POLYTECHNIQUE MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE

INF8215
Groupe 01

TP3

Classifications multiclasses : légumes secs

Par

Brando, Tovar **1932052**

Vega, Estefan **1934346**

Équipe : **BrandiniStifini**

Le 16 avril 2022

Table des matières

1 Contexte

Dans ce travail pratique, il nous était demandé de classifier des légumes secs dans leurs catégorie respectives. Il y en avait 7 en tout ; Sira, Horoz, Dermason, Barbunya, Cali, Bombay, Seker et nous devions déterminer la catégorie à l'aide de 16 *features*. Nous avions donc à résoudre un problème de classification multiclasses. Nous avons décidé d'utiliser la librairie *scikit-learn* et le modèle que nous avons utilisé se base sur les machines à vecteurs de support (*SVC OneVsOneClassifier*). Nous avons aussi exploré d'autres modèles telles que celui basée sur la descente de gradient stochastique (*SGDClassifier*) et celle basée sur les forêts aléatoires (*RandomForestClassifier*)

2 Prétraitement

Avant de commencer à résoudre le problème, il est utile de se familiariser avec les données. Nos données étaient constituées de 16 *features*. Il y avait 6000 données de test. Nous avons commencé par voir s'il manquait des valeurs dans certaines de no données test ce qui n'était pas le cas. Nous avons ensuite regardé si nos données étaient balancées.

Figure 1 – Diagramme à bandes des catégories

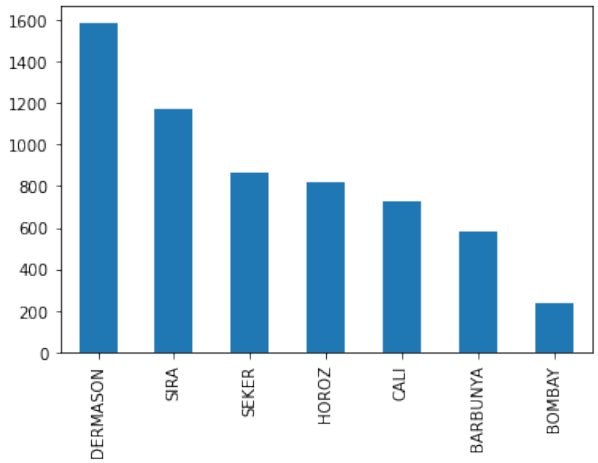
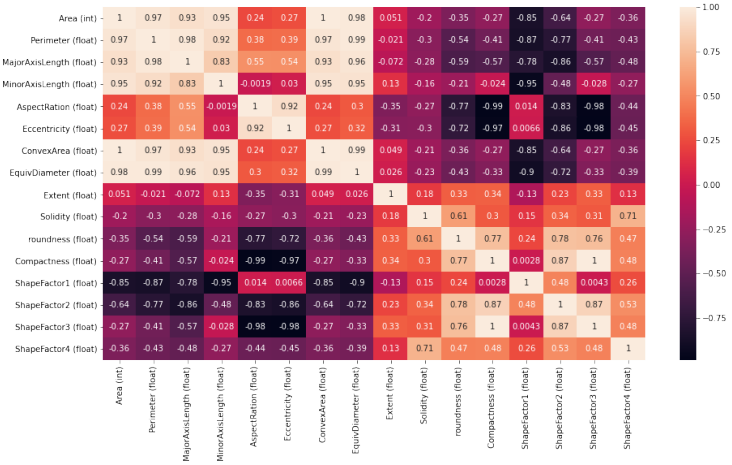


Figure 2 – Matrice de corrélation



Il a été intéressant de voir qu'il y a présence de débalancement (figure 1) et que certains attributs étaient très corrélés entre eux soit > 0.9 ou < -0.9 (figure 2). Nous avons essayé de retirer les attributs corrélés, mais il n'y a pas vraiment eu de gain et nous avons donc décider de garder la totalité des attributs.

Une fois les analyse de données terminées, nous avons dû faire quelques changements dans les données afin de pouvoir utiliser notre modèle. Nous avons en premier lieu, dû transformer les valeurs de X_train (attributs) en float. Nous avons ensuite retirer les valeurs de *ID* dans X_train et y_train. Nous pouvions ainsi entrainer notre modèle. Étant donné les mauvais résultats initiaux, nous avons dû faire appel à la normalisation. Nous sommes donc passées d'une précision de 0.268 à une de 0.933 sur le classificateur SVM. Le *scaler* qui a donné les meilleurs

résultats est le *StandardScaler* de *scikit-learn*.

3 Méthodologie

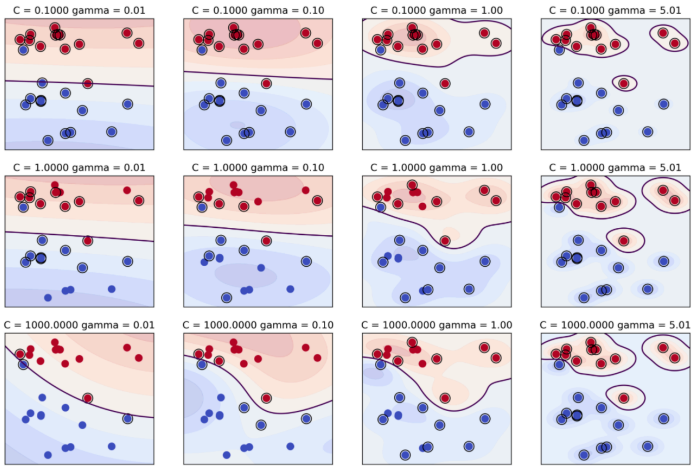
Le classificateur que nous avons choisi (*SVC*) est un classificateur binaire. Il était donc nécessaire de combiner ce classificateur avec une autre méthode. Nous avons alors choisi la méthode OneVsOne (/OneVsOneClassifier). En combinant ces deux algorithmes, nous obtenons un classificateur multiclasses qui est en fait composées de plusieurs classificateurs binaires. Dans notre cas, nous avons 7 * 6 / 2 = 21 classificateurs binaires au total.

En ce qui concerne la répartition des données, nous avons utilisé l'intégralité des données pour tester notre modèle. Nous avons pris cette décision, car nous avons utilisé en parallèle la méthode de vérification *k-fold cross-validation* avec k = 10. Ainsi, les données sont diviser en 10 parties dont 9 sont utilisées pour l'entraînement et la dernière comme partie de validation.

Comme nous pouvons le voir à la figure figure 1, nos données étaient assez déséquilibrées. Nous avons fait des recherches à ce sujet et il y avait plusieurs solutions possibles. Il y avait la possibilité de faire du *oversampling* et du *undersampling*. Nous n'avons malheureusement pas eu de succès avec ces deux méthodes. Étant donné que nous avons utilisés le modèle *SVC* de scikit-learn, nous avions accès à l'hyperparamètre *class_weight* avec comme valeur *balanced* qui associe un poids à chaque classe selon sa fréquence dans les données pour gérer les débalancements. Malheureusement cet hyperparamètre n'a pas impacté positivement nos résultats.

Le modèle que nous avons utilisé est basé sur les machines à vecteurs de support et il y a deux paramètres principaux à tenir en compte, *C* et *gamma*. D'abord il est utile de comprendre que ce modèle classifie les données de façon linéaire, ensuite que pour classifier des données de non linéaires, la méthode utilise une fonction appelé noyau (*kernel*). La valeur du gamma détermine à quel point les points proches et loin de la délimitation ont de l'importance. En d'autres mots, des valeurs élevées de gamma auront une meilleure délimitation, avec plus de courbatures, entre les données et donc le modèle aura plus tendance à souffrir de sur-apprentissage. À l'inverse, des valeurs faibles de gamma nous mèneront plus vers un problème de sous-apprentissage. La valeur de C nous indique plutôt notre niveau de tolérance aux erreurs et sa valeur impactera sur la généralisation de notre modèle. Une grande valeur de C augmente la précision et réduit les erreurs. Il fallait donc trouver un équilibre entre la précision de notre de modèle sur les données d'entrainement et sa capacité à généraliser. Nous avons trouvé un équilibre optimale avec le valeur *gamma* = 0.19 et *C* = 2.8. Les impacts de ses paramètres peuvent être observés sur la figure 3 (Amueller, 2022).

Figure 3 – SVM paramètres C et gamma



4 Résultats

Nous avons pris notre décision sur le classificateur avant tout selon la précision avec les hyperparamètres par défauts. Le tableau 1 montre les résultats des 3 classificateurs testés avec leurs hyperparamètres par défaut.

Table 1 – Comparaison des classificateurs testés avec k-fold cross-validation k=10

Classificateur	1	2	3	4	5	6	7	8	9	10	**Moyenne**	**Training set accuracy**
SVC OneVsOneClassifier	0.935	0.938	0.913	0.937	0.928	0.923	0.925	0.947	0.93	0.923	0.93	0.935
RandomForestClassifier	0.923	0.923	0.898	0.933	0.925	0.898	0.908	0.933	0.918	0.912	0.917	1.0
SGDClassifier	0.915	0.92	0.902	0.94	0.91	0.885	0.922	0.932	0.922	0.903	0.915	0.92

Nous voyons qu'avec les hyperparamètres par défauts, c'est bien le classificateur SVC qui donne les meilleurs résultats.

Finalement, nous avons optimisé notre classificateur avec *gamma* = 0.19 et *C* = 2.8 et avons comparé les différents *scalers*. Le tableau 2 montre les résultats sur les différents scalers testés.

Table 2 – Comparaison des scalers testés avec k-fold cross-validation k=10

Classificateur	1	2	3	4	5	6	7	8	9	10	**Moyenne**	**Training set accuracy**
StandardScaler	0.938	0.947	0.92	0.94	0.927	0.932	0.92	0.952	0.93	0.925	0.933	0.944
MinMaxScaler	0.927	0.928	0.89	0.933	0.925	0.915	0.918	0.935	0.927	0.918	0.922	0.924
MaxAbsScaler	0.907	0.922	0.872	0.913	0.917	0.903	0.905	0.927	0.913	0.902	0.908	0.91
Sans normalisation	0.27	0.268	0.268	0.265	0.268	0.265	0.27	0.272	0.267	0.27	0.268	1.0
Normalizer	0.263	0.263	0.263	0.265	0.265	0.265	0.265	0.265	0.265	0.263	0.264	0.264

5 Discussion

Les modèle que nous avons testés étaient tous simple d'implémentation avec *scikit-learn*. Ce qui était un peu plus difficile était d'ajuster les différents paramètres et de normaliser correctement les données, afin d'avoir un résultat optimal. Nous sommes assez satisfaits des résultats de notre modèle qui, sur kaggle donne une précision de **0.932**. En théorie, le modèle SVM est susceptible aux grands débalancements des données. Malgré cela, ne pas en tenir compte s'est avéré être le plus bénéfique pour nous. Le modèle SVM n'est aussi pas optimales pour gérer une grande quantité de données tests. En effet, SVM utilise une matrice noyau dont la taille est proportionnelle à la quantité de données test ce qui affecte les performances.

Il y a tout de même plusieurs avantages à utiliser le modèle SVM, notamment, l'utilisateur a beaucoup de contrôle sur celui-ci. Il y a d'abord les paramètres *gamma* et *C* qui permet à l'utilisateur de gérer le sur-apprentissage. Mais il y a surtout le paramètre *kernel* qui permet à l'utilisateur d'utiliser sa propre matrice pour un utilisateur qui aurait des connaissances expertes sur ses données (scikit-learn, 2022).

En ce qui concerne notre travail, il ne nous a pas été trop difficile de trouver des paramètres adéquats pour résoudre le problème de sur-apprentissage. Même avec les paramètres par défaut, nous avions un très bon modèle qui n'avait pas beaucoup de sur-apprentissage. Par contre, il est intéressant de voir que le classificateur *RandomForestClassifier* avait un *training accuracy* de 1.0 alors qu'avec la *cross-validation* il n'obtenait que 0.917 de précision.

Références

Amueller. (2022). *Support Vector Machines*. Github. https://amueller.github.io/aml/02-supervised-learning/07-support-vector-machines.html

Goyal, C. (2021). *Multiclass Classification Using SVM*. scikit learn. https://www.analyticsvidhya.com/blog/2021/05/multiclass-classification-using-svm/

scikit-learn. (2022). *sklearn.svm.SVC*. scikit learn. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html