Projet Fouille du texte

Classification des commentaires en chinois sur *Le voleur d'ombres* de Marc Lévy
Etudiant: Pan Wenzhu, Paris Nanterre
27 avril 2019

Introduction:

L'objectif du projet est de tester des algorithmes de classification sur un corpus chinois constitué de commentaires courts sur un livre de Marc Lévy à l'aide de Weka, et éventuellement d'entraîner un modèle de classification pour classer automatiquement le reste des commentaires.

Présentation du corpus:

Les commentaires sont crawlés du site web www.douban.com, similaire au babelio.fr où les lecteurs partagent leur avis sur un livre. Pour avoir une quantité plus importante d'échantillons, nous avons choisi de ne traiter que les commentaires courts. Tous les commentaires sur le livre *Le voleur d'ombre*, jusqu'au 30 mars 2019, ont été crawlés avec un script Python (ci-joint dans le mail). Le corpus est donc constitué de 25 590 commentaires, de 1 003 496 caractères, soit en moyenne 40 caractères par commentaire. Au début, nous avons voulu faire une classification binaire, c'est-à-dire, commentaire qui dit bon et celui qui dit mauvais. En lisant et classant manuellement des commentaires, nous nous sommes rendus compte de la nécessité d'une troisième catégorie, « moyen ». Ainsi, nous avons une classification en trois catégories, « bon », « mauvais » et « moyen ».

Les commentaires « bon » sont faciles à classer, souvent dotés d'une émotion forte:
Le premier amour touchant. L'amitié. La famille.
Roman guérissant.
Longtemps que j'ai pas pleuré à chaudes larmes par un livre littéraire. Pas lu l'originaire, mais la traduction est très jolie.
plein d'émotion douce, la sincérité et l'imagination.
Franchement, au début je croyais que l'enfant malice a des illusions, parce que ça n'a pas l'air d'une histoire magique (j'ai failli donner une mauvais critique si ça continue), mais en fait, c'est une histoire de l'amour qui grandit, de l'amitié précieuse, et de l'amour famille avec beaucoup d'impuissance et beaucoup de bienveillance.

Les commentaires classés « mauvais » sont souvent courts, en deux ou trois petits mots. C'est sans doute normal, un livre mauvais n'inspire pas beaucoup d'envie de commenter. Egalement, les commentaires « mauvais » sont très saillants:
Au-dessous de trois étoiles.
Pas trop belle-lettre, pas très touchant. Les français sont vraiment incompréhensibles.
Ne pers pas de temps avec ça.
Le héros me dégoûte.
Un livre pourri.

Pas terrible.
Qu'est-ce que c'est ce truc.

Les commentaires « moyen » sont entre les deux, et c'est toujours un peu délicat à classer:
J'aurais aimé ce livre il y a dix ans.
Trois étoiles sont peu, quatre étoiles sont trop. La traduction peut faire mieux. Sais pas si des collégiens puissent comprendre qqch?
Je croyais un livre d'enfant.
Hein……
C'est un livre bien ordinaire ! Pourquoi il est si connu ? C'est qu'une histoire d'apprentissage de la maturité.
C'est devenu bien à partir de 87%.
Comme ci comme ça, incapable de comprendre l'émotion.

En fait, une quatrième catégorie se relève sans doute nécessaire pour des commentaires nuls qui ne donne pas d'opinion, comme des citations ou des dates de lecture. Heureusement, ce genre de commentaires sont bien minoritaires, et nous pouvons les supprimer.

Après plusieurs centaines de classement manuel (placer le fichier dans le dossier correspondant), nous avons vite abandonné cette méthode et nous avons écrit un petit script en Python pour faciliter le travail. Pour connaître l'impact de la quantité de donné sur le résultat de l'algorithme, nous avons constitués des corpus de plusieurs gammes, c'est-à-dire, nous commençons avec un train data de 500 commentaires, obtenir un fichier arff avec ce train data, ensuite en ajouter 500 commentaires et obtenir un deuxième fichier arff, ainsi de suite. Le tableau ci-dessous montre le nombre d'instances, d'attributs et le nombre d'échantillons dans chaque catégorie. Nous sommes arrivés à la fin avec un train data de 2313 commentaires. Et avec ce dernier train data, nous avons comparé le mode vecteur à base du nombre d'occurence du mot et à base boolean, sans mots-vides ou avec mot-vides. Et nous avons lancé l'algorithme systématiquement avec test option, « test options: cross-validation Folds 10 ».

Il est sûr que la réparation d'échantillons dans chaque classe n'est pas très équitable, cela à cause de fait que les commentaires « bon » sont la majorité. Ainsi, dans une dernière expérience, nous l'avons fait avec autant d'échantillons dans chaque classe. Nous en détaillons dans la partie Expérience II.

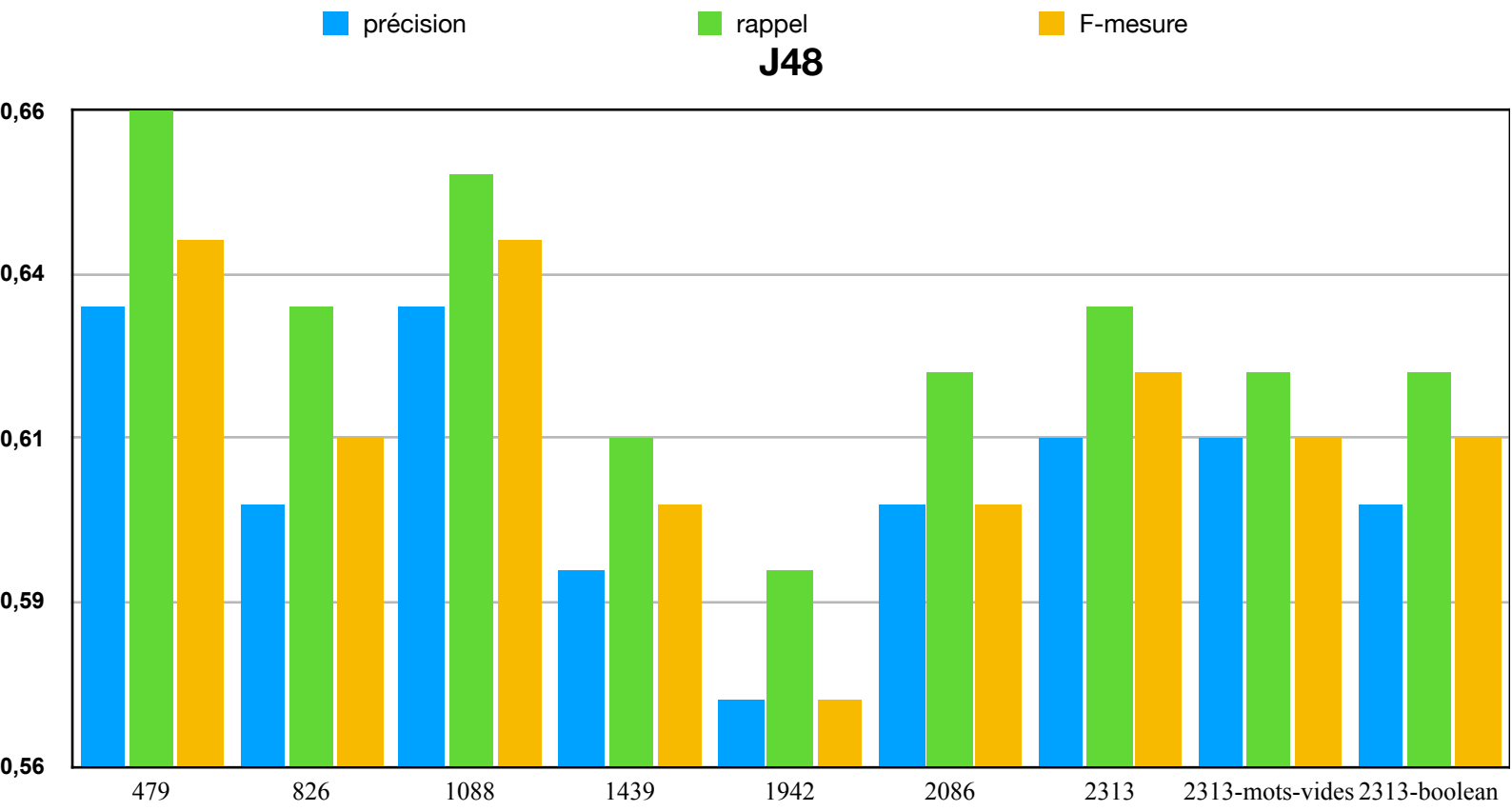
Vectorisation:

instances	attributs	bon	moyen	mauvais
479	2927	306	111	62
826	4100	503	197	126
1088	4741	687	233	168
1439	5576	831	360	248
1942	6326	964	604	375
2086	6569	1034	644	408
2313	6810	1142	720	451
2313 —mots-vides	6785	1142	720	451
2313 —boolean	6810	1142	720	451

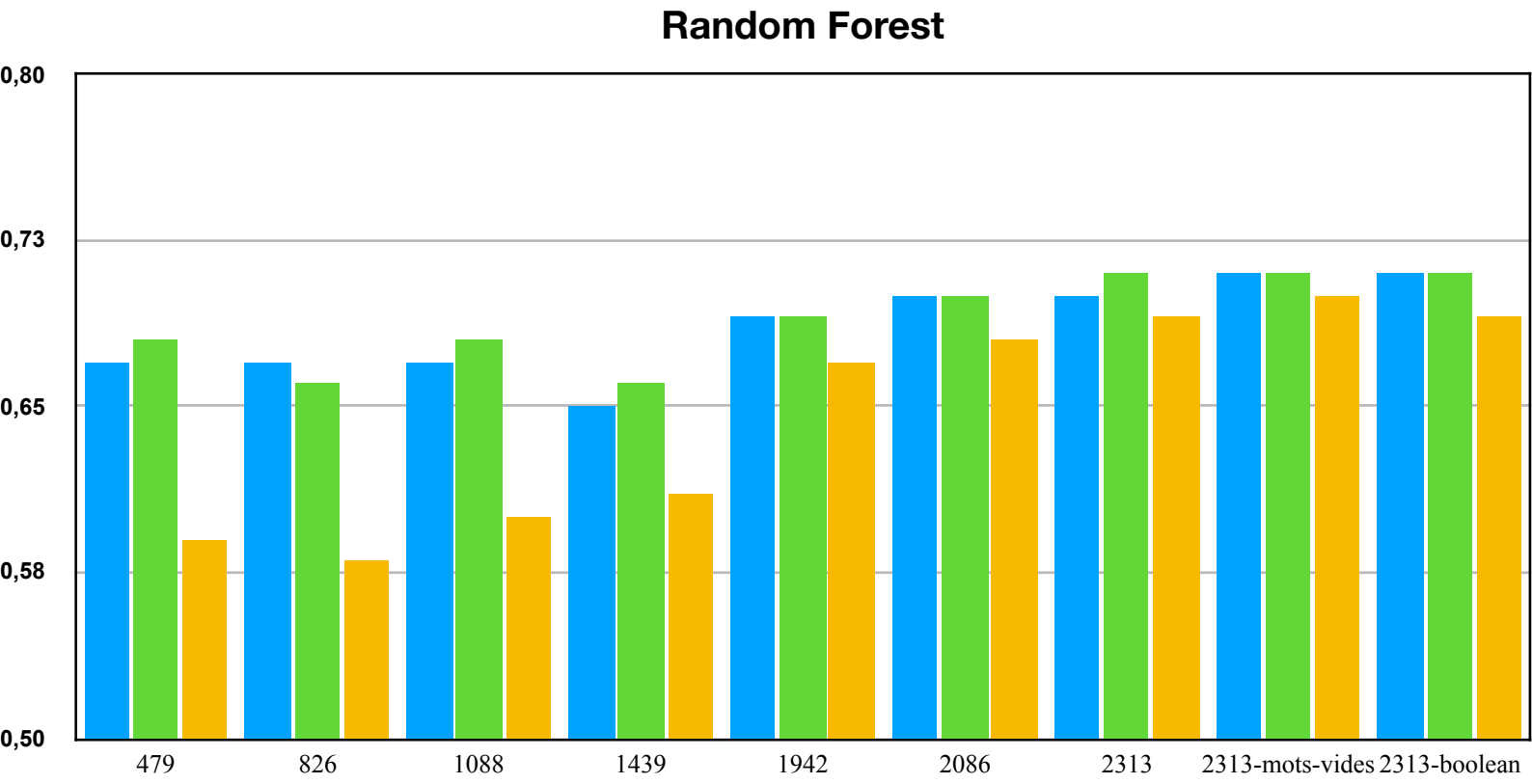
Le premier obstacle que nous avons eu est le script de vectorisation dont la segmentation en token (en occurence token = attribut) est faite selon l'espace entre deux mots, une méthode qui ne sera pas adéquate pour le chinois dont la segmentation en mot n'est pas une tâche intuitive comme pour la plupart des langues européennes, où il faut faire appel à un outil spécifique. Nous avons choisi le module Jieba, car il est entraîné sur un corpus des commentaires de produit en ligne. Nous avons intégré le module dans le script vectorisation et nous avons des mots (en deux caractères pour la plupart des cas) comme attribut. En vérifiant les attributs, il faut dire que le module Jieba a fait un joli travail de segmentation en mots.

Expériences:
Expérience I:

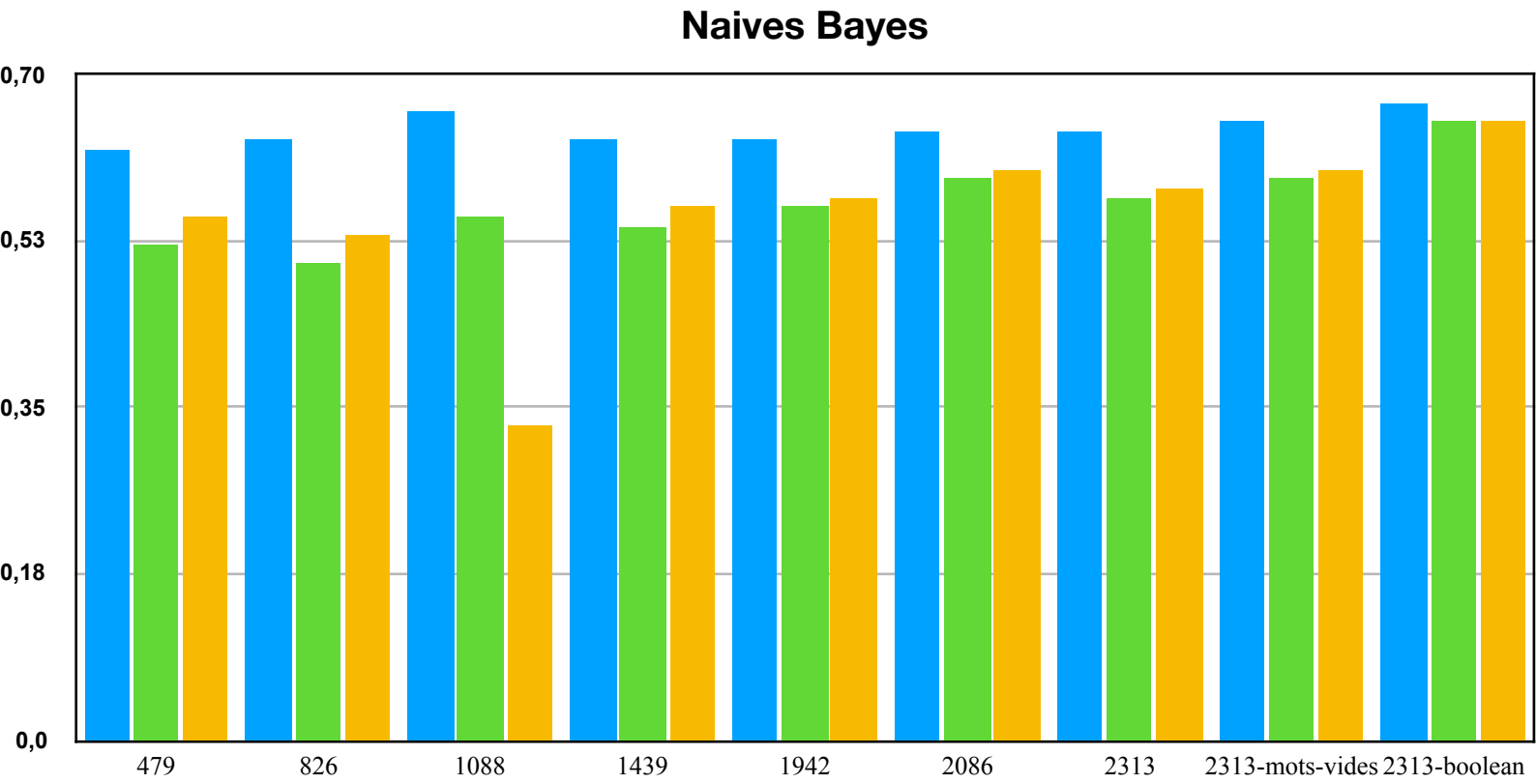
* nous avons gardé les paramètres de l'algorithme de weka.
* sauf précision, le train data est transformé en vecteur avec des nombres d'occurrences, sans enlever les mots-vides.
* test options: cross-validation Folds 10



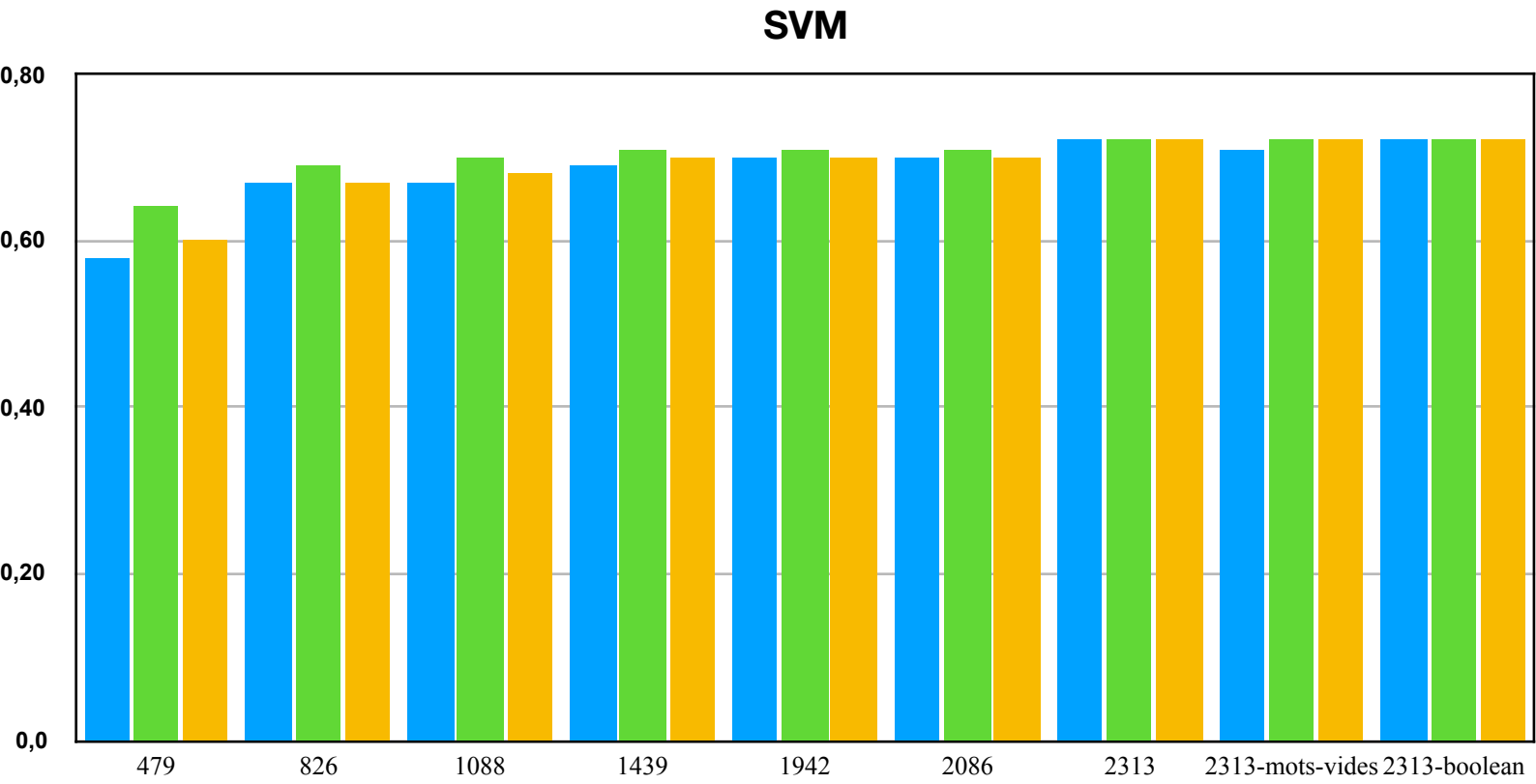
L'algorithme de l'arbre de décision est sans aucun doute le plus mauvais des quatre, d'abord il n'est pas stable, ensuite la performance baisse au fur et à mesure que la quantité d'échantillons augmente. Pourtant, J48 donne le meilleur résultat avec un train data de 479 instances. Une hypothèse que nous pouvons faire, c'est qu'avec peu de donnée, les attributs comme les adjectifs ont un poids plus importants, et avec l'augmentation de données, l'importance (l'entropie) des attributs est égalisée. Par exemple, dans les commentaires « bon », il y a souvent « romantique », « touchant », « guérissant ». Mais quand le nombre de commentaires mauvais augmente, nous y trouvons également ces adjectifs, cette fois, ces adjectifs se trouvent entre « pas » et « du tout ».



RandomForest fait partie de la famille de modèle de l'arbre. Et en général, il a une meilleure performance que les arbres de décision (comme son nom indique, Forêt d'arbres décisionnels). Je ne pourrai pas donner une hypothèse de la performance, mais toujours, on dit qu'il est souvent utilisé comme benchmark dans les compétition de machine learning, et il est difficilement interprétable.
L'inconvénient par rapport à l'arbre de décision est qu'il est plus lent à entraîner.



Notons d'abord dans ce diagramme, l'axes Y commence de 0.
Naives Bayes est toujours stable et il s'améliore à peine avec l'augmentation de données. Or, le score de la précision moyenne est toujours meilleure par rapport au rappel et à F-mesure. Cela sans doute vient du fait que dans les commentaires « bon », les adjectifs comme « romantique », « douce », « guérissant », sont très fréquemment employés. Et dans notre corpus, la catégorie « bon » a plus d'échantillons qu'aux autres.
En gros, Naives Bayes a le score le moins bon au niveau de F-mesure.

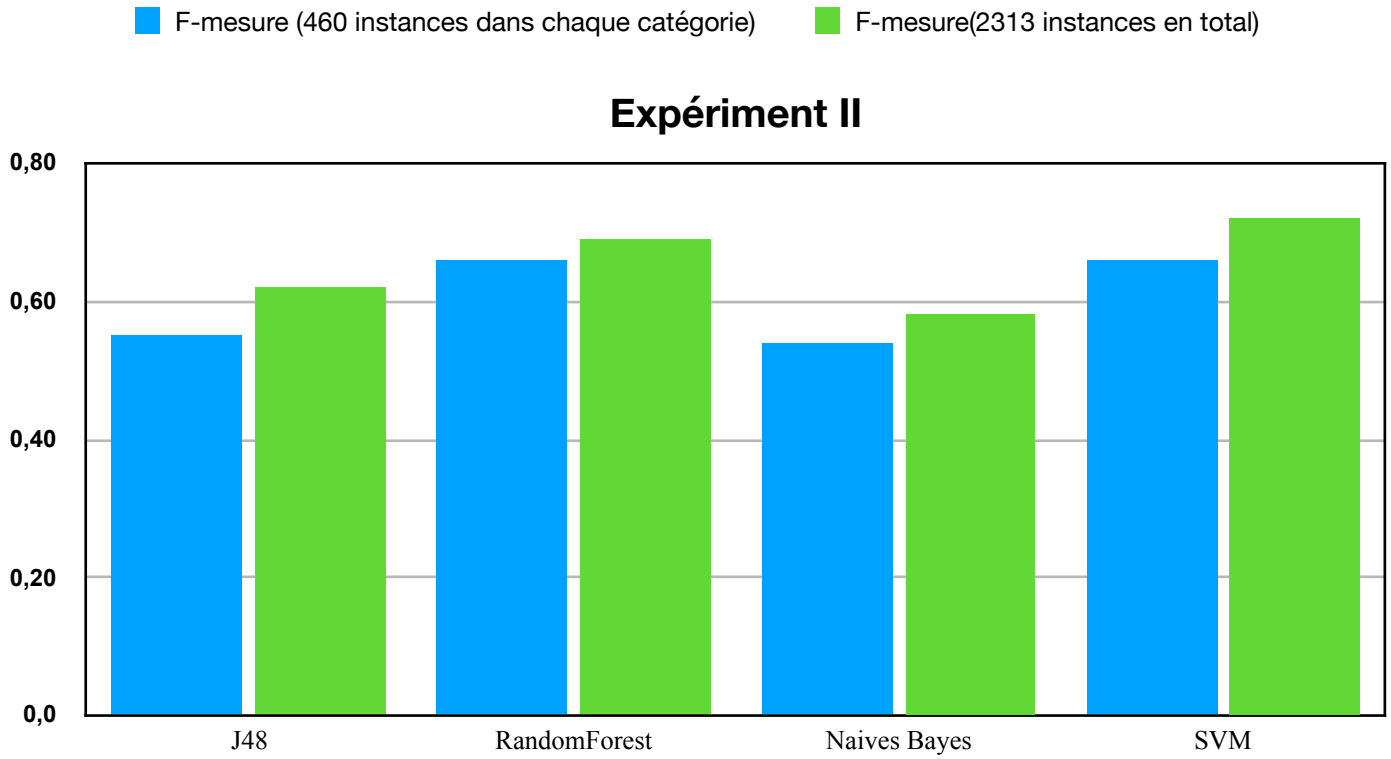


SVM est comme un bon élève, toujours bon, bon en toutes les matières: précision, rappel, F-mesure. Avec plus de donnée, il est encore meilleur. Je ne suis pourtant pas capable d'expliquer son fonctionnement, et les désavantages pour les autres algorithmes ne sont pas valables pour SVM. Mais quand même, nous pouvons tirer une conclusion, c'est que SVM a besoin de donnée. Le score augmente en même temps que le nombre d'instances.

Avant d'entamer expérience II, nous avons aussi fait une comparaison entre le mode avec mots-vides et sans mots-vides, entre sac de mot à la base d'occurence du mot et à la base booleanne, nous pouvons conclure que: dans notre cas, cela n'a qu'un impact minime sur le résultat.

Expériment 2:

Nous avons testé une dernière fois les quatre algorithmes avec un train data dont chaque catégorie contient un nombre (460) presque équivalent d'instances. Nous avons fait un diagramme avec, en bleu, le cas où chaque catégorie a un nombre équivalent d'instances (Expérience II); en vert, le cas avec 2313 instances en total(Un des tests de l'Expérience I).



Nous voyons que dans deux les deux cas, c'est SVM qui décroche le meilleur résultat, légèrement supérieur à RandomForest.
Nous ne pouvons pas dire de ce diagramme que, le fait d'avoir autant d'échantillons dans chaque catégorie n'a pas d'impact sur le résultat car les deux cas ne sont pas directement comparables, ayant chacun une quantité de donnée différente. Or, nous pouvons constater que, dans expérience II, nous n'avons pas obtenu une performance beaucoup meilleure que les tests de expérience I.

Conclusion:

Une chose que nous devons relever est que la classification pour la catégorie « bon » a toujours une meilleure précision que les deux autres catégories. Cela peut s'expliquer par le fait que le nombre d'échantillons dans la catégorie « bon » est beaucoup plus grand que les autres, et également, quand il s'agit d'un commentaire positif, c'est facilement identifié alors que pour des commentaires « moyen » et « mauvais », même un humain (en occurence moi) a une petite hésitation. Autrement dit, les attributs de commentaires « bon », surtout les adjectifs élogieux, sont plus représentatifs que les attributs des deux autres catégories.

Problème non-résolu:

Nous avons l'objectif initial d'obtenir un modèle qui sera employé à classer le reste des commentaires. Nous avons eu de l'article « How to Save Your Machine Learning Model and Make Predictions in Weka »(https://machinelearningmastery.com/save-machine-learning-model-make-predictions-weka/) une manière très rudimentaire (pas efficace) de procéder, mais pour employer cela pour plus de 20000 commentaire, cela reste encore un problème non-résolu; sinon, sans doute le module scikit-learn permet de monter un projet plus rapidement.

Résultat des expériences sous forme de tableau:

Dans chaque cellule de l'algorithme, il y a quatre chiffres qui sont en suivant l'ordre:
1, Correctly Classified Instances (%);
2, Précision (Weighted Avg.);
3, Rappel (Weighted Avg.);
4, F-mesure (Weighted Avg.).

	J48	RandomForest	Naives Bayes	SVM
479 instances 2927 attributs	66,8058 0.636 0.668 0.644	67.8497 0.665 0.678 0.595	52.4008 0,618 0,524 0,553	63.8831 0.582 0.639 0.598
826 instances 4100 attributs	63.1961 0.607 0.632 0.614	66.2228 0.671 0.662 0.584	50.3632 0.633 0.504 0.534	68.8862 0.666 0.689 0.670
1088 instances 4741 attributs	65.8088 0.637 0.658 0.644	67.739 0.668 0.677 0.599	54.8713 0.659 0.549 0.328	69.825 0.675 0.699 0.681
1439 instances 5576 attributes	61.779 0.599 0.618 0.605	66.0181 0.645 0.660 0.611	54.2043 0.632 0.542 0.563	70.8826 0.694 0.709 0.697
1942 instances 6326 attributes	59.2688 0.578 0.593 0.579	68.7951 0.688 0.688 0.668	56.1277 0.626 0.561 0.574	70.8033 0.701 0.708 0.701
2086 instances 6569 attributes	62.2723 0.608 0.623 0.609	70.278 0.702 0.703 0.684	58.9645 0.641 0.590 0.601	70.7574 0.699 0.708 0.700
2313 instances 6810 attributes	62.9918 0.618 0.630 0.621	70.5145 0.702 0.705 0.686	56.6796 0.637 0.567 0.581	72.2006 0.715 0.722 0.717
2313 instances sans mots vides	62.473 0.611 0.625 0.614	71.3792 0.711 0.714 0.697	59.4466 0.651 0.594 0.607	72.0277 0.713 0.720 0.715

2313 instances boolean	61.9974 0.607 0.620 0.610	71.1198 0.711 0.711 0.692	65.1535 0.668 0.652 0.657	72.1574 0.717 0.722 0.718
1383 instances 5149 attributes 461 instances dans chaque catégorie	56.1822 0.557 0.562 0.555	66.3774 0.66 0.664 0.659	54.1576 0.557 0.542 0.537	66.3051 0.659 0.663 0.659