PROJET

Approches Probabilistes pour le TAL

Statistiques Descriptives

Charlotte Dugrain

Alexandra Guérin

M1-LI - 2018

Sommaire

1. Extraction des Binomiales

Nous avons décidé de traiter ce problème en utilisant le paradigme de la programmation orientée objet avec python. Nous avons donc considéré que chaque binôme allait être représenté par un objet nommé “Binôme” qui serait lui même composé d’objets nommés “Eléments”.

1.1. L’objet Element

L’objet **Element** contient les informations que l’on peut extraire du fichier conll. Nous avons choisi de garder seulement les plus pertinentes pour la tâche qui nous est demandée à savoir :

- L’indice linéaire du mot : (int)

  La numérotation des mots dans le fichier conll commence à 1, nous retirons donc 1 pour que le premier mot ait l’index 0 et que cela corresponde aux numérotations informatiques commençant à 0.
- La forme du mot dont il est question : (str)

  Nous avons choisi de prendre la forme fléchie et non le lemme car de toute façon les informations de la flexion étant enregistrées dans le reste de l’objet, nous ne pouvions pas regrouper les éléments de mêmes formes fléchies dans le même binôme. Ces informations de flexion peuvent constituer des éléments du vecteur de features.
- La catégorie : (str)

  Elle est pertinente dans le sens ou pour former un binôme, les deux éléments doivent être de même catégorie.
- L’index linéaire du gouverneur : (int)

  Pour les mêmes raisons que l’index linéaire du mot, nous soustrairons 1 à chaque index linéaire de gouverneur. Cette informations pourra nous servir à l’extraction des binômes ayant des dépendances à longue distance.

Si applicable :

- Le genre : (str)

  Cela nous servira pour les features du classifieur, en effet un des facteurs supposés de l’ordre des binômes est le genre, on a plus souvent masculin suivi de féminin
- Le nombre : (str)

  Tout comme le genre cela nous servira pour les features du classifieur, un des facteurs supposés de l’ordre des binômes étant le nombre, on a plus souvent singulier suivi de pluriel

Pour la forme du mot, ayant remarqué que dans le corpus OS beaucoup de forme des mots était accolée à la ponctuation par exemple ‘carburant?’, nous avons fait une fonction **has_endpunctuation()** et **remove_endpunctuation()**, qui dans le cas où la forme du mot a de la ponctuation uniquement au niveau du dernier caractère on supprime ce caractère et enregistrer la forme du mot sans cette ponctuation. Pour les formes avec de la ponctuation à l’intérieur du mot ou à différents endroits on s’en occupe dans la phase de nettoyage

1.2. L’objet Binôme

L’objet **Binome** contiendra les informations relatives à un binôme extrait du corpus. Il contient :

- Le premier élément : (Element)
- Le deuxième élément : (Element)

L’ordre par lequel le binôme est stocké, celui qui détermine quelle partie du binôme sera stockée dans “premier” ou “deuxième” élément est l’ordre alphabétique.

- La catégorie des deux éléments : (str)

  Celle-ci sera toujours identique pour les deux éléments puisqu’elle est une condition de leur extraction en tant que binôme.

1.3. Les fonctions d’extraction

Nous avons ensuite élaboré 4 fonctions principales, **extractLineaire**(), **extractDependances**(), **assignClass**() et **saveBinoms**() qui nous permettent, en les utilisant à la suite, d’obtenir à partir d’un (ou plusieurs) fichier(s) conll, un fichier pkl (contenant un dictionnaire) et un fichier texte contenant tous deux les binômes pertinents ainsi que leurs comptes et une classe qui leur est attribuée en fonction de ces comptes pour ensuite pouvoir effectuer des statistiques et tirer les conclusions qui nous intéressent dans le reste du projet.

Les fonctions **extractLineaire() et extractDependances()** ont le même but mais n’ont pas la même façon d’y arriver et ne produisent pas non plus le même résultat. Leur but est d’extraire dans un dictionnaire les binômes se trouvant dans un ou plusieurs fichier(s) conll et de les enregistrer dans un dictionnaire.

- ***extractLineaire()*** n’utilise pas les informations de dépendance contenues dans le fichier conll, elle n’utilise que la forme du mot ainsi que les index dans la phrase de celui-ci. Cette fonction parcourt le fichier phrases par phrases, et dès qu’il rencontre

une conjonction de coordination “et” ou “ou”, regarde simplement le mot précédent et le mot suivant (à l’exception des binômes indirects contenant un déterminant ou une préposition, pour lesquels la fenêtre s’élargira à deux mots avant et deux mots après la conjonction.) pour statuer s’il s’agit d’un binôme ou non.

- ***extractDependances()*** effectuera le même travail mais en utilisant les informations de dépendances contenues dans le fichier conll. Il trouvera une conjonction, déterminera son gouverneur et son (ses) gouverné(s) au sein de la phrase et statuera s’ils forment un binôme ou non. Cela signifie que les binômes peuvent être à une distance qui n’est pas restreinte dans la phrase.

Une fois un potentiel binôme détecté, il faut pouvoir statuer s’il est pertinent de le garder pour notre étude. Nous avons pris le parti d’effectuer cette étape du “nettoyage” au cours de l’extraction et non après car cela nous évite de stocker des informations sur des binômes qui ne sont pas pertinents. Mais cela ralentit un peu le processus d’extraction. Le nettoyage est effectué par la fonction ***isBinome()*** qui est commune aux deux fonctions d’extraction. Elle vérifie :

- si les deux éléments sont effectivement de la même catégorie grammaticale (et pas PONCT)
- si ce ne sont pas des nombres
- si les caractères sont bien latins
- s’ils ne contiennent pas de ponctuation (permet par exemple de supprimer les URL)

Si le nettoyage détermine que ce binôme est pertinent alors il sera stocké dans un dictionnaire qui prendra la forme suivante :

clé (binôme) : (mot1, mot2)
val (liste) : [a, b, c, d, e, f, g, h]
a = nombre de binômes **directs** reliés par **et** dans l'ordre **alpha**
b = nombre de binômes **directs** reliés par **ou** dans l'ordre **alpha**
c = nombre de binômes **directs** reliés par **et** dans l'ordre **inverse**
d = nombre de binômes **directs** reliés par **ou** dans l'ordre **inverse**
e = nombre de binômes **indirects** reliés par **et** dans l'ordre **alpha**
f = nombre de binômes **indirects** reliés par **ou** dans l'ordre **alpha**
g = nombre de binômes **indirects** reliés par **et** dans l'ordre **inverse**
h = nombre de binômes **indirects** reliés par **ou** dans l'ordre **inverse**

Une fois les comptes terminés, il sera nécessaire d’attribuer une classe à chaque binôme en fonction de s’il est figé ou non grâce à la fonction ***assignClass()***. Pour cela nous avons décidé de déterminer 10 classes (numérotées de 1 à 10) qui classent les binômes de la sorte :

Classe	**chances de trouver le binôme dans l'ordre alphabétique**	**signification**
10	100 à 91%	binôme figé dans l'ordre alphabétique
9	90 à 81%	
8	80 à 71%	
7	70 à 61%	
6	60% à 51%	autant de chances dans chaque ordre
5	50% à 41%	
4	40 à 31%	
3	30 à 21%	
2	20 à 11%	binôme figé dans l'ordre inverse
1	10 à 0%	

Nous avons décidé que les binômes des classes 1, 2, 9 et 10 seraient ceux que l'on considérerait comme "figés". Nous avions réfléchi à les répartir en deux classes seulement, mais il nous semblait plus pertinent de garder cette information pour pouvoir effectuer des recherches et tirer des conclusions ensuite. (il nous suffirait de traiter ces classes comme étant deux classes séparées si nous voulons ensuite).

Une fois que la fonction d'extraction aura parcouru et l'intégralité des données qui lui ont été fournies, et que les classes auront été assignées, elle enregistrera les binômes qui ont été extraits de deux manières différentes :

- dans un fichier pkl, directement utilisable comme un dictionnaire par un autre programme
- dans un fichier txt, dans lequel il sera plus facile de regarder manuellement pour se rendre compte de son effet grâce à la fonction ***saveBinomes()***.

2. Description des données extraites

Nous allons, dans un premier temps, avant d'examiner et utiliser les caractéristiques pour chaque binômes pour entraîner un classifieur, essayer d'avoir une vision d'ensemble des données que nous avons pu extraire des 2 corpus de données étudiées. Nous avons donc 2 corpus : Wikipédia et Open Subtitles desquels nous avons extrait les binômes dans un premier temps de façon linéaire puis en utilisant les dépendances. Ce qui nous donne 4 jeux de données à étudier :

- binômes de **wikipédia** extraits **linéairement**
- binômes de **wikipédia** extraits par **dépendances**
- binômes de **OpenSubtitles** extraits **linéairement**
- binômes de **OpenSubtitles** extraits par **dépendance**

Cette vision d'ensemble nous permettra par la suite de donner de la valeur ou de relativiser justement certains résultats obtenus lors de l'étude plus poussée des binômes.

2.1. Description globale

Voici un tableau dans lequel on peut recenser les comptes effectués sur chacun des corpus de binômes extraits, en prenant en compte la totalité des données. Il permet de se rendre compte des grandes différences mais quelques graphiques plus bas seront plus représentatifs pour les données qui nous intéressent.

	Wikipédia Linéaire	**Wikipédia Dépendances**	**OpenSubtitles Linéaire**	**OpenSubtitles Dépendances**
binômes	2.544.106	3.032.037	37.657	48.041
total	6.334.057	10.034.731	52.109	84.509
directs	4.347.428	2.054.659	22.457	15.749
alpha	3.194.212	4.025.308	26.163	36.364
et	5.619.022	9.006.259	43.776	75.499
figés	2.407.062	2.786.061	36.893	46.218
figés et alpha	1.151.950	1.410.079	17.960	23.225

Les premières conclusions assez triviales que nous pouvons tirer de l'observation de ces données sont les suivantes :

- Le corpus Wikipédia fournit près de 60 fois plus de binômes que le corpus d'OpenSubtitles. Il constitue donc un échantillon beaucoup plus représentatif de la population totale sur laquelle nous voulons généraliser. Il sera plus intéressant d'entraîner un classifieur sur les binômes issus de wikipédia que sur ceux issus d'OpenSubtitles.

- L'extraction effectuée grâce aux informations de dépendances dans le fichier conll fournit plus de binômes que celle effectuée par l'algorithme d'extraction linéaire. Cela était prédictible car elle borne moins les distances entre les deux parties du binôme et couvre donc plus de données.

- Cependant on peut voir que le nombre de binômes directs extrait par l'algorithme d'extraction par dépendances est largement plus faible que celui extrait par l'algorithme linéaire. On peut donc facilement supposer que les dépendances du fichier conll ne sont pas toujours reportées correctement. Cela nous pousserait donc à douter de la fiabilité de ces dernières.

2.2. Nécessité d'un second nettoyage

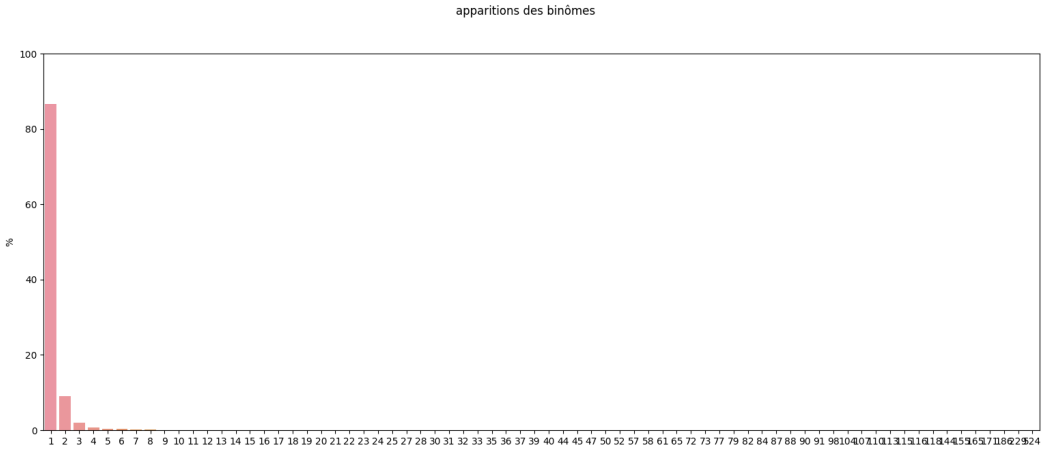
En inspectant le nombre de binômes extraits ainsi que leurs occurrences nous avons pu remarquer également qu'un très grand nombre de binômes n'apparaissaient qu'une seule fois et qu'un très petit nombre de binômes apparaissent beaucoup de fois. Par exemple, voici les données pour le corpus d'OpenSubtitles en extraction linéaire:

1: 32646, 2: 3389, 3: 732, 4: 301, 5: 141, 6: 106, 7: 71, 8: 49, 9: 30, 10: 23, 11: 18, 12: 14, 13: 12, 14: 12,	15: 5, 16: 8, 17: 7, 18: 5, 19: 5, 20: 4, 21: 5, 22: 2, 23: 3, 24: 3, 25: 5, 27: 3, 28: 3, 30: 2,	31: 2, 32: 2, 33: 1, 35: 3, 36: 1, 37: 1, 39: 1, 40: 2, 44: 1, 45: 2, 47: 3, 50: 1, 52: 1, 57: 1,	58: 2, 61: 1, 65: 1, 72: 1, 73: 2, 77: 1, 79: 1, 82: 1, 84: 1, 87: 1, 88: 1, 90: 1, 91: 1, 98: 1,	104: 1, 107: 1, 110: 1, 113: 1, 115: 1, 116: 1, 118: 1, 144: 1, 155: 2, 165: 1, 171: 1, 186: 1, 229: 1, 524: 1

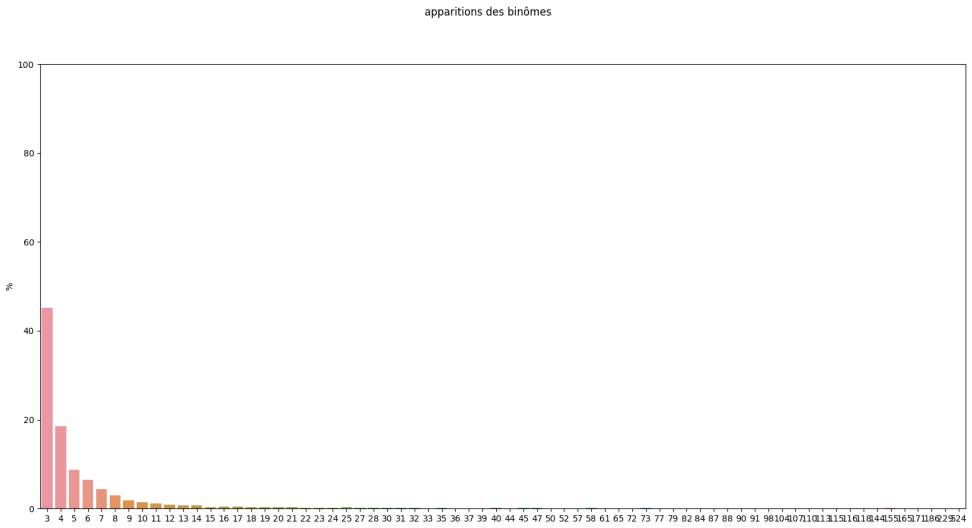
Nous avons donc décidé d'effectuer un second nettoyage une fois les binômes extraits, se basant sur leur fréquence d'apparition car un binôme n'apparaissant qu'une fois allait être considéré comme étant figé, or, le nombre d'occurrences n'est pas assez représentatif pour en tirer une telle conclusion. Nous avons donc fixé le minimum d'occurrences dans le corpus pour qu'**un binôme soit représentatif à 3.**

De même nous avons décidé d'évincer les quelques binômes dont les occurrences sont monstrueusement supérieures à celles des autres binômes. Nous considérerons que ceux-ci sont le fruit de "bruit" dans les données, typiquement des références ou titres répétitifs dans chaque corpus, liées à la source des données et non à la langue en elle même. Après quelques essais, il nous est apparu que **retirer seulement les 0,01% de l'extrême** nous permettait d'avoir un échantillon de taille raisonnable.

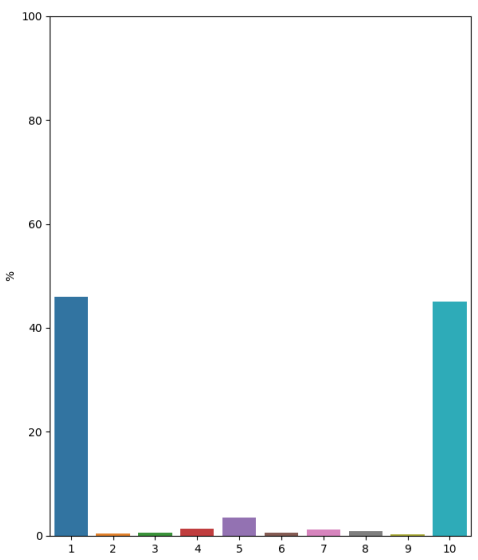
Ces deux bornes retirent les données en rouges sur les comptes d'occurrences et transforment la distribution des binômes avant nettoyage :

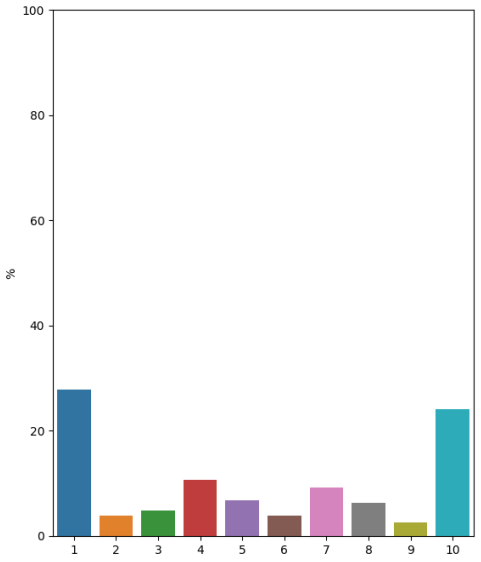


à la suivante après nettoyage :



On peut également voir l'effet direct sur la distribution des binômes par classes sur l'extraction wikipedia par dépendances :





avant nettoyage *après nettoyage*

Le nombre de binômes très figés (classes 1 et 10) est anormalement grand puisqu'il contient tous les binômes n'apparaissant qu'une fois ainsi que tous les binômes apparaissant des nombres incohérents de fois.

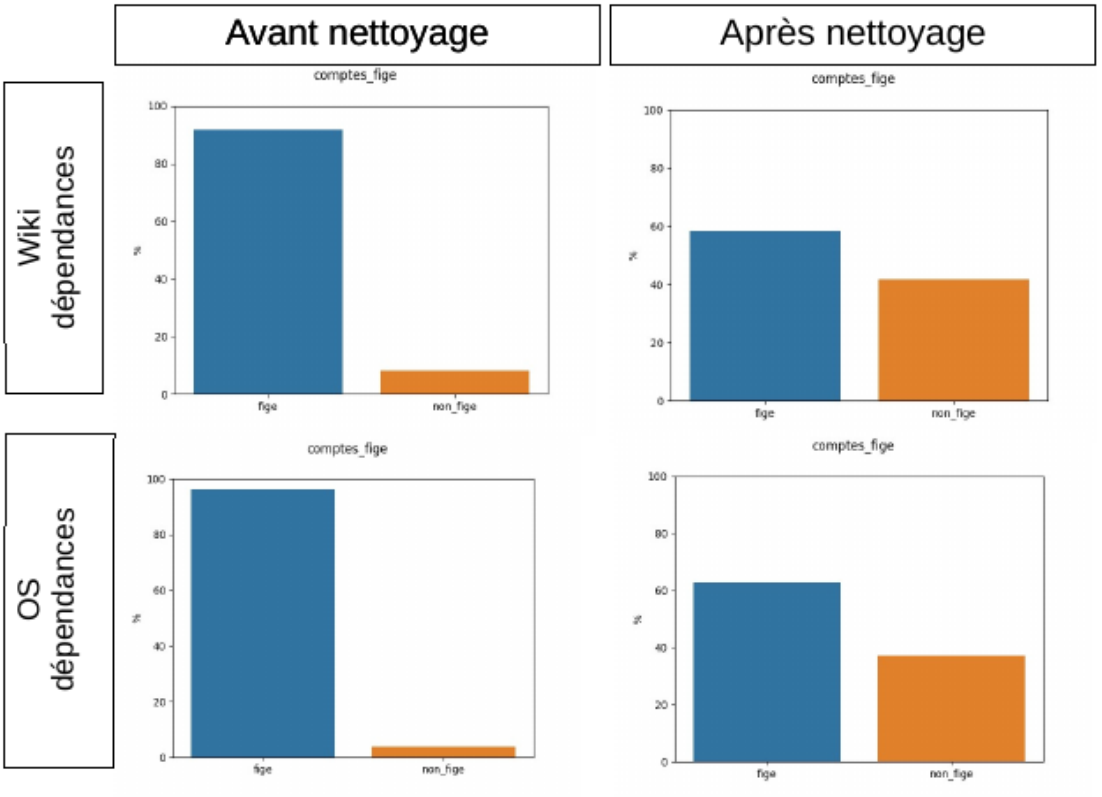
2.2. Description après un second nettoyage

Voici donc les données plus représentatives après avoir tout nettoyé, qui nous permettent d'anticiper quelques corrélations entre les caractéristiques des binômes:

	Wikipédia Linéaire	**Wikipédia Dépendances**	**OpenSubtitles Linéaire**	**OpenSubtitles Dépendances**
Bornes	3-134	3-155	3-44	3-77
Classes	1: 90542, 2: 9839, 3: 10919, 4: 23205, 5: 14035, 6: 8091, 7: 20484, 8: 14426, 9: 6691, 10: 102022	1: 109548, 2: 15112, 3: 18958, 4: 42103, 5: 26644, 6: 15409, 7: 36519, 8: 24726, 9: 10304, 10: 94862	1: 578, 2: 48, 3: 43, 4: 115, 5: 57, 6: 18 7: 84, 8: 56, 9: 42, 10: 542	1: 929, 2: 107, 3: 110, 4: 280, 5: 149, 6: 88, 7: 285, 8: 164, 9: 59, 10: 720

Comptes	'et': 2.263.216 'ou': 202.745 'fige': 209.094 'non_fige': 91.160 'alpha': 1.283.849 'inverse': 1.182.112	'et': 3.150.628 'ou': 253730 'fige': 229.826, 'non_fige': 164.359 'alpha': 1.621.532, 'inverse': 1.782.826	'et': 6.914 'ou': 1.633 'fige': 1.210, 'non_fige': 373 'alpha': 4.217 'inverse': 4.330	'et': 16.946 'ou': 2.021 'fige': 1.815, 'non_fige': 1.076 'alpha': 8.797 'inverse': 10.170

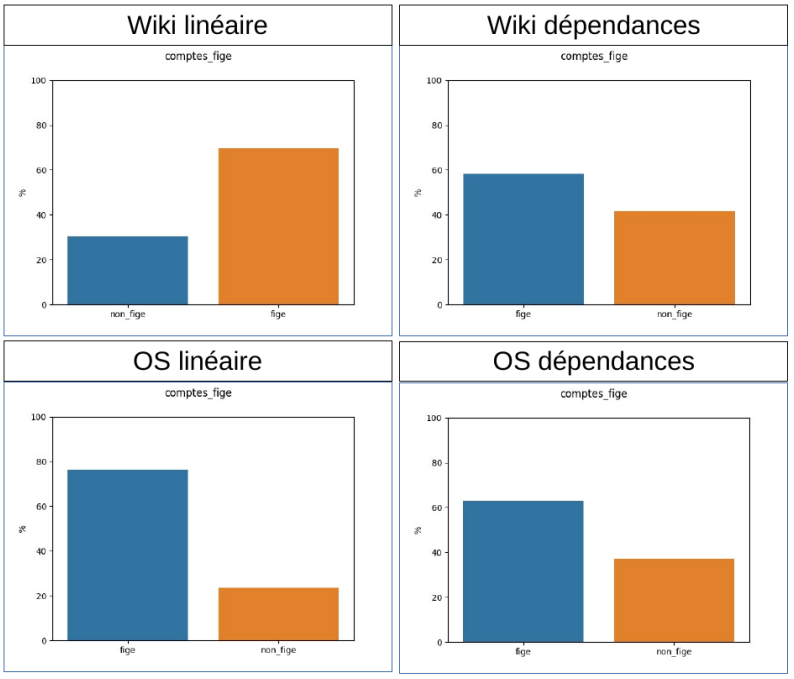
La seule différence majeure que l'on peut observer est celle qui concerne la distribution des binômes figés et non figés. Par exemple ici on observe une différence flagrante sur les extractions linéaires :



On peut voir que même si les proportions sont différentes, les binômes figés restent en part dominante. Les autres distributions quant à elles ne changent pas sensiblement (et/alpha).

Maintenant que nous avons comparé les distributions avant et après nettoyage, nous savons sur quelles distributions nous travaillons et nous allons pouvoir comparer les différents corpus entre eux pour voir si l'on peut pré-sentir certains facteurs comme jouant sur le caractère figé ou non d'un binôme. Ce qui varie selon les corpus sera plutôt décrit comme externe à la langue et donc moins pertinent, en revanche ce qui reste régulier pourrait être un facteur interne à la langue et donc être pertinent.

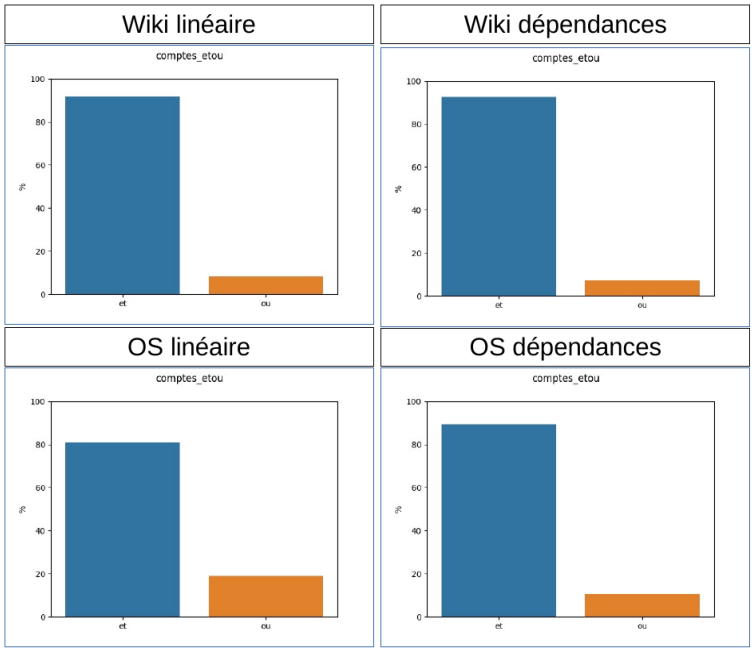
2.2.1. Proportion de binômes figés



échantillon :occurrences totales des binômes

Ces quatre histogrammes montrent que quelque soit le corpus, un binôme a statistiquement plus de chances d'être figé que non figé. Ce constat peut nous amener à faire un choix dans le classifieur, par exemple, lorsqu'il prédira pour un binôme une valeur à la limite entre figé ou non, on penchera plus pour figé.

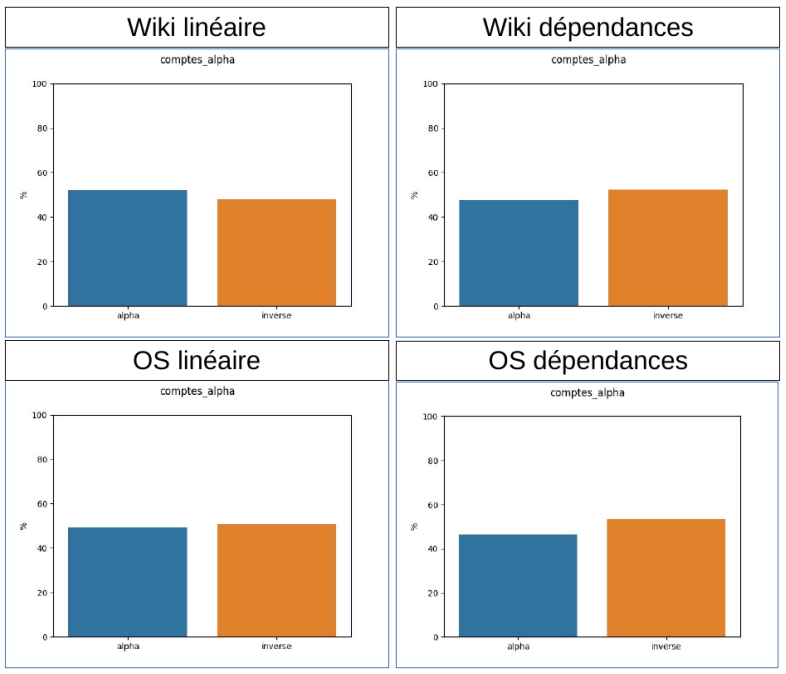
2.2.2. Proportion de binômes reliés par "et"



échantillon :occurrences totales des binômes

On remarque également que les binômes sont majoritairement des binômes joints par la conjonction "et". Cette information ne peut pas nous aider beaucoup pour la suite.

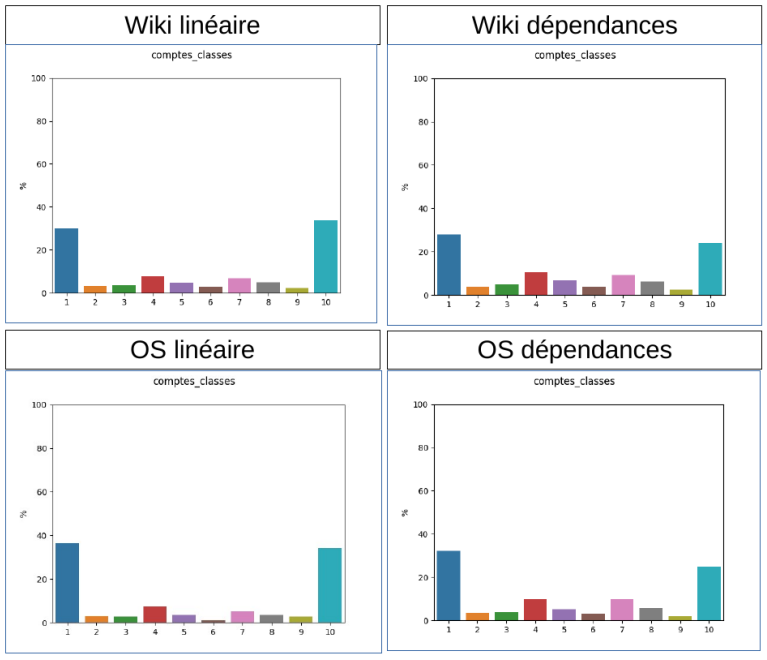
2.2.3. Proportion de binômes reliés dans l'ordre alphabétique



échantillon :occurrences totales des binômes

On voit grâce à ces distributions que la proportion de binômes dans l'ordre alphabétique ou dans l'ordre inverse semble équivalente.

2.2.4. Proportion de chaque classe



échantillon : binômes dédoublonnés

Même si toutes ces distributions suivent des schémas similaires, les extractions par dépendances semblent fournir plus de binômes non figés.

La distance entre deux mots serait-elle un facteur qui influe sur le caractère figé des binômes ?

3. Manipulation des données extraites

En lisant notamment le papier de S.Benor et R.Levy ainsi qu'une partie de la thèse de G.Couasnon, nous avons réalisé le grand nombre de facteurs rentrant en compte dans la définition de l'ordre des binômes et dans le fait qu'ils soient figés ou non. La première étape de notre réflexion a donc été de décider quels étaient les facteurs pertinents que nous allions manipuler pour arriver à tirer des conclusions à propos des données que nous avions extraites.

3.1. Extraction des features

Nous avons dans un premier temps décidé de définir le vecteur à partir des données les plus facilement accessibles, c'est à dire d'une à partir des annotations que nous avons faites, condensées dans le fichier semantic_features.pkl, et de la liste de prononciation des mots fr.pronunciation.json, et d'autre part grâce aux caractéristiques des mots composants le binôme, c'est à dire via les fichiers conll.

L'ensemble de ces facteurs nous permettant de représenter les binômes par des vecteurs. Pour ces caractéristiques on attribuera 1 si on fait face à l'ordre 'attendu' et 0 dans le cas inverse.

Les features tirés du fichier semantic_features.pkl sont les suivants :

- animé/inanimé
- positif/négatif
- concret/abstrait
- +général/-général
- causalité
- extralinguistique
- ordre logique
- ordre temporel

Les features tirés du fichier conll sont les suivants :

- singulier/pluriel
- masculin/féminin

Les features tirés du fichier fr.pronunciation.json :

- **-de syllabes/+ de syllabes** : Pour cela nous avons fait une fonction ***nb_syll()*** qui renvoie le nombre de syllabes d'un mot à partir de sa transcription phonétique, en effet le nombre de syllabes correspond au nombre de points + 1 qu'il y a dans la transcription phonétique.

- **voyelle courte/voyelle longue** : Nous avons fait une fonction **voyelle_longue()** qui teste si un mot contient le symbole ':' qui marque l'allongement d'une voyelle. Si le premier mot du binôme n'en contient pas et que le second en contient on attribue donc 1 à cette caractéristique.

Nous avons également décidé de prendre en compte l'ordre alphabétique puisque c'était un des facteurs listés dans les différents papiers. Nous avons opté pour une représentation par leur proportion, c'est à dire que nous ajoutons au vecteur une valeur correspondant à la proportion du binôme dans le sens alphabétique par rapport au nombre total d'occurrences du binôme.

3.2. Elaboration d'un classifieur

Nous touchons, à cette étape de notre réflexion, au but de ce projet qui est de pouvoir prédire si un binôme est figé ou non, et si oui, dans quel ordre. Nous avons choisi lors de la réalisation du classifieur qu'il afficherait 3 scores :

- le pourcentage de binômes dont la bonne classe a été prédite. (considère les classes comme étant 10 classes séparées)

- la proportion de binômes dont l'on a bien prédit s'ils étaient figés ou non. (considère les classes comme étant 2 classes distinctes : 1,2, 9,10 d'une part et les autres d'une autre)

- un score représentant si l'on a bien prédit que le binôme était figé dans l'ordre alpha, non figé ou figé dans l'ordre inverse. (considère donc les classes comme 3 distinctes, 1 et 2 d'une part, 9 et 10 de l'autre et enfin le reste.)

Nous avons également décidé que l'on arrêterait l'apprentissage sur le corpus de train dans les 2 cas suivants : soit la performance sur le corpus dev n'augmente plus, soit la performance sur le corpus train atteint 100%. (ce dernier cas n'arrivera jamais sur les données que l'on a extraites de wikipedia et openSubtitles mais doit être prévue pour un jeu de données plus restreint)

3.3 Manipulation du classifieur

L'élaboration de ce classifieur permet d'une part d'effectuer les prédictions, mais aussi d'essayer de se rendre compte des features qui seraient les plus décisifs dans la détermination du caractère figé ou non d'un binôme. Dans la deuxième partie nous avons déjà pu avoir un aperçu des données et de ce phénomène, mais il peut être intéressant de pousser la réflexion jusqu'à ce stade.

Nous avons pu voir justement dans la 2ème partie qui traite de la description des données, que le corpus le plus représentatif semble être celui extrait linéairement de wikipedia puisqu'il présente l'avantage de fournir un grand nombre de données (comparé à celui d'OpenSubtitiles) et les dépendances sont de façon plus certaine correctes (comparé à l'extraction par dépendances sur le même corpus).

Voici donc les résultats obtenus avec un classifieur entraîné sur l'ensemble des binômes du corpus wikipédia (linéaire) :

```
 Epoch  1

classe exacte :  train acc =  83.41429753744495  dev acc =  58.27882770982045  updates =  379762
figé non figé : train = 83.41429753744495  dev  =  61.61991761265369
figé alpha/figé inverse/non figé : train =  83.41429753744495  dev =  61.50042451495236

 Epoch  2

classe exacte :  train acc =  83.49854456597932  dev acc =  91.13392660608157  updates =  377833
figé non figé : train = 83.49854456597932  dev  =  92.95462406842552
figé alpha/figé inverse/non figé : train =  83.49854456597932  dev =  92.5709883337002

 Epoch  3

classe exacte :  train acc =  83.13456595747468  dev acc =  93.9593094556775  updates =  386167
figé non figé : train = 83.13456595747468  dev  =  95.60233954907078
figé alpha/figé inverse/non figé : train =  83.13456595747468  dev =  95.17153548630546

 classe exacte :  93.86973782477104
figé/non figé :  94.54817027632562
figé alpha/figé inverse/non figé :  94.15038717031562
```

3.3.1. Tentatives d'amélioration

Les résultats précédents furent calculés avant que l'on ne décide d'effectuer le second nettoyage sur les binômes (cf. 2.2). Nous nous sommes donc demandé si le classifieur sur les données une fois nettoyées (en enlevant les binômes au nombre d'occurrences trop bas et trop haut) serait meilleur, voici les résultats que l'on a pu obtenir :

```
 Epoch  1

classe exacte :  train acc =  54.04140207528457  dev acc =  54.443112176925126  updates =  124193
figé non figé : train = 54.04140207528457  dev  =  65.5209166000533
figé alpha/figé inverse/non figé : train =  54.04140207528457  dev =  62.30349054090061

 Epoch  2

classe exacte :  train acc =  54.06730612667822  dev acc =  62.343458566480145  updates =  124123
figé non figé : train = 54.06730612667822  dev  =  70.82334132693845
figé alpha/figé inverse/non figé : train =  54.06730612667822  dev =  67.80575539568345

 classe exacte :  60.857980282440714
figé/non figé :  68.73834265920597
figé alpha/figé inverse/non figé :  65.80735411670663
```

On voit que ce n'est absolument pas le cas, les performances sont bien inférieures, ce qui semble indiquer que même les binômes aux nombres d'occurrences très bas ne menaient pas notre classifieur en erreur bien au contraire.

La seconde amélioration que nous avons essayé d'apporter à notre classifieur consistait à pondérer la mise à jour des paramètres par la fréquence d'apparition du binôme, et ce sur tous les binômes car nous avons vu précédemment que notre nettoyage baissait les performances du classifieur :

```
 Epoch  1

classe exacte :  train acc =  82.66393558967461  dev acc =  56.377944089808494  updates =  396943
figé non figé : train = 82.66393558967461  dev  =  59.8023647055124
figé alpha/figé inverse/non figé : train =  82.66393558967461  dev =  59.74733498946574

 Epoch  2

classe exacte :  train acc =  83.7390132746938  dev acc =  94.29892141756548  updates =  372327
figé non figé : train = 83.7390132746938  dev  =  95.70453759315744
figé alpha/figé inverse/non figé : train =  83.7390132746938  dev =  95.33898305084746

 classe exacte :  94.1338783852836
figé/non figé :  95.10239377382965
figé alpha/figé inverse/non figé :  94.74077276836603
```

On peut voir que les résultats sont très légèrement supérieurs en pondérant, on décidera donc de garder cette pondération par la suite.

Nous avons également essayé de regarder l'influence du nombre d'epoch d'apprentissage de notre classifieur puisque celles-ci nous semblaient très limitées lors de nos tests mais nous avons pu nous rendre compte que les performances n'augmentaient pas si on fixait manuellement ce nombre donc nous avons laissé l'algorithme en l'état.

3.3.2. Quels features sont les plus importants ?

Essayons maintenant en manipulant le classifieur de déterminer quels facteurs sont les plus déterminants dans nos vecteurs de features pour déterminer les caractéristiques d'un binôme. Pour ce faire, entraînons le classifieur en le privant à tour de rôle des différents features et observons l'évolution de ses performances.

Nous avons précédemment décrit les différents facteurs choisis pour remplir le vecteur représentant chacun de nos binômes, nous arrivons donc au final à 12 facteurs : 7 sémantiques, 2 grammaticaux , 2 phonétiques et 1 orthographique.

Voici les résultats obtenus en entraînant le classifieur en lui fournissant uniquement les facteurs sémantiques :

```
 Epoch  1

classe exacte :  train acc =  44.87846634595438  dev acc =  44.9396245401088  updates =  1262115
figé non figé : train = 44.87846634595438  dev  =  94.51746800415081
figé alpha/figé inverse/non figé : train =  44.87846634595438  dev =  45.34763057765479

 classe exacte :  45.08391965724618
figé/non figé :  94.5631067961165
figé alpha/figé inverse/non figé :  45.484847293738454
```

On voit que les scores 1 et 3 baissent très nettement cependant cela peut être en grande partie dû au fait que beaucoup des binômes ne sont pas annotés et donc pas présents dans semantic_features.pkl. Mais on voit que ces caractéristiques sont suffisants pour le caractère figé/non figé du binôme. On pourrait alors supposer qu'avec des annotations sémantiques en nombre suffisant, on arrive à classifier les données correctement.

Voici les résultats obtenus en entraînant le classifieur en lui fournissant uniquement les facteurs grammaticaux :

```
 Epoch  1

classe exacte :  train acc =  43.68009713084057  dev acc =  45.28867016760479  updates =  1289554
figé non figé : train = 43.68009713084057  dev  =  94.62595515864281
figé alpha/figé inverse/non figé : train =  43.68009713084057  dev =  45.67073362472878

 classe exacte :  44.78990605715184
figé/non figé :  94.52144176722614
figé alpha/figé inverse/non figé :  45.14602413427145
```

Les résultats ne semblent pas probants non plus, pourtant tous les binomes pour lesquels c'est pertinant ont ces annotations.

Voici enfin les résultats du classifieur entraîné uniquement avec les facteurs phonétiques :

```
 Epoch  1

classe exacte :  train acc =  42.50705880040792  dev acc =  37.642684192321  updates =  1316413
figé non figé : train = 42.50705880040792  dev  =  94.57564227540014
figé alpha/figé inverse/non figé : train =  42.50705880040792  dev =  45.24543253356813

 Epoch  2

classe exacte :  train acc =  42.432987799685115  dev acc =  37.66469607873966  updates =  1318109
figé non figé : train = 42.432987799685115  dev  =  94.57564227540014
figé alpha/figé inverse/non figé : train =  42.432987799685115  dev =  45.105499827049464

 Epoch  3

classe exacte :  train acc =  42.53282642448012  dev acc =  44.88695324046413  updates =  1315823
figé non figé : train = 42.53282642448012  dev  =  94.57564227540014
figé alpha/figé inverse/non figé : train =  42.53282642448012  dev =  45.24543253356813

 classe exacte :  37.56141661098227
figé/non figé :  94.63228646672694
figé alpha/figé inverse/non figé :  45.18140010219724
```

En gardant uniquement un type de facteurs on voit donc que les performances du classifieur baissent mais on voit également que le score pour définir figé ou non figé reste haut et reste quasiment similaire. Nous pensons que cela est dû au fait que la quasi-totalité des binômes sont figés et que c'est la raison pour laquelle ce score reste constant et élevé.

Nous pouvons terminer l'étude de ce classifieur en regardant le poids des différents facteurs selon les classes :

	Classe 1	Classe 2	Classe 3	Classe 4	Classe 5	Classe 6	Classe 7	Classe 8	Classe 9	Classe 10
genre	1378	-408	-19543	7737	11126	217	-38	573	-85	-957
nombre	1677	-65	7	27	-969	-40	48	-3	-39	-643
voy	10	-8	0	2	34	-16	15	2	-9	-10
syll	2253	-123	-46	-2240	295	240	-179	574	93	-867
ordrealpha	-10299	4981	-11721	17037	-222	-216	-44	-957	207	1234
concret	-6	153	-2045	94	64	1628	1069	656	-90	-1523
logique	102	-266	426	18	-182	-3313	-63	2985	-422	715
temporel	358	-700	-2009	112	134	-227	2158	2229	-354	-1710
extra linguistique	-31	-119	0	0	64	1412	9	-88	-542	-705
general	-353	-489	-2025	112	120	2466	-863	-624	-24	1680
positif	-106	0	0	0	0	0	344	190	150	-578
causalité	1	-160	0	0	0	0	-141	-954	793	461
anime	-20	0	0	0	0	0	18	190	0	-208

On peut observer de grandes disparités entre ces poids de vecteurs, dont nous avons essayé de rendre compte par les couleurs dans le tableau précédent. En effet, certains poids sont restés à 0, ce qui veut dire qu'ils n'ont jamais été modifiés. D'autres peuvent atteindre des nombres faramineux comme le genre ou l'ordre alphabétique. On peut donc voir qu'il y a de grosses disparités mais pas assez de régularité dans ces dernières pour tirer quelque conclusion probante.

Conclusion

Suite aux différentes étapes de notre réflexion, il semble apparaître qu'il est possible de prédire à quel point un binôme est figé ou non (cf. résultats du premier classifieur). Cependant les manipulations que nous avons essayé d'opérer ne semblent pas fournir d'informations tangibles quand à la pertinence plus ou moins importante de certaines caractéristiques des binômes dans cette prédiction. L'observation des performances du classifieur n'utilisant que certaines données nous pousseraient à dire qu'ils seraient hiérarchisés dans cet ordre : sémantique (45%) puis grammaticaux (44%) puis phonétiques (37%) s'il fallait en donner un, mais les différences ne sont pas assez significatives et les données encore trop peu étudiées pour affirmer avec certitude que ça l'est. Ce qui est certain, c'est que nous avons tous besoin, puisque la combinaison de ces trois types de features permet quand même d'arriver à un classifieur avec une performance de presque 94%.