Data Pre-processing II (Text Data)

Applied Machine Learning for Educational Data Science

true

09/08/2021

Contents

1. Textual statistics via quanteda **2**

1.1. Tokenization 2

1.2. Basic statistics 5

1.3. Word length statistics 6

1.4. Text Entropy 7

1.5. Lexical diversity 9

1.6. Measures of readability 10

2. Lemmatization, Parts of Speech Tagging, and Dependency Parsing via udpipe **10**

2.1. Morphological annotation 11

2.2. Part of Speech Tags (POS) 11

2.3. Features 12

2.4. Syntactic relations 14

3. Natural Language Processing (NLP) **15**

3.1. Loading the required packages 16

3.2. Word Embeddings 16

3.3. Sentence Embeddings 19

4. Wrapping-up **21**

[Updated: Thu, Oct 14, 2021 - 10:14:59]

Generating features from text data is very different than dealing with continuous and categorical data. First, let's remember the dataset we are working with.

```
readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/ma
                        header=TRUE)

str(readability)
```

```
'data.frame':   2834 obs. of  6 variables:
 $ id             : chr  "c12129c31" "85aa80a4c" "b69ac6792" "dd1000b26" ...
 $ url_legal      : chr  "" "" "" "" ...
 $ license        : chr  "" "" "" "" ...
 $ excerpt        : chr  "When the young people returned to the ballroom, it presented a decidedly change
 $ target         : num  -0.34 -0.315 -0.58 -1.054 0.247 ...
 $ standard_error: num  0.464 0.481 0.477 0.45 0.511 ...
```

```r
readability[1,]$excerpt
```

```
[1] "When the young people returned to the ballroom, it presented a decidedly changed appearance. Instea
```

```r
readability[1,]$target
```

```
[1] -0.3402591
```

The excerpt column includes a plain text data and the target column includes a corresponding measure of readability for each excerpt. A higher target value indicates a more difficult text to read. What kind of features we can generate from the plain text to predict its readability?

In the following sections, we will demonstrate how to derive these features for a single observation. At the end, we will finish with some code to compile all these information in a useful tabular format to be able to use in predictive modeling later.

1. Textual statistics via `quanteda`

For this section, we will rely on two packages: `quanteda`, `quanteda.textmodels`.

```r
require(quanteda)
require(quanteda.textstats)
```

1.1. Tokenization

The first thing to do is tokenization of the plain text using the `tokens()` function from the `quanteda` package. This will create an object that contains every word, punctuations, symbols, numbers, etc.

```r
text <- as.character(readability[1,]$excerpt)

text
```

```
[1] "When the young people returned to the ballroom, it presented a decidedly changed appearance. Instea
```

```r
tokenized <- tokens(text)

tokenized[[1]]
```

```
  [1] "When"        "the"         "young"       "people"      "returned"
  [6] "to"          "the"         "ballroom"    ","           "it"
 [11] "presented"   "a"           "decidedly"   "changed"     "appearance"
 [16] "."            "Instead"    "of"         "an"         "interior"
 [21] "scene"        ","          "it"         "was"        "a"
 [26] "winter"       "landscape"  "."          "The"        "floor"
 [31] "was"          "covered"    "with"       "snow-white" "canvas"
 [36] ","            "not"        "laid"       "on"         "smoothly"
 [41] ","            "but"        "rumpled"    "over"       "bumps"
 [46] "and"          "hillocks"   ","          "like"       "a"
 [51] "real"         "snow"       "field"      "."          "The"
 [56] "numerous"     "palms"      "and"        "evergreens" "that"
 [61] "had"          "decorated"  "the"        "room"       ","
 [66] "were"         "powdered"   "with"       "flour"      "and"
 [71] "strewn"       "with"       "tufts"      "of"         "cotton"
 [76] ","            "like"       "snow"       "."          "Also"
 [81] "diamond"      "dust"       "had"        "been"       "lightly"
 [86] "sprinkled"    "on"         "them"       ","          "and"
 [91] "glittering"   "crystal"    "icicles"    "hung"       "from"
 [96] "the"          "branches"   "."          "At"         "each"
[101] "end"          "of"         "the"        "room"       ","
[106] "on"           "the"        "wall"       ","          "hung"
[111] "a"            "beautiful"  "bear-skin"  "rug"        "."
[116] "These"        "rugs"       "were"       "for"        "prizes"
[121] ","            "one"        "for"        "the"        "girls"
[126] "and"          "one"        "for"        "the"        "boys"
[131] "."            "And"        "this"       "was"        "the"
[136] "game"         "."          "The"        "girls"      "were"
[141] "gathered"     "at"         "one"        "end"        "of"
[146] "the"          "room"       "and"        "the"        "boys"
[151] "at"           "the"        "other"      ","          "and"
[156] "one"          "end"        "was"        "called"     "the"
[161] "North"        "Pole"       ","          "and"        "the"
[166] "other"        "the"        "South"      "Pole"       "."
[171] "Each"         "player"     "was"        "given"      "a"
[176] "small"        "flag"       "which"      "they"       "were"
[181] "to"           "plant"      "on"         "reaching"   "the"
[186] "Pole"         "."          "This"       "would"      "have"
[191] "been"         "an"         "easy"       "matter"     ","
[196] "but"          "each"       "traveller"  "was"        "obliged"
[201] "to"           "wear"       "snowshoes"  "."
```

We can also create a separate tokenization that includes only words.

```r
tokenized.words <- tokens(text,
                          remove_punct = TRUE,
                          remove_numbers = TRUE,
                          remove_symbols = TRUE,
                          remove_separators = TRUE)

tokenized.words[[1]]
```

```
  [1] "When"         "the"        "young"      "people"     "returned"
  [6] "to"           "the"        "ballroom"   "it"         "presented"
 [11] "a"            "decidedly"  "changed"    "appearance" "Instead"
 [16] "of"           "an"         "interior"   "scene"      "it"
 [21] "was"        "a"          "winter"     "landscape"  "The"       
 [26] "floor"      "was"        "covered"    "with"       "snow-white"
 [31] "canvas"     "not"        "laid"       "on"         "smoothly"  
 [36] "but"        "rumpled"    "over"       "bumps"      "and"       
 [41] "hillocks"   "like"       "a"          "real"       "snow"      
 [46] "field"      "The"        "numerous"   "palms"      "and"       
 [51] "evergreens" "that"       "had"        "decorated"  "the"       
 [56] "room"       "were"       "powdered"   "with"       "flour"     
 [61] "and"        "strewn"     "with"       "tufts"      "of"        
 [66] "cotton"     "like"       "snow"       "Also"       "diamond"   
 [71] "dust"       "had"        "been"       "lightly"    "sprinkled" 
 [76] "on"         "them"       "and"        "glittering" "crystal"   
 [81] "icicles"    "hung"       "from"       "the"        "branches"  
 [86] "At"         "each"       "end"        "of"         "the"       
 [91] "room"       "on"         "the"        "wall"       "hung"      
 [96] "a"          "beautiful"  "bear-skin"  "rug"        "These"     
[101] "rugs"       "were"       "for"        "prizes"     "one"       
[106] "for"        "the"        "girls"      "and"        "one"       
[111] "for"        "the"        "boys"       "And"        "this"      
[116] "was"        "the"        "game"       "The"        "girls"     
[121] "were"       "gathered"   "at"         "one"        "end"       
[126] "of"         "the"        "room"       "and"        "the"       
[131] "boys"       "at"         "the"        "other"      "and"       
[136] "one"        "end"        "was"        "called"     "the"       
[141] "North"      "Pole"       "and"        "the"        "other"     
[146] "the"        "South"      "Pole"       "Each"       "player"    
[151] "was"        "given"      "a"          "small"      "flag"      
[156] "which"      "they"       "were"       "to"         "plant"     
[161] "on"         "reaching"   "the"        "Pole"       "This"      
[166] "would"      "have"       "been"       "an"         "easy"      
[171] "matter"     "but"        "each"       "traveller"  "was"       
[176] "obliged"    "to"         "wear"       "snowshoes" 
```

Then, we will also create a document-feature matrix using the `dfm()` function from the `quanteda` package. This is simply an object that contains the information about the frequency of each single token in the text.

```
dm <- dfm(tokenized)

dm
```

```
Document-feature matrix of: 1 document, 106 features (0.00% sparse) and 0 docvars.
       features
docs    when the young people returned to ballroom  , it presented
  text1    1  19     1      1        1  3        1 14  2         1
[ reached max_nfeat ... 96 more features ]
```

```
featfreq(dm)
```

```
      when        the      young     people   returned         to   ballroom 
         1         19          1          1          1          3          1 
         ,         it  presented          a  decidedly    changed appearance 
        14          2          1          5          1          1          1 
         .    instead         of         an   interior      scene        was 
```

```
       11          1          4          2          1          1          6
   winter  landscape      floor    covered       with snow-white     canvas
        1          1          1          1          3          1          1
      not       laid         on   smoothly        but    rumpled       over
        1          1          4          1          2          1          1
    bumps        and   hillocks       like       real       snow      field
        1          9          1          2          1          2          1
 numerous      palms evergreens       that        had  decorated       room
        1          1          1          1          2          1          3
     were   powdered      flour     strewn      tufts     cotton       also
        4          1          1          1          1          1          1
  diamond       dust       been    lightly  sprinkled       them glittering
        1          1          2          1          1          1          1
  crystal    icicles       hung       from   branches         at       each
        1          1          2          1          1          3          3
      end       wall  beautiful  bear-skin        rug      these       rugs
        3          1          1          1          1          1          1
      for     prizes        one      girls       boys       this       game
        3          1          4          2          2          2          1
 gathered      other     called      north       pole      south     player
        1          2          1          1          3          1          1
    given      small       flag      which       they      plant   reaching
        1          1          1          1          1          1          1
    would       have       easy     matter  traveller    obliged       wear
        1          1          1          1          1          1          1
snowshoes
        1
```

1.2. Basic statistics

We can use the `texstat_summary()` function to obtain some basic information about any reading passage in this dataset such as number of characters, number of sentences, number of tokens, number of unique tokens, number of numbers, number of punctuation marks, number of symbols, number of tags, and number of emojis.

```
text_sm <- textstat_summary(dm)

text_sm
```

```
  document chars sents tokens types puncts numbers symbols urls tags emojis
1    text1    NA    NA    204   106     25       0       0    0    0      0
```

```
# for some reason it returns NAs for number of characters and number of sentences

text_sm$sents <- nsentence(text)
text_sm$chars <- nchar(text)

text_sm
```

```
  document chars sents tokens types puncts numbers symbols urls tags emojis
1    text1   992    11    204   106     25       0       0    0    0      0
```

1.3. Word length statistics

We can derive the distribution of word lengths in a passage. The number of words with length 1, 2, 3, . . . , 20 can be defined as predictive features. Summary statistics of this distribution (minimum word length, maximum word length, average word length, variability of word length) may also predict readability. Below is a code to extract these features and combine them in a vector.

```
# Word lengths

  wl <- nchar(tokenized.words[[1]])

  wl
```

```
  [1]  4  3  5  6  8  2  3  8  2  9  1  9  7 10  7  2  2  8  5  2  3  1  6  9  3
 [26]  5  3  7  4 10  6  3  4  2  8  3  7  4  5  3  8  4  1  4  4  5  3  8  5  3
 [51] 10  4  3  9  3  4  4  8  4  5  3  6  4  5  2  6  4  4  4  7  4  3  4  7  9
 [76]  2  4  3 10  7  7  4  4  3  8  2  4  3  2  3  4  2  3  4  4  1  9  9  3  5
[101]  4  4  3  6  3  3  3  5  3  3  3  3  4  3  4  3  3  4  3  5  4  8  2  3  3
[126]  2  3  4  3  3  4  2  3  5  3  3  3  3  6  3  5  4  3  3  5  3  5  4  4  6
[151]  3  5  1  5  4  5  4  4  2  5  2  8  3  4  4  5  4  4  2  4  6  3  4  9  3
[176]  7  2  4  9
```

```
# Summary stat. of word length

  min(wl)
```

```
[1] 1
```

```
  max(wl)
```

```
[1] 10
```

```
  mean(wl)
```

```
[1] 4.407821
```

```
  sd(wl)
```

```
[1] 2.12427
```

```
# Distribution of word lengths

  wl.tab <- table(wl)

  wl.tab
```

```
wl
 1  2  3  4  5  6  7  8  9 10
 5 18 50 45 20  9  9 10  9  4
```

```
# Word Length features combined

  wl.features <- data.frame(matrix(0,nrow=1,nco=30))
  colnames(wl.features) <- paste0('wl.',1:30)

  ind <- colnames(wl.features)%in%paste0('wl.',names(wl.tab))

  wl.features[,ind] <- wl.tab

  wl.features$mean.wl  <-   mean(wl)
  wl.features$sd.wl    <-   sd(wl)
  wl.features$min.wl   <-   min(wl)
  wl.features$max.wl   <-   max(wl)

  wl.features
```

```
  wl.1 wl.2 wl.3 wl.4 wl.5 wl.6 wl.7 wl.8 wl.9 wl.10 wl.11 wl.12 wl.13 wl.14
1    5   18   50   45   20    9    9   10    9     4     0     0     0     0
  wl.15 wl.16 wl.17 wl.18 wl.19 wl.20 wl.21 wl.22 wl.23 wl.24 wl.25 wl.26 wl.27
1     0     0     0     0     0     0     0     0     0     0     0     0     0
  wl.28 wl.29 wl.30  mean.wl   sd.wl min.wl max.wl
1     0     0     0 4.407821 2.12427      1     10
```

1.4. Text Entropy

In the `quanteda.textstats` package, `textstat_entropy()` provides a numerical measure of **text entropy**. The entropy is first computed by computing the proportion of each token, and then the entropy is computed using the following formula.

$$-\sum_i p_i * log_2(p_i),$$

where p_i is the proportion of i^th token.

```
# Do it yourself

y <- featfreq(dm)
p <- y/sum(y)
-sum(p*log(p,base=2))
```

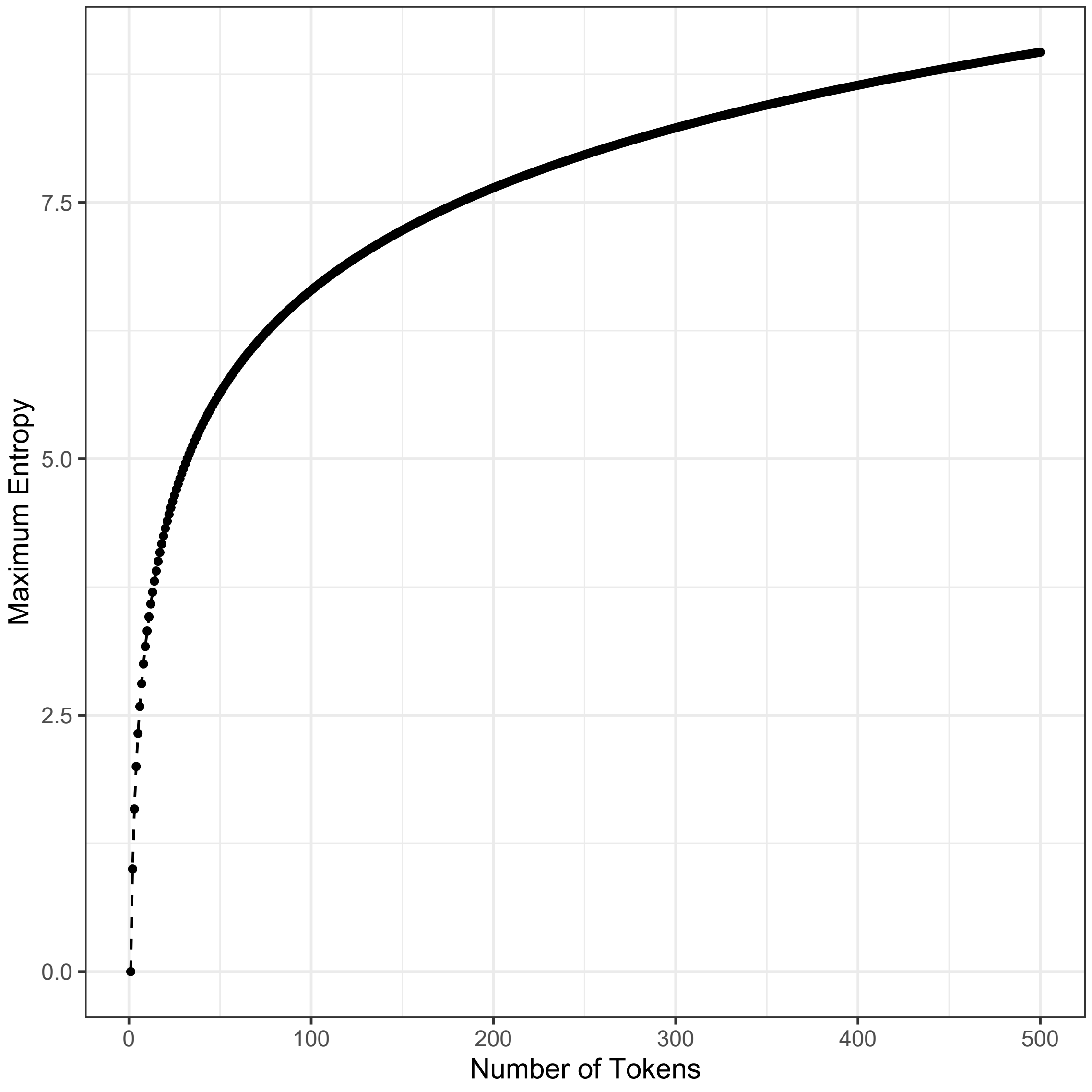
```
[1] 6.095194
```

```
# use the built-in function
# ?textstat_entropy

textstat_entropy(dm)
```

```
  document  entropy
1    text1 6.095194
```

Based on this definition of entropy, note that there is a maximum entropy for any text when each token appears only once in the text. We can find these for hypothetical texts with 1 to 500 tokens.

```r
n = 1:500
ent <- c()
for(i in 1:500){
 p <- rep(1/i,i)
 ent[i] <- -sum(p*log(p,base=2))
}

 ggplot()+
  geom_point(aes(x=n,y=ent),cex=1)+
  geom_line(aes(x=n,y=ent),lty=2)+
  theme_bw()+
  xlab('Number of Tokens')+
  ylab('Maximum Entropy')
```



The more tokens appear more than once, the entropy starts decreasing. See below for an hypothetical example

```r
# maximum entropy

p <- rep(1/50,50)
p
```

```
 [1] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[16] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[31] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[46] 0.02 0.02 0.02 0.02 0.02
```

```r
-sum(p*log(p,base=2))
```

```
[1] 5.643856
```

```r
# One of the tokens appears twice

p <- c(rep(1/50,48),2/50)
p
```

```
 [1] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[16] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[31] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[46] 0.02 0.02 0.02 0.04
```

```r
-sum(p*log(p,base=2))
```

```
[1] 5.603856
```

```r
# Two of the tokens appear twice

p <- c(rep(1/50,46),2/50,2/50)
p
```

```
 [1] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[16] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[31] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
[46] 0.02 0.04 0.04
```

```r
-sum(p*log(p,base=2))
```

```
[1] 5.563856
```

1.5. Lexical diversity

We can calculate a variety of indices regarding the lexical diversity by using the `textstat_lexdiv()` function. All these indices have two primary inputs: number of unique tokens (V), total number of tokens (N). The help page (`?textstat_lexdiv`) provides the formula for a total of 13 indices all a function of total number of tokens and number of unique tokens.

```
textstat_lexdiv(tokenized,
                remove_numbers = TRUE,
                remove_punct   = TRUE,
                remove_symbols = TRUE,
                measure        = 'all')
```

```
  document       TTR        C        R     CTTR        U         S        K
1    text1 0.5810056 0.895324 7.773325 5.496571 21.52215 0.8638636 238.1324
         I          D        Vm      Maas   MATTR MSTTR     lgV0   lgeV0
1 16.41275 0.01832904 0.1191548 0.2155545 0.61725  0.74 4.528431 10.4271
```

1.6. Measures of readability

We can calculate a variety of indices for readability. These indices exist for a long time and functions of variables such as number of words, number of characters, number of sentences, number of syllables, number of words matching the Dale-Chall List of 3000 "familiar words", average sentence length, average word length, ratio of number of words matching the Dale-Chall list of 3000 "familiar words" to the total number of all words, and number of "difficult" words not matching the Dale-Chall list of "familiar" words.

The `textstat_readability()` function provides about 46 different indices as functions of these variables, see `?textstat_readability` for all the formulas and references.

```
textstat_readability(text,
                     measure='all')
```

```
  document      ARI ARI.simple  ARI.NRI Bormuth.MC Bormuth.GP  Coleman
1    text1 7.276665   55.58714 5.270216  -2.864315   60485854 59.90359
  Coleman.C2 Coleman.Liau.ECP Coleman.Liau.grade Coleman.Liau.short Dale.Chall
1   59.48646         55.09734           7.967059           7.967735   41.62426
  Dale.Chall.old Dale.Chall.PSK Danielson.Bryan Danielson.Bryan.2
1       6.284634       5.587946        5.313453          87.13327
  Dickes.Steiwer      DRP      ELF Farr.Jenkins.Paterson   Flesch Flesch.PSK
1      -317.3508 386.4315 3.909091             -46.99924 78.89165   5.060137
  Flesch.Kincaid      FOG FOG.PSK  FOG.NRI  FORCAST FORCAST.RGL    Fucks
1       6.343295 9.012757 5.02533 15.20136 8.563536     7.84989 71.54545
  Linsear.Write      LIW      nWS    nWS.2    nWS.3    nWS.4      RIX Scrabble
1             4 33.02913 3.353659 4.166776 3.820909 4.344952 2.727273  1.68615
      SMOG   SMOG.C SMOG.simple  SMOG.de   Spache Spache.old   Strain
1 8.841846 8.787297    8.477226 3.477226 4.280939   4.869588 6.490909
  Traenkle.Bailer Traenkle.Bailer.2 Wheeler.Smith meanSentenceLength
1       -348.1211         -230.3328      39.09091           16.45455
  meanWordSyllables
1          1.314917
```

2. Lemmatization, Parts of Speech Tagging, and Dependency Parsing via udpipe

For this section, we will rely on the `udpipe` package.

```
require(udpipe)
```

We first need to download a pre-made language model. `udpipe` provides models for 65 different languages. We will download the model for English. When you run the code below, this will download a model file in your working directory.

```
udpipe_download_model(language = "english")
```

Note that, you have to download a model file only once. Once you download it, you can reload it to your R environment using the following code.

```
ud_eng <- udpipe_load_model(here('english-ewt-ud-2.5-191206.udpipe'))

ud_eng
```

```
$file
[1] "B:/UO Teaching/EDLD 654/c4-ml-fall-2021/english-ewt-ud-2.5-191206.udpipe"

$model
<pointer: 0x0000000017a252b0>

attr(,"class")
[1] "udpipe_model"
```

2.1. Morphological annotation

For a given plain text, we first run `udpipe_annotate()` function. This function returns a detailed analysis of the given text.

```
annotated <- udpipe_annotate(ud_eng, x = text)
annotated <- as.data.frame(annotated)
annotated <- cbind_morphological(annotated)

str(annotated)

head(annotated)
```

2.2. Part of Speech Tags (POS)

Two columns in this data frame provides POS tags. the column `upos` indicates Universal POS Tags. This is a list of values you will see in this column.

- ADJ: adjective
- ADP: adposition
- ADV: adverb
- AUX: auxiliary
- CCONJ: coordinating conjunction
- DET: determiner
- INTJ: interjection
- NOUN: noun

- NUM: numeral
- PART: particle
- PRON: pronoun
- PROPN: proper noun
- PUNCT: punctuation
- SCONJ: subordinating conjunction
- SYM: symbol
- VERB: verb
- X: other

We can simply count the number of times these tags appear in a text and use them as predictive features of how readable the text is.

```
table(annotated$upos)
```

The column `xpos` also provide similar information; however, these are language specific POS tags.

```
table(annotated$xpos)
```

While I couldn't find any reference to what these abbreviations mean, below is my best guess based on the information at this link (Oxford English part-of-speech Tagset) and this link (Brown Corpus).

- CC: coordinating conjunction (and, or)
- CD: cardinal numeral (one, two, 2, etc.)
- DT: singular determiner/quantifier (this, that)
- HYPH: hyphenation
- IN: preposition
- JJ: adjective
- MD: modal auxiliary (can, should, will)
- NN: singular or mass noun
- NNP: plural noun
- NNS: possessive plural noun
- PRP: ???
- RB: adverb
- TO: infinitive marker to
- VB: verb, base form
- VBD: verb, past tense
- VBG: verb, present participle/gerund
- VBN: verb, past participle
- WDT: wh- determiner (what, which)
- WRB: wh- adverb (how, where, when)

2.3. Features

The morphological features returned under the column `feats` distinguish additional lexical and grammatical properties of words, not covered by the POS tags. You will see these features independently appended to the data frame at the end with columns having a `morph_` prefix.Below is a list of these columns with links for more information.

- morph_case: https://universaldependencies.org/u/feat/Case.html

```
table(annotated$morph_case)
```

- morph_definite: https://universaldependencies.org/u/feat/Definite.html

```
table(annotated$morph_definite)
```

- morph_degree: https://universaldependencies.org/u/feat/Degree.html

```
table(annotated$morph_degree)
```

- morph_gender: https://universaldependencies.org/u/feat/Gender.html

```
table(annotated$morph_gender)
```

- morph_mood: https://universaldependencies.org/u/feat/Mood.html

```
table(annotated$morph_mood)
```

- morp_number: https://universaldependencies.org/u/feat/Number.html

```
table(annotated$morph_number)
```

- morph_numtype: https://universaldependencies.org/u/feat/NumType.html

```
table(annotated$morph_numtype)
```

- morph_person: https://universaldependencies.org/u/feat/Person.html

```
table(annotated$morph_person)
```

- morph_prontype: https://universaldependencies.org/u/feat/PronType.html

```
table(annotated$morph_prontype)
```

- morph_tense: https://universaldependencies.org/u/feat/Tense.html

```
table(annotated$morph_tense)
```

- morph_verbform: https://universaldependencies.org/u/feat/VerbForm.html

```
table(annotated$morph_verbform)
```

- morph_voice: https://universaldependencies.org/u/feat/Voice.html

```
table(annotated$morph_voice)
```

2.4. Syntactic relations

The column `dep_rel` in this data frame provides information about the syntactic relations. There are 37 possible tags for the universal syntactic relations. More information can be found at this link.

- acl, clausal modifier of noun (adnominal clause)
- acl:relcl, relative clause modifier
- advcl, adverbial clause modifier
- advmod, adverbial modifier
- advmod:emph, emphasizing word, intensifier
- advmod:lmod, locative adverbial modifier
- amod, adjectival modifier
- appos, appositional modifier
- aux, auxiliary
- aux:pass, passive auxiliary
- case, case marking
- cc, coordinating conjunction
- cc:preconj, preconjunct
- ccomp, clausal complement
- clf, classifier
- compound, compound
- compound:lvc, light verb construction
- compound:prt, phrasal verb particle
- compound:redup, reduplicated compounds
- compound:svc, serial verb compounds
- conj, conjunct
- cop, copula
- csubj, clausal subject
- csubj:pass, clausal passive subject
- dep, unspecified dependency
- det, determiner
- det:numgov, pronominal quantifier governing the case of the noun
- det:nummod, pronominal quantifier agreeing in case with the noun
- det:poss, possessive determiner
- discourse, discourse element
- dislocated, dislocated elements
- expl, expletive
- expl:impers, impersonal expletive
- expl:pass, reflexive pronoun used in reflexive passive
- expl:pv, reflexive clitic with an inherently reflexive verb
- fixed, fixed multiword expression
- flat, flat multiword expression
- flat:foreign, foreign words
- flat:name, names
- goeswith, goes with
- iobj, indirect object
- list, list
- mark, marker
- nmod, nominal modifier
- nmod:poss, possessive nominal modifier

- nmod:tmod, temporal modifier
- nsubj, nominal subject
- nsubj:pass, passive nominal subject
- nummod, numeric modifier
- nummod:gov, numeric modifier governing the case of the noun
- obj, object
- obl, oblique nominal
- obl:agent, agent modifier
- obl:arg, oblique argument
- obl:lmod, locative modifier
- obl:tmod, temporal modifier
- orphan, orphan
- parataxis, parataxis
- punct, punctuation
- reparandum, overridden disfluency
- root, root
- vocative, vocative
- xcomp, open clausal complement

We can similarly count the number of times these tags appear in a text and use them as predictive features.

```
table(annotated$dep_rel)
```

3. Natural Language Processing (NLP)

NLP is a more advanced modeling that goes beyond simple summary of text characteristics. To put it in a very naive way (and we will not do anything beyond it), NLP models takes a plain text, process the text to put it into little pieces (tokens, lemmas, words, POS tags, co-occurrences etc.), then use a very complex neural network model to convert it to a numerical vector that is meaningful and represent the text.

Most recently, a group of scholars called these models as part of Foundation Models. A brief list for some of these NLP models and some information along with them including links to original papers are listed below.These models are very expensive to train and uses enormous amount of data available to train. For instance, Bert/Roberta were trained using the entire Wikipedia and a Book Corpus (a total of ~ 4.7 billion words), GPT2 was trained using 8 million web pages, and GPT3 was trained on 45 TB of data from the internet and books.

Model	Developer	Year	# of parameters	Estimated Cost
Bert-Large	Google AI	2018	336 M	$ 7K
Roberta-Large	Facebook AI	2019	335 M	?
GPT2-XL	Open AI	2019	1.5 B	$ 50K
T5	Google AI	2020	11 B	$ 1.3 M
GPT3	OpenAI	2020	175 B	$ 4.6 M

All these models except GPT3 is open source and can be immediately utilized using open libraries (typically using Python), and these models can be customized to implement very specific tasks (e.g., question answering, sentiment analysis, translation, etc.). GPT3 is apparently the most powerful developed so far, and it can only be accessed through a private API, https://beta.openai.com/. You can explore some of the GPT3 applications on this website, https://gpt3demo.com/. Below are a few of them:

- Artificial tweets (https://thoughts.sushant-kumar.com/word)

- Creative writing (https://www.gwern.net/GPT-3)
- Interview with (artificial) Einstein (https://maraoz.com/2021/03/14/einstein-gpt3/)

If you have time, this series of Youtube videos provide some background and accessible information about these models. In particular, Episode 2 will give a good idea about what these numerical embeddings represent. If you want to get in-depth coverage of speech and language processing from scratch, this freely available book provides a good amount of material.

In this lecture, we will only scratch the surface and focus on the tools available to make these models accessible through R (a political way of saying I don't know what I am doing!). In particular, we will use the text package to connect with HuggingFace's Transformers library in Python and explore the word and sentence embeddings derived from the NLP models. The text package provides access to a large list of models from HuggingFace's library, see https://huggingface.co/transformers/pretrained_models.html.

-

3.1. Loading the required packages

First, we need to prepare the environment. From Lecture 1a, you should have installed the necessary packages. Now, we will load all these packages using the code below.

```
require(reticulate)

# List the available Python environments

virtualenv_list()

# Import the modules

reticulate::import('torch')
reticulate::import('numpy')
reticulate::import('transformers')
reticulate::import('nltk')
reticulate::import('tokenizers')

# Load the text package

require(text)
```

```
[1] "my.python"  "my.python2"
Module(torch)
Module(numpy)
Module(transformers)
Module(nltk)
Module(tokenizers)
```

3.2. Word Embeddings

A word embedding is a vector that numerically represents a word. The length of this vector represents the number of dimensions the word is represented. we can also consider these numbers as coordinates in a geometric space with many dimensions. The words with similar meanings in this space will be closer to each other. The number of dimensions depends on the model and how it is trained. For instance, Roberta

Base represents each word in 768 dimensions, while GPT2-Large represent each word in 1024 dimensions and GPT2-XL represents in 1600 dimensions.

For instance, let's consider a word (e.g., sofa) and get the word embeddings from different models using the textEmbed package. Note that it will download the model files when you first request word embeddings using a new model. Depending on the complexity of these models, some of these files may be big (larger than 10GB) and it mat take long to download them depending on your connection. Once you use a model once, then it will be quicker to get word embeddings. Below, I am asking the word embeddings from four different models: `roberta-base`, `gpt2-large`, `gpt-neo`. You can specify any model from this list. The name provided to the `model` argument of the `textEmbed` function is the one listed under the **Model id** column in this list.

```
tmp1 <- textEmbed(x     = 'sofa',
                  model = 'roberta-base',
                  layers = 1)

tmp1$x

length(tmp1$x)
```

```
# A tibble: 1 x 768
   Dim1   Dim2    Dim3  Dim4  Dim5   Dim6  Dim7   Dim8   Dim9  Dim10  Dim11
  <dbl>  <dbl>   <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 0.110 0.0218 -0.0114 0.238 0.654 -0.276 0.118 -0.205 0.0886 0.0395 -0.563
# ... with 757 more variables: Dim12 <dbl>, Dim13 <dbl>, Dim14 <dbl>,
#   Dim15 <dbl>, Dim16 <dbl>, Dim17 <dbl>, Dim18 <dbl>, Dim19 <dbl>,
#   Dim20 <dbl>, Dim21 <dbl>, Dim22 <dbl>, Dim23 <dbl>, Dim24 <dbl>,
#   Dim25 <dbl>, Dim26 <dbl>, Dim27 <dbl>, Dim28 <dbl>, Dim29 <dbl>,
#   Dim30 <dbl>, Dim31 <dbl>, Dim32 <dbl>, Dim33 <dbl>, Dim34 <dbl>,
#   Dim35 <dbl>, Dim36 <dbl>, Dim37 <dbl>, Dim38 <dbl>, Dim39 <dbl>,
#   Dim40 <dbl>, Dim41 <dbl>, Dim42 <dbl>, Dim43 <dbl>, Dim44 <dbl>, ...
[1] 768
```

If we use `roberta-base`, we get a vector with length 768 because Roberta Base was trained using 768 hidden states. You can try the following code yourself and see how the dimensions of the vector change depending on the model based on the number of hidden states used during the training.

```
tmp2 <- textEmbed(x     = 'sofa',
                  model = 'gpt2-large',
                  layers = 1)

tmp2$x

as.numeric(tmp2$x)

length(tmp2$x)
```

```
tmp3 <- textEmbed(x     = 'sofa',
                  model = 'EleutherAI/gpt-neo-2.7B',
                  layers = 1)

tmp3$x
```

```
as.numeric(tmp3$x)

length(tmp3$x)
```

As you noticed, we also specified argument `layers=1`, so the function returns the embeddings from the first layer. These models typically have multiple layers. For instance, `roberta-base` has 12 layers, `gpt-neo` has 32 layers, `gpt2-large` has 36 layers. Multiple layers enable the model to capture non-linear relationships to represent complex language structures. Each layer returns a different vector of embeddings with the same length. We can request embeddings from any layer or from multiple layers at the same time.

For instance, we can request the embeddings from the last layer of `roberta-base` and these embeddings will also have a length of 768, but the numbers would be different than the ones we got before from the first layer.

```
tmp1 <- textEmbed(x     = 'sofa',
                  model = 'roberta-base',
                  layers = 12)

tmp1$x

length(tmp1$x)
```

```
# A tibble: 1 x 768
     Dim1    Dim2   Dim3    Dim4  Dim5   Dim6   Dim7    Dim8   Dim9  Dim10
    <dbl>   <dbl>  <dbl>   <dbl> <dbl>  <dbl>  <dbl>   <dbl>  <dbl>  <dbl>
1 -0.0357 0.00915 0.0452 -0.0228 0.444 -0.214 0.0168 -0.0311 0.0108 -0.110
# ... with 758 more variables: Dim11 <dbl>, Dim12 <dbl>, Dim13 <dbl>,
#   Dim14 <dbl>, Dim15 <dbl>, Dim16 <dbl>, Dim17 <dbl>, Dim18 <dbl>,
#   Dim19 <dbl>, Dim20 <dbl>, Dim21 <dbl>, Dim22 <dbl>, Dim23 <dbl>,
#   Dim24 <dbl>, Dim25 <dbl>, Dim26 <dbl>, Dim27 <dbl>, Dim28 <dbl>,
#   Dim29 <dbl>, Dim30 <dbl>, Dim31 <dbl>, Dim32 <dbl>, Dim33 <dbl>,
#   Dim34 <dbl>, Dim35 <dbl>, Dim36 <dbl>, Dim37 <dbl>, Dim38 <dbl>,
#   Dim39 <dbl>, Dim40 <dbl>, Dim41 <dbl>, Dim42 <dbl>, Dim43 <dbl>, ...
[1] 768
```

Or, we can request the embeddings from the last four layers. Now, each layer will have a length of 768, and the returning object is now a vector of length 3072 (768 x 4). Instead of contenating, you can also ask to take the minimum, maximum, or mean of the requested layers, and it will return a vector with a length of 768 .

```
tmp1 <- textEmbed(x     = 'sofa',
                  model = 'roberta-base',
                  layers = 9:12,
                  context_aggregation_layers = 'concatenate')

tmp1$x

length(tmp1$x)



tmp1 <- textEmbed(x     = 'sofa',
                  model = 'roberta-base',
                  layers = 9:12,
                  context_aggregation_layers = 'mean')

tmp1$x

length(tmp1$x)
```

```
# A tibble: 1 x 3,072
    Dim1    Dim2   Dim3  Dim4  Dim5   Dim6    Dim7  Dim8   Dim9  Dim10  Dim11
   <dbl>   <dbl>  <dbl> <dbl> <dbl>  <dbl>   <dbl> <dbl>  <dbl>  <dbl>  <dbl>
1 0.0220 -0.100 0.0228 0.261 0.526 0.0779 -0.0801 0.111 -0.313 -0.264 -0.658
# ... with 3,061 more variables: Dim12 <dbl>, Dim13 <dbl>, Dim14 <dbl>,
#   Dim15 <dbl>, Dim16 <dbl>, Dim17 <dbl>, Dim18 <dbl>, Dim19 <dbl>,
#   Dim20 <dbl>, Dim21 <dbl>, Dim22 <dbl>, Dim23 <dbl>, Dim24 <dbl>,
#   Dim25 <dbl>, Dim26 <dbl>, Dim27 <dbl>, Dim28 <dbl>, Dim29 <dbl>,
#   Dim30 <dbl>, Dim31 <dbl>, Dim32 <dbl>, Dim33 <dbl>, Dim34 <dbl>,
#   Dim35 <dbl>, Dim36 <dbl>, Dim37 <dbl>, Dim38 <dbl>, Dim39 <dbl>,
#   Dim40 <dbl>, Dim41 <dbl>, Dim42 <dbl>, Dim43 <dbl>, Dim44 <dbl>, ...
[1] 3072
# A tibble: 1 x 768
     Dim1    Dim2   Dim3  Dim4  Dim5    Dim6   Dim7   Dim8   Dim9  Dim10  Dim11
    <dbl>   <dbl>  <dbl> <dbl> <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 0.00216 -0.129 0.0735 0.155 0.524 -0.0404 0.0564 0.0261 -0.164 -0.209 -0.556
# ... with 757 more variables: Dim12 <dbl>, Dim13 <dbl>, Dim14 <dbl>,
#   Dim15 <dbl>, Dim16 <dbl>, Dim17 <dbl>, Dim18 <dbl>, Dim19 <dbl>,
#   Dim20 <dbl>, Dim21 <dbl>, Dim22 <dbl>, Dim23 <dbl>, Dim24 <dbl>,
#   Dim25 <dbl>, Dim26 <dbl>, Dim27 <dbl>, Dim28 <dbl>, Dim29 <dbl>,
#   Dim30 <dbl>, Dim31 <dbl>, Dim32 <dbl>, Dim33 <dbl>, Dim34 <dbl>,
#   Dim35 <dbl>, Dim36 <dbl>, Dim37 <dbl>, Dim38 <dbl>, Dim39 <dbl>,
#   Dim40 <dbl>, Dim41 <dbl>, Dim42 <dbl>, Dim43 <dbl>, Dim44 <dbl>, ...
[1] 768
```

It is not completely undderstood what different dimensions represent or what these different layers mean (or, I couldn't find any document/source that explains it. Let me know if you come across one). Typically, it is recommended to use the last four layers of word embedding (I read or watched this somewhere, but I don't really remember the source).

3.3. Sentence Embeddings

Similar to words, we can also represent a whole sentence using a numerical vector. Let's consider the sentence *I like to drink Turkish coffee*, and let's see what we get from `roberta-base`.

```
tmp1 <- textEmbed(x     = 'I like to drink Turkish coffee',
                  model = 'roberta-base',
                  layers = 12,
                  context_aggregation_layers = 'concatenate')
```

Now, the returned object will have two major elements. First, it will return a matrix of word embeddings for each word in this sentence.

```
tmp1$singlewords_we
```

```
# A tibble: 6 x 770
  words       n     Dim1    Dim2     Dim3     Dim4   Dim5    Dim6     Dim7    Dim8
  <chr>   <int>    <dbl>   <dbl>    <dbl>    <dbl>  <dbl>   <dbl>    <dbl>   <dbl>
1 coffee      1 -0.106    0.0884  4.52e-2 -0.0600  0.185  -0.106  -0.0485  -0.0486
2 drink       1 -0.0850  -0.0404  5.39e-2 -0.00823 0.0154 -0.136  -0.00811 -0.0724
3 i           1 -0.00472 -0.0233 -7.32e-3 -0.153   0.221  -0.172  -0.0262   0.0784
4 like        1  0.0131  -0.0387  3.33e-2 -0.0927  0.201  -0.114  -0.0258   0.0566
5 to          1  0.00515  0.0257  6.82e-4 -0.0410  0.249  -0.0950 -0.0214   0.0614
6 turkish     1  0.0757   0.178   8.82e-2  0.0341  0.0779 -0.104  -0.0348   0.111
# ... with 760 more variables: Dim9 <dbl>, Dim10 <dbl>, Dim11 <dbl>,
#   Dim12 <dbl>, Dim13 <dbl>, Dim14 <dbl>, Dim15 <dbl>, Dim16 <dbl>,
#   Dim17 <dbl>, Dim18 <dbl>, Dim19 <dbl>, Dim20 <dbl>, Dim21 <dbl>,
#   Dim22 <dbl>, Dim23 <dbl>, Dim24 <dbl>, Dim25 <dbl>, Dim26 <dbl>,
#   Dim27 <dbl>, Dim28 <dbl>, Dim29 <dbl>, Dim30 <dbl>, Dim31 <dbl>,
#   Dim32 <dbl>, Dim33 <dbl>, Dim34 <dbl>, Dim35 <dbl>, Dim36 <dbl>,
#   Dim37 <dbl>, Dim38 <dbl>, Dim39 <dbl>, Dim40 <dbl>, Dim41 <dbl>, ...
```

It will also have a vector of embeddings for the whole sentence.

```
tmp1$x
```

```
# A tibble: 1 x 768
     Dim1  Dim2   Dim3    Dim4     Dim5    Dim6   Dim7    Dim8    Dim9   Dim10
    <dbl> <dbl>  <dbl>   <dbl>    <dbl>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl>
1 -0.0118 0.103 0.0109 -0.0462 -0.00487 -0.0450 0.0823 -0.0307 0.00255 -0.0749
# ... with 758 more variables: Dim11 <dbl>, Dim12 <dbl>, Dim13 <dbl>,
#   Dim14 <dbl>, Dim15 <dbl>, Dim16 <dbl>, Dim17 <dbl>, Dim18 <dbl>,
#   Dim19 <dbl>, Dim20 <dbl>, Dim21 <dbl>, Dim22 <dbl>, Dim23 <dbl>,
#   Dim24 <dbl>, Dim25 <dbl>, Dim26 <dbl>, Dim27 <dbl>, Dim28 <dbl>,
#   Dim29 <dbl>, Dim30 <dbl>, Dim31 <dbl>, Dim32 <dbl>, Dim33 <dbl>,
#   Dim34 <dbl>, Dim35 <dbl>, Dim36 <dbl>, Dim37 <dbl>, Dim38 <dbl>,
#   Dim39 <dbl>, Dim40 <dbl>, Dim41 <dbl>, Dim42 <dbl>, Dim43 <dbl>, ...
```

Now, let's check the numerical representation of the first reading passage in our dataset. I will use `gpt-neo` and request the last 4 layers.

```
text

tmp3 <- textEmbed(x     = text,
                  model = 'roberta-base',
                  layers = 9:12,
                  context_aggregation_layers = 'concatenate')

tmp3$x

length(tmp3$x)
```

```
[1] "When the young people returned to the ballroom, it presented a decidedly changed appearance. Instea
# A tibble: 1 x 3,072
```

```
     Dim1   Dim2   Dim3    Dim4   Dim5     Dim6   Dim7  Dim8  Dim9  Dim10  Dim11
    <dbl>  <dbl>  <dbl>   <dbl>  <dbl>    <dbl>  <dbl> <dbl> <dbl>  <dbl>  <dbl>
1 0.0888 0.0811 0.0540 -0.0734 -0.145 -0.00628 -0.107 0.169 0.104 0.0980 -0.180
# ... with 3,061 more variables: Dim12 <dbl>, Dim13 <dbl>, Dim14 <dbl>,
#   Dim15 <dbl>, Dim16 <dbl>, Dim17 <dbl>, Dim18 <dbl>, Dim19 <dbl>,
#   Dim20 <dbl>, Dim21 <dbl>, Dim22 <dbl>, Dim23 <dbl>, Dim24 <dbl>,
#   Dim25 <dbl>, Dim26 <dbl>, Dim27 <dbl>, Dim28 <dbl>, Dim29 <dbl>,
#   Dim30 <dbl>, Dim31 <dbl>, Dim32 <dbl>, Dim33 <dbl>, Dim34 <dbl>,
#   Dim35 <dbl>, Dim36 <dbl>, Dim37 <dbl>, Dim38 <dbl>, Dim39 <dbl>,
#   Dim40 <dbl>, Dim41 <dbl>, Dim42 <dbl>, Dim43 <dbl>, Dim44 <dbl>, ...
[1] 3072
```

So what?

NLP models provide meaningful contextual numerical representations of words or sentences. These numerical representations can be used as input features for predictive models to predict a certain outcome. In our case, we can utilize the embeddings from each reading passage to predict its reability.

4. Wrapping-up

This lecture provided different types of information we can extract from any given text. The code below extracts these different types of information for every single passage in the readability dataset using a `for` loop. At the end, it creates a single data with ??? input features and the outcome (readability score). If you run this by yourself, it may take a long time. The final dataset with input features and the outcome to predict can be downloaded from this link.

```
################################################################################

  require(quanteda)
  require(quanteda.textstats)
  require(udpipe)
  require(reticulate)
  require(text)

  ud_eng <- udpipe_load_model(here('english-ewt-ud-2.5-191206.udpipe'))

  virtualenv_list()

  reticulate::import('torch')
  reticulate::import('numpy')
  reticulate::import('transformers')
  reticulate::import('nltk')
  reticulate::import('tokenizers')

################################################################################

  readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/r

  input.list <- vector('list',nrow(readability))

  for(i in 1:nrow(readability)){

    # Assign the text to analyze
  text <- readability[i,]$excerpt

# Tokenization and document-feature matrix

  tokenized <- tokens(text,
                      remove_punct = TRUE,
                      remove_numbers = TRUE,
                      remove_symbols = TRUE,
                      remove_separators = TRUE)

  dm <- dfm(tokenized)

# basic text stats

  text_sm <- textstat_summary(dm)
  text_sm$sents <- nsentence(text)
  text_sm$chars <- nchar(text)

    # text_sm[2:length(text_sm)]

# Word-length features

  wl <- nchar(tokenized[[1]])

  wl.tab <- table(wl)

  wl.features <- data.frame(matrix(0,nrow=1,nco=30))
  colnames(wl.features) <- paste0('wl.',1:30)

  ind <- colnames(wl.features)%in%paste0('wl.',names(wl.tab))

  wl.features[,ind] <- wl.tab

  wl.features$mean.wl  <-   mean(wl)
  wl.features$sd.wl    <-   sd(wl)
  wl.features$min.wl   <-   min(wl)
  wl.features$max.wl   <-   max(wl)

  # wl.features

# Text entropy/Max entropy ratio

  t.ent <- textstat_entropy(dm)
  n     <-  sum(featfreq(dm))
  p     <- rep(1/n,n)
  m.ent <- -sum(p*log(p,base=2))

  ent <- t.ent$entropy/m.ent

  # ent

# Lexical diversity
```

```
  text_lexdiv <- textstat_lexdiv(tokenized,
                                 remove_numbers = TRUE,
                                 remove_punct   = TRUE,
                                 remove_symbols = TRUE,
                                 measure        = 'all')

  # text_lexdiv[,2:ncol(text_lexdiv)]

# Measures of readability

  text_readability <- textstat_readability(text,measure='all')

  # text_readability[,2:ncol(text_readability)]

# POS tag frequency

  annotated <- udpipe_annotate(ud_eng, x = text)
  annotated <- as.data.frame(annotated)
  annotated <- cbind_morphological(annotated)

  pos_tags <- c(table(annotated$upos),table(annotated$xpos))

  # pos.tags

# Syntactic relations

  dep_rel <- table(annotated$dep_rel)

    #dep_rel

# morphological features

  feat_names <- c('morph_abbr','morph_animacy','morph_aspect','morph_case',
                  'morph_clusivity','morph_definite','morph_degree',
                  'morph_evident','morph_foreign','morph_gender','morph_mood',
                  'morph_nounclass','morph_number','morph_numtype',
                  'morph_person','morph_polarity','morph_polite','morph_poss',
                  'morph_prontype','morph_reflex','morph_tense','morph_typo',
                  'morph_verbform','morph_voice')

  feat_vec <- c()

  for(j in 1:length(feat_names)){

    if(feat_names[j]%in%colnames(annotated)){
      morph_tmp   <- table(annotated[,feat_names[j]])
      names_tmp   <- paste0(feat_names[j],'_',names(morph_tmp))
      morph_tmp   <- as.vector(morph_tmp)
      names(morph_tmp) <- names_tmp
      feat_vec  <- c(feat_vec,morph_tmp)
    }
  }
```

```
        # feat_vec

    # Sentence Embeddings

      embeds <- textEmbed(x     = text,
                          model = 'roberta-base',
                          layers = 12,
                          context_aggregation_layers = 'concatenate')

      embeds$x


    # combine them all into one vector and store in the list object

      input.list[[i]] <- cbind(text_sm[2:length(text_sm)],
                                wl.features,
                                as.data.frame(ent),
                                text_lexdiv[,2:ncol(text_lexdiv)],
                                text_readability[,2:ncol(text_readability)],
                                t(as.data.frame(pos_tags)),
                                t(as.data.frame(c(dep_rel))),
                                t(as.data.frame(feat_vec)),
                                as.data.frame(embeds$x)
                                )

      print(i)
  }


  # Combine all elements of list in a single data frame

  require(gtools)

  readability_features <- smartbind(list = input.list)

  # Add the target score to the feature matrix

  readability_features$target <- readability$target

  # Export the final data with features

  write.csv(readability_features,
            here('data/readability_features.csv'),
            row.names = FALSE)
```

```
readability <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/ma
                        header=TRUE)

colnames(readability)
```