

Maria Paz Botero Garcia
Kéhina Manseri
Weiqi Zhang
Alix Sirven-Viénot

Enrichissement de corpus
Iris Eshkol-Travella

Dépôt GitHub : https://github.com/ASVienot/Enrichissement_corpus

RAPPORT

*Entraînement d'un modèle de prédictions automatiques
de valeurs d'accord/désaccord dans un corpus annoté
morpho syntaxiquement*



Données :

- Corpus de 378 fichiers XML de comptes-rendus du conseil d'administration de l'Université Paris Nanterre (1984-2018).
- Corpus annoté par les membres de projet *Archive-U* rattaché au LabEx : *Les passés dans le présent : histoire, patrimoine, mémoire.*

Contact :

- Dumoulin Hugo : hdumoulin@parisnanterre.fr
- Frédérique Sitri : Frederique.Sitri@u-pec.fr

Objectif :

- Prédire automatiquement (avec l'apprentissage de surface Weka, Scikit Learn) tour de parole (lié à locuteur, balise RDA), phrase (signe de ponctuation à reconstituer avec un parseur), par token ? la présence de désaccord (oui/non)
- Se familiariser avec les données et les conventions (petit document expliquant les principes de l'annotation manuelle réalisée)
- Tour de parole et phrase : text classification, token : token classification ?
- Réfléchir sur les propriétés linguistique (features) à ajouter test, entraînement, évaluation / validation croisée

SOMMAIRE :

I - Présentation du corpus

- A - Présentation générale
- B - Obtention du corpus
- C - Annotation du corpus

II - Obtention des données À compléter

- A - Extraction des données : extraction.py
- B - Extraction corpus : extraction_dossier.py
- C - Bilan des données
- D - Perspectives pour l'entraînement

III - Entraînement Naïve Bayes

- A - Premier essai
- B - Réduction de la classe majoritaire
- C - Modification des poids
- D - Modification des hyperparamètres

IV - Entrainement Gradient Boosting Classifier

- A - Fonctionnement
- B - Premier essai
- C - Obtention des hyperparamètres

V - Enrichissement du corpus à compléter**VI - Formes porteuses de désaccord**

- A - Script de catégorisation
- B - Analyse des résultats

VII - Conclusion**Perspectives de recherche supplémentaire**

I - Présentation du corpus

A - Présentation générale

Le sujet de notre projet porte sur un corpus numérique composé de 378 fichiers sous format XML. Chaque fichier correspond à une analyse linguistique poussée effectuée à l'aide d'un balisage en XML-TEI de comptes-rendus du conseil d'administration de l'Université Paris Nanterre.

Notre objectif était d'utiliser les informations présentes dans ces fichiers afin *d'entraîner un modèle permettant d'attribuer une valeur d'accord ou de désaccord à une phrase*. Le corpus, produit par les instances juridiques de l'université, a été collecté, annoté et formaté par les collaborateurs du projet *Archive-U* rattaché au LabEx (Laboratoire d'excellence) *Les passés dans le présent : histoire, patrimoine, mémoire*.

Ce LabEx réunit près de 300 chercheurs, enseignants et divers professionnels du milieu du patrimoine et des bibliothèques depuis 2018. Il a pour ambition d'étudier le rapport étroit entre les supports d'informations et les sciences sociales. Comment les normes sociétales et autres idées politiques sont-elles reflétées et partagées à travers une variété grandissante de médiums ? L'équipe du projet considère le discours comme une représentation de ces usages sociaux dont l'articulation dépend grandement des temporalités choisies. De quelle manière une opinion, une contestation ou tout autre positionnement subjectif s'observe-t-il dans un discours soumis à un format pouvant aller du papier au numérique ou de l'oral à l'écrit ? *C'est avec cette idée de temporalités comparées que le projet Archive-U a décidé de s'intéresser à l'évolution des universités depuis les années 1970 à partir de l'évolution de leurs productions écrites*.

La première pierre de l'Université Paris Nanterre est posée en 1964. À l'époque, la faculté sert d'antenne à l'Université de la Sorbonne submergée par le nombre grandissant d'étudiants. En 1970, l'établissement obtient officiellement le statut d'Université et adopte le nom « Université Paris X ». Tout comme dans les autres établissements d'enseignement supérieur, les politiques de l'Université Paris Nanterre sont discutées et appliquées par de nombreuses instances administratives. Aujourd'hui, ce rôle revient notamment aux conseils d'UFR, de directions, au conseil scientifique, à la commission des formations et de la vie universitaire, et surtout au conseil d'administration. Ce dernier, à la tête des conseils centraux de l'université, se réunit chaque mois. Les participants sont composés d'un public varié : professeurs, directeurs, représentants du personnel, étudiants et autres invités... À chaque séance, un individu est chargé de rédiger un compte rendu, un document contenant les grandes thématiques abordées lors du conseil ainsi qu'une transcription plus ou moins authentique des échanges oraux. Ils peuvent également contenir des documents additionnels présentés et discutés en séance tels que des conventions ou emplois du temps.

B - Obtention du corpus

Corpus original

Ces documents produits par le service juridique sont à termes versés au service des archives de l'université. Ce dernier trie et conserve les comptes-rendus pendant plusieurs années avant de les verser aux archives départementales des Haut-de-Seine, une action imposée par la loi. Cette conservation effective a permis la création de collections (d'originaux et de copies) conséquentes de comptes-rendus. Pendant plus de trois ans, le LabEx et le service des archives ont travaillé conjointement afin de permettre la collecte du corpus aujourd'hui étudié.

Corpus numérique

Une fois les documents papiers obtenus, il a fallu pouvoir rendre les données accessibles à l'équipe et exploitables. Un compte rendu plus ancien est généralement relié à l'aide de reliures en spirales et jaunis par le temps. Les plus récents sont eux plus propres mais toujours sous format papier. Des collaborateurs se sont relayés pendant plus de trois ans afin de numériser chaque document, c'est-à-dire de convertir les informations de chaque compte-rendu en données numériques pouvant être traitées par un ordinateur.

Suite à cette phase de collecte, Archive-U a pu récolter 378 comptes-rendus de séances ainsi que d'autres rapports d'équipes scientifiques de Nanterre et de l'Université de Créteil. *Les compte-rendus utilisés pour ce projet couvrent donc une période allant de 1984 (première séance : 18 juin 1984) à 2018 (dernière séance : 10 décembre 2018).*

C - Annotation du corpus

Structure de l'annotation

Chaque compte rendu est donc annoté à l'aide d'un balisage TEI-XML, un standard de représentation assez répandu en sciences sociales et permettant d'attribuer à chaque élément d'un document de nombreux attributs. Chaque fichier correspond à une séance du conseil.

Pour ce qui est de la structure de chaque fichier, elle reste homogène pour l'ensemble du corpus. Afin de l'illustrer, nous utiliserons le compte rendu du 3 décembre 2018. Les namespaces ainsi que les attributs généraux de la séance étudiée sont définis au début du fichier :

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0"
      xmlns:txm="http://textometrie.org/1.0"
      xmlns:tei="http://www.tei-c.org/ns/1.0">
  <teiHeader><fileDesc><titleStmt><title></title>
    </titleStmt><publicationStmt></publicationStmt>
    <sourceDesc></sourceDesc></fileDesc><encodingDesc><appInfo></appInfo>
    <classDecl></classDecl></encodingDesc></teiHeader>
    <text id="weban-upn_PV_CA_2018-12-10_TEI" institution="UPN"
          council="CA" date="2018-12-10" year="2018" genre="PV"
          project="archive-u" base="ArchivU">
```

Les échanges retranscrits dans le compte-rendu sont ensuite divisés en plusieurs niveaux.

Ils sont premièrement divisés en **tours de parole**, chaque changement d'énonciateur est ainsi indiqué. Les tours de paroles sont annotés à l'aide des balises **<RDA>** et disposent de trois attributs : **sex** (sex de l'énonciateur : *M, F, C* (*pour incarner le conseil*) ou *unknown*), **normalized name** (fonction de l'énonciateur : *président, robert, levan...*) et **label** (service ou collège de rattachement de l'énonciateur : *presidential office, collège usagers...*).

Chaque tour de parole est ensuite divisé en **phrases** annotées à l'aide des balises **<S>** et disposant de l'attribut **xml:id** permettant de numérotter les phrases au sein du fichier. Les phrases sont numérotées à

partir du début du fichier et ne sont pas réinitialisées à chaque tour de parole. Ces phrases sont à leurs tours divisées en **mots** représentés par les balises <W> disposant de nombreux attributs.

Voici donc une représentation plus concise de la structure de l'annotation et un extrait représentatif de l'apparence réelle des fichiers :

Élément	Balise	Exemple
Séance	<text>	<code>id="weban-upn_PV_CA_2018-12-10_TEI" institution="UPN" council="CA" date="2018-12-10" year="2018" genre="PV" project="archive-u" base="ArchivU"></code>
Tour de parole	<rda>	<code>sex="unknown" normalized_name="unknown" label="unknown"</code>
Phrase	<s>	<code>xml:id="weban-upn_PV_CA_2018-12-10_TEI.s1"</code>
Mot / Token	<w>	<code>id="w_webanupn_PV_CA_20181210_TEIanattuddep_1"> <txm:form>La</txm:form> <txm:ana resp="#src" type="#n">1</txm:ana> <txm:ana resp="#src" type="#foreign-id">weban-upn_PV_CA_2018-12-10_TEI.s1.w1</txm:ana> <txm:ana resp="#src" type="#form">la</txm:ana> <txm:ana resp="#src" type="#udlemma">le</txm:ana> <txm:ana resp="#src" type="#ttlemma">le</txm:ana> <txm:ana resp="#src" type="#udpos">DET</txm:ana> <txm:ana resp="#src" type="#tppos">DET:ART</txm:ana> <txm:ana resp="#src" type="#feats">Gender=Fem Number=Sing PronType=Art </txm:ana> <txm:ana resp="#src" type="#senttokenid">1</txm:ana> <txm:ana resp="#src" type="#head">2</txm:ana> <txm:ana resp="#src" type="#deprel">det</txm:ana> <txm:ana resp="#src" type="#edeprel">det</txm:ana> <txm:ana resp="#src" type="#headform">séance</txm:ana> <txm:ana resp="#src" type="#headudlemma">séance</txm:ana> <txm:ana resp="#src" type="#headtlemma">séance</txm:ana> <txm:ana resp="#src" type="#headudpos">NOUN</txm:ana> <txm:ana resp="#src" type="#headtppos">NOM</txm:ana> <txm:ana resp="#src" type="#headfeats"> Gender=Fem Number=Sing </txm:ana> <txm:ana resp="#src" type="#depid">_</txm:ana> <txm:ana resp="#src" type="#depform">_</txm:ana> <txm:ana resp="#src" type="#depudlemma">_</txm:ana> <txm:ana resp="#src" type="#depttlemma">_</txm:ana> <txm:ana resp="#src" type="#depudpos">_</txm:ana> <txm:ana resp="#src" type="#depttpos">_</txm:ana> <txm:ana resp="#src" type="#outdeprel">_</txm:ana> <txm:ana resp="#src" type="#deppattern">_</txm:ana> <txm:ana resp="#src" type="#ner">O</txm:ana> <txm:ana resp="#src" type="#headner">O</txm:ana></code>

```

<rda sex="unknown" normalized_name="unknown" label="unknown">
<p ana="opening_statement">
<s xml:id="weban-upn_PV_CA_2018-12-10_TEI.s1">
<hi rend="italic">

<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_1"><txm:form>La</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_2"><txm:form>séance</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_3"><txm:form>du</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_4"><txm:form>Conseil</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_5"><txm:form>d'</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_6"><txm:form>administration</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_7"><txm:form>de</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_8"><txm:form>l'</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>
<w id="w_webanupn_PV_CA_20181210_TEIanattuddep_9"><txm:form>Université</txm:form><txm:ana resp="#src" type="valeurvdi" value="désaccord"></txm:ana></w>

```

Les attributs des éléments `<w>` sont les plus nombreux et nous apportent une analyse morphosyntaxique poussée, allant par exemple jusqu'à proposer des annotations proposées par des outils différents (Exemple : `headudpos` vs `headtppos` -> annotation à l'aide de l'outil `TreeTagger`)

Ces attributs sont eux mêmes contenus dans des enfants de la balise `<w>` tels que `<txm:form>` ou `<txm:ana>` afin de séparer la **forme** du **lemme** et sa forme telle qu'elle est utilisée dans le texte de son analyse.

Enfin, certains mots disposent d'une balise spécifique de la forme :

```

<txm:ana type="#valeurvdi" resp="#adm-sitri">D</txm:ana></w>

```

Ces balises sont essentielles à notre projet car elles correspondent à la valeur d'accord/désaccord du mot. Ces balises peuvent prendre trois valeurs différentes : **D** (désaccord), **INDEF** (indéfini) et “ “ (admise comme étant “accord”). Nous remarquons ici que ces valeurs sont donc attribuées à des tokens et non pas à des phrases, unité lexicale et sémantique sur laquelle nous souhaiterions entraîner notre modèle.

Convention d'annotation

Lors de la collecte du corpus, nous avons pu également avoir accès à la convention de notation ayant été rédigée dans le cadre de l'annotation des comptes-rendus. **Dans cette dernière, l'équipe du projet liste notamment les formes porteuses d'une valeur d'accord/désaccord.** Parmi elles, nous retrouvons des **verbes**, des **noms** et des **adjectifs**.

Les verbes porteurs de désaccord sont principalement ceux utilisés pour transcrire le **discours indirect** tels que :

Accuse / avoue / condamne / condamnent / conteste / contredit / ne convainc pas / craint / critique / défend...

Les noms porteurs de désaccord sont eux principalement utilisés **à la suite de ces verbes permettant de transcrire le discours indirect :**

Condamne Condemnation : affirme|réaffirme sa condamnation, débute ses propos par une condamnation... Doute(s) : loc met en doute, émet un doute...

Enfin, les adjectifs porteurs du désaccord sont ceux généralement **présents dans des formes de type "se déclare..."**. Ils font **suite à des verbes porteurs d'opinions ou de jugements**. Ces adjectifs sont listés à la fin de la convention :

Aberrant, absurde, abusif (propos), affolant, ambigu, antisociale, tout à fait anormal...

Ces formes sémantiques et syntaxiques sont ainsi au centre des données utiles à l'entraînement de notre modèle. **À ce stade du projet, nous avons déjà pu émettre plusieurs hypothèses au sujet des éventuelles difficultés pouvant être rencontrées lors de l'entraînement du modèle :**

- **Le manque de données exploitables** : lors de l'analyse de la structure d'annotation, nous avons pu remarquer un nombre extrêmement limité de balises D. Au commencement du projet, ces dernières nous étaient même inconnues de par leurs raretés. Nous en trouvions en moyenne 3 dans les séances étudiées. L'annotation des valeurs d'accord n'ont en effet pas pu être terminées avant le commencement de notre projet. **Avec l'accord de notre professeure, nous avons décidé d'admettre qu'une phrase contenant un token porteur de la valeur D correspondait à une phrase porteuse de la valeur D. Les balises de type INDEF ne sont pas prises en compte et les phrases les contenant relèvent donc de l'accord.**
- **L'exactitude de la valeur de désaccord** : l'expression du désaccord, comme toute autre expression de sentiment ou de polarité, reste extrêmement complexe à détecter. En effet, l'attribution d'une valeur de désaccord à un mot peut premièrement sembler extrêmement subjective. L'indice d'accord/désaccord ou positif/négatif de nombreux mots tels que *incongru* ou *stupéfiant* **dépend grandement de leurs contextes linguistiques**. L'adjectif *stupéfiant* peut par exemple exprimer un sentiment de surprise positif. De plus, admettre qu'une phrase contenant un mot à valeur de désaccord exprime ce même sentiment est une vision pouvant s'avérer restreinte. Les formes telles que *pas choquant* relèvent plutôt de l'accord malgré la présence de *choquant*. De plus, certaines phrases mériteraient à disposer d'une valeur "neutre" et non d'une valeur "accord" lorsqu'elles ne contiennent pas de tokens porteurs d'une valeur de désaccord. L'exactitude de la valeur d'accord des mots semble généralement très correcte car annotée manuellement. Celle de la valeur des phrases, telle qu'expliquée précédemment, l'est beaucoup moins.

L'entraînement d'un modèle sur une classe sous-représentée est un exercice compliqué et nécessitant de nombreuses stratégies de compensation. Après l'obtention du corpus complet au bout de plusieurs semaines, nous avons pu obtenir des chiffres globaux permettant de confirmer ou réfuter nos hypothèses sur la sous-représentation de la classe désaccord. Nous allons maintenant pouvoir présenter ces chiffres après une présentation des programmes ayant permis l'extraction des données pertinentes pour l'entraînement.

II - Données d'entraînement

Afin de pouvoir commencer la phase d'entraînement, nous avons décidé de rédiger un premier script permettant de récupérer les données nécessaires à notre modèle pour repérer les structures exprimant l'accord ou le désaccord.

La première étape était ainsi de pouvoir extraire l'ensemble des phrases porteuses de valeurs d'accords/désaccord permettant à notre modèle de s'entraîner et d'être testé. De plus, nous cherchions à conserver une structure à plusieurs niveaux nous permettant de connaître le nombre de phrases total du corpus, le nombre total de phrases et de tours de paroles par séance, et d'autres informations utiles à l'établissement de statistiques textuelles.

Pour cela, nous avons rédigé un script dédié nommé “extraction.py” dont nous allons décrire le fonctionnement en détail.

A - Extraction fichier : extraction.py

Au vue de la structure XML de nos fichiers, nous avons cherché à utiliser une librairie ou un module de traitement adapté et permettant d'effectuer facilement des opérations sur nos éléments. Après quelques essais avec la librairie *ElementTree*, nous nous sommes finalement tournés vers la librairie *BeautifulSoup*.

En effet, *ElementTree* tend à restreindre nos actions et les opérations pouvant être effectuées sur les éléments. Les *namespaces* empêchent notamment l'extraction d'éléments spécifiques malgré une appellation homogène dans tout le corpus et très peu d'ambiguïtés au niveau des noms. *BeautifulSoup* offre une accès plus libre aux éléments et dispose également de méthodes et de fonctions aux syntaxes plus accessibles pour une structure de données peu complexe comme la nôtre. Si certains de nos éléments disposent de plusieurs dizaines d'attributs, **la structure globale de chaque séance reste organisée autour de 4 niveaux de profondeur : séance, tour de parole, phrase, mot.**

Afin de stocker les informations pertinentes pour chacun de nos niveaux de profondeurs, nous avons commencé par créer un ensemble de *dataclasses* (un type de données construit autour d'une liste d'instances d'un ensemble spécifique) dont le contenu sera développé lors de l'explication détaillée de la fonction d'extraction.

```
@dataclass
class Seance:
    id_seance: str
    nb_tours_parole: int

@dataclass
class Token:
    form: str
    lemme: str
```

Voici toutes les valeurs attribuées aux classes du programme.

La classe *Séance* contient :

- L'*id* de la séance.
- Le *nombre de tours de paroles* de la séance.

La classe *Token* contient :

- La *forme* du token.
- Le *lemme* du token.

```

@dataclass
class Phrase:
    info_phrase: str
    desaccord : bool
    tokens: List[Token]
    lemmes : List[Token]

@dataclass
class ToursParole:
    id_rda: int
    sex: str
    nom: str
    label: str
    nb_phrases: int
    phrases: List[Phrase]

```

La classe **Phrase** contient :

- Une *valeur d'accord*.
- Une *liste des formes des tokens* de la phrase.
- Une *liste des lemmes des tokens* de la phrase.
- Une *tuple composée des listes précédentes converties en chaînes de caractères et de l'id* de la phrase.

La classe **ToursParole** contient :

- L'*id* du tour de parole.
- Le *sex* de l'énonciateur.
- Le *nom* de l'énonciateur.
- Le *statut* de l'énonciateur.
- Le *nombre de phrases* du tour de parole.
- Une *liste des phrases en chaînes de caractères* du tour de parole.

Notre programme se charge de parcourir le contenu XML d'un seul fichier et de remplir des instances de nos classes au fur et à mesure. Au sein d'une fonction nommée **extraction_tours_parole**, nous commençons premièrement par ouvrir notre fichier et initialiser certaines variables utiles pour le remplissage de nos classes. Nous parcourons ensuite les niveaux des tours de paroles et des phrases afin d'accéder aux balises <w> correspondant donc aux tokens de nos phrases. En effet, notre sujet portant sur la valeur d'accord/désaccord des phrases de notre corpus, il a fallu pouvoir concaténer l'ensemble de nos tokens afin d'obtenir les phrases nécessaires à l'entraînement.

```

def extraction_tours_parole(chemin_fichier):

    with open (chemin_fichier, 'r') as tei:
        donnees = tei.read()
        soup = BeautifulSoup (donnees, 'lxml-xml')
        compteur_token = 0
        liste_tours_parole = []
        i = 1
        compteur_rda = 0

        for rda_tag in ft_progress(soup.find_all("rda")):
            id_rda = i
            i += 1
            sex = rda_tag['sex']
            nom = rda_tag['normalized_name']
            label = rda_tag['label']
            compteur_rda += 1

            liste_phrases_par_rda = []
            for s_tag in rda_tag.find_all("s"):
                words = s_tag.find_all("w")
                texte_parties = []
                liste_lemmes= []
                tok_desaccord = 0

```

Dans ces deux premières boucles, nous extrayons :

- L'**id du tour de parole** (à l'aide d'une variable "i" mise à jour à chaque boucle).
- Le **sex** de l'énonciateur.
- Le **statut** de l'énonciateur.
- Le **nom** de l'énonciateur.

Nous mettons également à jour un compteur **compteur_rda** permettant d'obtenir le nombre de tours de paroles de la séance étudiée. Nous initialisons enfin plusieurs variables utilisées pour la concaténation de nos tokens.

```
for w in words:  
    compteur_token += 1  
    txm_form = w.find("txm:form")  
    t xm_lemme = w.find("txm:ana", {"type": "#udlemma"})  
    ana_tag = w.find("txm:ana", {"resp": "#adm-sitri"})  
    if t xm_form is not None and t xm_form.text is not None:  
        texte_parties.append(t xm_form.text)  
    if t xm_lemme is not None and t xm_lemme.text is not None:  
        liste_lemmes.append(t xm_lemme.text)  
    token = Token(form=t xm_form.text, lemme= t xm_lemme.text if t xm_lemme is not None else "")  
    if ana_tag is not None and ana_tag.text is not None and ana_tag.text == "D":  
        tok_desaccord += 1
```

Nous cherchons ensuite à obtenir la forme et le lemme de chaque token afin de créer et alimenter deux listes dédiées correspondant donc à la phrase concernée par le tour de boucle. Nous cherchons ensuite à trouver une balise porteuse de la valeur de désaccord. Si cette dernière est absente, nous passons au token suivant, autrement, nous ajoutons 1 à la variable **tok_desaccord** initialisée précédemment.

```
desaccord = "D" if tok_desaccord > 0 else "A"  
texte = ' '.join(texte_parties)  
lemmes = ' '.join(liste_lemmes)  
id_phrase = s_tag.get('xml:id', "")  
info_phrase = (id_phrase, texte, lemmes)  
phrase = Phrase(info_phrase=info_phrase, desaccord=desaccord, tokens=texte_parties, lemmes=liste_lemmes)  
liste_phrases_par_rda.append(phrase)
```

Une fois l'ensemble des tokens de notre phrase parcouru, nous attribuons une valeur de désaccord à cette dernière si au moins un de ses tokens était annoté D. Nous faisons également en sorte d'obtenir nos phrases sous forme de chaînes de caractères (composées des formes et des lemmes) plutôt que sous forme de listes. Nous attribuons ensuite les valeurs correspondantes aux variables rattachées au tour de parole telles que le nombre des phrases de ce dernier par exemple.

```
text_elem = soup.find('text')  
id_seance = text_elem.get('id', "")  
nb_tours_parole = len(liste_tours_parole)  
info_seance = Seance(id_seance=id_seance, nb_tours_parole=nb_tours_parole)
```

Enfin, nous alimentons les variables rattachées à la séance elle-même telles que le nombre de tours de parole ou encore l'id de la séance.

La création et l'utilisation de nos dataclasses a permis l'obtention de données structurées pouvant être pertinentes pour l'entraînement du modèle mais aussi d'éventuelles enquêtes statistiques au niveau du corpus. Les données obtenues peuvent être affichées sur notre terminale et sont de la forme suivante :

```

Id_tour : 5, Personne: conseil, sexe: C, label: unknown, nb_phrases: 2, phrase: [(['fradhs-upn_2283W_CR_CA_1988-07-01_TEI.s31', 'Mesdames MORTUREUX e t BARTHELEMY , ainsi que Monsieur BECKER .', 'Mesdames MORTUREUX et BARTHELEMY , ainsi que Monsieur BECKER .'), ('fradhs-upn_2283W_CR_CA_1988-07-01_TEI.s32', 'Remerciements du Conseil , pour le travail qu' elle a accompli à Michèle ESTELLE , Chef de Cabinet , qui a souhaité quitter cette fonction .', 'remerciement de+le conseil , pour le travail que lui avoir accomplir , à Michèle Estelle , chef de cabinet , qui avoir souhaiter quitter ce fonction .')], tokens par phrase : [[['Mesdames', 'MORTUREUX', 'et', 'BARTHELEMY', ',', 'ainsi', 'que', 'Monsieur', 'BECKER', '.'], ['Remerciements', 'de', 'Conseil', ',', 'pour', 'le', 'travail', 'qu''', 'elle', 'a', 'accomplice', '','a', 'Michèle', 'ESTELLE', ',', 'Chef', 'de', 'Cabinet', ',', 'qui', 'a', 'souhaité', 'quitter', 'cette', 'fonction', '']], désaccords : ['A', 'A']]
-----
Id_tour : 6, Personne: unknown, sexe: unknown, label: unknown, nb_phrases: 1, phrase: [(['fradhs-upn_2283W_CR_CA_1988-07-01_TEI.s41', 'Approbation de la constitution de ce Bureau vote pour à l' unanimité .', 'approbation de la constitution de ce Bureau vote pour à le unanimité .')], tokens par phrase : [['Approbation', 'de', 'la', 'constitution', 'de', 'ce', 'Bureau', 'vote', 'pour', 'à', 'l''', 'unanimité', '.']], désaccords : ['A']
-----
Id_tour : 7, Personne: unknown, sexe: unknown, label: unknown, nb_phrases: 1, phrase: [(['fradhs-upn_2283W_CR_CA_1988-07-01_TEI.s43', 'Monsieur HERRE NSCHMIDT est nommé administrateur provisoire de l' UFR , suite au départ de Monsieur DENEUX , élu Vice-Président aux Etudes .', 'Monsieur HERRENSCHMI DT être nommer administrateur provisoire de le UFR , suite à+le départ de Monsieur DENEUX , élire vice-président à+le Etudes .')], tokens par phrase : [['Monsieur', 'HERRENSCHMIDT', 'est', 'nommé', 'administrateur', 'provisoire', 'de', 'l''', 'UFR', ',', 'suite', 'au', 'départ', 'de', 'Monsieur', 'DENEUX', ',', 'élu', 'Vice-Président', 'aux', 'Etudes', '.']], désaccords : ['A']
-----
Id_tour : 8, Personne: unknown, sexe: unknown, label: unknown, nb_phrases: 1, phrase: [(['fradhs-upn_2283W_CR_CA_1988-07-01_TEI.s45', 'Vote pour à l' unanimité .', 'vote pour à le unanimité .'])], tokens par phrase : [['Vote', 'pour', 'à', 'l''', 'unanimité', '.']], désaccords : ['A']

```

Au vu du fonctionnement de notre programme, nous devions maintenant le généraliser à l'ensemble de notre corpus et extraire les informations des 378 fichiers de ce dernier.

B - Extraction corpus : extraction_dossier.py

Le programme ***extraction_dossier.py*** permet de parcourir le dossier du corpus et de créer une liste de fichiers. Cette liste est ensuite parcourue et chacun de ses éléments est défini comme un argument de la fonction décrite plus haut et contenu dans le programme ***extraction.py***. Au terme de cette extraction, nous faisons appel à la librairie ***csv*** afin de créer un fichier tabulaire recensant l'ensemble des données pertinentes à l'entraînement. Nous avons décidé d'extraire les phrases composées de lemmes et de formes de tokens ainsi que les id des séances et des tours de parole pour une meilleure visualisation de la répartition des données. Après quelques essais, nous avons cependant seulement utilisé les phrases composées de forme pour l'entraînement de nos modèles. Nous avons en effet conclu que cet attribut serait le plus discriminant dans notre corpus au vu du nombre réduit de phrases annotées en désaccord. Ajouter d'autres données comme les Part of Speech ou bien les relations de dépendances syntaxiques n'auraient fait qu'augmenter le poids de la classe majoritaire.

Le fichier obtenu se présente comme tel (totaldonnees.csv) :

Tour	Id_phrase	phrase	lemmes	accord
1	weban-upn_PV_CA_20	Le Président ouvre la séance à 14 h	le président ouvrir le séance à 14 heure A	
1	weban-upn_PV_CA_20	Nous vous informons que les points	nous vous informer que le point de le o A	
2	weban-upn_PV_CA_20	M. BACQUE et M. COHN-BENDIT c	Monsieur BACQUE et Monsieur COHN A	
2	weban-upn_PV_CA_20	M. POLIT à M. DHOMPS ;	Monsieur POLIT à Monsieur DHOMPS A	
2	weban-upn_PV_CA_20	M. BOUSLIMI à M. VOILLIOT ;	Monsieur BOUSLIMI à Monsieur VOILLIOT A	
2	weban-upn_PV_CA_20	Mme GUILLMAN à M. EL GHOZY ;	Monsieur GUILLMAN à Monsieur EL G A	
2	weban-upn_PV_CA_20	M. SARKOZY à M. JOUVET ;	Monsieur Nicolas sarkozy à Monsieur J A	
2	weban-upn_PV_CA_20	M. BRUNET à M. DE L' ESTOILE ;	Monsieur Brunet à Monsieur DE le EST A	
2	weban-upn_PV_CA_20	M. PERL à Mme PAULET .	Monsieur PERL à Monsieur PAULET . A	
3	weban-upn_PV_CA_20	L' approbation du procès-verbal de	le approbation de+le procès-verbal de A	
4	weban-upn_PV_CA_20	Le Président estime que le budget 2	le président estimer que le budget 201 A	
4	weban-upn_PV_CA_20	Il ajoute que quelques évolutions or	lui ajouter que quelque évolution avoir A	
4	weban-upn_PV_CA_20	Ainsi , la notification en date du 13 c	ainsi , le notification en date de+le 13 c A	

C - Bilan des données

Une fois nos données obtenues, nous avons pu établir un bilan des annotations d'accord et de désaccord pour soutenir notre hypothèse de performance avec des chiffres. En plus du fichier obtenu, nous avons rédigé un programme supplémentaire permettant d'obtenir le nombre d'annotations D dans l'ensemble du corpus (et donc le nombre de mots annotés D). Ce programme, nommé `extraction_D.py`, associé au tableau présenté ci-dessus, nous a permis d'obtenir les valeurs suivantes :

Tokens	Phrases	Désaccord	% de tokens D	% de phrases D
2 815 682	127 573	905	0.03214%	0.709%

Bilan :

- Notre corpus est composé de **378 fichiers**, de **127 373 phrases** et de **2 815 682 tokens**.
- Parmi ces tokens, **0.32%** sont annotés comme porteurs de désaccord.
- Cela nous donne donc **0.709%** de phrases annotées comme porteuses de désaccord.
- **Notre corpus est donc divisé en deux classes** : accord (**A**) et désaccord (**D**).
- La première est sur-représentée avec **126 671 phrases contre 905** pour la seconde (ou **99.3%** contre **0.7%**).

Si notre corpus est volumineux, il reste extrêmement inégal au niveau de la répartition de ses deux classes. ***Ainsi, nous pouvons confirmer notre hypothèse, la performance de nos modèles risque d'être gravement impactée par la sous-représentation de la classe que nous cherchons à prédire.***

D - Perspective pour l'entraînement

Une fois l'ensemble de nos données obtenues, l'entraînement du modèle peut débuter. Deux questions principales se présentent alors à nous :

- **Quel outil d'apprentissage automatique utiliser ?**
- **Quel algorithme d'apprentissage utiliser ?**

Au commencement de notre projet, nous hésitions entre deux outils d'apprentissage automatique différents : *Weka* et *Scikit-Learn*. Si Weka dispose de sa propre interface utilisateur et ne nécessite pas de rédaction importante de code, il reste moins performant face à des ensembles de données massifs et peut parfois restreindre l'utilisateur quant aux ajustements possibles des hyper paramètres des algorithmes utilisés.

Scikit-Learn s'utilise à l'aide de python et permet ainsi l'intégration de nombreuses librairies permettant de manipuler les données (*Pandas*, *csv*) et de représenter graphiquement les résultats (*Matplotlib*). Il est ainsi possible de charger nos données, entraîner un modèle sur ces dernières, et créer une représentation plus explicite des résultats obtenus en un seul programme. Les fonctions alors créées peuvent être utilisées ou adaptées sur d'autres ensembles de données et avec des algorithmes

différents. Au vu de notre corpus conséquent et des hypothèses formulées précédemment, nous avons préféré utiliser Scikit-Learn.

Pour ce qui est des algorithmes utilisés, nous avons décidé de nous tourner vers deux algorithmes différents : *Naïves Bayes* et le *Gradient Boosting Classifier*. L'utilisation de ces derniers est développée dans leurs parties respectives. Pour chaque algorithme utilisé, nous avons eu recours à des “solutions” pouvant en théorie améliorer les performances de notre modèle.

III - Entraînement Naïve Bayes

Le premier algorithme utilisé pour entraîner notre modèle est donc l'algorithme Naïve Bayes, très souvent utilisé dans des tâches de classification de textes.

Naïve Bayes est un algorithme reposant sur une hypothèse d'indépendance modélisée par la formule suivante : la présence d'un mot dans une phrase ne dépendrait pas des autres. Dans notre cas, nous pouvons évidemment contester cette hypothèse en évoquant par exemple certaines formes en deux mots et exprimant le désaccord : “pas d'accord”.

Malgré tout, nous savons aussi que la plupart des phrases ayant été annotées comme “désaccord” dans notre corpus l'ont été sur la base de la présence d'adjectifs ou de noms tels que “bizarre”. La distribution générale des mots obtenue par Naïve Bayes dans nos deux classes peut ainsi s'avérer pertinente ici. Elle reste généralement une bonne indication permettant aux modèles de classification d'atteindre des performances satisfaisantes.

Nous faisons ici l'hypothèse que Naïve Bayes sera capable d'identifier certains schémas permettant de décrire le désaccord mais que sa performance sera grandement impactée par le manque d'annotations D.

A - Premier essai

L'entraînement du modèle à partir de l'algorithme Naïve Bayes s'est fait à partir des fonctions et méthodes proposées par Scikit-Learn sur leur [page dédiée](#). Le programme se nomme ***nv.py***.

Nous avons commencé par rédiger un programme classique dont le fonctionnement est le suivant :

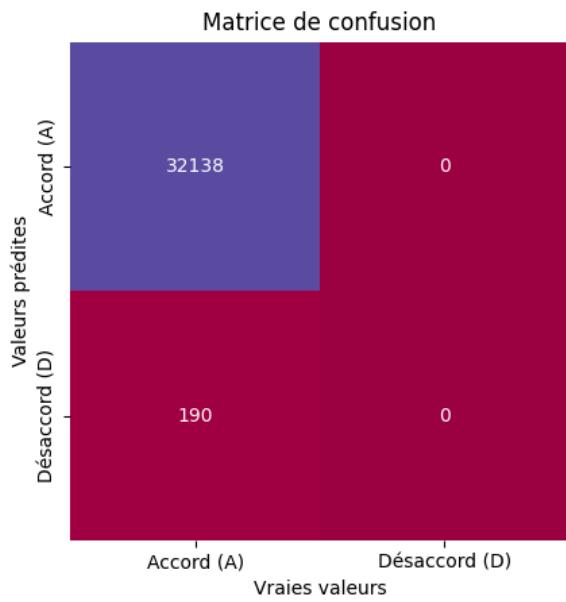
```
french_stop_words = stopwords.words('french')
corpus_dataframe = pd.read_csv('corpus_annotate.csv', header=0, usecols=[2,4], names=['phrase', 'accord'])
corpus_dataframe = corpus_dataframe.dropna(subset=['phrase'])
train_corpus, test_corpus=train_test_split(corpus_dataframe, test_size=0.2, random_state=42,stratify=corpus_dataframe['accord'])
```

Nous commençons par initialiser une liste de *stop words* français, nous avons pour cela utilisé la librairie *nltk* car Scikit-Learn ne proposait qu'une liste de *stop words* anglais.

Nous utilisons ensuite *Pandas* pour récupérer les lignes et colonnes d'intérêt, dans notre cas, les colonnes contenant nos phrases et les valeurs A ou D. **Nous divisons ensuite le corpus en deux ensembles** à l'aide des méthodes et fonctions de Scikit-Learn. Nous avons pris la décision de diviser notre corpus en 80/20 (80 = train, 20 = test), un ratio assez courant dans les tâches de classification et représentant environ **95 000 lignes et 32 000 lignes** respectivement. Nous indiquons aussi un *random state* de 42, un paramètre nous permettant d'obtenir la même division pour l'ensemble des exécutions de la session du programme.

```
model = make_pipeline(TfidfVectorizer(stop_words=french_stop_words), MultinomialNB())
model.fit(train_corpus['phrase'], train_corpus['accord'])
test_evaluation = model.predict(test_corpus['phrase'])
```

Nos données textuelles, donc nos phrases et mots sont ensuite converties en matrices. Chaque mot obtient ainsi un “score” qui sera utilisé pour les prédictions du modèle. Nous excluons bien sûr les *stop words* et indiquons au modèle de **s'entraîner sur** : les phrases et les accords, **afin de prédire** : l'accord des phrases de l'ensemble de test. **À l'aide d'une fonction supplémentaire permettant de générer une matrice de confusion à partir de nos données, nous obtenons les résultats suivants.**



Si nous lisons cette matrice en fonction de la classe D :

En bas à droite : **0 Vrais Positifs (VP)**, 0 D ont été reconnus comme des D.

En bas à gauche : **190 Faux Négatifs (FN)**, 190 D ont été mal reconnus et ont été annotés A.

En haut à gauche : **32 138 Vrais Négatifs (VN)**, tous les A reconnus en A. Donc tous les non D reconnus comme tels.

En haut à droite : **0 Faux Positifs (FP)**, aucun A n'a été reconnu à tort comme D.

La matrice nous permet ainsi d'obtenir les métriques suivantes :

	PRECISION	RAPPHEL	F-MESURE	SUPPORT
CLASSE A	0.99	1	1	32 138
CLASSE D	0	0	0	190
ACCURACY	/	/	0.99	32 328
MACRO AVG	0.50	0.50	0.50	32 328
WEIGHTED AVG	0.99	0.99	0.99	32 328

La **précision** mesure, en prenant D comme classe étudiée, le nombre de D prédis qui le sont réellement. Le **rappel** mesure lui la quantité de données D réelles qui ont été prédis par le modèle. La **f-mesure** est la moyenne harmonique de ces deux valeurs.

Comme nous pouvons l'observer ici, notre modèle obtient un score **d'accuracy** (d'exactitude) très élevé et nous ne pouvons en effet pas contester ce résultat : **le modèle annote les phrases correctement à 99 %.**

Cependant, ce taux n'est évidemment pas surprenant au vu de la répartition de nos classes A (correspondant pour rappel à 99,4 % des annotations de notre corpus) et D. **Le manque d'annotations D empêche le modèle de s'entraîner correctement.** S'il reconnaît certains schémas partagés aux phrases D (comme la présence de certains mots), ces derniers sont complètement éclipsés par les schémas bien plus conséquents retrouvés dans les phrases A.

Comme aucun D n'est trouvé, qu'il soit correct ou non, nous ne sommes pas en mesure d'analyser la performance réelle du modèle à partir des données fournies. L'algorithme admet que toutes les phrases appartiennent à la catégorie A.

Ce premier essai explicite une des faiblesses de Naïve Bayes : **l'algorithme offre des performances moins bonnes lorsque l'une des classes est fortement sous représentée**. Nous avons malgré tout décidé de continuer à l'utiliser afin de chercher une solution pouvant "compenser" le déséquilibre de nos données. ***Les stratégies trouvées sont les suivantes : réduction de la classe majoritaire, modification des poids et obtention des hyperparamètres.***

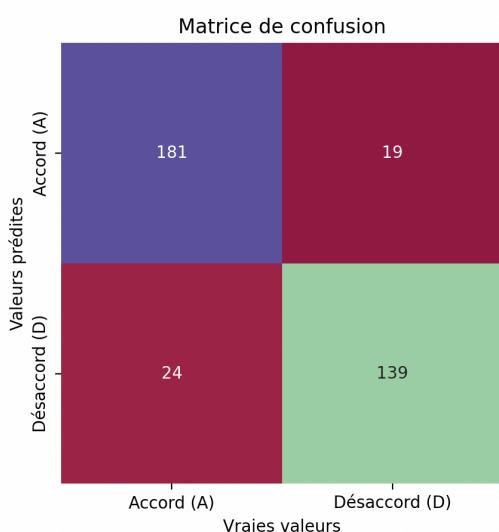
B - Réduction de la classe majoritaire

```
def reduction(fichier):
    df= pd.read_csv(fichier)

    lignes_A=df[df['accord'].str.contains('A')].head(1000)
    lignes_D = df[df['accord'].str.contains('D')]

    plus_petit=pd.concat([lignes_A,lignes_D])
    plus_petit.to_csv('A_1000.csv', index=False)
```

La première stratégie trouvée afin de compenser la sous représentation de notre classe minoritaire consiste en la réduction des données appartenant à la classe majoritaire, dans notre cas, les accords. Nous avons donc cherché à réduire le nombre de phrases (et donc de lignes) annotées en A dans notre fichier d'entraînement. Dans ce premier essai, nous avons utilisé la librairie **Pandas** pour laisser 812 phrases qui expriment le désaccord et 1000 phrases qui expriment l'accord.



Le corpus a été divisé en 80% pour l'entraînement et 20% pour le test. Nous avons pu ensuite relancer l'entraînement de notre modèle avec l'algorithme Naïve Bayes et notre script existant. Les nombres de vrais positifs, vrais négatifs, faux positifs et faux négatifs nous ont permis d'obtenir les métriques suivantes :

	PRECISION	RAPPEL	F-MESURE	SUPPORT
CLASSE A	0.88	0.91	0.89	200
CLASSE D	0.88	0.85	0.87	163
ACCURACY	/	/	0.88	363
MACRO AVG	0.88	0.88	0.88	363
WEIGHTED AVG	0.88	0.88	0.88	363

Nous remarquons des résultats vraiment satisfaisants. Nous avons obtenu un taux d'accuracy très élevé, ce qui peut être vérifié avec la matrice de confusion. Notre modèle annote correctement 88% des phrases. Avec une F-mesure de 0.89 pour la classe A et 0.88 pour la classe D. **Nous pourrions affirmer que cette solution nous permet de remarquer que, effectivement, notre script et l'utilisation de l'algorithme Naïve Bayes peut montrer des résultats corrects et que le déséquilibre dans le corpus est majoritairement responsable des résultats si inégaux.**

Néanmoins, nous savons bien que ces résultats sont plutôt artificiels étant donné la manipulation importante du corpus avant l'entraînement du modèle sur Scikit-learn. C'est pour cette raison que, même si cette solution donne déjà des résultats satisfaisants, nous avons décidé de continuer à chercher les meilleures conditions du modèle pouvant s'adapter à notre corpus.

C - Modification des poids

Une autre solution permettant de ne pas réduire la taille de notre classe A est d'augmenter la classe D. Pour cela, Scikit-Learn offre une fonctionnalité permettant de multiplier l'importance de l'occurrence d'une annotation voulue.

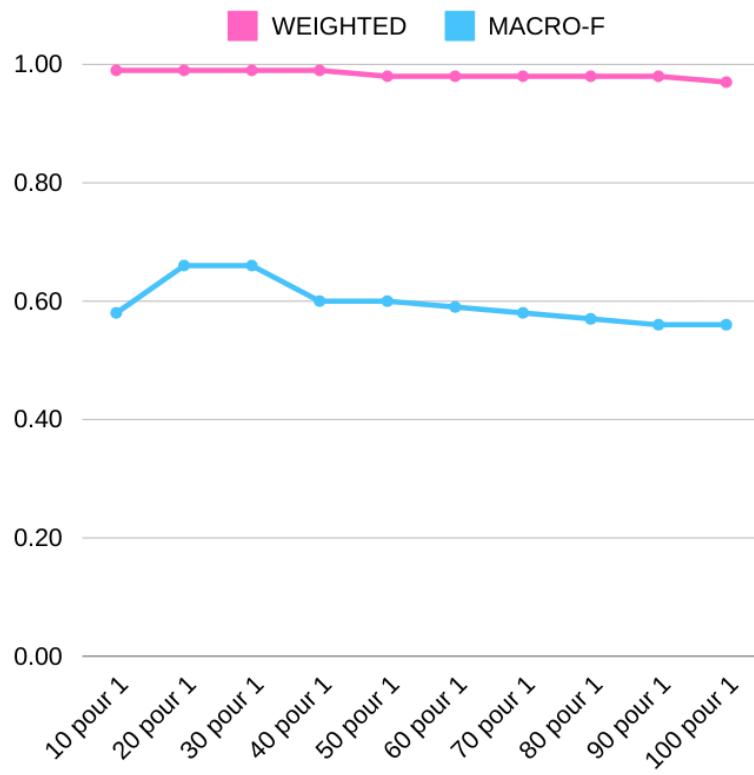
```
weights = np.where(train_dataframe['accord'] == 'D', 10, 1) # Permet de donner plus de "poids" aux désaccords.

# Permet d'entrainer le modèle
model.fit(train_dataframe['phrase'], train_dataframe['accord'], multinomialnb__sample_weight=weights)
# On "fit" les données d'entraînement et les données cibles.
# On dit que ces données vectorisées sont associées à telle ou telle réponse.
```

Dans ce premier exemple, nous déclarons par exemple que pour chaque phrase annotée D, cette dernière comptera en fait comme 10 phrases annotées D.

Nous avons décidé d'appliquer cette compensation pour tous les ratios allant de décimale en décimale jusqu'à 100. Les résultats obtenus sont les suivants :

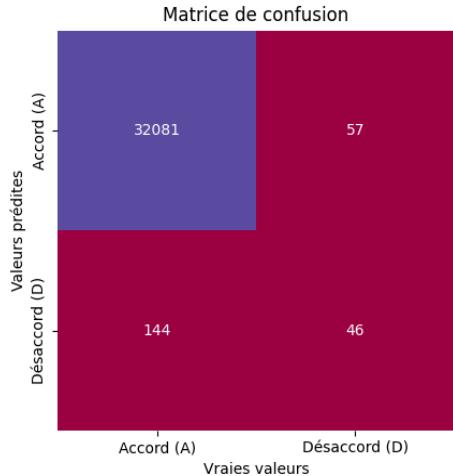
PERFORMANCE NAÏVE BAYES AVEC CHANGEMENT DE POIDS



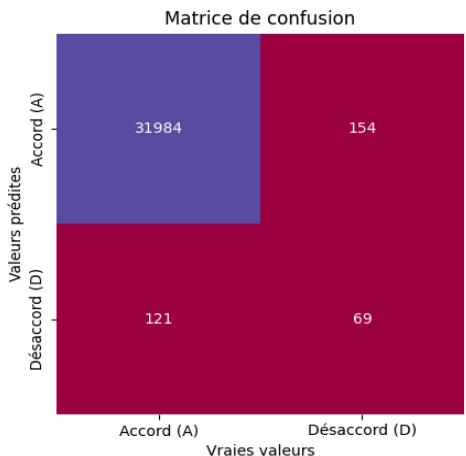
Afin d'identifier le ratio le plus performant, nous avons choisi de prendre en compte la **macro f-mesure** et la **moyenne pondérée** de nos deux classes. La macro f-mesure est la moyenne des f-mesures de nos deux classes, elle permet de donner autant d'importance à chaque classe peu importe sa taille dans le corpus. C'est pour cela que malgré la très basse proportion d'annotations D, la macro-mesure obtenue lors du premier essai de l'algorithme était de 0.50. La moyenne pondérée vise elle à évaluer la performance du modèle en prenant en compte les proportions de chaque classe dans le corpus. Si nous cherchions à évaluer le modèle sur sa performance générale à annoter l'accord, alors la moyenne pondérée serait intéressante car la plupart des annotations A, majoritaires dans le corpus, sont correctes. Dans notre cas, sachant que la classe D est bien trop sous-représentée et que nous cherchons malgré tout à la prédire, **la macro f-mesure est la valeur la plus pertinente**.

Nous pouvons premièrement déduire à partir du graphique que la manipulation des poids a permis au modèle de reconnaître des phrases comme appartenant à la classe D. De plus, les ratios les plus efficaces (meilleures macro f-mesure et moyenne pondérée) semblent être les ratios **20 pour 1 et 30 pour 1**. Voici leurs matrices de confusion et métriques respectives :

20 pour 1



30 pour 1



	PRECISION	RAPPEL	F-MESURE
CLASSE A	1	1	1
CLASSE D	0.45	0.24	0.31
ACCURACY	/	/	0.99
MACRO AVG	0.72	0.62	0.66
WEIGHTED AVG	0.99	0.99	0.99

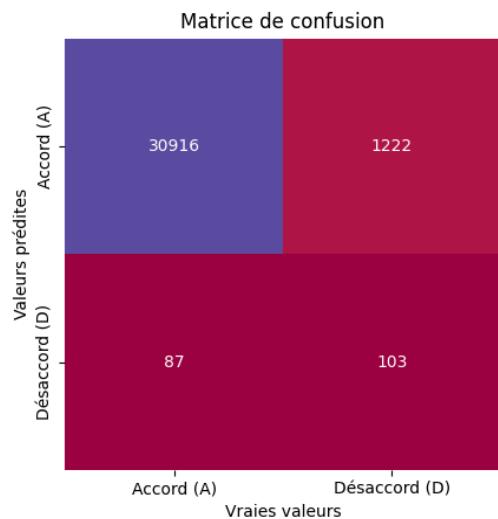
30 pour 1

	PRECISION	RAPPEL	F-MESURE
CLASSE A	1	1	1
CLASSE D	0.31	0.36	0.33
ACCURACY	/	/	0.99
MACRO AVG	0.65	0.68	0.66
WEIGHTED AVG	0.99	0.99	0.99

Les macro f-mesures et les moyennes pondérées de ces deux ratios sont identiques et respectivement à 0.66 et 0.99. Les différences notables se trouvent au niveau des métriques de la classe D et des macro précisions et rappels. La qualité des prédictions de la classe D sont plus ou moins équivalentes dans les deux cas (0.31 contre 0.33) mais le ratio 20 pour 1 présente un déséquilibre plus important entre le score de la précision et celui du rappel. Un modèle disposant de scores de précision et de rappel équivalents est capable d'identifier les cas de vrais positifs correctement tout en évitant un taux trop élevé de faux positifs. Dans le cas du ratio 20 pour 1, il semblerait que le modèle soit performant pour reconnaître les vrais classes D, bien plus que celui du ratio 30 pour 1 (0.45 contre 0.31). À l'inverse, il obtient un rappel plus bas signifiant que plusieurs D réels n'ont pas été reconnus. Nous pouvons observer ces différences de performances en comparant les nombres de **vrais positifs et de faux positifs : 46 contre 69 et 57 contre 154**. Le choix du meilleur ratio en fonction d'un équilibre précision/rappel dépend grandement des prédictions fausses considérées comme étant les plus “coûteuses” lors de l'application du modèle dans des contextes concrets. Il serait plus pertinent dans certains cas de préférer un modèle “équilibré”, avec des précisions et rappels équivalents, plutôt qu'un modèle qui, s'il prédit plus de vrais positifs, prédit également de nombreux faux positifs. **Au vu de l'application de notre modèle, nous pourrions tendre à favoriser un système équilibré.**

En comparaison à ces deux cas les plus performants, voici le cas considéré à partir du graphique comme **le moins performant** (outre le ratio 1 pour 1) :

100 pour 1



	PRECISION	RAPPEL	F-MESURE
CLASSE A	1	0.96	0.98
CLASSE D	0.08	0.54	0.14
ACCURACY	/	/	0.99
MACRO AVG	0.54	0.75	0.56
WEIGHTED AVG	0.99	0.96	0.97

Dans le cas d'un ratio 100 pour 1, le déséquilibre précision/mesure est d'autant plus accentué. Pour **1 D bien reconnu**, environ **12 A sont mal reconnus**. Ce rapport est plus de 5 fois plus élevé que pour les cas les plus performants.

L'augmentation du poids de la classe majoritaire est ainsi une solution envisageable pour compenser le manque d'annotations du corpus étudié. **Malgré tout, si des D sont bien reconnus, cette technique reste une manière artificielle de multiplier nos données qui ne sont donc plus dans leur ensemble “authentiques”**. Enfin, même les ratios les plus performants arrivent à peine à faire que le modèle puisse reconnaître correctement la moitié des phrases annotées D.

D - Modification des hyperparamètres

Enfin, après ces deux essais, nous avons décidé de chercher à obtenir les meilleurs hyperparamètres utilisés par le modèle et ne nécessitant pas de réduction de corpus au préalable. Scikit-Learn offre la possibilité d'entraîner un modèle plusieurs fois à partir de différentes valeurs attribuées à différents hyperparamètres et permettant donc d'identifier ceux étant les plus performants.

```

def entrainement():

    french_stop_words = stopwords.words('french')

    corpus_dataframe = pd.read_csv('corpus_total.csv', header=0, usecols=[2,4], names=['phrase', 'accord'])
    corpus_dataframe = corpus_dataframe.dropna(subset=['phrase'])
    train_corpus, test_corpus=train_test_split(corpus_dataframe, test_size=0.4, random_state=42,stratify=corpus_dataframe['accord'])
    #Cette partie on peut la modifier manuellement
    hyper_parameters = {
        #limite de la fréquence maximal de documents dans laquelle on va permettre un terme
        'tfidvectorizer_max_df': (0.2, 0.4, 0.8),
        #quantité maximale de caractéristiques qu'on va utiliser pour la vectorisation
        'tfidvectorizer_max_features': (None, 5000, 10000, 50000),
        #n-gramme, unigramme
        'tfidvectorizer_ngram_range': ((1, 1), (1, 2)),
        'multinomialnb_alpha': (1e-2, 1e-3)}
    #Je crée le pipeline avec NB
    pipeline = make_pipeline(TfidfVectorizer(stop_words=french_stop_words), MultinomialNB())
    #Je cherche les hyperparamètres : cv: division croisée, n_jobs: il va utiliser tous les processeurs possibles.
    grid_search = GridSearchCV(pipeline, hyper_parameters, cv=5, n_jobs=-1, verbose=1)
    grid_search.fit(train_corpus['phrase'], train_corpus['accord'])
    #J'obtiens le meilleur modèle
    best_model = grid_search.best_estimator_

    #Nous testons à partir de la division "test corpus"
    test_evaluation=best_model.predict(test_corpus['phrase'])
    print("Voici les meilleurs hyperparamètres:{grid_search.best_params_}")
    print("Taux d'accuracy :", accuracy_score(test_corpus['accord'], test_evaluation))

    return test_evaluation, test_corpus

```

Dans l'apprentissage automatique, le modèle est construit selon deux notions qui vont déclarer quelles sont les limites auxquelles il sera soumis. Ce sont les **hyperparamètres** et les **paramètres**. Les premiers correspondent à une configuration plutôt externe. Ceux-ci peuvent être établis par défaut, sur Scikit-learn au moins, ou peuvent être modifiés par l'utilisateur pour trouver un résultat qui s'adapte aux besoins du corpus et aux objectifs de l'apprentissage. Donc, ils sont établis au préalable et ne vont pas changer tout au long de l'apprentissage. Les deuxièmes correspondent à une configuration interne, où le même modèle va apprendre à partir des données qui peuvent être modifiées automatiquement tout au long de la construction du modèle.

Pour trouver les meilleurs hyperparamètres possibles pour notre modèle, nous avons utilisé ***GridSearchCV***, une technique qui prend comme entrée un ensemble des valeurs possibles pour chaque hyperparamètre et évalue chaque combinaison en utilisant la validation croisée, ce qui divise notre corpus en plusieurs plis (cinq, dans notre cas), puis entraîne le modèle avec tous les plis sauf un qui sera dédié au test. Il répète cette même procédure pour chaque division. Même si nous avons divisé notre corpus au préalable en 80% pour l'entraînement et et 20% sur le test , la validation croisée est faite sur les 80% dédiés à l'entraînement.

Nous avons choisi de modifier manuellement deux des hyperparamètres avec ***GridSearchCV***. Nous avons établi des valeurs différentes pour quatre d'entre eux : ***max_df***, ***max_feature***, ***ngram_range*** et ***multinomialnb_alpha***.

Max_df contrôle la quantité de mots à inclure dans le vocabulaire du vecteur selon leurs fréquences dans le corpus. On établit le pourcentage limite d'apparition d'un mot, au-delà de ça de ce dernier, le mot sera ignoré.

```
'tfidfvectorizer__max_df': (0.2, 0.4, 0.8),
```

Max_features établit la quantité maximale de caractéristiques que le vectoriseur va prendre en compte.

```
'tfidfvectorizer__max_features': (5000, 10000, 50000)
```

Ngram_range tient compte de la taille des n-grammes pendant la vectorisation.

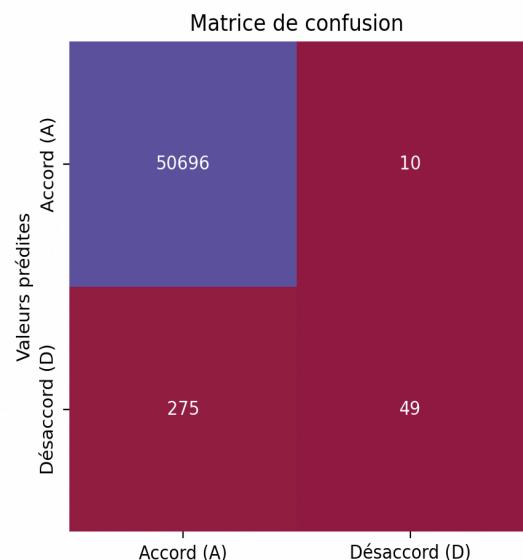
```
'tfidfvectorizer__ngram_range': ((1, 1), (1, 2))
```

Multinomial_alpha contrôle la méthode de **Laplace** qui permet d'éviter les probabilités de 0.

```
'multinomialnb__alpha': (1e-2, 1e-3) # 0.01 et 0.001
```

La matrice de confusion obtenue est la suivante :

Nous nous sommes demandés pourquoi le taux d'accuracy correspondait à 99% même si 275 désaccords avaient été classés comme accords. Normalement, cela se passe quand il y a un grand déséquilibre entre les classes. Notre modèle reconnaît bien la prédominance de la classe majoritaire (accords), ce qui donne une grande exactitude, malgré les prédictions incorrectes de la classe minoritaire (désaccord). De plus, nous avons encore une fois une précision bien plus élevée que le rappel pour les prédictions de la classe D.



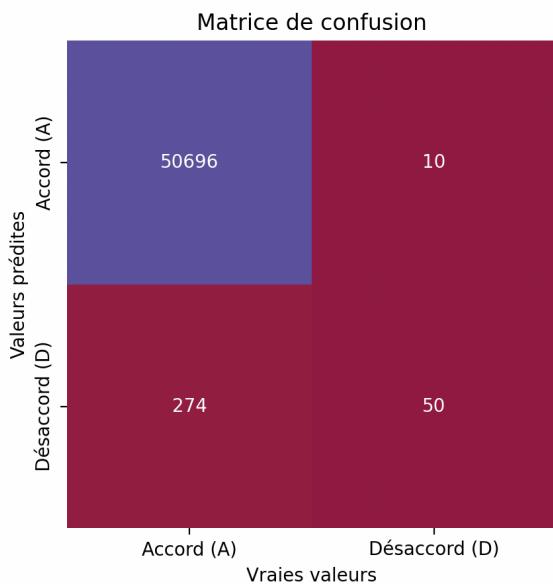
	PRECISION	RAPPHEL	F-MESURE	SUPPORT
CLASSE A	0.99	1	1	50 706
CLASSE D	0.74	0.15	0.25	324
ACCURACY	/	/	0.99	51030
MACRO AVG	0.87	0.58	0.62	51030
WEIGHTED AVG	0.99	0.99	0.99	51030

Les meilleurs hyperparamètres établis ont été :

'Tfidfvectorizer_max_df': 0.2,
'tfidfvectorizer_max_features': 50000,
'tfidfvectorizer_ngram_range': (1, 2),
'multinomialnb_alpha': 0.01.

Ces résultats nous permettent de déduire que le meilleur modèle est celui construit avec des **bigrammes, la moindre estimation de la méthode de Laplace, la moindre quantité possible de classes répétées**, pour réduire la quantité d'accords, et qui **tient compte de toutes les caractéristiques possibles**.

```
hyper_parameters = {
    # limite de la fréquence maximale de documents dans laquelle on va permettre un terme
    'tfidfvectorizer_max_df': (0.1, 0.2, 0.4, 0.8),
    # quantité maximale de caractéristiques qu'on va utiliser pour LA vectorisation
    'tfidfvectorizer_max_features': (5000, 10000, 50000, 80000),
    # n-gramme, unigramme
    'tfidfvectorizer_ngram_range': ((1, 1), (1, 2), (1, 3)),
    'multinomialnb_alpha': (1e-2, 1e-3, 1e-1)}
```



Pour le vérifier, on a répété la procédure avec d'autres valeurs d'hyperparamètres. Comme résultat de cette vérification, nous avons obtenu presque la même matrice de confusion et nous remarquons que les valeurs du premier essai sont presque les mêmes que celles suivant la modification des hyperparamètres.

Cela nous invite à conclure que la quantité inégale d'accords et de désaccords ne nous permet pas d'avoir des résultats satisfaisants, même si nous changeons les hyperparamètres manuellement.

E - Conclusion

Après avoir testé trois stratégies différentes visant à améliorer les performances de l'algorithme Naïve Bayes, nous en sommes arrivés aux conclusions suivantes.

En ne gardant uniquement les essais les plus fructueux, nous obtenons les chiffres ci-dessous.

	MACRO P	MACRO R	MACRO AVG	WEIGHTED AVG	ACCURACY
Réduction de la classe majoritaire	0.88	0.88	0.88	0.88	0.88
Modification des poids	0.65	0.68	0.66	0.99	0.99
Modification des hyperparamètres	0.87	0.58	0.62	0.99	0.99

Au vu de nos chiffres, il semblerait que la solution de la réduction de la classe majoritaire soit la plus efficace. Comme explicité précédemment, les valeurs de macro métriques (précision, rappel et f-mesure) permettent de mieux rendre compte de la performance du modèle à reconnaître la classe minoritaire. Ces résultats sont logiques, réduire la classe majoritaire sert bien sûr à avoir un corpus composé de deux classes à la représentation plus ou moins équivalente. Cette stratégie, s'il elle nous fait perdre un nombre très important de données (plus de 120 000 lignes), montre malgré tout que la reconnaissance de l'accord, comme du désaccord, peut être obtenue à partir d'un entraînement de modèle avec l'algorithme Naïve Bayes. La distribution globale de mots porteurs d'une valeur de désaccord semble donc bien être un paramètre discriminant dans la reconnaissance des oppositions. Un équilibre des classes est cependant recommandé et semble en effet être à l'origine de la bonne performance de notre modèle lorsque les poids des balises D sont intensifiés artificiellement. Si la modification des poids n'est ici pas la solution optimale, elle encourage à une annotation plus fournie du corpus afin d'obtenir des résultats encore plus élevés que lors de l'essai de la réduction de la classe majoritaire. Enfin, si la modification des hyperparamètres semble effectivement permettre une bonne reconnaissance de certains désaccords, l'équilibre entre le rappel et la précision n'est toujours pas atteint et mériterait d'être recherché au vu de l'objectif primaire de notre entraînement. Cet équilibre semble être le plus élevé lors d'une égalité quasi complète du nombre de phrases dans nos classes.

Afin de pouvoir observer et poser des hypothèses au sujet des formes les plus courantes responsables du choix de notre modèle, nous avons cherché à utiliser un autre algorithme évitant toute réduction ou augmentation artificielle de nos données. Il nous fallait ainsi trouver un algorithme de classification plus adapté à des corpus contenant une classe sous-représentée et à des tâches de reconnaissance des émotions / opinions.

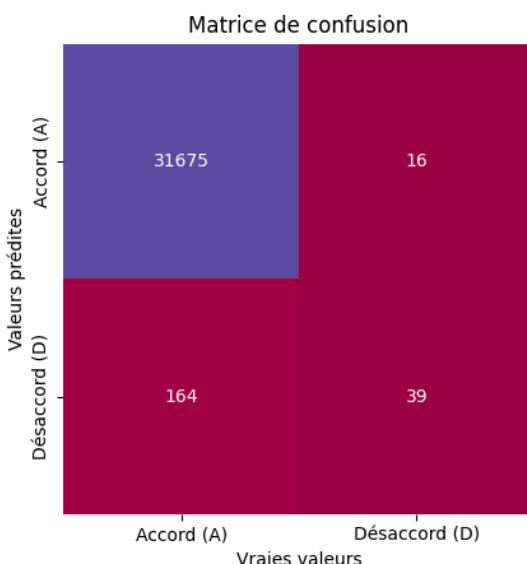
IV - Entraînement Gradient Boosting Classifier

La recherche d'un algorithme plus adapté à notre projet nous a mené à envisager plusieurs solutions. Nous souhaitions utiliser un algorithme capable d'identifier les classes sous-représentées et d'appréhender des données au contenu plus "complexe". Notre corpus, s'il est uniquement composé de texte, contient plusieurs entités pouvant influencer négativement la performance de notre modèle. Nous avons par exemple énormément de mixtures de données numériques parfois liées à des dates ou à des numéros de conventions. Nous pouvons également observer un nombre important de noms propres et de structures imbriquées les unes dans les autres : listes de mesures, mentions de textes légaux, de nombreuses abréviations...

Nous nous sommes donc tournés vers des algorithmes et outils permettant de prendre en compte ces éléments tels que **Support Vector Machines**, **Random Forest** et bien sûr **Gradient Boosting Classifier**.

Ces trois outils n'admettent premièrement pas l'hypothèse probabiliste de Naïve Bayes selon laquelle l'apparition des mots ne dépend pas de son contexte linguistique. Ils utilisent de plus une séquence d' entraînement permettant de contrer les erreurs du modèle précédent. Ils restent cependant plus prône à la surestimation de classes et nécessitent un réglage minutieux de leurs hyperparamètres. **Après quelques essais plus ou moins équivalents, nous avons décidé de développer l'utilisation du Gradient Boosting Classifier semblant être le plus performant.**

Afin d'illustrer rapidement ces essais, voici un aperçu des résultats obtenus à l'aide du SVC :



Liste des features les plus importants :

vote: 0.008860667987032209
fait: 0.009075575600224022
monsieur: 0.009636115737447335
président: 0.009715865126165847
être: 0.009864142129157365
contre: 0.010889346028019338
cette: 0.011125297928701077
étudiants: 0.011172758263752242
conseil: 0.011472158954444325
université: 0.015260591639515188

Comme vous le verrez au sein de cette partie, les résultats obtenus avec le SVC sur notre corpus d'origine ne sont pas meilleurs que ceux obtenus à l'aide du Gradient Boosting Classifier.

Cependant, nous constatons que parmi les features, et donc les termes extraits par le modèle, beaucoup ne portent pas d'indice d'opinion. Des mots comme université ou étudiants, évidemment très présents dans notre corpus, pourraient ainsi grandement influencer le choix du classifieur et participer à une mauvaise classification de nos phrases. Afin de résoudre ce problème, nous avons notamment tenté d'exclure les termes les plus fréquents de notre corpus. Nous avons utilisé *max_df*

afin d'exclure les termes présents dans au moins 0.03 % de notre corpus (représentant ainsi au moins 40 phrases environ). Ce nombre a été choisi après de nombreux essais durant lesquels nous avons fait en sorte de garder certaines termes comme "contre" tout en excluant d'autres présents dans la liste ci-dessus. Nous obtenions malgré tout des résultats équivalents avec une liste de features importants différente :

```
commission: 0.005520348855270634
année: 0.005622870177876892
deux: 0.005895862884250919
si: 0.0059013094318053376
ça: 0.006088670888382259
budget: 0.006129475913806335
projet: 0.006431185785765948
madame: 0.006883618537698903
abstentions: 0.007717304240997661
contre: 0.010938590441068792
```

Ces features différentes n'ont malheureusement que très peu améliorer la performance du modèle. Nous nous sommes donc détourné du SVC en faveur du Gradient Boosting Classifier. Au vu de ces informations et des résultats obtenus à l'aide de l'algorithme Naïve Bayes, nous pouvons admettre l'hypothèse suivante :

L'utilisation du Gradient Boosting Classifier permettra une meilleure reconnaissance des désaccords à partir de notre corpus complet (sans réduction ou augmentation de classes) mais risque de créer des situations d'overfitting', c'est à dire de 'sur-reconnaissance' de nos balises.

A - Fonctionnement

Le Gradient Boosting permet de réaliser des tâches variées (dont la classification) même s'il est généralement plus souvent utilisé pour des prédictions de données continues (régression). Nous pouvons ainsi y avoir recours dans de nombreux cas différents.

Le principe de Boosting repose sur l'obtention d'une fonction de prédiction correspondant à la somme des prédictions les plus faibles. Notre fonction vient utiliser d'autres fonctions moins performantes, car relevant de modèles assez faibles, afin de s'alimenter et de s'améliorer.

Le Gradient Boosting Classifier est donc un algorithme se reposant sur les arbres de décision.
Essayons d'expliquer son fonctionnement étape par étape en utilisant notre corpus en exemple.

Un premier "weak learner" , c'est-à-dire un modèle peu performant, effectue une prédiction initiale des phrases annotées en A et D très généralement réalisée à l'aide d'un arbre de décision. Cette prédiction obtenue peut être modélisée comme étant le logarithme des chances d'avoir une balise D. En prenant les proportions de l'ensemble d'entraînement obtenues lors de la division train/test Naïve Bayes :

1ère prédiction :

$$\ln\left(\frac{\text{total } D}{\text{total } A \text{ et } D}\right) = \ln\left(\frac{715}{95\,435 + 715}\right) = \ln(0.0074) \simeq -4,9$$

Nous pouvons ensuite transformer cette valeur en probabilité à l'aide de la formule :

Probabilité :

$$\frac{e^{\ln(\text{odds})}}{1 + e^{\ln(\text{odds})}} = \frac{e^{-4,9}}{1 + e^{-4,9}} \simeq 0.007$$

Notre modèle admet donc qu'il y a une probabilité de 0.007 qu'une phrase soit annotée en désaccord. Nous pouvons maintenant évaluer à quel point cette prédiction est "mauvaise" en calculant les "résiduels", la différence entre les données réelles et les données prédites. On attribue 1 à la classe D (car c'est elle dont nous avons calculé la prédiction) et 0 pour la classe A :

Pour les D : $1 - 0.007 \simeq 0.99$

Pour les A : $0 - 0.007 \simeq -0.007$

Ces résultats expliquent l'importance de la mauvaise reconnaissance des classes D. La classe A est elle complètement reconnue mais sa probabilité est légèrement surestimée.

Le Gradient Boosting Classifier va venir se baser sur ces mesures résiduels initiales et tenter de les améliorer en entraîner plusieurs modèles successivement. Des arbres de décisions vont être créés à partir de différents nœuds déterminés par l'algorithme lui-même (après vectorisation des phrases ou en fonction d'autres colonnes dans d'autres cas que le nôtre) et pour chaque nœud et classe seront obtenus de nouveaux résiduels.

Admettons par exemple que notre algorithme décide de diviser notre ensemble entre phrases courtes et longues. Pour chacun de ces sous-ensembles, les annotations A ou D des phrases seront prédites et permettront d'obtenir des résiduels. Si ces nouvelles valeurs semblent plus prometteuses que les résiduels de la prédiction initiale, alors nous mettons à jour ces dernières.

Imaginons que pour l'arbre phrases longues/courtes, nos résiduels soient les suivants :

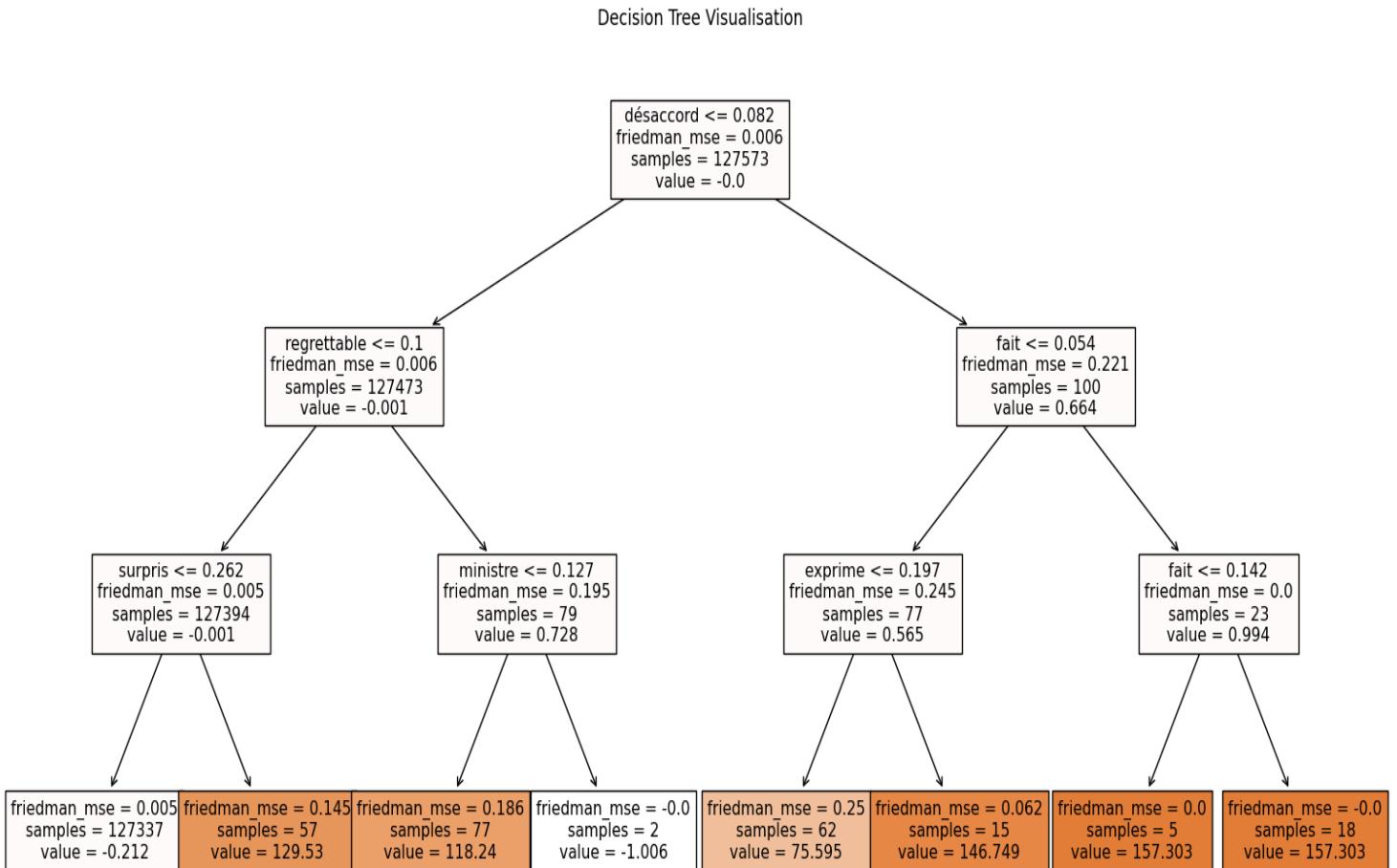
Pour les D : 0.96

Pour les A : -0.005

Nous pouvons voir que le modèle semble avoir mieux reconnu nos deux classes, nous pouvons ainsi additionner ces résiduels à notre prédiction initiale (sous valeur probabiliste). Les nouveaux résiduels sont multipliés par une valeur attribuée à l'hyper paramètre **learning rate** (généralement 0.1), permettant d'évaluer la contribution de l'arbre à la performance du modèle.

Nouvelle probabilité : $0.007 + 0.1 \times 0.96 = 0.103$

La probabilité d'annotation a augmenté grâce au second arbre créé par l'algorithme. Le Gradient Boosting va effectuer cette opération de nombreuses fois afin de tenter d'améliorer la prédiction précédente à l'aide de nouveaux arbres basés sur des nœuds différents et compensant la classe sous-représentée de l'arbre précédent. *Voici une représentation d'un arbre créé à partir de notre corpus :*



B - Premier essai

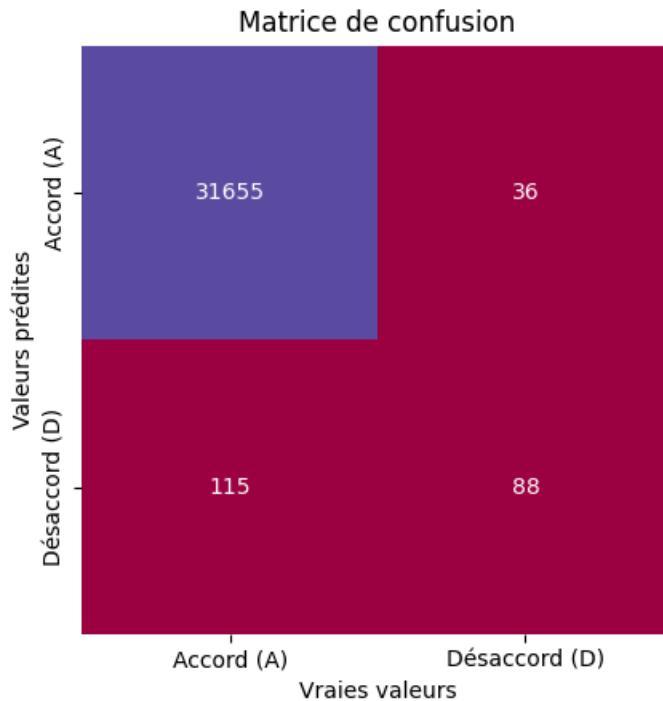
L'entraînement du modèle à partir de l'algorithme s'est fait à partir des fonctions et méthodes proposées par Scikit-Learn sur leur [page dédiée](#). Le fonctionnement du script utilisé pour le Gradient Boosting Classifier s'est avéré semblable à celui de l'algorithme Naïve Bayes et nous a donc permis de pouvoir ré-utiliser notre code existant en appliquant les modifications nécessaires.

Il a notamment fallu adapter notre ligne permettant de convertir nos données textuelles en vecteurs :

```
# Créer une pipeline
model = make_pipeline(TfidfVectorizer(stop_words=french_stop_words),
                      GradientBoostingClassifier(n_estimators=100, max_depth=3, learning_rate=0.1))
```

Les hyperparamètres de bases (également par défaut) sont explicités ici : ***n estimators***, ***max depth*** et ***learning rate***.

Le premier correspond au nombre de “boost” devant être effectués par l’algorithme, donc au nombre d’attributs devant être considérés par l’algorithme pour améliorer le modèle. **Max depth** correspond à la profondeur maximum des arbres créés. Enfin, le **learning rate**, comme expliqué plus haut, représente la contribution de chaque boost à la performance du modèle. Les valeurs choisies ici sont celles par défaut définies par la librairie Scikit-Learn. Ci-dessous vous trouverez la matrice résultante de ce premier essai.



Si nous lisons cette matrice en fonction de la classe D :

En bas à droite : 88 Vrais Positifs (VP), 88 D ont été reconnus comme des D.

En bas à gauche : 115 Faux Négatifs (FN), 115 D ont été mal reconnus et ont été annotés A.

En haut à gauche : 31 655 Vrais Négatifs (VN), tous les A reconnus en A. Donc tous les non D reconnus comme tels.

En haut à droite : 36 Faux Positifs (FP), 36 A ont été reconnus à tort comme D.

La matrice nous permet ainsi d’obtenir les métriques suivantes :

	PRECISION	RAPPHEL	F-MESURE	SUPPORT
CLASSE A	1	1	1	31 691
CLASSE D	0.71	0.43	0.54	203
ACCURACY	/	/	1	31 894
MACRO AVG	0.85	0.72	0.77	31 894
WEIGHTED AVG	0.99	1	0.99	31 894

Nous pouvons ici remarquer que contrairement à l’algorithme Naïve Bayes de base, le Gradient Boosting Classifier (quand il est utilisé avec ses hyperparamètres par défaut), permet la reconnaissance correcte de certaines phrases annotées D. Nous pouvons également observer que les valeurs de précision et de rappel sont déséquilibrées dans le cas de ces dernières. Le modèle semble reconnaître un nombre trop important de faux positifs par rapport au nombre de D réels reconnus semblant ainsi corroborer une partie de notre hypothèse.

Les valeurs de micro et macro-moyennes restent cependant considérablement meilleures que celles obtenues lors de l'entraînement du modèle avec Naïve Bayes et notre corpus initial (sans réduction ou augmentation de classe).

Avant de pouvoir comparer ces résultats à ceux des meilleurs chiffres obtenus précédemment à l'aide de Naïve Bayes, nous allons tenter d'obtenir les valeurs d'hyper paramètres optimales afin d'améliorer notre modèle.

C - Modification des hyperparamètres

La documentation de Scikit Learn indique que les valeurs par défaut des hyper paramètres du Gradient Boosting classifier sont généralement les plus efficaces pour de la classification de textes. Nous avons ainsi décidé de conserver les mêmes valeurs de ces derniers mais de changer celles des hyperparamètres évoqués lors de l'obtention des hyperparamètres de Naïve Bayes.

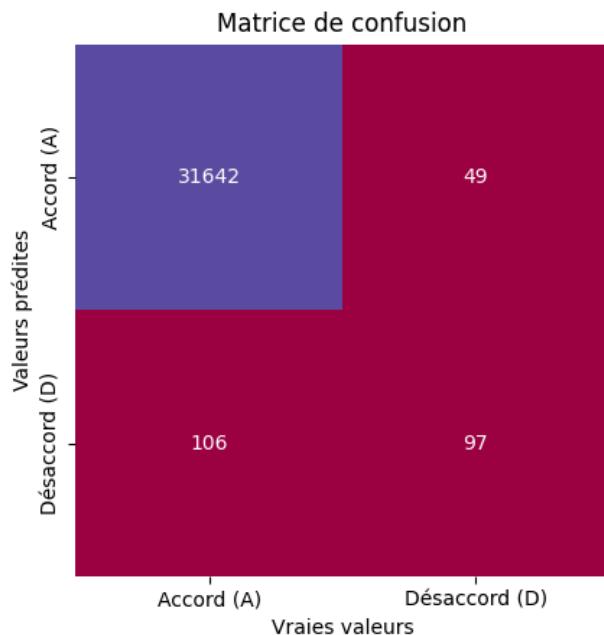
Ces derniers étaient :

- '*Tfidfvectorizer_max_df*' - Taille de notre vocabulaire.
- '*Tfidfvectorizer_max_features*' - Nombre de features maximum.
- '*Tfidfvectorizer_ngram_range*' - Taille des n-grams.

```
parameters = [  
    #limite de la fréquence maximal de documents dans laquelle on va permettre un terme  
    'tfidfvectorizer_max_df': (0.5, 0.7),  
    #quantité maximale de caractéristiques qu'on va utiliser pour la vectorisation  
    'tfidfvectorizer_max_features': (None, 10000),  
    #n-gramme, unigramme  
    'tfidfvectorizer_ngram_range': ((1, 1), (1, 2)),  
    'gradientboostingclassifier_n_estimators': [150],  
    'gradientboostingclassifier_max_depth': [3],  
    'gradientboostingclassifier_learning_rate': [0.1]
```

Nous avons ici décidé d'utiliser un **vocabulaire généralement grand**, un **nombre de features plus élevé** et des **monogrammes et bi-grammes**.

L'entraînement de nos modèles a permis d'établir la matrice de confusion suivante :



Ces métriques nous ont ensuite permis d'obtenir les chiffres suivants, ceux de l'essai sans modification des hyper paramètres sont indiqués en rouge pour comparaison :

	PRECISION	RAPPHEL	F-MESURE	SUPPORT
CLASSE A	1 1	1 1	1 1	31 691
CLASSE D	0.66 0.71	0.48 0.43	0.56 0.54	203
ACCURACY	/	/	1 1	31 894
MACRO AVG	0.83 0.85	0.74 0.72	0.78 0.77	31 894
WEIGHTED AVG	0.99 0.99	1 1	0.99 0.99	31 894

Nous pouvons ainsi observer que les valeurs obtenues après modification des hyperparamètres sont légèrement supérieures mais souffrent d'un rappel un peu plus bas. La diminution de la performance du modèle (rappel) et son augmentation (précision) creuse un peu plus l'écart entre précision et rappel semblant être commun à la plupart de nos essais. Il est donc pertinent de considérer cet écart comme un aspect non compensé par les moyennes légèrement meilleures obtenues. Il est également raisonnable de questionner les valeurs obtenues, ces dernières variant légèrement à chaque essai. Cependant, ces variations sont rarement si importantes et nos valeurs semblent plus au moins constantes après plusieurs vérifications.

Les valeurs des hyperparamètres ayant mené à cette matrice sont les suivants :

```
'gradientboostingclassifier_learning_rate': 0.1,  
'gradientboostingclassifier_max_depth': 3,  
'gradientboostingclassifier_n_estimators':100,  
'tfidfvectorizer_max_df': 0.5,  
'tfidfvectorizer_max_features': None,  
'tfidfvectorizer_ngram_range': (1, 2).
```

L'algorithme semble ainsi donner des résultats plus satisfaisants lorsque le vocabulaire est plus réduit, le nombre de features maximum n'est pas spécifié, et les n-grammes sont composés de monogrammes et de bigrammes.

Ainsi, si nos résultats ne sont pas à première vue meilleurs que ceux obtenus lors de la rééducation de la classe majoritaire avec Naïve Bayes, ils méritent de représenter notre corpus dans son intégralité. Nos classes désaccords n'ont pas été compensées par duplication et celles d'accords ont été conservées dans leur intégralité. Si nous pouvons maintenant avoir une vue d'ensemble des métriques obtenues par chacun de nos essais, il reste à tenter d'attribuer ces scores à des formes les ayant influencées.

V - Enrichissement du corpus

Comme mentionné dans la présentation, le problème principal est que l'annotation du corpus n'est pas finie et qu'il n'y a que très peu de phrases annotées « D ».

Une solution qui vient à l'esprit est de compléter l'annotation du corpus. Cependant, ce corpus, qui contient 120 milles phrases, est (ou devra être) annoté à la main, par l'équipe du projet d'ArchiveU. Notre travail, en revanche, n'est pas l'annotation du corpus, mais l'entraînement d'un modèle de classification. C'est pour cela que nous avons choisi de compléter l'annotation de manière automatique, en suivant le guide d'annotation fourni avec le corpus.

Dans la présentation de ce corpus, il y a une page dans laquelle on nous explique en quoi une phrase est considérée comme exprimant le désaccord. Dans le guide, il y a notamment ces deux listes-là :

Liste des verbes de discours indirect

Accuse/avoue/condamne/condamnent/conteste/contestent/ contredit/ne convainc pas/craint/ critique/défend (se défend, s'en défend)/défendent/dément/dénonce/dénoncent/déplore/ déplorent/désapprouve/déssole (loc se désole, X désole loc)/doute/s'émeut de /s'étonne que/ s'étonnent que//s'inquiète/s'inquiètent/s'insurge contre/interpelle/menace/menacent/ne nie pas/objecte/s'offusque de/s'oppose à/perçoit mal-ne perçoit pas/se plaint (loc)/se préoccupe/prétend/proteste/protestent/récuse/refuse/regrette/regrettent/rejette/rejettent/ reproche/réprouve/rétorque/

La première liste comporte principalement des verbes introducteurs de discours. Comme « avouer, contester, craindre, défendre », etc. Elle est suivie de d'autres listes des expressions composées de ces verbes.

Comme nous pouvons le remarquer, cette liste est déterminée de manière plutôt sémantique et non pas lexicale. Parfois, le mot tout seul ne peut pas exprimer le désaccord (comme « percevoir » ou « préoccuper » ou « convaincre »), et il faut la locution entière pour exprimer le désaccord (« ne convainc pas », « se préoccuper » et « perçoit mal »). Il faut vraiment regarder le sens du verbe, comment il est utilisé, pour déterminer son sens. Aussi, le verbe est plus polysémique. Un verbe peut avoir un sens très différent s'il s'emploie tout seul, dans la forme passive, au conditionnel, ou dans la forme pronominale. Le critère est « sémantique » aussi en raison de la présence de la négation. Pour les deux phrases suivantes : « je suis convaincu » et « je ne suis pas convaincu », on est d'accord que la première exprime l'accord et la deuxième le désaccord. Ce n'est pas seulement la présence ou l'absence du mot « convaincre » qui détermine si une phrase exprime l'accord ou le désaccord, mais la locution entière. Ce critère, trop sémantique, est difficile à implémenter pour le TAL.

aberrant, absurde, abusif (propos) affolant ambigu antisociale tout à fait anormal/	incongru impossible inadmissible incohérent incorrect incompatible incompréhensible indécent inéquitable injuste inquiétant	mal à propos peu courtoise pas en équilibre manque de clarté pas admissible pas judicieux pas sain/malsain pas logique pas normal
bizarre choquant contestable		

Il nous reste la deuxième liste, qui est une liste d'adjectifs bien que cette liste contienne encore pas mal de négations, qui sont tous les mots précédés de « pas ». Mais nous voyons que cette liste est plus transparente. Presque tous les adjectifs ici sont des adjectifs qualificatifs, donc pour la plupart du temps le mot tout seul apporte une émotion assez forte, qu'elle soit positive ou négative. Aussi, pour les adjectifs, puisqu'il y a moins de flexion et que la flexion est purement grammaticale, ils sont beaucoup moins polysémiques.

Cette liste est donc plus lexicale et moins sémantique. La présence d'un adjectif négatif peut assez bien déterminer si une phrase exprime le désaccord. Nous avons donc choisi cette liste pour enrichir le corpus.

Il faut donc encore voir comment on détermine notre liste d'adjectifs. La première chose à faire c'est d'enlever les négations et en général tous les adverbes (« très », « particulièrement », etc.) Mais puisque les négations sont enlevées, il faut supprimer l'adjectif en entier (« pas logique » exprime le désaccord, mais « logique » n'exprime rien du tout. Notons qu'on a encore « illogique » dans la liste.) Il y a eu ensuite une discussion dans le groupe pour déterminer en commun une liste de mots (qu'on appelle « angry words »). Voici le résultat :

1	aberrant
2	absurde
3	abusif
4	affolant
5	ambigu
6	anormal
7	antisocial
8	bizarre
9	choquant
10	contestable
11	contradictoire
12	dangereux
13	décalé
14	dératoire
15	discriminatoire

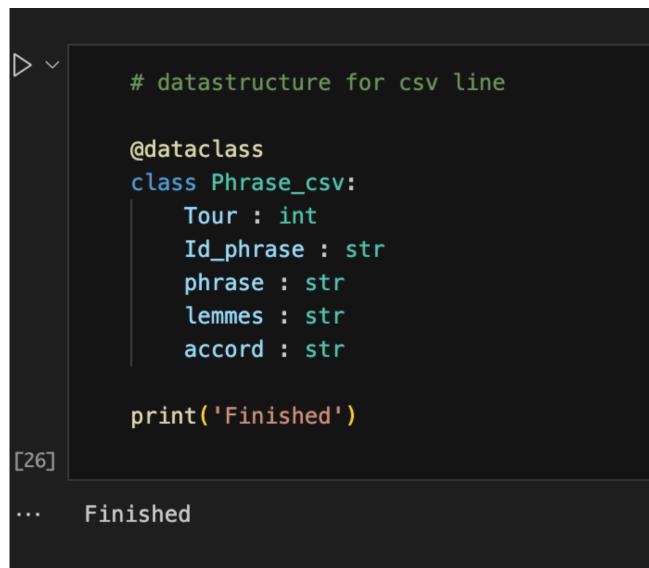
Au total 74 adjectifs sont inclus dans la liste.

Nous avons travaillé sur le csv déjà généré à partir du corpus. Voici sa structure :



```
Tour,Id_phrase,phrase,lemmes,accord
1,weban-upn_PV_CA_2013-12-16_TEI.s1,Le Président
1,weban-upn_PV_CA_2013-12-16_TEI.s2,Nous vous int...
2,weban-upn_PV_CA_2013-12-16_TEI.s3,M. BACQUE et...
2,weban-upn_PV_CA_2013-12-16_TEI.s4,M. POLIT à M...
2,weban-upn_PV_CA_2013-12-16_TEI.s5,M. BOUSLIMI à...
2,weban-upn_PV_CA_2013-12-16_TEI.s6,Mme GUILLMAN...
2,weban-upn_PV_CA_2013-12-16_TEI.s7,M. SARKOZY à...
2,weban-upn_PV_CA_2013-12-16_TEI.s8,M. BRUNET à M...
```

Le script utilisé pour l'enrichissement de l'annotation se nomme **annotation.py**. Il faut juste regarder l'en-tête du tableau. Pour chaque phrase, il y a cinq attributs : tour, id_phrase, phrase, lemmes, accord. Nous avons donc créé une dataclass qui reproduit cette structure :



```
# datastructure for csv line

@dataclass
class Phrase_csv:
    Tour : int
    Id_phrase : str
    phrase : str
    lemmes : str
    accord : str

    print('Finished')
[26]
...   Finished
```

Pour faire un test, nous avons choisi de prendre les premières 1000 phrases. Pour donner une idée, le script prend 30 secondes pour traiter 1000 phrases. Nous avons 120,000 phrases, et ça va prendre une heure pour traiter tout le corpus.

```

> ^
    # create a list of all phrases

    list_phrases = []

    count = 0
    for index, row in totaldata.iterrows():
        if count > 1000:
            break
        else:
            phrase = Phrase_csv(Tour=row['Tour'],
                                 Id_phrase=row['Id_phrase'],
                                 phrase=row['phrase'],
                                 lemmes=row['lemmes'].lower(),
                                 accord=row['accord'])
            list_phrases.append(phrase)
            count +=1

    print('Finished')
[28]
...   Finished

```

Ensuite, j'ai compté les phrases qui sont notées désaccord. Parmi 1000 phrases, il y en a deux. Cela correspond à 0,2%. Dans le bilan qu'on avait fait, il y a en a 0,7%.

```

> ^
    # count number of angry phrases

    nb_angry_phrases = 0

    for phrase in list_phrases:
        if phrase.accord == 'D':
            nb_angry_phrases += 1

    print(f'Number of angry phrases: {nb_angry_phrases}')
[32]
...   Number of angry phrases: 2

```

Ensuite, on a mis toutes les phrases qui contiennent les angry words en désaccord.

```

for phrase in list_phrases:
    if phrase.accord == 'D':
        continue
    else:
        for angry_word in angry_words:
            if angry_word in phrase.lemmes:
                phrase.accord='D'

nb_angry_phrases = 0

for phrase in list_phrases:
    if phrase.accord == 'D':
        nb_angry_phrases += 1

print(f'Number of angry phrases: {nb_angry_phrases}')

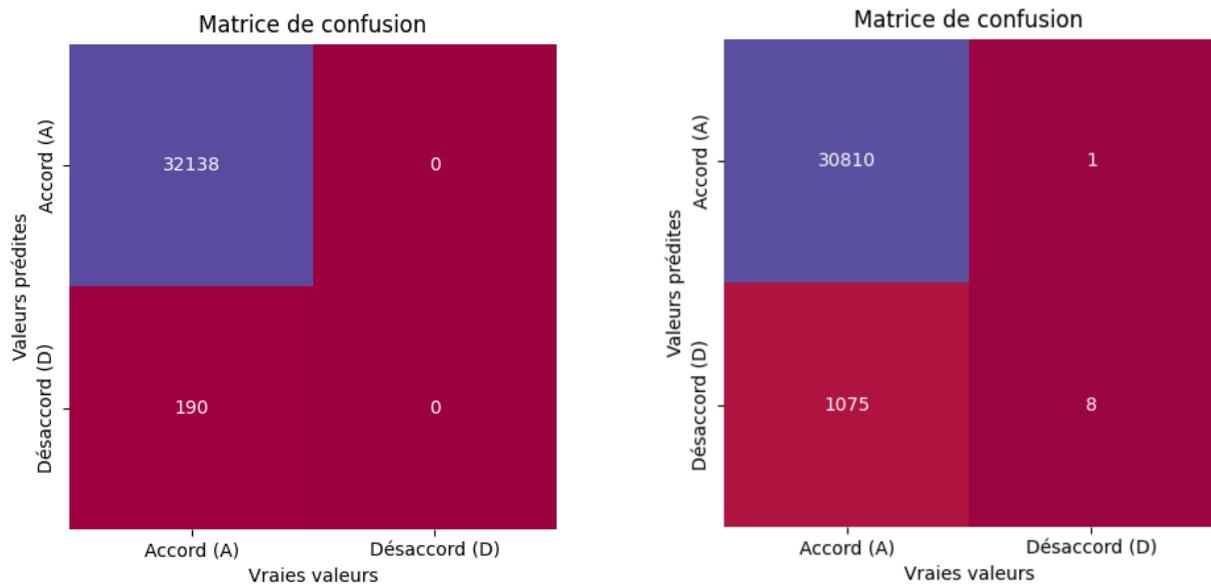
```

Number of angry phrases: 25

Sur l'ensemble du corpus, on a :

	Phrases	Désaccord	% de phrase D
Début	127573	905	0,7 %
Après enrichissement	127573	4431	3,4 %

Ensuite, nous avons utilisé le nouveau corpus enrichi pour entraîner les modèles (*nommé corpus_annotation.py*). Voici une comparaison des résultats pour naïve bayes.



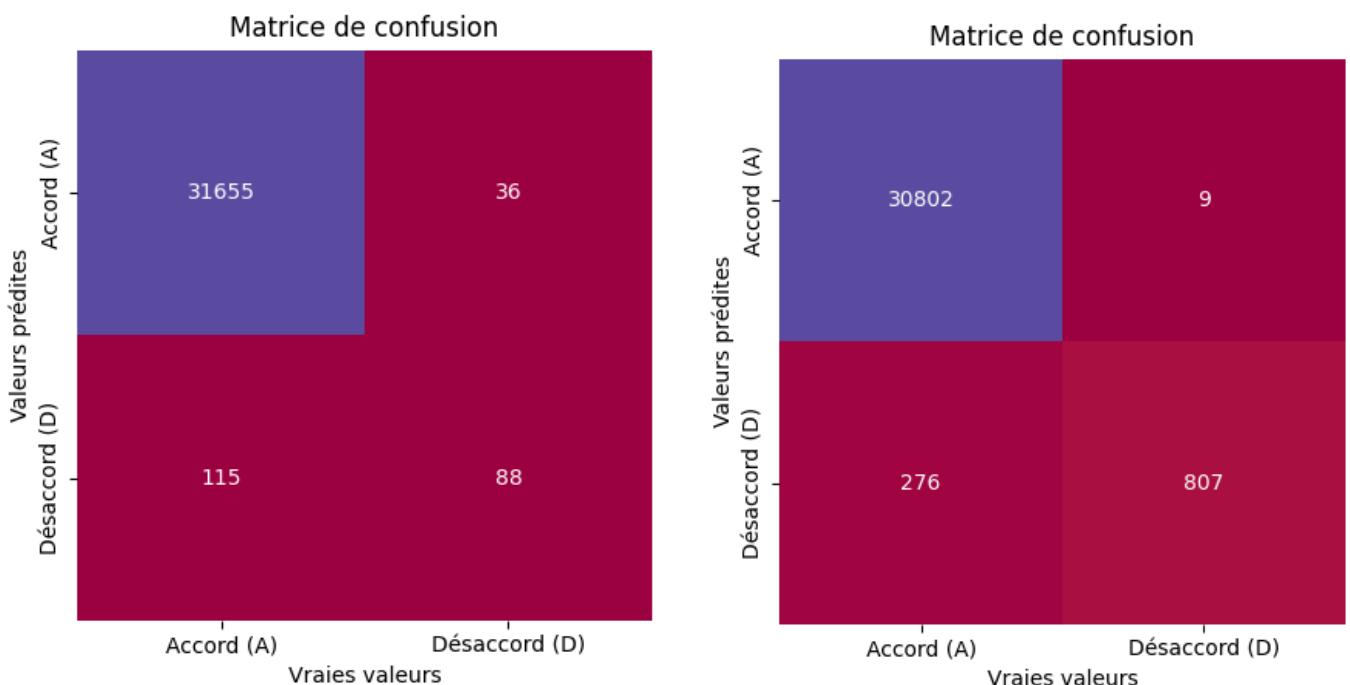
CLASSE A	0.99	1	1	32 138
CLASSE D	0	0	0	190
ACCURACY	/	/	0.99	32 328
MACRO AVG	0.50	0.50	0.50	32 328
WEIGHTED AVG	0.99	0.99	0.99	32 328

CLASSE A	0.97	1	0.98	30 811
CLASSE D	0.89	0.01	0.01	1083
ACCURACY	/	/	0.97	31 894
MACRO AVG	0.93	0.50	0.50	31 894
WEIGHTED AVG	0.96	0.97	0.95	31 894

On remarque en premier une baisse de précision (et accuracy). C'est normal parce qu'au début le modèle classe toutes les phrases en "accord", et que ces phrases représentent 99% du corpus. Accuracy ne peut donc pas être inférieur à 99%.

Ensuite, on remarque que le modèle commence à reconnaître certains désaccords, même si avec un rappel qui est très bas (seulement 1% des désaccords sont reconnus). Mais c'est le signe d'une amélioration et que le modèle commence à fonctionner.

Ensuite on a la comparaison pour gradient boosting.



	PRECISION	RAPPEL	F-MESURE	SUPPORT
CLASSE A	1	1	1	31 691
CLASSE D	0.71	0.43	0.54	203
ACCURACY	/	/	1	31 894
MACRO AVG	0.85	0.72	0.77	31 894
WEIGHTED AVG	0.99	1	0.99	31 894

	PRECISION	RAPPEL	F-MESURE	SUPPORT
CLASSE A	0.99	1	1	30 811
CLASSE D	0.99	0.75	0.85	1083
ACCURACY	/	/	0.99	31 894
MACRO AVG	0.99	0.87	0.92	31 894
WEIGHTED AVG	0.99	0.99	0.99	31 894

On voit que les résultats sont beaucoup améliorés. C'est l'une des meilleures solutions qu'on a, qui est comparable à celle de réduction de corpus (cf. III.B).

Cela montre que la solution d'enrichir le corpus et d'entraîner le modèle avec l'algorithme du gradient boosting est une solution qui marche bien. Il est curieux qu'elle marche beaucoup mieux que la solution d'enrichir le corpus et de l'entraîner avec naïve bayes.

Bien évidemment, cette approche peut encore être améliorée. Nous avons décidé la liste des "angry words" ensemble, à partir de la liste fournie à nous. Ensuite, on ne tient pas compte des verbes, ou des expressions de négations. Mais la comparaison des résultats montre déjà qu'enrichir les annotations est une solution qui nous mène à des améliorations.

VI - Formes porteuses du désaccord

Comme abordé lors de la présentation de la convention d'annotation, les valeurs de désaccords sont très souvent portées par des noms, adjectifs et autres verbes permettant d'exprimer un discours indirect. Si cette valeur tend à être plus ou moins "objective" pour certains tokens, elle reste compliquée à évaluer sans un contexte linguistique plus étendu : groupes nominaux, phrases...

Un des schémas semblant être le plus récurrent dans nos résultats semble être le déséquilibre des valeurs de précision et de rappel de notre modèle. Si ce déséquilibre est parfois nécessaire (dans le cas de tests médicaux par exemple), il tend à desservir notre objectif de classification binaire. Un équilibre est alors à trouver entre les modèles menant aux scores les plus élevés et ceux aux valeurs de précision et de rappel plus équilibrées.

Afin de voir l'importance de l'équilibre de nos valeurs, nous avons décidé de pouvoir identifier certaines formes récurrentes dans nos quatres catégories obtenues après chaque entraînement :

Les vrais positifs : les cas recherchés bien reconnus (D reconnus en D).

Les faux positifs : les cas reconnus mais non corrects (A reconnus en D) -> sur-estimation.

Les vrais négatifs : les cas non recherchés et non reconnus (A reconnus en A).

Les faux négatifs : les cas non recherchés et reconnus (D reconnus en A) -> sous-estimation.

Quelles formes reviennent le plus pour chaque catégorie et chaque modèle ?

A - Script de catégorisation

Chaque entraînement mené à l'aide de nos scripts permet la création de listes contenant nos phrases ainsi que leurs valeurs d'accord réelles et de valeurs prédites. Si les matrices de confusion obtenues à l'aide de ces valeurs permettent déjà de donner une idée de la performance du modèle, nous voulions avoir accès aux phrases appartenant à chacune de nos catégories.

Nous avons donc rajouté une fonction à l'ensemble de nos scripts d'entraînement se présentant ainsi :

```

def obtention_phrases(test_evaluation, test_corpus):
    VP = []
    FP = []
    VN = []
    FN = []

    for phrase, vraie_valeur, prediction in zip(test_corpus["phrase"], test_corpus["accord"], test_evaluation):

        if vraie_valeur == 'D' and prediction == 'D':
            VP.append((phrase, vraie_valeur, prediction))
        elif vraie_valeur == 'A' and prediction == 'D':
            FP.append((phrase, vraie_valeur, prediction))
        elif vraie_valeur == 'A' and prediction == 'A':
            VN.append((phrase, vraie_valeur, prediction))
        elif vraie_valeur == 'D' and prediction == 'A':
            FN.append((phrase, vraie_valeur, prediction))

    with open("data_resultats/data_prediction_gbc.csv", "w") as fichier:
        structure = csv.writer(fichier, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        structure.writerow(["phrase", "valeur réelle", "valeur prédictive"])
        if len(VP) > 0:
            structure.writerow(["VRAIS POSITIFS", " ", " "])
        for phrase in VP:
            structure.writerow([phrase[0], phrase[1], phrase[2]])
        if len(FP) > 0:
            structure.writerow(["FAUX POSITIFS", " ", " "])
        for phrase in FP:
            structure.writerow([phrase[0], phrase[1], phrase[2]])
        if len(VN) > 0:
            structure.writerow(["VRAIS NÉGATIFS", " ", " "])
        for phrase in VN:
            structure.writerow([phrase[0], phrase[1], phrase[2]])
        if len(FN):
            structure.writerow(["FAUX NÉGATIFS", " ", " "])
        for phrase in FN:
            structure.writerow([phrase[0], phrase[1], phrase[2]])

```

Nous créons quatre listes correspondant à chacune de nos catégories dans lesquelles nous rajoutons nos phrases en fonction de leurs valeurs d'accord réelles et prédites. Nous créons ensuite un fichier csv recensant l'ensemble de nos phrases catégorisées en fonction de leurs bonnes ou mauvaises classifications par le modèle (stockés dans *data_resultats*). Nous utilisons ensuite ce fichier afin d'obtenir les formes les plus récurrentes de chaque catégorie. Ces deux étapes auraient pu être réalisées dans un même programme ou sans la création d'un fichier csv intermédiaire, seulement, ces derniers nous permettraient d'avoir une meilleure vue d'ensemble des types de phrases classées par notre modèle (visualisation complexe avec un corpus aussi volumineux que le nôtre).

L'obtention des formes les plus récurrentes se fait à l'aide d'un programme nommé *stat_occurrences.py*. Le fonctionnement de ce dernier est décrit ci-dessous :

```

def top_expressions(categorie, num):
    # https://practicaldatascience.co.uk/machine-learning/how-to-use-count-vectorization-for-n-gram-analysis

    texte = " ".join(phrase) for phrase in categorie] # On doit transformer notre texte en un élément d'une liste
    french_stop_words = stopwords.words('french')
    vectorisation = CountVectorizer(ngram_range = (num, num), max_features = 100, stop_words=french_stop_words)
    # Donc on vectorise tout notre texte et on donne des scores à chaque mot, nous établissons la taille des bi-grams qu'on veut
    sac_de_mots = vectorisation.fit_transform(texte) # On "apprend" le vocabulaire
    sum_words = sac_de_mots.sum(axis = 0)
    words_freq = [(word, sum_words[0, i]) for word, i in vectorisation.vocabulary_.items()]
    words_freq = sorted(words_freq, key = lambda x: x[1], reverse = True) # On obtient les n-grams en fonction de leurs fréquences
    print(words_freq[:10])
    return words_freq[:10]

```

À l'aide du tutoriel présenté sur la page donnée en commentaire, nous avons pu faire en sorte de récupérer les expressions (n-grams dont la longueur est spécifiée en argument) les plus récurrentes par catégorie (également spécifiée en argument). Nous avons pour cela préalablement créé des listes : VP, FP, FN et VN à partir du fichier csv créé plus tôt. Nous avons ensuite eu recours à l'objet *Count Vectorizer* proposé par la librairie Scikit-Learn afin de transformer notre texte en format numérique. Chaque n-gramme sera associé à son nombre d'occurrences dans le corpus.

Nous faisons également en sorte de ne pas prendre en compte les *stop words* et de définir nous même la taille des n-grammes semblant être les plus pertinents. Une fois obtenus, nous ne conservons que les 10 formes les plus récurrentes et les illustrons à l'aide d'une représentation de diagrammes en camembert. Enfin, les phrases utilisées ici sont celles composées de lemmes. Nous avons décidé d'obtenir ces derniers uniquement pour les modèles nous ayant donné les meilleurs résultats lors de nos essais d'entraînement : *Naïve Bayes (avec réduction de la classe majoritaire)* et *Gradient Boosting Classifier (avec ses paramètres par défaut)*. *Vous pouvez cependant accéder à l'ensemble des diagrammes de n-grams et tri-grammes pour chaque catégorie et pour d'autres algorithmes dans le dossier data_results.*

B - Analyse des résultats

Les diagrammes obtenus sont une information précieuse nous permettant d'identifier le type de formes généralement associées à des sur ou sous-représentations de notre classe de désaccord en fonction de l'algorithme utilisé. Nous allons analyser ces formes et tenter de comprendre les choix effectués par les modèles tout en cherchant à fournir d'éventuelles solutions pouvant contrer les erreurs de classification.

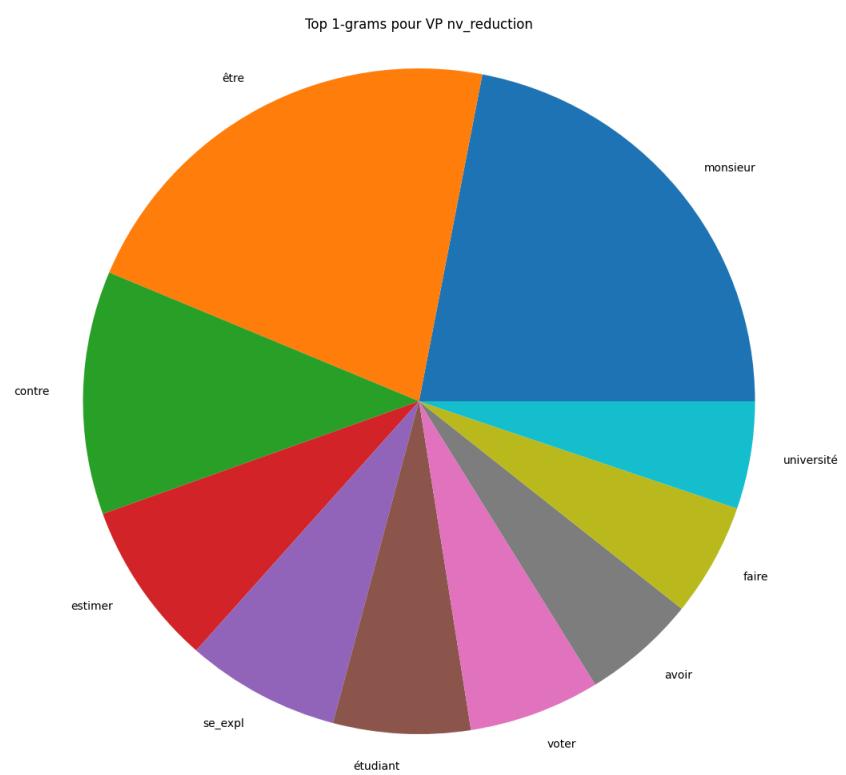
A - Naïve Bayes (NB)

1 - Vrais positifs : formes récurrentes

Les phrases composant la classe des vrais positifs correspondent aux phrases originellement annotées comme désaccord et étant reconnues comme telles par notre modèle.

Lorsque nous observons les monogrammes (donc les mots) les plus récurrents dans cette catégorie, nous obtenons les termes suivants :

Mono-grammes	Occurrences
Monsieur	80
Être	79
Contre	43
Estimer	29
Se_expl	27
Étudiant	24
Voter	23
Avoir	20
Faire	20
Université	19



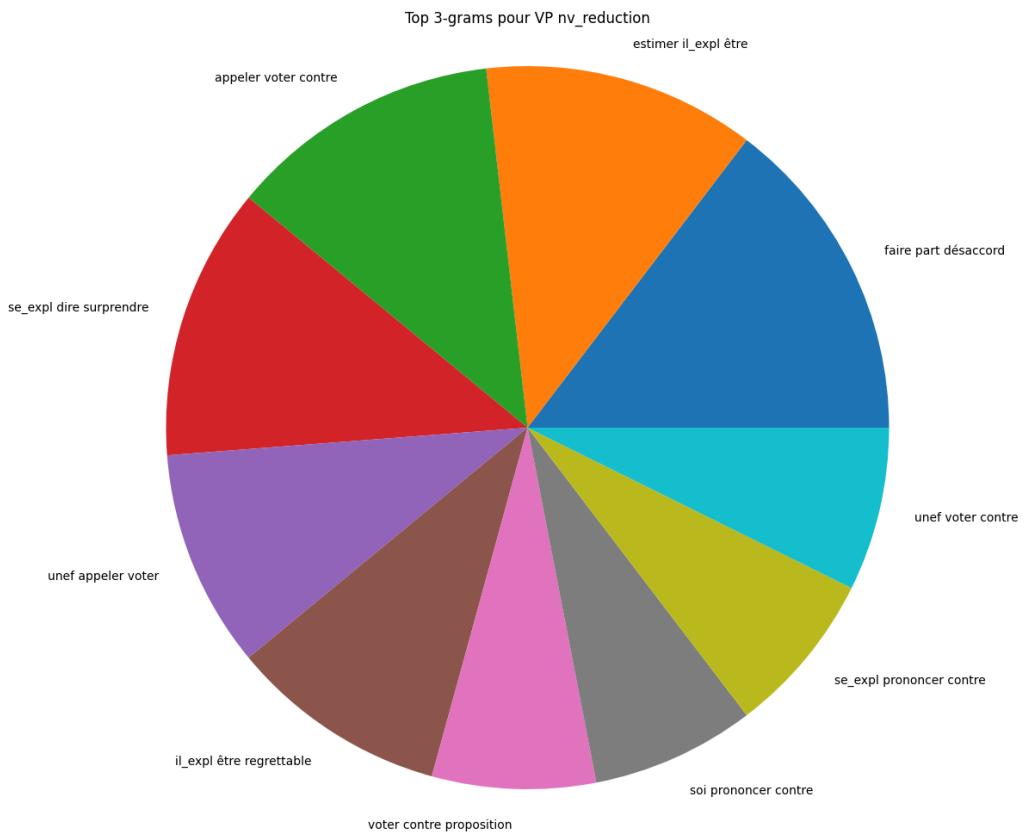
La répartition des monogrammes les plus fréquents dans notre catégorie VP nous indique premièrement quels termes apparaissent dans les phrases annotées D par les annotateurs du projet. Le mot le plus récurrent “Monsieur” semble faire référence à la manière dont le discours indirect est généralement représenté. Les échanges sont effectivement fréquemment exprimés de la manière suivante :

“ Monsieur LAKS trouver incongru le fait que parallèlement avoir être annoncer par Monsieur LEFEBVRE et Monsieur KREUTZER de+le discussion avec le INALCO .”

Il n'est donc pas surprenant de retrouver “**monsieur**” en haut du classement. Les autres termes les plus fréquents sont des verbes porteurs d'opinions tels que **estimer, s'expliquer** ou d'états tels que **être** et **avoir**. Nous retrouvons enfin **voter** et **contre**, montrant que les phrases exprimant l'opposition à certaines mesures sont fréquentes au sein des phrases annotées D par les membres du projet. La présence de mots sur représentés dans le corpus (dont **étudiant** et **université**) reste inévitable malgré leurs valeurs d'accord généralement neutre.

Lorsque nous observons les trigrammes (donc ensembles de trois mots) les plus récurrents dans cette catégorie, nous obtenons les termes suivants :

tri-grammes	Occurrences
Faire part désaccord	6
Estimer il_expl être	5
Appeler voter contre	5
Se_expl dire surprendre	5
Unef appeler voter	4
Il_expl être regrettable	4
Voter contre proposition	3
Soi prononcer contre	3
Se_expl prononcer contre	3
Unef voter contre	3



Les trigrammes les plus fréquents retrouvés semblent tous contenir l'un des monogrammes cités précédemment. Nous pouvons notamment observer la présence importante de la préposition *contre* précédée par des verbes d'opinion ou d'actions comme *voter* ou *prononcer*. *Expliquer* est le plus utilisé dans des formes d'expressions du désaccord. Sans tenir compte du contexte global de chaque phrase, les formes obtenues semblent être bien représentatives de la valeur de désaccord généralement admise par la convention de notation (discours indirects, adjetifs et noms).

2 - Faux positifs : formes récurrentes

Les phrases composant la classe des faux positifs correspondent aux phrases originellement annotées comme accord et étant reconnues comme désaccord.

Les résultats obtenus pour les monogrammes sont sensiblement les mêmes que pour la catégorie des vrais positifs montrant que leurs présences influencent grandement les choix du modèle.

Lorsque nous observons les trigrammes (donc les ensembles de 3 mots) les plus récurrents dans cette catégorie, nous obtenons les termes suivants :

tri-grammes	Occurrences
Cela pouvoir faire	1
Pouvoir faire craindre	1
Militant université 21	1
Université 21 préfet	1
Préfet avoir être	1
Saisir motif risque	1
Motif risque trouble	1
Risque trouble ordre	1
Trouble ordre public	1
Ordre public après	1

La répartition de ces formes semble indiquer que les formes récurrentes de la catégorie FP sont moins discriminantes que celles de la catégorie VP dans le choix de classification du modèle. Malgré tout, les mots contenus dans ces formes reviennent généralement plusieurs fois. De nombreux noms, tels que *trouble*, *ordre* ou *préfet* semblent revenir en majorité. Ces formes évoquent des situations de conflits liés aux politiques universitaires (évocation de la *préfecture* et des *militants*). Ces conflits menant à la prise de plusieurs décisions parfois difficiles à appliquer laissent à penser que les formes obtenues sont généralement contenues dans des phrases de débats. La cooccurrence de ces termes à des formes fortement représentées dans la classe des vrais positifs pourrait expliquer certains choix du modèle.

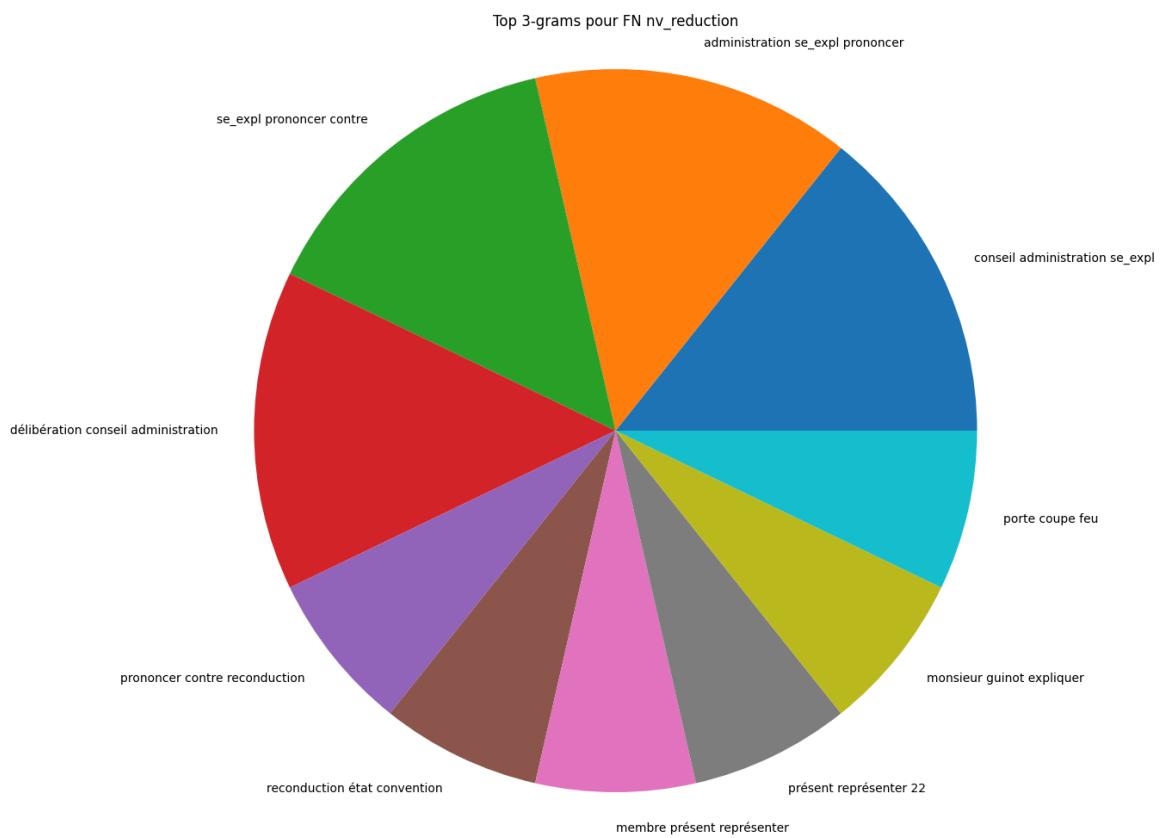
3 - Faux négatifs : formes récurrentes

Les phrases composant la classe des faux négatifs correspondent aux phrases originellement annotées comme désaccord et n'étant pas reconnues comme telles par notre modèle.

Lorsque nous observons les monogrammes (donc les mots) les plus récurrents dans cette catégorie, nous pouvons remarquer qu'ils sont encore une fois très similaires à ceux obtenus pour les deux autres catégories. Des mots comme "contre" sont eux absents, renforçant leur importance dans le choix des vrais positifs.

Les trigrammes les plus récurrents sont les suivants :

tri-grammes	Occurrences
<i>Conseil administration se_expl</i>	2
<i>Administration se_expl prononcer</i>	2
<i>Se_expl prononcer contre</i>	2
<i>Délibération conseil administration</i>	2
<i>Prononcer contre reduction</i>	1
<i>Reconduction état convention</i>	1
<i>Membre présent représenter</i>	1
<i>Présent représenter 22</i>	1
<i>Monsieur guinot expliquer</i>	1
<i>Porte coupe feu</i>	1



Les trigrammes obtenus ici peuvent paraître surprenants. En effet, nous retrouvons des termes auparavant majoritairement présents dans les phrases bien reconnues comme exprimant le désaccord par notre algorithme : *expliquer, se prononcer...*

Cependant, lorsque nous observons les phrases auxquelles appartiennent ces trigrammes (observable dans le fichier ***data_prediction_nv_reduction.csv***), nous constatons que de nombreuses expressions contenues dans ces dernières sont très souvent présentes dans des phrases annotées comme exprimant l'accord.

Dans la phrases annoté A mais étant en réalité D :

le conseil de administration se_expl prononcer contre le reconduction en le état de le convention avec IFG , à le unanimité de+le suffrage valablement exprimer par son membre présent ou représenter le 22 avril 2013 :

Si nous avons ***se expl prononcer contre***, nous avons également ***membre présent représenter*** qui est l'une des formes les plus courantes dans les phrases exprimant l'accord (6 occurrences). Le poids important de cette forme, en nombre d'occurrences, semble éclipser la valeur de désaccord exprimée par ***expl_prononcer contre*** (seulement à 3 nombre d'occurrences). Le même procédé semble se répéter pour d'autres formes pouvant ici paraître surprenantes. Pour ce qui est de la présence de *porte coupe feu* et *monsieur guinot*, nous faisons face à une simple répartition élevée de ces deux termes à cause des phrases utilisées pour l'entraînement du modèle.

Conclusion :

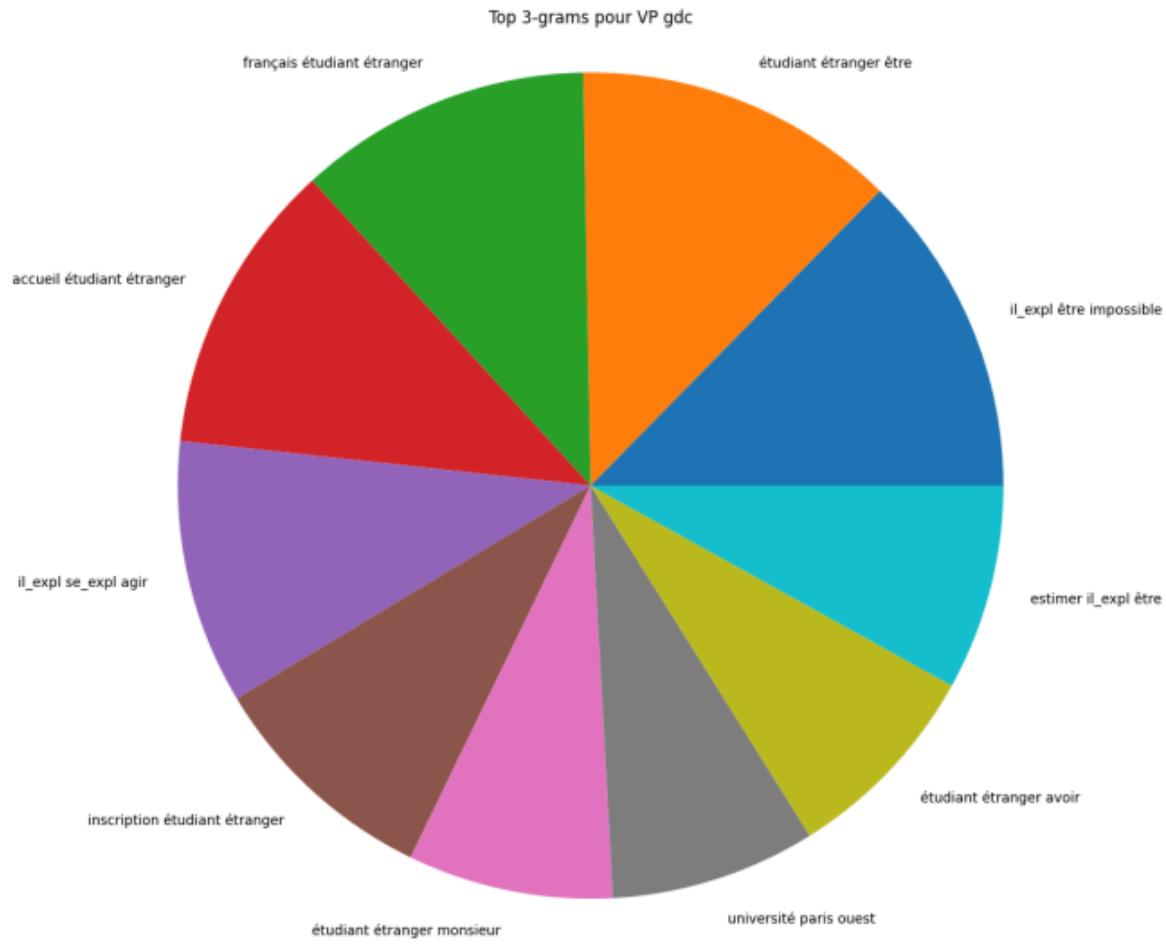
Les données obtenues nous ont permis de remarquer que l'algorithme Naïve Bayes effectue une classification basée sur des formes très "formelles" du désaccord de forme pronom verbe adverbe par exemple. Nous n'avons volontairement pas inclus les termes courants des vrais négatifs car correspondant majoritairement aux hypothèses des chercheurs des projets. La plupart de ces termes sont simplement les plus courants dans le corpus ou exprimant explicitement l'accord (élire, le conseil approuver...). Les formes obtenues pour nos autres catégories, si elles composent une grande partie de nos phrases exprimant le désaccord, ne suffisent pas à réellement identifier une opinion négative. De plus, l'hypothèse d'indépendance de l'algorithme, et donc son fonctionnement basé sur les distributions globales des termes, tend à passer à côté de phrases plus longues et contenant également des formes fortement présentes dans les phrases exprimant l'accord. Le poids élevé de ces formes pourrait être contré en tentant de donner nous même plus de poids aux formes porteuses de désaccord au sein d'une phrase. Cependant, cela demanderait l'obtention d'une liste plus longue et complexe de termes ciblés et les formes dont les unités sont dispersées dans la phrase seraient très compliquées à repérer sans l'aide de plusieurs expressions régulières. Un nombre plus conséquent de phrases annotées D permettrait également d'améliorer nos résultats comme illustré précédemment dans ce rapport.

B - Gradient Boosting Classifier (GBC)

Tout comme pour Naïve Bayes, nous allons maintenant chercher à analyser les n-grams les plus communs dans chacune de nos catégories.

1 - Vrais positifs : formes récurrentes

La classe des vrais positifs fait référence aux phrases annotées comme désaccord qui ont été classées comme désaccord par notre modèle.



La première observation pouvant être faite lorsque l'on compare ces résultats à ceux de Naïve Bayes est qu'il semble être possible d'identifier une certaine thématique récurrente au sein des phrases bien reconnues comme D.

tri-grammes	Thématique remarquable
Français étudiant étranger	Étudiant étranger
Étudiant étranger être	Étudiant étranger
Accueil étudiant étranger	Étudiant étranger
Il_expl être impossible	Expression nuance
Estimer il_expl être	Expression nuance
Étudiant étranger avoir	Étudiant étranger
Université paris ouest	
Étudiant étranger monsieur	Étudiant étranger
Inscription étudiant étranger	Étudiant étranger
Il_expl se_expl agir	Explication

La plupart semblent en effet concerner l'accueil des étudiants étrangers à l'université. Dans le cas des diagrammes de trigrammes de Naïve Bayes, il était impossible d'identifier une thématique. Nous avions surtout affaire à des termes très formels. Il semblerait ainsi que le Gradient Boosting Classifieur puisse analyser une phrase de manière plus complexe en tenant en compte son contexte linguistique plus généralement. Le nombre d'occurrences de termes n'est pas la seule feature utilisé, il est ici possible que les co-occurrences aient été utilisées par l'algorithme pour arriver à cette classification. Nous avons également utilisé notre programme sur le corpus complet et en utilisant Naïve Bayes (sans réduction de classes) afin de vérifier notre hypothèse et que nos résultats ne soient pas simplement basés sur une prédominance général du thème des étudiants étrangers. La plupart des termes obtenus sont là encore des expressions très formelles du désaccord et aucune thématique ne peut être identifiée.

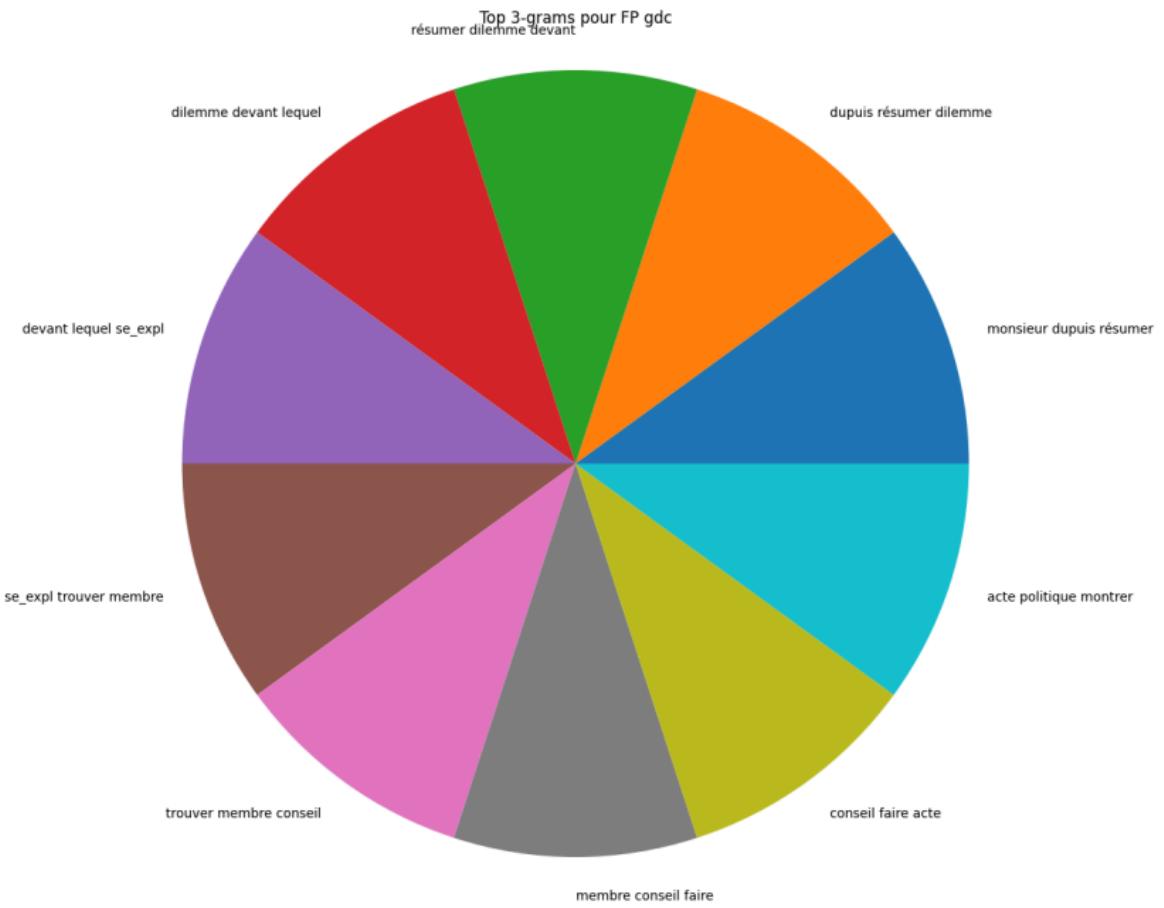
Les autres trigrammes obtenus relèvent eux de formes nous étant familières et contenant notamment des verbes permettant de retranscrire des discours indirects.

2 - Faux positifs : formes récurrentes

Les faux positifs font référence aux phrases annotées comme accord qui ont été classées comme désaccord par notre modèle. On peut relever facilement un thematique des trigrammes appartenant à cette classe :

tri-grammes	Thématique remarquable
Dilemme devant lequel	Dilemme
Résumer dilemme devant	Dilemme
Dupuis résumer dilemme	Dilemme
Monsieur Dupuis résumer	Verbe dynamique lié au vocabulaire des conseils d'administration
Conseil faire acte	Verbe dynamique lié au vocabulaire des conseils d'administration
Acte politique montrer	Verbe dynamique lié au vocabulaire des conseils d'administration
Membre conseil faire	Verbe dynamique lié au vocabulaire des conseils d'administration
Trouver membre conseil	Verbe dynamique lié au vocabulaire des conseils d'administration
Se_expl trouver membre	Verbe dynamique lié au vocabulaire des conseils d'administration
Devant lequel se_expl	Verbe dynamique lié au vocabulaire des conseils d'administration

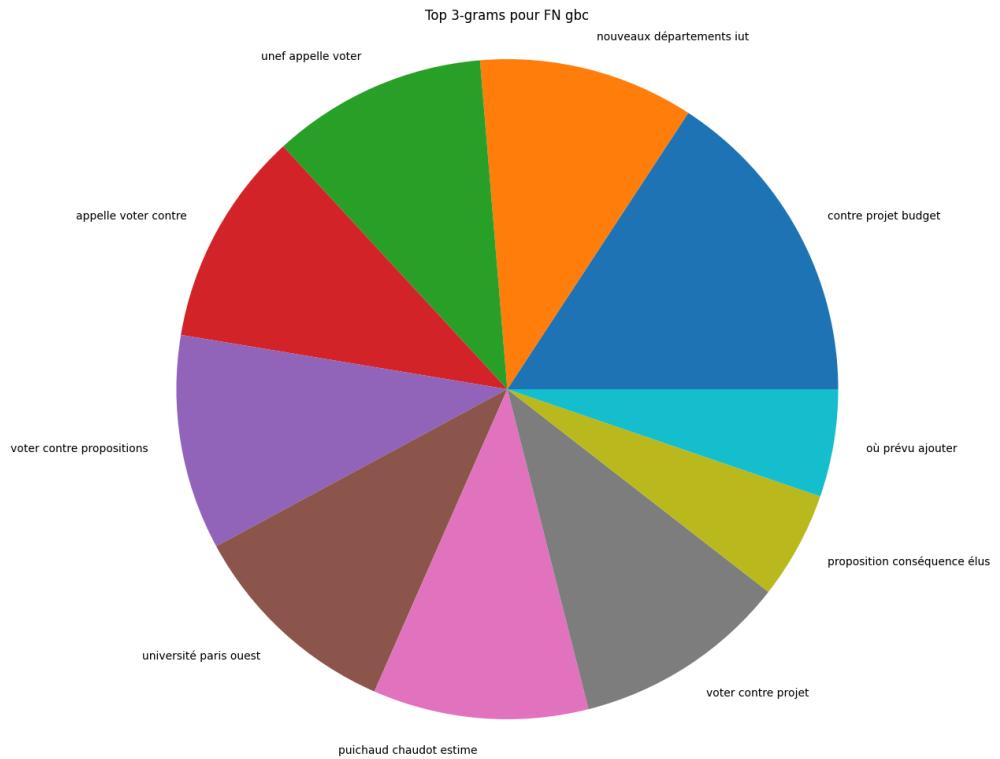
Nous remarquons donc que le nom *dilemme*, dans son acception liée à une situation problématique peut-être la cause que ce trigramme soit récurrent dans la classification en désaccord. Les autres trigrammes ne sont pas nécessairement liés à une nuance de désaccord, peut être que la cause de sa classification leur fréquence d'apparition dans le corpus. Ce sont des termes plutôt formels, liés aux procédures développées au sein des conseils d'administration, et moins discriminants que ceux classées comme VP. Étant donné que le vocabulaire des conseils d'administration est assez général, on ne peut pas savoir clairement les raisons qui se cachent derrière les décisions du modèle.



Faux négatifs :

Cette classe fait référence aux phrases annotées D, classées comme accords. On remarque ici qu'on peut comprendre les classifications en termes de thématique, même s'il y a des propositions utilisées, classées comme porteuses de désaccord selon le Naïve Bayes, comme la préposition “contre”, par exemple.

Voyons les tri-grammes proposés :



tri-grammes	Thématique remarquable
Unef appelle voter	Voter
Appelle voter contre	Utilisation préposition contre
Voter contre proposition	Utilisation préposition contre
Université paris ouest	Nom propre
Puichaud chaudot estime	Verbe dynamique lié au vocabulaire des conseils d'administration
Voter contre projet	Utilisation préposition contre
Proposition conséquence élus	Verbe dynamique lié au vocabulaire des conseils d'administration
Contre projet budget	Utilisation préposition contre
Où prévu ajouter	Verbe dynamique lié au vocabulaire des conseils d'administration
Nouveaux départements iut	Vocabulaire des conseils d'administration

Il est intéressant de voir le cas de la préposition “contre”, classée comme accord dans le Gradient Boosting Classifier. Lors de l'utilisation du Naïve Bayes, on a vu que cette proposition était classée comme vrai positif et que la plupart des tri-grammes, après la réduction, la possédaient. Cependant, ces trigrammes ont été classées comme accord. Si on fait une comparaison plus détaillée avec les VP du Naïve Bayes après réduction. On remarque que les trigrammes les plus fréquents pour Naïves Bayes, se trouvent parmi les Faux négatifs du Gradient Boosting. On pourrait se demander à partir de cela si c'est pour l'hypothèse d'indépendance conditionnelle du le Naïves Bayes que ces trigrammes ont été classés de forme correcte. Le choix du GBC ne reste pas trop clair pour nous, ce sera un point qui pourrait être abordé avec plus d'exemples, en tenant compte du poids des tokens annotés comme D, tels que la préposition **contre**, dans notre cas.

VP Naïve Bayes	FP Gradient Boosting Classifier
Faire part désaccord	Unef appeler voter
Estimer il_expl être	Appelle voter contre
Appeler voter contre	Voter contre proposition
Se_expl dire surprendre	Université paris ouest
Unef appeler voter	Puichaud chaudot estime
Il_expl être regrettable	Voter contre projet
Voter contre proposition	Proposition conséquence élus
Soi prononcer contre	Contre projet budget
Se_expl prononcer contre	Où prévu ajouter
Unef voter contre	Nouveaux départements iut

Conclusion

On a vu tout au long de l'utilisation du Naïve Bayes et du Gradient Boosting Classifier pour l'identification des n-grammes les plus fréquents, qu'il y a des grosses différences entre les deux. Les mots grammaticaux, comme la préposition **contre**, a été un point clé dans les vrais positifs du NB, tandis que ce sont les mots lexicaux qui semblent avoir un impact plus important dans le choix du GBC. Aussi dans les FP des deux algorithmes, on remarque que les mots de conflit comme *ordre*, *trouble* et *risque* sont récurrents dans le NB, tandis que le GBC a classé des trigrammes qui faisaient référence plutôt au vocabulaire des procédures administratives. Finalement, le GBC a classé comme FN des tri-grammes contenant la préposition **contre**, ce qui nous amène à nous demander si pour la reconnaissance des trigrammes le NB est plus exact, même si la classification selon thématique est plus intéressante, pour le GBC.

VII - Conclusion

Ci-dessous, vous trouverez un tableau comparatif des meilleurs résultats pour chaque solution.

Pour rappel voici la liste des solutions que nous avons apporté au problème apporté par le corpus:

- Essai 1 : Corpus originel - Naive Bayes
- Essai 2 : Réduction de la classe majoritaire - Naive Bayes
- Essai 3 : Modification des poids - Corpus originel - Naive Bayes (30)
- Essai 4 : Modification des hyperparamètres - Corpus originel - Naive Bayes
- Essai 5 : Corpus originel - Gradient Boosting Classifier
- Essai 6 : Modification des hyperparameters - Corpus originel - Gradient Boosting Classifier
- Essai 7: Corpus enrichi - Naive Bayes
- Essai 8: Corpus enrichi - Gradient Boosting Classifier

	1	2	3	4	5	6	7	8
macro-précision	0.50	0.88	0.65	0.87	0.85	0.83	0.93	0.99
macro rappel	0.50	0.88	0.68	0.58	0.72	0.74	0.50	0.87
Macro-AVG	0.50	0.88	0.66	0.62	0.77	0.78	0.50	0.92

Comme dans les précédentes conclusions, nous avons choisi de prendre en considération en premier lieu la macro average car elle traite toutes les classes de manière égale même les classes minoritaires. Notre corpus étant très inégal, cette mesure montre le mieux la capacité à reconnaître le désaccord qui est la classe minoritaire.

La meilleure macro-f-mesure est celle du dernier test. Dans cet essai, le corpus est enrichi pour donner plus de matière à la machine et il est entraîné avec le Gradient Boosting Classifier.

La seconde meilleure option est celle où l'on opère une réduction de la classe majoritaire (phrases exprimant l'accord) dans le corpus et puis on l'entraîne avec Naive Bayes. Cette solution rééquilibre les classes artificiellement.

Ces deux options sont assez artificielles car on n'utilise pas le corpus donné: dans la solution 8 le corpus est enrichi en annotation de désaccord et dans le second on supprime une partie des phrases exprimant l'accord afin d'avoir un corpus plus équilibré.

On voit donc que les tests donnent des résultats peu concluants avec le corpus originel. Ce qui montre que les modèles sont efficaces mais que le problème vient du corpus qui ne comporte pas suffisamment d'annotations.

Difficultés rencontrées

Lors de ce projet, nous avons rencontré plusieurs difficultés. Au début du projet, nous avons eu du mal à comprendre le but de ce corpus et de notre devoir. Où étaient les balises d'annotations ? Les annotations étaient-elles sur les tours de paroles, les phrases ou les tokens ? Devions-nous annoter ou compléter l'annotation ?

Une seconde difficulté que nous avons rencontrée est que nous n'avions que peu d'informations sur le corpus. Le manque d'informations nous a fait prendre du retard sur le projet. La documentation n'étant pas claire, nous ne savions pas si l'annotation était complète. Nous nous posions aussi des questions sur la motivation des créateurs à annoter au niveau du token et non de la phrase ou du tour de parole.

Finalement, très peu de tokens portent l'information du désaccord. Il s'est également avéré qu'annoter le désaccord sur les tokens n'est pas la méthode idéale car elle ne permet pas de rendre compte du contexte des tokens qui pourrait indiquer un désaccord. Par exemple, les verbes de discours indirect qui sont annotés ne donnent aucune information sur ce que pense le locuteur sans les éléments qui l'entourent. Les verbes "avouer", "penser", "sembler" ou "estimer" sans leur dépendants syntaxiques expriment seulement une opinion mais ne permettent pas de savoir si l'opinion exprimée est de celui de l'accord ou celui du désaccord.

Nous aurions aimé avoir plus de temps afin de compléter nos essais et tester des techniques différentes.

Pour la suite de ce projet, il serait intéressant d'essayer les pistes suivantes:

- Pour les modèles :
- Au niveau de la vectorisation des phrases ou des documents, il serait bon de le faire avec un autre type de vectorisation que TF-IDF (utilisé ici). L'utilisation des techniques de word embedding comme **Tok2vec** ou **doc2vec** pourrait améliorer les résultats de nos tests.
- Modifier les features pourrait être une autre piste à explorer. Par exemple, prendre en compte la taille des phrases.
- Avec des algorithmes et des vecteurs différents il se peut que d'autres hyperparamètres se distinguent.

Perspectives de recherche supplémentaire :

Comme vu précédemment avec les recherches de formes ou de "thèmes" à l'origine de conflit, quand ce corpus sera complété, il permettra d'ouvrir plusieurs pistes de recherches dans plusieurs domaines.

En morpho-syntaxe, suivre la piste des différentes façons d'exprimer le désaccord serait un premier pas vers une analyse linguistique de ce corpus. Une exploration plus poussée de ces formes d'expression du désaccord permettrait de meilleures performances de notre modèle et éventuellement l'exploitation de ces résultats.

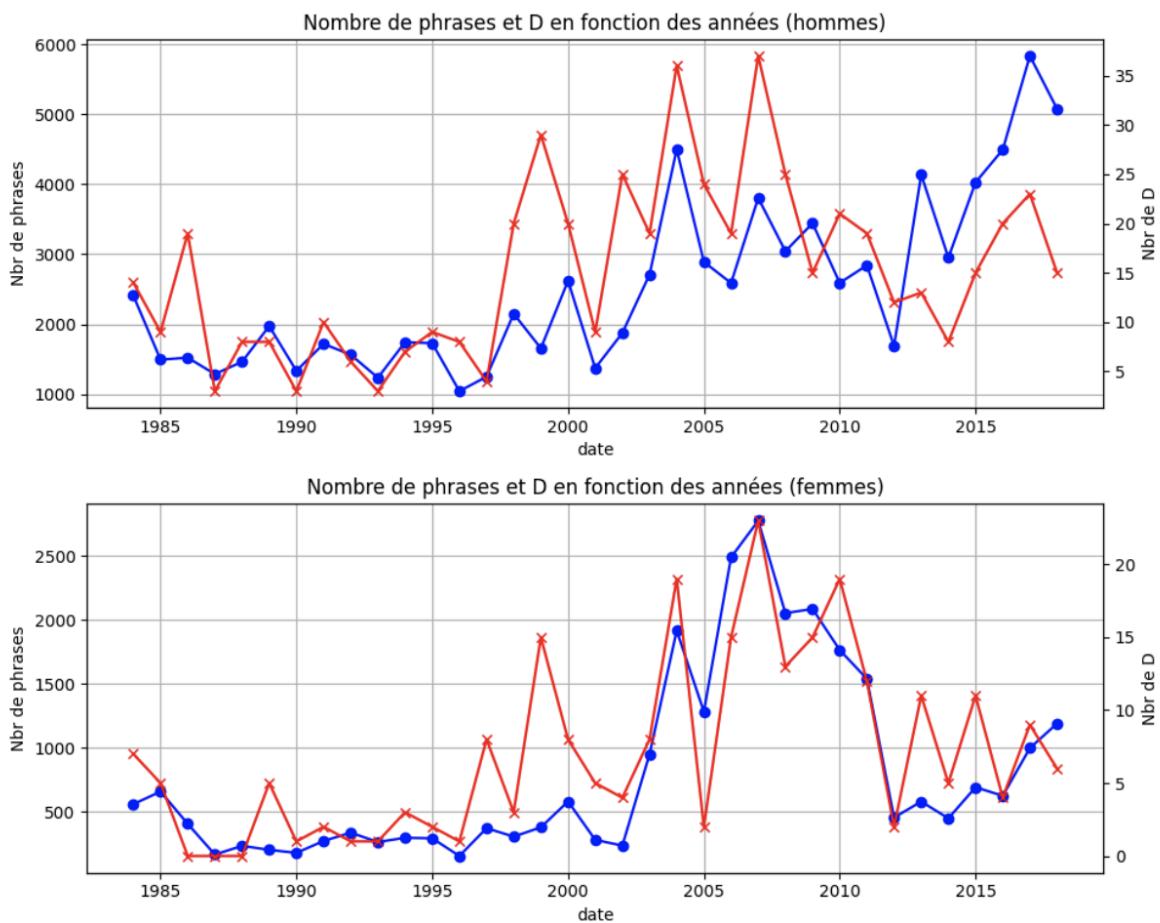
Ensuite, continuer l'exploration des thèmes qui seraient potentiellement propices aux désaccords. Pour cela, il faudrait partir des résultats trouvés en partie IV. En repérant les thèmes récurrents dans les phrases exprimant le désaccord, on peut isoler les parties des contre-rendus (un tour de parole au-dessus et un tour de parole au-dessous) qui sont en rapport avec ces sujets et voir également les déclencheurs de conflits.

L'étude sociolinguistique serait l'un d'eux et permettrait de revenir aux prémisses de création de ce corpus. Dans les sources de questionnements à l'origine de la création de ce corpus, les chercheurs posent la question du lien entre le genre et l'expression de conflit dans les tours de paroles. En effet, selon une étude les tours de paroles des "locutrices" comporteraient plus "de questions et de verbes de DI [Discours Indirect] indiquant un dissensus" (Lethier et al. 2023).

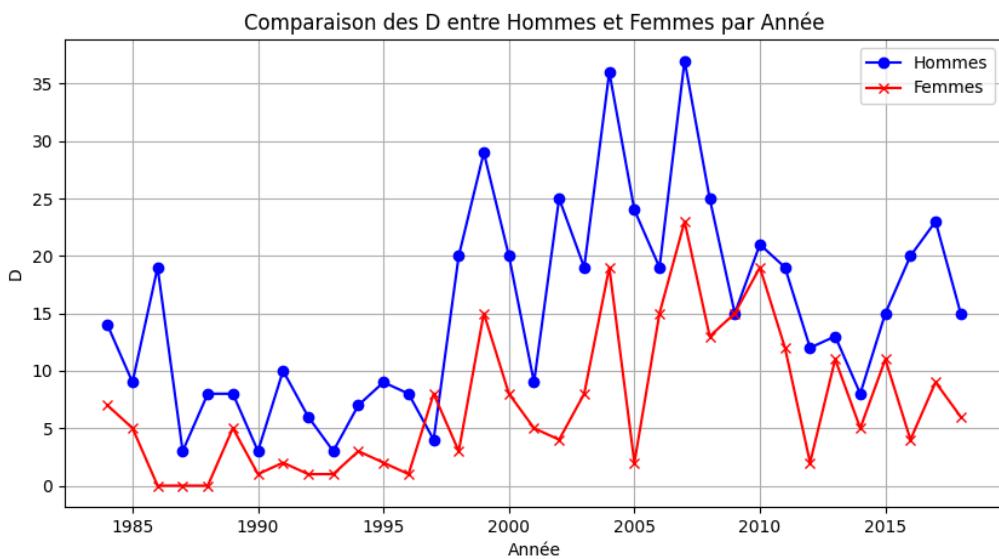
Dans la convention d'annotation, il est écrit : "Les femmes seraient ainsi plus « opposantes ». ". Dans un souhait de rendre compte d'un des objectifs généraux du corpus, nous avons créé les graphiques ci-dessous (à l'aide du script stats_sexe.py). Ces deux graphiques représentent pour les hommes puis pour les femmes, le nombre de phrases exprimant le désaccord en rouge et le nombre de phrases en bleu, le tout pour chaque année.

Désaccord = rouge

Phrase = bleu



On peut voir que les hommes parlent beaucoup plus que les femmes, les deux graphiques n'étant même pas sur la même échelle de valeur. Selon les années le nombre de phrases en désaccord augmente et pour la plupart des années le nombre de phrases augmente également.



Pour ce qui est du nombre de phrases exprimant le désaccord, nous pouvons voir avec le graphique ci-dessus, que les augmentations et diminutions des phrases en désaccords se suivent entre les hommes et les femmes selon les années.

Il serait intéressant d'inclure une approche statistique de ces données afin de voir le pourcentage de phrases en désaccord sur le nombre de phrases pour chaque genre. Nous pourrions ainsi véritablement offrir une comparaison de l'expression du désaccord entre les genres. Étudier également la forme que prend ce désaccord comme vu précédemment offrirait une nouvelle perspective de comparaison plus linguistique. Pour calculer ces statistiques, il serait également pertinent de comparer le pourcentage de désaccord exprimé en rapport au nombre de tokens dit par chaque genre.

Ici, la représentation est globale sur la totalité des assemblés, il serait intéressant de voir ces mêmes données à plusieurs niveaux. Par exemple, année par année ou décennie par décennie et voir l'évolution du désaccord et de la présence des femmes dans les assemblées. Trouver le nombre de femmes et d'hommes selon les années et assemblées, nous permettrait également d'analyser les différents statuts des énonciateurs femmes et hommes en rapport avec le nombre d'expressions du désaccord.

En revanche, pour accéder à ces analyses, le corpus n'est pas suffisant. Il faudrait reprendre l'annotation et enrichir le corpus afin de le rendre utilisable.