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 numpy as np
import math
import statistics
import itertools
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/nikolausschultze/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [2]:
```
df_sst = pd.read_csv("SST-2/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("QNLI//dev.tsv",delimiter="\t",quoting=QUOTE_NONE)
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 [4]:
```
# demo - split a sentence into tokens
some_sentence = df_sst.sentence.iloc[-100]
word_tokenize(some_sentence)
```

Out[4]:
```
['really', ',', 'save', 'your', 'disgust', 'and', 'your', 'indifference']
```

Problem 1 – Representing English Text

In [5]:
```
tokenizedSST = []
for i in df_sst.index:
    sentenceSST = df_sst.sentence.iloc[i]
    sentenceSST = sentenceSST.lower()
    tokenizedSST.extend(word_tokenize(sentenceSST))

print(tokenizedSST[:10])
#print(len(tokenizedSST))
```

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

In [6]:
```
tokenizedQNLI = []
for i in df_qnli.index:
    sentenceQNLI = df_qnli.sentence.iloc[i]
    sentenceQNLI = sentenceQNLI.lower()
    tokenizedQNLI.extend(word_tokenize(sentenceQNLI))

print(tokenizedQNLI[:10])
#print(len(tokenizedQNLI))
```

```
['as', 'of', 'that', 'day', ',', 'the', 'new', 'constitution', 'heralding', 'the']
```

Problem 2 – Word Probability

In [7]:
```
def wordProbabilityFunction(data):
    dict = {}
    sum = 0
    for i in data:
        if i not in dict:
            count = data.count(i) / len(data)
            dict[i] = count
            sum = sum + count

    return dict, sum
    #print(len(data))

#choose the word, go through the dataframe and count how many times that word appears in the dataframe, return the word and the count into the dict, move
```

In [8]:
```
dictSST, sumSST = wordProbabilityFunction(tokenizedSST)
N = 10
outputSST = dict(itertools.islice(dictSST.items(), N))
print("SST Dictionary first 10 world probabilities : " + str(outputSST))
#print("SST Dictionary: ", dictSST)
print("Probability Distribution Sum: ", sumSST)
```

```
SST Dictionary first 10 world probabilities : {'hide': 2.208187960959237e-05, 'new': 0.0010788575466400842, 'secretions': 4.731831344912065e-06, 'from': 0.0029321581567308725, 'the': 0.04290982391278284, 'parental': 1.2618216919767068e-05, 'units': 9.4636626898253e-06, 'contains': 3.6277373644330032e-05, 'no': 0.0016671819105242237, 'wit': 0.00032334180885690311}
Probability Distribution Sum:  0.9999999999998211
```

In [9]:
```
dictQNLI, sumQNLI = wordProbabilityFunction(tokenizedQNLI)
N = 10
outputQNLI = dict(itertools.islice(dictQNLI.items(), N))
print("QNLI Dictionary first 10 world probabilities : " + str(outputQNLI))
print("Probability Distribution Sum: ", sumQNLI)
```

```
QNLI Dictionary first 10 world probabilities : {'as': 0.007995208689483538, 'of': 0.03309725662584749, 'that': 0.007268371535894126, 'day': 0.00037214062637777925, ',': 0.05396039028247799, 'the': 0.06861342729884055, 'new': 0.0017153356824710136, 'constitution': 8.14057612020142e-05, 'heralding': 1.7444091686145903e-05, 'second': 0.0004477316866110781 5}
Probability Distribution Sum:  0.9999999999998687
```

Problem 3 – Entropy

In [10]:
```
def entropyFunction(dict):
    entropy = 0
    for value in dict:
        pX = dict[value]
        temp = pX * math.log2(pX)
        entropy = entropy + temp

    entropy = entropy * -1
    return entropy
```

In [11]:
```
entropySST = entropyFunction(dictSST)
print("SST's Entropy: ", entropySST)
```

```
SST's Entropy:  10.079162530566823
```

In [12]:
```
entropyQNLI = entropyFunction(dictQNLI)
print("QNLI's Entropy: ", entropyQNLI)
```

```
QNLI's Entropy:  10.037404792966129
```

Problem 4 – KL Divergence

In [13]:
```
#KL Divergence = - sum of P(X = x) * log_2(Q(X = x)/ P(X = x)))
def klDivergenceFunction(p, q):
    klDivergence = 0
    averageProbability = statistics.mean(list(p.values()))
    for value in p:
        if value in q:
            pX = p[value]
            qX = q[value]
            temp = pX * math.log2((qX) / (pX))
            klDivergence = klDivergence + temp
        else:
            #pass
            klDivergence = klDivergence + (pX * math.log2((averageProbability) / (pX)))

    klDivergence = klDivergence *- 1
    return klDivergence
```

In [14]:
```
klDivergenceSST = klDivergenceFunction(dictSST, dictQNLI)
print("KL Divergence where SST is P(X) and QNLI is Q(X): ", klDivergenceSST)
```

```
KL Divergence where SST is P(X) and QNLI is Q(X):  2.4303639551580383
```

In [15]:
```
klDivergenceQNLI = klDivergenceFunction(dictQNLI, dictSST)
print("KL Divergence where QNLI is P(X) and SST is Q(X): ", klDivergenceQNLI)
```

```
KL Divergence where QNLI is P(X) and SST is Q(X):  1.5924811708541398
```

Problem 5 – Entropy Rate

Moview review by Rick Bentley on Aug 9, 2023 from https://www.rottentomatoes.com/m/top_gun_maverick/reviews?intcmp=rt-what-to-know_read-critics-reviews. Review: "It is all of the flying sequences that are shot in such a way that it makes the moviegoer feel like they are a passenger that gives the movie its energy and makes it so much fun."

In [16]:
```
review = ["It", "is", "all", "of", "the", "flying", "sequences", "that", "are", "shot", "in", "such", "a", "way", "that", "it", "makes", "the", "moviegoe
review = list(map(str.lower, review))
print(review)
```

```
['it', 'is', 'all', 'of', 'the', 'flying', 'sequences', 'that', 'are', 'shot', 'in', 'such', 'a', 'way', 'that', 'it', 'makes', 'the', 'moviegoer', 'fee
l', 'like', 'they', 'are', 'a', 'passenger', 'that', 'gives', 'the', 'movie', 'its', 'energy', 'and', 'makes', 'it', 'so', 'much', 'fun']
```

In [17]:
```
#Entropy Rate = (1 / n) * sum of p(x_1n)log(p(x_1n))
def entropyRateFunction(dict, review):
    entropy = 0
    n = len(review)
    uniqueWords = []
    averageProbability = statistics.mean(list(dict.values()))
    for value in review:
        if value in dict:
            pX = dict[value]
            temp = pX * math.log2(pX)
            entropy = entropy + temp
        else:
            entropy = entropy + (averageProbability * math.log2(averageProbability))
            uniqueWords.append(value)
    entropy = entropy * -1
    entropyRate = entropy / n
    print(uniqueWords)
    return entropy, entropyRate
```

In [18]:
```
entropySST, entropyRateSST = entropyRateFunction(dictSST, review)
print("Entropy for SST is: ", entropySST)
print("Entropy Rate for SST is: ", entropyRateSST)
```

```
['passenger']
Entropy for SST is:  2.135225952504406
Entropy Rate for SST is:  0.05770880952714611
```

In [19]:
```
entropyQNLI, entropyRateQNLI = entropyRateFunction(dictQNLI, review)
print("Entropy for QNLI is: ", entropyQNLI)
print("Entropy Rate for QNLI is: ", entropyRateQNLI)
```

```
['moviegoer', 'fun']
Entropy for QNLI is:  1.8970224089161896
Entropy Rate for QNLI is:  0.051270875916653774
```

PROBLEM 6 – Observed Entropy Rate (Answer in Blackboard) Refer to your results from Problem 5. Which distribution gives you the lowest entropy rate for your movie review? Does this match what you expected? Why or why not?

The distribution that gives me the lowest entropy rate for the movie review is QNLI. I expected this because when we calculated the entropy for both dictionaries, the entropy for the QNLI dictionary was lower than the entropy for the SST dictionary and when we calculated the entropy based on the movie review, the entropy was lower for QNLI than SST. SST was also significantly bigger than QNLI meaning that there are more words which could lower the probability for each word. But that would also decrease the chances of a word in the review being in QNLI.

PROBLEM 7 – Zero probabilities (Answer in Blackboard) Problem 5 required that you handle "zero probabilities" cases, where a token occurred in one dataset but not the other. How did you handle these tokens? (Hint: Dropping the word from both probability distributions is not an ideal solution).

I handled zero probabilities by first calculating the average word probability for the dataset by taking the mean of the list of all the values from the dictionary. I then went through the review checking to see if the word was also in the dictionary. If it was, I added the probability for that word to entropyRate which is the sum of all the word probabilities. If the word was not in the dictionary, I added the average word probability for the current dictionary to the sum. After iterating through the entire review, I took the sum and divided by n to get the entropy rate.

In [ ]:
```
```