SEMAFOR'S DATA STATISTICAL PARSING DOCUMENTATION

Summary

Context

This parsing starts from a point where we have semafor's output as a Json file for each abstract of our corpus, this Json have the following structure: each line has a Json object with the attribute "frames", this attribute have an Array that brings us information about the identified frames, each position in the Array is a identified frame, this information comes in two attributes: "target" and "annotationSets".
"target" is where general information about the frame is, his name and also the name and position (counting spaces) of the lexical unit that called it, in the attributes "start", "end" and "text".
"annotationSets" is where we will find information about the result of semafor's analysis, like "rank", "score", that is the score that semafor gave to the frame*, and "frameElements" that are the frame elements identified for that frame.
The first line in this Json files brings the frames identified in the name of the journal, the ones identified for the name of the abstract will be find at the second line, the rest of the file have the ones identified for the abstract itself.
The objective of this parsing is to extract quantitative and qualitative statistical analysis for this semafor results, initially, these are:

1. Occurrence of frames in the corpus, for all abstracts, per field, per discipline and per journal
2. Occurrence of Lexical Units in the corpus, for all abstracts, per field, per discipline, per journal and also for each frame (considering frame subdivisions)

Tools

The tools used to extract this analysis were the programming language Python and the Jupyter Notebook environment. Python is well known for having great performance in processing large amounts of data, this will be helpful in dealing with our corpus, Jupyter allow us to have a practical environment to generate our statistical info, with the concept of live code that can be used to run code from a point where data have already been treated, without needing to run all the code every time.

Python Modules

The following Python libraries were used to facilitate some tasks in the code:

Os:
: This module is used to be possible to pass some commands to the operational system, in our context it was used to handle directories

Sys:
: This module is used to access some operational system variables, in our context it was used also to handle directories

Matplotlib:
: This module can be used to many things (check documentation linked), we used it to deal with charts plotting.

Json:
: This module is used to handle Json formatted text and objects, we used it to handle the Json objects that Semafor outputs.

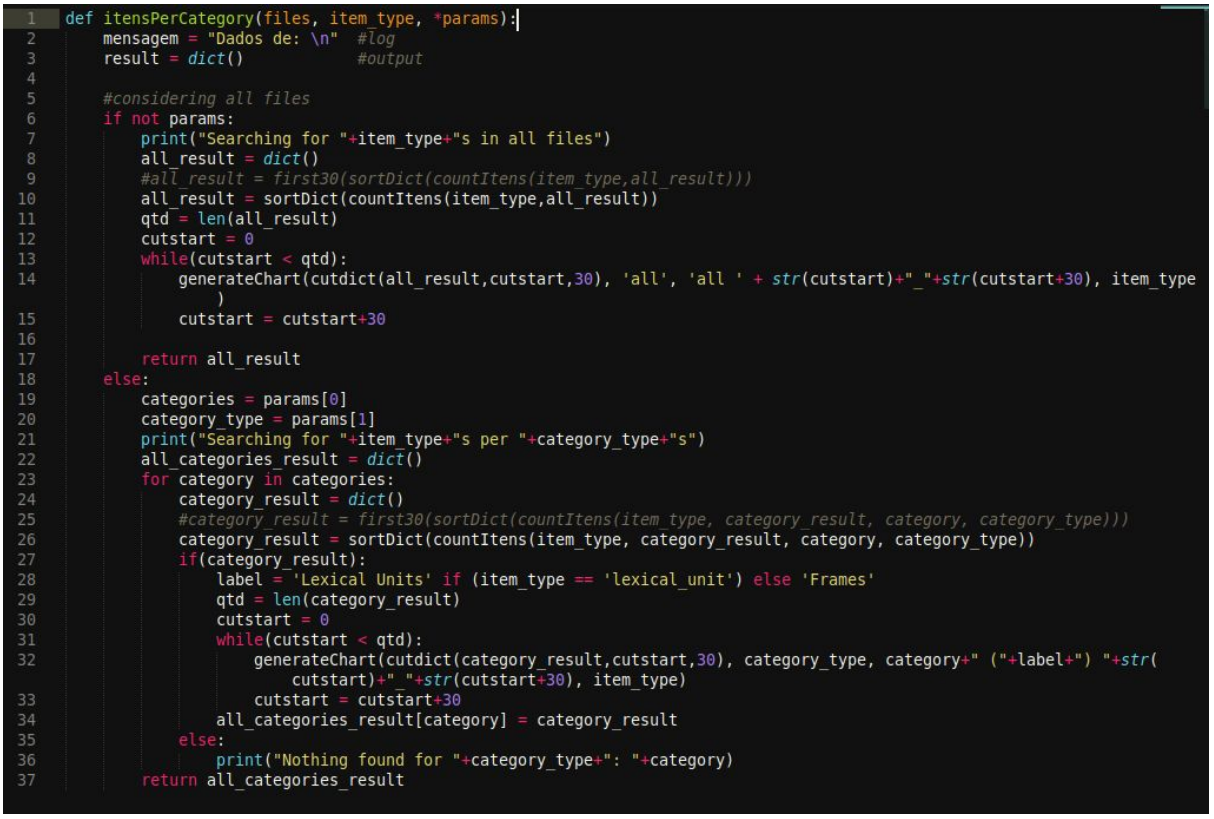
Math:
: This module handles math calculations and we used it exactly for that.

Inflection:
: This module can be used to deal with linguistics problems in text processing, we used it to get the singular of words.

Code

Here are some of the most important parts of the code.

itensPerCategory function:
This function can receive a minimum of two e maximum of four parameters, the first two are an array with the file paths of semafor output and the item type that it will search for (frames or lexical units), the other two are optional parameters that, if used, will be an array with the categories (names of journals, fields or disciplines) and the category type (by journal, field or discipline). Basically it will go through the files that match the categories using the **countItems** function to count the occurence of each item, respecting the item type.



```
def itensPerCategory(files, item_type, *params):
    mensagem = "Dados de: \n"  #log
    result = dict()            #output

    #considering all files
    if not params:
        print("Searching for "+item_type+"s in all files")
        all_result = dict()
        #all_result = first30(sortDict(countItens(item_type,all_result)))
        all_result = sortDict(countItens(item_type,all_result))
        qtd = len(all_result)
        cutstart = 0
        while(cutstart < qtd):
            generateChart(cutdict(all_result,cutstart,30), 'all', 'all ' + str(cutstart)+"_"+str(cutstart+30), item_type
                )
            cutstart = cutstart+30

        return all_result
    else:
        categories = params[0]
        category_type = params[1]
        print("Searching for "+item_type+"s per "+category_type+"s")
        all_categories_result = dict()
        for category in categories:
            category_result = dict()
            #category_result = first30(sortDict(countItens(item_type, category_result, category, category_type)))
            category_result = sortDict(countItens(item_type, category_result, category, category_type))
            if(category_result):
                label = 'Lexical Units' if (item_type == 'lexical_unit') else 'Frames'
                qtd = len(category_result)
                cutstart = 0
                while(cutstart < qtd):
                    generateChart(cutdict(category_result,cutstart,30), category_type, category+" ("+label+") "+str(
                        cutstart)+"_"+str(cutstart+30), item_type)
                    cutstart = cutstart+30
                all_categories_result[category] = category_result
            else:
                print("Nothing found for "+category_type+": "+category)
        return all_categories_result
```

countItems function:

```
def countItems(item_type, items_counted, *params):
    if params:
        category = params[0]
        category_type = params[1]
    #traversing all files
    for file_name in files:
        file_location = path+'/'+file_name
        abstract = getInfoFromFile(file_location)
        file_content = open(file_location,"r").readlines()

        #traversing all lines in each file (each line has a json with information about frames)
        for line in file_content:
            jsonline = json.loads(line)

            #verifying each item
            for frame in jsonline.get('frames'):

                if(item_type == 'frame'):
                    frame_name = frame.get('target').get('name').lower()
                    if(frame_name[len(frame_name)-1] == 's' or frame_name[len(frame_name)-4:len(frame_name)-1] == 'ies'):
                        frame_name =inflection.singularize(frame_name)
                    if params:
                        if(abstract.get(category_type) == category):
                            if(items_counted.get(frame_name) != None):
                                items_counted[frame_name] += 1
                            else:
                                items_counted[frame_name] = 1
                    else:
                            if(items_counted.get(frame_name) != None):
                                items_counted[frame_name] += 1
                            else:
                                items_counted[frame_name] = 1

                if(item_type == 'lexical_unit'):
                    for span in frame.get('target').get('spans'):
                        lexical_unit = span.get('text').lower()
                        if(lexical_unit[len(lexical_unit)-1] == 's' or lexical_unit[len(lexical_unit)-4:len(lexical_unit)-1] == 'ies'):
                            lexical_unit =inflection.singularize(lexical_unit)
                        if params:
                            if(abstract.get(category_type) == category):
                                #se ja existir o index, soma
                                if(items_counted.get(lexical_unit) != None):
                                    items_counted[lexical_unit] += 1
                                #senao cria o index sem modificacoes
                                else:
                                    items_counted[lexical_unit] = 1
                        else:
                            if(items_counted.get(lexical_unit) != None):
                                items_counted[lexical_unit] += 1
                            else:
                                items_counted[lexical_unit] = 1
    return items_counted
```

getTitle function:

This function is used to get the title of the abstract from the raw text file.

```
def getTitle(title):
    txtFiles = list()
    for (dirpath, dirnames, filenames) in os.walk("../raw"):
        txtFiles += [os.path.join(dirpath, file) for file in filenames]

    for txtFile in txtFiles:
        if txtFile.endswith(".txt"):
            for line in list(open(txtFile)):
                if(line.startswith("TI ")):
                    nextline = list(open(txtFile))[list(open(txtFile)).index(line)+1];
                    if(not nextline.startswith("SO ")):
                        fileTitle = line.replace("TI ","").replace("\n"," ").replace("  "," ")+" ".join(nextline.split())
                    else:
                        fileTitle = line.replace("TI ","").replace("\n"," ")
                    if(fileTitle == title):
                        return fileTitle

```

getJournal and searchJournal functions:
This functions get the Journal of an abstract based on the file name

```
def searchJournal(file1,file2):
    retorno = ""
    for j in getAllJournals():
        if(file1.find(j) != -1):
            retorno = file1
        elif(file2.find(j) != -1):
            retorno = file2
        elif( (file2.find(j) != -1) and (file1.find(j) != -1) ):
            retorno = "False"
    return retorno

def getJournal(file):
    file = file.split('.seg.json')[0]
    file1 = file.split('_')[-3].replace('-',' ').upper()
    file2 = file.split('_')[-2].replace('-',' ').upper()
    journal = searchJournal(file1,file2)
    return journal
```

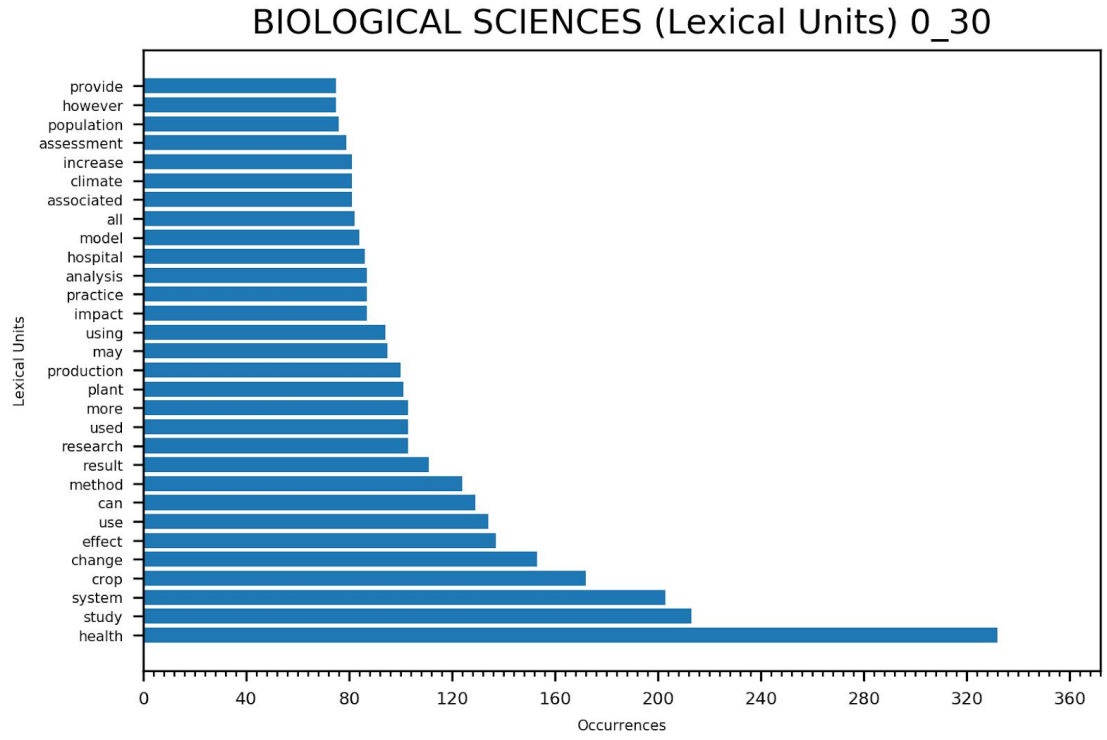
getInfoFromFile function:
This function gets abstract information (journal, field, discipline) from a json file 'file'

```
def getInfoFromFile(file):
            fields = dictionaries.dictionaryFields()
            disciplines = dictionaries.dictionaryDisciplines()
            journals = getAllJournals()
            journal = ""

            journal = getJournal(file)

            field = fields.get(journal)
            discipline = disciplines.get(field)
            abstract = dict()
            abstract['journal'] = journal
            abstract['field']   = field
            abstract['discipline'] = discipline
            return abstract
```

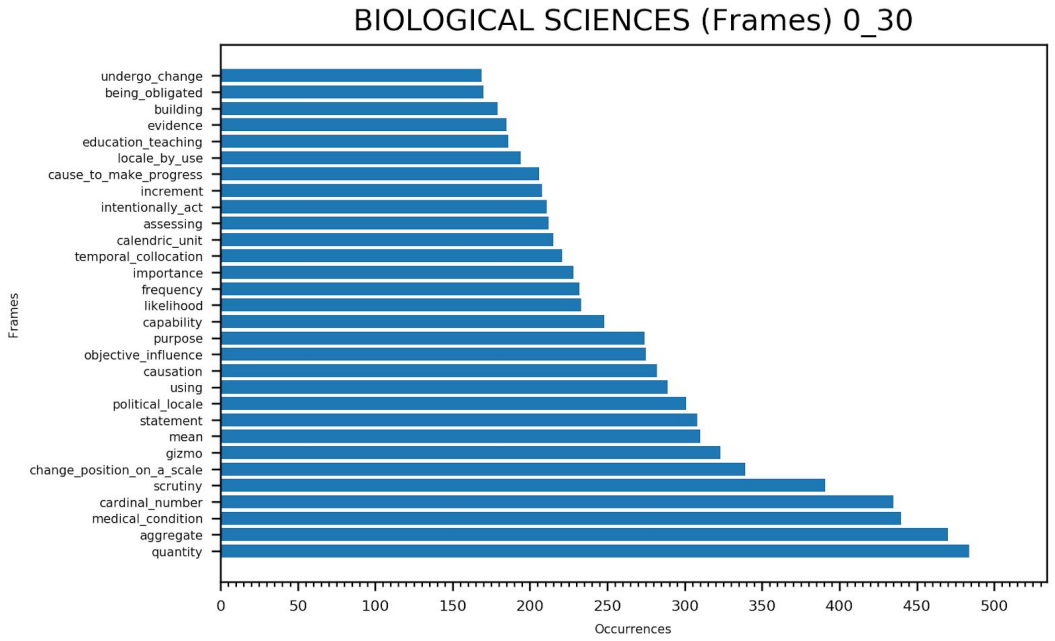
Results

The objective of this work is to generate statistical analysis about semafor's output, the complete code and the results we already have will be available in our GitHub repository, here's some examples of our results:

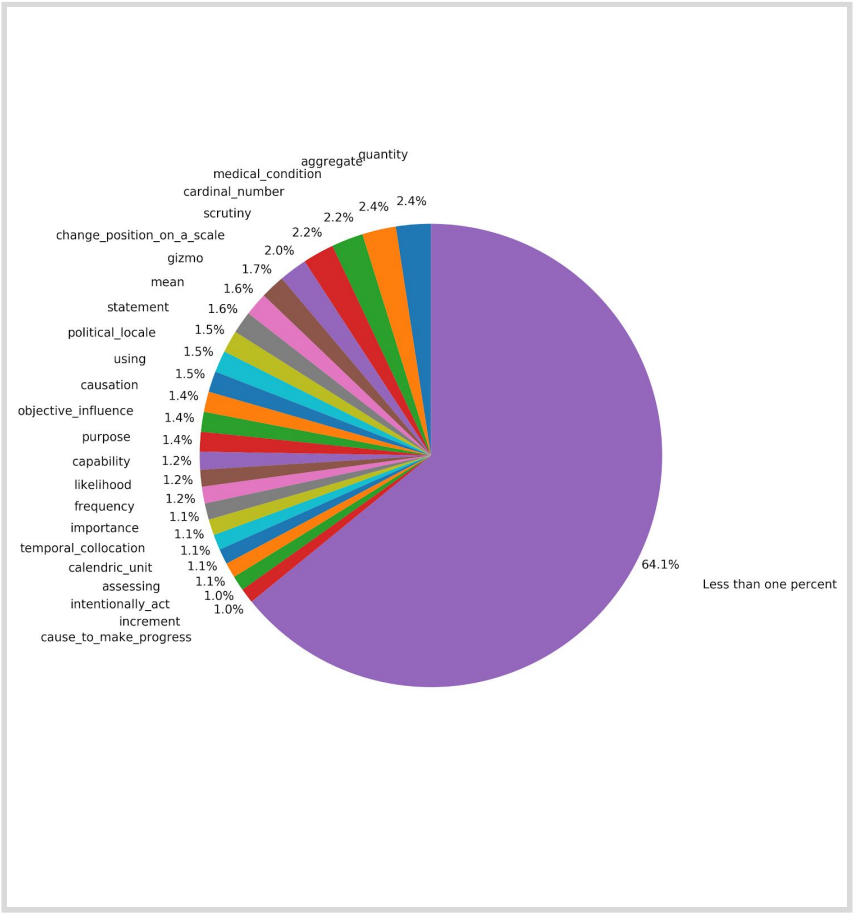
Counting of the thirty lexical units that most evoke frames in Biological Sciences (discipline):



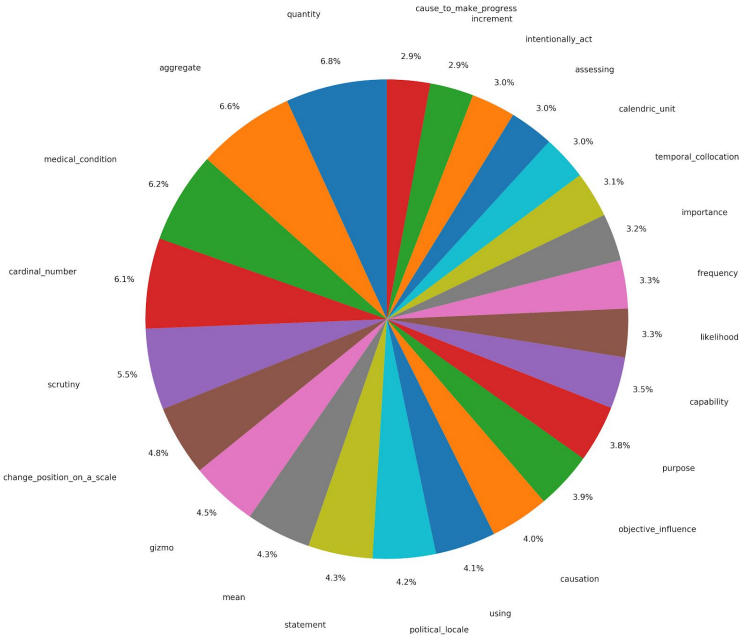
Counting of the thirty frames that are more evoked in Biological Sciences (Discipline):



Percentage of the thirty more evoked frames in Biological Sciences (Discipline), grouping frames with less than one percent of occurrence:



Percentage of the more evoked frames in Biological Sciences (Discipline), considering only frames with more than one percent of occurrence:


quantity
6.8%
cause_to_make_progress
2.9%
increment
2.9%
intentionally_act
3.0%
assessing
3.0%
calendric_unit
3.0%
temporal_collocation
3.1%
importance
3.2%
frequency
3.3%
likelihood
3.3%
capability
3.5%
purpose
3.8%
objective_influence
3.9%
causation
4.0%
using
4.1%
political_locale
4.2%
statement
4.3%
mean
4.3%
gizmo
4.5%
change_position_on_a_scale
4.8%
scrutiny
5.5%
cardinal_number
6.1%
medical_condition
6.2%
aggregate
6.6%