Travail pratique 3

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Classification multiclasses : légumes secs

Équipe Kaggle : **Ghali Harti**

INF6804 - Vision par ordinateur

Hiver 2022
Département de génie informatique
Polytechnique Montréal

Date de remise : 18 avril 2022

Ghali Harti	1953494
Sanmar Simon	1938126

1 Prétraitement

Puisque toutes les colonnes de données sont numériques, aucune transformation directe n'est requise. Par contre, pour pouvoir entrainer les divers modèles, la colonne représentant les classes textuellement doit être transformée en donnée numérique, pour ce faire, il a suffi d'assigner à chaque classe un chiffre entre 0 et le nombre de classes. Par la suite, l'opération inverse est effectuée après avoir prédit des classes à partir de nouvelles données.

Après avoir effectué ce pré-traitement brut, on explore plus en profondeur les données et leurs relations. Ainsi, on génère une matrice de corrélation qui représente la corrélation entre chaque paire de catégorie de données. Ainsi, on obtient le graphique suivant :

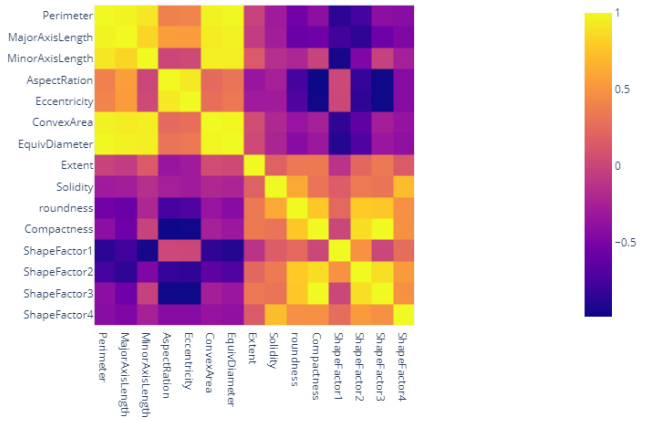

FIGURE 1 – Matrice de corrélation des données.

On remarque alors une certaine corrélation entre certaines paires de donnée, ce qui indique qu'il serait potentiellement possible de se passer de certaines composantes du jeu de donnée. Cette possibilité et approche sera explorée plus en détail dans la section Méthodologie.

Par ailleurs, il a également été utile de dresser le portait de la répartition des classes afin de voir s'il y a des classes qui sont sur-représentées ou sous-représentées. Le graphique suivant présente cette répartition des classes :

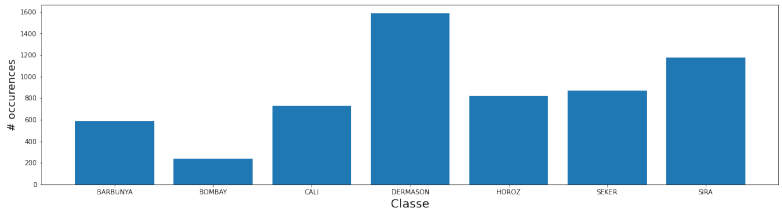

FIGURE 2 – Nombre d'occurrences par classe.

Comme on peut le voir, l'ensemble de donnée est assez équilibré au niveau de la réparation des classes, sauf pour les classes Dermason et Bombay, qui sont respectivement sur-représentée et sous-représentée.

2 Méthodologie

2.1 Répartition du jeu de donnée

2.1.1 Colinéarité des données

Après l'exploration des données et comme on peut le voir avec la matrice de corrélation, certaines catégories de données sont corrélées. On a d'abord normalisé les données à l'aide de la fonction dédiée de `Scikit-learn`, pour ensuite appliquer une analyse par principales composantes qui permet de choisir le nombre de catégories de donnée à évaluer et prendre en compte lors de l'entrainement.

2.1.2 Balancement des classes

Certaines classes étant déséquilibrées, la librairie `Imbalanced learn` a été utilisée afin de faie du *oversampling* pour la classe qui est moins représentée.

2.1.3 Division des données d'entrainement

Afin de s'assurer de ne pas être en présence de sur-apprentissage lors de la phase d'entrainement et afin d'avoir plus de chance d'avoir des résultats similaires lors de la phase de test par rapport aux performances d'entrainement, on a divisé l'ensemble d'entrainement en deux sous-ensembles en répartissant l'ensemble d'entrainement en un ensemble d'entrainement à hauteur de 80 % et les 20 % restants représentent un ensemble de validation qui agit comme proxy et n'interagit pas avec le modèle lors de son entrainement.

2.2 Exploration de divers modèles et de leurs paramètres

On a expérimenté plusieurs méthodes et algorithmes de classifications et essayer de trouver leurs meilleurs paramètres à l'aide d'un `GridSearch`. Le tableau suivant résume les principales méthodes utilisées et les paramètres correspondants dont les valeurs ont été explorées.

Modèle	Nature des hyperparamètres expérimentés
Logistic regression	penalty, class_weight, C, solver
Decision tree	ccp_alpha, splitter, criterion, min_sample_leaf
Random forest	cc_alpha, min_sample_leaf, max_depth, n_estimators
Réseau neuronal	Réseau, nombre de couches et neurones, activation, régularisation, normalisation

3 Résultats

Après avoir exploré plusieurs hyper-paramètres, les résultats suivant ont été obtenus avec le meilleur jeu de paramètre trouvé pour chacune des techniques :

Modèle	Performance
Logistic regression	0.3
Decision tree	0.920
Random forest	0.933
Réseau neuronal	0.81

Comme on peut le voir, les meilleurs résultats ont été obtenus avec un *Random forest*. Ce score a été obtenu avec les hyper-paramètres suivant :

Paramètre	Valeur
n_estimator	300
ccp_alpha	0.0001

Les valeurs des hyper-paramètres non spécifiés dans le tableau précédent est celle par défaut. Il est à noter que tel que spécifié dans la méthodologie, cette méthode a été mis au send d'un *pipeline* composé d'une étape de normalisation (*StandardScaler*), ainsi qu'un PCA pour sélectionner le nombre de classes. Le PCA a été fixé à 14 composantes. Par ailleurs, la méthode pour l'*oversampling* des classes sous-représentées n'a pas eu d'impact significatif.

4 Discussion

Le modèle de *Random forest* a donné une très bonne précision qui est nettement plus élevée que la performance minimale attendue dans le cadre de ce travail et cette performance nous a également permis d'avoir un bon classement dans le cadre de la compétition.
La méthodologie utilisée a permis de trouver la bonne combinaison de paramètre et la division du jeu de donnée en des ensembles d'entrainement et de vérification a permis d'avoir un score assez proche de celui obtenu par le jeu de donnée de test. Par ailleurs, les techniques utilisées suite à l'exploration de la répartition du jeu de donnée a permis ont permis de palier aux désavantages de celle-ci et de la mauvaise représentation de certaines classes. Finalement, la méthode utilisée reste assez simple à mettre en place et ne demande pas une grande expertise dans les divers algorithmes de classification puisque l'on utilise des méthodes et librairies qui font la majorité du travail. Il suffit de quelques lignes de code pour avoir un modèle fonctionnel donnant de bons résultats.
Les désavantages notables de la méthodologie et la façon de faire utilisée est que nous n'avons utilisé qu'un modèle à la fois, tandis que la combinaisons des résultats de plusieurs modèle, par exemple avec du *boosting* ou du *stacking* de modèles, aurait pu améliorer les résultats.