Activités du Lundi 4 mars au Vendredi 8 mars :

Fait :

- Modification de l'input dans dataframe_id.py pour faire les id artificiels → prendre en compte les fichiers .txt plutôt que .tml
- Tokenization
- Récupération du contexte (contexte des 4 mots d'avant/d'après + contexte du POS des 4 mots d'avant/d'après) (corpus .txt)
- Analyse des lexèmes ayant comme POS 'RB' (adverbe)

- <u>Tokenization</u>

La tokenization a été indispensable pour les tâches où l'on utilise les fichiers .txt en input, c'est-à-dire :

a) la création des identifiants artificiels
b) la récupération du contexte des events/timex/signaux

Dans les fichiers .txt nous avons des timex qui ressemblent à : a#t1-minute#t1-and#t1-a#t1-half#t1.

Lorsque l'on tokenize ce genre de séquence nous obtenons :

Tokenization : 'a', '#', 't1-minute', '#', 't1-and', '#', 't1-a', '#', 't1-half', '#', 't1'

nltk.word_tokenize() ne tokenize pas les tirets et les points s'ils sont dans un mot. La tokenization par simple split() n'est pas possible car si un mot est accolé à un signe de ponctuation, le split() ne les sépareront pas. C'est pour cette raison que l'outil utilisé pour la tokenization est nltk, nous avons des dates séparées par des tirets et les identifiants de documents comportent un point, nltk permet de garder ces éléments en totalité sans les tokenizer au point ou au tiret.

Cependant, dans notre cas, nous avons 't1-minute' qui n'est pas censé se tokenizer de cette façon. Nous avons réfléchi à comment réunir tous les éléments. Si nous avons un # nous prenons l'index du # -1 et l'index du # + 1 ce qui nous donne :

Regroupement avec # : 'a#t1-minute', 't1-minute#t1-and', 't1-and#t1-a', 'a#t1-half', 'half#t1'

Les éléments ne se rassemblent pas comme voulu. L'élément qui nous pose problème est le '-' qui n'est pas tokenizé. Nous avons réfléchi à un symbole qui est tokenizé par nltk mais pas présent dans le corpus. Nous avions trouvé le '$' mais finalement, ils sont présents dans le corpus lorqu'il est évoqué des prix en dollars. Si nous avons par exemple to$29 ça va fausser les identifiants artificiels au niveau du n° de token (to$29 : id=1 / to : id=1 ; $: id=2 ; 29 : id=3).

Nous avons ensuite constaté que dans le corpus il n'y a aucune occurrence du symbole **>** et que nltk le tokenize. Ce qui nous donne :

Dans le fichier .txt : 'a#t1>minute#t1>and#t1>a#t1>half#t1'

Tokenization : 'a', '#', 't1', '>', 'minute', '#', 't1', '>', 'and', '#', 't1', '>', 'a', '#', 't1', '>', 'half', '#', 't1'

➔ tous les éléments sont bien tokenizés.

Enfin il suffit de rassembler les éléments à l'aide du # (index du # -1 et l'index du # + 1), ce qui donne :

Regroupement avec # : 'a'#'t1', '>', 'minute#t1', '>', 'and#t1', '>', 'a#t1', '>', 'half#t1'

Et de rassembler les éléments à l'aide du > (index du > -1 et l'index du > + 1) :
Regroupement avec > : 'a#t1>minute#t1>and#t1>a#t1>half#t1'

- **Modification du séparateur d'unités inter timex/event/signaux dans le script transformation_TML_to_TXT.py**

Le séparateur qui avait été déterminé pour séparer les unités dans les timexs était le tiret. Ce qui nous donnait a#t1-minute#t1-and#t1-a#t1-half#t1. Le séparateur a été modifié par **>**, qui nous donne a#t1>minute#t1>and#t1>a#t1>half#t1.

- **Modifications dans le script dataframe_id.py**

Modification de l'input : avant nous prenions en compte les fichiers .tml pour créer les identifiants artificiels et nous avions constaté que si un timex contenait plusieurs unités lexicales (ex : « a minute and a half »), celles-ci étaient séparées.

- Résultat dans le csv dataframe_id_ponctuation.csv avec le traitement des fichiers **.tml** :

word	idSentence	idWord	idTimex
a	1	1	
minute	1	2	
and	1	3	
a	1	4	
half	1	5	

Pour remédier à cette difficulté, l'utilisation des fichiers .txt permet de traiter les unités lexicales qui composent les timex réunies par des symboles :

- Le **#** permet de relier chaque unité composant le timex à son identifiant (tid)
- Le **>** permet de relier les unités entre elles (dans le cas où le timex comprend plusieurs mots)

- Résultat dans le csv dataframe_id_ponctuation.csv avec le traitement des fichiers **.txt** :

word	idSentence	idWord	idTimex
a#t1>minute#t1>and#t1>a#t1>half#t1	1	1	1

- **Analyse du contexte (POS des 4 mots d'avant/ d'après l'event/timex/signal)**

Dans l'objectif de détecter les marqueurs de négation, les adverbes de modalité et les modaux dans l'entourage des event/timex/signaux, l'analyse des POS est nécessaire. Pour effectuer le part-of-speech tagging nous avons utilisé Nltk (nltk.pos_tag()).

Nltk ne permet pas de tagger les marqueurs de négations comme étant des négations, de même pour les adverbes de modalité. Ces derniers sont annotés comme étant des adverbes (RB). En revanche, les modaux sont bien annotés (MD).

Pour identifier les négations et les adverbes de modalité, nous avons fait, pour chacun, une liste de termes :

RB_neg = ['not', 'no', 'neither', 'nor', 'n\'t', 'never', 'off'] (à voir ensemble)

RB_MD = ['probably', 'possibly', 'evidently', 'allegedly', 'surely', 'certainly', 'apparently'] (à voir ensemble)

Ensuite on peut faire quelque chose comme ça :

```
l = [0,0]    # liste de deux éléments
for context in posContext:
    for tuplWordPos in context:
        if tuplWordPos[1] == 'RB':
            if tuplWordPos[0].lower() in RB_neg:
                l[0] = True
            elif tuplWordPos[0].lower() in RB_MD:
                l[1] = True
        elif tuplWordPos[1] == 'MD':
            l[1] = True
        else:
            l[0] = False
            l[1] = False
```

en sortie on obtient, par exemple :

[('he', 'PRP'), (**'could', 'MD'**), **("n't", 'RB'**), ('yet', 'RB')] [**True**, **True**]

[(',', ','), ('as', 'IN'), ('investors', 'NNS'), (**'apparently', 'RB'**), ('that', 'IN'), ('the', 'DT'), ('thrift', 'NN'), (**'would', 'MD'**)][**False**, **True**]