PROJET FINAL

À partir d'une critique de produit textuelle, déduire la note numérique associée.

L'intégralité du code est disponible à l'adresse :
https://github.com/stressGC/Text-Mining-MultinomialNB-SVM

Par

DEMON ANTOINE

COSSON GEORGES

TABLE DES MATIÈRES

I. FOUILLE DE TEXTE

a. Qu'est-ce-que c'est ?

La fouille de texte est une spécialisation de la fouille de données et fait parti du domaine de l'intelligence artificielle.
Elle consiste en un ensemble de traitements informatiques ayant pour but d'extraire des connaissances d'un texte brut.

b. Exemples d'applications

- **Texte clustering** : regroupe des textes similaires sans informations extérieures.
- **Information retrieval** (IR) : récupérer des documents pertinents à une requête, par exemple les services de moteurs de recherches.
- **Information extraction (IE)** : extraire des noms, dates, entités dans un document
- **Text classification** : classification automatique d'un document dans une catégorie

c. Problématique

Notre problématique s'inscrit dans le domaine de la classification de texte. Nous souhaitons, à partir de critiques de produits Amazon, prédire la note mise par l'utilisateur sur celui-ci. Ce genre de fouille de texte est très utile pour aider à la prise de décisions commerciales etc... Cela apporte une plus-value à la critique textuelle.

d. Jeu de données

Ce jeu de données contient des critiques d'Amazon. Les données couvrent une période de 18 ans, et comprennent environ 35 millions d'entrées jusqu'en mars 2013. Les révisions incluent des informations sur les produits et les utilisateurs, des évaluations et une analyse en texte brut. Le jeu de données que nous avons choisi est un sous jeu de données traitant uniquement de critiques sur les téléphones portables et leurs accessoires.

Nous avons donc un dataset contenant 9 variables et environ 200 000 entrées. Les variables sont :

- **productId**: identifiant unique Amazon
- **productTitle**: nom du produit
- **productPrice**: prix du produit
- **userId**: identifiant unique de l'utilisateur
- **userProfileName**: nom de l'utilisateur
- **reviewHelpfulness**: proportion d'utilisateurs ayant trouvé la critique utile
- **overall**: note attribuée au produit
- **reviewTime**: heure et date où la critique a été émise
- **reviewText** : critique textuelle

II. PRÉTRAITEMENT

a. Récupération des données

Le jeu de données que nous avons utilisé pour notre projet est téléchargeable à l'adresse web suivante : http://jmcauley.ucsd.edu/data/amazon/. Le jeu de données complet contient environ 200 000 critiques d'articles. Nous avons gardé les 50 000 premières.
Il est dans un format json, qui n'est pas très pratique d'utilisation avec python qui ne le supporte pas nativement, et encore moins avec le package « pandas » avec lequel nous travaillons. Nous avons donc converti les données en format csv.

{ "reviewerId" : "A30TL5EWN6DFXT", …, "overall" : 4.0}\n

devient :

reviewerId, …, overall
A30TL5EWN6DFXT, …, 4.0

De plus, nous gardons uniquement les variables « **reviewText** » et « **overall** », et nous débarrassons des métadonnées qui nous sont inutiles.
Cette opération est assez longue (environ 1 minute pour 50 000 lignes), c'est pourquoi nous enregistrons notre nouveau dataset en format csv pour ne pas avoir à la reproduire à chaque fois que nous lancerons le script. Le script de conversion est disponible sur le GitHub « data_fetch.py ».

b. Prétraitement

Nous avons mis en place un « pipeline » de prétraitement dans lequel nous appliquons les filtres suivants :

- Mise en minuscule

Dans nos algorithmes, nous nous basons sur la fréquence d'apparition des mots pour faire notre analyse. C'est pourquoi nous convertissons toute la séquence de mots en casse minuscule.
Ex : "Computer" = "computer" en terme de sens

- Retrait de la ponctuation

La ponctuation va créer du bruit dans les données. Elle ne doit pas être tokenisée sinon elle aura beaucoup de poids dans l'analyse. Attention cependant, quelqu'un qui critique avec plusieurs « !!!!!!! » aura surement un avis très positif ou très négatif. Ces cas ne peuvent pas facilement être gérés avec nos algorithmes.

- Gestion des espaces

Nous ajoutons dans un premier temps un espace supplémentaire après chaque token valide.

Ex : « J'aime ce livre.Il est passionant »
["j", "aime", "ce", "**livreil**", "est", "passionant"]

Nous retirons ensuite tous les espaces inutiles, retours chariots etc…
Nous ne voulons pas que le système différencie "livre_" et "livre".

- Mise sous forme de tokens

Nous transformons notre séquence de mots en un tableau de mots, en coupant la phrase sur chaque espace.

- Retirer les chiffres / numéros

Nous retirons les tokens composés uniquement de chiffres

- Retirer les mots trop courts

Nous retirons tous les mots de 2 lettres ou moins, ils sont trop communs et apportent peu de sens à la phrase.

- Retirer les "stop-words"

Nous retirons les **stop- words**, ce sont les mots tellement commun qu'il est inutile de les garder pour notre analyse. Nous avons une liste de stop-words, appelée « anti-dictionnaire » qui provient de la bibliothèque python *NLTK*, et qui permet de filtrer les stop-words.

- Lemmatisation

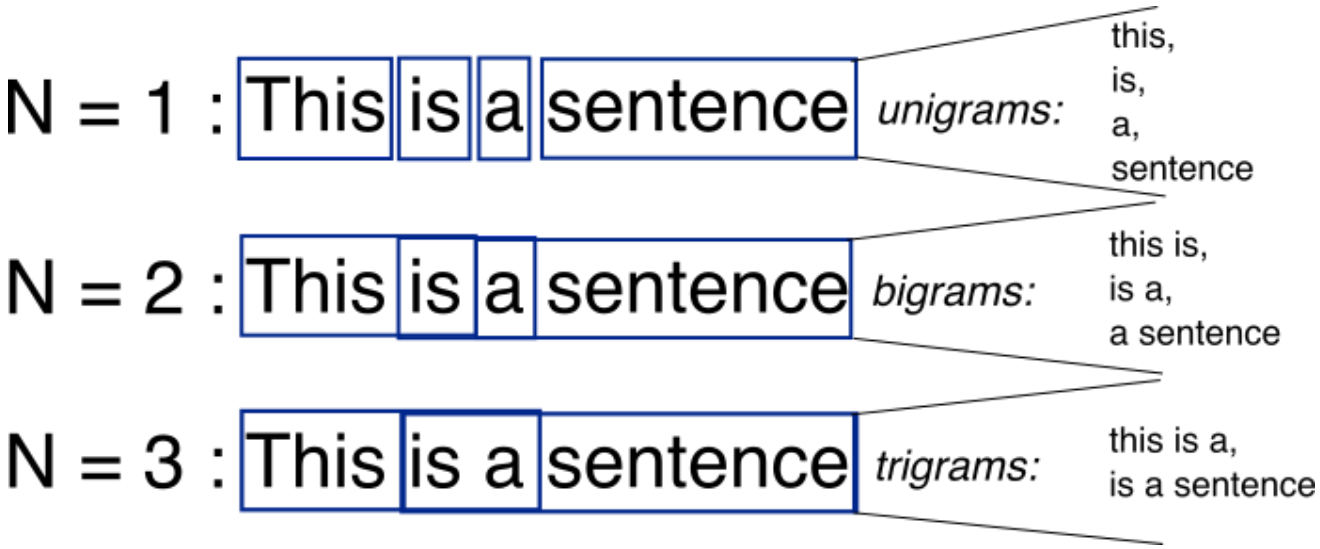
Mise en forme canonique de chacun des tokens restants.

Ex : la forme canonique de « petit », « petite », « petits », « petites » est « petit ».

Cela permet d'apporter plus de poids à plusieurs tokens ayant le même sens.

c. Prétraitement par Ngrams

Un n-gramme est une sous-séquence de n éléments construite à partir d'une séquence donnée.



Cette manière de découper du texte permet de conserver une partie de la structure de langage, et ainsi de garder un minimum de contexte. Par exemple : « il apprécie ce jouet» et « il n'apprécie pas ce jouet ». Ces phrases sont très similaires en termes de fréquence des mots. Seulement, avec des bi-grams nous aurions […, ('n', 'apprecie'),('apprecie', 'pas'), …].

Nous conservons dans ce cas la négation, alors que si nous nous basons sur un token = un mot, le mot « pas » aura un poids moindre : il est très présent dans le lexique.

La tokenisation en ngrams se produit après le prétraitement dont nous venons de parler, afin de ne pas conserver d'informations peu utiles qui rendent le problème plus complexe.

Utilisé dans les correcteurs orthographiques, comme dans les versions précédentes dans la suite Office (http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx)

III. MULTINOMIAL NAIVES BAYES

a. Présentation de l'algorithme

Le classifieur naïf de Bayes appartient à la famille des "classificateurs probabilistes" simples basés sur l'application du théorème de Bayes avec de fortes hypothèses d'indépendance (naïve) entre les caractéristiques. Il fait parti des algorithmes de classification supervisée.

Source : Stuart J. Russell and Peter Norvig. 2003. Artificial Intelligence: A Modern Approach (2 ed.). Pearson Education

Le principe est de calculer la probabilité que chaque critique appartienne à chaque classes cibles.

Ex : la phrase « Je ne recommande pas ce produit. »

La probabilité que la phrase appartienne à une catégorie x est donnée par :

P(« je ne recommande pas ce produit » | note=x) x P(note=x)

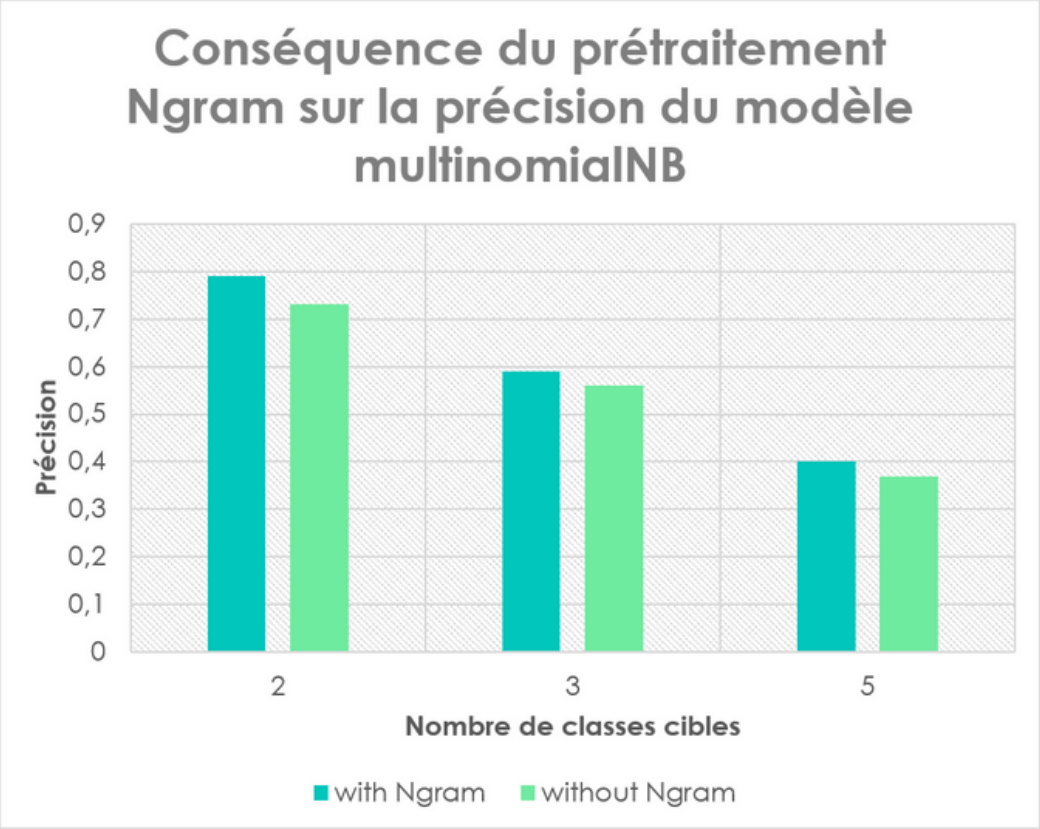
Nous devrions diviser cette probabilité par P(« je ne recommande pas ce produit ») mais comme nous allons comparer les probabilités que la phrase appartienne à chacune des critiques, nous pouvons simplifier la formule.

De plus, nous ne passons pas la critique brute dans le classifier mais bien des tokens ou ngrams qui ont fait l'objet d'un traitement afin d'améliorer les performances et le temps d'exécution.

Pour donner un ordre d'idée nous avons 37 757 mots uniques sur les 50 000 critiques avec lesquelles nous travaillons.

b. Prétraitement avec Ngram

Nous avons lancé l'algorithme avec et sans le prétraitement par Ngrams, nous avons pu constater que peu importe le nombre de classes cibles, nous avions toujours environ 3 points de performance en plus quand Ngrams était utilisé.



c. Librairie NLTK

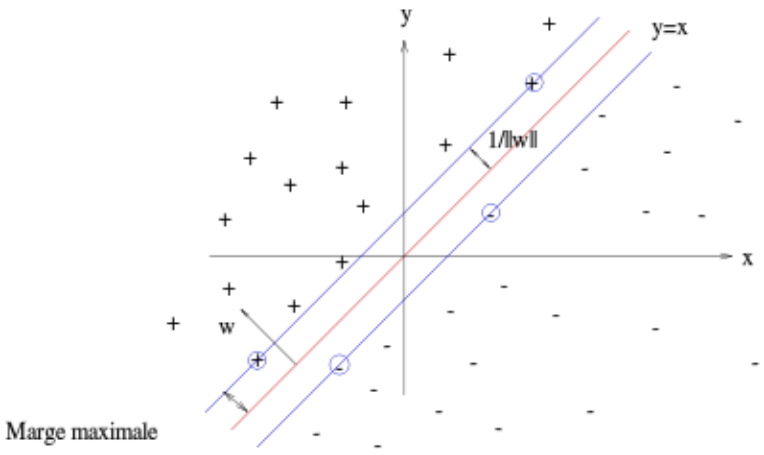
Nous avons utilisé la librairie Python **NLTK** (Natural Language Toolkit). Elle fournit des ressources, notamment une cinquantaine de corpus et ressources lexicales tels que WordNet, mais également des outils de traitement de texte pour la classification, la tokenisation, la lemmatisation, le balisage, l'analyse ou encore le raisonnement sémantique.
Dans nos programmes nous avons utilisé ses fonctionnalités comme son classifieur naïf bayésien, son anti-dictionnaire de « stop-words », sa fonction de tokenisation, sa fonction « everygram » qui décompose les phrases en dictionnaires de ngram.

IV. SUPPORT VECTOR MACHINE

a. Présentation de l'algorithme

Les SVM sont des classificateurs supervisés basés sur deux idées clés, qui traitent des problèmes de discrimination non linéaire et reformulent le problème de classification.

La première idée clé est le concept de marge maximale. La marge est la distance entre la frontière et les échantillons les plus proches. Ce sont les vecteurs supports. Dans le SVM, la limite de séparation est choisie de manière à maximiser la marge à partir d'un ensemble d'apprentissage.



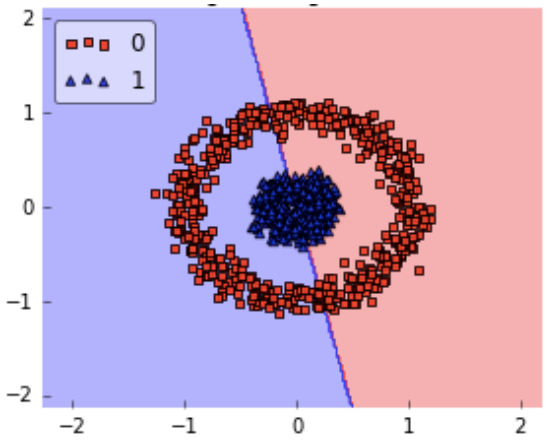
La deuxième idée des SVM est de transformer l'espace de données de représentation en un espace plus grand, où une séparation linéaire est impossible. C'est rendu possible par une fonction du noyau, qui n'exige pas de connaissance explicite de la transformation à appliquer pour le changement d'espace. Les fonctions du noyau permettent de transformer un produit scalaire dans un espace de dimensions supérieures, ce qui est coûteux, en une simple évaluation d'une fonction. Cette technique est connue sous le nom d'astuce du noyau (= kernel trick).

b. Astuces du noyau

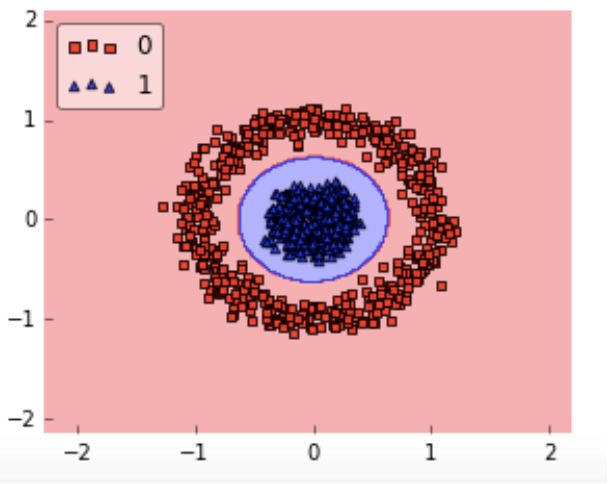
« La classification par SVM peut utiliser plusieurs classifieurs linéaires, appelés « kernel trick » ou astuce du noyau, qui permettent de transformer l'espace de représentation des données d'entrées en un espace de plus grande dimension, où un classifieur linéaire peut être utilisé. »

Source : M. Aizerman, E. Braverman et L. Rozonoer, « Theoretical foundations of the potential function method in pattern recognition learning », Automation and Remote Control, vol. 25, 1964, p. 821-837.

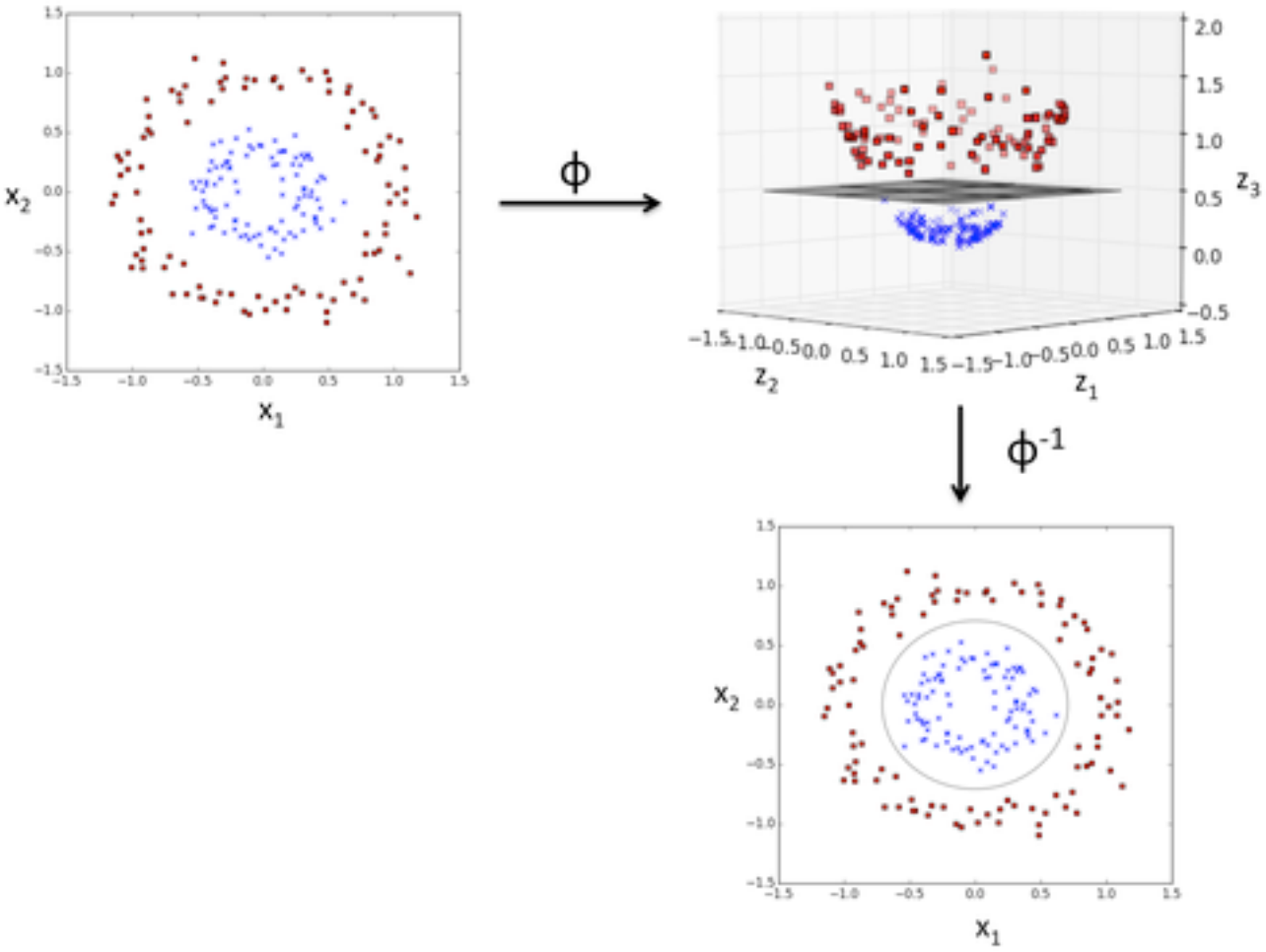
Kernel linéaire



Kernel RBF

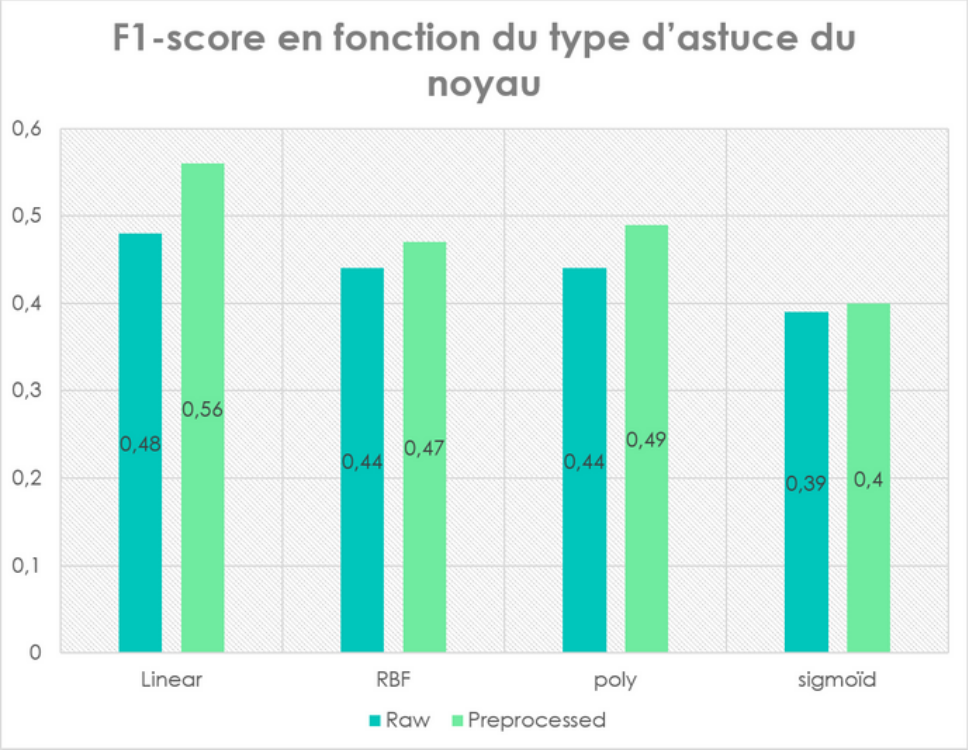


La région de décision SVM du noyau RBF est aussi une région de décision linéaire. En réalité, le noyau RBF créé des combinaisons non linéaires des variables pour élever nos échantillons dans un espace de plus grande dimension où nous pouvons utiliser une limite de décision linéaire pour séparer nos classes cibles.



c. Choix du type d'actuce de noyau

Il n'existe malheureusement aucune méthode pour connaitre automatiquement l'astuce du noyau qui correspond le mieux à notre problème. C'est pourquoi nous avons comparé les performances de chacune d'entres elles.



Sur ce graphique, nous pouvons clairement déduire que :

- Les données prétraitées offrent de meilleurs résultats (amélioration de 3 points environ).
- Le kernel linéaire a de meilleurs résultats (0.53 au f1-score avec le kernel linéaire/ données prétraitées contre 0.48 avec le kernel RBF/données prétraitées).

Suite à cette expérimentation, nous avons utilisé un kernel linéaire au sein de notre classifieur SVM afin de maximiser nos performances.

d. Représentation tfidf

Nous avons choisi la méthode de pondération tfidf comme variable de notre classifier SVM.
La fréquence (tfi,j) d'un terme est simplement le nombre d'occurrences de ce terme dans le texte considéré. La fréquence inverse de document (idfi) est une mesure de l'importance d'un terme dans un texte. Dans le schéma TF-IDF, elle vise à donner un poids plus important aux termes les moins fréquents, qui sont alors considérés comme plus discriminants. Elle consiste à calculer le logarithme de l'inverse de la proportion de documents du corpus qui contiennent le terme.
Ces deux mesures forment tfidf :

$$tfidf_{i,j} = tf_{i,j} * idf_i$$

Nous obtenons cette représentation du texte grâce à la librairie python sklearn, tout comme l'implémentation de l'algorithme SVM ainsi que la fonction de séparation du jeu de données en jeu d'entrainement et de test « train_test_split ».

V. RÉSULTATS

a. Comparaison des algorithmes

L'algorithme Multinomial Naives Bayes traite les variables comme étant indépendantes alors que SVM est capable de prendre en compte les corrélations inter-variables jusqu'à un certain degré.

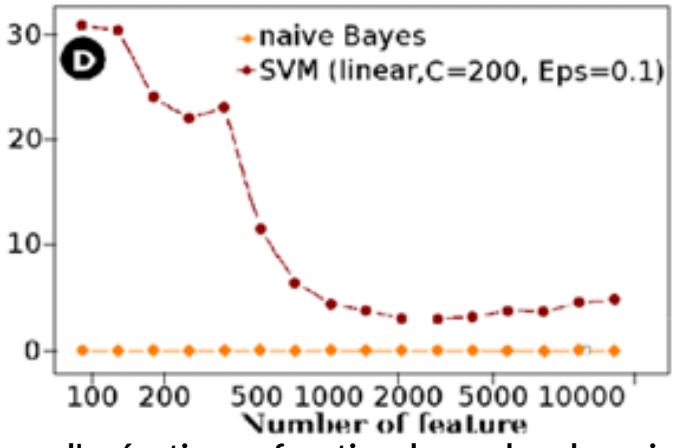
Naives Bayes est considéré comme un outil probabiliste alors que SVM est plus considéré comme étant géométrique.

NB est généralement plus rapide, sa complexité est de :

O(NB) = O(Nd) avec d = dimensions des variables

SVM a une dimension de :

O(SVM) = O(NR) avec R = nombre d'itération

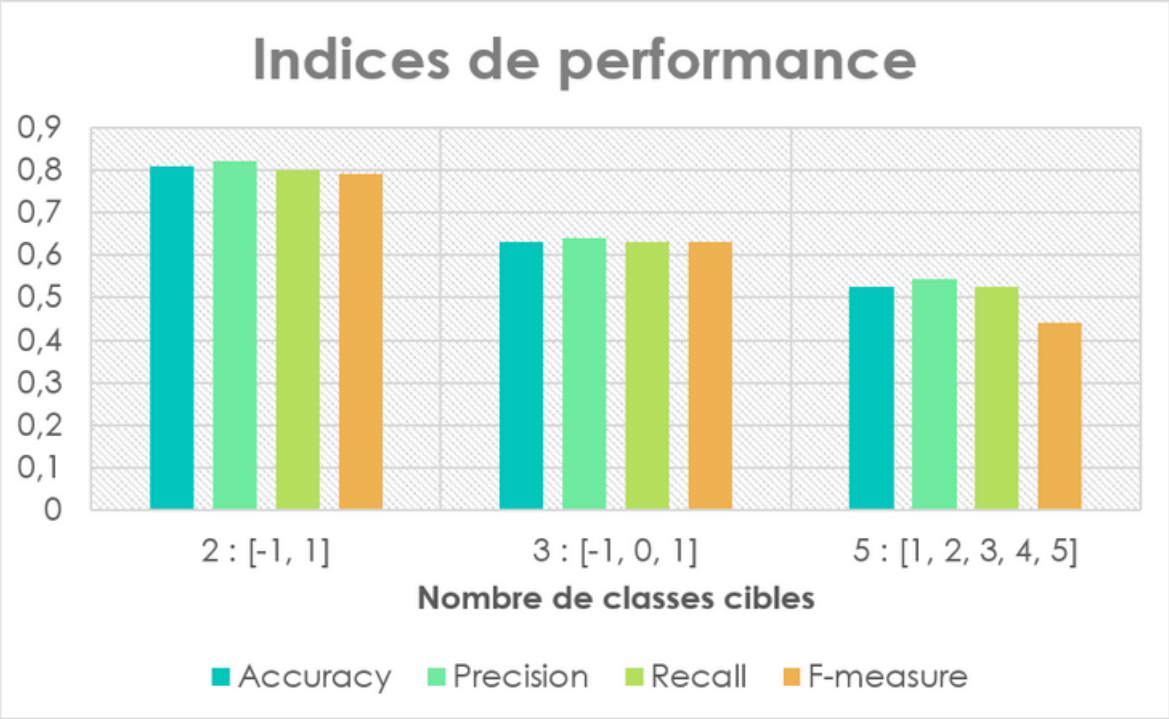


Temps d'exécution en fonction du nombre de variables

Source : Colas, Fabrice & Brazdil, Pavel. (2006). Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks. International Federation for Information Processing Digital Library; Artificial Intelligence in Theory and Practice;. 217. 10.1007/978-0-387-34747-9_18.

b. Résultats en fonction du nombre de classes

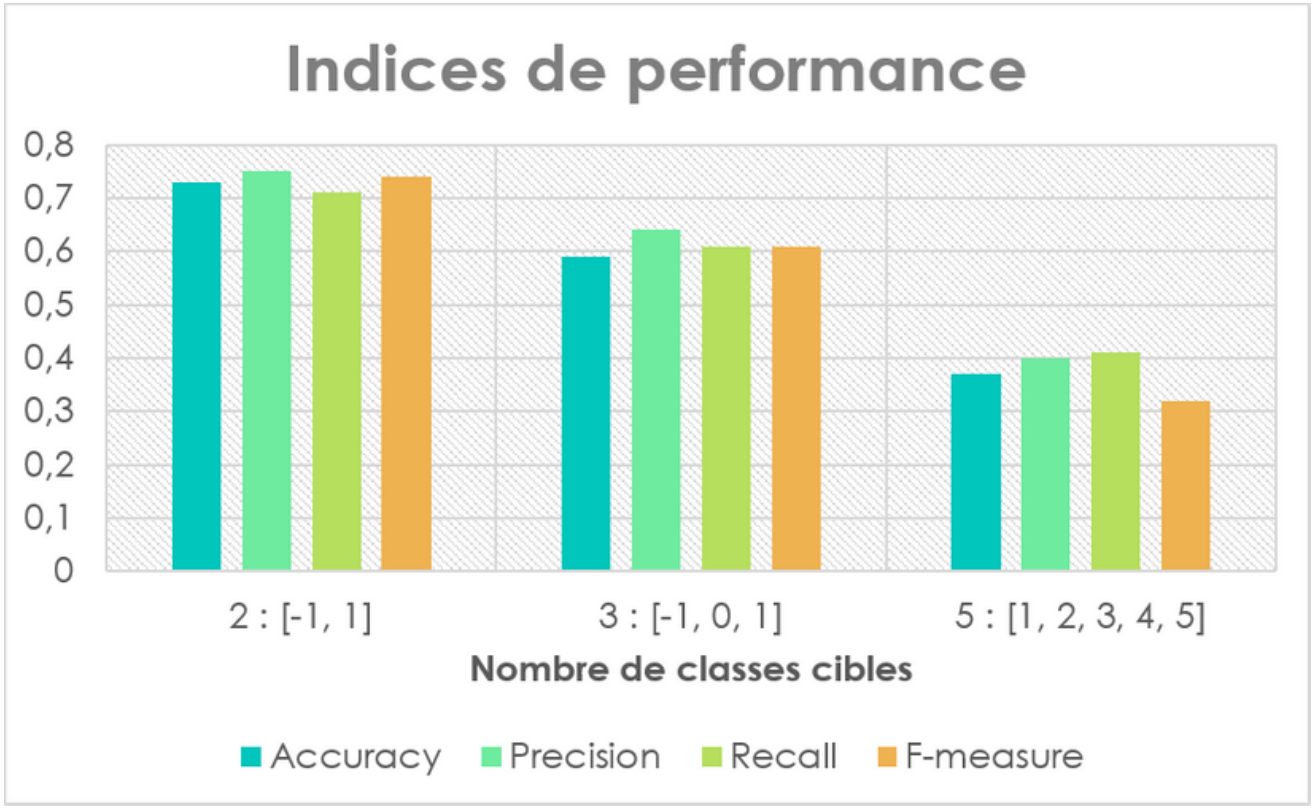
Résultats obtenus avec l'algorithme SVM – tfidf en fonction du nombre de classes cibles.



Nous obtenons 52% de précision sur le classement de notes allant de 1 à 5. Ce résultat moyen peut être expliqué par le fort chevauchement des classes cibles notamment les classes 5 (très positif) et 4 (positif) et les classes 1 (très négatif) et 2 (négatif).

En simplifiant le problème à seulement deux classes, positif et négatif, nous obtenons 81% de précision.

Performances obtenues avec le classifieur multinomial Naives Bayes, le prétraitement expliqué dans le chapitre II (dont Ngram).



Performances obtenues avec le classifieur multinomial Naives Bayes, le prétraitement expliqué dans le chapitre II (dont Ngram). Nous avons une précision de 0.74 avec deux classes cibles, et une précision de 0.35 sur le problème à 5 classes cibles.

c. Conclusion

Dans notre étude, SVM permet d'obtenir de meilleurs résultats que Naives Bayes, notamment dans le problème à 5 classes cibles (35% pour NB contre 50% pour SVM). Nos résultats expérimentaux sont en adéquation avec l'avis de la majorité des experts en machine learning, qui pensent que SVM offre de meilleurs résultats.

De plus, le prétraitement par ngrams apporte une légère augmentation des performances (environ 3 points), mais ne semble pas réellement adaptée à notre problème. Les ngrams permettent de forts gains de performance dans la reconnaissance de langage ou la correction orthographique.

Pour conclure sur la comparaison de ces algorithmes, les performances dépendent fortement du jeu de données utilisé. NB est plus performant quand le nombre d'entrées d'entrainement est limité (jusqu'à 50 entrées environ).

Source : Wang, Sida, and Christopher D. Manning. "Baselines and bigrams: Simple, good sentiment and topic classification." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2. Association for Computational Linguistics, 2012

NB performe mieux dans la classification de texte courts quand dans la classification de textes plus longs.

Source : NG, Andrew Y. et JORDAN, Michael I. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. Advances in neural information processing systems, 2002, p. 841-848.

L'intégralité du code est disponible à l'adresse :
https://github.com/stressGC/Text-Mining-MultinomialNB-SVM