How to Use A Parallelism Identifier to Assist Corpus-based Discourse Analysis

I. Basic information about the tool:

1. **Main function:** This identifier is trained to identify the following 4 types of inner-sentence level parallelism in a text.
   - **type 1: …A cc B…**e.g. Yet every so often, the oath is taken amidst **gathering clouds** and **raging storms**. (Obama, 2009)
   - **type 2: …A, B…** e.g. Homes have been lost, **jobs shed, businesses shuttered**. (Obama, 2009)
   - **type 3: …A cc B cc C…** e.g. Americans are **generous and strong and decent**… (Bush, 2001)
   - **type 4: …A, B (,) cc C…:** e.g. A new national pride will **stir our souls, lift our sights**, and **heal our divisions**. (Trump, 2017)

   ***Note:** "**cc**" refers to coordinating words, namely "and", "or", "but", "while (as in conj.)" or part of a paired conjunction, e.g. "not only… but also…"; "either… or…"; "**A, B, C**" refer to the parallel elements with similar grammatical structures.
2. **Train/Test data:** 3 manually annotated American inaugural speeches
3. **Performance:** Cross validated precision, recall and f-measure scores (approx.) are respectively 0.90, 0.82, 0.85 (Logistic Regression model)
4. **Functions provided (6):**
   tag_file(path, clf); getParallelism (path, Format); parallelism_Dispersion(path); para_NonPara_ratio(path, percentage); process_file_sent(path, withTag); evaluate(path_manual, path_pred, lablel, show_FP, show_FN)

II. Necessary Downloads and Installation:

1. Download "Parallelism_Identifier" zip file and decompress it.

You should be able to find the following files:

- Training data: 3 annotated American inaugural speeches from 2001, 2009, 2017
- 2 Python-scripts (.ipynb file): *Parallelism_Identifier_Mac_JingyingWang.ipynb* and *Parallelism_Identifier_Windows_JingyingWang.ipynb* respectively for ***Mac*** users and ***Windows*** system users.
- 2 .txt sample files (TED-Talks) for trying out different functions provided.
- 3 .csv sample files: *predicted_lr.csv* (based on Logistic Regression model) *and predicted_nb.csv* (based on Naive Bayes Algorithm) contain assigned tags, *manual_tag.csv* contains manually assigned tags. (These files are specifically for trying out the function: *evaluate*)

2. (If not done before) Download the Stanford POS-Tagger package. It will be used for preprocessing of the texts.

link: https://nlp.stanford.edu/software/tagger.shtml#Download (Training data used version: 3.9.2

2018-10-16 (full))

3. (If not done before) Download and install Java Development Kit (jdk). The Stanford POS-Tagger needs Java environment. (Training data used version: "12.0.1" 2019-04-16)

link: https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html

III. Adjust the Python-script to Your Computer

Before running the code and using functions, you need to adjust some parameters according to YOUR operating system and YOUR file system. Please note that some parameters such as *path* can be written in different ways according to the operating system. (See corresponding Python-script for specific examples.)

- *java_path* (***Mac*** users please **ignore** this parameter!): Enter the path to YOUR Java jdk. Note that for some Windows systems, escape character "\" is needed (see screenshot below).

```
java_path = "C:\\Program Files\\Java\\jdk1.8.0_191\\bin\\java.exe"
os.environ["JAVAHOME"] = java_path
```

- *jar & model:* Enter the paths of *stanford-postagger-3.9.2.jar* file and the tagger model to be used (here used *english-left3words-distsim.tagger*).

```
from nltk import word_tokenize
from nltk.tag.stanford import StanfordPOSTagger

# Enter the paths of Stanford POS Tagger .jar file as well as the model to be used
jar = "/Users/Shared/stanford-postagger-full-2018-10-16/stanford-postagger-3.9.2.jar"
model = "/Users/Shared/stanford-postagger-full-2018-10-16/models/english-left3words-distsim.tagger"

# Instantiate an English pos-tagger using the jar and model defined above
pos_tagger_en = StanfordPOSTagger(model, jar, encoding = "utf-8")
```

- *Path for the Training data*: There are 2 blocks where the path for *Training_data* folder (extracted from the *Parallelism_Identifier.zip* file) is need twice (see the screenshot below). Replace the given paths (Altogether 4 places) according to YOUR directory. You can search for "data_set_raw1" and "data_set_raw2" in the python script to find these 2 code blocks.

```
data_set_raw1 = []
for file in os.listdir("American-Inaugural-Address-Corpus/Tagged"):
    if file.endswith("csv"):
        with open("American-Inaugural-Address-Corpus/Tagged/" + file, encoding =
            reader = csv.reader(f, delimiter = ",")
            rows = [row for row in reader]
            data_set_raw1.extend(rows[1:])    # the first row is the header ["sent
print(len(data_set_raw1))
data_set = [(sent, tag) for [sent, tag] in data_set_raw1]
```

IV. Try Out Functions

After adjusting all parameters, you can use the following **6** functions to assist corpus-based discourse analysis:

(1) tag_file(path, clf)

Function: Process all sentences in the text (chosen sentence closers are: . *! ? : ;*) and write them with its corresponding tag ("Non-parallelism" or "Parallelism") into a csv file.

@param *path:* The path to the Plain Text file to be analyzed (as string).

@param *clf:* One of the trained classifiers. Options are: **NB** (NaiveBayesClassifier from nltk); **KNC** (KNeighborsClassifier from sklearn); **LR** (LogisticRegression from sklearn); **SVC** (LinearSVC from sklearn)

```
# Train classifiers in different ways
NB = nltk.NaiveBayesClassifier.train(featuresets)
KNC = KNC.fit(X, y)
LR = LR.fit(X, y)
SVC = SVC.fit(X, y)
```

After running the script, you can find the **tagged csv file** in the same folder as your original .txt file and the filename ends with "*tagged.csv*". The following screenshot shows how the content looks like.

testted.txttagged

sentences	tag
So when I was a little girl , a book sat on the coffee table in our living room , just steps from our front door .	Non-parallelism
And the living room is a first impression .	Non-parallelism
Ours had white carpet and a curio of my mother 's most treasured collectibles .	Non-parallelism
That room represented the sacrifices of generations gone by who , by poverty or by policy , could n't afford a curio of collectibles let alone a middle class house to put them in .	Parallelism
That room had to stay perfect .	Non-parallelism

(2) getParallelism (path, Format)

Function: Print out sentences which are tagged as "Parallelism" directly in Jupyter Notebook and write these sentences (when requested) into a new file according to your need.

@param *path*: Path to the automatically tagged file (as string).

@param *Format:* Options are *"txt" and "xlsx". (*Datatype: *string).*

- If you choose to just take a brief look at sentences identified as "Parallelism", but NOT to write them into a new file, you can simply pass an empty string ("") as the second parameter (see screenshot below).

```
getParallelism("test_classifier/phase2t_lr/3.txttagged.csv","")
```

```
1 I grew up in the west of Ireland , wedged between four brothers , two older than me and two
younger than me .
2 And those issues stayed with me and guided me , and in particular , when I was elected the
first woman President of Ireland , from 1990 to 1997 .
3 I led trade delegations here to the United States , to Japan , to India , to encourage inve
stment , to help to create jobs , to build up our economy , to build up our health system , o
ur education -- our development .
```

- If you choose NOT to print the sentences and only need tagged files, you can disable the printing function by deleting the codes within the red squares shown in the following screenshot.

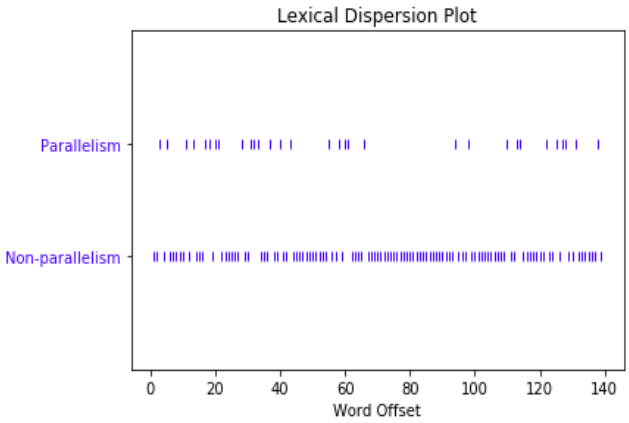
```
def getParallelism (path, Format):
    with open (path,  encoding = "utf-8") as f:
        reader = csv.reader(f, delimiter = ",")
        rows = [row for row in reader]
        parallel = []
        for [sent, tag] in rows:
            if tag == "Parallelism":
                parallel.append(sent)

        n = 1
        for s in parallel:
            print(n, s)
            n += 1
    if Format  == "txt":
```

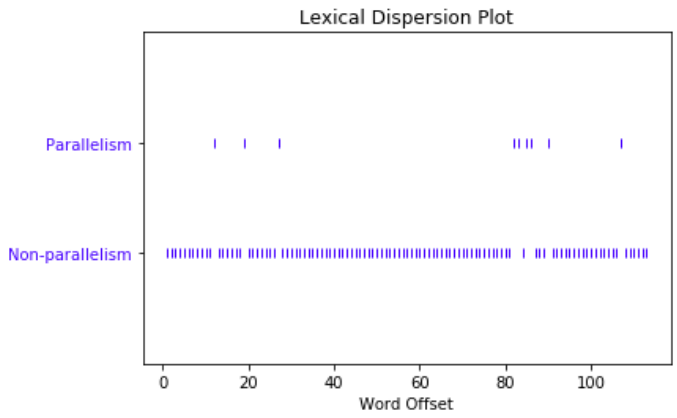
(3) parallelism_Dispersion(path)

Function: Plot the distribution of *Parallelism* sentences and *Non-parallelism* sentences. With this function you can have an overall view about how the parallelisms are distributed in a text and whether they are frequent used or not in an intuitive way.

@param *path:* Path to the automatically tagged file (as string).

This function treats each sentence as one unit regardless of the sentence length. The plotted graphics look like this (see next page):





According to the dispersion plot above, sentences with parallel structures are not homogeneously distributed throughout a text. Therefore, the following function is provided to get the ratio between sentences with/without parallelism in different positions of a text. With this function, quantitative analysis is possible.

(4) para_NonPara_ratio(path, percentage)

Function: Return 3 values respectively for the ratio between sentences with / without parallelism in text-initial, text-medial and text-final positions and print out the corresponding information. (Example see screenshot below)

@param *path:* Path to the automatically tagged file (as string).

@param *percentage:* Enter a float number which specifies the percentage of the text (based on number of sentences) considered as the text-initial part. This percentage also applies for the text-final part. The rest of the text is the text-medial part.

```
para_NonPara_ratio("test_classifier/phase2t_lr/3.txttagged.csv", 0.15)
```

```
 21  sentences in text_initial part.
 98  sentences in text_medial part.
 21  sentences in text_final part.
In text-initial part, Parallelism/Non-parallelism ratio is:  0.5384615384615384
In text-medial part, Parallelism/Non-parallelism ratio is:  0.225
In text-final part, Parallelism/Non-parallelism ratio is:  0.4

(0.5384615384615384, 0.225, 0.4)
```

Note: If you want to apply the classifier at a large scale, it is recommended that first of all evaluate different classifiers and select the most suitable one for your own corpus, especially when the texts are not *speeches*. The simplest way is to review the output for at least one typical text of the corpus to see whether most of them meet you demand or not. Another more

valid way is to compute the precision, recall and f-measure scores based on manually assigned tags. The following functions could be helpful in this situation.

(5) process_file_sent(path, withTag)

Function: Segment the given text (Sentence closers are: . ! ? : ;) and write the processed sentence into a new csv file (One sentence per line).

@param *path:* The path to the Plain Text file to be processed (as string).

@param *withTag:* Enter a boolean value to specify whether the words in the output file are combined with PoS-tags or not.

If enter *True:* Sentences in the output file looks like this:

I_PRP 'm_VBP an_DT astronomer_NN who_WP builds_VBZ telescopes_NNS ._.

If enter *False:*

I 'm an astronomer who builds telescopes.

The original name of the output csv file ends with "processed_*sent*"(without PoS-tag) or "processed_*sent(tag)*"(with PoS-tag). Please CHANGE the name before doing further manual annotation work in order to differentiate the automatically annotated file and the manually annotated file.

(6) evaluate(path_manual, path_pred, lablel, show_FP, show_FN)

Function: Get precision, recall and f-measure scores for a given text to evaluate a classifier. False Positives and False Negatives can be printed out if specified.

@param *path_manual:* Path to the manually annotated **csv** file.

@param *path_pred:* Path to the automatically annotated **csv** file. (The file should be generated by using the function ***tag_file(path, clf)*** mentioned above. The name of which ends with "*tagged.csv*")

@param *label*: label (tag) name: "t" or "f". If you do not use other tag names respectively for sentence with / without parallelism. You need to change all "t" s and "f" s into the corresponding names you used in the function-defining block (see screenshot on the next page)

@param *printFP:* Enter a boolean value to specify whether you want the False Positives printed or not.

@param *printFN:* Enter a boolean value to specify whether you want the False Negatives printed or not.

* Please pay attention to the **order** of two boolean values.

```
# Label name should be "t"(parallelism) or "f"(non-parallelism)
def evaluate(path_manual, path_pred, lablel, show_FP, show_FN):
    manual_tags = []
    pred_tags = []
    with open(path_manual, encoding = "utf-8" ) as f:
        reader = csv.reader(f, delimiter = ",")
        rows = [row for row in reader]
        for [sent, tag] in rows[1:]:
            manual_tags.append(tag)
    with open(path_pred, encoding = "utf-8") as f:
        reader = csv.reader(f, delimiter = ",")
        rows = [row for row in reader]
        for [sent, tag] in rows:
            if tag == "Parallelism":
                pred_tags.append("t")
            if tag == "Non-parallelism":
                pred_tags.append("f")

        if show_FP == True:
            print("False Positives:\n")
            for i in range(len(manual_tags)):
                if manual_tags[i] == "f" and pred_tags[i] == "t":
                    print(rows[i+1][0], "\n")     # Take the header row into account
        if show_FN == True:
            print("\nFalse Negatives:\n")
            for i in range(len(manual_tags)):
                if manual_tags[i] == "t" and pred_tags[i] == "f":
                    print(rows[i+1][0],"\n")
```

Now you can try different output files provided by different classifiers and find out the most suitable classifier for your corpus. The screenshot below suggest that the classifier based on Logistic Regression model is better than the classifier based on Naive Bayes algorithm.

```
evaluate("test_classifier/phase2t_lr/combine_m.csv","test_classifier/phase2t_lr/lr_tagged.csv","t",False, False)
```

```
Altogether  256 manually assigned tags.
Altogether  256 predicted tags.

Precision=0.78 Recall=0.83 F_Measure=0.80
```

```
evaluate("test_classifier/phase2t_lr/combine_m.csv","test_classifier/phase2t_nb/nb_tagged.csv","t",False, False)
```

```
Altogether  256 manually assigned tags.
Altogether  256 predicted tags.

Precision=0.72 Recall=0.83 F_Measure=0.77
```