Mesure de la qualité d'un vin avec l'apprentissage automatique

Frédérique Roy (1894397), Louis Plessis (1933334)
Équipe Kaggle : FlyingLotus

1. Prétraitement des attributs

La première étape de prétraitement était de supprimer la première colonne des données (« id ») : cet attribut est un identificateur arbitraire qui est inutile pour la prédiction.

Ensuite, les valeurs « white » et « red » ont été remplacées par des valeurs numériques « 0 » et « 1 », pour qu'elles puissent être interprétables par Scikit-learn.

Pour sélectionner les attributs utiles à notre prédiction, nous avons utilisé la fonctionnalité « RFE » de Scikit Learn [1], pour garder les 10 attributs les plus significatifs. Nous en avons conclu que les attributs de couleur du vin (« color ») et d'acidité (« fixed acidity ») n'étaient pas significatifs (Figure 1), et pouvaient donc être supprimés de notre matrice de données. Cela s'explique sûrement par la faible variance des valeurs : l'acidité varie grossièrement de 5 à 10, avec de rares extrêmes entre 3 et 5 et entre 10 et 15.

```
Out[27]: array([3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Figure 1: Classement de l'importance des attributs selon la fonctionnalité RFE de Scikit Learn

2. Méthodologie

a. Échantillonnage du jeu de données

La séparation des données en ensemble d'entraînement et de validation a été faite de façon arbitraire, avec un ratio de 80:20. Ainsi, les 3600 premières entrées des données « Train » ont été utilisées pour l'entraînement, et les 947 entrées restantes pour la validation.

Cette séparation permet de réserver une assez grande quantité de données pour obtenir le modèle le plus juste possible, tout en en disposant d'assez pour valider sa performance. C'est avec l'ensemble de validation que nous avons mesuré la performance de notre modèle avant de soumettre nos prédictions sur Kaggle.

b. Choix du modèle

Notre méthode pour trouver le meilleur modèle était de tester différents classifieurs, sans réglage des hyperparamètres dans un premier temps. Pour l'entraînement des données, plusieurs classifieurs ont été considérés :

- Naïve Bayes

- Régression logistique
- Machines à vecteurs de support
- K plus proches voisins
- Arbres de décision
- Forêts d'arbres de décision

Les quatre premiers classifieurs nous dont donné des performances médiocres, aux alentours de 45% (voir Tableau 1). Pour Naïve Bayes, une raison probable de cette mauvaise performance était la grande quantité de données (plus de 4500 entrées). Les machines à vecteurs de support ainsi que les k plus proches voisins auraient probablement pu être de bons candidats pour notre modèle, si une bonne configuration des hyperparamètres avait été trouvée. Cependant, les arbres de décision nous ont donné d'emblée une performance acceptable, ce qui nous a encouragé les privilégier, en particulier la forêt d'arbres de décision.

La forêt d'arbres de décision permet de combiner plusieurs arbres sur différents échantillons de nos données et de trouver automatiquement la meilleure configuration [2], ce qui explique la bonne performance trouvée d'emblée avec ce classifieur.

c. Réglage des hyperparamètres

Pour améliorer la performance de notre modèle, nous avons essayé plusieurs combinaisons de valeurs d'hyperparamètres grâce à la fonctionnalité « Grid Search » de Scikit Learn. Cette fonctionnalité permet de tester automatiquement la performance de notre modèle en fonction de différentes valeurs d'hyperparamètres définies [3]. Les différents hyperparamètres considérés pour notre modèle sont les suivants :

- ***max_depth*** : définit la profondeur maximale des arbres.
- ***n_estimators*** : définit le nombre d'arbres pour notre forêt.
- ***criterion*** : définit le mode de mesure de la qualité des *splits* des arbres. Entre « *gini* » et « *entropy* », c'est toujours « *entropy* » qui nous donnait la meilleure performance, c'est-à-dire le gain d'information pour chaque arbre.

Le choix des différentes valeurs à tester s'est d'abord effectuée de manière arbitraire, en testant manuellement différentes valeurs puis en évaluant la performance du modèle. Une fois la gamme de valeurs optimales trouvée, nous avons utilisé *GridSearch* pour trouver la meilleure combinaison. Toutefois, une itération de *GridSearch* étant très longue à effectuer (parfois plusieurs heures d'attente), nous avons procédé par petites itérations, en testant de très petites gammes de paramètres à chaque fois. À notre dernière itération, la gamme de paramètres était [17;19] pour *max_depth*, et {400;700;1000} pour *n_estimators*.

3. Résultats

Une fois les meilleurs paramètres trouvés, notre modèle présentait une performance d'environ 68% sur l'ensemble de validation. Les performances des autres modèles sont présentées dans le tableau suivant :

Tableau 1: Meilleures performances obtenues pour les différents modèles considérés

Modèle	Meilleure performance (*Validation accuracy*)
Naive Bayes	0.45195353748680045
Logistic Regression	0.4857444561774023
Linear SVC	0.470960929250264
K-Neighbours	0.4804646251319958
Decision Trees	0.5913410770855333
Random Forest	0.6821541710665259

Les meilleures valeurs des hyperparamètres de notre modèle sont de 18 pour *max_depth* (soit une profondeur maximale de 18 pour chaque arbre), et 700 pour *n_estimators* (soit 700 arbres dans notre forêt, ce qui permet de bien meilleurs résultats qu'avec un seul arbre seulement).

4. Discussion

Les résultats présentés dans le tableau ne sont pas tout à fait comparables entre eux, étant donné que le seul modèle pour lequel une optimisation des hyperparamètres a été effectuée est la forêt d'arbres décisionnels. Cependant, on se rend compte que cette optimisation n'a amenée qu'une amélioration minime du modèle, la performance étant passée de 66 à 68%.

On peut supposer que les autres modèles auraient pu être plus performants si un meilleur réglage des hyperparamètres avait été effectué. Il faut cependant garder à l'esprit que trouver les paramètres optimaux est une tâche extrêmement longue et nécessite beaucoup de ressources (le processeur est très sollicité). C'est pour cette raison que nous avons préféré nous concentrer sur un seul classifieur, en essayant de l'optimiser au maximum. Si nous avions eu de meilleures ressources machine, nous aurions pu tenter de mieux paramétrer les autres classifieurs.

Pour trouver un modèle encore plus performant, il doit probablement exister de meilleurs classifieurs, plus adaptés à notre ensemble de données. En effet, en fonction de la variance et des autres caractéristiques de l'ensemble de données, un classifieur peut être plus adapté qu'un autre. De plus, peut-être qu'une meilleure séparation des ensembles d'entraînement et de validation (répartition et échantillonnage différents) aurait permis d'obtenir un indicateur de performance plus proche de la réalité. Enfin, il est certain qu'il est possible d'améliorer la performance du modèle actuel, en paramétrant d'autres hyperparamètres tels que le nombre d'attributs maximums à considérer (max_features) ou encore le nombre minimal d'échantillons requis pour séparer un nœud (min_samples_split) [2].

Références

[1] «RFE,» 2021. [En ligne]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html. [Accès le 06 12 2021].

[2] «RandomForestClassifier,» 06 12 2021. [En ligne]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[3] «GridSearchCV,» 2021. [En ligne]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accès le 06 12 2021].