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.

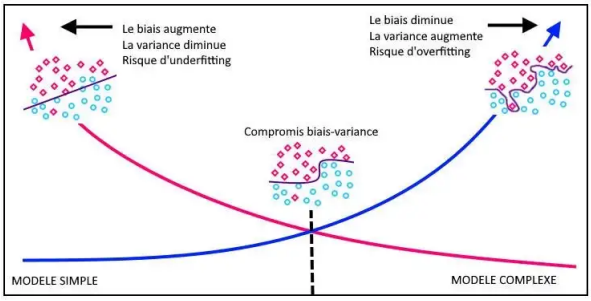

FIGURE 2 – Biais et variance

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 :

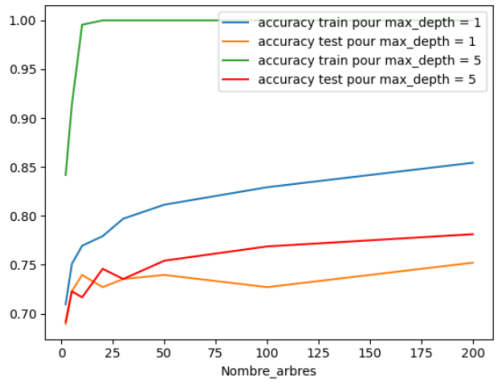

FIGURE 7 – Adaboost - Decision Trees

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 :

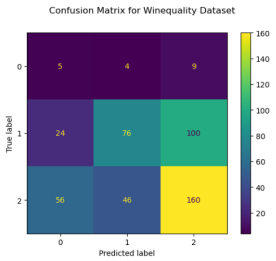

FIGURE 13 – Matrice de confusion du MLP sur données normalisées équilibrées

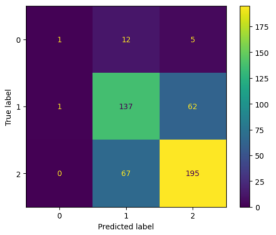

FIGURE 14 – Matrice de confusion du MLP sur données normalisées non équilibrées

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.