# CSI5386: Natural Language Processing
# ASSIGNMENT 1

**PROFESSOR**: Dr. Diana Inkpen

---

**TEAM: GROUP 27**

| TEAM MEMBERS | STUDENT ID | STUDENT EMAIL |
|---|---|---|
| Divya Reddy | 300290332 | dredd037@uottawa.ca |
| Ishveen Manjeet Singh Sahi | 300303960 | isahi065@uottawa.ca |
| Syeda Zainab | 300303434 | szain039@uottawa.ca |


---

**Task Distribution among team members:**

Part 1 was completed by all three of us. Because it did not take long, each of the seven subsections was completed independently. Following execution, we compared our results and reached a conclusion.

Part 2 was shared evenly across the datasets. We each completed two sentence embedding models, for a total of six models performed.

Tasks done by Divya:

- Implemented the Part 1 code from scratch
- Part 2: InferSent, SBERT - Roberta
- Contributed to the report for both Part1 and 2

Tasks done by Syeda:

- Implemented the Part 1 code from scratch
- Part 2: Universal Sentence Encoder, Doc2Vec
- Contributed to the report for both Part1 and 2

Tasks done by Ishveen:

- Implemented the Part 1 code from scratch
- Part 2: SimpCSE, SBERT- MpNet
- Contributed to the report for both Part1 and 2

---

**Programming Language used**: Python

**Dataset used:**

Contract Understanding Atticus Dataset (CUAD) v1 is a dataset that comprises 510 files which are a collection of text files with commercial legal contracts specially curated for layers to review the contracts with corporate transactions.

**Corpus creation:**

We concatenate 510 files of the CUAD v1 dataset to form the corpus.

In a command prompt in this folder run the following command:

copy /b *.txt merged_file.txt

As a result, we get all text files in this folder sorted in ascending by date and combined into a single file called

'merged_file.txt'

---

## Part 1: Corpus processing: tokenization and word counting

**NLP Tool - Natural Language Toolkit (NLTK)**

NLTK is the most popular and powerful tool that incorporates numerous Python modules and libraries to perform varied natural language processing. It comprises pre-trained models which help to analyze data efficiently.

**Dataset pre-processing**

Dataset preprocessing is a preliminary step that disregards the non-essential attributes of the corpus to perform the required operations on the obtained clean data that is suitable to gain meaningful insights. Data pre-processing is performed heavily while dealing with sparse data.

We performed the following data pre-processing on our corpus.

1. **Converting all literals of the corpus to lowercase**

   It is a general practice to convert the data to lowercase to maintain consistency of flow while performing NLP tasks.

   Function used: lower() - converts all uppercase characters in a string into lowercase characters and returns it.

   **Code Snippet:**

```
#loading the dataset
corpus = nltk.data.load('megred_file.txt')

#converting all the literals of the corpus to lower case
corpus_data = corpus.lower()
```

2. **Tokenization**

Tokenizing means splitting sentences into tokens.

Function used: word_tokenize() - extracts tokens in the form of words from the corpus.

**Code Snippet:**

```
#extracting tokens in the form of words from the corpus
tokens = word_tokenize(corpus_data)
#print(tokens)
```

3. **Eliminating Stop Words**

Stop words are the most commonly used words. These words do not hold much significance. Eliminating them from the corpus helps the model to focus only on important features of the dataset.

**Code Snippet:**

```
#(f)Loading the 'stopwords.txt' file
#Storing the stop words in a list

file = open('stopwords.txt','r')
data = file.readlines()
stop_words = []
for word in data:
    word = word.rstrip('\n')
    stop_words.append(word)

# print(stop_words)
print("Total stopwords to be removed : ", len(stop_words))

#(f)Excluding Stop Words from the corpus

without_stop_words = []
for word in words_from_corpus:
    if word not in stop_words:
        without_stop_words.append(word)
print("\nNo.of tokens with stop words:\n",len(words_from_corpus))
print("\nNo.of tokens without stop words:\n",len(without_stop_words))
```

**a) Submit a file output.txt with the tokenizer's output for the whole corpus. Include in your report the first 20 lines from output.txt.**

**Steps involved:**

1. Take the pre-processed corpus as input to perform word tokenization
2. Create an 'output.txt' file
3. Write the extracted tokens to the 'output.txt' file

**Code Snippet:**

```
#(a)Performing tokenization for the whole corpus
#extracting tokens in the form of words from the corpus
tokens = word_tokenize(corpus_data)
#print(tokens)

#creating an 'output.txt' file which stores the extracted tokens(words)
output = open('output.txt','w')

for words in tokens:
    output.write(words + "\n")
output.close()
```

**First 20 lines from the 'output.txt' file:**

Users > divya > Code > Assignment1 > PART1 > output.txt

```
1   co-branding
2   and
3   advertising
4   agreement
5   this
6   co-branding
7   and
8   advertising
9   agreement
10  (
11  the
12  ``
13  agreement
14  ''
15  )
16  is
17  made
18  as
19  of
20  june
```

**b) How many tokens did you find in the corpus? How many types (unique tokens) did you have? What is the type/token ratio for the corpus? The type/token ratio is defined as the number of types divided by the number of tokens.**

**Steps involved:**

1. After performing tokenization, the variable 'tokens' holds all the extracted tokens from the corpus. Calculate the length of 'tokens' using the len() function by passing 'tokens' to it.
2. To find the unique tokens use the set() function – which removes duplicate strings and returns unique strings.
   Calculate the length of 'unique_tokens' using the len() function by passing 'unique_tokens' to it.
3. Calculate the type/token ratio by dividing the number of unique tokens by the total number of tokens found in the corpus.

**Code Snippet: 1.**

```
#(b)Total number of tokens found in the corpus
total_tokens = len(tokens)
print(f'Total No.of Tokens : {total_tokens}')
```

**Output:**

```
Total No.of Tokens : 4775318
```

**Code Snippet: 2.**

```
#(b)Unique tokens found in the corpus
unique_tokens = set(tokens)
unique_tokens_len = len(unique_tokens)
print(f'Unique Tokens (CORPUS): {unique_tokens_len}')

print("\nA List of all the unique tokens in the dataset\n")
print(unique_tokens)
```

**Output:**

```
Unique Tokens (CORPUS): 47067
```

**Code Snippet: 3.**

```
#(b)type/token ratio for the corpus
ty_token_ratio = (unique_tokens_len/total_tokens)
print(f'Type Token Ratio : {ty_token_ratio}')
```

**Output:**

```
Type Token Ratio : 0.009856306951704578
```

c) **For each token, print the token and its frequency in a file called tokens.txt (from the most frequent to the least frequent) and include the first 20 lines in your report.**

**Steps involved:**

1. Calculate the frequency of each token using the FreqDist() function from NLTK by passing 'tokens' to it.
2. Sort the token's frequency in descending order.
3. Create a 'tokens.txt' file and append the tokens with their frequency in the format {word : frequency}

**Code Snippet: 1.**

```
#(c)Calculating the frequency of a token
#Using the FreqDist module of nltk
token_freq = FreqDist(tokens)
print(token_freq,"\n")
```

**Output:**

```
<FreqDist with 47067 samples and 4775318 outcomes>
```

**Code Snippet: 2.**

```
#Sorting in descending order of the frequency of tokens
token_dict = {}

for word, freq in token_freq.items():
    token_dict[word] = freq
token_dict_sorted = dict( sorted(token_dict.items(), key=operator.itemgetter(1),reverse=True))
print("Sorted Dictionary in Descending Order \n")

for i in token_dict_sorted.items():
    print(i)
#print(token_dict_sorted)
```

**Code Snippet: 3.**

```
#creating a 'tokens.txt' file which stores the tokens with their frequency
#from the most frequent to the least frequent
tokens_file = open('tokens.txt','w')
for word, freq in token_dict_sorted.items():
    write_data = f'{word} : {freq}\n'
    tokens_file.write(write_data)
tokens_file.close()
print("Data written to the file 'tokens.txt'")
```

**First 20 lines from the 'tokens.txt' file:**

Users > divya > Code > Assignment1 > PART1 > tokens.txt

```
1   the : 257132
2   , : 240576
3   of : 156122
4   to : 129875
5   and : 129054
6   or : 105155
7   . : 103440
8   in : 79933
9   ) : 78092
10  ( : 75436
11  * : 67765
12  any : 62236
13  -- : 58712
14  a : 50444
15  shall : 48794
16  by : 44310
17  agreement : 43617
18  this : 39986
19  be : 39701
20  for : 38724
```

**d) How many tokens appeared only once in the corpus?**

**Steps involved:**

From the obtained frequencies of each token in the above question, check for the tokens with frequency = 1.

**Code Snippet:**

```
#(d)Number of tokens found only once in the corpus
one_occ_tokens = {}
for word, freq in token_dict.items():
    if freq == 1:
        one_occ_tokens[word] = freq

print(f'Tokens with single occurance : {len(one_occ_tokens)}')
print("\n",one_occ_tokens)
```

**Output:**

```
Tokens with single occurance : 20416
```

**e) From the list of tokens, extract only words, by excluding punctuation and other symbols, if any. Please pay attention to the end of the sentence dot (full stops). How many words did you find? List the top 20 most frequent words in your report, with their frequencies. What is the type/token ratio when you use only words (called lexical diversity)?**

**Steps involved:**

1. For each token of the corpus check if it is a word using the **isaplha()** function – which returns True if all the characters are alphabet letters (a-z).
   Append the tokens which satisfy the condition to 'words_from_corpus' list.
   Calculate the length of 'words_from_corpus' using the len() function by passing 'words_from_corpus' to it.
2. Calculate the frequency of each token using the FreqDist() function from NLTK by passing 'words_from_corpus' to it.
   In a tabular format, print the 20 most frequent words with their frequency using the **most_common()** function – which produces a sequence of the n most frequently encountered input values and their respective counts

3. Calculate Lexical diversity i.e. the type/token ratio when using only words by dividing the number of unique words in 'words_from_corpus' by the total number of tokens found in the corpus.

**Code Snippet: 1.**

```
#(e)Extracting only words from the tokens
#i.e. excluding punctuations and other symbols
words_from_corpus = []
for token in tokens:
    if token.isalpha(): #to identify words
        words_from_corpus.append(token)

print("No. of ""WORDS"" Extracted from Tokens:",len(words_from_corpus))
```

**Output:**

```
No. of WORDS Extracted from Tokens: 3809034
```

**Code Snippet: 2.**

```
#Listing the top 20 most frequent words
#Using the FreqDist module of nltk
freq_dist = FreqDist(words_from_corpus)
#Making 2 columns as 'Word', 'Freq'
freq_dist_table = pd.DataFrame(freq_dist.most_common(20), columns = ['Word','Freq'])
freq_dist_table.index += 1
print(freq_dist_table)
```

**Output:**

| | Word | Freq |
|---|---|---|
| 1 | the | 257132 |
| 2 | of | 156122 |
| 3 | to | 129875 |
| 4 | and | 129054 |
| 5 | or | 105155 |
| 6 | in | 79933 |
| 7 | any | 62236 |
| 8 | a | 50444 |
| 9 | shall | 48794 |
| 10 | by | 44310 |
| 11 | agreement | 43617 |
| 12 | this | 39986 |
| 13 | be | 39701 |
| 14 | for | 38724 |
| 15 | such | 36172 |
| 16 | with | 33883 |
| 17 | as | 32907 |
| 18 | party | 32826 |
| 19 | that | 27654 |
| 20 | other | 26395 |

**Code Snippet: 3.**

```
# LEXICAL DIVERSITY - type/token ratio when we use only words

lexical_diversity = (len(set(words_from_corpus))/len(tokens))
#set() - removes duplicate strings and returns unique strings
print("Lexical Diveristy : ",lexical_diversity)
```

**Output:**

```
Lexical Diveristy :  0.0053160857559643145
```

**f) From the list of words, exclude stop words. List the top 20 most frequent words and their frequencies in your report. You can use this list of stop words (or any other that you consider adequate). Also, compute the type/token ratio when you use only word tokens without stop words (called lexical density).**

**Steps involved:**

1. Load the given 'stopwords.txt' file and store these stop words in a list.
   Check for every word in 'words_from_corpus', if the word exists in the list of stop words, exclude it from the list 'without_stop_words'.
2. Calculate the frequency of each token using the FreqDist() function from NLTK by passing 'without_stop_words' to it.
   In a tabular format, print the 20 most frequent words with their frequency using the **most_common()** function – which produces a sequence of the n most frequently encountered input values and their respective counts.
3. Calculate Lexical density i.e. the type/token ratio when using only word tokens without stop words by dividing the number of unique words in 'without_stop_words' by the total number of tokens found in the corpus.

**Code Snippet: 1.**

```
#(f)Loading the 'stopwords.txt' file
#Storing the stop words in a list

file = open('stopwords.txt','r')
data = file.readlines()
stop_words = []
for word in data:
    word = word.rstrip('\n')
    stop_words.append(word)

# print(stop_words)
print("Total stopwords to be removed : ", len(stop_words))

#(f)Excluding Stop Words from the corpus

without_stop_words = []
for word in words_from_corpus:
    if word not in stop_words:
        without_stop_words.append(word)
print("\nNo.of tokens with stop words:\n",len(words_from_corpus))
print("\nNo.of tokens without stop words:\n",len(without_stop_words))
```

**Output:**

```
Total stopwords to be removed :  779

No.of tokens with stop words:
 3809034

No.of tokens without stop words:
 1816856
```

**Code Snippet: 2.**

```
#Listing the top 20 most frequent words after removal of stop words
#Using the FreqDist module of nltk

freqdist_filtered_words = FreqDist(without_stop_words)
filtered_words_table = pd.DataFrame(freqdist_filtered_words.most_common(20), columns = ['Word', 'Freq'])
filtered_words_table.index += 1
print(filtered_words_table)
```

**Output:**

```
            Word   Freq
1      agreement  43617
2          party  32826
3        parties  13509
4        section  13292
5        company  12390
6    information  10920
7        product  10756
8           date  10116
9       products   8168
10        rights   8048
11      services   7861
12    applicable   7533
13      business   7254
14           set   6981
15  confidential   6866
16       written   6799
17         terms   6714
18         right   6676
19        notice   6655
20          term   6574
```

**Code Snippet: 3.**

```
# LEXICAL DENSITY - type/token ratio when we use only word tokens without Stop Words

lexical_density = (len(set(without_stop_words))/len(tokens))
print("Lexical Density : ", lexical_density)
```

**Output:**

```
Lexical Density :  0.005186670290858117
```

g) **Compute all the pairs of two consecutive words (bigrams) (excluding stop words and punctuation). List the most frequent 20 pairs and their frequencies in your report.**

**Steps involved:**

1. From the NLTK library use the **bigrams()** function – which extracts lists of word pairs from a given text, in our case 'without_stop_words'.
2. Calculate the frequency of each bigram using the FreqDist() function from NLTK by passing 'bigrams' to it.
   In a tabular format, print the 20 most frequent bigrams with their frequency using the **most_common()** function – which produces a sequence of the n most frequently encountered input values and their respective counts.

**Code Snippet:**

```
#(g)Computing the pair of Bigrams excluding stop words and punctuations

#nltk.bigrams() - extracts lists of word pairs from a given text
bigrams = list(nltk.bigrams(without_stop_words))
#print(bigrams)

#Using the FreqDist module of nltk
bigrams_freq = FreqDist(bigrams)
most_freq_table = pd.DataFrame(list(bigrams_freq.most_common(20)), columns = ['bigram','freq'])
most_freq_table.index += 1
print("The list of top 20 most occouring bigrams")
print(most_freq_table)
```

**Output:**

```
The list of top 20 most occouring bigrams
                         bigram  freq
1   (confidential, information)  3603
2      (intellectual, property)  2920
3             (effective, date)  2838
4             (written, notice)  2387
5           (terms, conditions)  2087
6               (set, section)  1825
7             (prior, written)  1814
8            (term, agreement)  1708
9     (confidential, treatment)  1530
10     (termination, agreement)  1439
11            (parties, agree)  1417
12       (securities, exchange)  1410
13           (receiving, party)  1367
14               (party, party)  1363
15          (pursuant, section)  1353
16           (written, consent)  1330
17             (united, states)  1265
18           (applicable, law)  1249
19          (agreement, party)  1215
20           (terms, agreement)  1202
```

**Table:**

| | |
|---|---|
| # of tokens (b) | 4775318 |
| # of types (b) | 47067 |
| type/token ratio (b) | 0.009856306951704578 |
| tokens appeared only once (d) | 20416 |
| # of words (excluding punctuation) (e) | 3809034 |
| type/token ratio (excluding punctuation) (e) | 0.0053160857559643145 |
| List the top 3 most frequent words and their frequencies (e) | the : 257132 |
| | of : 156122 |
| | to : 129875 |
| type/token ratio(excluding punctuation and stopwords) (f) | 0.005186670290858117 |
| List the top 3 most frequent words and their frequencies (excluding stopwords) (f) | agreement : 43617 |
| | party : 32826 |
| | parties : 13509 |
| List the top 3 most frequent bigrams and their frequencies (g) | (confidential, information) : 3603 |
| | (intellectual, property) : 2920 |
| | (effective, date) : 2838 |

## Part 2: Evaluation of pre-trained sentence embedding models

The datasets used to evaluate the similarity:

- STS2016.input.answer-answer.txt
- STS2016.input.input.headlines.txt
- STS2016.input.plagiarism.txt
- STS2016.input.postediting.txt
- STS2016.input.question-question.txt

Embedding used:

- Universal Sentence Encoder
- Doc2VEC
- SimpCSE
- InferSent
- SBERT - Roberta
- SBERT- MpNet

Please read the README file. It contains additional information. The code also has comments that can help to understand it.

**Description:**

1. **Universal Sentence Encoder:**
   Universal sentence embedding models encode textual data vectors of high-dimension to perform numerous NLP operations. These model encoders expect the meaning of word sequences rather than separate words. These models are not only trained from individual words but also are optimized for texts having more-than-word lengths like sentences, phrases or paragraphs.

2. **SimpCSE: Simple Contrastive Learning of Sentence Embeddings**
   The Simple Contrastive Learning of Sentence Embeddings (SimCSE) methodology sends every sentence twice to the sentence embedding encoder. They are encoded at slightly different positions in vector space owing to the drop-out. This leads to minimizing the distance between the current two embeddings while distance to the embeddings of other sentences in the same batch is maximised.

3. **SBERT: Sentence BERT**
   Sentence-BERT (SBERT) is a sentence embedding technique that fine-tunes models which are pre-trained in a Siamese network architecture on the NLI task. SBERT performs a pooling

operation by sending each sentence of a pair to BERT and then obtains sentence embeddings from the output contextualised word embeddings.

4. **Doc2Vec:**
   The Doc2Vec technique uses the Word2vec algorithm to create word embeddings. It works on the assumption that a word's meaning is given by its neighboring words.

5. **InferSent:**
   InferSent is a sentence embedding technique which gives semantic representations for English language sentences. The model is trained on natural language inference data and is usually generalised well for a variety of different tasks.

6. **SBERT- MpNet**
   The MPNet model stands for Masked and Permuted Pre-training for Language Understanding. It adopts a novel pre-training method, named masked and permuted language modelling, to seek advantages of the masked and permuted language modelling for natural language understanding.

Data preprocessing steps:

1) Open the file
2) Split the lines using '\\n'
3) Take the first 2 lines of the paragraph in line1 and line2.
4) Remove all special characters from lines

```
with open('STS2016.input.answer-answer.txt') as f:
    lines = str(f.readlines())
s1=str(lines).split('\\n')
line1=[]
line2=[]
for i in range(0,len(s1)-1):
  line1.append(s1[i].split('\\t')[0])
  line2.append(s1[i].split('\\t')[1])
import re
string1=[]
for i in line1:
  string1.append(re.sub(r"[^a-zA-Z0-9]+",' ',i))
string2=[]
for i in line2:
  string2.append(re.sub(r"[^a-zA-Z0-9]+",' ',i))
strin=string1 +string2
```

STEPS:

1. Import libraries and load the model

```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
model = hub.load(module_url)
```

2. Loop over sentences and find cosine similarity of sentence 1 with sentence 2

```
def cosine(u, v):
    return np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))
```

```
similar=[]
for i in range(0, len(string1)-1):
  query = string1[i]
  query_vec = model([query])[0]
  sim = cosine(query_vec, model([string2[i]])[0])
  print("Sentence = ", string1[i], string2[i], "; similarity = ", sim)
  similar.append(sim)
  x=x+1
```

3. The cosine similarity is multiplied by 5 and stored

```
final=[]
for i in similar:
  final.append(i*5)
print(final)
```

4. Store the output data in text file and run it against the perl script and gold standard file
   Command: /correlation-noconfidence.pl STS2016.gs.<dataset>.txt  <model>.tx

**1. Universal Sentence Embedding:**

| | |
|---|---|
| STS2016.input.answer-answer.txt | 0.70202 |
| STS2016.input.headlines.txt | 0.76078 |
| STS2016.input.plagiarism.txt | 0.83741 |
| STS2016.input.postediting.txt | 0.85083 |
| STS2016.input.question-question.txt | 0.74987 |

2. **SimpCSE**

| | |
|---|---|
| STS2016.input.answer-answer.txt | 0.76472 |
| STS2016.input.headlines.txt | 0.79535 |
| STS2016.input.plagiarism.txt | 0.84309 |
| STS2016.input.postediting.txt | 0.84535 |
| STS2016.input.question-question.txt | 0.72033 |

3. **SBERT**

| | |
|---|---|
| STS2016.input.answer-answer.txt | 0.76449 |
| STS2016.input.headlines.txt | 0.93481 |
| STS2016.input.plagiarism.txt | 0.84103 |
| STS2016.input.postediting.txt | 0.85077 |
| STS2016.input.question-question.txt | 0.74247 |

4. **Doc2Vec**

| | |
|---|---|
| STS2016.input.answer-answer.txt | 0.17036 |
| STS2016.input.headlines.txt | 0.41351 |
| STS2016.input.plagiarism.txt | 0.78026 |
| STS2016.input.postediting.txt | 0.73988 |
| STS2016.input.question-question.txt | 0.05413 |

5. **InferSent**

| | |
|---|---|
| STS2016.input.answer-answer.txt | 0.43127 |
| STS2016.input.headlines.txt | 0.57937 |
| STS2016.input.plagiarism.txt | 0.66033 |
| STS2016.input.postediting.txt | 0.75669 |
| STS2016.input.question-question.txt | 0.57318 |

6. **SBERT-Mpnet**

| | |
|---|---|
| STS2016.input.answer-answer.txt | 0.74429 |
| STS2016.input.headlines.txt | 0.84362 |
| STS2016.input.plagiarism.txt | 0.83397 |
| STS2016.input.postediting.txt | 0.85523 |
| STS2016.input.question-question.txt | 0.82987 |

**SIMILARITY:**

| DATASETS | USE | SimCSE | SBERT-Roberta | Doc2Vec | InferSent | SBERT-MpNet | Best Score |
|---|---|---|---|---|---|---|---|
| STS2016.input.answer-answer.txt | 0.70202 | 0.76472 | 0.76449 | 0.17036 | 0.43127 | 0.74429 | 0.76472 |
| STS2016.input.headlines.txt | 0.76078 | 0.79535 | 0.93481 | 0.41351 | 0.57937 | 0.84362 | 0.93481 |
| STS2016.input.plagiarism.txt | 0.83741 | 0.84309 | 0.84103 | 0.78026 | 0.66033 | 0.83397 | 0.84103 |
| STS2016.input.postediting.txt | 0.85083 | 0.84535 | 0.85077 | 0.73988 | 0.75669 | 0.85523 | 0.85523 |
| STS2016.input.question-question.txt | 0.74987 | 0.72033 | 0.74247 | 0.05413 | 0.57318 | 0.82987 | 0.82987 |