Université de Montréal

IFT 3335 – Devoirs

Rapport Devoir 2

par :
Jaydan Aladro (20152077)
Jean-Yves Grenier (20146099)
Ismail El Azhari (20196185)

21 janvier 2023

Table des matières

1 Introduction

Le but de ce travail pratique est de mettre en pratique les algorithmes de classification supervisée en utilisant la bibliothèque Scikit-Learn pour résoudre le problème de désambiguïsation de sens de mots. La désambiguïsation de sens de mots consiste à déterminer le sens d'un mot qui a plusieurs significations possibles en fonction du contexte dans lequel il est utilisé. Pour résoudre ce problème, nous utiliserons des ensembles de données de phrases contenant des occurrences du mot ambigu, dans lesquels le sens de chaque occurrence du mot est annoté manuellement. Nous utiliserons ces données comme exemples pour entraîner un classifieur qui pourra être utilisé pour désambiguïser le mot dans de nouveaux textes. Pour ce faire, nous avons utilisé les modèles suivants : naïves Bayes, arbre de décision, forêt aléatoire, machine à vecteur de support et perceptron multicouche. Une fois ces modèles entraînés, nous analyserons l'optimalité de chacun avec des graphes et pourrons ainsi comparer leur performance entre eux.

2 Traitement du corpus

Les mots de notre corpus sont catégorisées grammaticalement et par groupe nominaux, et le mot ambigu, 'interest', est annoté de son sens. De plus, on constate que si le mot ambigu apparaît plusieurs fois dans la phrase, seul un est annoté de son sens et les autres sont annotés avec une astérisque.
Les 6 sens données de notre mot "interest" représentent donc nos classes et nous voulons extraire de l'information textuelles de notre mot ambigu, qui sont nos caractéristique, pour pouvoir correctement déterminer son sens.

2.1 Isolation des mots

Tout d'abord, si nous voulons observer les mots autour de notre mot ambigu, nous devons les isoler et donc les séparer de leur groupe grammatical.

Pour accomplir cette tache, nous avons tout de suite penser à faire du Regex. Le Regex est connu pour être délicat et assez compliqué, nous avions donc réussi à isoler les mots avec l'expression suivante : $`[a-zA-Z\_0-9']+|[-;',.:?!`"]+/[-;',.:?!`"]$`
Cette expression prend les mots contenant toute les lettres de l'alphabet et des nombre. Il garde aussi les mot avec des apostrophe tel " 's " qu'on peut retrouver et notre mot ambigu avec "_". Ensuite. il peut aussi accepter les ponctuations.
Cependant cette expression était problématique car les acronyme de noms par exemple était un cas particulier et donc exclu.

Nous avons donc décidé de faire plus simple. On a supprimé avec du Regex les crochets et "=". Par la suite, on divise tous les mots espacé avec la fonction 'split' de Python. On peut donc renvoyer la liste des mots en excluant tout ce qui vient après un "/"

2.2 Isolation des catégories

Pour les groupes on commence en faisant quelque chose de similaire à l'isolation des mots. La différence est qu'on veut maintenant garder le mots ambigu dans la liste avec sont groupe. Ceci nous aidera plus tard lorsque nous ferrons nos sacs de mots, pour repérer où ce situe "interest". En revanche on ne veut pas garder les mots "interest" qui n'ont pas de sens attaché, on inclus donc tous mots avec une astérisque.

2.3 Utilisation des "stopwords"

Plus tard lors de l'intégration nous avons dû inclure un moyen d'exclure les "stopwords" durant notre sélection de catégories. Pourquoi seulement dans l'isolation des groupes et pas dans l'isolation des mots ? Car dans notre sorti des mots isolés nous avons une liste de mots que la bibliothèque de Scikit peut juste exclure lors de la vectorisation, mais la sortie de nos groupes sont juste des acronymes et ne seront donc pas exclus (ex : himself/PP est un stop word mais Scikit ne va pas comprendre d'exclure PP).

Le choix d'intégrer des "stopwords" pour une liste de mots ce fait à l'aide d'un paramètre lors que l'appel de la fonction "vectorize". Si on veut supprimer les "stopwords" alors on donne à l'attribut "stop_words" de notre "CountVectorizer" de Scikit la valeur "english" pour remplacer sont par défaut : "none".
Pour supprimer les "stopwords" pour une liste de catégories, on donne juste la liste des "stopwords" (lu du fichier donné sur Studium) et le programme ne va pas les inclure dans le choix des mots auxquels on prend leurs catégories.

2.4 Suppression de la ponctuation

Une caractéristique à laquelle nous avons pensé et que nous avons exploré, est la suppression de la ponctuation dans les phrases. Cela donnera à nos modèles plus spécifiquement des mots dans un espace de "n" mots autour de "interest" et pourra donc peut être mieux définir le mot. Nous verrons les performances de cette caractéristique plus loin.
Pour l'implémenter, un simple paramètre indiquant pour chaque fonctions de isolation si nous voulons ou pas garder la ponctuation, et par la suite une vérification si le mot en question est une ponctuation, suffit pour créer des listes de mots/catégories avec ou sans ponctuation.

2.5 Création des étiquette

Pour pouvoir entraîner nos modèles et évaluer leurs performance, on va récupérer les classes auquel appartiennent chaque phrase de notre corpus. Cette extraction est assez facile car il nous faut juste trouver le chiffre après chaque "_" de nos lines.

3 Différents modèles

3.1 Naïve Bayes

Nous avons commencé en utilisant le classifieur Naïve Bayes multinomial, qui est un algorithme de classification supervisée basé sur l'hypothèse de l'indépendance naïve. Cette hypothèse consiste à considérer que les différentes caractéristiques d'un exemple sont indépendantes les unes des autres. Dans notre cas, cette indépendance concerne les catégories de mots qui apparaissent dans la phrase, ce qui simplifie grandement le calcul de la probabilité conjointe de toutes les caractéristiques.
Nous avons utilisé le module **MultinomialNB** de la bibliothèque **scikit-learn** en lui donnant en entrée une matrice de vecteurs de caractéristiques (X) et un vecteur d'étiquettes (y). Puis on a séparé les données en un ensemble d'entraînement et un ensemble de test grâce à la fonction **train_test_split** de scikit-learn. Le pourcentage de données réservé au test est de 30% et la graine aléatoire utilisée est fixée à 42 pour s'assurer que les résultats seront toujours les mêmes chaque fois que le code est exécuté avec la même graine.
Après on a calculé le score de précision en comparant les étiquettes prédites par le classifieur aux étiquettes réelles des données de test, ceci est donné en divisant le nombre de prédictions correctes par le nombre total de prédictions.
Ensuite on a créé deux matrices de vecteurs de caractéristiques à partir de la liste de lignes, une matrice qui inclut les **stop words** et une autre qui les exclut. Ces matrices sont obtenues en utilisant la fonction vectorize du module main pour voir l'impact de l'inclusion ou l'exclusion des mots qui n'ont à priori aucune signification sémantique dans la phrase.
Pour résumer, nous avons créé des dictionnaires de résultats pour les phrases avec et sans ponctuation, avec et sans mots de liaison. Pour chaque combinaison de données, nous avons appelé la fonction model et stocké les objets dans le dictionnaire de résultats. Ensuite, nous avons tracé quatre graphiques de la précision en fonction du nombre de mots/catégories autour de la cible, un pour chaque combinaison de données.

3.2 Arbre de décision

Le modèle de l'arbre de décision consiste à créer un ensemble d'attributs pour décrire un problème. Son objectif est de créé un modèle qui prédit la valeur d'une cible à en apprenant de simples règles de décision déduite des caractéristiques des données en important le module **DecisionTreeClassifier** de la librairie **sklearn.tree** . Il est simple à interpréter et à comprendre, l'arbre peut aussi être visualisé. Il est possible de valider le modèle en utilisant des tests de statistique. Il performe bien même si les hypothèses sont déformées par le vrai modèle à partir lequel a été généré par les données fournies. Il peut par contre créer des arbres trop complexes qui ne généralisent pas très bien les données et donc le modèle devient surentraîné. Il peut aussi devenir instable, car de petites variations dans les données peuvent résulter en un arbre complètement différent et ainsi mitige l'ensemble des décisions d'un ensemble. L'arbre peut aussi figer sur un maximum local et faire fit d'un maximum global, ainsi

on peut ne pas obtenir la meilleure décision possible. Donc nous créons un arbre pour guider à prendre une décision en suivant la valeur des attributs. À chaque nœud, on utilise un attribut pour choisir d'aller à un enfant et lorsqu'on arrive à une feuille, on prend une décision. On peut "stocker" tous les exemples lors de l'apprentissage en créant un chemin pour chaque exemple. L'arbre choisira comme racine du sous-arbre l'attribut le plus significatif. La performance d'apprentissage est la précision de prédiction en utilisant **accuracy_score** et sera mesurée sur un autre ensemble de tests.

3.3 Forêt aléatoires

Les forêts aléatoires sont un modèle de machine learning utilisé pour la classification et la régression. Elles construisent plusieurs arbres de décision en choisissant une caractéristique pour maximiser l'homogénéité des groupes de données. Elles combinent ces prédictions pour en soustraire une prédiction finale. Ici nous avons divisé les données en jeux de données d'entraînement et de test avec une taille de test de 0,3 et une graine aléatoire de 42.
Ensuite, nous avons instancié un classifieur de forêt aléatoire de scikit-learn en utilisant la fonction **RandomForestClassifier**, et nous l'avons entraîné en utilisant les données d'entraînement.Les prédictions sont stockées dans la variable d'instance y_pred, pour ensuite calculer le nombre de points mal étiquetés en comparant les vraies cibles y_test aux prédictions y_pred.
La fonction "model" nous aide à spécifier le nombre de mots ou de catégories à considérer autour de du mot cible dans chaque ligne. En variant ce nombre entre 1 et 50, nous pouvons constater son effet sur la précision de notre modèle. La fonction crée d'abord deux matrices de caractéristiques en utilisant la fonction "vectorize". La première matrice, "x_nostopwords", est créée en utilisant les lignes "lines" et en excluant les stopwords. La seconde matrice, "x_stopwords", est créée en utilisant "categories" et en excluant les mots de liaison, si "categories" est spécifié. Si "categories" n'est pas spécifié, alors "x_stopwords" est créée en utilisant "lines" et en incluant les mots de liaison.
Enfin, nous avons créé des dictionnaires de résultats pour les phrases en fonction du nombre de mots autour de "interest" et en fonction des catégories autour de "interest", avec et sans les mots de liaison, et finalement nous avons tracé les graphiques de la précision.

3.4 Machine à vecteurs de support (SVM)

Les SVM sont un type de modèle de machine learning utilisé pour la classification, la régression et les problèmes d'optimisation convexe. Elles fonctionnent en trouvant un hyperplan optimal de séparation entre les classes de données dans un espace de dimensions élevées. Elles maximisent la distance entre cet hyperplan et les points les plus proches de chaque classe afin de créer une marge maximale, plus la marge est grande, plus le modèle est considéré comme stable et moins sensible aux bruits de données.
Nous avons encore séparé nos données aléatoirement en données d'entraînement et de test avec le même ratio de 70/30. Ensuite, Nous avons crée un objet SVM en utilisant

la classe **svm** de **scikit-learn** et on l'a entraîné avec les données d'entraînement en appelant la méthode **fit**.
Notre fonction model prend en entrée 3 arguments, parmi lesquels un entier n qui représente le nombre de mots ou de catégories à inclure autour de la cible dans chaque ligne. En modifiant ce nombre, nous pouvons obtenir un graphe montrant l'impact de ce paramètre sur la précision de notre modèle. Notre fonction crée d'abord deux matrices de caractéristiques x en utilisant la fonction vectorize. La première matrice, **x_nostopwords** est créée en utilisant lines et en excluant les mots de liaison (c'est-à-dire les mots qui n'apportent pas de signification à une phrase, comme "your", "she", etc.). La seconde matrice, **x_stopwords**, est créée en utilisant categories et en excluant les mots de liaison, si categories est spécifié. Si categories n'est pas spécifié, alors **x_stopwords** est créée en utilisant lines et en incluant les mots de liaison.
Enfin, nous avons créé des dictionnaires de résultats pour les phrases en fonction du nombre de mots autour de interest et en fonction des catégories autour de interest, avec et sans les mots de liaisons., et enfin nous avons tracé les graphiques de la précision.

3.5 Perceptron multicouche (MLP)

Le MLP est constitué d'un ensemble de neurones représentant les caractéristiques d'entrée. Chaque neurone de la couche cachée transforme les valeurs de la couche précédente avec une sommation linéaire pondérée. La couche de sortie reçoit les valeurs de la dernière couche cachée et les transforme en valeur de sortie. La création du modèle est faite en 2 sections, une avec les stopwords et l'autre sans. Ensuite on a sélectionné de façon à optimiser les dimensions des couches cachées. Nous avons utilisé pour l'entraînement le module **MLPCLassifier** de la librairie **sklearn_neural_network**. Nous avons pris une approche avec des couches de formats différents, une avec un nombre élevé (32,16) , une avec un nombre jugé moyen (16,8) et la dernière une couche plus petite (2,2). Nous avons ensuite fourni une suite d'hyperparamètre α [0.5, 0.1, 0.01, 0.0001, 0.00001] à la fonction
$find\_optimal\_hyperparameter()$ et avons testé l'exactitude des modèles. Le candidat optimal retenu a été α=0.00001. Un désavantage du MLP est que dans les couches cachées, il existe plusieurs minimums locaux donc on doit initialiser les poids de façon aléatoire pour valider différents tests de précision du modèle.

4 Comparaisons et analyses

4.1 Performance des différents algorithmes

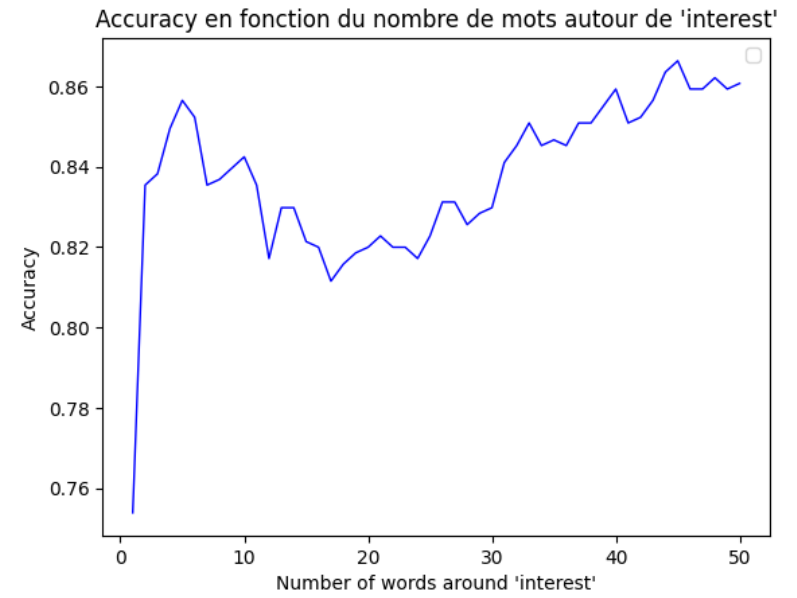
Dans tous les algorithmes on observe la même tendance dans leur prédiction. En prenant en dessous de 5 mots autour de notre mot ambigu, la prédiction est très bonne (>0.80).En revanche, avec juste un mot autour de "interest" la précision est très différente de 2 ou plus. On pourrait ce dire que parfois nous avons des stopwords ou de la ponctuation autour du mot et cela nous donne donc pas beaucoup de contexte. La précision baisse rapidement en ajoutant des mots jusqu'à environ 15 mots. Ceci semble normal car plus on s'éloigne du mots, moins les autres mots auront de corrélation avec le mot cible.
Quand on prend plus de 15 mots, on commence à reprendre une bonne précision. Cela est car nous commençons à prendre des phrases complète (ou presque au complet) et donc avons le contexte complet de la phrase ce qui nous donnera une bonne prédiction sur le mot ambigu.
Si nous augmentons au dessus de 50 mots autour, la précision commence à ce stabiliser au dessus de 0.80.

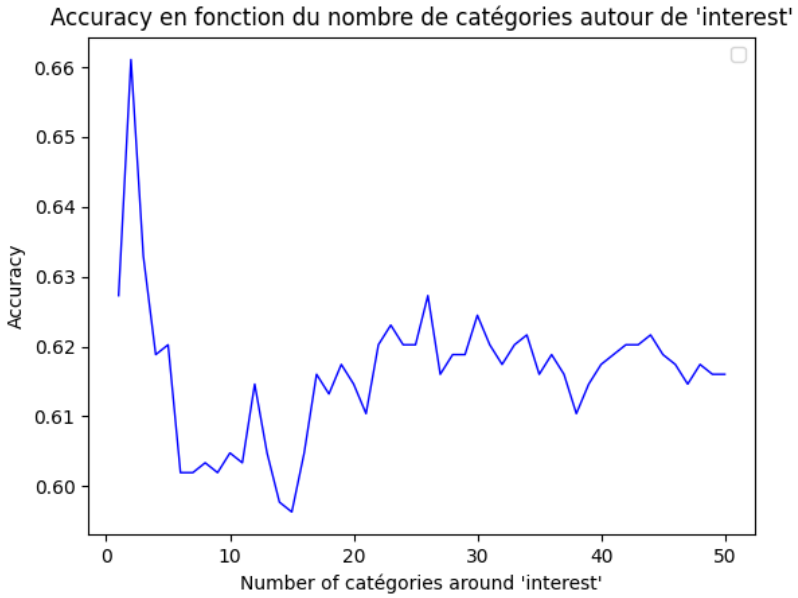
Les tendances pour les catégories autour des mots sont complètement différentes qui sont on prenait des mots. Les précisions sont assez bonnes pour des catégories proches de "interest" (< 3 mots) mais moins que les mot (0.10 de moins). Plus de catégories que ça, affecte très peu la précision et elle stagne entre 0.5 et 0.6. Ceci est prédicables car une catégorie donne moins de contexte qu'un mot, car beaucoup de mots ont les mêmes catégories, donc plus on s'éloigne du mot ambigu moins les catégories ont d'importance sur le sens de "interest" mais si on a la structure complète de la phrase est un peu mieux pour prédire.

On créer 50 de chaque modèles, sauf pour le MLP, car nous prenons des sacs de mots/catégories de 1-51 pour chaque modèle. Cela veut dire que les temps d'entraînements sont pour le totale des 50 modèles.

4.1.1 Naive Bayes

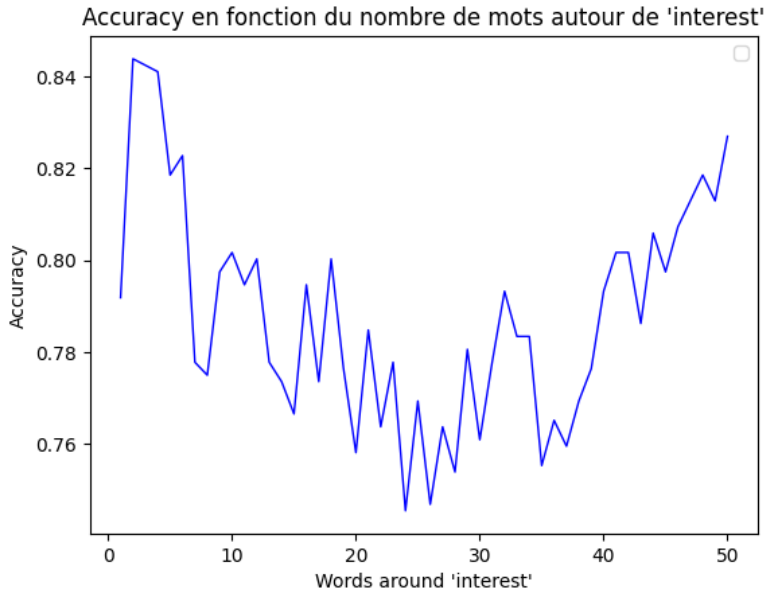
Le modèle de Naive Bayes multinomiale est le plus rapide pour s'entraîner. Il effectue son entraînement sur seulement les mots en moins de 4 secondes et des catégories en environ 2 secondes !
Les prédictions varie entre 0.80 et 0.86 pour les mots (sans compter n=1) ce qui est très bon !
Cependant, pour les catégories, les prédictions varient de 0.66 à 0.6

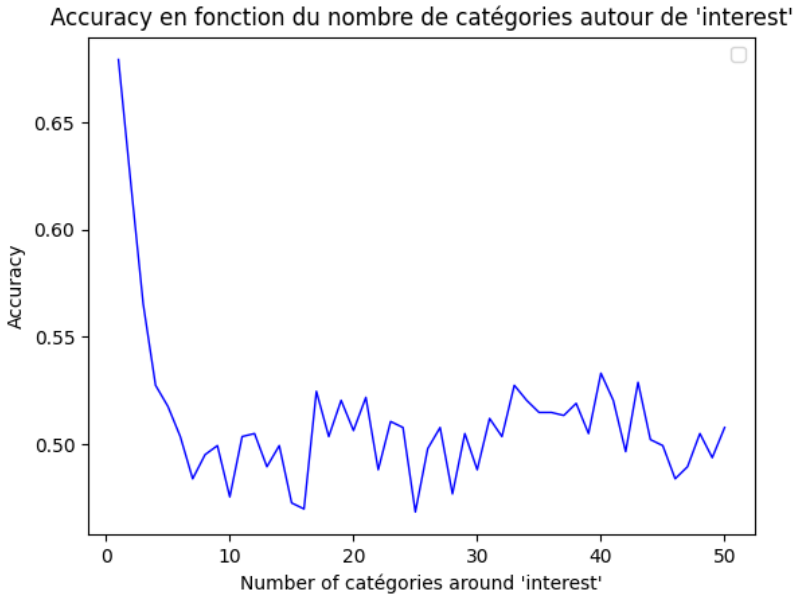




4.1.2 Decision Tree

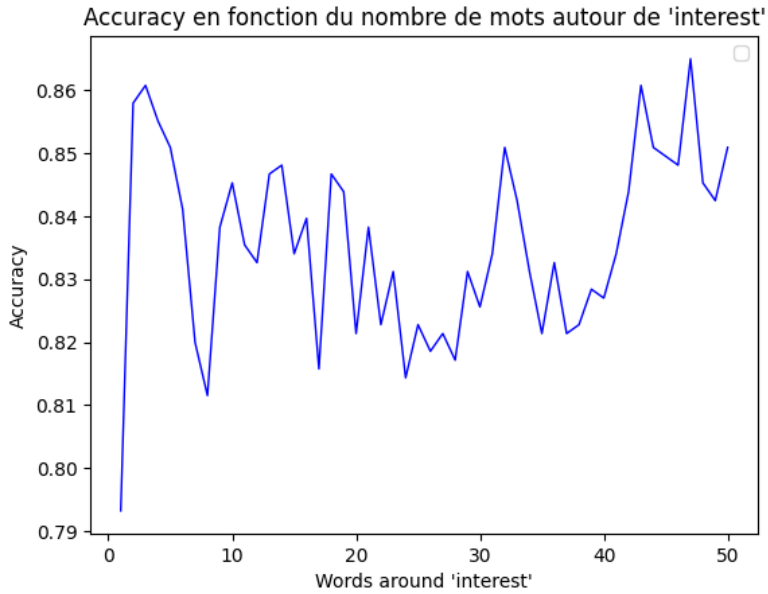
Les arbres de décision sont moins précis mais pour les $n = 1$ catégorie, on observe une bonne précision comparé à Naive Bayes. L'entraînement est légèrement plus long sur les mots que Naive Bayes avec un temps d'environ 11s mais reste similaire pour les catégories avec environ 3s. Parmis tous les modèles, les arbre de décisions ont la moine bonne précision globale.

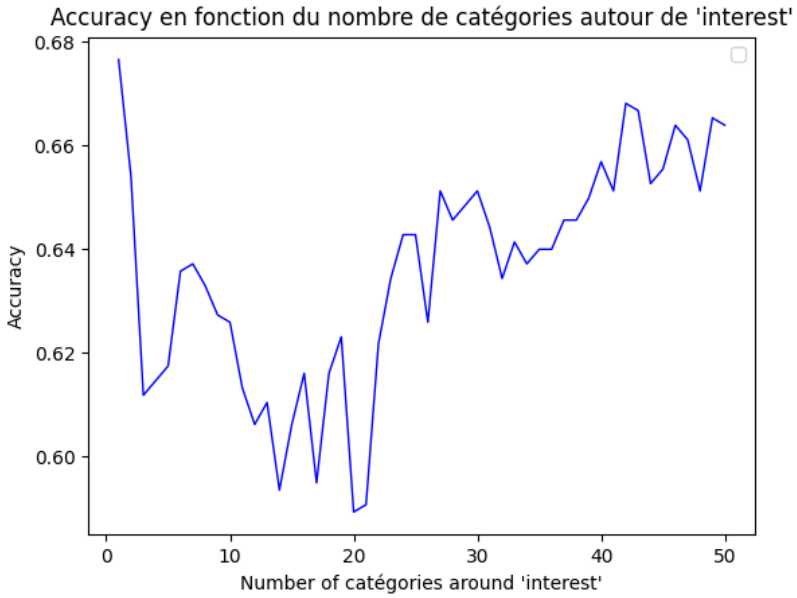




4.1.3 Forêt aléatoire

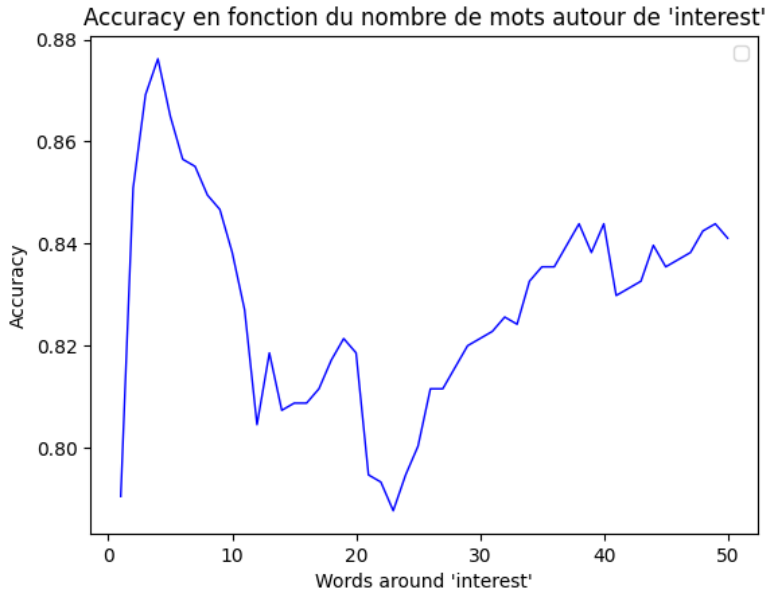
Les forêts aléatoires sont parmi les plus lents à entraîner. On prend environ 1m23 pour faire nos entraînements sur les mots et 25s sur les catégories. En revanche, les précisions de prédictions sont beaucoup plus stable que tous les autres modèles et garde une forte précision entre 0.80 et 0.86 pour les mots. Les précisions des catégories suivent étrangement les mêmes tendances que sur les mots dans les modèles auparavant.

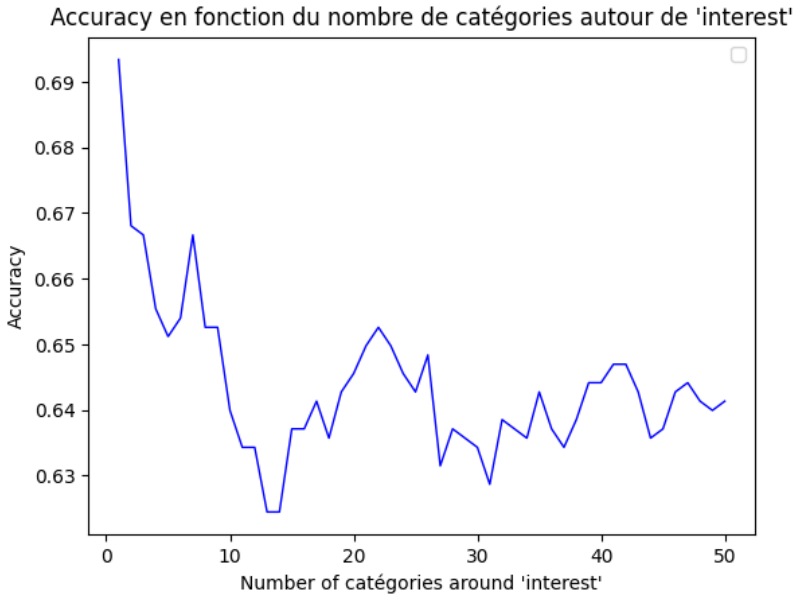




4.1.4 SVM

Les SVM ont aussi un entraînement long (1m08s mots et 33s catégories) mais ont une des meilleure précision pour en dessous de $n = 5$. Même si ont retrouve les mêmes tendances qu'avant, les précisions reste haute pour les mots mais malheureusement sont légèrement bas que les autres modèles pour les catégories.





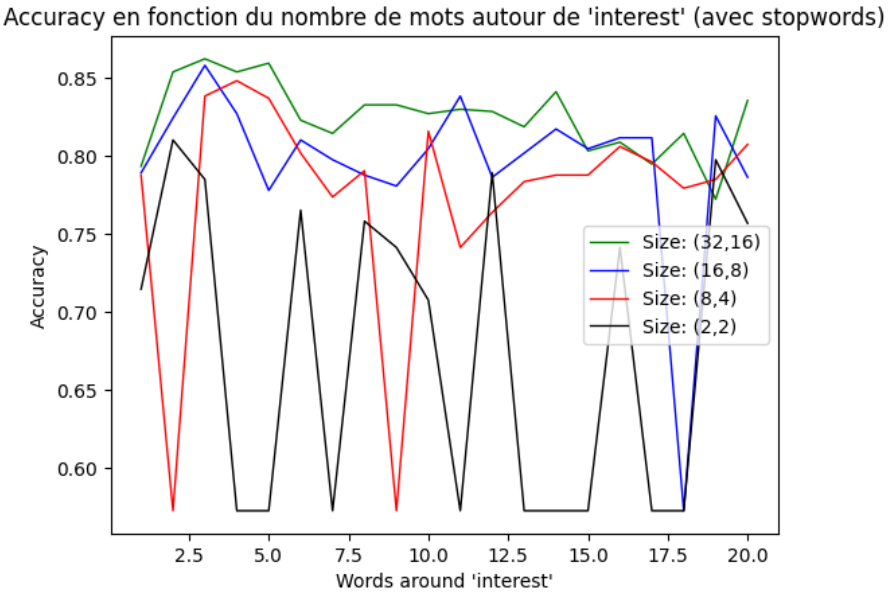
4.1.5 MLP

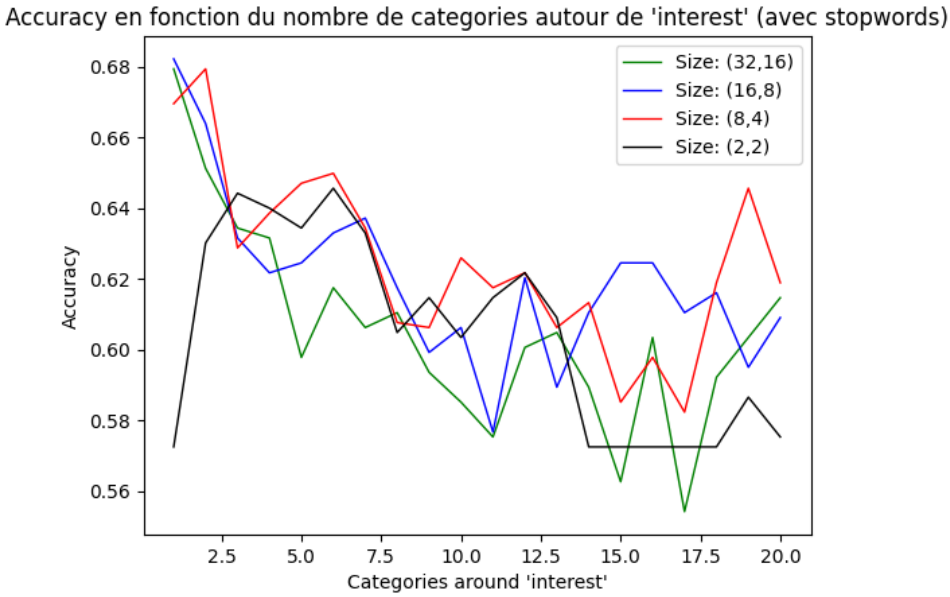
(Veuillez notez qu'on a fait que jusqu'à $n = 20$ pour le MLP car c'est très long)

Le MLP est bien évidemment le plus long car chaque couches doivent être entraîné et cela dépend aussi de notre nombre d'itération. Donc plus nous avons beaucoup de couche caché, plus on prend longtemps pour entraîner nos modèles. On remarque que pour obtenir une précision assez constante, il nous faut un grand nombre de couche (32 couches caché avec 16 perceptron dans notre cas), si nous n'avons pas assez de couche on se retrouve dans du sous apprentissage.
Il nous faut un grand nombre d'itérations car notre hyperparamètre de convergence est assez petit, cependant on constate qu'après un certain nombre d'itérations, on ne gagne pas plus de précision pour le temps que ça prend à entraîner. Nous avons

essayé avec 50,100 et 500 itération et trouvé qu'un maximum de 100 itération est optimal, 50 itérations ferait l'affaire aussi.
On remarque que la précision ne baisse pas autant lors de rajout de mots autour de "interest" (pour un grand réseau) ceci est car les poids des couches sont déjà très ajusté et l'ajout ne va donc pas autant influencer les poids. Une observation intéressante est que pour les catégories, certes on retrouve la tendance qu'on a une meilleure précision proche du mot ambigu, mais un réseau de tailles moyenne donne une meilleure précision qu'un plus grand réseau.

Voici les résultats observés pour 100 itérations :





4.2 Impact de différentes options

4.2.1 Catégories vs. words

Pour tous nos modèles les catégories ont une moins bonne précision que les mots. On suppose que la structure de la phrase ne donne pas beaucoup de contexte sur le mot "interest". Vu que les mots autour ont un sens eux-même, peut-être que certains mots ont la même catégorie mais un sens différente et donc leur utilisation va changer le sens de "interest" mais pour nos modèle c'est la même catégorie.

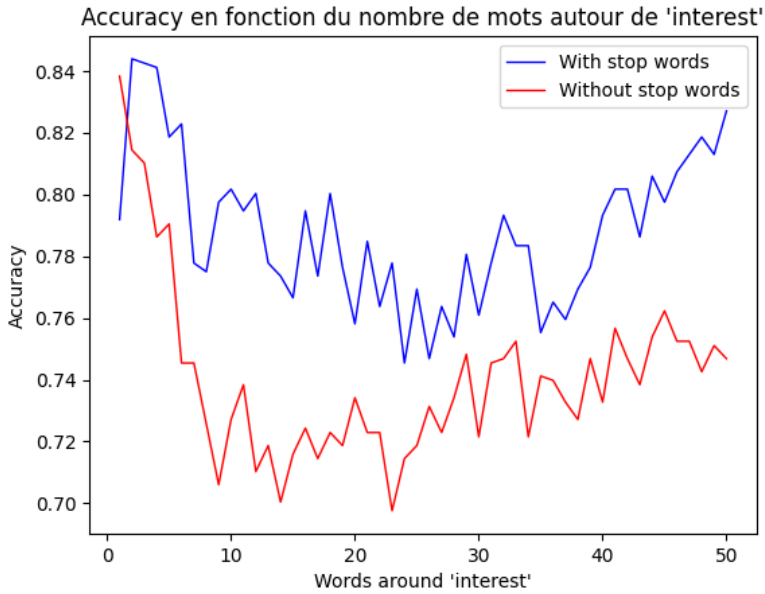
Les catégories aux alentours proche du mots ambigu vont être plus utile que la structure de la phrase au complet car certains sens de "interest" forme des groupes nominaux. Par exemple la structure "an/DT interest_6/NN rate/NN" sera plus spécifique à certains sens comme le sens 5 ou 6 que les autres.
Voici quelques explications plausibles pour ses résultats :
1. La baisse initiale de la précision lorsque le nombre de mots augmente peut-être due au fait que les vecteurs de caractéristiques deviennent trop grands et trop espacés, ce qui peut rendre difficile pour le classifieur de modéliser les données de manière précise.
2. L'augmentation de la précision lorsque le nombre de mots augmente davantage peut être due au fait que les vecteurs de caractéristiques sont maintenant assez grands pour capturer suffisamment de contexte et de signification pour améliorer les performances du classifieur.
3. La précision globale plus faible du classifieur lorsqu'il est entraîné sur des vecteurs de caractéristiques sans mots de liaisons peut être due au fait que les vecteurs de caractéristiques contiennent moins de contexte et de signification, comme mentionné ci-dessus

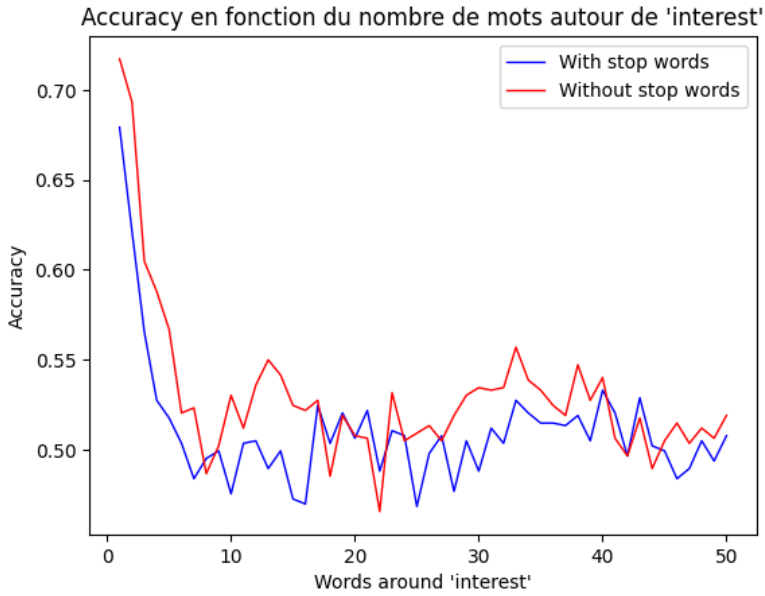
4.2.2 Stopwords

De manière générale, enlever les stopwords améliore la performance pour les $n = 1$ mots autour de "interest". Ceci est sûrement car les stopwords accroché au mot "interest" ne vont pas donner beaucoup de contexte au mot, on prend donc un autre plus proche qui a plus de contexte. Sinon au delà de $n = 1$, il semblerait que les stopwords donne moins de contexte au mot ambigu et donc la précision est moins bonne tout le long.

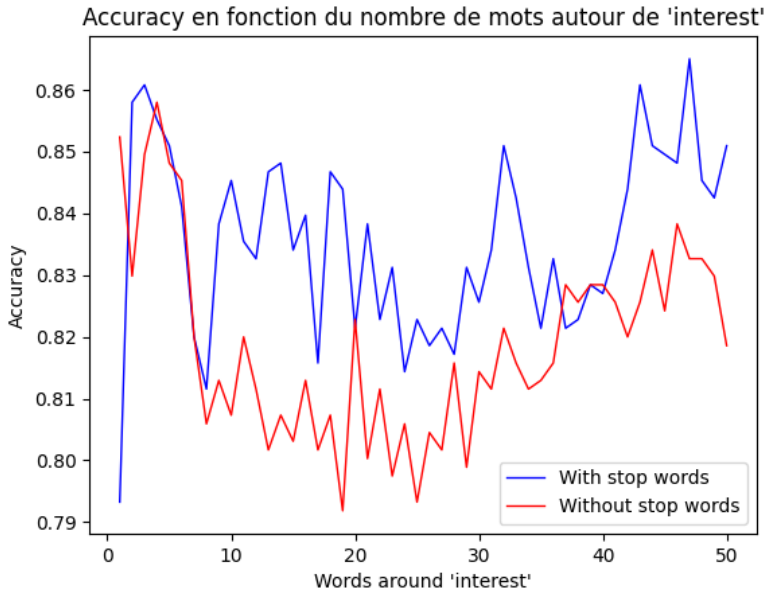
Tout change pour les catégories, les précions reste plus basse que les mots mais est mieux qu'avec des stopwords ! On suppose que les stopwords brouille la structure autour du mot "interest".

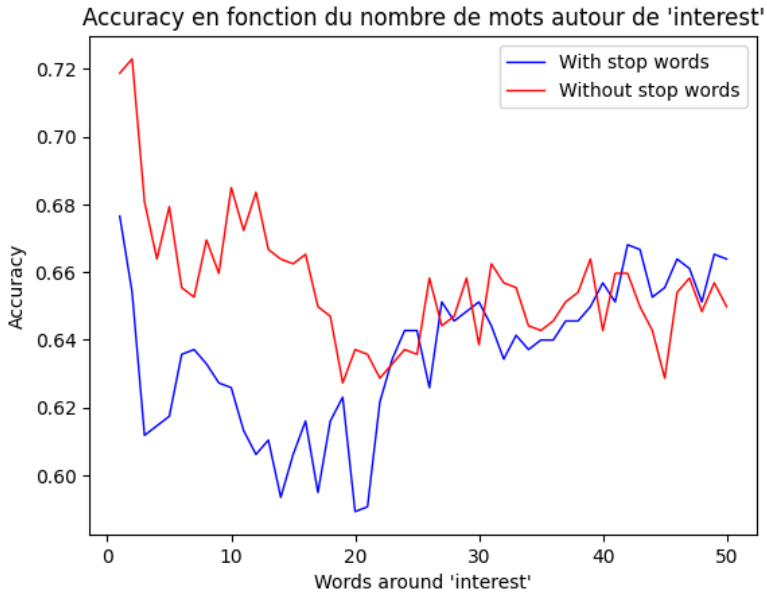
Voici les résultats pour nos modèles : **Naive Bayes** (Graphes dans partie de la ponctuation) **Decision Tree**



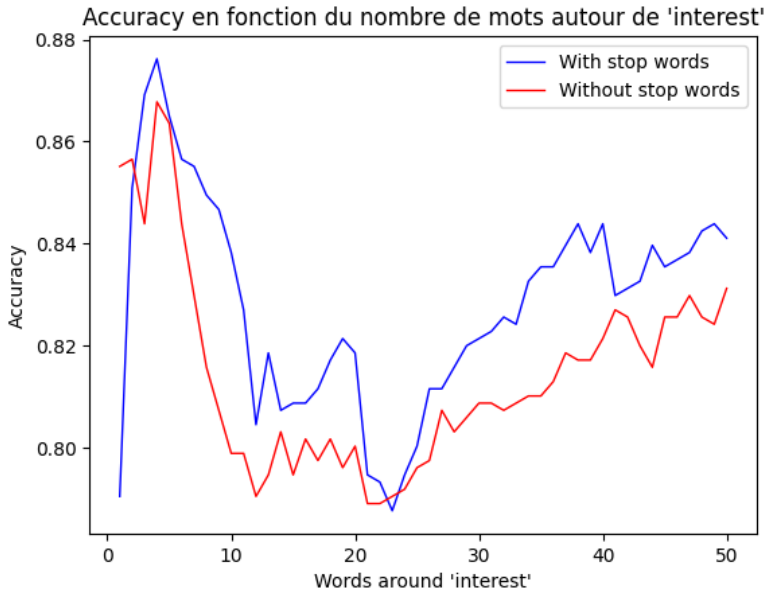


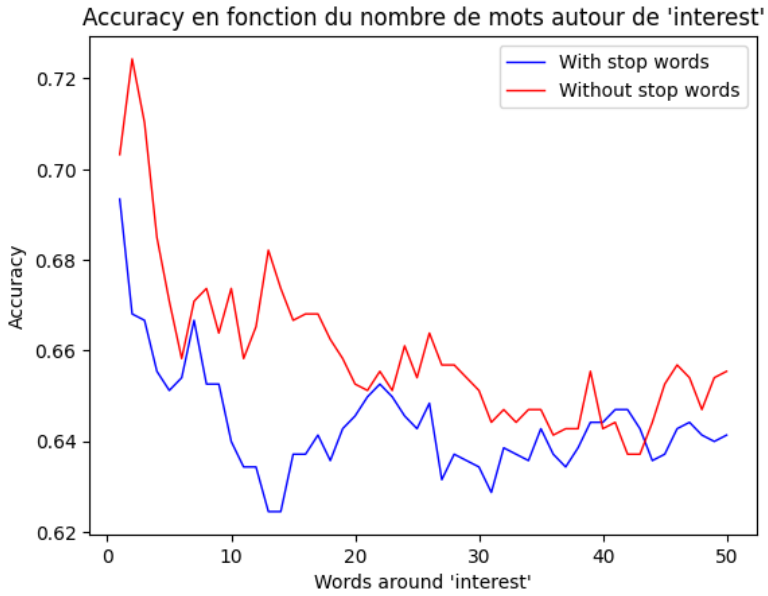
Forêt aléatoire


Accuracy en fonction du nombre de mots autour de 'interest'
Accuracy
Words around 'interest'
With stop words
Without stop words
0.86
0.85
0.84
0.83
0.82
0.81
0.80
0.79
0
10
20
30
40
50

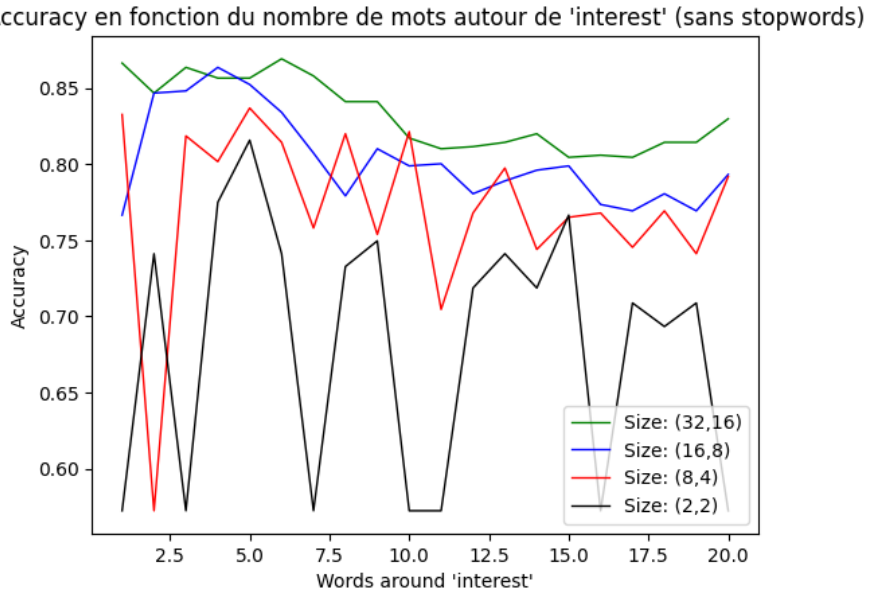

Accuracy en fonction du nombre de mots autour de 'interest'
Accuracy
Words around 'interest'
With stop words
Without stop words
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0
10
20
30
40
50

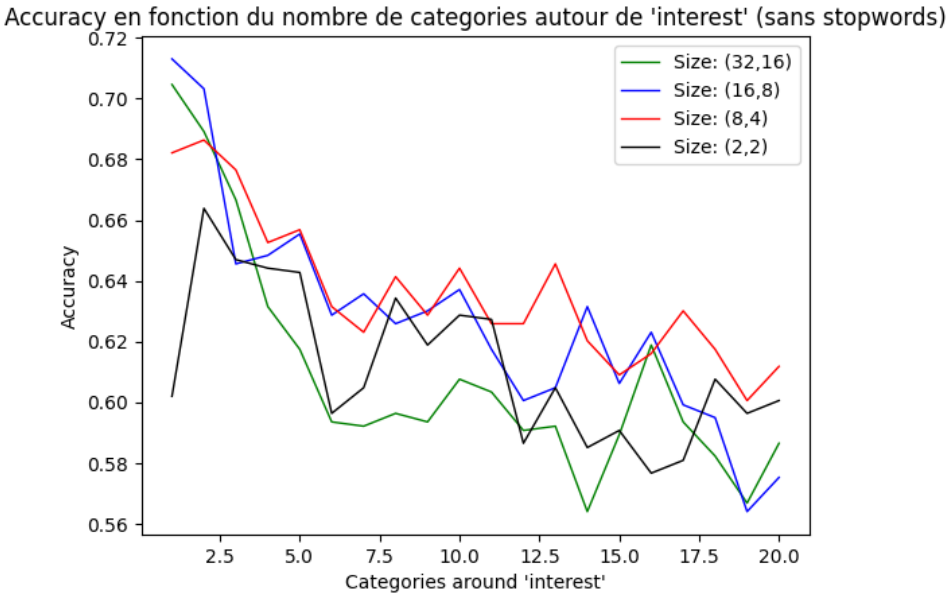
SVM


Accuracy en fonction du nombre de mots autour de 'interest'
With stop words
Without stop words
Accuracy
Words around 'interest'
0.88
0.86
0.84
0.82
0.80
0
10
20
30
40
50


Accuracy en fonction du nombre de mots autour de 'interest'
With stop words
Without stop words
Accuracy
Words around 'interest'
0.72
0.70
0.68
0.66
0.64
0.62
0
10
20
30
40
50

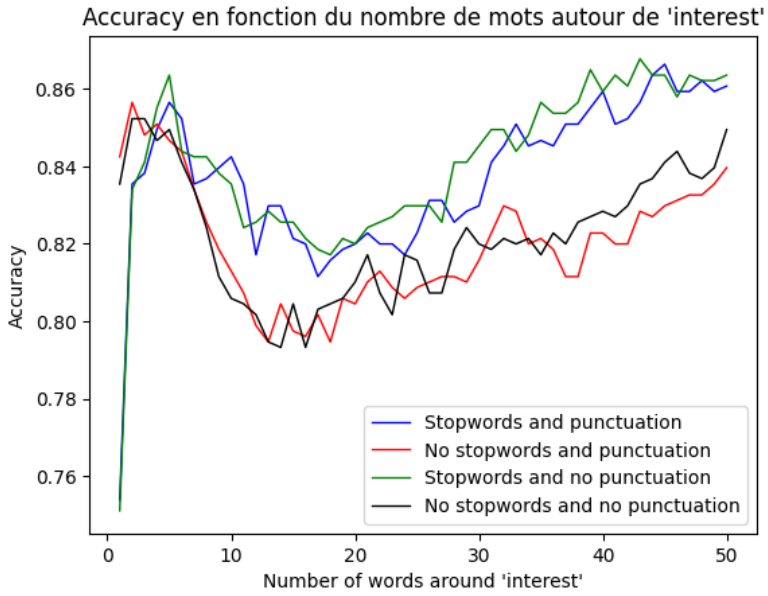
MLP

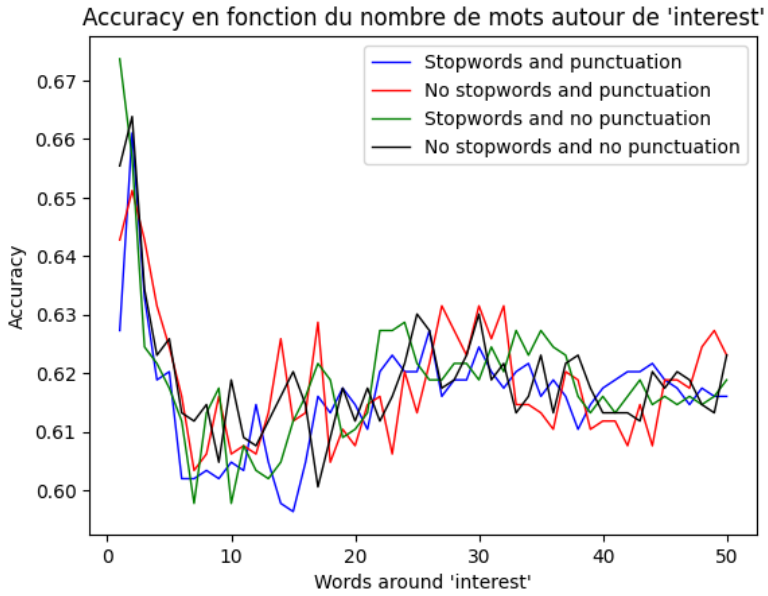




4.2.3 Ponctuation

Une caractéristique que nous avons voulu testé est d'enlever toute les ponctuations. Nous l'avons seulement testé sur Naive Bayes. Malheureusement on observe peu de différence sur les performances de nos modèles mis à part une légère amélioration pour $n = 1$ catégories (même théorie que pour les stopwords).





4.2.4 Troncature des mots

Un autre aspect que nous n'avons pas eu le temps d'explorer est la troncature des mots pour leur donner un sens général. Notre théorie est que cela va grandement aider la précision en combinaison avec la suppression des stopwords car on va regrouper des variation de mot en une seul definition dans nos modèles.
Par exemple "computers, computer, computation" sont considérés comme tous des mots différents par nos modèles (vectorisation count) mais la plus part du temps vont être accordé à qu'un seul mot ambigu similaire dans plusieurs phrases. En prenant donc que "comput", notre modèle retrouvera ce mot dans d'autres phrases ayant le même sens et ne considérera pas les mots en entiers comme des cas séparés.

5 Conclusion

Pour conclure, nous avons utilisé 5 algorithmes de machine learning différents (Naive Bayes, arbre de décision, forêt aléatoire, SVM et MLP) pour prédire la classe d'un mot ambigu dans un corpus de phrases en anglais. Nous avons comparé la performance de chaque algorithme en fonction du nombre de mots ou de catégories pris en compte autour du mot cible, avec ou sans ponctuation et avec ou sans mots de liaison. D'après les résultats, il semblerait que les meilleures caractéristiques pour prédire le sens du mot "interest" dans le corpus analysé soient les mots qui l'entourent.

En outre, il a été constaté que l'inclusion des mots de liaison et de la ponctuation a eu un petit impact positive, bien que ce dernier reste négligeable, sur la précision quand on utilise les catégories de mots. Or dans le cas d'utilisation des mots à la place des catégories de mots, l'impact de la suppression des stopswords est plus marqué et à un impact négative sur la qualité de précision des modèles.
Pour les modèles, Naïve Bayes semble être la plus efficace du point de vue du temps, du coût et de la précision. Les SVM sont bons mais plus longs à entrainer mais pour donner à la fin une performance similaire aux naïves bayes, et les MLP sont excellents pour les prédictions cependant il leur faut un grand réseau de neurones avec plusieurs couches ce qui entraîne un coût important au niveau du temps.