IA 717: CHAI & fairness: linguistics of AI ethics charters & manifestos

Student Version

Project Supervisor

Maria Boritchev maria.boritchev@telecom-paris.fr

Project student

Josephine Bernard josephine.bernard@telecom-paris.fr
Laury Magne laury.magne@telecom-paris.fr
Dan Hayoun dan.hayoun@telecom-paris.fr
Nicolas Allègre nicolas.allegre@telecom-paris.fr

Year 2024-2025

Context and objectives

> The recent years have seen a surge of initiatives with the goal of defining what “ethical” artificial intelligence would or should entail, resulting in the publication of various charters and manifestos discussing AI ethics; these documents originate from academia, AI industry companies, non-profits, regulatory institutions, and the civil society. The contents of such documents vary wildly, from short, vague position statements to verbatims of democratic debates or impact assessment studies. As such, they are a marker of the social world of artificial intelligence, outlining

> the tenets of different actors, the consensus and dissensus on important goals, and so on. Mapping AI Ethics: a meso-scale analysis of its charters and manifestos, M. Gornet et al, 2024.

The objective of this project to study the linguistic specificities of a corpus of charters and manifestos of AI ethics, called `MapAIE`. We will study the semantic structure of sentences from MapAIE using the Abstract Meaning Representation (AMR) abstraction method. In particular, we will focus on the usage and the environment of the word *fairness*, in order to see its context and the (possibly different?) meanings associated with its usage.

Evaluation

The project is evaluated through a presentation with a report and your completed project. Grades will be partly individual and partly collective.

Table of Contents

0 - Code Python initial

0.1) Première partie imports et fonctions globales

In [41]:
```
# python -m pip install matplotlib numpy scipy sklearn tabulate
# !pip install Unidecode wordcloud spacy
# !python -m spacy download en_core_web_sm

# Global import
```

```python
import itertools
import math
import os
import re
import shutil
import string
import typing
from collections import defaultdict
from collections.abc import Iterable
from itertools import islice
from pathlib import Path
from pprint import pprint

import matplotlib.pyplot as plt
import numpy as np
import numpy.typing as npt
import spacy
from scipy.stats import norm  # type: ignore[import-untyped]
from sklearn.feature_extraction.text import CountVectorizer
from tabulate import tabulate
from unidecode import unidecode
from wordcloud import WordCloud

from Corpus import Corpus

EXEC_PYTHON_IS_NOT_EXIST = False  # Execute action of git repo (dll, parse, prepro, corpus)
EXEC_MAPAIE_DLL = False
EXEC_MAPAIE_PARSE = False
EXEC_MAPAIE_LANG = False
EXEC_MAPAIE_PREPRO = True
EXEC_MAPAIE_CORPUS = True
PATH_DATA_FOLDER = '../data'
PATH_LOG_FOLDER = '../log'
PATH_DATA_TXT = '../data/txts'
PATH_DATA_DOCS = '../data/docs'
PATH_DATA_PREPROCESSED = '../data/preprocessed'
PATH_DATA_CORTEX = '../data/corpus_cortext'
PATH_DATA_IRAMUTEQ = '../data/corpus_iramuteq'
FILENAME_DATA_IRAMUTEQ = 'corpus.txt'
FILENAME_DATA_LANG = 'corpus_lang.csv'
FILENAME_DATA_LANG_PREPRO = 'corpus_lang_preprocessing.csv'
PATH_DATA_FILE_LANG = os.path.join(PATH_DATA_FOLDER, FILENAME_DATA_LANG)
PATH_DATA_FILE_IRAMUTEQ = os.path.join(PATH_DATA_IRAMUTEQ, FILENAME_DATA_IRAMUTEQ)
TYPE_METHOD = ['cortex', 'iramuteq', 'txt']
CHARSET = 'UTF-8'
```

```python
def take(n: int, iterable: Iterable) -> list[any]:
    """Return the first n items of the iterable as a list."""
    return list(islice(iterable, n))
#end def take

def dict_revert(x: dict[str, any], sort: bool=False) -> list[list[str, any]]:
    tmp = x
    if sorted is True:
        tmp = dict(sorted(x.items(), key=lambda item: item[1], reverse=True))

    return [[list(x.keys())[i], list(x.values())[i]] for i in range(len(x))]


def preparation_data(exec_actions: dict[str, bool] | bool, force: bool = False):
    """Vérifie la présence des documents et exécute les actions associées.

    :param dict[str, bool] | bool exec_actions: liste des actions à faire
        exec_actions = bool : alors si True => exec_actions['all']=True
        'all' = toutes les actions :
        'dll' = mapaie-1 : télécharge les documents (dl_docs.py) dans data/docs
        'parse' = mapaie-2 : parse les PDF et HTML vers des TXT (parse_docs.py) dans data/txts
        'prepro' = mapaie-3 : exécute les étape NLP de pré-process (preprocess.py) sur les TXT dans data/preprocessed
        'lang' = mapaie-3bis : trouve la langue sur les TXT (create_corpus_before_lang.py)
        'corpus' = mapaie-4 : crée les corpus avec les algo (Iramuteq et Cortex)
    :param bool force: force l'exécution de l'action même si les données existes.
    """
    if type(exec_actions) != dict:
        if exec_actions is True:
            exec_actions = {}
            exec_actions['all'] = True
        else:
            exec_actions = {}

    if exec_actions.get('all') is True:
        exec_actions['dll'] = True
        exec_actions['parse'] = True
        exec_actions['prepro'] = True
        exec_actions['lang'] = True
        exec_actions['corpus'] = True

    if exec_actions.get('corpus') is True:
        if not os.path.exists(PATH_DATA_CORTEX) or not os.path.exists(PATH_DATA_IRAMUTEQ):
            exec_actions['prepro'] = True
    if exec_actions.get('prepro') is True and not os.path.exists(PATH_DATA_PREPROCESSED):
        exec_actions['parse'] = True
    if exec_actions.get('lang') is True and not os.path.exists(PATH_DATA_TXT):
```

```
        exec_actions['parse'] = True
    if exec_actions.get('parse') is True and not os.path.exists(PATH_DATA_TXT):
        exec_actions['dll'] = True

    # Actions :
    print(exec_actions)
    actions = ['dll', 'parse', 'prepro', 'lang', 'corpus']
    actions_path = {'dll': PATH_DATA_DOCS, 'parse': PATH_DATA_TXT, 'prepro': PATH_DATA_PREPROCESSED, 'lang': PATH_DATA_TXT, 'corpus': [P
    exec_python = {'dll': 'dl_docs.py', 'parse': 'parse_docs.py', 'prepro': 'preproccess.py', 'lang': 'create_corpus_before_lang.py', 'c
    for action in actions:
        if exec_actions.get(action) is True:
            print(f'Action à faire pour {action}')
            folders = []
            if type(actions_path[action]) == list:
                folders = actions_path[action]
            else:
                folders = [actions_path[action]]
            if force is True:
                if action =='lang':
                    # os.remove(PATH_DATA_FILE_LANG)
                    print(f'\tRemove fichier {PATH_DATA_FILE_LANG}')
                else:
                    for folder in folders:
                        print(f'\tRemove folder {folder}')
                        # shutil.rmtree(folder)
                        pass

            to_do = False
            for folder in folders:
                to_do = to_do or not os.path.exists(folder)
            if to_do:
                print(f'\tAction : {exec_python[action]}')
                # python dl_docs.py
                # python parse_docs.py
                # python preproccess.py
                # python create_corpus_before_lang.py
                # python create_corpus.py -t themes.json -d data/preprocessed/ -m iramuteq
                # python create_corpus.py -t themes.json -d data/preprocessed/ -m cortext
                # import dl_docs  # ??
                pass
```

In [5]:
```
liste_actions = {'dll': EXEC_MAPAIE_DLL, 'parse': EXEC_MAPAIE_PARSE, 'prepro': EXEC_MAPAIE_PREPRO, 'lang': EXEC_MAPAIE_LANG, 'corpus': E
if EXEC_PYTHON_IS_NOT_EXIST:
    preparation_data(liste_actions)

# CORTEX n'est absolument pas optimisé !!! => Il copie les TXT x fois dans les catégories !
```

```
#   =>ça prends beaucoup trop de place et ce n'est pas scalable.
#   =>il faut générer une liste de document pour chaque tag et ce référer soit au dossier TXT,
#   soit n'avoir les documents en un seul exemplaire.
```

0.2) Deuxième partie chargement des données

`Corpus` : Classe unique permettant de charger et de manipuler le corpus des données de manière uniforme.

Dans la suite du notebook la variable `list_corpus` contient les types de corpus :

- cortex
- iramuteq
- txt

Utilisation :

```
list_corpus['txt'].get_tags()
# >>> set()
list_corpus['cortex'].get_tags()
# >>> {'Trust', 'Solidarity', 'Responsibility', 'Transparency', 'Beneficence', 'Dignity', 'Privacy', 'Non-maleficence',
'Freedom and autonomy', 'Justice and fairness', 'Sustainable'}
list_corpus['iramuteq'].get_tags()
# >>> {'Trust', 'Solidarity', 'Responsibility', 'mapaie', 'Transparency', 'Beneficence', 'Dignity', 'Privacy', 'Non-
maleficence', 'Freedom and autonomy', 'Justice and fairness', 'Sustainable'}
```

In [8]:
```
# Chargement de tout les corpus
list_corpus = {method: Corpus(method) for method in TYPE_METHOD}

# Exemple :
# >>> list_corpus['txt'].get_tags()
# set()
# >>> list_corpus['cortex'].get_tags()
# {'Trust', 'Solidarity', 'Responsibility', 'Transparency', 'Beneficence', 'Dignity', 'Privacy', 'Non-maleficence', 'Freedom and autonom
# >>> list_corpus['iramuteq'].get_tags()
# {'Trust', 'Solidarity', 'Responsibility', 'mapaie', 'Transparency', 'Beneficence', 'Dignity', 'Privacy', 'Non-maleficence', 'Freedom a
```

1 - Général

Our main corpus is called *MapAIE - Mapping AI Ethics*. It is a collection of 436 common charters and manifestos around artificial intelligence and AI ethics. The corpus' datasheet provides detailed information about the collection process and the contents of the corpus.

1.1 QUESTION 1.1

1.1 Question: Using the git repository https://gitlab.telecom-paris.fr/tiphaine.viard/mapaie, build the data locally. How many documents are there in the `./pdf` folder? in the `./txt` folder?

1.1 Answer:

- Il n'y a pas de dossier `pdf` mais un dossier `docs` contenant la récupération d'Internet des documents au format HTML ou PDF.
- Le dossier `txts` correspond aux documents ayant été parsés en texte brut.
- Le dossier `preprocessed` correspond aux textes après avoir subit une étape de NLP de découpage en token.
- Les dossiers `corpus_*` correspondent aux résultats de création de corpus textuel par les algorithme IRAMUTEQ et CORTEX.

Différents méthodes de comptage existent, soit par Python soit le plus simple via un simple `ls | wc -l` (Linux) :

```
(Get-ChildItem docs |Measure-Object).count
#    =>629
(Get-ChildItem docs\*.pdf |Measure-Object).count
#    =>307
(Get-ChildItem docs\*.html |Measure-Object).count
#    =>321
(Get-ChildItem txts |Measure-Object).count
#    =>625
```

In [21]:
```
list_dir = os.listdir(PATH_DATA_DOCS)
print("Nombre de document téléchargé (./docs) : ", len(list_dir))
print("\tNombre de HTML téléchargé (./docs) : ", len([x for x in list_dir if '.html' in x]))
print("\tNombre de PDF téléchargé (./docs) : ", len([x for x in list_dir if '.pdf' in x]))
print("Nombre de document parsé (./txts/) : ", len(os.listdir(PATH_DATA_TXT)))
```

```
Nombre de document téléchargé (./docs) :  624
        Nombre de HTML téléchargé (./docs) :  318
        Nombre de PDF téléchargé (./docs) :  306
Nombre de document parsé (./txts/) :  620
```

1.2 QUESTION 1.2

Let's explore the data. As you can see, documents in `./pdf` folder and documents in the `./txt` folder are the same, and the name formatting is such that `n.txt` contains plain text extracted from `n.pdf` .

1.2 Question: Why is it useful to have the data both in .pdf and .txt format? Cite one advantage and one drawback for linguistic analysis for each of the

formats.

1.2 Answer:

PDF/HTML :

- *D'un part, une grande partie non négligeable des documents scientifiques sont publiés en PDF. Mais des articles sont plus sur des sites web. D'où la nécessité de récupérer ces deux formats.*
- L'avantage, c'est que ces format, PDF et HTML, permettent de garder la mise en forme et les metadata (chapitre, indentation, etc)
- Leur inconvénient, c'est qu'ils sont difficiles à parser, les retours à la ligne ne sont pas ou mal détectés.

TXT :

- L'avantage c'est qu'ils sont bien plus faciles à parser/utiliser car les mots sont directement accéssible.
- L'inconvénient c'est la perte d'information lors de l'état de *parsing*. Il n'y plus de mise en forme ou de structure.

1.4 QUESTION 1.4

The following presents a non exhaustive list of keywords frequently enocountered in documents and discourse related to AI and ethics of AI:

- fairness;
- algorithmic fairness;
- bias;
- gdpr;
- discrimination;
- biometric;
- regulation;
- artificial general intelligence.

1.4 Question: among these keywords, which ones are the most frequent? Please provide visualizations and comment these visualisations.

1.4 Answer:

Différents méthodes de comptage existent, soit par Python soit le plus simple via un simple `cat |find | wc -w` (Linux) :

```
$list=('fairness', 'algorithmic fairness', 'bias', 'gdpr', 'discrimination', 'biometric', 'regulation', 'artificial general intelligence')
$list | %{ $tmp=(Get-Content .\data\txts\*.txt | Select-String -Pattern $_ -AllMatches).Matches.count;$a=$_+' : '+$tmp;$a }
```

```
# fairness : 3059
# algorithmic fairness : 72
# bias : 7753
# gdpr : 2793
# discrimination : 4037
# biometric : 2261
# regulation : 7378
# artificial general intelligence : 95
```

2 - Analyse sur les 3 corpus

2.1 - Data Presentation and Retrieval (multi)

We now want to know more about words and key-words used throughout the corpus. There are several terms that are widely used when writing about ethics of AI, and it is interesting from a linguistic point of view to see where and how these words occur.

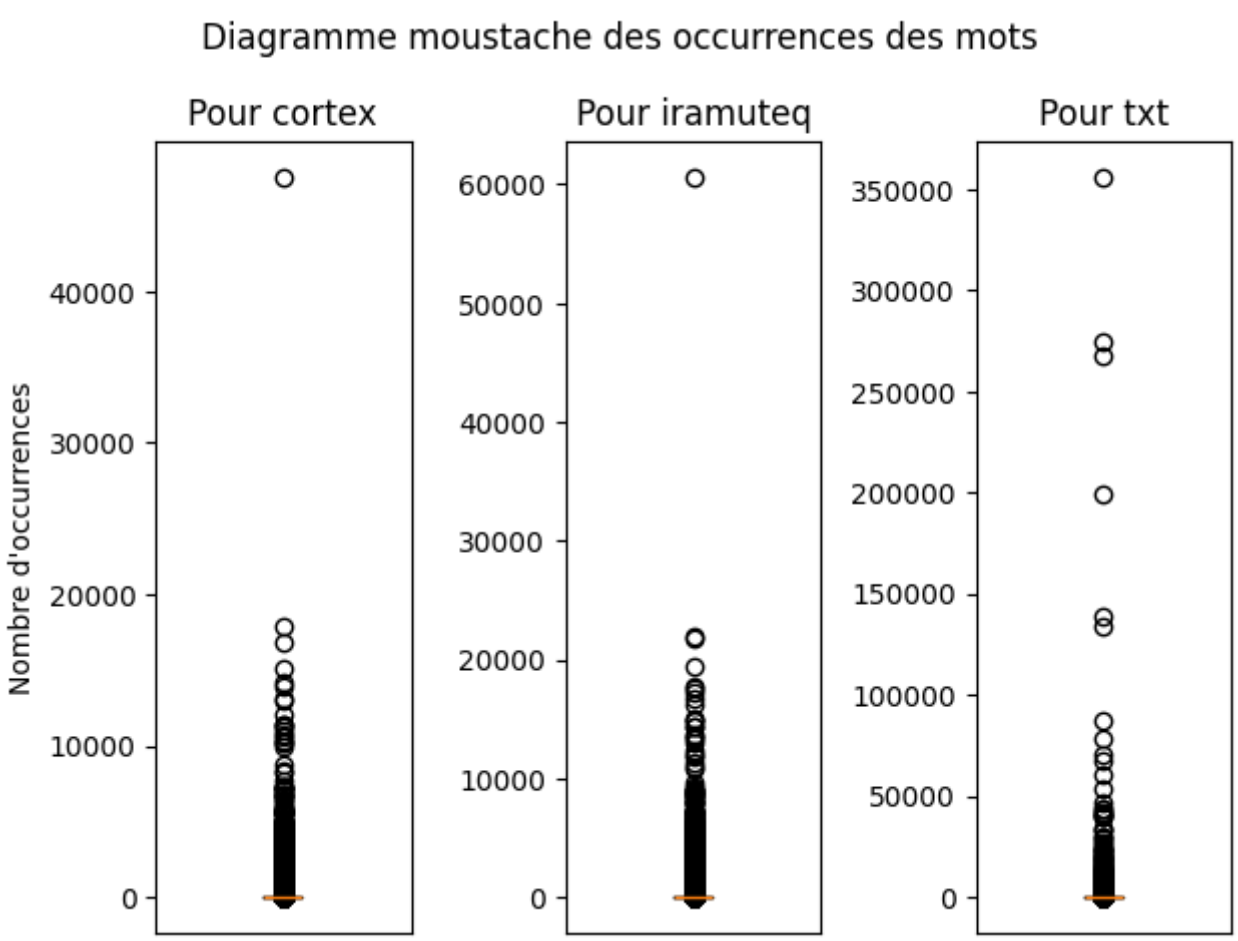
1.3 QUESTION 1.3

1.3 Question: what are the most frequent words, excluding stopwords, in the whole corpus? Please provide visualizations and comment these visualisations.

In [6]:
```
# data = list_corpus['txt']
info_model = {}
plt.close('all')
fig = plt.figure()
fig.suptitle('Diagramme moustache des occurrences des mots')
for i, method in enumerate(list_corpus):
    info_model[method] = {}
    data = list_corpus[method]
    model = data.type_model
    corpus = data.get_corpus()
    print(f"\n*** MODEL : {model}")
    print(f"{data.type_model} : Taille corpus = {len(corpus)}")

    # Sortir tous les tokens
    tokens = {}
    for word in corpus.split():
        if tokens.get(word) == None:
            tokens[word] = 0
        tokens[word] += 1
```

```
    # Supprimer tous les mots à 1
    copy = tokens.copy()
    for word in copy :
        if tokens[word] == 1:
            tokens.pop(word, None)

    # Supprimer tous les stopwords
    # On ne le fait pas pour ressortir plus d'information de comparaison entre les 3 corpus

    # Sortir le mot le plus utilisé
    cle_max = max(tokens, key=tokens.get)
    print(f"{data.type_model} : Mot le plus utilisé : {cle_max} = {tokens[cle_max]}")
    print(f"{data.type_model} : Nb initial mot : {len(copy)}, Nb mot >1 lettre et stopword : {len(tokens)}")

    # Extraire les occurrences et les mots
    occurrences_values = list(tokens.values())
    mots = list(tokens.keys())

    # Créer le diagramme en boîte (boxplot)
    ax = fig.add_subplot(1, len(list_corpus), i +1)
    ax.boxplot(occurrences_values)
    ax.set_title(f"Pour {model}")
    if i == 0:
        ax.set_ylabel("Nombre d'occurrences")
    ax.get_xaxis().set_visible(False)

    tokens = dict(sorted(tokens.items(), key=lambda item: item[1], reverse=True))
    info_model[method]['tokens'] = tokens

fig.tight_layout()
plt.show()
```

```
*** MODEL : cortex
cortex : Taille corpus = 30012540
cortex : Mot le plus utilisé : data = 47499
cortex : Nb initial mot : 116180, Nb mot >1 lettre et stopword : 62737

*** MODEL : iramuteq
iramuteq : Taille corpus = 40105251
iramuteq : Mot le plus utilisé : data = 60534
iramuteq : Nb initial mot : 154432, Nb mot >1 lettre et stopword : 81494

*** MODEL : txt
txt : Taille corpus = 65113631
txt : Mot le plus utilisé : the = 355298
txt : Nb initial mot : 473509, Nb mot >1 lettre et stopword : 199792
```



In [7]:
```python
NB_TOP = 10
top_x = {}
for model in list_corpus:
    tokens = info_model[model]['tokens']
    top_x[model] = {word: tokens[word] for word in take(NB_TOP, tokens)}

# pprint(top_x, sort_dicts=False)

headers = ['cortex', 'cortex', 'iramuteq', 'iramuteq', 'txt', 'txt']
tmp = []
for model in list_corpus:
    data_tmp = top_x[model]
    tmp.append(list(data_tmp.keys()))
    tmp.append(list(data_tmp.values()))

print(tabulate(np.array(tmp).T, headers=headers, tablefmt='pipe'))
```

cortex	cortex	iramuteq	iramuteq	txt	txt
data	47499	data	60534	the	355298
systems	17878	systems	21949	and	274848
use	16875	use	21791	of	267744
intelligence	15079	intelligence	19373	to	198824
also	14170	also	17698	in	138304
public	13926	human	17608	a	133383
system	13097	public	17303	for	87271
human	12998	system	16635	AI	78249
artificial	12003	artificial	16222	is	70804
research	11421	des	15060	that	67586

In [8]:
```
stats_info = {}
for model in list_corpus:
    tokens = info_model[model]['tokens']

    occurrences_values = list(tokens.values())
    mediane = np.median(occurrences_values)
    moyenne = np.mean(occurrences_values)
    ecart_type = np.std(occurrences_values)
    Q3 = np.percentile(occurrences_values,90)

    stats_info[model] = {'mediane': mediane, 'moyenne': moyenne, 'sig': ecart_type, 'Q3': Q3}

# pprint(stats_info, sort_dicts=False)
headers=list(stats_info[list(stats_info.keys())[0]].keys())
print(tabulate(stats_info.values(), showindex=list(stats_info.keys()), headers='keys', tablefmt='pipe'))
```

	mediane	moyenne	sig	Q3
cortex	5	56.8662	404.194	61
iramuteq	4	58.7168	462.474	55
txt	3	44.9017	1457.88	29

1.3 Answer:

On remarque que le mot data est bien plus utilisé que tous les autres. Une quinzaine d'autres points surélevés apparaissent.

Nous remarquons aussi, que la fréquence d'apparition des mots sont similaires quelques soit la méthode si les stop-word sont retiré. Ainsi, pour ces mots fréquents le pre-processing n'impacte pas le classement (ils ne sont pas retiré du corpus).

1.4 QUESTION 1.4

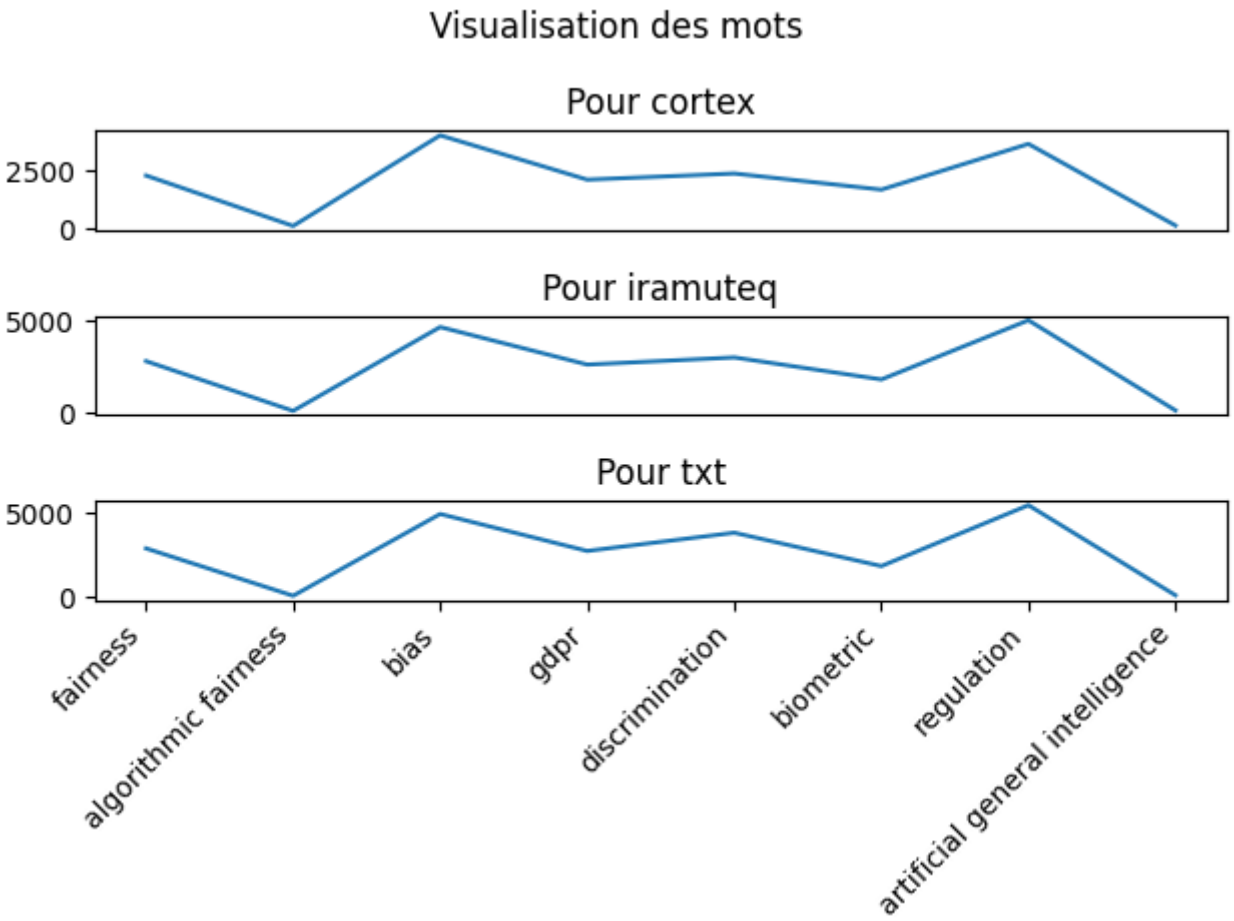
The following presents a non exhaustive list of keywords frequently enocountered in documents and discourse related to AI and ethics of AI:

- fairness;
- algorithmic fairness;
- bias;
- gdpr;
- discrimination;
- biometric;
- regulation;
- artificial general intelligence.

1.4 Question: among these keywords, which ones are the most frequent? Please provide visualizations and comment these visualisations.

In [14]:
```
list_word = ['fairness','algorithmic fairness','bias','gdpr','discrimination','biometric','regulation','artificial general intelligence'

ngram_freqs = {}
plt.close('all')
fig = plt.figure()
fig.suptitle('Visualisation des mots')
for i, method in enumerate(list_corpus):
    data = list_corpus[method]
    model = data.type_model
    corpus = list(data.data.values())

    # Faire les 1-3-gram
    vectorizer = CountVectorizer(ngram_range=(1, 3), min_df=3)
    X = vectorizer.fit_transform(corpus)  # Ajuster le vectorizer au corpus
    vocab = vectorizer.get_feature_names_out()  # liste de mot unique
    occurrences = np.asarray(X.sum(axis=0)).flatten()  # Fréquence de chaque n-gram
    vocabulaire_occurrences = dict(zip(vocab, occurrences))

    ngram_freqs[model] = vocabulaire_occurrences

    courbe = [vocabulaire_occurrences[x] if vocabulaire_occurrences.get(x) != None else 0 for x in list_word]
    ax = fig.add_subplot(len(list_corpus), 1, i +1)
    ax.plot(list_word, courbe)
    ax.set_title(f"Pour {model}")
    if i == 2:
        for label in ax.get_xticklabels():
            label.set_rotation(45)
            label.set_horizontalalignment('right')
    else:
        ax.get_xaxis().set_visible(False)
```

```
fig.tight_layout()
plt.show()

stats_mots = {}
tmp = []
for model in list_corpus:
    data_tmp = [ngram_freqs[model][word] if ngram_freqs[model].get(word) != None else 0 for word in list_word]
    tmp.append(data_tmp)
```



In [15]:
```
# Powershell :
# fairness : 3059
# algorithmic fairness : 72
# bias : 7753
# gdpr : 2793
# discrimination : 4037
# biometric : 2261
# regulation : 7378
```

```
# artificial general intelligence : 95
powershell_count = [3059, 72, 7753, 2793, 4037, 2261, 7378, 95]
tmp.append(powershell_count)
headers = list(list_corpus.keys()) + ['Powershell (txt)']
print(tabulate(np.array(tmp).T, showindex=list_word, headers=headers, tablefmt='pipe'))
```

```
|                                |   cortex |   iramuteq |   txt |   Powershell (txt) |
|:-------------------------------|---------:|-----------:|------:|-------------------:|
| fairness                       |     2275 |       2807 |  2892 |               3059 |
| algorithmic fairness           |       68 |         80 |    80 |                 72 |
| bias                           |     4035 |       4679 |  4934 |               7753 |
| gdpr                           |     2095 |       2619 |  2740 |               2793 |
| discrimination                 |     2364 |       3011 |  3822 |               4037 |
| biometric                      |     1658 |       1803 |  1841 |               2261 |
| regulation                     |     3659 |       5050 |  5457 |               7378 |
| artificial general intelligence |       87 |        106 |   109 |                 95 |
```

1.4 Answer:

	cortex	iramuteq	txt	Powershell (txt)
fairness	2275	2807	2892	3059
algorithmic fairness	68	80	80	72
bias	4035	4679	4934	7753
gdpr	2095	2619	2740	2793
discrimination	2364	3011	3822	4037
biometric	1658	1803	1841	2261
regulation	3659	5050	5457	7378
artificial general intelligence	87	106	109	95

- Nous constatons qu'en compte à la manière brute (shell OS), nous avons un nombre plus important.
  - Cela est dû à la non prise en charge de la case.
  - Ainsi, notre analyse avancée sur les données TXT devra prendre en compte la case suivant l'état final recherché.
- Nous constatons que l'algo Iramuteq a des valeurs proche de la manière brute.
  - Cela est dû que la totalité du document TXT est préservé (copie).
  - Mais comme une mise en place de tag spécifique est mis en place, certain document n'apparaisent plus dans les résultats de cette méthode.
- Nous constatons que l'algo Cortex a des valeurs moindre.
  - Cela est dû au fait que cette méthode utilise les données ayant déjà eux une pré-nettoyage, ce qui retire beaucoup de texte dont des mots

potentiellement recherché.

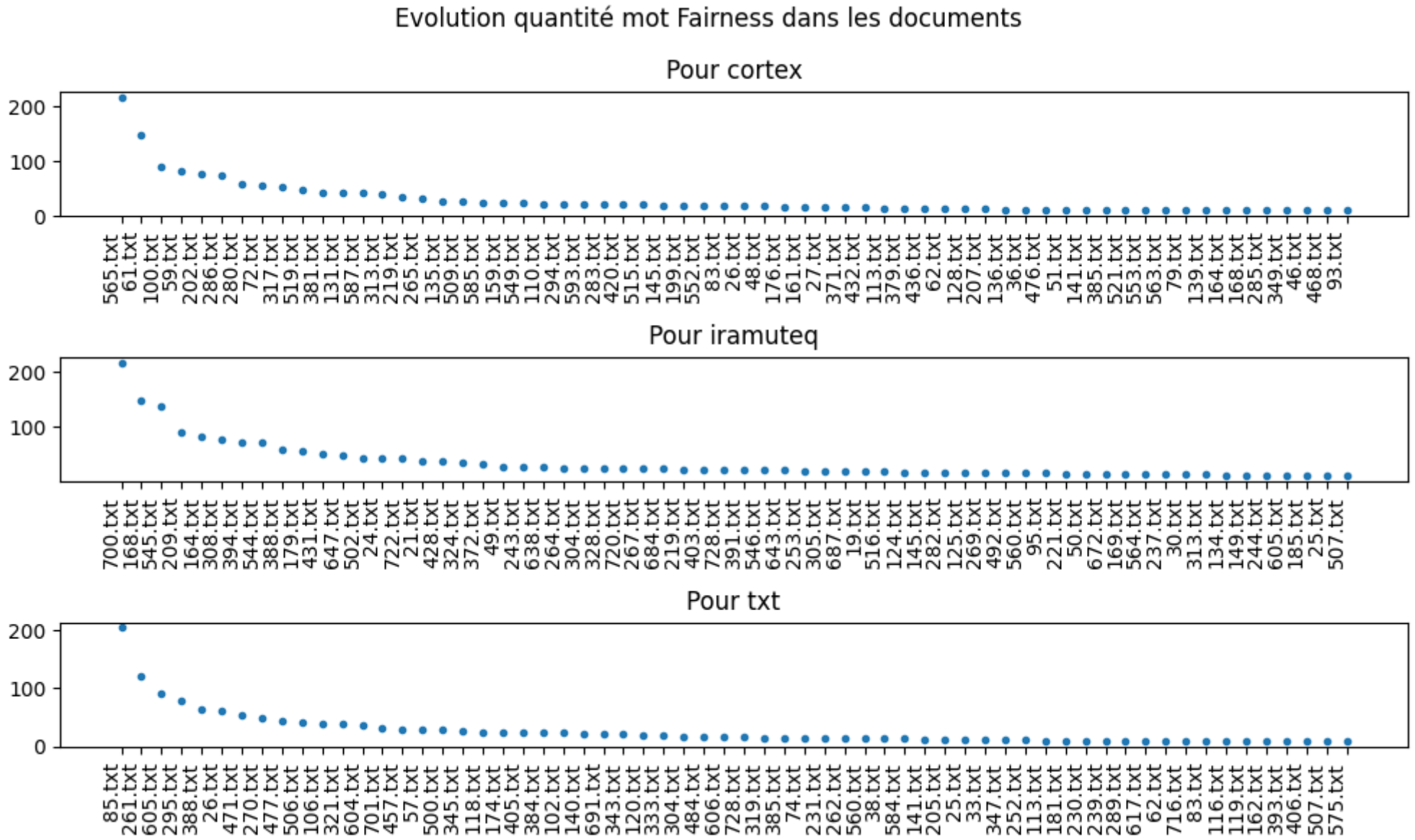
2.2 ''Fairness'' in the data (multi)

Among the keywords, we will now focus on "fairness". It's a word that is particularly important as an ethical concept, and which is quite widely used in the corpus.

2.1 QUESTION 2.1

2.1 Question: which are the documents in which the word "fairness" appears the most (top 10% of the corpus)? Please provide a visualization and comment it.

In [32]:
```
i = 0
WORD_FAIRNESS = 'fairness'

count_by_file = {}
for method in list_corpus:
    count_by_file[method] = {}
    corpus = list_corpus[method].data

    tmp = {filename: corpus[filename].count(WORD_FAIRNESS) for filename in corpus}
    count_by_file[method][WORD_FAIRNESS] = dict(sorted(tmp.items(), key=lambda item: item[1], reverse=True))
    # count_by_file[method][WORD_FAIRNESS] = tmp

n_10per = len(list_corpus['txt'].data) // 10  # 10%

plt.close('all')
fig = plt.figure(figsize=(10, 6))
fig.suptitle('Evolution quantité mot Fairness dans les documents')
for i, method in enumerate(list_corpus):
    ax = fig.add_subplot(len(list_corpus), 1, i +1)
    tmp = count_by_file[method][WORD_FAIRNESS]
    ax.plot(list(tmp.keys())[:n_10per], list(tmp.values())[:n_10per], '.')
    ax.set_title(f"Pour {method}")
    for label in ax.get_xticklabels():
        label.set_rotation(90)
        label.set_horizontalalignment('right')

fig.tight_layout()
plt.show()
```



In [ ]:
```
# X.toarray()
# fairness_index = vectorizer.vocabulary_.get('fairness')
# freq_fairness = X[:, fairness_index].toarray().flatten()
# freq_fairness.shape
# # Obtenir les indices des documents triés par fréquence du mot 'fairness'
# top_documents_indices = np.argsort(freq_fairness)[-n_10per:]
# print("Top documents where 'fairness' appears the most:", top_documents_indices)
# top_documents = [corpus[i] for i in top_documents_indices]
# print("\nTop documents:\n", top_documents)
```

En Powershell :

- 274 fichiers contenant le mots 'fairness'

```
$file_nb = @{}
Get-ChildItem .\data\txts\*.txt | %{ $nb=(Get-Content -Path $_ | Select-String -Pattern 'fairness' -
AllMatches).Matches.Count;$_.Name+' = '+$nb} | %{$tmp=$_.split('=');$file_nb[$tmp[0].trim()]=[int]$tmp[1]}
($file_nb.GetEnumerator() |Sort-Object -Property Value -Descending | %{if ($_.Value -ne 0) {$_}})[0..62]
```

```
Name                           Value
----                           -----
85.txt                         228
261.txt                        150
605.txt                        145
295.txt                        98
26.txt                         87
388.txt                        78
477.txt                        76
604.txt                        73
270.txt                        62
471.txt                        59
506.txt                        53
701.txt                        53
```

2.1 Answer:

2.2 QUESTION 2.2

Now we want to know more about the context in which the word "fairness" appears in the corpus.

2.2 Question: Explore the context in which the word "fairness" appears in the documents selected in the previous question (top 10% of the corpus). Please provide a visualisation and comment it.

Hint: you could for example consider n-grams containing "fairness" (but feel free to explore context by other means!)

In [ ]:
```
# To-do
```

2.2 Answer:

2.3 QUESTION 2.3

We will now focus on a sentence-level analysis of our corpus and extract a sub-corpus of sentences that use the word "fairness".

2.3 Question: What information will we lose by focusing on extracted sentences? Please provide three answers.

2.3 Answer:

---

3 - Analyse avancée sur les données brutes txt

3.1 TXT : Data Presentation and Retrieval

1.3 QUESTION 1.3

1.3 Question: what are the most frequent words, excluding stopwords, in the whole corpus? Please provide visualizations and comment these visualisations.

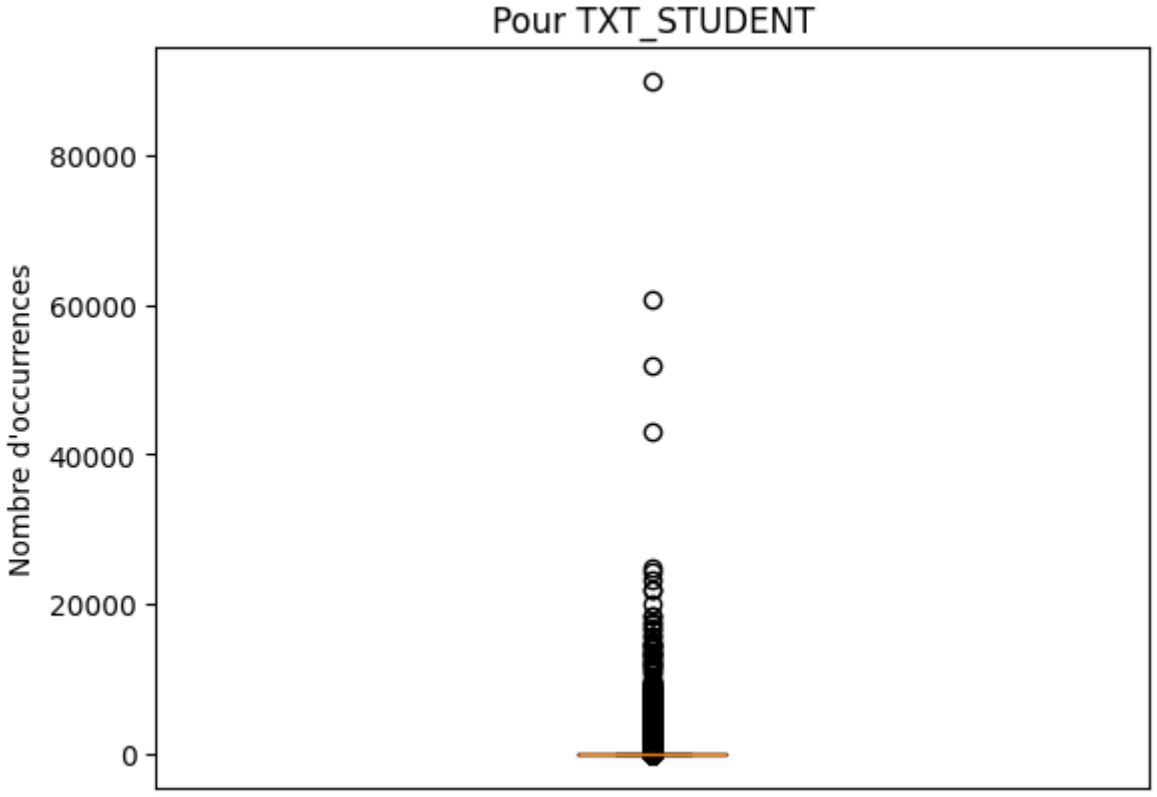
1.3 Answer:

In [10]:
```
model = 'TXT_STUDENT'
data = list_corpus['txt']
corpus = data.get_corpus()
# Nouveau stop_words :
stop_words = ['the','and','of','to','in','a','for','is','that','on','be','are','or','de','by','an','this','it','will','which','la','thei
car_strip = ".-'"
car_important = string.ascii_letters + string.digits + car_strip

# Nettoyage :
corpus_clean = [] # corpus pour les analyses sémantiques (POS-tag et n-gram)
tokens = {}  # compte les apparitions des mots dans tout le corpus
for word in corpus.split():
    word = unidecode(word)  # tout en ASCII
    word = word.lower()  # tout en minuscule

    # Retirer tout les caractères non alpha jugée inutile
    word = "".join(l for l in word if l in car_important)
    # Retire ceux qui sont au début ou à la fin
    word = word.strip(car_strip)

    if len([x for x in word if x.isalnum()]) == 0:  # contient des lettres/chiffre
```

```
        continue
    if word in stop_words:  # pas de stop word
        corpus_clean.append(word)
        continue

    if len(word) == 1:  # pas 1 seule lettre
        continue

    if (word not in tokens):
        tokens[word] = 1
    else :
        tokens[word] += 1

    corpus_clean.append(word)

#On supprime tous les mots à 1
copy = tokens.copy()
for word in copy :
    if (tokens[word]==1) :
        tokens.pop(word,None)

# on récupère le corpus et initialise les stopwords
corpus_clean = " ".join(corpus_clean) # as string

#Sortir le mot le plus utilisé
cle_max = max(tokens, key = tokens.get)
print(f"{model} : Mot le plus utilisé : {cle_max} = {tokens[cle_max]}")
print(f"{model} : Nb initial mot : {len(copy)}, Nb mot >1 lettre et stopword : {len(tokens)}")

# Extraire les occurrences et les mots
occurrences_values = list(tokens.values())
mots = list(tokens.keys())
plt.close('all')
fig = plt.figure()
ax = fig.add_subplot()
ax.boxplot(occurrences_values)
ax.set_title(f"Pour {model}")
ax.set_ylabel("Nombre d'occurrences")
ax.get_xaxis().set_visible(False)

# Calculer la position des points extrêmes
outliers = ax.lines[5].get_ydata()

# Afficher les mots associés aux outliers (valeurs extrêmes)
for outlier in outliers:
    # Trouver le mot correspondant à la valeur d'occurrence
```

```
    mot_associe = [mot for mot, count in tokens.items() if count == outlier]

# Sans retirer les unicode et filtre des car
# TXT_STUDENT : Mot le plus utilisé : ai = 78323
# TXT_STUDENT : Nb initial mot : 432953, Nb mot >1 lettre et stopword : 180092
#
# TXT_STUDENT : Mot le plus utilisé : ai = 89739
# TXT_STUDENT : Nb initial mot : 284702, Nb mot >1 lettre et stopword : 120965
```

```
TXT_STUDENT : Mot le plus utilisé : ai = 89739
TXT_STUDENT : Nb initial mot : 284702, Nb mot >1 lettre et stopword : 120965
```



In [13]:
```
top = sorted(tokens.items(),key =lambda x : x[1], reverse = True) #Liste des mots les plus utilisés

# Calculer la médiane des occurrences
mediane = np.median(occurrences_values)
moyenne = np.mean(occurrences_values)
ecart_type = np.std(occurrences_values)
Q3 = np.percentile(occurrences_values, 90)
stat_info = {'mediane': float(mediane), 'moyenne': float(moyenne), 'sig': float(ecart_type), 'Q3': float(Q3)}

pprint(stat_info, sort_dicts=False)
```

```
{'mediane': 4.0,
 'moyenne': 50.21592196089778,
 'sig': 548.2398413711222,
 'Q3': 38.0}
```

3.2 TXT : ''Fairness'' in the data

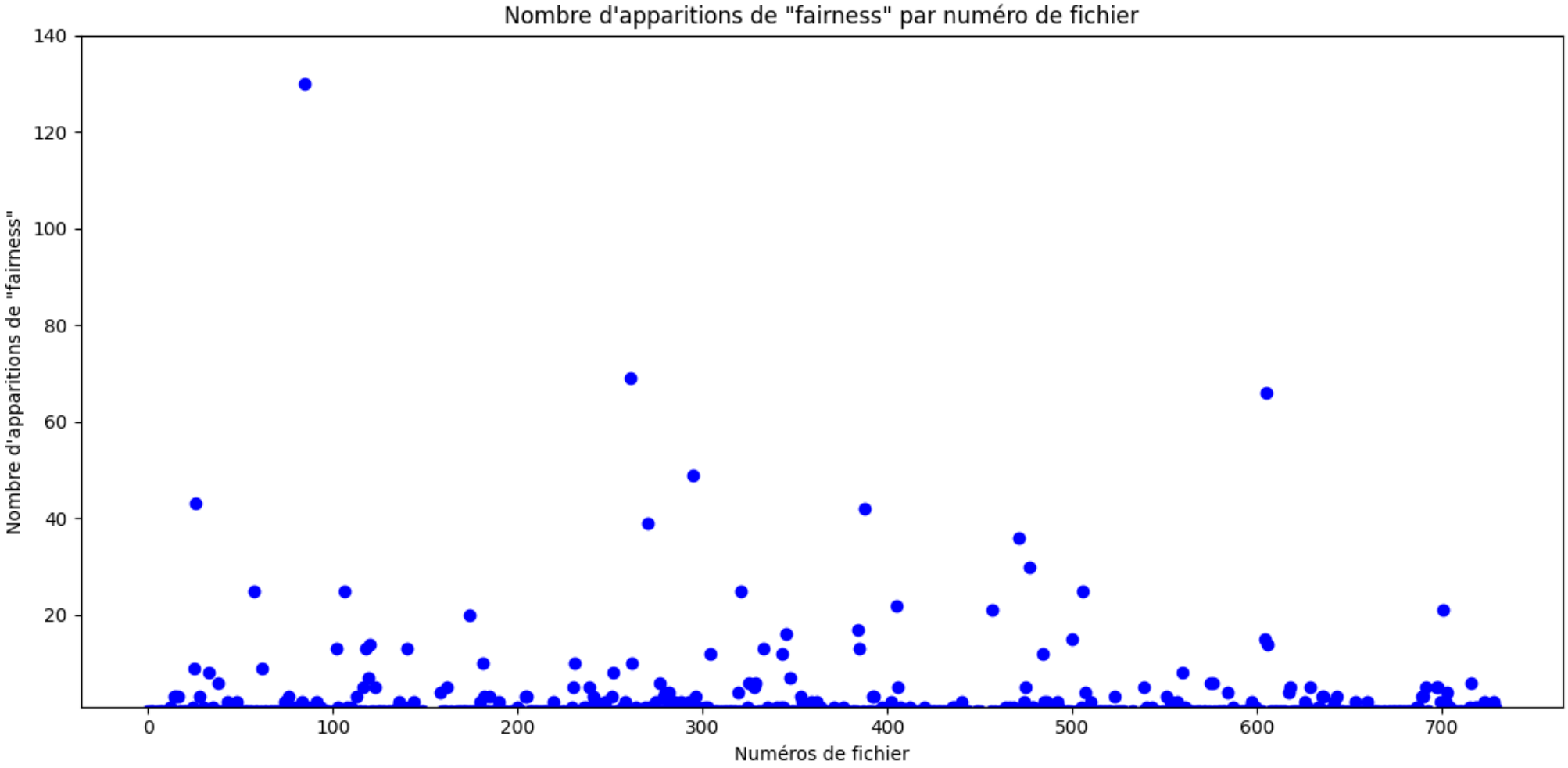
2.1 QUESTION 2.1

2.1 Question: which are the documents in which the word "fairness" appears the most (top 10% of the corpus)? Please provide a visualization and comment it.

2.1 Answer:

In [17]:
```
model = 'TXT_STUDENT'
data = list_corpus['txt']

nb_fairness_by_file = {}
for filename in data.data:
    data_file = data.data[filename]
    nb_fairness_by_file[filename] = 0
    for word in data_file.split():
        if word == 'fairness' :
            nb_fairness_by_file[filename] += 1

#Liste du top des fichiers qui ont le plus de fairness
top_fairness = sorted(nb_fairness_by_file.items(),key =lambda x : x[1], reverse = True) #Liste des mots les plus utilisés
```

In [20]:
```
fichiers = [int(x[0].split('.')[0]) for x in top_fairness]  # Extraire les numéros de fichiers
valeurs = [x[1] for x in top_fairness]

# Créer un plot avec les numéros de fichiers sur l'axe X
plt.close('all')
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot()
ax.scatter(fichiers, valeurs, marker='o', linestyle='-', color='b')
ax.set_xlabel('Numéros de fichier')
ax.set_ylabel('Nombre d\'apparitions de "fairness"')
ax.set_title('Nombre d\'apparitions de "fairness" par numéro de fichier')
ax.set_ylim(1, max(valeurs) + 10)  # Fixer les limites de Y pour qu'il commence à 1
fig.tight_layout()
plt.show()
```



Ici on a tracé la présence du mot fairness par id, pour les 10% les plus élevés

On remarque que seuls 256 documents ont le mot fairness dedans

Rq : regarder aussi le mot unfairness et un-fairness (on associe ces 2 mots en 1 seul)

On va perdre les titres, les références par ex. de schéma, bref les infos en dehors du texte

3.3 TXT : Fairness, N-gram

Nous nous interessons uniquement au 2-gram et 3-gram.

In [16]:
```
# Vectorisation pour 1-grams, 2-grams, et 3-grams, avec stopwords
```

```
def get_ngrams_freqs(corpus_clean, stop_words, ngram = (2, 3)):
    if not corpus_clean.strip():
        raise ValueError("Le corpus est vide. Impossible de générer des n-grams.")

    vectorizer = CountVectorizer(ngram_range=ngram, stop_words=stop_words)
    X = vectorizer.fit_transform([corpus_clean])

    if len(vectorizer.vocabulary_) == 0:
        raise ValueError("Si vocabulaire vide, peut-être que le corpus contient uniquement des stopwords.")

    ngrams = vectorizer.get_feature_names_out()
    freqs = np.asarray(X.sum(axis=0)).ravel()
    vocab = vectorizer.vocabulary_
    return ngrams, freqs, vocab


def get_ngram_on_word(ngrams, freqs, vocab, word = "fairness"):
    '''
    It returns a dictionnaire.
    '''
    fairness_ngrams = [ngram for ngram in ngrams if word in ngram]
    fairness_ngram_counts = {fairness_ngrams[i]: freqs[vocab[fairness_ngrams[i]]] for i in range(len(fairness_ngrams))}
    #print(f"frequence associées {freqs}")
    return dict(sorted(fairness_ngram_counts.items(), key=lambda item: item[1], reverse=True))


# Générer le nuage de mots
def word_cloud(fairness_ngram):
    wordcloud = WordCloud(
        width=800,
        height=400,
        background_color="white",
        colormap="viridis"
    ).generate_from_frequencies(fairness_ngram)

    # Afficher le nuage de mots
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

In [44]:

```
# Les stopwords utilisés ici sont ceux après la réflexion
stop_words = ['the','of','to','in','a','for','is','that','on','be','are','de','by','an','this','it','will','which','la','their','at','et

# on récupère les bi et tri grams
corpus_bi_tri_ngram, corpus_bi_tri_freqs, corpus_bi_tri_vocab = get_ngrams_freqs(corpus_clean, stop_words)
```

```
# on récupère ceux associés à fairness. Attention pour utiliser un autre terme simplement rappeler get_ngram_on_word
fairness_ngram = get_ngram_on_word(corpus_bi_tri_ngram, corpus_bi_tri_freqs, corpus_bi_tri_vocab)
```

In [45]:
```
word_cloud(fairness_ngram)
```

3.4 TXT : Fairness, POS-tag

In [18]:
```
def clean_data_for_pos(data):
    """
    Nettoie les données pour le POS tagging tout en conservant l'ordre d'origine.

    Returns: list[str]: Liste des documents nettoyés dans l'ordre des valeurs du dictionnaire.
    """
    cleaned_documents = []

    for key, line in data.items():
        newline = []
        for word in line.split(" "):
            # Conserve les mots alphanumériques avec ponctuation utile
```

```
            if re.match(r'^[a-zA-Z0-9]+([\'\-\.][a-zA-Z0-9]+)?$', word) or re.match(r'^[\.\,\!\?]$', word):
                # Supprime les mots à 1 caractère inutiles, sauf ponctuation utile ou mots spécifiques
                if len(word) > 1 or word in ['a', 'I', '.', ',', '!', '?']:
                    newline.append(word)
        # Reconstruit le document nettoyé
        cleaned_documents.append(" ".join(newline))

    return cleaned_documents
```

In [19]:
```
# Le vocabulaire utilisé ici ne sont pas les tokens
dict_data = list_corpus['txt'].data #data dict[str, str]: pour chaque fichier (clés = nom de fichier), son contenu

list_data_pos = clean_data_for_pos(dict_data)

# Charger le modèle de langue anglais de spaCy
nlp = spacy.load("en_core_web_sm")
#Traiter en parallèle chaque document pour qu'il fasse nlp(doc) ce qui correspond au POS tagging
batch_size = 40
corpus_POS = list(nlp.pipe(list_data_pos, batch_size=batch_size))

# 20min
```

In [42]:
```
# Dictionnaire pour stocker les rôles de "fairness" et leur fréquence
fairness_roles = defaultdict(int)

# Parcourir chaque document pour trouver les occurrences de "fairness"
for doc in corpus_POS:
    for token in doc:
        if token.text.lower() == "fairness":
            # Compter chaque rôle grammatical de "fairness"
            fairness_roles[token.dep_] += 1

# Convertir le defaultdict en dictionnaire classique pour affichage
fairness_roles = dict(fairness_roles)
# pprint(fairness_roles)
print(tabulate(dict_revert(tmp, sort=True), headers=['Rôle grammatical', 'Nb occurence'], tablefmt='pipe'))
```

Rôle grammatical	Nb occurence
pobj	648
compound	363
conj	249
dobj	218
nmod	93
nsubj	80
nsubjpass	18
appos	14
npadvmod	13
ROOT	9
attr	7
xcomp	5
dep	2
relcl	2
pcomp	2
advcl	2
amod	2
ccomp	2
punct	1
acl	1

Pour Fairness nous avons ces usages: {'compound': 346, 'conj': 252, 'pobj': 682, 'nsubj': 60, 'dobj': 254, 'nmod': 60, 'punct': 1, 'appos': 14, 'nsubjpass': 16, 'npadvmod': 13, 'ROOT': 10, 'attr': 14, 'xcomp': 2, 'advcl': 1, 'oprd': 1, 'relcl': 1, 'dep': 2, 'ccomp': 1, 'acl': 1}

Pour aller plus loin et pour faire les commentaires et explicationss :

Axes de réflexion :

- Possible de chercher les synonymes du mot :
  - synonymes : bias, justice, unbiased, impartiality, objectivity, balance, honesty, neutrality, equity
- Possible de chercher les traductions du mot dans les langues des docs
- Possible de fusionner les variances d'un mot (pluriel,)

Analyse manuelle pour déterminer les stop-words :

LISTE des nouveaux stopword et de notre réflexion

```
stop_words =
['the','and','of','to','in','a','for','is','that','on','be','are','or','de','by','an','this','it','will','which','la','their
```

Méthode :

- Lister sur un corpus non nettoyé ou speudo-nettoyé sans retirer les stop-word ni les mots les plus fréquent
- Pour chacun :
  - déterminer si cela reste un stop-word à retirer
  - retirer les vraies mots comme IA ou data
  - confirmer le retrait des caractères non alphabétique
  - confirmer le retrait des tokens à une lettre
  - on garde les auxiliaires et les négations afin de garder le context de l'utilisation du mot

Première passe :

- the :X
- and : X
- of : X
- to : X
- in : X
- a : X
- for : X
- AI : on garde
- is : X
- that : X
- on : X
- be : X
- as : X
- are : X
- The : X
- or : X
- e : X
- de : X
- data : on garde
- with : X
- by : X
- \u200b : X
- t : X

- an : X
- Non alpha-numérique : X
- o : X
- s : X
- Tous les mots d'une lettre : X
- can : on garde
- from : X
- this : X
- not : on garde
- it : X
- will : X
- which : X
- have : on garde
- la : X
- their : X
- use : X
- at : X
- should : X
- et : X
- also : X
- In : X
- such : X
- des : X
- has : on garde
- systems : on garde
- more : on garde
- public : on garde
- this : X
- human : on garde
- may : on garde

On remarque que les mots les plus utilisés sont des stopwords, on va devoir choisir lesquels on enlève et lesquels on garde, car, par exemple, on ne veut pas effacer la négation. Je mets X quand j'enlève le mot Comment on choisit : en fonction du gain de sens apporté on le juge utile ou non (par ex. on garde les négations, les comparaisons, superlatifs, mot-clé)

Il faudra enlever les stop-words and et or car en vérité ils sont significatifs (montrent que fairness est utilisé dans un contexte décoratif) pour étudier la fréquence des mots mais reste important pour une analyse syntaxique linguistique.

3.5 TXT : Fairness, élargissement vocabulaire

Focus uniquement la corrélation entre fairness et justice, et, fairness et bias.

=>PS : voir le rapport