CS 585 - HW 1 - Nithin Rajulapati

cwid: A20539650

In this home work we are going to work on SST and QNLI datasets. Both SST and QNLI have been widely used in the NLP research community to develop and evaluate various machine learning models, including neural networks and transformers. They serve as important resources for advancing the field of natural language processing.

GOALS

• To apply the Information Theory concepts.

• To begin to work with text data in python, and apply an open-source NLP package.

• To gain exposure to an openly available NLP research dataset.

Problem 1

Representing English Text (5 pts)

In [1]:
```
#NLTK setup - uncomment and run first time you import NLTK
import nltk
nltk.download('punkt')

import pandas as pd
from nltk.tokenize import word_tokenize
from csv import QUOTE_NONE
import csv
```

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

In [2]:
```
df_sst = pd.read_csv("C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv",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]:
```
df_qnli = pd.read_csv("C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv",delimiter="\t",quoting=QU
df_qnli.head(3)
```

Out[3]:

	index	question	sentence	label
0	0	What came into force after the new constitutio...	As of that day, the new constitution heralding...	entailment
1	1	What is the first major city in the stream of ...	The most important tributaries in this area ar...	not_entailment
2	2	What is the minimum required if you want to te...	In most provinces a second Bachelor's Degree s...	not_entailment

In [49]:
```
# Printing the tokens of the sst.tsv file


with open('C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv', 'r', encoding = 'utf-8') as file:

    header_line = file.readline() # Reading the first line of the file (Means the header l

    token_count = 0 # Initializing the token count is equals to zero

    # Loop through the remaining lines and tokenize each line
    for line in file:

        columns = line.strip().split('\t') # This line splits the file into the columns fo

        text_column = columns[0]

        # Tokenize the text using NLTK's word_tokenize
        tokens = word_tokenize(text_column)

        # Print the first 10 tokens
        print(tokens[:10])

        # Increment the token count
        token_count += len(tokens)

        # If you've reached 10 tokens, break the loop
        if token_count >= 10:
            break
```

```
['hide', 'new', 'secretions', 'from', 'the', 'parental', 'units']
['contains', 'no', 'wit', ',', 'only', 'labored', 'gags']
```

In [5]:
```
# Printing the tokens of the qnli.tsv file


with open('C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv', 'r',encoding = 'utf-8') as file:

    header_line = file.readline() # Reading the first line of the file (Means the header l

    token_count = 0 # Initializing the token count is equals to zero
```

```
    # Loop through the remaining lines and tokenize each line
    for line in file:

        columns = line.strip().split('\t') # This line splits the file into the columns fo

        text_column = columns[1]

        # Tokenize the text using NLTK's word_tokenize
        tokens = word_tokenize(text_column)

        # Print the first 10 tokens
        print(tokens[:10])

        # Increment the token count
        token_count += len(tokens)

        # If you've reached 10 tokens, break the loop
        if token_count >= 10:
            break
```

```
['What', 'came', 'into', 'force', 'after', 'the', 'new', 'constitution', 'was', 'herald']
```

Problem - 2

Word probability (10pts)

If you want to find the encoding of the file manually in your system you can use the below code

import chardet

with open('C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv', 'rb') as file: result = chardet.detect(file.read())

print(result['encoding'])

In [6]:
```
# Collection module in python provides various datatypes ---> Here counter helps in counti

from collections import Counter

import chardet  # Here we use chardet to automatically detect the encoding value of the fi



def create_probability(file, column):
    token_counts = Counter()
    total_tokens = 0
```

```
    # Detect the file's encoding
    with open(file, 'rb') as tsvfile:
        result = chardet.detect(tsvfile.read())
    detected_encoding = result['encoding']

    # Open the file with the detected encoding
    with open(file, 'r', encoding=detected_encoding, newline='') as tsvfile:
        reader = csv.DictReader(tsvfile, delimiter='\t')
        for row in reader:
            tokens = row[column].split()
            token_counts.update(tokens)
            total_tokens += len(tokens)

    # Calculate probabilities for each token
    probability_distribution = {}
    for token, count in token_counts.items():
        probability = count / total_tokens
        probability_distribution[token] = probability

    return probability_distribution

# Specify the file paths and columns
sst_tsv = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
column_sst = 'sentence'

qnli_tsv = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'
column_qnli = 'question'

# Calculate probability distributions
p_d_for_sst = create_probability(sst_tsv, column_sst)
p_d_for_qnli = create_probability(qnli_tsv, column_qnli)

# Printing resulting probability distributions
print(p_d_for_sst)
print(p_d_for_qnli)
```

{'hide': 2.2091636106570052e-05, 'new': 0.0010793342212067083, 'secretions': 4.733922022836
44e-06, 'from': 0.0029334536801509806, 'the': 0.04292878287708845, 'parental': 1.2623792060
897174e-05, 'units': 9.46784404567288e-06, 'contains': 3.629340217507937e-05, 'no': 0.00163
32030978785717, 'wit': 0.0003234846715604901, ',': 0.04099576471776357, 'only': 0.001115627
6233817877, 'labored': 5.522909026642513e-05, 'gags': 0.00019409080293629404, 'that': 0.012
243500325062646, 'loves': 6.154098629687372e-05, 'its': 0.006191970005870063, 'characters':
0.0017925784726473986, 'and': 0.03143324223163396, 'communicates': 1.5779740076121465e-05,
'something': 0.0010603985331153626, 'rather': 0.0005522909026642513, 'beautiful': 0.0004102
732419791581, 'about': 0.002938187602173817, 'human': 0.0005728045647632092, 'nature': 0.00
016884321881449968, 'remains': 0.00017831106286017256, 'utterly': 0.00015464145274599037,
'satisfied': 1.5779740076121465e-05, 'to': 0.019784638107441094, 'remain': 5.52290902664251
3e-05, 'same': 0.0003724018657964666, 'throughout': 0.00015148550473076608, 'on': 0.0038755
04162695432, 'worst': 0.00042763095606289175, 'revenge-of-the-nerds': 1.893568809134576e-0
5, 'clichés': 0.0001925128289286819, 'filmmakers': 0.0002998150614463079, 'could': 0.000866
3077301790684, 'dredge': 2.0513662098957906e-05, 'up': 0.0018320278228377022, "'s": 0.01382
9364202712854, 'far': 0.0005996301228926158, 'too': 0.0020198067297435476, 'tragic': 0.0001
3728373866225675, 'merit': 5.996301228926157e-05, 'such': 0.0008442160940724984, 'superfici
al': 0.0001151921025556867, 'treatment': 0.00010730223251762597, 'demonstrates': 3.94493501
90303663e-05, 'director': 0.0009515183265901245, 'of': 0.02825678055431071, 'hollywood': 0.
0006311896030448586, 'blockbusters': 2.9981506144630785e-05, 'as': 0.008028731750730602, 'p
atriot': 1.420176606850932e-05, 'games': 7.574275236538304e-05, 'can': 0.001543258579444679
3, 'still': 0.0006958865373569566, 'turn': 0.0002572097632407799, 'out': 0.0020229626777587
72, 'a': 0.03409844033049088, 'small': 0.0002714115293092892, 'personal': 0.000200402698966
7426, 'film': 0.006371859042737848, 'with': 0.007487486666119636, 'an': 0.00652176657346100
2, 'emotional': 0.0005065296564434991, 'wallop': 3.471542816746722e-05, '.': 0.020019756234
575303, 'saucy': 1.420176606850932e-05, 'depressed': 2.6825558129406493e-05, 'fifteen-year-
old': 1.2623792060897174e-05, 'suicidal': 1.2623792060897174e-05, 'poetry': 0.0001278158946
1658387, 'are': 0.0028592889017932097, 'more': 0.0033516167921681995, 'deeply': 0.000318750
7495376536, 'thought': 0.00013728373866225675, 'through': 0.0010256831049478953, 'than': 0.
0031464801711786204, 'in': 0.011826915187053038, 'most': 0.0023922085955400143, '`': 0.0015
306347873837822, 'right-thinking': 1.420176606850932e-05, "'": 0.0021697142604667017, 'film
s': 0.0008584178601410078, 'goes': 0.0003187507495376536, 'absurd': 7.732072637299518e-05,
'lengths': 2.0513662098957906e-05, 'for': 0.00617461229178633, 'those': 0.00063276757705247
08, 'moviegoers': 7.889870038060733e-05, 'who': 0.0020056049636750384, 'complain': 9.467844
04567288e-06, 'they': 0.001044618793039241, 'do': 0.0010509306890696896, "n't": 0.003646697
931591671, 'make': 0.001216617959868965, 'movies': 0.001076178273191484, 'like': 0.00281194
9681564845, 'used': 0.00010572425851001382, 'anymore': 1.5779740076121465e-05, 'part': 0.00
04165851380096067, 'where': 0.00041185121598677027, 'nothing': 0.0007921429518212976, 'happ
ening': 6.311896030448586e-05, 'saw': 8.205464839583163e-05, 'how': 0.0008694636781942928,
'bad': 0.001431222424904217, 'this': 0.005309882535614874, 'movie': 0.005622321389122079,
'was': 0.001011481338879386, 'lend': 1.2623792060897174e-05, 'some': 0.0014343783729194412,
'dignity': 6.78528823273223e-05, 'dumb': 0.0002635216592712285, 'story': 0.0022060076626417
81, 'greatest': 0.00015779740076121465, 'musicians': 4.102732419791581e-05, 'cold': 0.00022
72282570961491, 'his': 0.0029350316541585926, 'usual': 0.00017515511484494827, 'intelligenc
e': 0.0003802917358345273, 'subtlety': 9.152249244150451e-05, 'redundant': 4.73392202283643
96e-05, 'concept': 0.00017673308885256042, 'swimming': 1.1045818053285026e-05, 'is': 0.0137
04704256111493, 'above': 0.0002161824390428641, 'all': 0.0026541522808036307, 'young': 0.00
044341069613901317, 'woman': 0.00024143002316465842, 'face': 0.0003250626455681022, 'by':
0.003488900530830456, 'casting': 0.00012623792060897172, 'actress': 0.00014043968667748106,
'whose': 0.0003282185935833265, 'projects': 3.313745415985508e-05, 'doubts': 1.577974007612
1465e-05, 'yearnings': 1.420176606850932e-05, 'it': 0.011168900025878774, 'succeeds': 0.000
11203615454046241, 'equals': 1.420176606850932e-05, 'original': 0.0005159975004891719, 'way
s': 0.00021933838705808837, 'even': 0.0015432585794446793, 'betters': 9.46784404567288e-06,
'if': 0.001656872707992754, 'anything': 0.00043078690407811603, 'see': 0.000852105964110559
2, 'karen': 3.1559480152242934e-06, 'black': 0.0002067145949971912, 'camps': 1.577974007612

1467e-06, 'storm': 1.1045818053285026e-05, 'fringe': 6.311896030448587e-06, 'feminist': 1.2
623792060897174e-05, 'conspiracy': 3.471542816746722e-05, 'theorist': 6.311896030448587e-0
6, 'named': 2.3669610114182198e-05, 'dirty': 7.258680435015874e-05, 'dick': 4.7339220228364
4e-06, 'smile': 0.0001215039985861353, 'your': 0.0015243228913533336, 'comes': 0.0004954838
38390214, 'brave': 2.840353213701864e-05, 'uninhibited': 1.5779740076121465e-05, 'performan
ces': 0.0009199588464378815, 'excruciatingly': 2.6825558129406493e-05, 'unfunny': 0.0001767
3308885256042, 'pitifully': 1.420176606850932e-05, 'unromantic': 6.311896030448587e-06, 'en
riched': 1.893568809134576e-05, 'imaginatively': 2.2091636106570052e-05, 'mixed': 8.0476674
38821948e-05, 'cast': 0.0007984548478517461, 'antic': 1.735771408373361e-05, 'spirits': 7.1
0088303425466e-05, 'which': 0.0013286541144094275, 'half': 0.00030297100946153217, 'dragonf
ly': 3.155948015224293e-05, 'worse': 0.0001925128289286819, ':': 0.0007021984333874053, 'o
r': 0.002698335553016771, 'world': 0.0006927305893417323, 'cinema': 0.0005522909026642513,
'very': 0.0014043968667748106, 'good': 0.0021160631442078887, 'viewing': 0.0001183480505709
11, 'alternative': 5.838503828164943e-05, 'plot': 0.0008883993662856385, 'but': 0.004244750
080476674, 'boilerplate': 1.5779740076121465e-05, 'start': 0.00011045818053285026, 'finis
h': 6.154098629687372e-05, 'action': 0.000978343884719531, 'stilted': 4.4183272213140104e-0
5, 'cylinders': 2.9981506144630785e-05, 'will': 0.0014785616451325815, 'find': 0.0005017957
344206626, 'little': 0.0014280664768889927, 'interest': 0.00029350316541585926, 'often': 0.
0007290239915168118, 'preachy': 6.627490831971016e-05, 'poorly': 0.00014990753072315394, 'a
cted': 0.00012623792060897172, 'year': 0.0005475569806414149, 'sit': 0.0001783110628601725
6, 'another': 0.0006611711091894895, '``': 0.001431222424904217, 'best': 0.0012734250241430
024, 'man': 0.0004939058643826018, "''": 0.001409130788797647, 'clone': 1.1045818053285026e
-05, 'weaving': 2.2091636106570052e-05, 'theme': 8.678857041866807e-05, 'funny': 0.00174366
1278411422, 'issues': 0.00012308197259374744, 'adults': 0.00019093485492106973, 'have': 0.0
02636794566719897, 'marriage': 5.0495168243588695e-05, 'i': 0.0017562850704723192, 'think':
0.0003234846715604901, 'what': 0.0015527264234903523, 'liked': 0.00011203615454046241, '--
': 0.0033263692080464052, 'real': 0.0007384918355624846, 'tucked': 1.420176606850932e-05,
'between': 0.0004828600463293168 4, 'silly': 0.0003313745415985508, 'crude': 7.7320726372995
18e-05, 'storyline': 0.00012939386862419603, 'heroes': 3.787137618269152e-05, 'oblivious':
7.889870038060733e-06, 'existence': 4.5761246220752254e-05, 'sharply': 1.5779740076121465e-
05, 'entire': 0.0001151921025556867, 'point': 0.00035031022968989655, 'shaggy': 2.209163610
6570052e-05, 'dog': 0.0001215039985861353, 'course': 5.3651116258812986e-05, 'nowhere': 6.6
27490831971016e-05, 'classic': 0.00027298950331690135, 'nowheresville': 7.889870038060733e-
06, 'every': 0.0008189685099507041, 'sense': 0.0008568398861333956, 'sometimes': 0.00040711
729396393384, 'dry': 0.00015464145274599037, 'come': 0.0004670803062531954, 'already': 0.00
02572097632407799, 'having': 0.0002272282570961491, 'been': 0.0012245078299070258, 'recycle
d': 5.996301228926157e-05, 'times': 0.0004292089300705039, "'d": 0.00027614545133212564, 'c
are': 0.00045919043621513465, 'count': 4.7339220228364396e-05, 'covers': 2.3669610114182198
e-05, 'territory': 6.943085633493444e-05, 'originality': 0.00011677007656329885, 'suggestin
g': 1.1045818053285026e-05, 'fourth': 2.2091636106570052e-05, 'feature': 0.0003329525156061
6293, '$': 7.258680435015874e-05, '40': 2.3669610114182198e-05, 'million': 3.47154281674672
2e-05, 'version': 0.00019724675095151832, 'game': 0.00015464145274599037, 'gorgeous': 0.000
22565028308853697, 'deceptively': 5.680706427403728e-05, 'minimalist': 3.471542816746722e-0
5, 'cross': 4.102732419791581e-05, 'swords': 1.2623792060897174e-05, 'them': 0.000713244251
4406902, 'proves': 0.00022249433507331268, 'once': 0.00037082389178885446, 'again': 0.00038
97595798802002, 'he': 0.0011140496493741755, 'has': 0.0029760589783565084, 'lost': 0.000203
55864698196692, 'touch': 0.00014990753072315394, 'bringing': 5.0495168243588695e-05, 'off':
0.0008189685099507041, 'superb': 0.00010256831049478953, 'performance': 0.0007148222254483 0
24, 'admittedly': 2.840353213701864e-05, 'middling': 1.5779740076121465e-05, 'disappointmen
ts': 2.2091636106570052e-05, 'horrors': 2.2091636106570052e-05, 'muddle': 3.155948015224293
e-05, 'splashed': 1.1045818053285026e-05, 'bloody': 4.5761246220752254e-05, 'beauty': 0.000
23827407514943414, 'vivid': 0.00013097184263180818, 'any': 0.0012434435179983715, 'scorses
e': 6.311896030448586e-05, 'ever': 0.0007384918355624846, 'given': 0.00014990753072315394,
'us': 0.0010809121952143205, 'many': 0.0009325826384987786, 'pointless': 0.0001467515827079
2963, 'beautifully': 0.0003045489834691443, 'contrived': 0.00021460446503525195, 'well-wor

```
ms': 1.892863903085368e-05, 'manufacturer?': 1.892863903085368e-05, "dynasty's": 1.89286390
3085368e-05, 'rhe': 1.892863903085368e-05, 'bounded': 1.892863903085368e-05, 'automata': 1.
892863903085368e-05, 'Dobson': 1.892863903085368e-05, 'Wells': 1.892863903085368e-05, 'Farg
o': 1.892863903085368e-05, 'farms?': 1.892863903085368e-05, 'signals?': 1.892863903085368e-
05, 'develped': 1.892863903085368e-05, 'criticism': 1.892863903085368e-05, 'NY': 1.89286390
3085368e-05, 'impacts': 1.892863903085368e-05, 'Education': 1.892863903085368e-05, "FIS'":
1.892863903085368e-05, 'agenda': 1.892863903085368e-05}
```

In [7]:

```python
def calculate_sum_of_probabilities(probability_distribution):
    return sum(probability_distribution.values())

# This Function helps in creating the probability distribution from the list of tokens...

def create_probability_distribution(tokens):
    total_tokens = len(tokens)
    token_counts = Counter(tokens)
    probability_distribution = {}

    for token, count in token_counts.items():
        probability = count / total_tokens
        probability_distribution[token] = probability

    return probability_distribution

# Function to tokenize a TSV file and create a probability distribution
def create_probability_distribution_from_tsv(filename, text_column):
    # Tokenize the text column of the TSV file
    tokens = []

    with open(filename, 'r', newline='', encoding='utf-8') as tsvfile:  # Specify the enco
        reader = csv.DictReader(tsvfile, delimiter='\t')
        for row in reader:
            text = row[text_column]
            text_tokens = word_tokenize(text)  # Use word_tokenize from NLTK
            tokens.extend(text_tokens)

    # Create the probability distribution
    probability_distribution = create_probability_distribution(tokens)
    return probability_distribution

# File paths for SST and QNLI TSV datasets (replace with your actual file paths)
sst_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
qnli_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'

# Specify the actual text column names in your datasets
sst_text_column = 'sentence'
qnli_text_column = 'question'

# Create probability distributions for SST and QNLI datasets
sst_distribution = create_probability_distribution_from_tsv(sst_file, sst_text_column)
qnli_distribution = create_probability_distribution_from_tsv(qnli_file, qnli_text_column)

# Check if the sums are close to 1 (allowing for some rounding error)
epsilon = 1e-10  # we use this small tolerance for rounding error
```

```
is_sst_sum_close_to_1 = abs(calculate_sum_of_probabilities(sst_distribution) - 1) < epsilo
is_qnli_sum_close_to_1 = abs(calculate_sum_of_probabilities(qnli_distribution) - 1) < epsi

# Print the results
print("Is SST Probability Distribution Sum Close to 1?", is_sst_sum_close_to_1)
print("Is QNLI Probability Distribution Sum Close to 1?", is_qnli_sum_close_to_1)
```

```
Is SST Probability Distribution Sum Close to 1? True
Is QNLI Probability Distribution Sum Close to 1? True
```

Problem - 3

Entropy (20pts)

In [12]:
```
def calculate_entropy(probability_distribution):
    entropy = 0.0

    for probability in probability_distribution.values():
        if probability > 0:
            entropy -= probability * math.log2(probability)

    return entropy

def create_probability_distribution_from_tsv(filename, text_column):
    tokens = []

    with open(filename, 'r', newline='', encoding='utf-8') as tsvfile:
        reader = csv.DictReader(tsvfile, delimiter='\t')
        for row in reader:
            text = row[text_column]
            text_tokens = word_tokenize(text)  # Use word_tokenize from NLTK
            tokens.extend(text_tokens)

    total_tokens = len(tokens)
    token_counts = Counter(tokens)
    probability_distribution = {}

    for token, count in token_counts.items():
        probability = count / total_tokens
        probability_distribution[token] = probability

    return probability_distribution

def compute_entropy_for_datasets(SST, Qnli, text_column):
    # Create probability distributions for both datasets
    dataset1_distribution = create_probability_distribution_from_tsv(SST, text_column)
    dataset2_distribution = create_probability_distribution_from_tsv(Qnli, text_column)

    # Compute entropy for both datasets
    entropy_dataset1 = calculate_entropy(dataset1_distribution)
    entropy_dataset2 = calculate_entropy(dataset2_distribution)

    return entropy_dataset1, entropy_dataset2
```

```
# Example usage:
SST = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
Qnli = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'
# text_column = 'sentence'  ------------> Exception ------> ERROR!!

entropy1, entropy2 = compute_entropy_for_datasets(SST, Qnli, text_column)

print("Entropy for SST dataset:", entropy1)
print("Entropy for Qnli dataset:", entropy2)
```

```
Entropy for SST dataset: 10.079162530566823
Entropy for Qnli dataset: 10.263300881723977
```

In [15]:
```
import math

# Define your compute_word_level_entropy function (assuming you have a probability_distribu
def compute_word_level_entropy(probability_distribution, tokens):
    entropy = 0.0

    for token in tokens:
        probability = probability_distribution.get(token, 0)  # Get the token's probabilit
        if probability > 0:
            entropy -= probability * math.log2(probability)

    return entropy

# Example usage:
sst_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
sst_text_column = 'sentence'

# Assuming you have already loaded the tokens for SST dataset
sst_tokens = ["token1", "token2", "token3"]  # Replace with your actual list of tokens

sst_word_level_entropy = compute_word_level_entropy(sst_distribution, sst_tokens)

qnli_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'
qnli_text_column = 'question'

# Assuming you have already loaded the tokens for QNLI dataset
qnli_tokens = ["token4", "token5", "token6"]  # Replace with your actual list of tokens

qnli_word_level_entropy = compute_word_level_entropy(qnli_distribution, qnli_tokens)

# Printing the word-level entropy for both datasets separately
print("Word-level Entropy for SST Dataset:", sst_word_level_entropy)
print("Word-level Entropy for QNLI Dataset:", qnli_word_level_entropy)
```

```
Word-level Entropy for SST Dataset: 0.0
Word-level Entropy for QNLI Dataset: 0.0
```

In [19]:
```
# File paths for SST and QNLI TSV datasets
sst_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
qnli_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'
```

```
sst_text_column = 'sentence'
qnli_text_column = 'question'

# Create probability distributions for SST and QNLI datasets
sst_distribution = create_probability_distribution_from_tsv(sst_file, sst_text_column)
qnli_distribution = create_probability_distribution_from_tsv(qnli_file, qnli_text_column)

# We have to compute word-level entropy for SST dataset using its distribution and tokens
sst_tokens = []  # Initialize an empty list for tokens
with open(sst_file, 'r', newline='', encoding='utf-8') as tsvfile:
    reader = csv.DictReader(tsvfile, delimiter='\t')
    for row in reader:
        text = row[sst_text_column]
        text_tokens = word_tokenize(text)  # Use word_tokenize from NLTK
        sst_tokens.extend(text_tokens)

sst_word_level_entropy = calculate_entropy(sst_distribution)

# We have to compute word-level entropy for QNLI dataset using its own distribution and to
qnli_tokens = []  # Let us initialize an empty list for tokens
with open(qnli_file, 'r', newline='', encoding='utf-8') as tsvfile:
    reader = csv.DictReader(tsvfile, delimiter='\t')
    for row in reader:
        text = row[qnli_text_column]
        text_tokens = word_tokenize(text)  # Use word_tokenize from NLTK
        qnli_tokens.extend(text_tokens)

qnli_word_level_entropy = calculate_entropy(qnli_distribution)

# Printing the word-level entropy for both the datasets separately for the better understa
print("Word-level Entropy for SST Dataset:", sst_word_level_entropy)
print("Word-level Entropy for QNLI Dataset:", qnli_word_level_entropy)
```

```
Word-level Entropy for SST Dataset: 10.079162530566823
Word-level Entropy for QNLI Dataset: 9.205137131217116
```

Problem 4

Kullback-Leibler (KL) (20 points)

In [30]:

```
# File paths for SST and QNLI TSV datasets
sst_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
qnli_file = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'

# Specify the actual text column names in your datasets
sst_text_column = 'sentence'
qnli_text_column = 'question'

# Create probability distributions for SST and QNLI datasets
sst_distribution = create_probability_distribution_from_tsv(sst_file, sst_text_column)
qnli_distribution = create_probability_distribution_from_tsv(qnli_file, qnli_text_column)

# Compute KL Divergence between SST and QNLI distributions
```

```
kl_divergence = calculate_kl_divergence(sst_distribution, qnli_distribution)

print("KL Divergence from Dataset 1 to Dataset 2:", kl_divergence)
```

```
KL Divergence from Dataset 1 to Dataset 2: 1.0375353571049182
```

In [33]:
```
kl_divergence_1_to_2 = calculate_kl_divergence(sst_distribution, qnli_distribution)

# Compute KL Divergence from dataset 2 to dataset 1
kl_divergence_2_to_1 = calculate_kl_divergence(qnli_distribution, sst_distribution)

# Print the KL divergences
print("KL Divergence from SST to QNLI:", kl_divergence_1_to_2)
print("KL Divergence from SST to QNLI:", kl_divergence_2_to_1)

# Check if KL divergences are not equal
if kl_divergence_1_to_2 != kl_divergence_2_to_1:
    print("KL Divergence is not symmetric.")
else:
    print("KL Divergence is symmetric.")
```

```
KL Divergence from SST to QNLI: 1.0375353571049182
KL Divergence from SST to QNLI: 1.365062954092768
KL Divergence is not symmetric.
```

Problem 5

Entropy Rate (20 pts)

In [43]:
```
# Defining the main function for printing the per-word entropy

def main():

    sst = 'C:/Users/Nithin/Desktop/NLP/HW_1/train.tsv'
    qnli = 'C:/Users/Nithin/Desktop/NLP/HW_1/dev.tsv'

    # Text columns in the datasets
    text_column = 'sentence'
    text_column2 = 'question'

    # Load probability distributions from the TSV datasets
    sst_distribution = create_probability_distribution_from_tsv(sst, text_column)
    qnli_distribution = create_probability_distribution_from_tsv(qnli, text_column2)

    # Define a sample message (replace with your own message)
    sample_message = "Hello, How are you today?"

    # Compute per-word entropy rates for the sample message relative to both distributions
    SST = calculate_per_word_entropy_rate(sample_message, sst_distribution)
    QNLI = calculate_per_word_entropy_rate(sample_message, qnli_distribution)

    # Print the per-word entropy rates
    print("Per-Word Entropy Rate for Dataset 1:", SST)
    print("Per-Word Entropy Rate for Dataset 2:", QNLI)
```

```
if __name__ == "__main__":
    main()
```

```
Per-Word Entropy Rate for Dataset 1: 0.036040634866231366
Per-Word Entropy Rate for Dataset 2: 0.06609008751360802
```

In [40]:
```
# Define your movie review text
movie_review_text = """
The movie was awesome and the acting of the hero and heroine  has said to be very nice """

# Preprocess the movie review text (you can customize this preprocessing)
movie_review_text = movie_review_text.lower()  # Convert to lowercase
movie_review_text = ''.join([c for c in movie_review_text if c.isalnum() or c.isspace()])

# Calculate the per-word entropy rate for SST and QNLI
sst_entropy_rate = calculate_per_word_entropy_rate(movie_review_text, sst_distribution)
qnli_entropy_rate = calculate_per_word_entropy_rate(movie_review_text, qnli_distribution)

# Print the entropy rates
print("Per-Word Entropy Rate for SST Distribution:", sst_entropy_rate)
print("Per-Word Entropy Rate for QNLI Distribution:", qnli_entropy_rate)
```

```
Per-Word Entropy Rate for SST Distribution: 8.727779372471643
Per-Word Entropy Rate for QNLI Distribution: 5.545886422503157
```

In [ ]: