Département de génie informatique et génie logiciel

INF8215
Intelligence artificielle : méthodes et algorithmes

TP3 - Vinho Verde

Soumis auprès de Quentin Cappart (Ph.D.)
par **William Balea (1904905) et Jean-Michel Lasnier (1905682)**
Automne 2021 (10 décembre 2021)

Note importante

L'ensemble des étapes de développement de notre algorithme peuvent être retrouvés dans le notebook Jupyter **'my_wine_tester.ipynb'**. Le fichier python 'my_wine_tester.py' contient uniquement ce qui a été retenu de notre algorithme de classification pour les résultats optimaux.

1 - Description de notre méthodologie de travail

Pour mener à bien le développement de notre algorithme de classification, nous avons suivi une méthodologie structurée en quelques étapes.

Premièrement, nous avons développé une version fonctionnelle de base de notre algorithme nous permettant de faire des prédictions. Ceci nous a permis d'avoir un résultat initial sans traitement de données et avec les paramètres par défaut de l'algorithme. Ce résultat est donc notre référence pour la suite.

Deuxièmement, nous avons fait quelques étapes de pré-traitements sur les données utilisées. Nous avons fait une évaluation de l'importance des caractéristiques d'entraînement dans le but de retirer celles qui n'apportent pas d'informations supplémentaires. Nous avons aussi remarqué un débalancement des classes dans les données d'entraînement. Nous avons donc tenté une technique pour balancer les données. Pour chaque étape du prétraitement, nous avons évalué l'impact sur les prédictions pour décider ce qu'on conservait pour notre algorithme final.

Finalement, la dernière étape de notre méthodologie fut de faire une recherche sur les paramètres optimaux de notre algorithme (*hyper parameter tuning)*. Pour cette étape, la classe `GridSearchCV` de la librairie SKLearn a été utilisée.

2 - Prétraitement de données

2.1 - Feature Selection

```
Variable: alcohol              Importance: 0.1249
Variable: density              Importance: 0.1011
Variable: volatile acidity     Importance: 0.1007
Variable: total sulfur dioxide Importance: 0.0903
Variable: free sulfur dioxide  Importance: 0.0864
Variable: sulphates            Importance: 0.0856
Variable: chlorides            Importance: 0.0841
Variable: residual sugar       Importance: 0.0835
Variable: pH                   Importance: 0.0817
Variable: citric acid          Importance: 0.079
Variable: fixed acidity        Importance: 0.0767
Variable: color_red            Importance: 0.0031
Variable: color_white          Importance: 0.003
```

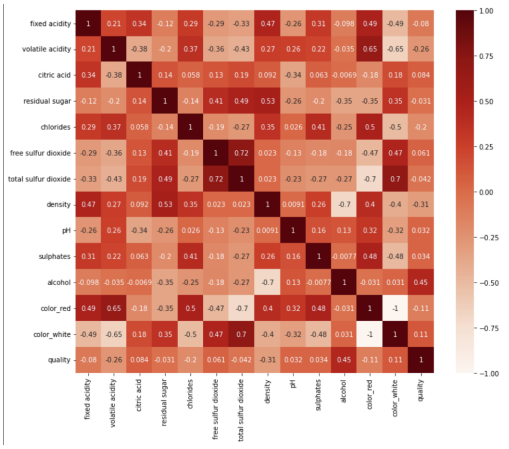

Figure 1 : Importance des caractéristiques (gauche) d'après la carte des chaleurs (droite)

La première étape du prétraitement était de faire une sélection des caractéristiques d'entraînement. Afin d'obtenir une vision initiale de la corrélation entre les données, nous avons d'abord fait une étude de corrélations visible à la figure 1. Ensuite, nous avons évalué l'importance de chaque caractéristique. En effet, il est possible qu'une caractéristique n'ait pas beaucoup d'influence sur la qualité du vin. Ainsi, afin de mieux diriger l'apprentissage et de réduire le temps de calcul, nous utilisons `feature_importances_` de notre algorithme `RandomForestClassifier`. Grâce à cela, nous pouvons classer les caractéristiques du vin

selon sa pertinence dans l'apprentissage du modèle. Il est d'ailleurs intéressant de souligner que nous avons effectué un encodage one-hot de la couleur afin de la qualifier numériquement tout en évitant un biais. Si on attribuait la couleur rouge à 1 et le blanc à 2, il aurait été possible que le modèle priorise les vins blancs à cause du nombre 2 qui est plus grand. Enfin, nous avons pu déterminer que l'acide citrique, l'acidité fixe et la couleur n'avaient pas beaucoup d'influence sur la qualité du vin. Ils pourraient être enlevés sans trop affecter la prédiction de notre modèle.

2.2 - Données non-balancés

Nous avons détecté un débalancement important au niveau de la distribution des classes dans les données d'entraînement. En effet, nous pouvons remarquer qu'il y a beaucoup plus de données pour des vins de qualité 5 ou 6, mais presque pas de données pour les vins de qualité 9. À cause de cette disproportion, il se pourrait que notre algorithme ait tendance à classer un vin à une qualité 5 puisqu'il n'a pas eu assez d'exemples d'un vin d'une autre qualité. Pour tenter de remédier à cela, nous utilisons la librairie `imblearn` et sa fonction RandomOverSampler. Il va répéter les données des classes minoritaires aléatoirement afin d'obtenir le même nombre de vins dans chaque classe de qualité.

3- Algorithme utilisé - *RandomForestClassifier*

3.1 - Description de l'algorithme

Pour notre algorithme de machine learning, nous avons décidé d'utiliser un `RandomForestClassifier` de la librairie `sklearn`. Puisque nous avions déjà utilisé les forêts aléatoires dans le cadre d'un autre cours, cela nous semblait une approche intéressante afin d'entamer ce projet. Voici comment l'algorithme fonctionne.

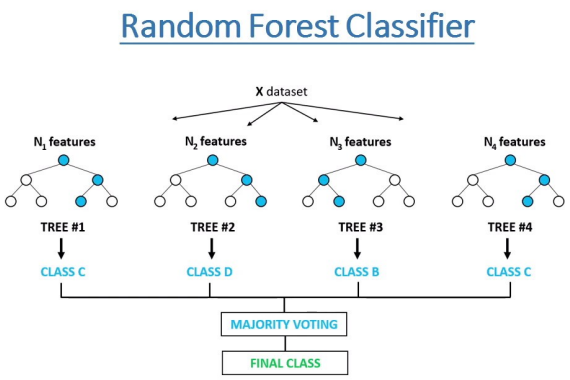

Figure 2 : Visualisation de l'algorithme Random Forest Classifier

L'algorithme de forêt aléatoire est un algorithme à base d'arbres de décisions. Ces arbres binaires sont construits à partir des caractéristiques de sorte à pouvoir les classer de plus en plus précisément. Les classements sont faits en appliquant une condition de sélection (par exemple alcool <= 6) à chaque nœud. Plus on est profond dans l'arbre, plus les conditions sont raffinées et plus les données sont distinguées.

Une forêt est un ensemble de plusieurs arbres de décision. Pour faire cela, plusieurs échantillons sont créés à partir des données présentes : c'est le *bagging*. L'algorithme fait un tirage au sort avec remise des données afin de créer plusieurs jeux de données. Cela évite de créer des arbres tous semblables.

Pour l'apprentissage du modèle, nous avons utilisé 80% des données pour l'entraînement et 20% des données pour l'ensemble de validation. La méthode `train_test_split` de la librairie SKLearn a été utilisé

3.2 - Optimisation des paramètres: *Hyperparameter tuning*

L'algorithme de RandomForestClassifier possède plusieurs paramètres qui peuvent être ajustés afin de l'optimiser et obtenir une forêt plus appropriée au problème. Parmi ces paramètres, nous avons le *criterion* qui permet de spécifier la formule utilisée pour les conditions de décisions, la profondeur maximale de l'arbre, le nombre de caractéristiques utilisées par arbres et le nombre d'arbres dans la forêt. *GridSearchCV* permet de tester toutes les combinaisons de ces paramètres de façon automatique afin de trouver les meilleurs paramètres à appliquer à notre problème. Ainsi nous avons trouvé qu'un *criterion* 'gini', qu'une profondeur complète de l'arbre, qu'un *max feature* 'auto' et qu'un maximum d'arbres étaient les paramètres optimaux.

3.3 - Résultats

Tableau 1 : Résultat des différentes configuration de notre algorithme de classification

ID	**Algo**	**Détails algo**	**Prétraitement sur les données**	**hyperparameter tuning**	**Result**
1	RFRegressor	1000 arbres	x	x	0.48410
2	RFClassifier	1000 arbres	x	x	0.66153
3	RFClassifier	10000 arbres	x	Oui	0.66256
4	RFClassifier	10000 arbres	Imbalanced dataset oversampling	x	0.66256
5	RFClassifier	200 arbres Max_depth=8 criterion=entropy max_feature=auto	x	Oui	0.60615
6	RFClassifier	5000 arbres criterion=entropy	x	Oui	0.66153
7	RFClassifier	5000 arbres	'Color' enlevé		0.66153

Le tableau 1 offre un résumé des résultats obtenus sur l'ensemble de données 'test_public.csv'. Le résultat 1 est plus faible car nous avions initialement utilisé un RandomForestRegressor. On aperçoit que le facteur le plus important a été le nombre d'arbres aléatoires utilisés dans la forêt. Le résultat 5 est le plus bas avec une *accuracy* de 0.60615 pour 200 arbres. Et nos meilleurs résultats ont été obtenus lorsqu'on utilisait 10 000 arbres avec une *accuracy* de 0.66256. L'utilisation de la technique du *hyperparameter tuning*, nous a permis de trouver que le paramètre 'criterion=gini' et le 'max_features=auto' étaient aussi à utiliser mais n'avaient pas d'impact majeur sur les résultats.Nous avons fait quelques soumissions avec le balancement des données mais comme le démontre le résultat 4, nous n'avons pas constaté d'amélioration dans le score obtenue. On avait finalement établi que la caractéristique 'color' ne semblait pas contribuer à l'entraînement et cela n'a pas eu d'impact négatif sur le score comme on peut voir avec le résultat 7. Pour conclure, pour notre meilleure soumission, nous avons utilisé 10000 arbres de décision, avec les paramètre optimaux du RandomForestRegressor trouvé grâce au hyperparameter tuning, avons enlevé la caractéristique 'color' et n'avons pas appliquer le balancement des données.