**Mini projet – Ensemble Learning**

*Nono Armel TCHIASSO*

*Rudy LOMBARD*

1. **Base de données**

Dans un premier temps, on a chargé la base de données présente dans le document excel « beer\_quality.xlsx ». On a séparé en deux pour avoir d’un côté les observations et de l’autre, les labels. Les labels sont représentés par la variable « quality » qui mesure la qualité d’une bière. De plus, on a normalisé toutes les observations X car les variables n’ont pas la même unité. Cela peut fausser le résultat que nous obtiendrons à la fin de notre apprentissage. Enfin, on a divisé cette base de données en deux sous-ensembles d’apprentissage et de test (70/30).

1. **Classification binaire**
2. On crée une nouvelle variable quantitative ybin à deux modalités comme énoncé dans le TP. Il est très important d’appliquer le calcul de la médiane sur les donnés d’apprentissage (données estimées) et non sur tout l’ensemble des données (apprentissage + test). Le modèle n’ayant pas connaissance des données en base de test, il est donc normal d’appliquer cette médiane que sur y\_train. En revanche, la variable y\_bin doit être créée sur l’ensemble de la base de données (apprentissage + test) car cela nous permettra d’appliquer une prédiction sur ce nouveau label.

Une image contenant texte

Description générée automatiquement

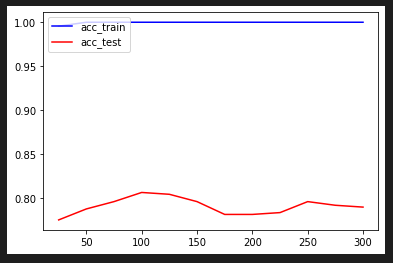
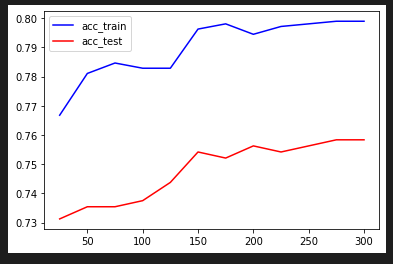
1. On optimise un arbre de décision grâce à la library scikit-learn et GridSearchCV. On entraîne ce modèle sur la base d’apprentissage et on prédit les réponses pour la base de test comme suivent :

Une image contenant texte

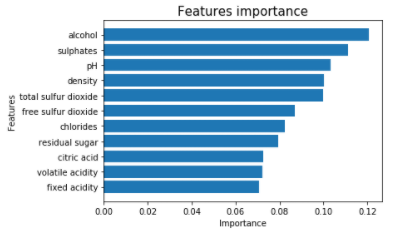
Description générée automatiquement

1. Par la suite, on entraîne un ensemble d’arbres de décision très peu profonds à l’aide d’AdaBoost qui permet d’utiliser la méthode du boosting.

Premièrement, on a tracé les courbes accuracy en fonction de n\_estimators pour max\_depth = 1 en train et en test (à gauche), deuxièmement, on a tracé les courbes accuracy en fonction de n\_estimators pour max\_depth = 5 en train et en test (à droite) :



Ensuite, on a mesuré l’importance d’une caractéristique dans la décision d’AdaBoost à l’aide d’un histogramme. On a classé de manière décroissante l’importance de la caractéristique. Ici on utilise un arbre de décision comme classificateur de base, donc l'importance de la caractéristique AdaBoost est déterminée par l'importance moyenne de la caractéristique fournie par chaque arbre de décision.



Sur ce graphique, nous observons que les 5 premières features ont le plus d’importance quant à la décision de notre modèle.

En théorie, avec le boosting, on entraîne nos arbres "en série", des modèles relativement faibles où chacun est en situation d'underfitting. Donc ici, ce type d'architecture doit permettre de réduire le biais de l'algorithme.

Dans notre cas, après la visualisation des graphes d'accuracy train/test, nous sommes en situation d'underfitting avec un biais haut et une variance haute car l'accuracy en train est supérieure à celle en test. De plus, l'écart entre les deux accuracy est très important.

1. **Classification multiclasse**
2. On a créé une nouvelle variable quantitative ymulti discrète à 3 modalités maintenant toujours avec le même procédé (calcul de la médiane sur les données estimées) :

- qualité basse (0) : y < m,

* moyenne (1) : y = m,
* haute (2) : y > m

1. On a déterminé les effectifs des différentes classes sur la base d’apprentissage. On a remarqué que la classe 2 étaient en sous-effectif.



On a donc rééquilibré les données d’apprentissage à l’aide de SMOTE.



On visualise que la classe 2 a été rééquilibrée.

1. Ici, la méthode a été de resplit nos données rééchantillonnées pour avoir 70% en apprentissage et 30% en validation.

Pour nos données équilibrées :



On optimise ce réseau à une couche cachée avec un GridSearchCV, nous donnant comme modèle :





Une image contenant texte

Description générée automatiquement

On observe que les résultats sont assez faibles et que la reconnaissance pour chaque classe est très hétérogène.

Pour les données non-équilibrées :

Une image contenant texte

Description générée automatiquement

Après optimisation avec GridSearchCV, on obtient :

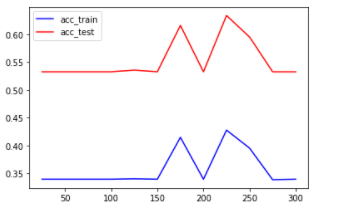
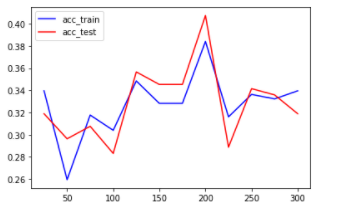
Une image contenant texte

Description générée automatiquement

La aussi, on observe des données très hétérogène avec 0% de TR pour la classe 1, expliqué par le déséquilibre en apprentissage.

1. Le principe du Bagging est d’entraîner plusieurs copies d’un modèle (en over-fitting), en entrainant chaque copie sur des données aléatoires du dataset. Ensuite, on regroupe les prédictions de chaque modèle pour faire notre prédiction finale.

Avec nos données, nous avons les résultats suivants (à gauche pour les données équilibrées, à droite les données non-équilibrées) :



A

Pour les données équilibrées, on observe ici que les résultats sont assez faibles, mais que notre modèle n’est pas en sur-apprentissage, les résultats en test sont assez proches de ceux en train.

Pour les données non-équilibrées, on observe ici que notre test est assez supérieur au train (20% environ) peu importe le nombre d’estimateurs. On observe ici une variance élevée, notre modèle est en sur-apprentissage. Il faudrait soit régulariser notre modèle, soit disposer de plus de données afin que celui-ci soit performant.

1. Nous avons développé un RandomForestClassifier, après l’avoir optimisé nous obtenons un classifieur pour **les données équilibrées**de :

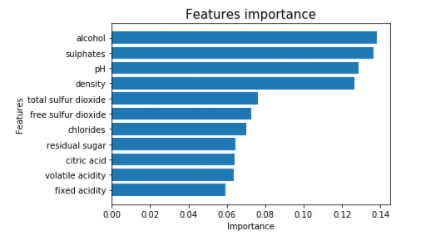
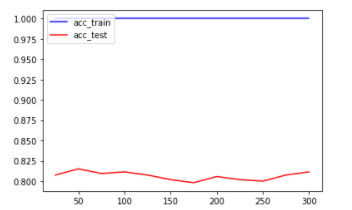


Une image contenant texte

Description générée automatiquement



On observe ici des résultats plus performants que sur nos autres classifieurs, d’une part, un score global de 80% de TR, et d’autre part, un TR majoritairement élevé pour l’ensemble de nos classes.



On observe aussi que nos résultats sont performants malgré le sous-apprentissage visible sur cette courbe qui représente une faible variance mais un biais plutôt élevé.

On distingue 4 features majoritaires qui ont permis au classifieurs de prendre ses décisions, contrairement à l’arbre de décision optimisé en partie B qui se basait sur toutes les features pour donner un résultat.

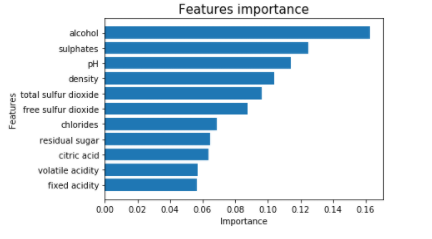
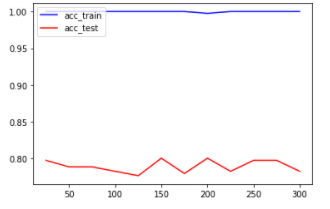
**Pour les données non-équilibrées :**



Une image contenant texte

Description générée automatiquement

On observe ici que nos données sont plutôt bien généralisées malgré la mauvaise répartition entre nos classes. Nous conservons 0% de TR pour la classe 1, mais nos 2 autres classes sont bien détectés par notre modèle.



Ce modèle est lui-aussi en cas de sous apprentissage comme pour les données équilibrées.

Ici, on remarque qu’une feature se distingue, le modèle se concentre majoritairement sur la teneur en alcool afin de prendre une décision. Ce modèle ne généralise pas aussi bien que celui avec des données équilibrées.

**D. Conclusion générale sur l’ensemble des méthodes**

**Classification binaire :**

Une image contenant table

Description générée automatiquement

**Classification multi classe sur données équilibrées :**

Une image contenant table

Description générée automatiquement

**Classification multi classe sur données non-équilibrées :**

Une image contenant table

Description générée automatiquement

Au cours de ce projet, nous avons utilisé 2 algorithmes dans le cadre de la classification binaire, et 3 algorithmes dans le cadre de la classification multi classe.

Nous avons observé que dans le cadre de la classification binaire, qu’un arbre de décisions optimisé avec des données non-équilibrés et à l’aide d’Adaboost, nous renvoie un modèle performant à hauteur de 75% de TR en test avec un biais et une variance plutôt équilibrée (pour max\_depth=1) et de bonnes prédictions en test.

Dans la classification multi classe, les performances du réseau de neurone avec Bagging sont très basses avec un TR global et pour chaque classe faible que ce soit avec des données équilibrées et non-équilibrées.

Pour le classifieur en forêt aléatoire, nous avons observé des performances beaucoup plus efficaces pour nos données. Tout d’abord, un TR global atteignant 80% de TR pour nos deux types des dataset, mais aussi une généralisation par classe plus performante que pour notre MLP. Nous avons observé que les temps d’apprentissage et d’inférence ici importe peu car nous disposons de 1600 données, cela entraine des temps de calculs assez faibles.