```python
# importing modules
import pandas as pd
import spacy
from collections import defaultdict
import math
```

PROBLEM 1 – Reading the data

```python
# Step 1: Reading in the data from "train.tsv" using pandas
data = pd.read_csv("train.tsv", sep='\t')
data.head()
```

```
                                            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
```

```python
# Step 2: Spliting  the dataset into train, validation, and test sets
# Validation set 100 rows
validation_set = data.sample(n=100, random_state=42)
validation_dataset=pd.DataFrame(validation_set)
validation_dataset.to_csv('validation_dataset.csv',index=False)
data = data.drop(validation_set.index)

# Testing  set 100 rows
test_set = data.sample(n=100, random_state=42)
test_dataset=pd.DataFrame(test_set)
test_dataset.to_csv("test_dataset.csv",index=False)
data = data.drop(test_set.index)

# Training set is equal to the remaining rows
training_set = data
train_dataset=pd.DataFrame(training_set)
train_dataset.to_csv("train_dataset.csv",index=False)

print("****** First 5 rows of the training dataset are ")
train_dataset.head(5)
```

```
****** First 5 rows of the training dataset are

                                            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
```

```python
print("****** First 5 rows of the testing  dataset are \n")
test_dataset.head(5)
```

```
****** First 5 rows of the testing  dataset are
```

```
                                                sentence  label
38900                                          burnt out      0
44118  it could be , by its art and heart , a necessa...      1
5530                                               dying      0
39031  , an interesting and at times captivating take...      1
24722                                  clumsy and rushed      0
```

```python
print("****** First 5 rows of the validation   dataset are \n")
validation_dataset.head(5)
```

```
****** First 5 rows of the validation   dataset are
```

```
                                                sentence  label
49752                                      crisp framing      1
24709                                        dislocation      0
34945                         of the problems with the film      0
28707  's ) a clever thriller with enough unexpected ...      1
3013                                   in perfect balance      1
```

```python
# Step 3: Calculating  the prior probability of each class in the
training set
positive_count = train_dataset['label'].sum()
negative_count = len(train_dataset) - positive_count

prior_probability_positive = positive_count / len(train_dataset)
prior_probability_negative = negative_count / len(train_dataset)


# Step 4: printing  the results

print(f"Prior Probability of Positive Class in Training Set  is
```

```
{prior_probability_positive.round(4)} or
{prior_probability_positive.round(4)*100}")
print(f"Prior Probability of Negative Class in Training Set  is
{prior_probability_negative.round(4)} or
{prior_probability_negative.round(4)*100}")
```

```
Prior Probability of Positive Class in Training Set  is  0.5579 or
55.78999999999999
Prior Probability of Negative Class in Training Set  is 0.4421 or
44.21
```

PROBLEM 2 – Tokenizing data

```
# Loading  the spaCy model
nlp = spacy.load("en_core_web_sm")

# A function for tokenizing
def tokenizer(sentence):
    # Tokenizing  the sentence using spaCy
    tokens = [tok.text for tok in nlp(sentence)]

    # Adding  start and end symbols
    tokens = ['<s>'] + tokens + ['</s>']

    return tokens



# Applying the tokenizing  function to all sentences in the training
set
tokenized_sentences = [tokenizer(sentence) for sentence in
train_dataset['sentence']]

# Displaying  the tokenization of the first  sentence
for i in range(1):
    print(f"Bellow is a tockenized  of sentence")
    print(tokenized_sentences[i])
```

```
Bellow is a tockenized  of sentence
['<s>', 'hide', 'new', 'secretions', 'from', 'the', 'parental',
'units', '</s>']
```

```
# Collecting  all unique tokens from the training set
unique_tokens = set()

for sentence in train_dataset['sentence']:
    tokens = tokenizer(sentence)
    unique_tokens.update(tokens)

# Vocabulary size, including start and end symbols
vocabulary_size = len(unique_tokens)

print(f"Vocabulary size including start and end symbols is
{vocabulary_size}")
```

```
Vocabulary size including start and end symbols is 13882
```

PROBLEM 3 – Bigram counts

```
def count_bigrams(tokenized_sequences):
    bigram_counts = defaultdict(lambda: defaultdict(int))

    for sequence in tokenized_sequences:
        for i in range(len(sequence) - 1):
            wi, wj = sequence[i], sequence[i + 1]
            bigram_counts[wi][wj] += 1
```

```
    return bigram_counts


# Applying  the function to the tokenized sentences from problem 2
bigram_counts = count_bigrams(tokenized_sentences)

# To find the count of "<s>", "the"
start_the_count = bigram_counts["<s>"]["the"]

# Displaying  the count of "<s>", "the"
print("Count *****\n")
print(f'Count of "<s>", "the" is  {start_the_count}')
```

```
Count *****

Count of "<s>", "the" is  4426
```

PROBLEM 4 – Smoothing

```
def smoothing_function(wm, wm_1, bigram_counts, alpha,
vocabulary_size):

    # Calculating  the count of the bigram (wm_1, wm)
    bigram_count = bigram_counts.get(wm_1, {}).get(wm, 0)

    # Calculating  the total count of unigrams following wm_1

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

    # Applying  Laplace (add-one) smoothing to calculate the
probability

    prob = (bigram_count + alpha) / (total_count_wm_1 + alpha *
vocabulary_size)

    # Calculating  the negative log-probability
    log_prob = -math.log(prob)
```

```
    return log_prob



# Calculating  the log probability for "academy" followed by "award"
with alpha=0.001

log_prob_alpha_0_001 = smoothing_function("academy","award",
bigram_counts, vocabulary_size, 0.001)

# Calculating  the log probability for "academy" followed by "award"
with alpha=0.5
log_prob_alpha_0_5 = smoothing_function("academy","award",
bigram_counts, vocabulary_size,0.5)




# printing  the results
print(f'Log Probability "academy" , "award" with alpha=0.001 is
{log_prob_alpha_0_001}\n')
print(f'Log Probability "academy" , "award" with alpha=0.5 is
{log_prob_alpha_0_5} \n')



Log Probability "academy" , "award" with alpha=0.001 is -
5.5331628899972465

Log Probability "academy" , "award" with alpha=0.5 is  -
0.6872576282345477
```

PROBLEM 5 – Sentence log-probability

```
def sentence_log_probability(sentence, bigram_counts, alpha,
vocabulary_size):
    # Tokenizing  the sentence
```

```
    tokens = sentence.split()

    # Initializing  the log probability
    log_prob = 0.0

    # Calculating  the log probability for each bigram in the sentence

    for i in range(1, len(tokens)):
        wi, wm_1 = tokens[i], tokens[i - 1]
        log_prob += smoothing_function(wi, wm_1, bigram_counts, alpha,
vocabulary_size)

    return log_prob

# Using the bellow  sentences as examples
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."


# Calculating  log probability for each  sentence
log_prob1 = sentence_log_probability(sentence1, bigram_counts,
vocabulary_size,0.001)
log_prob2 = sentence_log_probability(sentence2, bigram_counts,
vocabulary_size,0.001)





# printing  the results
print(f'Log Probability for Sentence 1 is  {log_prob1} \n')
print(f'Log Probability for Sentence 2 is {log_prob2}\n')
```

```
Log Probability for Sentence 1 is  -21.656772595585622

Log Probability for Sentence 2 is -23.47224806279462
```

PROBLEM 6 – Tuning Alpha

```
# List of alpha values
alpha_values = [0.001, 0.01, 0.1]

# Initializing  variables to store the best alpha and best log
likelihood
best_alpha = None
best_log_likelihood = float('-inf')  # Initializing  with negative
infinity

# Iterating  over the alpha values and calculating  log likelihood for
each

for alpha in alpha_values:
    total_log_likelihood = 0.0  # Initializing  the total log
likelihood for this alpha

    # loopig through the dataset sentenses
    for sentence in validation_dataset['sentence']:
        total_log_likelihood += sentence_log_probability(sentence,
bigram_counts, alpha, vocabulary_size)

    # Checking if this alpha has a better log likelihood

    if total_log_likelihood > best_log_likelihood:
        best_log_likelihood = total_log_likelihood
        best_alpha = alpha




# printing  the log likelihood for each alpha value
for alpha in alpha_values:
    total_log_likelihood = 0.0

    for sentence in validation_dataset['sentence']:
        total_log_likelihood += sentence_log_probability(sentence,
bigram_counts, alpha, vocabulary_size)

    print(f'Log likelihood for alpha={alpha}: {total_log_likelihood}\
n')
```

```
Log likelihood for alpha=0.001: 3844.9122163187403

Log likelihood for alpha=0.01: 4215.431147046351

Log likelihood for alpha=0.1: 5026.045015426829
```

```python
# Displaying  the best alpha
print(f'Best alpha {best_alpha}\n')
```

```
Best alpha 0.1
```

PROBLEM 7 – Applying Language Models

```python
# Separating  the training dataset into positive and negative
sentences
positive_training = train_dataset.loc[train_dataset['label'] == 1]
negative_training = train_dataset.loc[train_dataset['label'] == 0]

# Computing  vocabulary size and bigram counts for both datasets
vocabulary_size_positive = len(set(word for sentence in
positive_training['sentence'] for word in tokenizer(sentence)))
vocabulary_size_negative = len(set(word for sentence in
negative_training['sentence'] for word in tokenizer(sentence)))

bigram_counts_positive = count_bigrams([tokenizer(sentence) for
sentence in positive_training['sentence']])
bigram_counts_negative = count_bigrams([tokenizer(sentence) for
sentence in negative_training['sentence']])

# Initializing  variables to keep track of predictions
predicted_labels = []
```

```python
# Prior probabilities as computed in Problem 1
prior_probability_positive = positive_training.shape[0] /
train_dataset.shape[0]
prior_probability_negative = negative_training.shape[0] /
train_dataset.shape[0]

# For each sentence in the test set
```

```
for sentence in test_set['sentence']:
    # Computing  log probabilities for positive and negative
sentiments
    log_prob_positive = sentence_log_probability(sentence,
bigram_counts_positive, selected_alpha, vocabulary_size_positive) +
math.log(prior_probability_positive)
    log_prob_negative = sentence_log_probability(sentence,
bigram_counts_negative, selected_alpha, vocabulary_size_negative) +
math.log(prior_probability_negative)

    # Assigning  the sentiment label based on the comparison of scores
    if log_prob_positive > log_prob_negative:
        predicted_labels.append(1)  # Positive sentiment
    else:
        predicted_labels.append(0)  # Negative sentiment

# Calculating  the class distribution of predicted labels
predicted_positive_count = predicted_labels.count(1)
predicted_negative_count = predicted_labels.count(0)




# Comparing  predicted labels to true sentiment labels in the test set
true_labels = test_dataset['label'].tolist()

# Calculating  the accuracy of the experiment
correct_predictions = sum(1 for true, predicted in zip(true_labels,
predicted_labels) if true == predicted)
accuracy = correct_predictions / len(test_dataset)

# printing  the class distribution and accuracy
print('******* Class Distribution of Predicted Labels *********\n')
print(f'Predicted Positive Sentiment =  {predicted_positive_count}\n')
print(f'Predicted Negative Sentiment = {predicted_negative_count}\n')
print(f'Accuracy = {accuracy}\n')
```

```
******* Class Distribution of Predicted Labels *********

Predicted Positive Sentiment =  46

Predicted Negative Sentiment = 54

Accuracy = 0.14
```