HW3: Natural Language Processing

In [1]:
```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import spacy
nlp = spacy.load('en_core_web_sm')
import string
from sklearn.preprocessing import normalize
import numpy as np
```

In [2]:
```
data = pd.read_csv("qa.csv")
data.head()
```

Out[2]:

	question	chatgpt_answer	human_answer
0	What happens if a parking ticket is lost / des...	If a parking ticket is lost or destroyed befor...	In my city you also get something by mail to t...
1	why the waves do n't interfere ? first , I 'm ...	Interference is the phenomenon that occurs whe...	They do actually . That 's why a microwave ove...
2	Is it possible to influence a company's action...	Yes, it is possible to influence a company's a...	Yes and no. This really should be taught at ju...
3	Why do taxpayers front the bill for sports sta...	Sports stadiums are usually built with public ...	That 's the bargaining chip that team owners u...
4	Why do clothing stores generally have a ton of...	There are a few reasons why clothing stores ma...	Your observation is almost certainly a matter ...

Q1. Tokenize function

In [3]:
```
def tokenize(docs, lemmatized=True, remove_stopword=True, remove_punct = True):

    tokenized_docs = []

    for doc in docs:
    # Create a new spacy document for the input string
        doc = nlp(doc)

    # Initialize a list to store tokens boolean parameter to indicate if tokens
        each_lemma = []
    # Initialize a list to store tokens boolean parameter to indicate if tokens
        each_stopword = []
    # Initialize a list to store tokens
        each_token_doc = []

        for token in doc:
            # to recognize this token is lemmatized or not
            # and create a list of boolean parameter
            if token.text.lower() == token.lemma_.lower():
                each_lemma.append(False)
```

```
            else:
                each_lemma.append(True)
            # to create a list of tokens
            each_token_doc.append(token.text.lower())
            # to create a list of boolean parameter
            each_stopword.append(token.is_stop)

        if lemmatized==True:
            for x in range(len(each_token_doc)):
                if each_lemma[x] == True:
                    # to change a token if this token is lemmatized
                    each_token_doc[x] = doc[x].lemma_.lower()



        if remove_stopword == True:
            index = 0
            for each_word in each_token_doc[:]:
                if each_stopword[index] == True:
                    # to remove a token if this token is a stop word
                    each_token_doc.remove(each_word)
                index += 1

        if remove_punct == True:
            punct = string.punctuation
            for word in each_token_doc[:]:
                 if word in (punct):
                    # # to remove a token if this token is a punctuation
                    each_token_doc.remove(word)

        # Add the list of tokens to the list of tokenized docs
        tokenized_docs.append(each_token_doc)

    return tokenized_docs
```

Q2. Sentiment Analysis

Analysis:

- Try different tokenization parameter configurations (lemmatized, remove_stopword, remove_punct), and observe how sentiment results change.
- Do you think, in general, which tokenization configuration should be used? Why does this configuration make the most senese?
- Do you think, overall, ChatGPT-generated answers are more posive or negative than human answers? Use data to support your conclusion.

In [4]:
```
def compute_sentiment(gen_tokens, ref_tokens, pos, neg ):

    result = None
    # to count the number of row
    number = len(gen_tokens)
    # to initialize a list of the sentiment for ChatGPT-generated and human ans
    sentiment_gen = []
    sentiment_ref = []
```

```
    for x in range(number):
        pos_gen = 0
        neg_gen = 0
        pos_ref = 0
        neg_ref = 0

        for each_word in gen_tokens[x]:
            # to find this token is positive or negative
            if each_word in pos:
                pos_gen +=1

            elif each_word in neg:
                neg_gen +=1

        for each_word in ref_tokens[x]:
            # to find this token is positive or negative
            if each_word in pos:
                pos_ref +=1

            elif each_word in neg:
                neg_ref +=1

        # to avoid divied by 0
        if pos_gen == 0 and neg_gen == 0:
            sentiment_gen.append(0)
        else:
            sentiment = (pos_gen - neg_gen)/(pos_gen + neg_gen)
            sentiment_gen.append(sentiment)
        # to avoid divied by 0
        if pos_ref == 0 and neg_ref == 0:
            sentiment_ref.append(0)
        else:
            sentiment = (pos_ref - neg_ref)/(pos_ref + neg_ref)
            sentiment_ref.append(sentiment)


    data = {
        "gen_sentiment" : sentiment_gen,
        "ref_sentiment" : sentiment_ref
    }

    result = pd.DataFrame(data)

    return result
```

In [5]:
```
pos = pd.read_csv("positive-words.txt", header = None)
neg = pd.read_csv("negative-words.txt", header = None)
```

In [6]:
```
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=False, \
                      remove_stopword=False, remove_punct = False)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,
                      remove_stopword=False, remove_punct = False)
result = compute_sentiment(gen_tokens,
                           ref_tokens,
                           pos[0].values,
                           neg[0].values)
result.head()
```

Out[6]:

	gen_sentiment	ref_sentiment
0	0.000000	-0.500000
1	-0.777778	0.076923
2	0.666667	0.200000
3	1.000000	0.200000
4	0.600000	-0.333333

In [7]:
```
gen_tokens_1 = tokenize(data["chatgpt_answer"], lemmatized=True, \
                        remove_stopword=True, remove_punct = True)
ref_tokens_1 = tokenize(data["human_answer"], lemmatized=True, \
                        remove_stopword=True, remove_punct = True)
result_1 = compute_sentiment(gen_tokens_1,
                             ref_tokens_1,
                             pos[0].values,
                             neg[0].values)
result_1.head()
```

Out[7]:

	gen_sentiment	ref_sentiment
0	-0.230769	-0.500000
1	-0.777778	-0.142857
2	0.666667	0.111111
3	1.000000	0.200000
4	0.600000	-0.333333

In [8]:
```
gen_tokens_2 = tokenize(data["chatgpt_answer"], lemmatized=True, \
                        remove_stopword=False, remove_punct = True)
ref_tokens_2 = tokenize(data["human_answer"], lemmatized=True, \
                        remove_stopword=False, remove_punct = True)
result_2 = compute_sentiment(gen_tokens_2,
                             ref_tokens_2,
                             pos[0].values,
                             neg[0].values)
result_2.head()
```

Out[8]:

	gen_sentiment	ref_sentiment
0	-0.230769	-0.500000
1	-0.777778	-0.066667
2	0.666667	0.200000
3	1.000000	0.200000
4	0.600000	-0.333333

In [9]:
```
gen_tokens_3 = tokenize(data["chatgpt_answer"], lemmatized=False, \
                        remove_stopword=False, remove_punct = True)
ref_tokens_3 = tokenize(data["human_answer"], lemmatized=False, \
                        remove_stopword=False, remove_punct = True)
result_3 = compute_sentiment(gen_tokens_3,
```

```
                                ref_tokens_3,
                                pos[0].values,
                                neg[0].values)
result_3.head()
```

Out[9]:

	gen_sentiment	ref_sentiment
0	0.000000	-0.500000
1	-0.777778	0.076923
2	0.666667	0.200000
3	1.000000	0.200000
4	0.600000	-0.333333

- After try different tokenization parameter configurations (lemmatized, remove_stopword, remove_punct), we have observed that using lemmatized or remove_stopword may lead to a decrease in sentiment in some answers, but there is no significant difference when using remove_punct. In general, we recommend using tokenization configurations such as: (lemmatized=False, remove_stopword=False, remove_punct = False) or (lemmatized=False, remove_stopword=False, remove_punct = True). - I believe that using lemmatized may alter the meaning of words and lead to misunderstanding, while removing stop words can delete words that may be positive or negative in the answer, which can affect the sentiment analysis. However, we don't consider punctuation tokens as important in the list of positive (negative) words, so we prefer to use (lemmatized=False, remove_stopword=False, remove_punct = True). - In general, it is evident that ChatGPT-generated answers have a more positive sentiment than human answers. After analyzing the sentiment table comparing ChatGPT-generated answers to human answers, it can be observed that some ChatGPT-generated answers have higher sentiment scores than their corresponding human answers

In [10]:
```
result = compute_sentiment(gen_tokens,
                           ref_tokens,
                           pos[0].values,
                           neg[0].values)
result.head(10)
```

Out[10]:

	gen_sentiment	ref_sentiment
0	0.000000	-0.500000
1	-0.777778	0.076923
2	0.666667	0.200000
3	1.000000	0.200000
4	0.600000	-0.333333
5	0.666667	0.333333
6	-1.000000	0.333333
7	0.777778	0.000000
8	1.000000	0.000000
9	0.285714	-0.111111

In [11]:
```
# set if the sentiment of ChatGPT-generated answers have the higher score than
# it it will get 1 points, otherwise 0
larger = np.where(result["gen_sentiment"]>result["ref_sentiment"],1,0)
# to find the equal values between ChatGPT-generated and human answers
```

```
equal = np.where(result["gen_sentiment"]==result["ref_sentiment"],1,0)
# calculate the proportion of the sentiment of ChatGPT-generated answers
# have the higher score than the human answers
proportion=sum(larger)/(len(result["gen_sentiment"])-sum(equal))*100
print("The rate of the sentiment of ChatGPT-generated answers which is higher \
than the human answer is :", proportion," %")
```

```
The rate of the sentiment of ChatGPT-generated answers which is higher than th
e human answer is : 59.66850828729282  %
```

Q3: Performance Evaluation

Analysis:

- Try different tokenization parameter configurations (lemmatized, remove_stopword, remove_punct), and observe how precison and recall change.
- Which tokenization configuration can render the highest average precision and recall scores across all questions?
- Do you think, overall, ChatGPT is able to mimic human in answering these questions?

In [12]:

```
def bigram_precision_recall(gen_tokens, ref_tokens):

    result = None
    # to initialize a list of the bigrams for ChatGPT-generated and human answe
    bigrams_gen = []
    bigrams_ref = []
    for each_doc_gen in gen_tokens:
        bigrams_gen.append(list(nltk.bigrams(each_doc_gen)))
    for each_doc_ref in ref_tokens:
        bigrams_ref.append(list(nltk.bigrams(each_doc_ref)))
    # to initialize a total list of the bigrams appeared in both answers.
    bigrams_gen_ref = []
    # to initialize a list of the precision and recal.
    precision = []
    recal = []
    for x in range(len(bigrams_gen)):
        cor_bi = 0
        # to initialize a list of the bigrams appeared in both answers
        each_bigrams_gen_ref = []
        for each_big_gen in bigrams_gen[x]:
            for each_big_ref in bigrams_ref[x]:
                # to find the bigram in chatGPT appeared in human answer or not
                if each_big_gen == each_big_ref:
                    each_bigrams_gen_ref.append(each_big_gen)
                    cor_bi +=1
                    break
        # to avoid divied by 0
        if len(bigrams_gen[x]) != 0:
            # to calculate the precision in each paired answer
            precision.append(cor_bi/len(bigrams_gen[x]))
        else:
            precision.append(0)

        # to avoid divied by 0
        if len(bigrams_ref[x]) != 0:
            # to calculate the recal in each paired answer
```

```
            recal.append(cor_bi/len(bigrams_ref[x]))
        else:
            recal.append(0)

        bigrams_gen_ref.append(each_bigrams_gen_ref)



    data = {
        "overlapping": bigrams_gen_ref,
        "precision": precision,
        "recal": recal
    }
    result = pd.DataFrame(data)

    return result
```

In [13]:
```
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
result = bigram_precision_recall(gen_tokens,
                                 ref_tokens)
result.head()
```

Out[13]:

	overlapping	precision	recal
0	[(it, goes), (to, pay)]	0.016807	0.042553
1	[(can, cancel), (out, of), (radio, stations), ...	0.033333	0.015695
2	[(to, influence), (a, company), (to, vote), (t...	0.143885	0.060423
3	[(to, be), (the, local), (be, the)]	0.017647	0.051724
4	[(., as), (as, a), (a, result), (result, ,), (...	0.028571	0.072165

In [14]:
```
gen_tokens_1 = tokenize(data["chatgpt_answer"], lemmatized=True,\
                        remove_stopword=True, remove_punct = True)
ref_tokens_1 = tokenize(data["human_answer"], lemmatized=True,\
                        remove_stopword=True, remove_punct = True)
result_1 = bigram_precision_recall(gen_tokens_1,
                                   ref_tokens_1)

result_1.head()
```

Out[14]:

	overlapping	precision	recal
0	[]	0.000000	0.000000
1	[(radio, station)]	0.013699	0.005525
2	[(influence, company)]	0.018182	0.009091
3	[]	0.000000	0.000000
4	[(small, size), (small, size), (size, clothing...	0.054545	0.222222

In [15]:
```
gen_tokens_2 = tokenize(data["chatgpt_answer"], lemmatized=True,\
                        remove_stopword=False, remove_punct = True)
ref_tokens_2 = tokenize(data["human_answer"], lemmatized=True,\
```

```
                              remove_stopword=False, remove_punct = True)
result_2 = bigram_precision_recall(gen_tokens_2,
                                   ref_tokens_2)

result_2.head()
```

Out[15]:

	overlapping	precision	recal
0	[(it, go), (to, pay)]	0.017699	0.046512
1	[(can, cancel), (out, of), (radio, station), (...	0.042781	0.021680
2	[(to, influence), (a, company), (to, vote), (t...	0.148438	0.063973
3	[(to, be), (the, local), (be, the)]	0.019231	0.056604
4	[(there, be), (small, size), (small, size), (s...	0.045455	0.117647

In [16]:

```
gen_tokens_3 = tokenize(data["chatgpt_answer"], lemmatized=False,\
                              remove_stopword=False, remove_punct = True)
ref_tokens_3 = tokenize(data["human_answer"], lemmatized=False,\
                              remove_stopword=False, remove_punct = True)
result_3 = bigram_precision_recall(gen_tokens_3,
                                   ref_tokens_3)

result_3.head()
```

Out[16]:

	overlapping	precision	recal
0	[(it, goes), (to, pay)]	0.017699	0.046512
1	[(can, cancel), (out, of), (radio, stations), ...	0.032086	0.016260
2	[(to, influence), (a, company), (to, vote), (t...	0.148438	0.063973
3	[(to, be), (the, local), (be, the)]	0.019231	0.056604
4	[(as, a), (a, result), (it, 's)]	0.013636	0.035294

- After trying different tokenization parameter configurations such as lemmatized, remove_stopword, and remove_punct, we noticed that the precision and recall rates were lower when we combined all three methods, as removing stop words can change the meaning of two words when combined. The rates were higher when we did not use the remove_stopword method. - Among the four different tokenization parameter configurations, the one with lemmatized=True, remove_stopword=False, and remove_punct=True had the highest average precision and recall scores across all questions. - In my opinion, it is challenging for ChatGPT to mimic human-like answers for these questions, as evidenced by the relatively low precision and recall scores.

In [17]:

```
# to creare a dictionary of four different tokenization parameter configuratio
dict_situ = {0: "lemmatized=False, remove_stopword=False, remove_punct = False"
             1: "lemmatized=True, remove_stopword=True, remove_punct = True",
             2: "lemmatized=True, remove_stopword=False, remove_punct = True",
             3: "lemmatized=False, remove_stopword=False, remove_punct = True"}

precision_array=np.array([result["precision"],result_1["precision"],\
                          result_2["precision"],result_3["precision"]]).T
list_pre = []
# to identify each answers in each configuration belong one of the highest prec
for x in range(precision_array.shape[0]):
    A = np.where(precision_array[x,:]==np.amax(precision_array,axis=1)[x],1,0)
    list_pre.append(list(A))
```

```
array_list_pre=np.array(list_pre)
# to sum how many answers have the highest precision
total_pre = array_list_pre.sum(axis = 0)

print("The precision highest in 4 situations at ",\
      dict_situ[np.argsort(total_pre)[-1]],", with ", \
      max(total_pre)," answers.\n")

recal_array = np.array([result["recal"],result_1["recal"],\
                            result_2["recal"],result_3["recal"]]).T
list_rec = []
# to identify each answers in each configuration belong one of the highest reca
for y in range(recal_array.shape[0]):
    B=np.where(recal_array[y,:]==np.amax(recal_array,axis=1)[y],1,0)
    list_rec.append(list(B))
array_list_rec = np.array(list_rec)
total_rec = array_list_rec.sum(axis = 0)

print("The recal highest in 4 situations at ",\
      dict_situ[np.argsort(total_rec)[-1]],", with ",\
      max(total_rec)," answers.")
```

```
The precision highest in 4 situations at  lemmatized=True, remove_stopword=Fal
se, remove_punct = True , with  120  answers.

The recal highest in 4 situations at  lemmatized=True, remove_stopword=False,
remove_punct = True , with  128  answers.
```

Q4 Compute TF-IDF

In [18]:

```
def compute_tfidf(tokenized_docs):

    smoothed_tf_idf = None
    # process all documents to get list of token list
    docs_tokens={idx:nltk.FreqDist(doc) \
             for idx,doc in enumerate(tokenized_docs)}
    # get document-term matrix
    dtm=pd.DataFrame.from_dict(docs_tokens, \
                           orient="index" )
    dtm=dtm.fillna(0)
    # sort by index (i.e. doc id)
    dtm = dtm.sort_index(axis = 0)
    # convert dtm to numpy arrays
    tf=dtm.values
    # sum the value of each row
    doc_len=tf.sum(axis=1)

    # divide dtm matrix by the doc length matrix
    tf=np.divide(tf, doc_len[:,None])
    # set float precision to print nicely
    np.set_printoptions(precision=2)
    # get document freqent
    df=np.where(tf>0,1,0)


    smoothed_idf=np.log(np.divide(len(tokenized_docs)+1,np.sum(df, axis=0)+1))+
    # get normalized term smoothed_tf_idf matrix
    smoothed_tf_idf = normalize(tf*smoothed_idf)
```

```
    return smoothed_tf_idf
```

Q5. Assess similarity.

Analysis:

- Try different tokenization parameter configurations (lemmatized, remove_stopword, remove_punct), and observe how similarities change.
- Based on similarity, do you think ChatGPT-generate answers are more (or less) relevant to questions than human answers?

In [19]:
```
from sklearn.metrics.pairwise import cosine_similarity

def assess_similarity(question_tokens, gen_tokens, ref_tokens):

    result = None
    # count the number of documents
    number = len(question_tokens)
    # Concatenate these three token lists into a single list
    corpus = question_tokens + gen_tokens + ref_tokens
    # Calculate the smoothed normalized tf_idf matrix for the concatenated
    tf_idf_corpus = compute_tfidf(corpus)
    # Split the tf_idf matrix into sub-matrices
    # corresponding to question_tokens, gen_tokens, and ref_tokens respectively
    tf_idf_ques = tf_idf_corpus[:number,:]
    tf_idf_gen = tf_idf_corpus[number:2*number,:]
    tf_idf_ref = tf_idf_corpus[2*number:,:]

    # to initialize a list of the similarity among
    # question,  ChatGPT-generated answer and human answer
    simi_que_gen = []
    simi_que_ref = []
    gen_ref_similarities = []
    for i in range(number):
        # to take each document and reshape into (1,n)
        each_gen = tf_idf_gen[i].reshape(1, -1)
        each_ques = tf_idf_ques[i].reshape(1, -1)
        each_ref = tf_idf_ref[i].reshape(1, -1)
        # calculate cosince distance of every pair of documents
        # and get the number in lists
        each_simi_gen = cosine_similarity(each_ques, each_gen)[0][0]
        each_simi_ref = cosine_similarity(each_ques, each_ref)[0][0]
        each_simi_gen_ref = cosine_similarity(each_gen, each_ref)[0][0]
        simi_que_gen.append(each_simi_gen)
        simi_que_ref.append(each_simi_ref)
        gen_ref_similarities.append(each_simi_gen_ref)

    result = pd.DataFrame({
        "question_ref_sim": simi_que_ref,
        "question_gen_sim": simi_que_gen,
        "gen_ref_sim": gen_ref_similarities
    })

    return result
```

In [20]:
```
question_tokens = tokenize(data["question"], lemmatized=False,\
                           remove_stopword=False, remove_punct = False)
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)

result = assess_similarity(question_tokens, gen_tokens, ref_tokens)
result.head()
```

Out[20]:

	question_ref_sim	question_gen_sim	gen_ref_sim
0	0.125593	0.570514	0.171820
1	0.136946	0.535158	0.265380
2	0.198697	0.464419	0.451627
3	0.140980	0.418178	0.349791
4	0.177977	0.285982	0.142059

In [21]:
```
question_tokens = tokenize(data["question"], lemmatized=True,\
                           remove_stopword=True, remove_punct = True)
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=True,\
                      remove_stopword=True, remove_punct = True)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,\
                      remove_stopword=True, remove_punct = True)

result_1 = assess_similarity(question_tokens, gen_tokens, ref_tokens)
result_1.head()
```

Out[21]:

	question_ref_sim	question_gen_sim	gen_ref_sim
0	0.068972	0.669553	0.126164
1	0.049655	0.655120	0.128041
2	0.154221	0.415794	0.226399
3	0.061222	0.517329	0.274767
4	0.289738	0.296129	0.448696

In [22]:
```
question_tokens = tokenize(data["question"], lemmatized=True,\
                           remove_stopword=False, remove_punct = True)
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=True,\
                      remove_stopword=False, remove_punct = True)
ref_tokens = tokenize(data["human_answer"], lemmatized=True,\
                      remove_stopword=False, remove_punct = True)

result_2 = assess_similarity(question_tokens, gen_tokens, ref_tokens)
result_2.head()
```

Out[22]:

	question_ref_sim	question_gen_sim	gen_ref_sim
0	0.124752	0.628915	0.204169
1	0.122527	0.573531	0.245706
2	0.240246	0.458176	0.484535
3	0.190396	0.494629	0.416328
4	0.279193	0.306618	0.401073

In [23]:
```
question_tokens = tokenize(data["question"], lemmatized=False,\
                           remove_stopword=False, remove_punct = True)
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = True)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,\
```

```
                        remove_stopword=False, remove_punct = True)

result_3 = assess_similarity(question_tokens, gen_tokens, ref_tokens)
result_3.head()
```

Out[23]:

	question_ref_sim	question_gen_sim	gen_ref_sim
0	0.117081	0.581144	0.163013
1	0.112437	0.539063	0.234389
2	0.209142	0.478742	0.444447
3	0.145250	0.438908	0.347541
4	0.185069	0.295427	0.125363

Based on similarity through trying different tokenization parameter configurations (lemmatized, remove_stopword, remove_punct), we see the ChatGPT-generated answers are more relevant to the questions compared to human answers. We can see the rate below:

In [24]:

```
# to creare a dictionary of four different tokenization parameter configuratio
dict_situ = {0: "lemmatized=False, remove_stopword=False, remove_punct = False"
             1: "lemmatized=True, remove_stopword=True, remove_punct = True",
             2: "lemmatized=True, remove_stopword=False, remove_punct = True",
             3: "lemmatized=False, remove_stopword=False, remove_punct = True"}

gen_array=np.array([result["question_gen_sim"],result_1["question_gen_sim"],\
                            result_2["question_gen_sim"],result_3["question_ge
list_gen = []
# to identify each answers in each configuration belong one of the highest simi
# for question and chatGPT answers
for x in range(gen_array.shape[0]):
    A = np.where(gen_array[x,:]==np.amax(gen_array,axis=1)[x],1,0)
    list_gen.append(list(A))
array_list_gen=np.array(list_gen)
# to sum how many answers have the highest precision
total_gen = array_list_gen.sum(axis = 0)

print("The highest similarity between question and ChatGPT answer in 4 situatio
      dict_situ[np.argsort(total_gen)[-1]],", with ", \
      max(total_gen)," answers.\n")

ref_array = np.array([result["question_ref_sim"],result_1["question_ref_sim"],\
                            result_2["question_ref_sim"],result_3["question_re
list_ref = []
# to identify each answers in each configuration belong one of the highest simi
#for question and human answer
for y in range(ref_array.shape[0]):
    B=np.where(ref_array[y,:]==np.amax(ref_array,axis=1)[y],1,0)
    list_ref.append(list(B))
array_list_ref = np.array(list_ref)
total_ref = array_list_ref.sum(axis = 0)

print("The highest similarity between question and human answer in 4 situations
      dict_situ[np.argsort(total_ref)[-1]],", with ",\
      max(total_ref)," answers.")
```

```
The highest similarity between question and ChatGPT answer in 4 situations at
lemmatized=True, remove_stopword=True, remove_punct = True , with  136  answer
s.

The highest similarity between question and human answer in 4 situations at  l
emmatized=True, remove_stopword=False, remove_punct = True , with  130  answer
s.
```

In [25]:
```
larger=np.where(result_1["question_gen_sim"]>result_1["question_ref_sim"],1,0)
equal=np.where(result_1["question_gen_sim"]==result_1["question_ref_sim"],1,0)
# calculate the proportion of the similarity of ChatGPT-generated answers relev
# have the higher score than the human answers
rate = sum(larger)/(len(result["question_gen_sim"])-sum(equal))*100
print("The rate of the similarity of ChatGPT-generated answers which is higher
than the human answer at ",dict_situ[np.argsort(total_gen)[-1]],"is :", rate,"
larger_2=np.where(result_2["question_gen_sim"]>result_2["question_ref_sim"],1,0
equal_2=np.where(result_2["question_gen_sim"]==result_2["question_ref_sim"],1,0
rate_2 = sum(larger_2)/(len(result["question_gen_sim"])-sum(equal_2))*100
print("The rate of the similarity of ChatGPT-generated answers which is higher
than the human answer at ",dict_situ[np.argsort(total_ref)[-1]],"is :", rate_2,
```

```
The rate of the similarity of ChatGPT-generated answers which is higher than t
he human answer at  lemmatized=True, remove_stopword=True, remove_punct = True
is : 90.95477386934674  %
The rate of the similarity of ChatGPT-generated answers which is higher than t
he human answer at  lemmatized=True, remove_stopword=False, remove_punct = Tru
e is : 83.5  %
```

Q6 (Bonus): Further Analysis (Open question)

- Can you find at least three significant differences between ChatGPT-generated and human answeres? Use data to support your answer.
- Based on these differences, are you able to design a classifier to identify ChatGPT generated answers? Implement your ideas using traditional machine learning models, such as SVM, decision trees.

- Can you find at least three significant differences between ChatGPT-generated and human answeres? Use data to support your answer? + Firstly, in my opinion, the answers generated by ChatGPT tend to have a more positive tone compared to those written by humans. This could be attributed to the fact that ChatGPT's responses are created using data collected from reliable papers or materials, which may limit the use of negative language.

In [26]:
```
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
result=compute_sentiment(gen_tokens, ref_tokens, pos[0].values, neg[0].values)
result.head(10)
```

Out[26]:

	gen_sentiment	ref_sentiment
0	0.000000	-0.500000
1	-0.777778	0.076923
2	0.666667	0.200000
3	1.000000	0.200000
4	0.600000	-0.333333
5	0.666667	0.333333
6	-1.000000	0.333333
7	0.777778	0.000000
8	1.000000	0.000000
9	0.285714	-0.111111

+ Secondly, ChatGPT's responses usually contain the keywords of the question, while human answers sometimes do not include it. Therefore, the similarity between the question and ChatGPT's answer is often higher than the similarity between the question and a human answer.

In [27]:
```
print("Question: ", data["question"][108],"\n")
print("ChatGPT answer: ", data["chatgpt_answer"][108],"\n")
print("Human answer: ", data["human_answer"][108],"\n")
print("---------")
print("The rate of the similarity of ChatGPT-generated answers which is higher than the human answer is :", rate," %")
```

```
Question:  Why ca n't we do lethal injection by just injecting air into the bl
oodstream ? If we 're running out of the drugs , would n't this be relatively
cheap ? Please explain like I'm five.

ChatGPT answer:  Injecting air into the bloodstream, also known as air embolis
m, is not a safe or effective method of lethal injection.
Air embolism occurs when an air bubble enters a blood vessel and blocks the fl
ow of blood. This can be extremely dangerous and potentially fatal, as the air
bubble can travel to the heart or brain and cause serious problems.
Furthermore, injecting air into the bloodstream does not provide a quick and p
ainless death. On the contrary, it can cause a person to experience severe pai
n, panic, and distress as their body struggles to deal with the air bubble.
In summary, injecting air into the bloodstream is not a viable option for leth
al injection because it is not safe and can cause a person to experience signi
ficant suffering.

Human answer:  Takes longer it 's sloppier and far more painful .

---------
The rate of the similarity of ChatGPT-generated answers which is higher than t
he human answer is : 90.95477386934674  %
```

+ Thirdly, the length of ChatGPT-generated answers usually longer than the human answer.

In [28]:
```
gen_tokens = tokenize(data["chatgpt_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
ref_tokens = tokenize(data["human_answer"], lemmatized=False,\
                      remove_stopword=False, remove_punct = False)
length_gen = np.array([len(x) for x in gen_tokens])
length_ref = np.array([len(x) for x in ref_tokens])
arr_gen_ref = np.array([length_gen,length_ref]).T
```

```
length_gen_higher = np.where(length_gen>length_ref,1,0)
rate_length = length_gen_higher.sum()/len(length_gen_higher)*100
print("The rate of the length of ChatGPT-generated answers \
which are longer than the human answer is ", rate_length,"%")
```

```
The rate of the length of ChatGPT-generated answers which are longer than the
human answer is  72.0 %
```

- Based on these differences, are you able to design a classifier to identify ChatGPT generated answers? Implement your ideas using traditional machine learning models, such as SVM, decision trees. + I believe it is possible to develop a classifier to distinguish ChatGPT-generated answers from human answers. + To create the dataset, I plan to include input features such as the tokens of the question and answer, length of the answer, sentiment score, and similarity score between the question and answer. The output column will have a value of 1 if the answer is ChatGPT-generated and 0 otherwise. + The dataset will have a shape of (400, 6). + I intend to shuffle the dataset and split it into training and testing sets to train models such as SVM, decision trees, Random Forest, or Logistic Regression.

In [ ]:

In [ ]: