Data mining - Projet Final

Amazon reviews

Rodéric Bouesse, Frédéric Sallin

PREPROCESSING

Avant de commencer le preprocessing, nous avons nettoyé les fichiers contenant les reviews car il y avait du code HTML dans ces fichiers. Si nous l'avions laissé, nous aurions pu avoir des résultats assez faibles.

Le preprocessing est fait en plusieurs étapes:

- Suppression de la ponctuation
- Suppression des "stop words"
- Application de l'algorithme de stemming

Suppression de la ponctuation

Nous utilisons une librairie de Python pour remplacer chaque ponctuation par un espace. Cette librairie s'appelle ***re*** . La fonction de suppression de ponctuation lit chaque review et applique à cette dernière une "soustraction" avec une expression régulière prenant en compte que des lettres entre [a -zA-Z], ce qui aura pour but de supprimer tous les caractères ",", ".", "!", etc.

Suppression des "stop words"

Nous utilisons le principe de tokenization pour chaque review, c'est à dire que nous décomposons chaque review par l'ensemble de mots qui la compose.

Ex:

review: "Ce produit est super"

Application de la tokenization: "Ce", "produit", "est", "super".

Le séparateur utilisé pour "tokenizer" chaque review est donc le caractère espace.

Maintenant que nous avons l'ensemble des mots, nous comparons l'ensemble des mots par les stops words. Si il y a des mots "stop words" dans l'ensemble des mots de la review, on supprime ces mots de la review et on reconstruit une phrase avec les mots restants.

Application de l'algorithme de Stemming

Pour le stemming, nous utilisons une librairie de Python. Elle s'appelle ***Gensim***.

Le stemming est une technique de transformation de mots en leur racine grâce à un ensemble de règles prenant en compte la longueur et au préfixe du mot.

Chaque review est "tokenizé" et pour chaque mot de la review nous appliquons l'algorithme de stemming.

Par exemple, "like" devient "lik".

Nous reconstruisons la review avec les mots stemmés.

MODÈLES

Nous avons décidé de concaténer toutes les reviews ensemble quelque soit leur document puis nous avons construit les matrices X et Z.

La matrice X est de la taille (n x d) où n représente le nombre de reviews et d le nombre de mots uniques de l'ensemble des reviews.
Nous regardons pour chaque review, l'ensemble des mots qui la compose puis nous mettons le nombre de fois que chaque mot apparaît à l'indice réservé à ce mot.

La matrice Z est de la taille (n x 200) où n représente le nombre de reviews et 200 le nombre d'éléments de chaque vecteur du fichier word2vec.txt
Pour construire cette matrice, nous avons fait un dictionnaire qui permet à partir d'une clé (mot) d'avoir son vecteur associé, puis pour chaque ligne de la matrice Z nous faisons la moyenne pondérée des vecteurs de chaque mot de la review avec leur nombre d'occurrence.

Dans le fichier word2vec.txt, les vecteurs associés aux mots représentent leur sémantique, c'est donc pour cela que nous avons décidé de faire une moyenne pondérée car elle nous donne une sémantique de la phrase (connotation positive ou négative).

Nous avons utilisé 3 modèles:

- Naive Bayes (Multinomial): C'est un algorithme d'apprentissage supervisé qui se base sur le théorème de Bayes avec l'hypothèse naïve que les features sont indépendantes deux à deux
  Nous avons choisi le modèle Multinomial car il est plus adapté à la recherche de texte, et donc aux bags of words, que le modèle Gaussien. En effet, avec ce modèle, les instances représentent les fréquences selon lesquelles un événement a été généré par une loi multinomiale (p_1,..., p_n), p_i, est la probabilité que l'événement i se passe. Ainsi, une instance est en fait, dans notre cas, un histogramme des apparitions des différents mots.
  L'avantage de Naive Bayes est le faible nombre de données nécessaires à l'apprentissage.
  Nous avons utilisé la librairie **scikit learn** pour le modèle Multinomial puis nous avons séparé le bag of words et la liste de notes en 2 ensembles (66% pour l'apprentissage et 34% pour les tests). La fonction **fit** permet d'entraîner

le modèle et la fonction **predict** permet d'utiliser le modèle entraîné sur les données de tests.

- Réseaux neuronaux
  Les réseaux neuronaux permettent d'effectuer efficacement des opérations de classification. Nous utilisons une catégorie de réseau neuronal appelée multi-layer perceptron. Cet algorithme utilise l'apprentissage supervisé et apprend une fonction
  non linéaire $f : R^m \rightarrow R^o$ sur un ensemble de données.
  Le réseau prend en entrée un ensemble de features $X = (x_1, x_2, x_3, ..., x_m)$ ainsi qu'un résultat, ou label y .

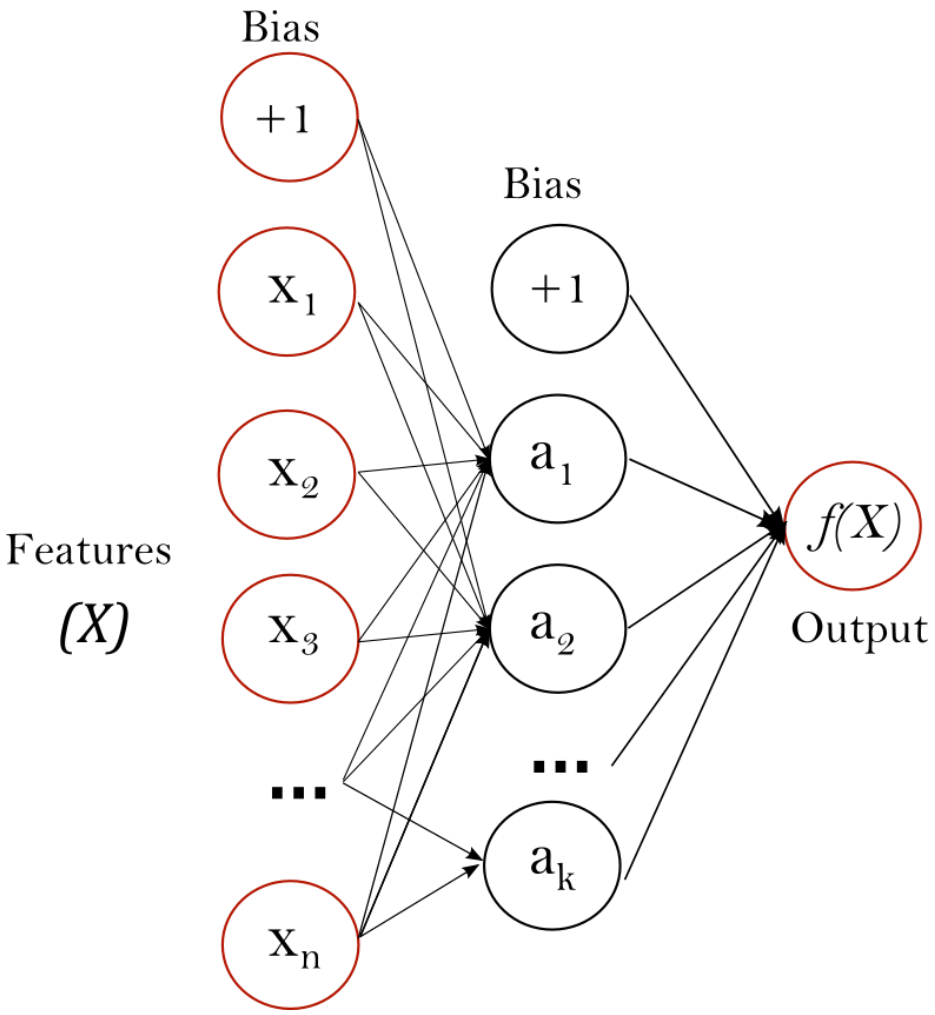
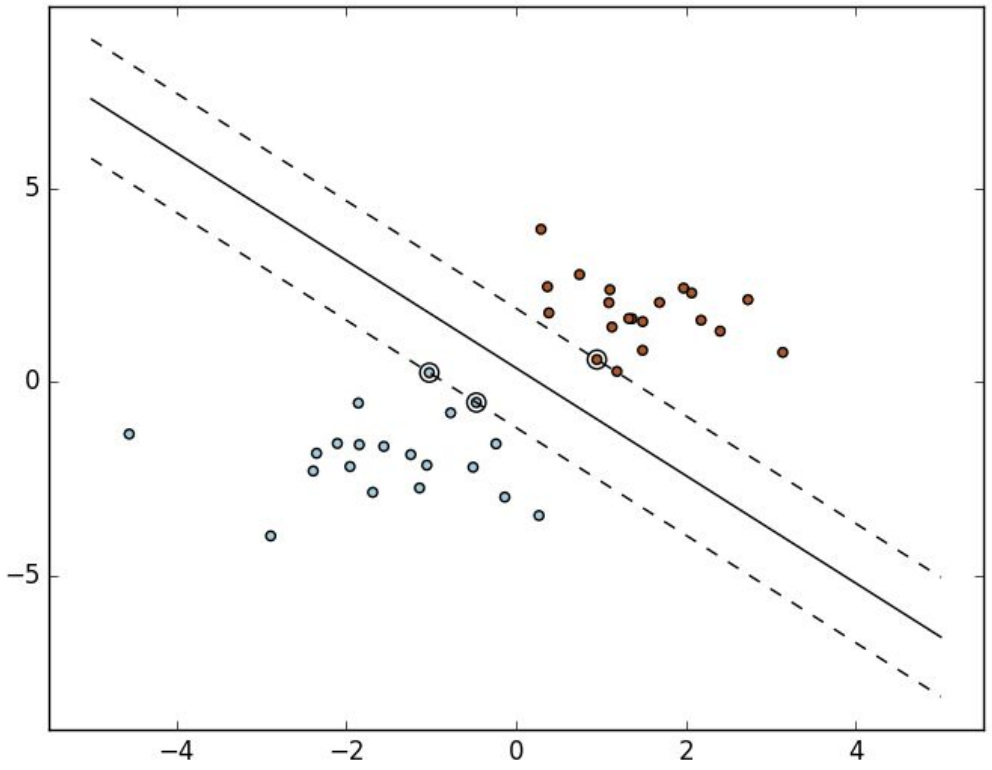


Schéma de réseau neuronal -
http://scikit-learn.org/stable/_images/multilayerperceptron_network.png

Dans notre travail, les features X sont soit la représentation bag-of-words de la review, c'est-à-dire chaque features est le nombre d'occurence d'un mot spécifique dans la review, soit la représentation vectorielle de la review. Le résultat (label) y est la note de l'utilisateur, c'est-à-dire un nombre entier entre 1 et 5.
Un réseau contient plusieurs couches cachées de neurones. Chaque neurone

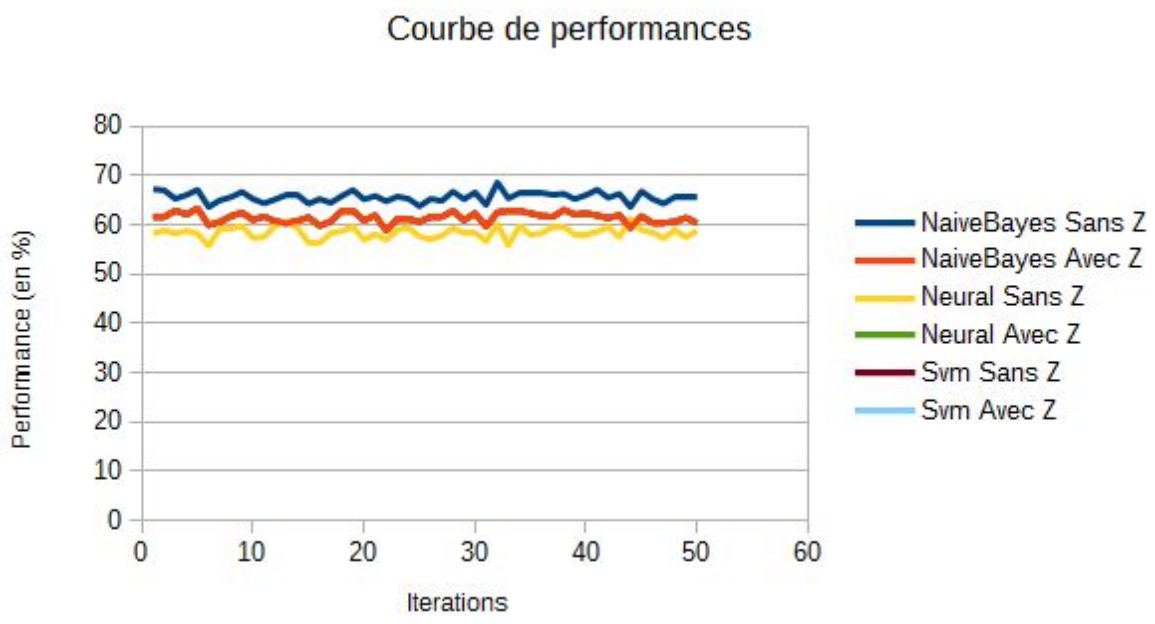
sert à calculer une nouvelle valeur en utilisant celle de la couche précédente. Le neurone fait une somme pondérée linéaire $w_1x_1 + w_2x_2 + w_3x_3 + ... + w_mx_m$.
Nous utilisons la descente de gradient stochastique, qui est mieux adaptées pour des ensembles de données de plusieurs milliers d'instances.
Idem, nous entraînons le réseau sur 66% de l'ensemble de données et nous le testons sur les 34% restants.

- SVM
  SVM (Support Vector Machine) apprend principalement une fonction linéaire de la forme $w^T x + b = 0$
  Pour cela, il construit un hyper-plan ou un ensemble d'hyper-plans dans un espace de beaucoup de dimensions. L'hyper-plan qui a la plus grande distance par rapport au plus proche point de chaque classe est choisi.
  Même si une SVM fonctionne d'abord pour des problèmes de classification binaire, on peut l'étendre à des problèmes multi-classes, comme c'est le cas avec notre travail. Nous avons choisi une implémentation de l'approche "one-against-one". C'est-à-dire que l'algorithme construit et entraîne $nb_{classes} * (nb_{classes} - 1)/2$ classifieurs qui comparent deux à deux toutes les classes. L'algorithme agrège les résultats de tous les classifiers pour séparer les classes.
  La SVM est utile lorsque le problème demande de travailler dans des espaces de grandes dimensions.
  C'est le cas dans le projet car chaque ligne du bag of words est un vecteur de plusieurs milliers de composantes.



Exemple de séparation calculée grâce à svm -
http://scikit-learn.org/stable/_images/sphx_glr_plot_separating_hyperplane_0011.png

RÉSULTATS

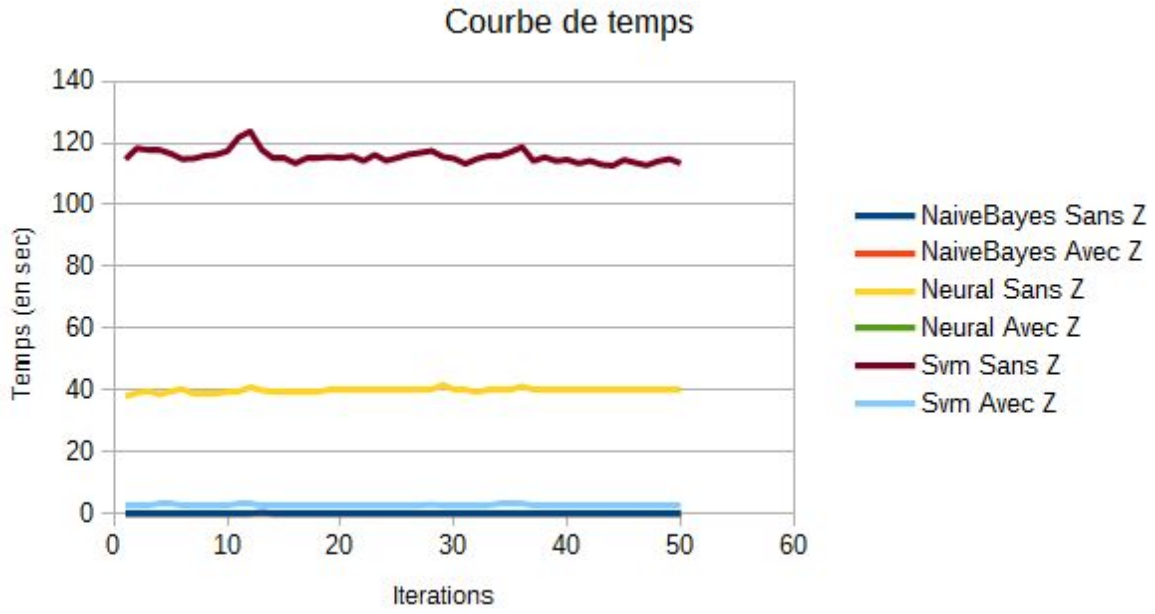


Pourcentage de précision de chaque algorithme

On remarque que nos modèles ont une efficacité d'environ 60%, ce qui est assez acceptable.
Le modèle qui a la plus grande performance est le Naive Bayes sans l'utilisation de la matrice Z et donc de la sémantique des mots.

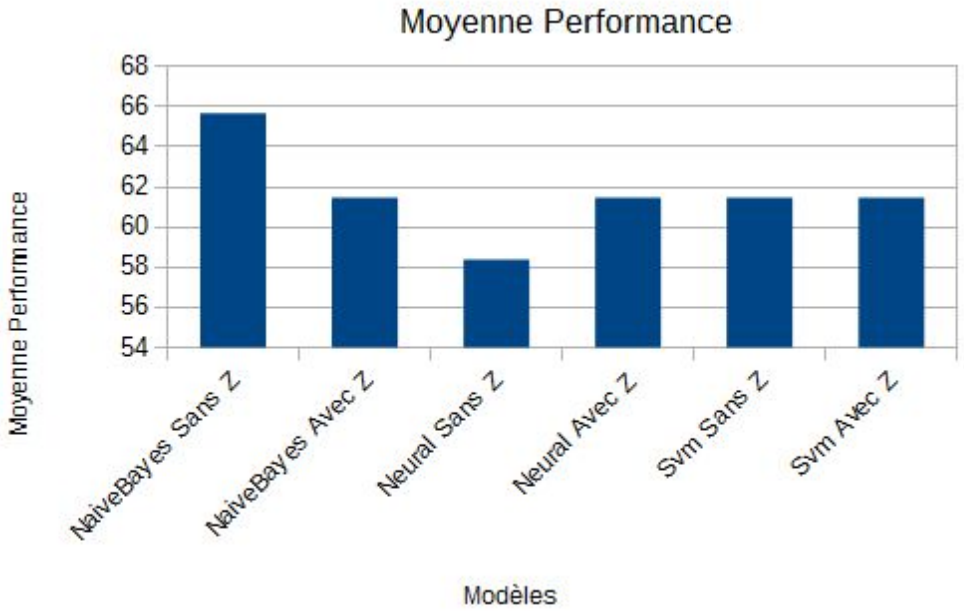
Les réseaux neuronaux ont des performances moins élevées, surement due à un taux d'apprentissage pas optimal.

On remarque aussi que les modèles qui utilisent la matrice Z n'ont pas de très bons résultats. Un explication de ces mauvais résultats est notre utilisation du stemming, lorsque nous construisons la matrice Z, nous ne prenons pas en compte les mots stemmés qui n'ont pas une référence dans la liste de mots uniques. Nous aurions du faire ou utiliser une métrique capable de trouver le mot du Word2Vec le plus proche du mot stemmé ainsi cela permet de prendre un vecteur proche.

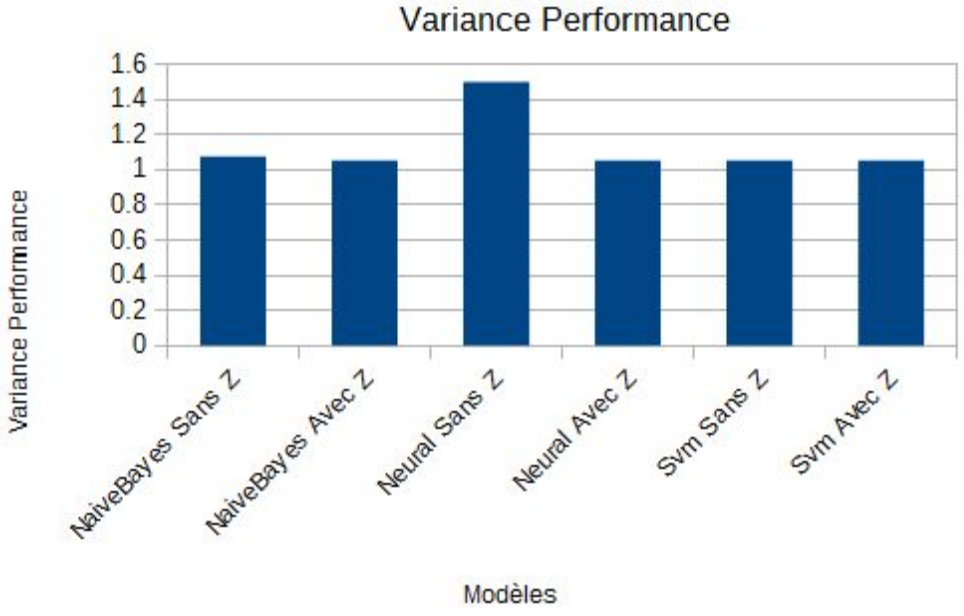


Temps d'exécution de chaque algorithme

Certains algorithmes ont des temps très petits (voir le fichier Resultats.csv), ce sont principalement ceux utilisant la matrice Z. Les modèle SVM et Neural Network n'utilisant pas la matrice Z prennent beaucoup de temps comparés aux autres modèles et ont des temps d'apprentissage plus faibles. Il faut toutefois noter que dans les cas d'utilisation de la matrice Z, nous ne comptons pas le temps d'initialisation et de construction de cette matrice, qui est grand (plus de 5 minutes). De plus, cette matrice Z est de dimension vraiment plus petite que la matrice X contenant la représentation bag of words de la review. La matrice Z ne contient que 200 features, contrairement aux plusieurs milliers de mots de la matrice X. Étant donné que les algorithmes sont sensibles au changement d'échelles, c'est donc pour cela que les algorithmes utilisant Z sont plus rapides (en temps d'apprentissage seulement).



Moyenne du pourcentage de précision de chaque algorithme



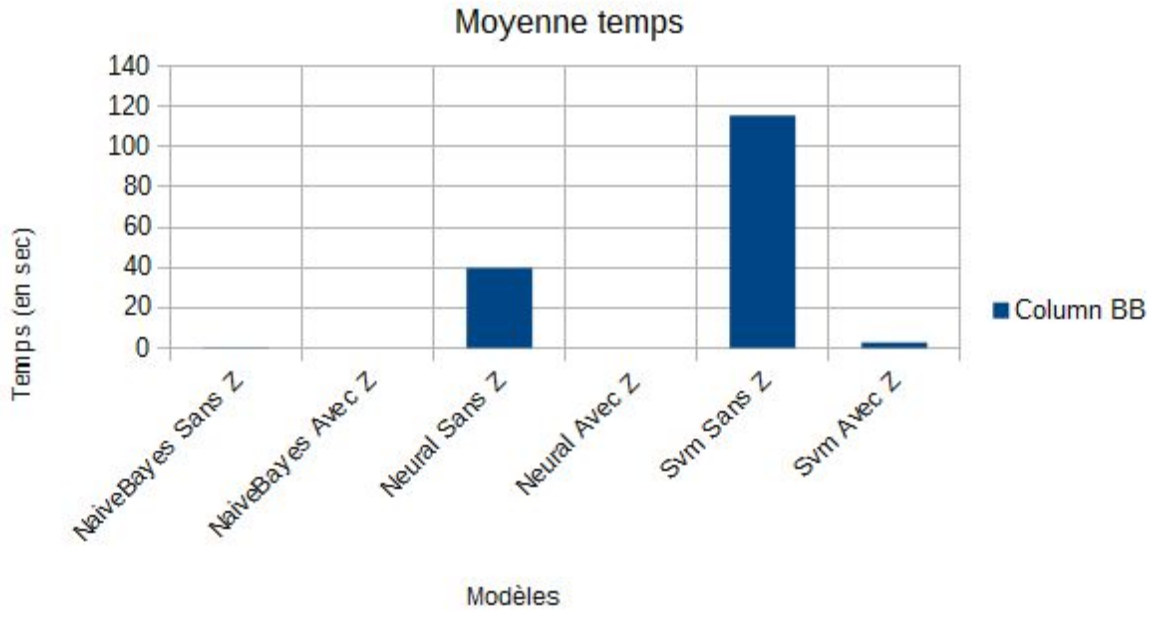
Variance du pourcentage de précision de chaque algorithme

Les 2 graphiques ci-dessus reprennent les infos du premier graphique (graphique de performance). La moyenne et la variance des performances des modèles sont de bonnes mesures de leur stabilité.

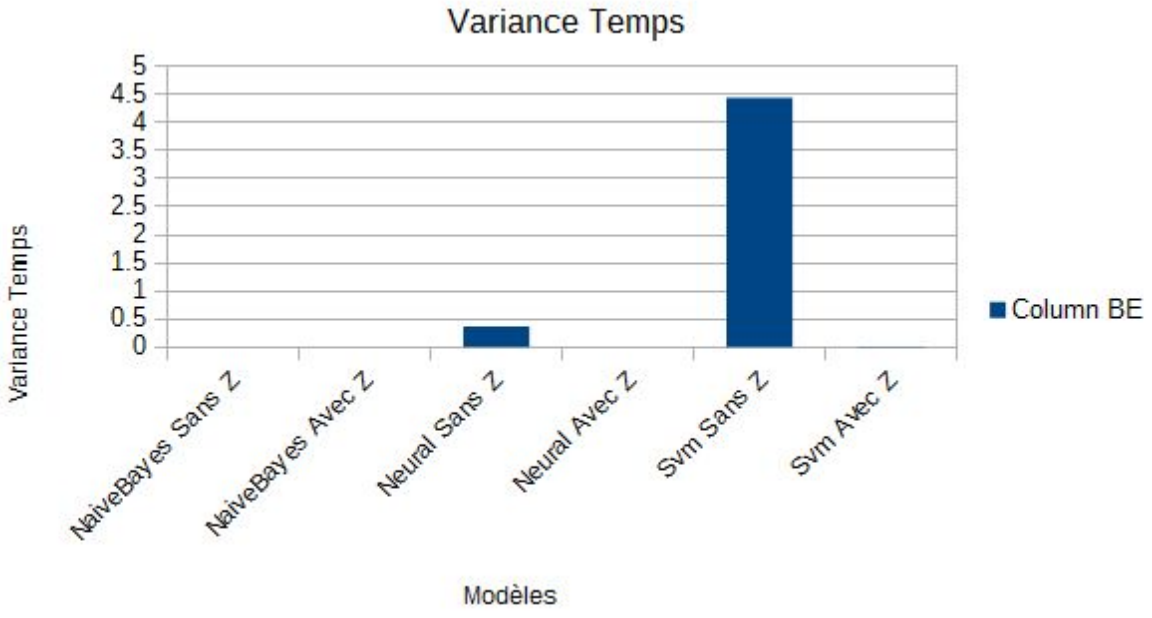
La variance sur la performance des algorithmes reste proche de 1. La différence la plus notable est pour le réseau neuronal sans Z qui totalise une variance de 1.5. Étant donné qu'à chaque itération il est réentraîné, le nombre de mots, qui est le nombre de features, change, et donc les paramètres, fixes, du réseau peuvent moins bien convenir et influencer sur la performance du réseau sur l'itération en cours.

En regardant les graphes de performances, on peut voir que le modèle Neural Network sans Z est le moins bon algorithme, il ne franchit pas la barre des 60% de précision .
Le modèle offrant la meilleure performance est le modèle Naive Bayes sans Z.

Une chose importante que l'on peut remarquer à travers les valeurs dans le fichier Resultat.csv est que 4 modèles ont des valeurs identiques. Nous supposons que les modèles sont assez bien équilibrés et sont cohérents entre eux.



Moyenne du temps d'exécution de chaque algorithme



Variance du temps d'exécution de chaque algorithme

Les modèles Neural Network et SVM sans Z sont sans conteste moins rapide que les autres, en raison de la taille des entrées comme expliqué précédemment. En regardant ces graphiques avec les graphiques de performance, on peut voir que Naive Bayes Sans Z donne une performance élevée et un temps réduit, ce qui implique qu'il est notre meilleur classifieur.

Comparaisons des performances des modèles avec ou sans matrice Z:

Pour Naive Bayes, le meilleur modèle est celui n'utilisant pas la matrice Z.
Une des explications possibles sur l'infériorité de la performance du modèle n'utilisant pas la matrice Z est que ce modèle n'est pas adapté au word2vec car la matrice Z contient des valeurs continues et non discrètes, or le Naive Bayes multinomial utilise des valeurs discrètes. De plus, la matrice Z contient des valeurs négatives (provenant de word2vec), ce qui pose un problème car le modèle Naive Bayes multinomial n'est pas compatibles avec des valeurs négatives. (Pour faire fonctionner le modèle Naive Bayes multinomial sur la matrice Z, nous avons dû normaliser les valeurs des vecteurs entre 0 et 1).

Pour SVM, le deux modèles sont équivalents, sauf par rapport aux temps d'apprentissage.

Pour les réseaux neuronaux, le meilleur modèle est celui utilisant la matrice Z. Les réseaux utilisent beaucoup de paramètres: taux d'apprentissage, la fonction d'activation, de la fonction d'optimisation des poids, etc. Il est difficile de trouver la combinaison la plus efficace de tous ces paramètres afin d'obtenir ensuite un score de performance plus élevé. Une possible explication est la forme des données d'entrée (valeurs discrètes, valeurs continues)