize))
        )

    return logProb

alpha = 0.001
vocabularySize = len(vocabulary)

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."

logProb1 = logProbability(sentence1, bigramCounts, alpha, vocabularySize)
logProb2 = logProbability(sentence2, bigramCounts, alpha, vocabularySize)

print(f"Log Probability of Sentence 1: {logProb1}")
print(f"Log Probability of Sentence 2: {logProb2}")
```

```
Log Probability of Sentence 1: -71.27052642123148
Log Probability of Sentence 2: -145.60202109372278
```

Problem 6- Tuning Alpha

In [10]:
```python
alphaValues = [0.001, 0.01, 0.1]

logLikelihoods = {}

for alpha in alphaValues:
    logLikelihood = 0.0
    for sentence in val['sentence']:
        logLikelihood += logProbability(sentence, bigramCounts, alpha, vocabularySize)
    logLikelihoods[alpha] = logLikelihood

bestAlpha = max(logLikelihoods, key = logLikelihoods.get)

for alpha, logLikelihood in logLikelihoods.items():
    print(f"Alpha = {alpha}: Log-Likelihood = {logLikelihood}")

print(f"Selected Alpha: {bestAlpha}")

selectedAlpha = bestAlpha
```

```
Alpha = 0.001: Log-Likelihood = -4060.954140773797
Alpha = 0.01: Log-Likelihood = -4564.418451383303
Alpha = 0.1: Log-Likelihood = -5571.937644465253
Selected Alpha: 0.001
```

Problem 7- Applying Language Models

In [11]:
```python
def tokenize_and_pad_sentence(sentence):
    tokens = ['<s>'] + sentence.split() + ['</s>']
    return tokens

priorPositiveProbability = (train['label'] == 1).mean()
priorNegativeProbability = (train['label'] == 0).mean()

positive_tokenized = train[train['label'] == 1]['sentence'].apply(tokenize_and_pad_sentence)
negative_tokenized = train[train['label'] == 0]['sentence'].apply(tokenize_and_pad_sentence)

positive_bigramCounts = countBigramFrequencies(positive_tokenized)
negative_bigramCounts = countBigramFrequencies(negative_tokenized)

selectedAlpha = bestAlpha

def sentiment_score(sentence, bigramCounts, alpha, vocabularySize):
    return logProbability(sentence, bigramCounts, alpha, vocabularySize)

def classify_sentiment(sentence):
    positive_score = sentiment_score(sentence, positive_bigramCounts, selectedAlpha, vocabularySize)
    negative_score = sentiment_score(sentence, negative_bigramCounts, selectedAlpha, vocabularySize)
    if positive_score > negative_score:
        return 1
    else:
        return 0

test['predicted_label'] = test['sentence'].apply(classify_sentiment)
predicted_distribution = test['predicted_label'].value_counts()

positive_predictions = test[test['label'] == 1]
negative_predictions = test[test['label'] == 0]

correct_predictions = (test['predicted_label'] == test['label']).sum()
positive_correct = (positive_predictions['predicted_label'] == positive_predictions['label']).sum()
negative_correct = (negative_predictions['predicted_label'] == negative_predictions['label']).sum()

total_sentences = len(test)

accuracy = correct_predictions / total_sentences
positive_accuracy = positive_correct / len(positive_predictions)
negative_accuracy = negative_correct / len(negative_predictions)

print("Class Distribution of Predicted Labels:")
print(predicted_distribution)
print("\nAccuracy of the Experiment: {:.2f}%".format(accuracy * 100))
print("Accuracy of Positive Predictions: {:.2f}%".format(positive_accuracy * 100))
print("Accuracy of Negative Predictions: {:.2f}%".format(negative_accuracy * 100))
```

```
Class Distribution of Predicted Labels:
1    60
0    40
Name: predicted_label, dtype: int64

Accuracy of the Experiment: 88.00%
Accuracy of Positive Predictions: 86.36%
Accuracy of Negative Predictions: 91.18%
```

```
/var/folders/qz/8_cb5kss6j909hn7j1mwr93c0000gn/T/ipykernel_26106/80856143.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['predicted_label'] = test['sentence'].apply(classify_sentiment)
```

Problem 8- Markov Assumption

Where in this homework did you apply the Markov assumption? Imagine you applied the 2nd-order Markov assumption, using trigrams. Do you think your accuracy results would increase or decrease? Why? Or, if you are not sure, give a benefit or drawback of using trigrams for this experiment. (Note: You do not need to rerun this experiment with trigrams to answer this question.)

Answer: I applied the Markov Assumption when calculating the log probability of a sentence using bigrams. Each word's probability is conditioned on the previous word, which is a first-order Markov assumption. If we were to apply a 2nd-order Markov assumption using trigrams, it would mean that each word's probability would be conditioned on the previous two words. This approach would result in a more complex model. The accuracy of the model could potentially increase or decrease. One thing that would determine what happens to the accuracy would be the size of the dataset. If the dataset is small, the model might overfit which would result in bad accuracy, but if the dataset is larger, then the model provide more context and gain more information about the relationships between words which could lead to an increase in accuracy.