QUESTION 1 •Read in these two GLUE datasets (see section “DATA” above). Also convert alphabetical characters to lower case •Convert each dataset into a single list of tokens by applying the function “word_tokenize()” in the NLTK :: nltk.tokenize package. We will use these lists represent two distributions of English text. • To show you have finished this step, print the first 10 tokens from each dataset.

In [1]:
```python
import pandas as pd
import nltk
import math
from collections import Counter
import numpy as np
from nltk.tokenize import word_tokenize
from csv import QUOTE_NONE
#SST dataset
df_sst=pd.read_csv('C:/Users/mano2/OneDrive/nlp/SST-2/train.tsv',delimiter='\t')
df_sst.sentence.str.lower()
#print(df_sst.sentence)
s=' '.join(df_sst['sentence'])
token1 = word_tokenize(s)
print(token1[:10])
#list[:10]
#QNLI dataset
df_qnli = pd.read_csv('C:/Users/mano2/OneDrive/nlp/QNLI/dev.tsv',delimiter='\t', quoting=QUOTE_NONE)
#print(df_qnli)
df_qnli.sentence.str.lower()
#print(df_qnli.sentence)
t=' '.join(df_qnli['sentence'])
token2 = word_tokenize(t)
print(token2[:10])
```

```
['hide', 'new', 'secretions', 'from', 'the', 'parental', 'units', 'contains', 'no', 'wit']
['As', 'of', 'that', 'day', ',', 'the', 'new', 'constitution', 'heralding', 'the']
```

QUESTION 2 • Write a python function that creates a probability distribution from a list of tokens. This function should return a dictionary that maps a token to a probability (I.e., maps a string to a floating-point value) • Apply your function to the list created in Problem 1 to create SST and QNLI distributions. • Show that both probability distributions sum to 1, allowing for some small numerical rounding error.

In [2]:
```python
def create_probability_distribution(tokens):
    token_count = len(tokens)
```

```
    token_probability = {}


    for token in tokens:
        if token in token_probability:
            token_probability[token] += 1
        else:
            token_probability[token] = 1

    for token, count in token_probability.items():
        token_probability[token] = count / token_count
        w = token_probability[token]
    return token_probability

sst_distribution = create_probability_distribution(token1)
qnli_distribution = create_probability_distribution(token2)
#print("probability distribution of SST :",sst_distribution)
#print("probability distribution of QNLI:",qnli_distribution)

# sum of probabilities for each distribution
sum_sst_probabilities = sum(sst_distribution.values())
sum_qnli_probabilities = sum(qnli_distribution.values())
print("sum of probability of SST:",sum_sst_probabilities)
print("sum of probability of SST:",sum_qnli_probabilities)
```

```
sum of probability of SST: 1.0000000000000093
sum of probability of SST: 0.9999999999997673
```

QUESTION 3 • Write a python function that computes the entropy of a random variable, input as a probability distribution. • Use this function to compute the word-level entropy of SST and QNLI, using the distributions you created in Problem 2.

In [3]:

```
def entropy(n):


    # If prob_distribution is a list, convert it to a dictionary
    if isinstance(n, list):
        n = {str(value): pr for value, pr in enumerate(n)}

    # Compute entropy
    entropy_value = -sum(p * np.log2(p) for p in n.values() if p > 0)
```

```
    return entropy_value
a = create_probability_distribution(token1)
b = create_probability_distribution(token2)
print("entropy of SST:",entropy(a))
print("entropy of QNLI:",entropy(b))
```

```
entropy of SST: 10.078985771131196
entropy of QNLI: 10.24463425143881
```

QUESTION 4 • Write a python function to compute the KL divergence between two probability distributions. • Apply this function to the distributions you created in Problem 2 to show that KL divergence is not symmetric.

In [4]:

```
def kl_divergence(p, q):

    k = 0
    for key in p.keys():
        if key in q:
            k += p[key] * math.log2(p[key] / q[key])
    return k

n = kl_divergence(sst_distribution, qnli_distribution)
d = kl_divergence(qnli_distribution, sst_distribution)
print("KL Divergence (P || Q):", n)
print("KL Divergence (Q || P):", d)
```

```
KL Divergence (P || Q): 0.8917789637618023
KL Divergence (Q || P): 0.4481196433279539
```

QUESTION 5 • Write a python function that computes the per-word entropy rate of a message relative to a specific probability distribution. • Find a recent movie review online (any website) and compute the entropy rates of this movie review using the distributions you created for both SST and QNLI datasets. Show results in your notebook.

In [5]:

```
def combine_distributions(distribution1, distribution2):
    combined_distribution = {**distribution1, **distribution2}
    return combined_distribution
distribution1 = sst_distribution
distribution2 = qnli_distribution
combined_distribution = combine_distributions(distribution1, distribution2)
#print(combined_distribution)
```

```python
def per_word_entropy_rate(message, probability_distribution):
    words = message.split()
    entropy_rate = 0.0
    for word in words:
        if word in probability_distribution:
            probability = probability_distribution[word]
            entropy_rate += -probability * math.log2(probability)
    num_words = len(words)
    if num_words > 0:
        entropy_rate /= num_words
    return entropy_rate
message = "Mission Impossible is a must-see for action movie enthusiasts. It combines heart-stopping action with compelling st
probability_distribution = combined_distribution
entropy_rate = per_word_entropy_rate(message, probability_distribution)
print("Per-word Entropy Rate:", entropy_rate)
probability_distribution = sst_distribution
entropy_rate = per_word_entropy_rate(message, probability_distribution)
print("Per-word Entropy Rate(SST):", entropy_rate)
probability_distribution = qnli_distribution
entropy_rate = per_word_entropy_rate(message, probability_distribution)
print("Per-word Entropy Rate(QNLI):", entropy_rate)
```

```
Per-word Entropy Rate: 0.035689368723000013
Per-word Entropy Rate(SST): 0.040103691872084344
Per-word Entropy Rate(QNLI): 0.03539999808969854
```