Prédire la qualité du vin à partir de ses caractéristiques phyico-chimiques

Katia Jodogne-del Litto (2160101) et Slimane Aglagal (2103355) - Equipe Dedale

I. Introduction

Le but de cette compétition Kaggle est de prédire la qualité d'un vin en fonction de ses caractéristiques physico-chimiques (couleur, pH, densité, etc.). Pour ce faire, on dispose d'un dataset d'entraînement, où la qualité des vins a été déterminée par un panel d'oenologues[1]. Il nous faut prédire la qualité des vins d'un dataset de test à l'aide d'apprentissage supervisé.

II. Prétraitement des attributs (Feature design)

Dans un premier temps, nous avons analysé les données pour comprendre leur répartition et pour les pré-traiter afin de les utiliser avec un modèle intelligent. Le dataset d'entraînement comporte 4547 observations caractérisées par 12 attributs.

Pour la préparation et le nettoyage de la base de données, nous avons vérifié les éléments suivants :

1) **Valeurs manquantes** : Nous n'avons pas trouvé des valeurs redondantes ou des valeurs manquantes dans la base.
2) **Distribution des valeurs** : Nous avons analysé les données et leur affichage pour mieux comprendre la distribution de chaque variable. Les instances sont inégalement réparties selon la couleur du vin et selon la qualité (figure 1). Les classes de valeurs 1, 2 et 10 n'ont aucune instance dans la base. Ces données déséquilibrées peuvent conduire les algorithmes à un sous-apprentissage ou un sur-apprentissage, d'où la nécessité d'équilibrer les instances de la base. Le traitement de cet aspect est détaillé en bas.

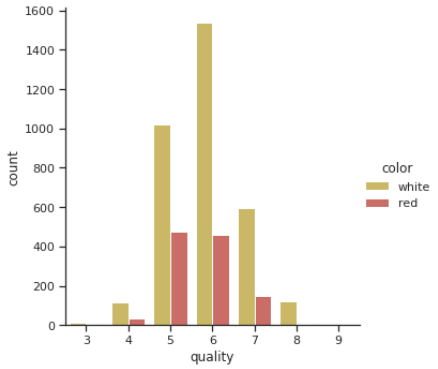

Fig. 1: Visualisation de la distribution des vins selon la robe et la qualité

3) **Variable à prédire** : La qualité ne représente pas une catégorie classique, puisqu'elle comporte une relation d'ordre. Les vins de qualité 7 sont par exemple plus proches de la qualité 8 que de la qualité 3. Il n'est donc pas nécessaire d'utiliser l'encodage one-hot. On veut au contraire conserver l'ordre.

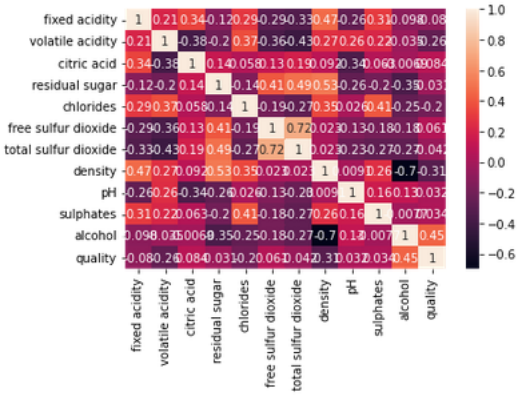

Fig. 2: Matrice de corrélation des attributs

4) **Numérisation des attributs** : l'attribut de la couleur contient les valeurs "white" et "red" qui représentent deux catégories différentes. Il faut les transformer en caractéristiques numériques pour les passer dans un modèle de calcul. Nous leur attribuons donc respectivement les valeurs 1 et 0.
5) **Sélection des attributs** : La corrélation entre les variables est examinée afin de déterminer les interactions entre les attributs. La matrice de la figure 2 montre une forte corrélation entre les attributs "Free sulfur dioxide" et "total sulfur dioxide".
Pour affiner la sélection des attributs, nous l'avons effectuée en fonction de notre modèle : une fois le premier entraînement terminé, nous avons utilisé le package `feature_selection. SelectFromModel` - librairie de sélection des attributs de *scikit-learn*[2], qui indique le poids des attributs dans le modèle (voir table I). Au vue de ces valeurs, aucun attribut ne semble superflu.
6) **Normalisation des valeurs** : après exploration des échelles de valeur (voir figure 6 en annexe), nous avons appliqué une normalisation à chaque attribut, afin qu'ils aient une moyenne de zéro et un écart de 1. L'objectif est de ramener toutes les valeurs à une échelle commune, pour que toutes les features aient la même importance au début de l'entraînement, sans perdre les caractéristiques des données.

Attributs	color	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulfates	alcohol
Poids	0.0719	0.0995	0.078	0.0815	0.0836	0.0861	0.0856	0.1030	0.0773	0.0838	0.1459	0.0340

TABLE I: Poids des attributs dans le modèle de forêt aléatoire final (classes 3 et 9 supprimées)

III. Méthodologie

A. Répartition des données

Après la transformation des attributs, on sépare le dataset d'entraînement en deux, pour obtenir un dataset d'entraînement et un dataset de validation, qui nous permettra de sélectionner le meilleur modèle avant de l'utiliser sur le dataset de test et de l'envoyer sur Kaggle. On choisit d'utiliser 20% du dataset pour la validation. Les instances sont prises de manière aléatoire pour avoir la même distribution dans les deux datasets.

B. Déséquilibre entre les classes

Comme mentionné plus haut, les données sont déséquilibrées entre les classes. En utilisant le ré-échantillonnage, ce problème peut être résolu. Le rééchantillonnage consiste à ajouter des copies d'exemples de la classe sous-représentée en créant de manière artificielle de telles instances (sur-échantillonnage) ou en retirant de la classe sur-représentée (sous-échantillonnage). D'autres techniques permettent de gérer le déséquilibre des données, ce qui nous a donné quatre possibilités de traitement :

- Si on ne **change pas les données**, le déséquilibre peut compromettre la performance de certains classificateurs
- On peut rééquilibrer les classes en **créant des instances artificielles**. Mais il y a vraiment très peu d'instances des classes 3 et 9, donc ces données serait très artificielles.
- Il est également possible de rééquilibrer les classes en **ajoutant leur poids respectifs dans les paramètres du classificateur**.
- Enfin, on peut **supprimer simplement les classes très peu représentées** puisque le nombre d'instances est négligeable et que les datasets sont identiquement distribués: il y aura donc également très peu de ces classes dans les datasets de validation et de test et cela n'influence pas négativement la performance. La mesure que l'on cherche ici à maximiser est la précision, et elle n'est pas affecté par la disparition des classes minoritaires.

La partie IV montre plus en détails l'influence du pré-traitement des attributs sur la performance.

C. Méthodes choisies

Nous avons choisi d'utiliser un algorithme de forêts aléatoires pour cette tâche. Il s'agit d'une extension des arbres de décision. Les résultats seront comparés avec d'autres méthodes testées, mais non présentées en détails ici (réseau de neurones et SVM).

1) Fonctionnement des forêts aléatoires: Tout d'abord, il faut comprendre le principe des arbres de décision. Un arbre de décision est créé avec un dataset d'entraînement. A chaque nœud on choisit un attribut qui permet d'effectuer une **partition entre les données**, puis on recommence jusqu'à aboutir à des feuilles où une décision fiable peut être prise pour l'ensemble des données se trouvant dans la feuille. La création d'arbre de décision repose sur des mécanismes gloutons, puisqu'à chaque nœud on choisit la partition qui semble optimale.

Le principe des forêts aléatoires est d'utiliser plusieurs arbres de décision. On construit un certain nombre N d'**arbres de décision semi-indépendants** en n'utilisant qu'une sélection d'attribut pour les décisions. Ensuite la décision finale est prise par un vote majoritaire sur l'ensemble des décisions de tous les arbres.

Cela permet de compenser les problèmes de sur-apprentissage qui peuvent survenir avec les arbres de décision simples, et de ne pas tomber dans des **minimum locaux**, ce qui peut survenir à cause des mécanismes gloutons.

Pour l'implémentation, nous avons utilisé la librairie *scikit-learn* et le package `RandomForestClassifier`.

L'algorithme des forêts aléatoires comporte un grand nombre d'hyper-paramètres qu'il faut régler de manière fine pour obtenir les meilleures prédictions possibles.

2) Sélection des hyperparamètres: Pour la recherche d'hyper-paramètres nous avons utilisé le package `GridSearchCV` de la librairie *scikit-learn*. Il permet de renseigner un certain nombre d'hyper-paramètres et les valeurs que l'on souhaite explorer, et une recherche exhaustive est effectuée en entraînant le classificateur avec chaque combinaison d'hyper-paramètres possible. La performance pour chaque combinaison est évaluée à l'aide de **validation croisée**: le dataset d'entraînement est divisé en 5, et entraîné 5 fois en laissant à chaque fois de côté une des parties qui servira pour le calcul de la performance. La précision finale est ensuite la moyenne des 5, et on choisit la combinaison d'hyper-paramètres qui donne la meilleure performance.

Les hyper-paramètres pris en compte et leur valeur finale dans le modèle sont indiqués dans la table II. Les autres hyper-paramètres de l'algorithme reçoivent leur valeur par défaut dans la librairie *scikit-learn*.

IV. Résultats

Tout d'abord, nous avons essayé les trois algorithmes sur la base pré-traitée sans équilibrer les données. Les résultats sont présentés sur la figure 3 . Random Forest est l'algorithme qui donne de bons résultats en termes de précision sur les données de validation et de test (Dataset sur Kaggle). Ensuite,

Paramètres	Description	Observations	Meilleures valeurs
criterion	Mesure qui détermine selon quel critère un arbre doit être divisé : par critère de gini ou entropie	'gini','entropy'	'entropy'
n_estimators	Nombre d'arbres créés dans la forêt aléatoire	$[50, 100, 250, 500, 600, 700, 800, 900]$	500
max_depth	Profondeur maximale à laquelle l'arborescence peut être étendue	$[1, 2, 5, 10, 15, 20, 25, 30]$	20
min_samples_leaf	Nombre minimum d'instances devant être présentes dans une feuille	$[1, 2, 3, 4, 5, 10]$	1
bootstrap	Détermine si chaque arbre est formé à partir d'un dataset fait d'instances tirées au sort avec replacement ou du dataset entier.	True, False	True

TABLE II: Hyper-paramètres pour la sélection du modèle

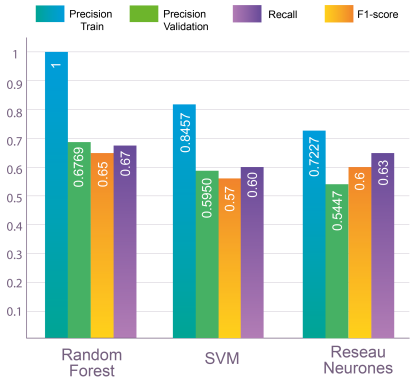

Fig. 3: Comparaison des performances pour 3 méthodes

nous avons cherché le meilleur équilibre des données qui peut améliorer les résultats de prédiction.

Nous avons testé plusieurs scénarios développé avec la partie III-B, qui forment toutes les combinaisons entre la suppression des classes sous-représentées 3 et 9, la génération d'instances (SMOTE), l'utilisation du poids des classes.

Nous avons constaté que le fait de supprimer les classes 3 et 9, et d'ajouter le paramètre de poids des classes donne le meilleur résultat, avec la figure 4.

De plus, visualiser la matrice de confusion des résultats avec les classes 3 et 9 (figure 5) montre que les instances de ces deux classes sont de toute façon systématiquement mal classées. Les inclure dans le dataset d'entraînement n'améliore donc pas la performance, on peut les retirer.

Nous avons obtenu une précision de **0.6769 sur les données de validation et 0.6717 sur les données de test public (Kaggle)**. En conclusion, les classes minoritaires d'un ensemble de données n'obtiendront pas une bonne représentation sur un classificateur et la représentation de chaque classe peut être

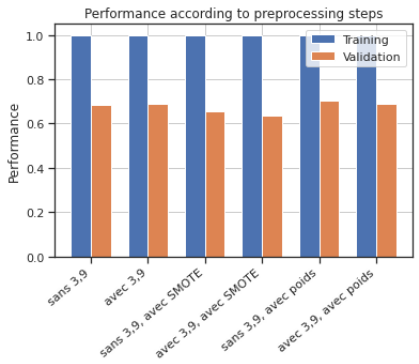

Fig. 4: Performances avec les différents scénarios de prétraitement des données

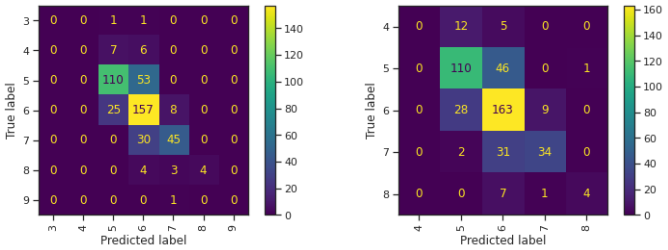

Fig. 5: Matrice de confusion sur le set de validation du modèle entraîné avec et sans les classes extrêmes 3 et 9 (et avec le poids des différentes classes)

résolue par l'équilibrage des poids des classes.

V. Discussion

Quels sont donc les avantages et les inconvénients de cette méthode de classification ? Tout d'abord, les algorithmes de forêts aléatoires et d'arbres de décision sont **parmi les méthodes de classification les plus efficaces**. Les forêts aléatoires fonctionnent également bien avec des données donc les relations ne sont **pas linéaires**, comme c'est le cas ici. Il y a également **peu de risques de sur-apprentissage** par rapport aux arbres de décision simples, surtout en ajoutant un paramètre de poids pour chaque classe.

Cependant, cet algorithme possède certains inconvénients. Ainsi l'apprentissage peut être assez long. De plus, quand le nombre d'attributs augmente, la performance diminue. Dans notre cas il n'y avait qu'une dizaine d'attributs donc ce problème ne nous a pas touchés. Il y a également un **grand nombre d'hyper-paramètres** à régler finement. Mais cela reste faible par rapport à un réseau de neurones par exemple, dont la structure peut varier énormément.

Enfin, nous avons choisi ici de nous concentrer sur la prédiction des valeurs les plus représentées, car la mesure à maximiser est la précision. Dans le cas d'autres applications, il peut être plus intéressant justement de se pencher sur ces classes minoritaires (détection d'*outliers*, de maladies, ou d'attaques informatiques par exemple). Il faudrait déployer des techniques de prétraitement des données plus sophistiquées et s'intéresser à la métrique recall ou au score f1 plutôt qu'à la précision.

REFERENCES

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4) :547-553, 2009.
[2] Scikit-Learn Documentation (scikit-learn.org)

APPENDIX

```
color
['white' 'red']

fixed acidity
min :  3.9 ; max :  15.9

volatile acidity
min :  0.08 ; max :  1.58

citric acid
min :  0.0 ; max :  1.23

residual sugar
min :  0.7 ; max :  31.6

chlorides
min :  0.009000000000000001 ; max :  0.611

free sulfur dioxide
min :  1.0 ; max :  289.0

total sulfur dioxide
min :  6.0 ; max :  440.0

density
min :  0.98711 ; max :  1.0103

pH
min :  2.72 ; max :  4.01

sulphates
min :  0.23 ; max :  2.0

alcohol
min :  8.0 ; max :  14.9
```

Fig. 6: Répartition des valeurs des attributs