TextMining-NLP

December 10, 2021

[1]:
```
import pandas as pd
pd.options.plotting.backend = 'plotly'
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import os
import string
import copy
import re
import math
```

[2]:
```
# Folder Path
path = "datasets/Tel_text"

# Read text File
def read_text_file(file):
    file_path = f"{path}/{file}"
    with open(file_path, 'r', encoding="utf8", errors='ignore') as f:
        return f.read()

i = 0
list_documents = []
# iterate through all file
for file in os.listdir(path):
    # Check whether file is in text format or not
    if file.endswith(".txt"):
        i += 1
        globals()[f"{file[:-4]}"] = read_text_file(file)
        list_documents.append(globals()[f"{file[:-4]}"])
```

1 Pre-processing

[3]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
from num2words import num2words
```

[4]:
```
def convert_lower_case(data):
    return np.char.lower(data)

def remove_stop_words(data):
    stop_words = stopwords.words('english') + stopwords.words('french')
    words = word_tokenize(str(data))
    new_text = ""
    for w in words:
        if w not in stop_words and len(w) > 1:
            new_text = new_text + " " + w
    return new_text

def remove_punctuation(data):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        data = np.char.replace(data, symbols[i], ' ')
        data = np.char.replace(data, "  ", " ")
    data = np.char.replace(data, ',', '')
    return data

def remove_apostrophe(data):
    return np.char.replace(data, "'", "")



def stemming(data):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " + stemmer.stem(w)
    return new_text

def convert_numbers(data):
    tokens = word_tokenize(str(data))
    new_text = ""
    for w in tokens:
        try:
            w = num2words(int(w), lang="en")
        except:
            a = 0
        new_text = new_text + " " + w
    new_text = np.char.replace(new_text, "-", " ")
    return new_text
```

[5]:
```
def preprocess(data):
    data = convert_lower_case(data)
    data = remove_punctuation(data) #remove comma seperately
    data = remove_apostrophe(data)
    data = remove_stop_words(data)
    data = convert_numbers(data)
    data = stemming(data)
    data = remove_punctuation(data)
    data = convert_numbers(data)
    data = stemming(data) # needed again as we need to stem the words
    data = remove_punctuation(data) # needed again as num2word is giving few␣
 ↪hypens and commas fourty-one
    data = remove_stop_words(data) # needed again as num2word is giving stop␣
 ↪words 101 - one hundred and one

    return data
```

[6]:
```
test_documents = list_documents[0:10]

for idx, val in enumerate(test_documents):
    test_documents[idx] = preprocess(val)
```

1.1 TF - IDF

[7]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(test_documents)
tfidf_matrix.shape
```

[7]:
```
(10, 27674)
```

1.2 Cosine similarity

[8]:
```
from sklearn.metrics.pairwise import cosine_similarity

similarities = cosine_similarity(tfidf_matrix)
print('pairwise dense output:\n {}\n'.format(similarities))
```

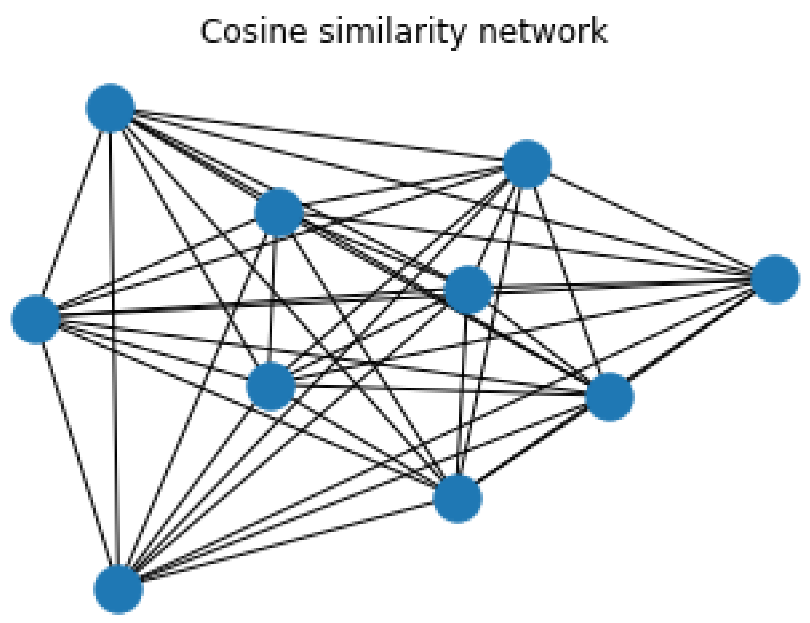
```
pairwise dense output:
 [[1.         0.39341766 0.35595603 0.43856948 0.27935166 0.31300715
  0.20123686 0.46267617 0.24483929 0.3867369 ]
 [0.39341766 1.         0.48390909 0.48189967 0.40508729 0.34038657
  0.32625398 0.53073916 0.14308435 0.50425888]
 [0.35595603 0.48390909 1.         0.4588742  0.3928195  0.34337006
  0.2722677  0.52273473 0.14249812 0.49117823]
 [0.43856948 0.48189967 0.4588742  1.         0.35365867 0.3734062
  0.26171301 0.5985373  0.17954202 0.46007272]
 [0.27935166 0.40508729 0.3928195  0.35365867 1.         0.25434593
  0.23210766 0.41064956 0.12300812 0.37437635]
 [0.31300715 0.34038657 0.34337006 0.3734062  0.25434593 1.
  0.17729127 0.4364209  0.13133622 0.35203154]
 [0.20123686 0.32625398 0.2722677  0.26171301 0.23210766 0.17729127
  1.         0.27395733 0.0822706  0.26595698]
 [0.46267617 0.53073916 0.52273473 0.5985373  0.41064956 0.4364209
  0.27395733 1.         0.18318258 0.51444692]
 [0.24483929 0.14308435 0.14249812 0.17954202 0.12300812 0.13133622
  0.0822706  0.18318258 1.         0.14406519]
 [0.3867369  0.50425888 0.49117823 0.46007272 0.37437635 0.35203154
  0.26595698 0.51444692 0.14406519 1.        ]]
```

[9]:
```python
adj_matrix = similarities
for i in range(10):
    adj_matrix[i, i] = 0
```

[19]:
```python
import networkx as nx

G = nx.from_numpy_matrix(similarities)
f = plt.figure()
plt.title("Cosine similarity network")
plt.axis('off')
nx.draw(G, ax=f.add_subplot(111))
f.show()
f.savefig("plots/graph.png")
```

```
/var/folders/37/9ncc0zbd061cflxwwq4w1_z00000gn/T/ipykernel_69320/2336132023.py:8
: UserWarning: Matplotlib is currently using
module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  f.show()
```



1.3 Stanza

```
[11]: import stanza
      import spacy
      import spacy_stanza
```

```
Init Plugin
Init Graph Optimizer
Init Kernel
```

```
[12]: # Import data

      df_3000 = pd.read_excel("datasets/7000_sentences.xlsx", sheet_name="3000",␣
       ↪usecols=["ID", "English", "German", "French"], index_col="ID")
      df_6000 = pd.read_excel("datasets/7000_sentences.xlsx", sheet_name="6000",␣
       ↪usecols=["ID", "English sentence", "German", "French"], index_col="ID")
      df_6000.rename(columns={"English sentence": "English"}, inplace = True)
      df = df_3000.append(df_6000)
```

```
[ ]: stanza.download(lang='en', model_dir ='./stanza_models')
     nlp_en = spacy_stanza.load_pipeline("en", dir = './stanza_models')
```

```
[14]: df_eng = df["English"].dropna()
```

```
[15]: from lemminflect import getLemma, getAllInflectionsOOV
      import random
```

```python
list_df_noun = []
list_df_verb = []
dict_type = {"VB"  : "infinitive",
             "VBD" : "past tense",
             "VBG" : "present participle",
             "VBN" : "past participle",
             "VBP" : "non-3rd person singular present",
             "VBZ" : "3rd person singular present",
             "MD"  : "Modal"}

count = 0
for phrase in df_eng[:50]:
    count += 1
    if count % 50 == 0:
        print(count)

    list_noun = []
    list_verb = []
    doc = nlp_en(phrase)
    for token in doc:
        # Get noun and number
        if token.pos_ == 'NOUN' and len(token.morph.get("Number")) > 0:
            list_noun.append({token.text: token.morph.get("Number")[0]})

        # Get verb
        if token.pos_ == 'VERB':
            prefix = ''
            if len(token.morph.get("Voice")) > 0 and token.morph.
 ↪get("Voice")[0] == "Pass":
                temp_token = token
                while temp_token.nbor(-1).pos_ == "AUX":
                    prefix = temp_token.nbor(-1).text + ' ' + prefix
                    temp_token = temp_token.nbor(-1)

            # Get lema
            lemma_verb = getLemma(token.text, upos='VERB')[0]
            # Get inflections
            inflections_verb = getAllInflectionsOOV(lemma_verb, upos='VERB')
            i, D1, D2, D3, F_D1, F_D2, F_D3 = 0, '', '', '', '', '' ,''
            # Shuffle inflections dict for randomness
            l = list(inflections_verb.items())
            random.shuffle(l)
            inflections_verb = dict(l)
            for key, value in inflections_verb.items():
                if not (value in [prefix + token.text, D1, D2, D3]):
                    i += 1
                    globals()[f"D{i}"] = value[0]
                    globals()[f"F_D{i}"] = dict_type[key]

            list_verb.append({"Answer": prefix + token.text,
                              "F_Answer": dict_type[token.tag_],
                              "Lemma": lemma_verb,
                              "D1": D1,
                              "D2": D2,
                              "D3": D3,
                              "F_D1": F_D1,
                              "F_D2": F_D2,
                              "F_D3": F_D3
                             })

    # Add noun
    for noun in list_noun:
        for key, value in noun.items():
            list_df_noun.append([phrase, key, value])

    # Add verb
    for verb in list_verb:
        question = phrase.replace(verb["Answer"], '...')
        list_df_verb.append([phrase, question, verb["Lemma"],
                             verb["Answer"], verb["D1"], verb["D2"], verb["D3"],
                             verb["F_Answer"], verb["F_D1"], verb["F_D2"],␣
↪verb["F_D3"]
                            ])
```

```
50
```

[16]:
```
df_noun = pd.DataFrame(list_df_noun, columns=["Phrase", "Noun", "Number"])
df_noun
```

[16]:
```
                                               Phrase        Noun Number
0   The beauty of the landscape struck the travell…      beauty   Sing
1   The beauty of the landscape struck the travell…   landscape   Sing
2   The beauty of the landscape struck the travell…  travellers   Plur
3           Nobody knows the truth about this affair.       truth   Sing
4           Nobody knows the truth about this affair.      affair   Sing
..                                                ...         ...    ...
67              The road is wide enough for two cars.        cars   Plur
68             The trip was too long, I am exhausted.        trip   Sing
69  My stay here has been too short, I have to com…        stay   Sing
70                          She bought a pretty dress.       dress   Sing
71   This swiss knife is very useful when you travel.       knife   Sing

[72 rows x 3 columns]
```

```
[17]: df_verb = pd.DataFrame(list_df_verb, columns=["Phrase", "Question", "Lemma",␣
       ↪"Answer", "D1", "D2", "D3",
                                                  "F_Answer", "F_D1", "F_D2",␣
       ↪"F_D3"])
      df_verb.head(10)
```

```
[17]:                                              Phrase  \
      0  The beauty of the landscape struck the travell…
      1          Nobody knows the truth about this affair.
      2  In a dictatorship, freedom of expression is li…
      3                 He did not help you out of kindness.
      4                        His wickedness had no limits.
      5                 His elegance impressed the assembly.
      6  There is a big difference between the western …
      7                                  He has high ideals.
      8  He was struck by the modernity of the undergro…
      9  The quality of his work was acknowledged by th…

                                              Question        Lemma  \
      0   The beauty of the landscape … the travellers.       strike
      1           Nobody … the truth about this affair.         know
      2      In a dictatorship, freedom of expression …        limit
      3                 He did not … you out of kindness.         help
      4                       His wickedness … no limits.         have
      5                     His elegance … the assembly.      impress
      6  There … a big difference between the western…           be
      7                                  He … high ideals.         have
      8        He … by the modernity of the underground.       strike
      9           The quality of his work … by the jury.  acknowledge

                 Answer           D1            D2            D3  \
      0          struck       strike      striking       striked
      1           knows       knowed         knows        knowed
      2      is limited        limit       limited       limited
      3            help        helps       helping        helped
      4             had        haved         haves          have
      5       impressed    impressed     impressed       impress
      6              is           be           bed           bes
      7             has        haved        having          have
      8      was struck      striked        strike       strikes
      9  was acknowledged  acknowledged  acknowledging  acknowledges

                          F_Answer                          F_D1  \
      0                  past tense                    infinitive
      1  3rd person singular present                    past tense
      2             past participle                    infinitive
      3                  infinitive  3rd person singular present
```

```
4                    past tense                  past tense
5                    past tense             past participle
6  3rd person singular present                  infinitive
7  3rd person singular present             past participle
8               past participle                  past tense
9               past participle             past participle

                           F_D2                         F_D3
0            present participle                  past tense
1  3rd person singular present             past participle
2                    past tense             past participle
3            present participle                  past tense
4  3rd person singular present                  infinitive
5                    past tense                  infinitive
6                    past tense  3rd person singular present
7            present participle                  infinitive
8                    infinitive  3rd person singular present
9            present participle  3rd person singular present
```

[18]:
```
#df_verb.to_csv("English_MCQ.csv")
```

[ ]:
```
```