This notebook aims to demonstrate my bridge knowledge between Ophthalmology and Medical Text Processing.

This is done through a code example whose purpose is to prepare a medical text (such as a retinography report) for the application of semantic analysis or clustering tools.

The code below is a mere prototype without clinical validation

In [ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# by: Rafael_Scherer, MD, PhD
# date: 20/12/22
# version ='1.1'
```

In [15]:
```
# Libs
from time import time
start_time = time()

from nltk import word_tokenize
from functools import wraps
from types import FunctionType, MethodType
from re import search
from spacy import displacy  # Visualizador de relação entre palavras
from spacy import load
from nltk import data
from nltk import RSLPStemmer
from csv import reader


data.path.append("/nltk_data")

nlp = load('en_core_web_md')
end_time = time()
run_time = round(end_time - start_time, 5)
print(f'Libs import time: {run_time}')

#Global variables
DEBUG = True
TRIGGER = 0.1
PUNCTUATION = r'!#$&()*@[\]^_`{|}~'

"""
# StopWords
nltk.download('stopwords')
STOPWORDS = nltk.corpus.stopwords.words('english')

# Maintains terms possibly relevant to image reports
keep = ['but', 'or', 'against', 'between', 'into','before', 'after', 'above', 'below','up', 'down', 'in', 'out','over', 'under','no', 'nor', 'not','don', "don't",
```

```
STOPWORDS = [w for w in STOPWORDS if w not in keep]
"""

STOPWORDS = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves',


with open('sources/ICD.csv', 'r', encoding='utf-8') as f:  # ICD-10 Database
    CIDS = list(reader(f, delimiter=";"))


# Decorator for timing runtime
def timefunc(func):
    """
    timefunc is a decorator for printing function execution time
    with accumulated execution time greater than TRIGGER

    :param func: decorated function
    :return: func
     """

    @wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time()
        ret = func(*args, **kwargs)
        end_time = time()
        run_time = round(end_time - start_time, 5)
        wrapper.calls += 1  # n of executions

        # print _time of exec._ or _n. exec. * time of exec._ from func with time superior -> "trigger"
        if run_time > TRIGGER or run_time * wrapper.calls > TRIGGER:
            print(f"Execution of: {func.__qualname__} -- Time: {run_time} seconds -- N of exec.: {wrapper.calls}")
        return ret

    wrapper.calls = 0
    return wrapper


def double_spaces(txt):
    """
    double_spaces takes a text string and returns its version without double spaces or spaces at the beginning and end
    :param txt: input text
    :return: text without double spaces
    """
    adjusted = " ".join(txt.split())

    return adjusted


def double_symbols(txt):
    """
    double_symbols receives a text in string and returns its version without duplication of points, commas and dashes
```

```
    :param txt: input text
    :return: text without double symbols
    """
    adjusted = txt
    duplicate = True
    while duplicate is True:
        adjusted = adjusted.replace(',,', ',')
        adjusted = adjusted.replace('..', '.')
        adjusted = adjusted.replace('--', '-')
        adjusted = adjusted.replace('//', '/')
        if ',,' in adjusted or '..' in adjusted or '--' in adjusted or '//' in adjusted:
            duplicate = True
        else:
            duplicate = False
    return adjusted


def rem_punc(txt):
    """
    rem_punc takes a text string and returns its version without punctuation (eg: "?/!%$")
    :param txt: input text
    :return: string with inconvenient punctuation changed to "."
    """

    without_punc = txt.translate(str.maketrans('', '', PUNCTUATION))

    return without_punc


class MetaClasse(type):
    # Print when initializing the metaclass
    def __new__(mcs, cls, bases, classdict):
        new_cls = super().__new__(mcs, cls, bases, classdict)

        # key is attribute name and val is attribute value in attribute dict
        # Adds the timefunc decorator to all methods of the class
        for key, val in classdict.items():
            if DEBUG is True:
                if isinstance(val, FunctionType) or isinstance(val, MethodType):
                    setattr(new_cls, key, timefunc(val))
        return new_cls

    # Print for audit when calling a Class
    def __call__(cls, *args, **kwds):
        if DEBUG is True:
            print('Calling Class: ', str(cls))
            # print('__call__ *args=', str(args))
        return type.__call__(cls, *args, **kwds)


class Texto(object, metaclass=MetaClasse):
```

```
    """
    Texto Class processes grouped words
    """
    def __init__(self, texto: str):
        # Lower case + Remove punctuation marks + Double spaces + Double Symbols
        self.texto = double_symbols(double_spaces(rem_punc(texto.lower())))
        self.tokenized = word_tokenize(self.texto, language='english')

    def datas(self):
        """
        datas takes a text string and returns its version without data that may contain any
        data in different date/time formats
        :return: text without data in date format, List of Match Object Regex

        *Apply after numerical anonymization of the anon_numerico function
        """

        # Wanted numbers
        datas = [
            '([\d]{1,2}\W?(Jan(uary)?|Feb(ruary)?|Mar(ch)?|Apr(il)?|May|Jun(e)?|Jul(y)?|Aug(ust)?|Sep(tember)?|Oct(ober)?|Nov(ember)?|Dec(ember)?)\W[\d]{4})',
            '([\d]{1,2}\W?(jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dec(ember)?)\W[\d]{4})',
            '([\d]{1,2}\W?(Jan(uary)?|Feb(ruary)?|Mar(ch)?|Apr(il)?|May|Jun(e)?|Jul(y)?|Aug(ust)?|Sep(tember)?|Oct(ober)?|Nov(ember)?|Dez(ember)?))',
            '([\d]{1,2}\W?(jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dez(ember)?))',
            '((Jan(uary)?|Feb(ruary)?|Mar(ch)?|Apr(il)?|May|Jun(e)?|Jul(y)?|Aug(ust)?|Sep(tember)?|Oct(ober)?|Nov(ember)?|Dez(ember)?)\W[\d]{4})',
            '((jan(uary)?|feb(ruary)?|mar(ch)?|apr(il)?|may|jun(e)?|jul(y)?|aug(ust)?|sep(tember)?|oct(ober)?|nov(ember)?|dez(ember)?)\W[\d]{4})',
            '[\d]{1,2}\W?[\d]{1,2}\W?[\d]{4}', '[\d]{1,2}\W?[\d]{1,2}\W?[\d]{2}']


        regex = [datas]

        new = self.texto
        mos = []
        for i in range(0, 10):
            for q in range(0, len(regex)):
                for r in regex[q]:
                    mo = search(r, new)
                    if mo is not None:
                        new = new[:mo.start()] + '****' + new[mo.end():]
                        mos.append(mo)

        return new, mos


    def remove_stopwords(self):
        removed = []
        for item in self.tokenized:
            if item not in STOPWORDS:
                removed.append(item)

        return removed
```

```python
def lemalize(self, complete=False):
    """
    lemalize is the function that transforms the words in a text into their canonical form

    :param complete: if complete is True, it generates the complete report, if not, it just lemalize
    :return: list of words in their canonical form + list of parts of speech
    + entities present in the text + relationships[start position in the text, word, related word, start position in the text]
    """

    doc = nlp(self.texto)
    texto = self.texto
    lemmas = []
    gramatic = []
    relation = []
    for token in doc:
        gramatic.append(token.pos_)
        if token.pos_ == 'VERB':
            lemmas.append(token.lemma_)
        else:
            lemmas.append(token.orth_)

    if complete is True:
        displacy.render(doc, style='dep', jupyter=True)
        tok_l = doc.to_json()['tokens']

        for t in tok_l:
            head = tok_l[t['head']]
            relation.append(
                [t['start'], texto[t['start']:t['end']], t['dep'], texto[head['start']:head['end']], head['start']])
            # print(f"'{texto[t['start']:t['end']]}' is {t['dep']} of '{texto[head['start']:head['end']]}'")

        return lemmas, gramatic, doc.ents, relation

    else:
        return lemmas


def stemming(self):
    """
    stemming is a tool for admin use to make words reduced to their stem

    *Use tokens to be transformed preferentially after lemalization
    :return: final word reduced to its stem
    """

    stemmer = RSLPStemmer()
    st = []
    for palavra in self.tokenized:
        if palavra.isupper() is True:  # Does not stem acronyms
            st.append(palavra)
```

```
            else:
                st.append(stemmer.stem(palavra))

        return st


    def icd(self):
        """
        icd takes a text string and returns its version without data that may contain any
        ICD-10 facilitating semantic analysis and extracting tags
        :return: text without ICD10, List of Match Object Regex
        """

        # Wanted
        cid_format = ['[a-z]\W?[0-9]{2}\W?[0-9]?']

        regex = [cid_format]

        new = self.texto
        mos = []
        desc = []
        for i in range(0, 10):
            for q in range(0, len(regex)):
                for r in regex[q]:
                    mo = search(r, new)
                    if mo is not None:
                        achado = mo.group()
                        adjusted = ""
                        for pos, char in enumerate(achado):  # Fits the found cid to the search format
                            if pos == 0:
                                adjusted = adjusted + char
                            elif char.isalnum() is True:
                                adjusted = adjusted + char
                        for cid in CIDS:
                            if cid[0].lower() == adjusted:  # Check if it is a valid ICD10
                                mos.append(mo)
                                desc.append(cid[1])
                                new = new[:mo.start()] + '**** ' + new[mo.end():]
        return new, mos, desc
```

```
Libs import time: 1.13066
```

Example text

In [36]:
```
report = "Diagnosis: Age-related macular degeneration exudative in the healing phase (H35.3). Report: Vitreoretinal interface without change in reflectivity;  Sen
report
```

Out[36]: "Diagnosis: Age-related macular degeneration exudative in the healing phase (H35.3). Report: Vitreoretinal interface without change in reflectivity; Sensorineural retina with atrophy of the outer retina! Attenuated foveal depression; Thickness of the sensorineural retina in the central subfield: 215 µm; RPE-Bruch's membrane complex with irregular fusiform hyperreflective line suggestive of neovascular membrane with fibrosis. Superficial and deep retinal vascularizations with irregularities; Presence of choroidal neovascularization. Last exam on 12/5/2020... "

Lower case + Remove punctuation marks + Double spaces + Double Symbols

In [25]:
```
proc = Texto(report)
proc.texto
```

```
Calling Class:  <class '__main__.Texto'>
```

Out[25]: "diagnosis: age-related macular degeneration exudative in the healing phase h35.3. report: vitreoretinal interface without change in reflectivity; sensorineural retina with atrophy of the outer retina attenuated foveal depression; thickness of the sensorineural retina in the central subfield: 215 µm; rpe-bruch's membrane complex with irregular fusiform hyperreflective line suggestive of neovascular membrane with fibrosis. superficial and deep retinal vascularizations with irregularities; presence of choroidal neovascularization. last exam on 12/5/2020."

Remove and identify dates

In [27]:
```
proc.datas()
```

Out[27]: ("diagnosis: age-related macular degeneration exudative in the healing phase h35.3. report: vitreoretinal interface without change in reflectivity; sensorineural retina with atrophy of the outer retina attenuated foveal depression; thickness of the sensorineural retina in the central subfield: 215 µm; rpe-bruch's membrane complex with irregular fusiform hyperreflective line suggestive of neovascular membrane with fibrosis. superficial and deep retinal vascularizations with irregularities; presence of choroidal neovascularization. last exam on ****.",
 [<re.Match object; span=(547, 556), match='12/5/2020'>])

Remove and identify ICD-10

In [28]:
```
proc.icd()
```

Out[28]: ("diagnosis: age-related macular degeneration exudative in the healing phase **** . report: vitreoretinal interface without change in reflectivity; sensorineural retina with atrophy of the outer retina attenuated foveal depression; thickness of the sensorineural retina in the central subfield: 215 µm; rpe-bruch's membrane complex with irregular fusiform hyperreflective line suggestive of neovascular membrane with fibrosis. superficial and deep retinal vascularizations with irregularities; presence of choroidal neovascularization. last exam on 12/5/2020.",
 [<re.Match object; span=(75, 80), match='h35.3'>],
 ['Degeneration of macula and posterior pole'])

Remove less important words

In [32]:
```
print(proc.remove_stopwords())
```

```
['diagnosis', ':', 'age-related', 'macular', 'degeneration', 'exudative', 'in', 'healing', 'phase', 'h35.3', '.', 'report', ':', 'vitreoretinal', 'interface', 'without', 'change', 'in', 'reflectivity', ';', 'sensorineural', 'retina', 'atrophy', 'outer', 'retina', 'attenuated', 'foveal', 'depression', ';', 'thickness', 'sensorineural', 'retina', 'in', 'central', 'subfield', ':', '215', 'µm', ';', 'rpe-bruch', "'s", 'membrane', 'complex', 'irregular', 'fusiform', 'hyperreflective', 'line', 'suggestive', 'neovascular', 'membrane', 'fibrosis', '.', 'superficial', 'deep', 'retinal', 'vascularizations', 'irregularities', ';', 'presence', 'choroidal', 'neovascularization', '.', 'last', 'exam', '12/5/2020', '.']
```

Stemming

In [33]:
```
print(proc.stemming())
```

```
['diagnosil', ':', 'age-related', 'macul', 'degeneration', 'exudativ', 'in', 'the', 'healing', 'phas', 'h35.3', '.', 'report', ':', 'vitreoret', 'interfac', 'wit
hout', 'chang', 'in', 'reflectivity', ';', 'sensorine', 'retin', 'with', 'atrophy', 'of', 'the', 'out', 'retin', 'attenuated', 'fove', 'depression', ';', 'thickn
es', 'of', 'the', 'sensorine', 'retin', 'in', 'the', 'centr', 'subfield', ':', '215', 'µm', ';', 'rpe-bruch', "'s", 'membran', 'complex', 'with', 'irregul', 'fus
iform', 'hyperreflectiv', 'lin', 'suggestiv', 'of', 'neovascul', 'membran', 'with', 'fibrosil', '.', 'superfic', 'and', 'deep', 'ret', 'vascularizatiom', 'with',
'irregulariti', ';', 'presenc', 'of', 'choroid', 'neovascularization', '.', 'last', 'ex', 'on', '12/5/2020', '.']
```

Lemmatization

In [34]:
```
print(proc.lemalize())
```

```
['diagnosis', ':', 'age', '-', 'relate', 'macular', 'degeneration', 'exudative', 'in', 'the', 'healing', 'phase', 'h35.3', '.', 'report', ':', 'vitreoretinal',
'interface', 'without', 'change', 'in', 'reflectivity', ';', 'sensorineural', 'retina', 'with', 'atrophy', 'of', 'the', 'outer', 'retina', 'attenuate', 'foveal',
'depression', ';', 'thickness', 'of', 'the', 'sensorineural', 'retina', 'in', 'the', 'central', 'subfield', ':', '215', 'µm', ';', 'rpe', '-', 'bruch', "'s", 'me
mbrane', 'complex', 'with', 'irregular', 'fusiform', 'hyperreflective', 'line', 'suggestive', 'of', 'neovascular', 'membrane', 'with', 'fibrosis', '.', 'superfic
ial', 'and', 'deep', 'retinal', 'vascularizations', 'with', 'irregularities', ';', 'presence', 'of', 'choroidal', 'neovascularization', '.', 'last', 'exam', 'o
n', '12/5/2020', '.']
```