esiea
INGÉNIEUR·E·S D'UN NUMÉRIQUE UTILE

Mini-Projet Ensemble Learning : Conclusion

Auteurs : Sarra Madad & Manelle Nouar
Enseignant : Lionel PREVOST

20 janvier 2023

Conclusion générale sur les méthodes d'ensemble.

Comparer les différentes techniques mises en œuvre en termes de performances (accuracy, temps d'apprentissage et d'inférence) en rassemblant les résultats dans un graphe ou un tableau.

Contexte.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

FIGURE 1 – Base de données - Beer Quality

La base de données « beer quality » comporte 1600 exemples décrits par 11 caractéristiques quantitatives. L'objectif est d'évaluer, sachant ces variables, la qualité d'une bière, évaluée de 1 (très médiocre) à 10 (particulièrement excellente).

Nous allons séparons la base de données en deux parties :
- 70% apprentissage ;
- 30% test.

Nous allonsprocéder de la façon suivante :
- Classification binaire à l'aide d'arbres de décision et de boosting ;
- Classification multi-classes à l'aide de réseau de neurones, de random forest et de bagging.

Critères :
Nous comparerons les accuracy en apprentissage et en test ainsi que les temps d'inférence de chacune des solutions afin de proposer le modèle le plus pertinents pour ce type de problème.
Nous regardons aussi le compromis biais/variance.

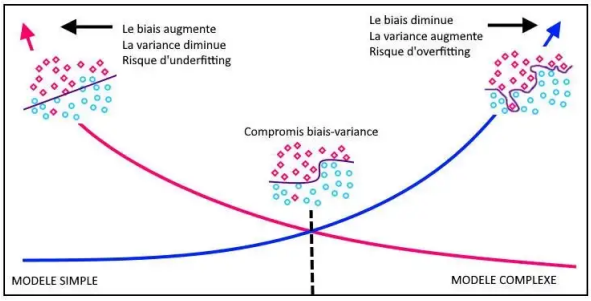


FIGURE 2 – Biais et variance

1 | Classification binaire : Decision Trees

Le dataset se compose des labels/classes (qualité de bière) suivantes :

```
Classe : 3 Fréquence : 10
Classe : 4 Fréquence : 53
Classe : 5 Fréquence : 681
Classe : 6 Fréquence : 638
Classe : 7 Fréquence : 199
Classe : 8 Fréquence : 18
```

FIGURE 3 – Répartition des classes (multi-classe)

Pour pouvoir effectuer une classification binaire - donc à deux classes - nous souhaitons passer de 6 classes présentes à seulement deux classes.
Pour cela, nous allons passer à une classe 0 toutes les observations dont la qualité est inférieure à la médiane, 1 sinon.
Voici ce que l'on a après manipulation :

```
1    855
0    744
Name: quality, dtype: int64
```

FIGURE 4 – Répartition des classes (binaire)

Le modèle utilisé pour la classification binaire est l'arbre de décision, dont deux paramètres doivent être optimisés :
- max_depth (profondeur maximum) : il s'agit d'arrêter le développement de l'arbre une fois qu'il a atteint une certaine profondeur, cela évitera que l'arbre construise des branches avec trop peu d'exemples et donc permettra d'éviter un sur apprentissage ;
- min_samples_split (nombre d'exemples minimum dans un noeud) : consiste à ne pas splitter une branche si la décision concerne trop peu d'exemples. Cela permet également d'empêcher le surapprentissage.

Pour chercher leur optimum, nous effectuons un random_search, qui va initialiser aléatoirement des couples de paramètres et ressortir celui avec le meilleur score (accuracy). Voici les paramètres optimaux dans notre cas :

```
Results from Random Search

The best estimator across ALL searched params:
DecisionTreeClassifier(criterion='entropy', max_depth=77, min_samples_split=5)

The best score across ALL searched params:
0.7381646380525304

The best parameters across ALL searched params:
{'min_samples_split': 5, 'max_depth': 77, 'criterion': 'entropy'}
```

FIGURE 5 – Paramètres optimaux du random_search

Ainsi - avec ces paramètres - nous arrivons à ces résultats :

```
Temps d'apprentissage :  1.336912473895062e-05
Accuracy en apprentissage: 0.9973190348525469
Erreur en apprentissage : 0.002680965147453085
Accuracy en test: 0.7395833333333334
Erreur en apprentissage : 0.26041666666666663
```

FIGURE 6 – Paramètres optimaux du random_search

Pour améliorer nos résultats, nous cherchons non plus UN arbre (profond ici), mais UN ENSEMBLE d'arbres peu profonds. Pour se faire, nous utilisons la méthode du Boosting avec Adaboost : celui-ci va toujours apprendre sur différentes versions des données en dirigeant l'apprentissage et en se focalisant sur les données mal classées à l'étape précédente.
Le BOOSTING se fait sur des apprenants faibles (weak learner) qui sont des modèles qui font juste un peu mieux que le hasard, ce qui est notre cas avec notre arbre de décision. Il va les combiner pour en faire un ensemble performant (strong classifier). En bouclant sur le nombre d'arbres présents dans l'Adaboost - ce faisant pour des arbres de profondeur 1 et 5 (donc peu profonds) - nous gagnons en accuracy :

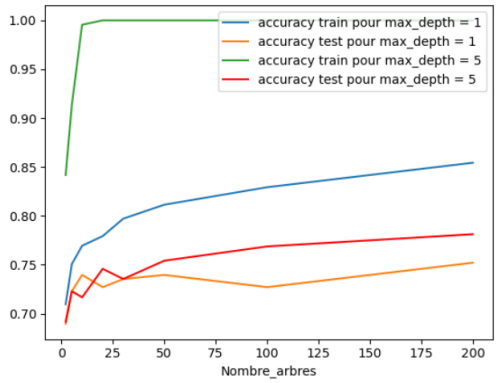


FIGURE 7 – Adaboost - Decision Trees

On peut mesurer l'importance des variables en récupérant leurs poids : les exemples les plus difficiles à classer sont plus importants donc les poids les plus forts représentes les variables les plus importantes. On peut voir ici nos features, classées par ordre d'importance.

	Feature	Important Variables
1	volatile acidity	0.113951
4	chlorides	0.112886
6	total sulfur dioxide	0.107854
9	sulphates	0.103885
10	alcohol	0.090760
8	pH	0.087782
0	fixed acidity	0.087550
7	density	0.085772
3	residual sugar	0.071735
2	citric acid	0.069198
5	free sulfur dioxide	0.068626

FIGURE 8 – Poids des variables - Decision Trees

Biais et Variance :
En orientant l'apprentissage à chaque étape, le boosting agit sur le biais ; en les combinant, il agit sur la variance.

De cette partie, on retient à l'optimum : entre 100 et 200 arbres, de profondeur 5 (peu profonds), d'une accuracy en test de 78.12% pour un temps d'inférence de 0.000358.

2 | Classification multi-classes : MLP et Random Forest

Pour pourvoir effectuer une classification multi-classes - donc à trois classes - nous allons passer de 6 classes présentes à trois classes.
Pour cela, nous allons nous aider des quantiles : nous allons passer à une classe 0 toutes les observations en dessous de Q1, à 1 celles entre Q1 et Q2, et à 2 celles au-dessus de Q3. Voici ce que l'on a après manipulation :

```
2    855
1    681
0     63
Name: quality, dtype: int64
```

FIGURE 9 – Classes de 0 à 2

Nous remarquons un déséquilibre des classes, que nous rectifions à l'aide de SMOTE().
Nous allons désormais utiliser un modèle plus robuste : un réseau de neurones à une couche cachée de 50 neurones, avec une fonction d'activation de sortie softmax qui va nous permettre une classification multi-classes.

```
1    593
2    593
0    593
Name: quality, dtype: int64
```

FIGURE 10 – Classes de 0 à 2, équilibrées

Nous faisons tourner notre modèle sur deux types de données :
- normalisées équilibrées ;
- normalisées non équilibrées.

Comparons alors les accuracy et temps d'inférence :

```
Temps d'inférence 0.00030754522502656866
Accuracy en apprentissage: 0.5031
Accuracy en test: 0.5021
```

FIGURE 11 – MLP sur données normalisées équilibrées

```
Temps d'inférence 0.0003258879425485183
Accuracy en apprentissage: 0.7194
Accuracy en test: 0.6937
```

FIGURE 12 – MLP sur données normalisées non équilibrées

Nous remarquons que nous avons une accuracy en test nettement meilleure sur nos données normalisées non équilibrées. Cela s'explique avec les matrices de confusion :

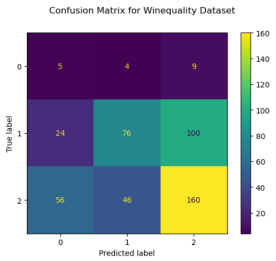


FIGURE 13 – Matrice de confusion du MLP sur données normalisées équilibrées

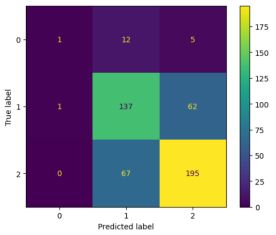


FIGURE 14 – Matrice de confusion du MLP sur données normalisées non équilibrées

Sur les données équilibrées, la classe 0 est présente mais très mal classée, tandis que l'autre cas ne classe qu'une seule observation de classe 0 correctement. L'impact sur l'accuracy y est peu visible puisque la classe est largement minoritaire !

Pour monter notre accuracy, nous allons faire ici du BAGGING, qui lui prend des apprenants forts comme notre MLP, contrairement à la partie précédente où le BOOSTING prenait des apprenants faibles comme notre arbre de décision.
Le bagging fonctionne sur le principe de l'OUT OF BAG (exemple non pris en compte à l'étape précédente) qui mesure l'erreur directement durant l'apprentissage, sans avoir à passer par une validation croisée.
En bouclant sur le nombre de réseaux, nous obtenons :

	estimators	accuracy_train	accuracy_test	temps_inference
0	1	0.529937	0.545833	0.000014
1	2	0.683646	0.668750	0.000389
2	5	0.687221	0.664583	0.000701
3	10	0.711349	0.689583	0.001912
4	50	0.701519	0.675000	0.008794
5	100	0.709562	0.687500	0.020007
6	200	0.715818	0.691667	0.039385

FIGURE 15 – Bagging sur MLP

Biais et variance :
Le bagging réduit avant tout la variance. Il faut donc que les modèles de base aient un biais faible. La variance est limitée (comme le sur-apprentissage) car augmenter les modèles n'aboutit pas au sur-apprentissage. En pratique, une centaine suffit, mais on peut l'ajuster à l'étude. Nous contrôlons l'overfitting du MLP avec de l'early stopping et de la validation via la base de test.
De cette partie, on retient à l'optimum : entre 100 et 200 estimateurs (MLP) sur des données normalisées non équilibrées, d'une accuracy en test de 68.9%, pour un temps d'inférence de 0.022230.

Nous pouvons aussi un random forest, qui est une combinaison d'arbres profonds. En effectuant un random_search pour trouver le nombre optimal de forêts et la profondeur de leurs arbres, nous arrivons à :

```
Results from Random Search

The best estimator across ALL searched params:
RandomForestClassifier(max_depth=85)

The best score across ALL searched params:
0.7774903907751441

The best parameters across ALL searched params:
{'n_estimators': 100, 'max_depth': 85, 'criterion': 'gini'}
```

FIGURE 16 – random$_search sur Random Forest$

Avec ces résultats, nous faisons un random forest avec des données équilibrées et non équilibrées. Voici les résultats :

```
Accuracy en apprentissage:  1.0
Accuracy en test:  0.7645833333333333
Temps d'inférence: 0.0002495539412102175
```

FIGURE 17 – Random Forest sur données non équilibrées

```
Accuracy en apprentissage:  1.0
Accuracy en test:  0.75625
Temps d'inférence: 0.0003262066862431885
```

FIGURE 18 – Random Forest sur données équilibrées

Cette fois-ci, le RF est plus efficace sur les données équilibrées.
Les variables importantes sont très similaires à celle du boosting sur decision trees optimal :

	Feature	Important Variables
1	volatile acidity	0.146983
10	alcohol	0.134722
6	total sulfur dioxide	0.119898
9	sulphates	0.117635
3	residual sugar	0.073572
8	pH	0.071186
7	density	0.070969
2	citric acid	0.067303
4	chlorides	0.066823
5	free sulfur dioxide	0.066527
0	fixed acidity	0.064381

FIGURE 19 – Poids des variables - Random Forest

De cette partie, on retient à l'optimum le Random Forest sur données équilibrées, similaire au boosting sur decision trees, avec une accuracy en test de 77.5%, et un temps d'inférence de 0.000235.

Ainsi, le modèle le plus performant pour cette classification de qualité de bière est un Random Forest, sur des données équilibrées à 3 classes, avec 100 estimateurs et une profondeur d'arbre maximum de 85, donc arbres peu biaisés car profonds. Le sur-apprentissage est évité avec le bagging. Le paramétrage est simple et l'importance des variables est mesurée.

TD - Ensemble Learning

- Pensez à mesurer les temps d'apprentissage et d'inférence de chaque solution

A. Base de données

La base de données « beer quality » comporte 1600 exemples décrits par 11 caractéristiques quantitatives. L'objectif est d'évaluer, sachant ces variables, la qualité d'une bière, évaluée de 1 (sans commentaire !) à 10 (particulièrement excellente).

Charger la base et la séparer en deux : la matrice X des observations et le vecteur y des labels. Analyser rapidement les variables prédictives X et la variable à prédire y (quality).

Diviser la base de données en deux sous-ensembles d'apprentissage et de test (70/30).

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

```
Classe : 3 Fréquence : 10
Classe : 4 Fréquence : 53
Classe : 5 Fréquence : 681
Classe : 6 Fréquence : 638
Classe : 7 Fréquence : 199
Classe : 8 Fréquence : 18
```

B. Classification binaire

1. Créer une nouvelle variable quantitative ybin à deux modalités :

- 0 : mauvaise qualité : y < m
- 1 : bonne qualité : y >= m

en fonction de la médiane m de la variable y.

```
1    855
0    744
Name: quality, dtype: int64
```

2. Optimiser rapidement un arbre de décision pour réaliser la classification en faisant une recherche aléatoire (random search).

```
 Results from Random Search

 The best estimator across ALL searched params:
 DecisionTreeClassifier(criterion='entropy', max_depth=20)

 The best score across ALL searched params:
 0.7265334721332478

 The best parameters across ALL searched params:
 {'min_samples_split': 2, 'max_depth': 20, 'criterion': 'entropy'}
```

Lorsqu'on lance plusieurs fois la cellule, la combinaison de tuple paramètres est initié aléatoirement.

Source : https://www.kaggle.com/general/212697

- max_depth : la profondeur maximale de l'arbre. Si None, les noeuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins de min_samples_split échantillon

=> La "maximum tree depth" qui signifie profondeur maximale de l'arbre, il s'agit d'arrêter le développement de l'arbre une fois qu'il a atteint une certaine profondeur, cela évitera que l'arbre construise des branches avec trop peu d'exemples et donc permettra d'éviter un sur apprentissage.

- min_samples_split : le nombre minimum d'échantillons requis pour diviser un noeud interne

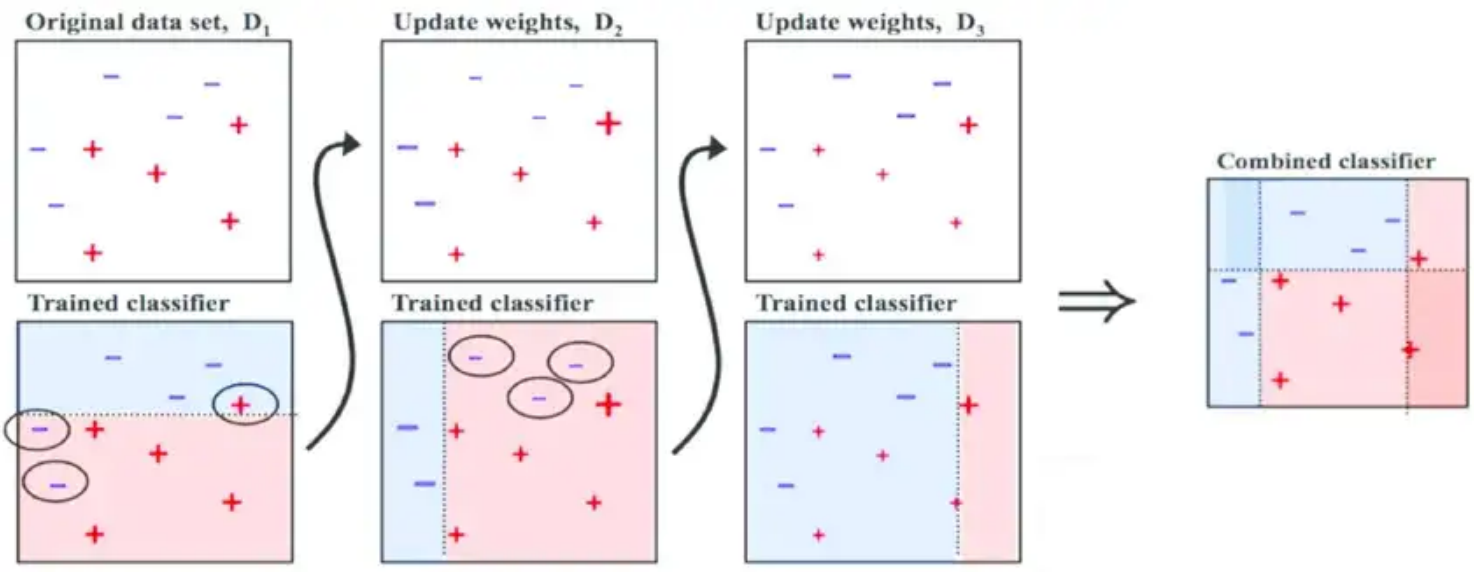
=> Le "minimum sample split" ou encore nombre d'exemples minimum pour un split consiste à ne pas splitter une branche si la décision concerne trop peu d'exemples. Cela permet également d'empêcher le surapprentissage.

- Entropie ou Gini ? https://quantdare.com/decision-trees-gini-vs-entropy/

```
Temps d'apprentissage :  7.150213669409594e-06
Accuracy en apprentissage: 0.9991063449508489
Erreur en apprentissage : 0.0008936550491510653
Accuracy en test: 0.7645833333333333
Erreur en apprentissage : 0.23541666666666672
```

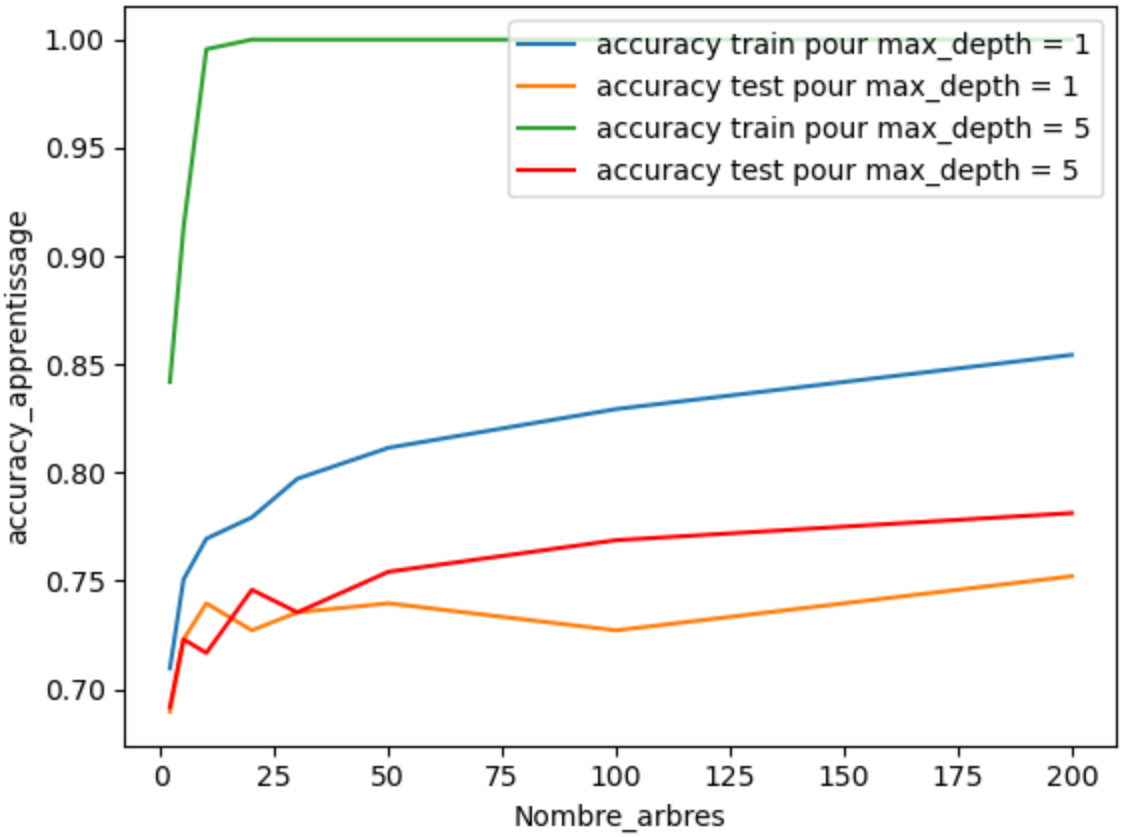
3. Entraîner un ensemble d'arbres de décision « faibles » (peu, voire très peu profonds) à l'aide de l'algorithme AdaBoost . Analyser les performances en fonction des différents paramètres :

- En particulier, tracer les courbes accuracy en fonction de n_estimators pour max_depth = 1, en apprentissage et en test.
- Tracer la courbe accuracy en fonction de n_estimators pour max_depth = 5, en apprentissage et en test.
- Peut-on mesurer l'importance d'une caractéristique dans la décision AdaBoost ? Expliquez. Afficher les variables par ordre d'importance.
- Conclure sur le biais et la variance de l'algorithme.



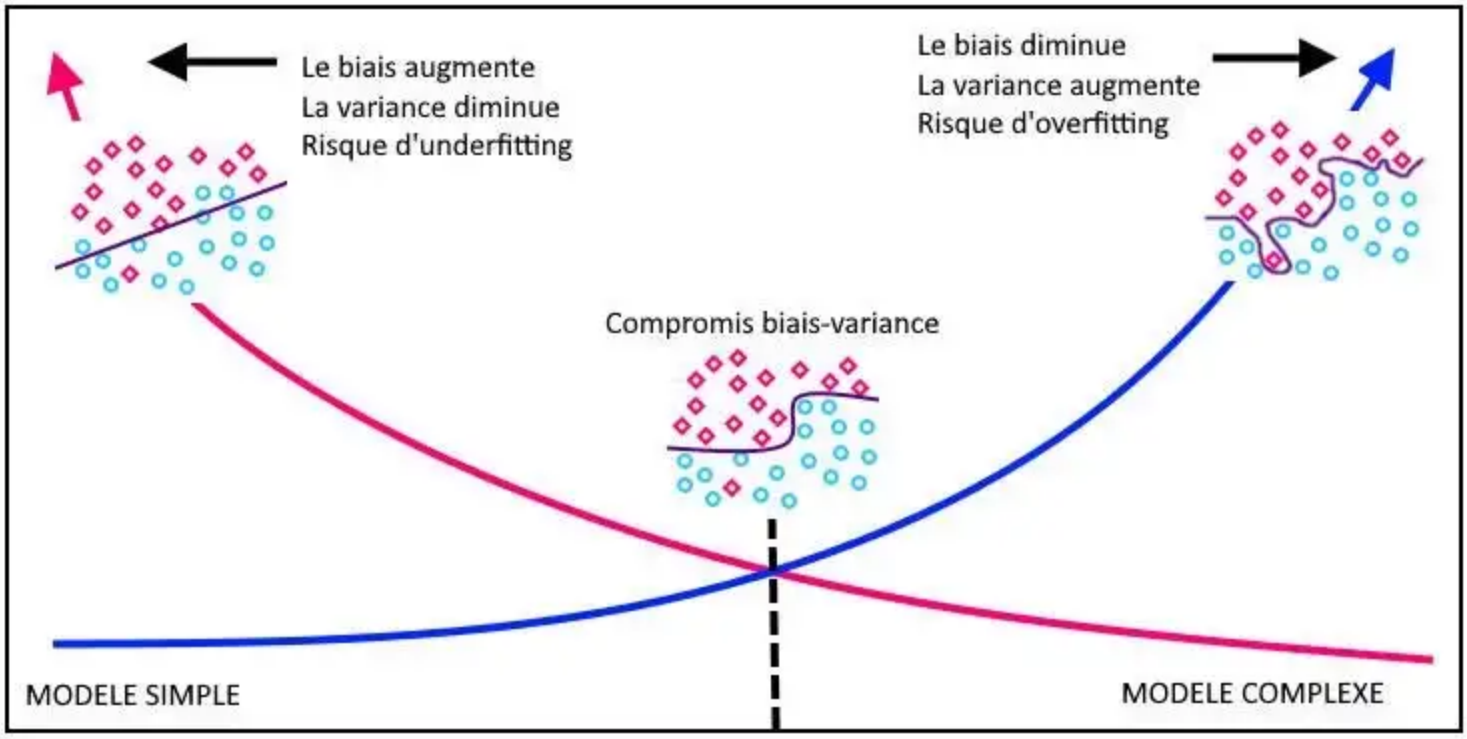
	Nombre_arbres	accuracy_apprentissage	accuracy_test	temps_inference
0	2	0.714030	0.681250	0.000007
1	5	0.737265	0.697917	0.000007
2	10	0.764075	0.720833	0.000015
3	20	0.776586	0.745833	0.000037
4	30	0.785523	0.725000	0.000056
5	50	0.806971	0.733333	0.000080
6	100	0.831099	0.737500	0.000156
7	200	0.856122	0.739583	0.000308

	Nombre_arbres	accuracy_apprentissage	accuracy_test	temps_inference
0	2	0.841823	0.691667	0.000034
1	5	0.913315	0.722917	0.000028
2	10	0.995532	0.716667	0.000038
3	20	1.000000	0.745833	0.000074
4	30	1.000000	0.735417	0.000111
5	50	1.000000	0.754167	0.000177
6	100	1.000000	0.768750	0.000358
7	200	1.000000	0.781250	0.000760



Compromis biais et variance

Le biais est l'erreur du modèle et la variance est à quelle point notre modèle "colle" aux données d'entrainement. Une variance trop haute couplée à un biais très bas signifie généralement un sur-apprentissage et une mauvaise généralisation, il faut donc trouver un compromis entre les deux.



Au-delà d'un certain nombre d'arbres :

- plus d'améliorations notables.
- pas d'overfitting.

Adaboost se base sur le BOOSTING : toujours apprendre sur différentes versions des données. On dirige l'apprentissage en se focalisant sur les données mal classées à l'étape précédente.

Biais et variance.

En orientant l'apprentissage à chaque étape, boosting agit sur le biais ; en les combinant, il agit sur la variance. Arbres profonds peu biaisés => la combinaison réduit la variance individuelle.

En fin d'apprentissage, le poids des exemples « difficiles » augmente et l'algorithme se concentre sur ces exemples.

=> augmentation de la marge (les exemples difficiles sont ceux dont la marge est la plus faible.

On peut mesurer l'importance des variables en récupérant leurs poids : les exemples les plus difficiles à classe augmente donc les poids les plus forts représentes les variables les plus importantes.

Le BOOSTING se fait sur des apprenants faibles (weak learner) qui sont des modèle qui fait juste un peu mieux que le hasard. Le Boosting combine des weak learner de manière appropriée et permet de produire un ensemble performant (strong classifier), nettement meilleur que chaque modèle pris individuellement.

```
Temps d'inférence 0.0002641788649708166
Accuracy en apprentissage 1.0
Accuracy en test 0.75625
```

	Feature	Important Variables
1	volatile acidity	0.127885
4	chlorides	0.120800
9	sulphates	0.099618
10	alcohol	0.099212
6	total sulfur dioxide	0.091011
7	density	0.090682
8	pH	0.088309
0	fixed acidity	0.080547
5	free sulfur dioxide	0.077808
2	citric acid	0.070415
3	residual sugar	0.053713

C. Classification multiclasse

1. Créer une nouvelle variable quantitative ymulti discrète à 3 modalités : qualité basse (0), moyenne (1) ou élevée (2).

On se base sur les quartiles :

- Q1 => 25% à G, et 75% à D
- Q2 => 50% à G, et 50% à D
- Q3 => 75% à G, et 25% à D
- Q4 => dernière valeur

Ici, on va diviser en 3.

2. Déterminer les effectifs des différentes classes. Si nécessaire, équilibrer les données d'apprentissage par sous-

échantillonnage ou augmentation de données (SMOTE). Dans la suite, on présentera les résultats obtenus avec et sans équilibrage.

```
2    855
1    681
0     63
Name: quality, dtype: int64
```

On va donc utiliser la librairie imbalanced-learn.

```
2    587
1    587
0    587
Name: quality, dtype: int64
```

Partie 1

3. Entraîner un réseau de neurones à une couche cachée pour effectuer cette tâche de classification, avec early stopping sur la base de validation. L'optimiser rapidement en prenant soin d'éviter l'over-fitting. Etudier les matrices de confusion des réseaux optimaux dans le cas des données déséquilibrées puis, équilibrées. Conclure.

Un réseau de neurones a une couche cachée est un MPL.

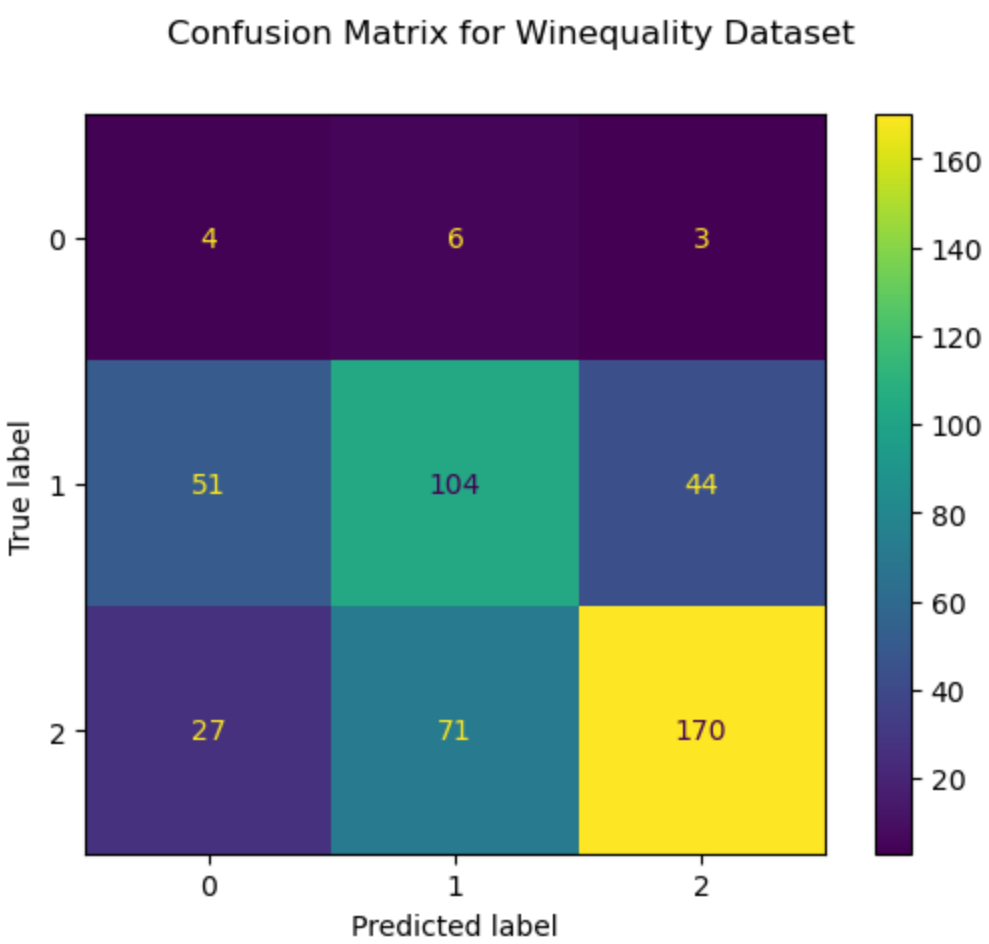
Pour nos données équilibrées :

```
'train_labels = to_categorical(y_train_equil)\ntest_labels = to_categorical(y_test)'
```

```
1119
1761
480
1119
1761
480
```

```
Temps d'inférence 0.0004577834385487926
Accuracy en apprentissage: 0.6712
Accuracy en test: 0.5792
```

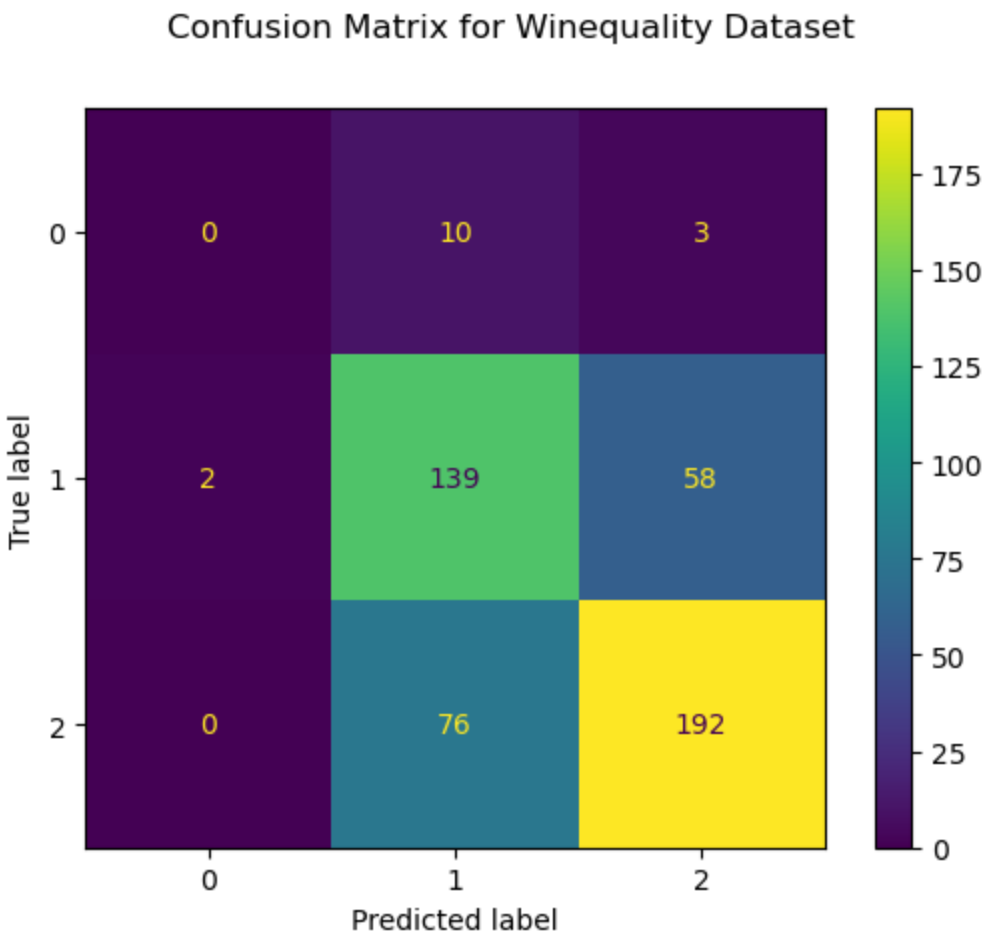
```
array([1, 0, 0, 2, 0, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 0, 1, 2, 1, 0, 0,
       0, 2, 0, 1, 0, 1, 2, 2, 1, 1, 1, 2, 1, 0, 2, 2, 2, 2, 1, 2, 2, 0,
       2, 2, 1, 0, 2, 2, 1, 2, 1, 2, 2, 1, 0, 2, 2, 2, 1, 1, 1, 1, 1, 2,
       2, 2, 1, 2, 2, 2, 1, 0, 0, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 1,
       2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 0, 0, 2, 1, 2, 1, 2, 2, 0, 1, 0,
       1, 2, 0, 1, 1, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 1, 0, 2, 2,
       1, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 0, 1, 2, 1, 1, 2, 2, 2, 0, 2,
       2, 1, 2, 2, 2, 2, 2, 1, 1, 0, 1, 1, 2, 2, 2, 2, 0, 2, 2, 2, 1, 0,
       1, 2, 1, 1, 2, 1, 1, 2, 2, 0, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 0, 0,
       0, 2, 1, 0, 2, 2, 2, 0, 2, 0, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       1, 2, 2, 2, 2, 2, 1, 2, 0, 2, 1, 1, 0, 2, 1, 0, 2, 2, 1, 2, 1, 2,
       1, 2, 1, 1, 0, 2, 0, 2, 2, 0, 1, 1, 1, 2, 1, 0, 2, 0, 1, 2, 2, 1,
       2, 2, 1, 2, 2, 0, 2, 2, 1, 1, 1, 2, 2, 2, 1, 2, 0, 0, 2, 2, 1, 1,
       1, 1, 2, 2, 0, 0, 2, 1, 1, 0, 2, 0, 1, 0, 2, 0, 2, 1, 2, 1, 0, 1,
       0, 2, 0, 2, 1, 2, 2, 2, 2, 1, 2, 0, 2, 0, 2, 1, 1, 1, 2, 1, 0, 1,
       0, 1, 1, 0, 2, 1, 2, 2, 0, 1, 2, 0, 1, 2, 1, 0, 2, 1, 2, 1, 2, 2,
       1, 2, 2, 2, 2, 1, 0, 1, 2, 2, 0, 1, 0, 1, 1, 2, 0, 1, 1, 1, 2, 1,
       1, 0, 0, 1, 2, 2, 0, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2, 0,
       0, 0, 2, 1, 2, 1, 0, 1, 0, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2,
       0, 1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 0, 2, 2,
       1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 2, 0, 2, 1, 0, 1, 2, 1,
       1, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 1, 0, 1, 0, 2, 1, 2], dtype=int64)
```



Pour nos données non-équilibrées :

```
Temps d'inférence 0.00040724237014184154
Accuracy en apprentissage: 0.7158
Accuracy en test: 0.6896

array([0, 1, 2], dtype=int64)

array([[2.58719549e-02, 6.05248511e-01, 3.68879534e-01],
       [4.36152462e-02, 4.44312869e-01, 5.12071885e-01],
       [2.45230795e-01, 6.63211311e-01, 9.15578944e-02],
       ...,
       [2.46912717e-04, 1.23163206e-02, 9.87436767e-01],
       [1.24091564e-02, 6.19616651e-01, 3.67974193e-01],
       [2.12629216e-02, 1.76790409e-01, 8.01946669e-01]])

array([1, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 1, 1, 1,
       1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1,
       2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2,
       2, 2, 1, 2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 1,
       2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 0, 2, 2, 2, 1, 2, 2, 1, 1, 1,
       1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2,
       1, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 2, 1, 2,
       2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1,
       1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2,
       1, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       1, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2,
       1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1,
       2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1,
       1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 0, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1, 1, 1,
       1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2,
       1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2,
       1, 1, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1,
```

```
1, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2,
1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2,
1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 1,
1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2], dtype=int64)
```

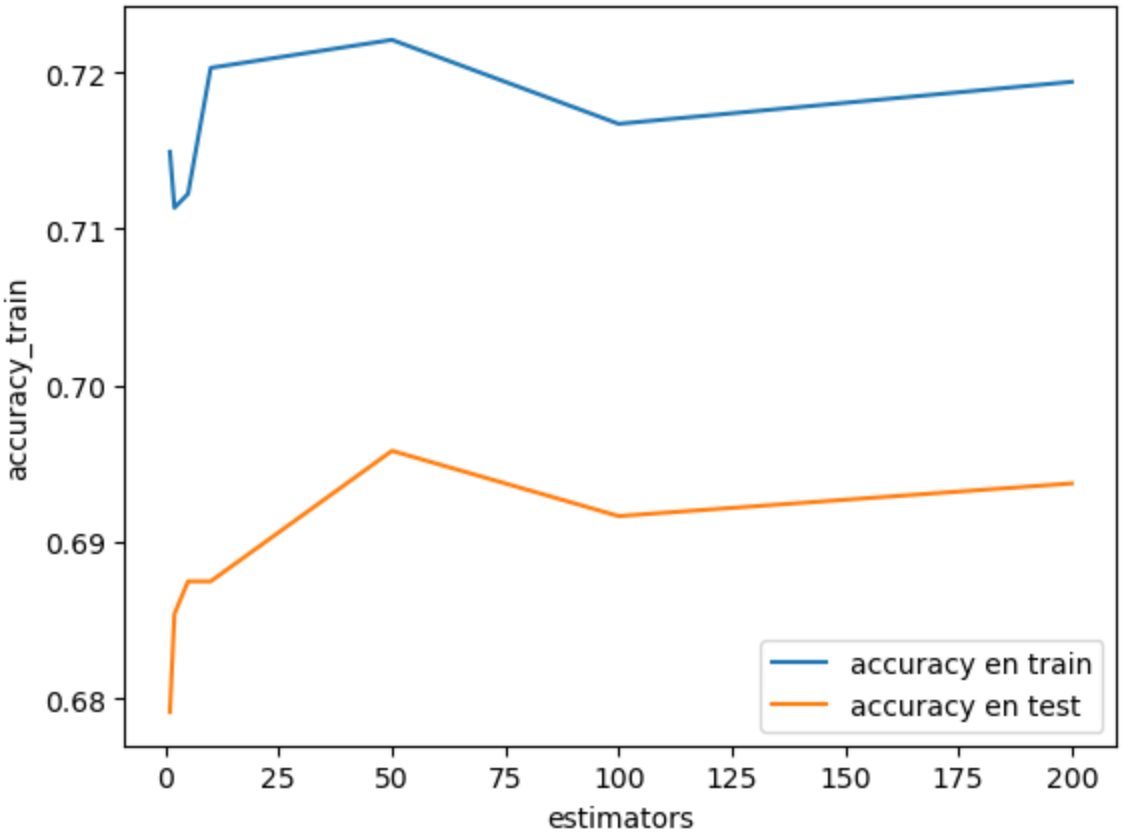


On optimise un réseau de neurones avec un GridSearch sur nos hyperparamètres comme le nombre de neurones dans notre couche cachée, le solveur...

4. Faire un bagging en utilisant comme classifieur de base le réseau de neurones.

- Tracer la courbe accuracy en fonction de n_estimators, en apprentissage et en test.
- Conclure sur le biais et la variance.

	estimators	accuracy_train	accuracy_test	temps_inference
0	1	0.714924	0.679167	0.000370
1	2	0.711349	0.685417	0.000739
2	5	0.712243	0.687500	0.001845
3	10	0.720286	0.687500	0.003916
4	50	0.722073	0.695833	0.018402
5	100	0.716711	0.691667	0.037326
6	200	0.719392	0.693750	0.075633



A la partie précédente et avec la classification biclasse, nous avions utilisé du BOOSTING avec des arbres de décisions.

Dans cette partie, nous faisons de la classification multiclasse avec du BAGGING : pour le bagging, nous avons besoin de modèles robustes de base, il n'est pas bon avec les apprenants faibles. C'est pour cela qu'ici, on préfère utiliser un MLP, plus robustes qu'un decision tree.

Principe de l'OUT OF BAG : Estimation de l'erreur OOB (out-of-bag) : mesure l'erreur directement durant l'apprentissage, sans avoir à passer par une validation croisée.

=> Sur-apprentissage (overfitting). Augmenter les modèles n'aboutit pas au sur-apprentissage. En pratique, une centaine suffit, mais on peut l'ajuster à l'étude.

=> Bagging peut s'appliquer à tout type de modèle.

=> Biais et variance. Biais bagging = biais du modèle de base. Le Bagging réduit avant tout la variance. Il faut donc que les modèles de base aient un biais faible.

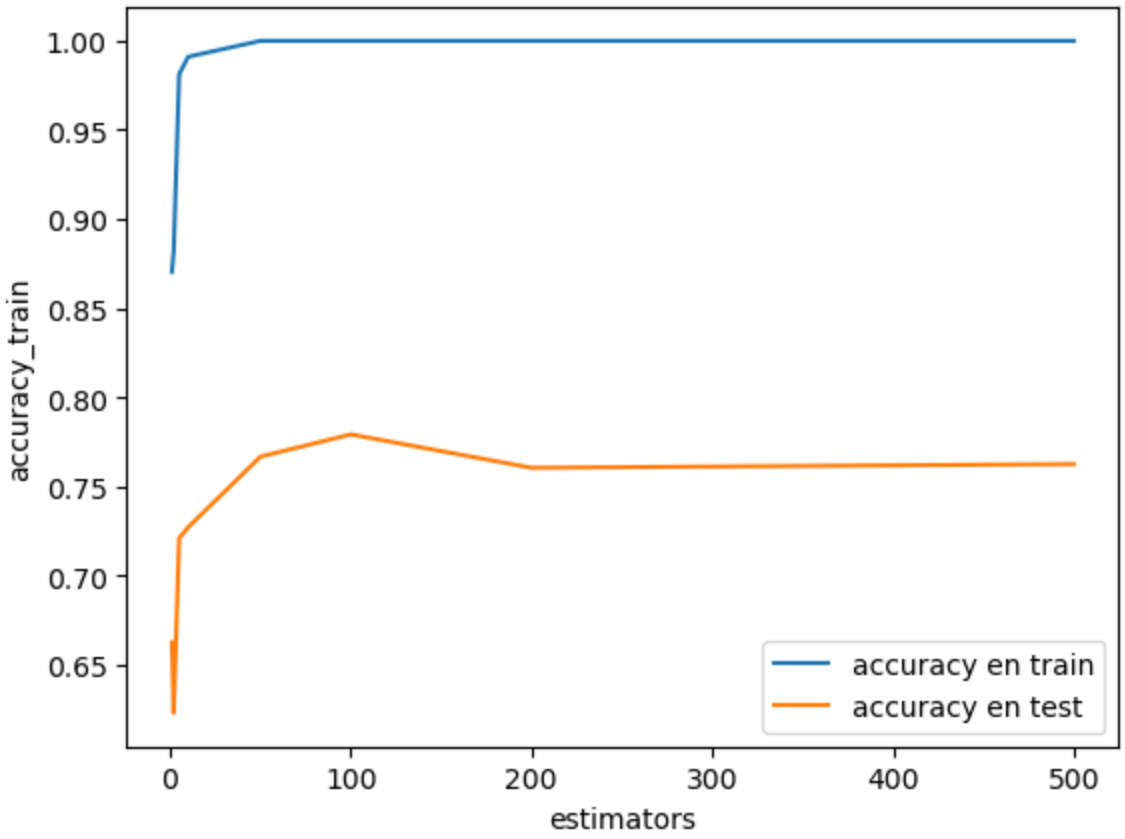
https://machinelearningmastery.com/strong-learners-vs-weak-learners-for-ensemble-learning/

Partie 2

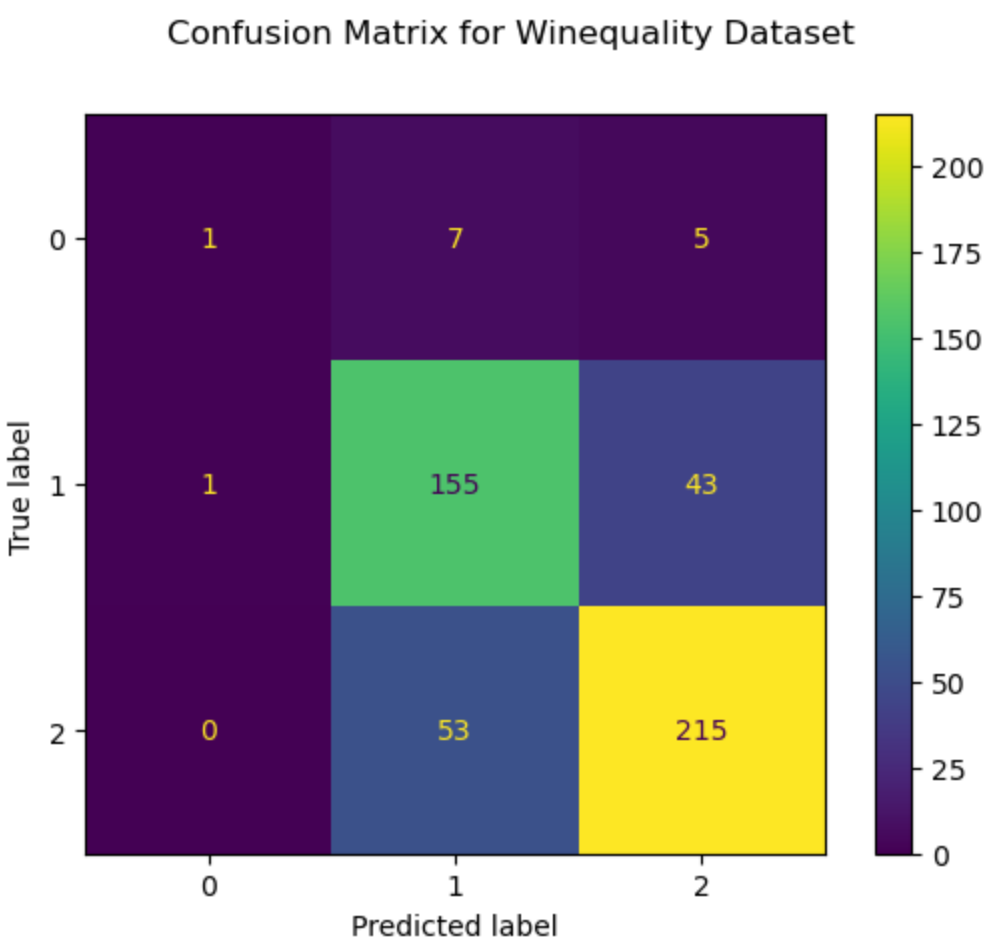
5. Entraîner une forêt aléatoire :

- Tracer la courbe accuracy en fonction de n_estimators pour max_depth = None, en apprentissage et en test.
- Faire une recherche aléatoire (random search) pour optimiser les paramètres max_depth et n_estimators. Choisir les paramètres optimaux et donner les performances en apprentissage et en test. Comparer avec les résultats obtenus précédemment (matrice de confusion sur les données déséquilibrées et équilibrées).
- Afficher les variables par ordre d'importance. Retrouve-t-on les mêmes variables qu'en B.3 ?
- Conclure sur le biais et la variance.

	estimators	accuracy_train	accuracy_test	temps_inference
0	1	0.870420	0.662500	0.000007
1	2	0.882038	0.622917	0.000007
2	5	0.981233	0.720833	0.000014
3	10	0.991063	0.727083	0.000029
4	50	1.000000	0.766667	0.000098
5	100	1.000000	0.779167	0.000200
6	200	1.000000	0.760417	0.000398
7	500	1.000000	0.762500	0.000986

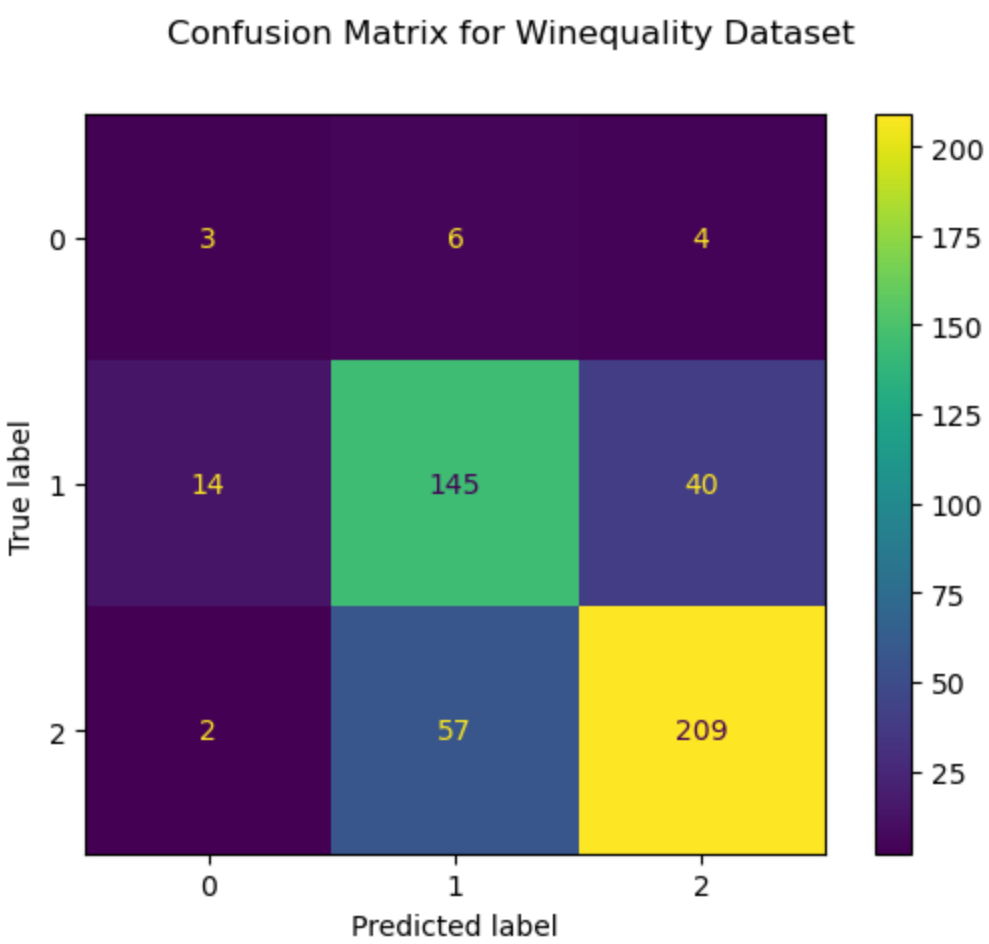



```
 Results from Random Search

 The best estimator across ALL searched params:
 RandomForestClassifier(max_depth=51)

 The best score across ALL searched params:
 0.7783752402306213

 The best parameters across ALL searched params:
 {'n_estimators': 100, 'max_depth': 51, 'criterion': 'gini'}

Accuracy en apprentissage:  1.0
Accuracy en test:  0.7729166666666667
Temps d'inférence: 0.00021116109273602006
```



	Feature	Important Variables
10	alcohol	0.173255
9	sulphates	0.114101
1	volatile acidity	0.113685
6	total sulfur dioxide	0.103415
7	density	0.094820
4	chlorides	0.078954
0	fixed acidity	0.075087
8	pH	0.068253
2	citric acid	0.061896
5	free sulfur dioxide	0.059116
3	residual sugar	0.057418

```
Accuracy en apprentissage:  1.0
Accuracy en test:  0.7395833333333334
Temps d'inférence: 0.00030423126868417584
```



	Feature	Important Variables
9	sulphates	0.144207
6	total sulfur dioxide	0.139527
10	alcohol	0.126566
1	volatile acidity	0.112758
5	free sulfur dioxide	0.080424
8	pH	0.075744
7	density	0.070882
4	chlorides	0.067949
2	citric acid	0.064626
0	fixed acidity	0.062112
3	residual sugar	0.055204

Random Forest :

Pour rendre le bagging (plus) efficace, il faut que les arbres :

1. Soient individuellement performants
2. De grande profondeur (biais faible)
3. Et surtout, très fortement différents les uns des autres

→ Utiliser RSM pour décorréler les échantillons

:) Bonnes performances en prédiction

:) Paramétrage simple (B, max_depth)

:) Pas de problème d'overfitting (on peut augmenter B)

:) Evaluation de l'erreur intégrée (OOB)

:) Mesure de l'importance des variables

MAIS :

:(Problème si nombre de variables pertinentes très faibles, dans l'absolu et relativement au nombre total de variables

:(Explicabilité réduite

NB :

Scaling is done to Normalize data so that priority is not given to a particular feature. Role of Scaling is mostly important in algorithms that are distance based and require Euclidean Distance.

Random Forest is a tree-based model and hence does not require feature scaling.

This algorithm requires partitioning, even if you apply Normalization then also> the result would be the same.

Pour exporter le notebook en pdf :

- Ouvrir le prompt anaconda
- installer la bibliothèque : pip install nbconvert[webpdf]
- taper : jupyter nbconvert --to webpdf --allow-chromium-download TP_MADAD_NOUAR.ipynb

OU jupyter nbconvert --to webpdf --allow-chromium-download TP_MADAD_NOUAR.ipynb

OU jupyter nbconvert --to webpdf --no-input TP_MADAD_NOUAR.ipynb

OU jupyter nbconvert TP_MADAD_NOUAR.ipynb --to pdf