Linguistic Features NBME Items NERA

September 25, 2020

1 Tutorial on Linguistic Feature Extraction from Item Text

1.1 NERA 2020

This tutorial presents Python code for the extraction of linguistic features from item text. The code can be used for the extraction of features at several linguistic levels including: 1. Lexical features 2. Syntactic features 3. Features of cohesion 4. Readability features

For illustration purposes the data used in this tutorial consists of 10 freely available items from the United States Licensing Examination (USMLE) training materials available at: https://www.usmle.org/pdfs/step-1/samples_step1_2020.pdf (Items 1 - 10)

2 System prerequisites

Python 3 (Current Python version: 3.8.3; IDE: Jupyter Notebook)

Spacy Version 2.3.1

Spacy model for extracting data for English: “en_core_web_sm” (Note: compatible with Spacy 2.3.1 or earlier versions)

3 Load the necessary packages

[338]:
```
import pandas as pd
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
from collections import Counter
import re
import spacy
import en_core_web_sm
import glob
```

4 Load the data

The data format for this code is one where every item (or document) is a separate .txt file in a designated folder. The code in the following cells takes each item and assignes it to a cell in a dataframe column, allowing the extraction of data for the whole item. An alternative approach would be to extract data for the stem and each distractor separately, where the data needs to be loaded with a separator for the different item parts. The feature extraction functions presented later can then be applied to each column, as needed.

[339]:
```
#Code to collect all .txt files from a given folder and assign them to a list␣
↪called "corpus"

#Initiate an empty list to store the items
corpus = []

# substitute the path from this command with the path to the folder where your .
↪txt items are contained.
file_list = glob.glob(r"I:\Measurement Consulting\Victoria\NERA Workshop/*.txt")

#Assign each .txt file in the folder to the corpus list
for file_path in file_list:
    with open(file_path, 'r') as file_input:
        corpus.append(file_input.read())
```

[340]:
```
#We can check different elemnts in the corpus list, where each element is an item
corpus[1]
```

[340]:
```
"A 67-year-old woman with congenital bicuspid aortic valve is admitted to the
hospital because of a 2-day history of fever and chills. Current medication is
lisinopril. Temperature is 38.0°C (100.4°F), pulse is 90/min, respirations are
20/min, and blood pressure is 110/70 mm Hg. Cardiac examination shows a grade
3/6 systolic murmur that is best heard over the second right intercostal space.
Blood culture grows viridans streptococci susceptible to penicillin. In addition
to penicillin, an antibiotic synergistic to penicillin is administered that may
help shorten the duration of this patient's drug treatment. Which of the
following is the most likely mechanism of action of this additional antibiotic
on bacteria? (A) Binding to DNA-dependent RNA polymerase (B) Binding to the 30S
ribosomal protein (C) Competition with p-aminobenzoic acid (D) Inhibition of
dihydrofolate reductase (E) Inhibition of DNA gyrase"
```

[341]:
```
#Create a dataframe, where each item is a row in the column "Raw Item"
df = pd.DataFrame({"Raw Item": corpus})

#See the first five elements of the dataframe
df.head()
```

```
[341]:                                             Raw Item
       0  A 22-year-old woman comes to the office becaus...
       1  A 67-year-old woman with congenital bicuspid a...
       2  A 12-year-old girl is brought to the physician...
       3  During an experiment, drug X is added to a mus...
       4  A 30-year-old woman, gravida 2, para 0, aborta...
```

```
[342]: #Ckech the shape of the dataframe (rows and columns)
       df.shape
```

```
[342]: (10, 1)
```

```
[343]: #Convert the "Raw Item" variable to a string variable (usually it is initially␣
        ↪recognized as an object-type variable)
       df['Raw Item'] = pd.Series(df['Raw Item'], dtype= "string")
```

```
[344]: #Make sure that the "Raw Item" variable is a string variable
       df.dtypes
```

```
[344]: Raw Item    string
       dtype: object
```

```
[345]: #Check that we are working with a pandas dataframe and not a pandas series
       type(df)
```

```
[345]: pandas.core.frame.DataFrame
```

5 Data Preprocessing

Once we have loaded the data, it is important to preprocess it in a way that makes it machine-readable and useful for feature extraction. This may involve different steps depending on the application of interest. In this case, we perform the following preprocessing steps:

1. Convert all words to lowercase. This is important for matching certain phrases later on, where the matching is case-sensitive and the word or phrase of interest would not be matched to a capitalized version of the same word or phrase.
2. Tokenize the data. This step refers to identifying the boundaries of individual words and sentences within each item.
3. Lemmatization. This process converts each word to its base form (e.g. 'has' to 'have', 'are' to 'be') for the purpose of easier matching.
4. Removal of words that contain non-alphabetic characters (e.g., numbers, '#', '@', etc.)
5. Stopword removal. This step refers to the removal of words that may not be important for the analysis (e.g. 'a', 'an', 'the', etc.). One could use a predefined list of stopwords (usually these remove all forms of the verb 'to be', as well as pronouns) or create a customized list of stopwords depending on what is important for the specific application.

[346]:
```
#Use one of the following two commands to load the English-language package of␣
↪Spacy and load it into an nlp variable

#nlp = spacy.load('en_core_web_sm')
nlp = en_core_web_sm.load()
```

[347]:
```
#Convert all words in the Raw Item to lower case.
df['Raw Item'] = df['Raw Item'].str.lower()

#Check the result in the first five rows
df.head()
```

[347]:
```
                                            Raw Item
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...
```

[348]:
```
# Create a function to preprocess the text

#Customized list of stopwords
stopwords = ['a', 'an', 'the', 'with', 'to', 'be', 'have']

def preprocess(text):
    '''This is a function to perform tokenization, lemmatization, removal of␣
↪non-alphabetic characters
    and stopword removal'''
          # Create Doc object
    doc = nlp(text, disable=['ner'])
    # Generate lemmas
    lemmas = [token.lemma_ for token in doc]
    # Remove stopwords and non-alphabetic characters
    a_lemmas = [lemma for lemma in lemmas
            if lemma.isalpha() and lemma not in stopwords]
    return ' '.join(a_lemmas)
```

[349]:
```
#The below command creates a new column in our dataframe called "Item" which␣
↪will store the preprocessed version
# of the items. Thus, we have two versions of each item; 'Raw Item' contains the␣
↪full text without lemmatization
#and stopword removal, while "Item" contains the preprocessed data.

#Apply the preprocess function to the items and store the result in a new column
df['Item'] = df['Raw Item'].apply(preprocess)

df.head()
```

[349]:
```
                                            Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                                Item
0  old woman come office because of history of it...
1  old woman congenital bicuspid aortic valve adm...
2  old girl bring physician because of history of...
3  during experiment drug x add muscle bath conta...
4  old woman gravida para aborta at week gestatio...
```

6 Lexical Features

This section presents functions which allow the extraction of lexical features, i.e. features at the level of individual words.

First, the extraction of each feature type is defined as a function. The function is then applied to the item text and the output is stored in a new variable. Depending on the type of information needed, the function may be applied to the raw text (“Raw Item) or to the preprocessed text (”Item"). For example, applying a word-count function to the raw text will return the full word count, while applying it to the preprocessed text will return the number of content words, after stopwords have been removed.

[350]:
```python
def count_words(string):
    '''This function returns the number of words in a string'''
    # Split the string into words
    words = string.split()
    # Return the number of words
    return len(words)

#Application to the raw data to get the full word count
df['Word_Count'] = df['Raw Item'].apply(count_words)

#Application to the preprocessed data to get the content-word count
df['Word_Count_No_stop_words'] = df['Item'].apply(count_words)
df.head()
```

[350]:
```
                                            Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...
```

```
                                                Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words
0                       129
1                       106
2                        84
3                        72
4                       106
```

[351]:
```
def word_length(string):
    '''This function returns the average word length in characters for the words␣
 ↪in an item'''
    #Get the length of the full text in characters
    chars = len(string)
    #Split the string into words
    words = string.split()
    #Compute the average word length and round the output to the second decimal␣
 ↪point
    avg_word_length = chars/len(words)
    return round(avg_word_length, 2)

#Application to the preprocessed data
df['Avg_Word_Length'] = df['Item'].apply(word_length)
df.head()
```

[351]:
```
                                            Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                                Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words  Avg_Word_Length
0                       129             5.82
1                       106             6.93
```

```
2                         84            6.87
3                         72            6.38
4                        106            6.62
```

[352]:
```
#The following lexical functions refer to counting different parts of speech␣
↪(POS) in the items

def nouns(text, model=nlp):
    '''This function returns the number of nouns in an item'''
    # Create doc object
    doc = model(text)
    # Generate list of POS tags
    pos = [token.pos_ for token in doc]
    # Return number of nouns
    return pos.count('NOUN')

df['Noun_Count'] = df['Item'].apply(nouns)
df.head()
```

[352]:
```
                                           Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                               Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words  Avg_Word_Length  Noun_Count
0                       129             5.82          32
1                       106             6.93          39
2                        84             6.87          32
3                        72             6.38          36
4                       106             6.62          41
```

[353]:
```
def verbs(text, model=nlp):
    '''This function returns the number of verbs in an item'''
    # Create doc object
    doc = model(text)
    # Generate list of POS tags
    pos = [token.pos_ for token in doc]
    # Return number of verbs
    return pos.count('VERB')

df['Verb_Count'] = df['Item'].apply(verbs)
df.head()
```

[353]:
```
                                            Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                                Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words  Avg_Word_Length  Noun_Count  Verb_Count
0                       129             5.82          32           7
1                       106             6.93          39           8
2                        84             6.87          32           5
3                        72             6.38          36           2
4                       106             6.62          41           6
```

[354]:
```
def adjectives(text, model=nlp):
    '''This function returns the number of adjectives in an item'''
    # Create doc object
    doc = model(text)
    # Generate list of POS tags
    pos = [token.pos_ for token in doc]
    # Return number of adjectives
    return pos.count('ADJ')

df['Adjective_Count'] = df['Item'].apply(adjectives)
#df.head()
```

[355]:
```
def adverbs(text, model=nlp):
    '''This function returns the number of adverbs in an item'''
    # Create doc object
    doc = model(text)
    # Generate list of POS tags
    pos = [token.pos_ for token in doc]
    # Return number of adverbs
    return pos.count('ADV')
```

```
df['Adverb_Count'] = df['Item'].apply(adverbs)
df.head()
```

[355]:
```
                                             Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                                 Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words  Avg_Word_Length  Noun_Count  Verb_Count  \
0                       129             5.82          32           7
1                       106             6.93          39           8
2                        84             6.87          32           5
3                        72             6.38          36           2
4                       106             6.62          41           6

   Adjective_Count  Adverb_Count
0               16             2
1               13             2
2                5             3
3                6             2
4               12             2
```

Other types of POS can be extracted using a similar function by changing the "return pos.count('ADV')" command, where the tag for the POS of interest is entered instead of 'ADV'. The full list of POS tags in Spacy is available here: https://spacy.io/api/annotation

7 Syntactic Features

This section describes the extraction of features at the syntactic level

[356]:
```
def sentence_counter(text):
    '''This function returns the number of sentences in an item'''
    doc = nlp(text)
    #Initialize a counter variable
    counter = 0
    #Update the counter for each sentence which can be found in the doc.sents␣
 ↪object returned by the Spacy model
    for sentence in doc.sents:
        counter = counter + 1
    return counter

#Note that this function is applied to the raw text in order to identify␣
 ↪sentence boundaries
df['Sentence_Count'] = df['Raw Item'].apply(sentence_counter)
#df.head()
```

```
[357]: def avg_sent_length(text):
           '''This function returns the average sentence length in an item'''
           doc = nlp(text)
           #Initialize a counter variable
           sent_number = 0
           #Update the counter for each sentence which can be found in the doc.sents␣
        ↪object returned by the Spacy model
           for sent in doc.sents:
               sent_number = sent_number + 1
           #Get the number of words
           words = text.split()
           #Compute the average sentence length and round it to the second decimal point
           avg_sent_length = len(words)/sent_number
           return round(avg_sent_length, 2)

       #Note that this function is applied to the raw text in order to identify␣
        ↪sentence boundaries
       df['Avg_Sent_Length_in_Words'] = df['Raw Item'].apply(avg_sent_length)
       #df.head()
```

The functions that follow refer to the extraction of specific types of phrases within the text such as noun phrases, prepositional phrases, and verb phrases. Before we introduce the function that extract this data, let's illustrate the way the identification of these phrases works

```
[358]: #Let's take an example of a string to use for the illustration
       doc = nlp("A 12-year-old girl is brought to the physician because of a 2-month␣
        ↪history of intermittent yellowing of the eyes")
```

```
[359]: #We use the doc.noun_chunks object returned by the Spacy model to count he␣
        ↪number of noun phrases (NPs)
       # in the string, i.e. phrases whose head is a noun
       for np in doc.noun_chunks:
           print(np.text)
```

```
A 12-year-old girl
the physician
a 2-month history
intermittent yellowing
the eyes
```

[360]:
```
#Next, we take the same string but we count the number of prepositional phrases␣
↪(PPs)

#Initialize a list to store the PPs
pps = []

#For each token in the doc object, take the ones that start with a the 'ADP' tag␣
↪(the tag stands for
# 'conjunction, subordinating or preposition')
for token in doc:
    if token.pos_ == 'ADP': #conjunction, subordinating or preposition
        #identify the phrases in the syntactic subtree that start with this tag
        pp = ' '.join([tok.orth_ for tok in token.subtree])
        pps.append(pp)

print(pps)
```

```
['to the physician', 'of', 'of intermittent yellowing of the eyes', 'of the
eyes']
```

[361]:
```
#To extract verb phrases, we need to define a verb phrase (VP) pattern and match␣
↪the pattern to the text

#Import a package to match the patterns
from spacy.matcher import Matcher
#Import a package to define the span within which to look for a match
from spacy.util import filter_spans

#Create the VP patterns to match phrases starting with verbs, adverbs, or␣
↪auxiliary verbs
pattern = [{'POS': 'VERB', 'OP': '?'},
           {'POS': 'ADV', 'OP': '*'},
           {'POS': 'AUX', 'OP': '*'},
           {'POS': 'VERB', 'OP': '+'}]

# instantiate a Matcher instance
matcher = Matcher(nlp.vocab)
matcher.add("Verb phrase", None, pattern)

# call the matcher to find matches
matches = matcher(doc)
#define the span as the start and end of an input
spans = [doc[start:end] for _, start, end in matches]

print (filter_spans(spans))
```

```
[is brought]
```

Now that we have reviewed the examples of how the phrase identification methods works, let's wrap these in functions

[362]:
```
def get_nps(text):
    '''This is a function that outputs the number of noun phrases in an item'''
    doc = nlp(text)
    NP_count = 0
    for np in doc.noun_chunks:
        NP_count = NP_count + 1
    return NP_count

df['Number_of_NPs'] = df['Item'].apply(get_nps)
#df.head()
```

[363]:
```
def get_pps(text):
    '''This is a function that outputs the number of prepositional phrases in an␣
 ↪item'''
    doc = nlp(text)
    pps = 0
    for token in doc:
        # You can try this with other parts of speech for different subtrees.
        if token.pos_ == 'ADP':

            #Use the command below if you wanted to get the actual PPs
            #pp = ' '.join([tok.orth_ for tok in token.subtree])

            #This command counts the number of PPs
            pps = pps + 1
    return pps

df['Number_of_PPs'] = df['Item'].apply(get_pps)
#df.head()
```

[364]:
```
#We can modify this pattern for the extraction of verb phrases
pattern = [{'POS': 'VERB', 'OP': '?'},
           {'POS': 'ADV', 'OP': '*'},
           {'POS': 'AUX', 'OP': '*'},
           {'POS': 'VERB', 'OP': '+'}]


def get_vps(text):
    '''This function returns the number of verb phrases in an item'''
    doc = nlp(text)
    vps = 0
    # instantiate a Matcher instance
    matcher = Matcher(nlp.vocab)
    matcher.add("Verb phrase", None, pattern)
    # call the matcher to find matches
    matches = matcher(doc)
    spans = [doc[start:end] for _, start, end in matches]
    for match in matches:
        vps = vps +1
    return vps

df['Number_of_VPs'] = df['Item'].apply(get_vps)
df.head()
```

```
[364]:                                          Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                            Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words  Avg_Word_Length  Noun_Count  Verb_Count  \
0                       129             5.82          32           7
1                       106             6.93          39           8
2                        84             6.87          32           5
3                        72             6.38          36           2
4                       106             6.62          41           6

   Adjective_Count  Adverb_Count  Sentence_Count  Avg_Sent_Length_in_Words  \
0               16             2              19                      9.58
1               13             2               9                     15.11
2                5             3              11                      9.36
3                6             2              15                      6.40
4               12             2              12                     11.67

   Number_of_NPs  Number_of_PPs  Number_of_VPs
0             31             25              9
1             20             11             12
2             19             10              5
3             16              9              2
4             28             18              8
```

8 Features of Cohesion

This section presents the extraction of features related to text cohesion. Cohesion generally refers to the presence or absence of explicit cues in the text that allow the reader to make connections between the ideas in the text (Crossley, Kyle and McNamara, 2016). Thus, features of cohesion are useful in eassay scoring and readability research.

A useful resource for further reading on cohesion can be found here: https://link.springer.com/article/10.3758/s13428-015-0651-7

[Crossley, S.A., Kyle, K. & McNamara, D.S. The tool for the automatic analysis of text cohesion (TAACO): Automatic assessment of local, global, and text cohesion. Behav Res 48, 1227–1237 (2016)]

One type of cohesion features are those related to different types of connectives. Connectives are measures of local cohesion and signify the way ideas in a text are organized, as can be seen from the connectives lists in the next cell.

[365]:
```
#First, we create lists of different types of connectives that we will later␣
↪match to the text

#Connectives to instruct, recount and sequence
temporal_connectives = ['afterwards', 'once', 'at this moment', 'at this point',␣
↪'before', 'finally',
                        'here', 'in the end', 'lastly', 'later on', 'meanwhile',␣
↪'next', 'now',
                        'on another occasion', 'previously','since', 'soon',␣
↪'straightaway', 'then',
                        'when', 'whenever', 'while']

#Connectivaes preceding an explanation
explanative_connectives = ['besides', 'e.g.', 'for example', 'for instance', 'i.
↪e.', 'in other words',
                          'in that', 'that is to say']


#Connectives to show cause or conditions
causal_connectives = ['accordingly', 'all the same', 'an effect of', 'an outcome␣
↪of', 'an upshot of',
                      'as a consequence of', 'as a result of', 'because',␣
↪'caused by', 'consequently',
                      'despite this', 'even though', 'hence', 'however', 'in␣
↪that case', 'moreover',
                      'nevertheless', 'otherwise', 'so', 'so as', 'stemmed␣
↪from', 'still', 'then',
                      'therefore', 'though', 'under the circumstances', 'yet']


#Connectives for showing results
```

```
exemplifying_connectives = ['accordingly', 'as a result', 'as exemplified by',␣
 ↪'consequently', 'for example',
                             'for instance', 'for one thing', 'including',␣
 ↪'provided that', 'since', 'so',
                             'such as', 'then', 'therefore', 'these include',␣
 ↪'through', 'unless', 'without']


#Connectives to show similarity or add a point
additive_connectives = ['and', 'additionally', 'also', 'as well', 'even',␣
 ↪'furthermore', 'in addition', 'indeed',
                         'let alone', 'moreover', 'not only']

#Connectives showing a difference or an opposite point of view
contrastive_connectives = ['alternatively', 'anyway', 'but', 'by contrast',␣
 ↪'differs from', 'elsewhere',
                            'even so', 'however', 'in contrast', 'in fact', 'in␣
 ↪other respects', 'in spite of this',
                            'in that respect', 'instead', 'nevertheless', 'on the␣
 ↪contrary', 'on the other hand',
                            'rather', 'though', 'whereas', 'yet']
```

Similar to the illustration on phrase extraction, let's first see what the matching of connectives produces in the following examples:

[366]:
```
#Consider the following example string:
test_string = "once, while I walked in a park, I saw an old man. He first sat on␣
 ↪a bench and then opened a book"

#Now let's see how many temporal connectives we have in the string:
temps = []
for string in temporal_connectives:
    if string in test_string:
        temps.append(string)
print(temps)
```

```
['once', 'then', 'while']
```

[367]:
```
#Consider another example string"

test_string_2 = 'for instance, one might choose to participate in a trial. in␣
 ↪other words, that is to say that the person is able to make their own␣
 ↪decisions.'

#Let's see how many explanative connectives the string contains
explan = []
for string in explanative_connectives:
    if string in test_string_2:
        explan.append(string)
print(explan)
```

```
['for instance', 'in other words', 'that is to say']
```

[368]:
```
def temporal_connectives_count(text):
    '''This function counts the number of temporal connectives in a text'''
    count = 0
    for string in temporal_connectives:
        for match in re.finditer(string, text):
            count +=  1
    return count

#Note that we apply the function to the raw text (and remember that it is␣
 ↪important to lowercase all words)
df['Temporal_Connectives_Count'] = df['Raw Item'].
 ↪apply(temporal_connectives_count)
#df.head()
```

[369]:
```
def explanative_connectives_count(text):
    '''This function counts the number of explanative connectives in a text'''
    count = 0
    for string in explanative_connectives:
        for match in re.finditer(string, text):
            count +=  1
    return count

df['Explanative_Connectives_Count'] = df['Raw Item'].
 ↪apply(explanative_connectives_count)
#df.head()
```

[370]:
```
def causal_connectives_count(text):
    '''This function counts the number of causal connectives in a text'''
    count = 0
    for string in causal_connectives:
        for match in re.finditer(string, text):
            count +=  1
    return count

df['Causal_Connectives_Count'] = df['Raw Item'].apply(causal_connectives_count)
#df.head()
```

[371]:
```
def exemplifying_connectives_count(text):
    '''This function counts the number of exemplifying connectives in a text'''
    count = 0
    for string in exemplifying_connectives:
        for match in re.finditer(string, text):
            count +=  1
    return count

df['Exemplifying_Connectives_Count'] = df['Raw Item'].
 ↪apply(exemplifying_connectives_count)
#df.head()
```

[372]:
```
def additive_connectives_count(text):
    '''This function counts the number of additive connectives in a text'''
    count = 0
    for string in additive_connectives:
        for match in re.finditer(string, text):
            count +=  1
    return count

df['Additive_Connectives_Count'] = df['Raw Item'].
 ↪apply(additive_connectives_count)
#df.head()
```

[373]:
```
def contrastive_connectives_count(text):
    '''This function counts the number of contrastive connectives in a text'''
    cont_con = 0
    for string in contrastive_connectives:
        if string in text:
            cont_con = cont_con + 1
    return cont_con

df['Contrastive_Connectives_Count'] = df['Raw Item'].
 ↪apply(contrastive_connectives_count)
df.head()
```

[373]:
```
                                            Raw Item  \
0  a 22-year-old woman comes to the office becaus...
1  a 67-year-old woman with congenital bicuspid a...
2  a 12-year-old girl is brought to the physician...
3  during an experiment, drug x is added to a mus...
4  a 30-year-old woman, gravida 2, para 0, aborta...

                                                Item  Word_Count  \
0  old woman come office because of history of it...         182
1  old woman congenital bicuspid aortic valve adm...         136
2  old girl bring physician because of history of...         103
3  during experiment drug x add muscle bath conta...          96
4  old woman gravida para aborta at week gestatio...         140

   Word_Count_No_stop_words  Avg_Word_Length  Noun_Count  Verb_Count  \
0                     129          5.82         32           7
1                     106          6.93         39           8
2                      84          6.87         32           5
3                      72          6.38         36           2
4                     106          6.62         41           6

   Adjective_Count  Adverb_Count  Sentence_Count  Avg_Sent_Length_in_Words  \
0               16             2              19                      9.58
1               13             2               9                     15.11
2                5             3              11                      9.36
3                6             2              15                      6.40
4               12             2              12                     11.67

   Number_of_NPs  Number_of_PPs  Number_of_VPs  Temporal_Connectives_Count  \
0             31             25              9                           0
1             20             11             12                           0
2             19             10              5                           3
3             16              9              2                           1
4             28             18              8                           0

   Explanative_Connectives_Count  Causal_Connectives_Count  \
0                             11                         2
1                              3                         2
2                              7                         1
3                             10                         0
4                              6                         2

   Exemplifying_Connectives_Count  Additive_Connectives_Count  \
0                               1                           4
1                               1                           3
2                               0                           3
3                               0                           1
4                               1                           1

   Contrastive_Connectives_Count
0                              0
1                              0
2                              0
3                              0
4                              0
```

[374]:
```
df.shape
```

[374]:
```
(10, 20)
```

9 Readability Metrics

Readability formulae are a popular and effective method to match the reading difficulty of a text to a US school grade level or some other difficulty criterion. Useful information about different readability formulae can be found in William DuBay's monograph entitled "The Priciples of Readability": http://www.impact-information.com/impactinfo/readability02.pdf

An important consideration when applying readability formulae is the fact that they should only be applied to texts with certain characteristics, usually ones longer than 100 words or, in some cases such as the SMOG formula, the text being assessed has to have at least 15 sentences. Therefore, an important first step for the illustration on how these formulae can be derived automatically would be to select those items from our dataframe that are longer than 100 words.

[375]:
```
#Select raw items longer than 100 words and assign them to a new dataframe␣
↪entitled df_long items
long_items = []
for item in df['Raw Item']:
    if len(item.split())>= 100:
        long_items.append(item)
df_long_items = pd.DataFrame({"Long Items": long_items})

#Check the shape of the new dataframe (rows and columns)
df_long_items.shape
```

[375]:
```
(5, 1)
```

[376]:
```
from readability import Readability

# The Flesch-Kincaid Grade Level formula is computed as follows:
#0.39 x(total words/total sentences) + 11.8 x(total syllables/total words)- 15.59

def FK(text):
    '''This is a function that computes the Flesch-Kincaid Grade Level of a␣
↪text'''
    r = Readability(text)
    fk = r.flesch_kincaid()
    return fk.grade_level
    #or fk.score

#Apply the function to the new dataframe and check the output
df_long_items['FK_Grade_Level'] = df_long_items['Long Items'].apply(FK)
df_long_items.head()
```

[376]:
```
                                          Long Items FK_Grade_Level
0  a 22-year-old woman comes to the office becaus...              7
1  a 67-year-old woman with congenital bicuspid a...             13
2  a 12-year-old girl is brought to the physician...             13
3  a 30-year-old woman, gravida 2, para 0, aborta...             11
```

```
4  after being severely beaten and sustaining a g...            13
```

[377]:
```
#Flesch Reading Ease = 206.835 - (1.015 x Average sentence length) - (84.6 x␣
 ↪Average number of syllables per word)

#90-100: Very Easy
#80-89 : Easy
#70-79 : Fairly Easy
#60-69 : Standard
#50-59 : Fairly Difficult
#30-49 : Difficult
#0-29 : Very Confusing

def Flesch(text):
    '''This is a function that computes the Flesch Score of a text'''
    r = Readability(text)
    f = r.flesch()
    return f.ease

#The code can be augmented to output a score (f.score) or grade level(f.
 ↪grade_levels) instead of reading ease

#Apply the function to the new dataframe and check the output
df_long_items['Flesch_Reading_Ease'] = df_long_items['Long Items'].apply(Flesch)
df_long_items.head()
```

[377]:
```
                                          Long Items FK_Grade_Level  \
0  a 22-year-old woman comes to the office becaus...              7
1  a 67-year-old woman with congenital bicuspid a...             13
2  a 12-year-old girl is brought to the physician...             13
3  a 30-year-old woman, gravida 2, para 0, aborta...             11
4  after being severely beaten and sustaining a g...             13

  Flesch_Reading_Ease
0            standard
1           difficult
2      very_confusing
3           difficult
4           difficult
```

[378]:
```
#The Dale-Chall Formula is based on the use of familiar words, rather than␣
 ↪syllable or letter counts.

def dale_chall(text):
    '''This is a function that computes the Dale-Chall Score of a text'''
    r = Readability(text)
    dc = r.dale_chall()
    return dc.score
    # or dc.grade_levels

#Apply the function to the new dataframe and check the output
df_long_items['Dale_Chall_score'] = df_long_items['Long Items'].apply(dale_chall)
df_long_items.head()
```

[378]:

```
                                          Long Items FK_Grade_Level  \
0  a 22-year-old woman comes to the office becaus...              7
1  a 67-year-old woman with congenital bicuspid a...             13
2  a 12-year-old girl is brought to the physician...             13
3  a 30-year-old woman, gravida 2, para 0, aborta...             11
4  after being severely beaten and sustaining a g...             13

  Flesch_Reading_Ease  Dale_Chall_score
0            standard         12.501852
1           difficult         12.702741
2      very_confusing         12.150553
3           difficult         11.807129
4           difficult         10.574742
```

[379]:

```
#ARI relies on a factor of characters per word, instead of the usual syllables␣
 ↪per word.

def ARI(text):
    '''This is a function that computes the Army Readability Index (ARI) of a␣
 ↪text'''
    r = Readability(text)
    ari = r.ari()
    return ari.score
    # or ari.grade_levels
    # or ari.ages

#Apply the function to the new dataframe and check the output
df_long_items['ARI_Score'] = df_long_items['Long Items'].apply(ARI)
df_long_items.head()
```

[379]:

```
                                          Long Items FK_Grade_Level  \
0  a 22-year-old woman comes to the office becaus...              7
1  a 67-year-old woman with congenital bicuspid a...             13
2  a 12-year-old girl is brought to the physician...             13
3  a 30-year-old woman, gravida 2, para 0, aborta...             11
4  after being severely beaten and sustaining a g...             13

  Flesch_Reading_Ease  Dale_Chall_score  ARI_Score
0            standard         12.501852   5.000680
1           difficult         12.702741  12.124500
```

```
2      very_confusing         12.150553  11.099008
3           difficult         11.807129   9.221875
4           difficult         10.574742  12.485576
```

[380]:
```
#The Gunning fog index estimates the years of formal education needed to␣
↪understand the text on a first reading.
#A fog index of 12 requires the reading level of a U.S. high school senior␣
↪(around 18 years old).

def Fog(text):
    '''This is a function that computes the Gunning Fog Index of a text'''
    r = Readability(text)
    fog = r.gunning_fog()
    return fog.score
    #or fog.grade_level

#Apply the function to the new dataframe and check the output
df_long_items['Gunning_Fog_Score'] = df_long_items['Long Items'].apply(Fog)
df_long_items.head()
```

[380]:
```
                                          Long Items FK_Grade_Level  \
0  a 22-year-old woman comes to the office becaus...              7
1  a 67-year-old woman with congenital bicuspid a...             13
2  a 12-year-old girl is brought to the physician...             13
3  a 30-year-old woman, gravida 2, para 0, aborta...             11
4  after being severely beaten and sustaining a g...             13

  Flesch_Reading_Ease  Dale_Chall_score  ARI_Score  Gunning_Fog_Score
0            standard         12.501852   5.000680          12.369524
1           difficult         12.702741  12.124500          17.181034
2      very_confusing         12.150553  11.099008          17.652910
3           difficult         11.807129   9.221875          15.204444
4           difficult         10.574742  12.485576          16.545455
```

10 Saving the Output

The output of the dataframes containing the extracted features can be exported to any desired folder

[381]:
```
df_export = df.to_csv('Lexical, Syntactic, and Cohesion Features Output.csv')
df_long_items_export = df_long_items.to_csv('Readability Features Output.csv')
```