RAPPORT DE PROJET D'APRENTISSAGE AUTOMATIQUE

PRÉDIRE SI UNE EAU EST POTABLE OU NON

Benoît GINIES
benoit.ginies@student-cs.fr

Romé LEVENQ
rome.levenq@student-cs.fr

Victor GAUTHIER
victor.gauthier@student-cs.fr

November 7, 2021

Projet GitHub : https://github.com/benginies/Projet_Apprentisage_Automatique.git

1 CONTEXTE

L'eau est la ressource la plus essentielle, cruciale pour soutenir la vie de la plupart des créatures existantes et des êtres humains. Les organismes vivants ont besoin d'une eau de qualité suffisante pour poursuivre leur vie. Il existe certaines limites de pollution que les êtres humains peuvent tolérer. Le dépassement de ces limites peut entrainer de graves conséquences.

Le développement industriel rapide a entraîné une dégradation de la qualité de l'eau à un rythme inquiétant. De plus, les infrastructures, avec l'absence de sensibilisation du public, et les qualités hygiéniques moindres, affectent considérablement la qualité de l'eau potable. En effet, les conséquences d'une eau potable polluée sont très dangereuses et peuvent affecter gravement la santé, l'environnement et les infrastructures. Selon le rapport de l'Organisation des Nations Unies (ONU), environ 1,5 million de personnes meurent chaque année à cause de maladies liées à l'eau contaminée. Dans les pays en développement, on annonce que 80 % des problèmes de santé sont causés par l'eau contaminée. Cinq millions de décès et 2,5 milliards de maladies sont signalés chaque année. Un tel taux de mortalité est plus élevé que les décès résultant d'accidents, de crimes et d'attaques terroristes.

Il est donc très important de proposer de nouvelles approches pour analyser et, si possible, prédire la qualité de l'eau.

2 DATA

Le jeu de données utilisé est accessible sur Kaggle au lien [1] des références. Il est constitué de 3276 instances (n) comportant chacunes 10 valeurs de variables. La variable à expliquer est la potabilité de l'eau. Les 9 variables explicatives sont le pH, la dureté de l'eau, la solvabilité des solides dans l'eau, chloramine, sulfate, la conductivité, le carbone organique, le trihalométhane et la turbidité.

Pour commencer l'analyse des données, nous mettons en place un modèle composé d'arbres décisionnels pour évaluer l'importance respective de chacune des features, avant un quelconque début de traitement sur les données. On déduit des résultats (figure 2) que l'ensemble des variables sont à prendre en compte puisque nous n'observons pas de discontinuité forte.

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690300	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833605	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762125	8.114887	359.950170	481.792305	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196010	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

Figure 1: Description statistique du jeu de données par attributs

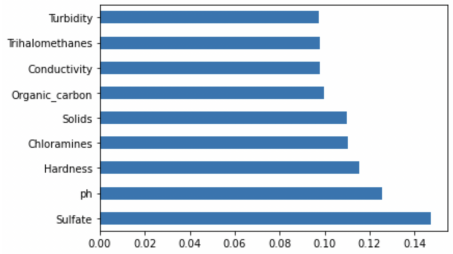

Figure 2: Importance des variables explicatives

3 DATA PREPROCESSING

3.1 DONNEES MANQUANTES

Des données manquent dans trois des features dont nous disposons : le ph pour lequel 15% des entrées manquent, le sulfate à 24% et les trihalométhanes où c'est 5%. Un tel volume ne permet pas de simplement ôter les données manquantes. Il convient donc de trouver une méthode de remplacement, qui permettra le bon fonctionnement des algorithmes d'apprentissage. Nous mettrons en place plusieurs stratégies de remplacement et évaluerons l'impact du choix de chacune de ces stratégies sur nos résultats:

- Remplacer par la moyenne sur tout le dataset
- Remplacer par la moyenne en divisant le dataset
- Remplacer par les plus proches voisins (en divisant ou non)

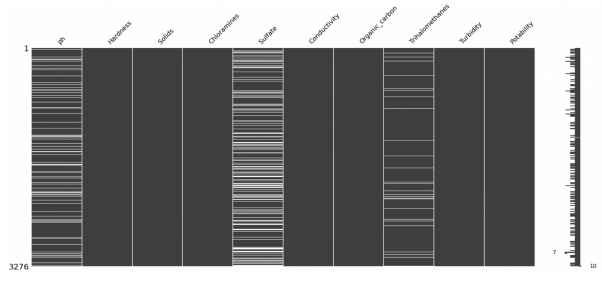

Figure 3: Répartition des valeurs manquantes dans les données

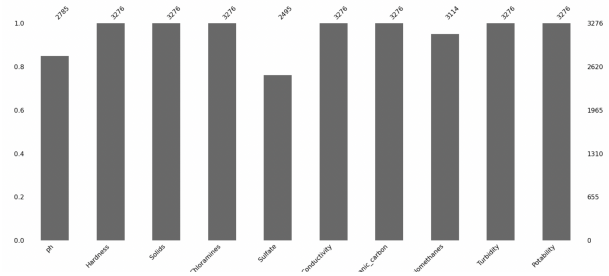

Figure 4: Nombre de valeurs par attributs

NB : La stratégie qui nous permettra d'obtenir les meilleurs résultats en terme d'accuracy est de remplacer par la moyenne en divisant le dataset par classe d'intérêt.

3.2 STANDARDISATION DES DONNEES

Avant de procéder à une exploration plus poussée des données, nous commençons par normaliser ces dernières en soustrayant la moyenne et en divisant par l'écart type (à l'aide de standard scaler). Il s'agit là d'un Z-Score (où se positionne la valeur par rapport à sa série).

3.3 EXPLORATION DES DONNEES

Parallèlement au traitement des données manquantes, il convient d'explorer les features de manière un peu plus profonde. Le calcul de la matrice de corrélation (voir Figure 5) nous indique qu'il n'existe pas de corrélation significative entre les variables étudiées : les coefficients de corrélation n'excèdent par 10% en valeur absolue. D'autre part, il n'existe pas de forte corrélation entre une variable quelconque et la variable cible (maximum 3% en valeur absolue).

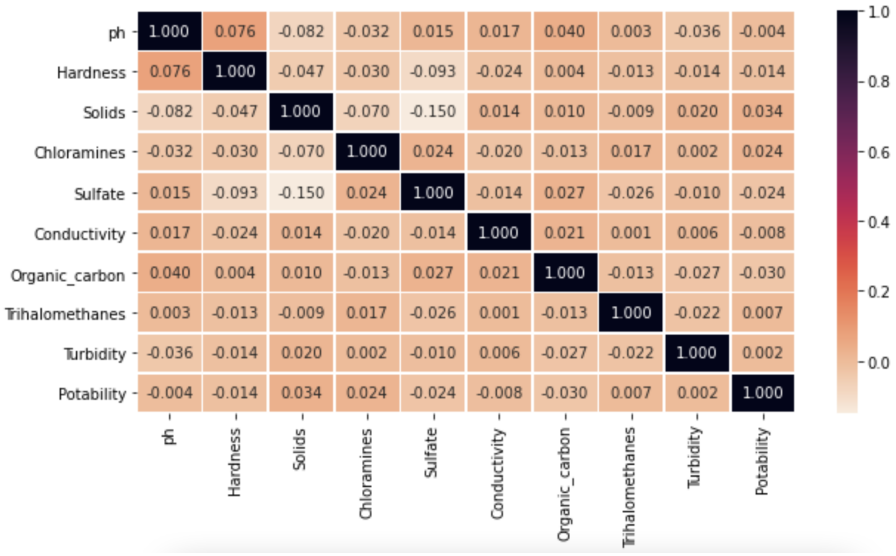

Figure 5: Description statistique du jeu de données par attributs

Les deux classes que nous étudions (potable ou non potable) sont représentées de manière relativement équilibrées (40% des instances pour la première, 60% pour la deuxième). Il n'est donc pas nécessaire d'équilibrer artificiellement les jeux de données avant traitement.

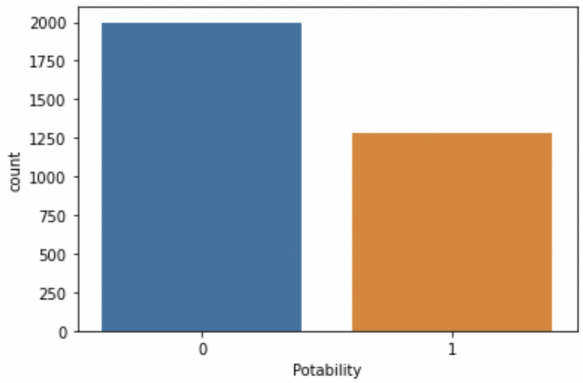

Figure 6: Répartition des instances potables et non potables

Étant donné que le nombre de variables (il y en a neuf) n'est pas trop grand, et qu'en parallèle le nombre d'instances (il y en a 3276) n'est pas non plus limitant, on peut en pratique se passer d'une réduction de dimensions. Cependant nous en avons pratiqué une tout de même, afin d'en apprendre davantage sur les données. En faisant sortir autant de directions principales que possible (soit 9), on observe que chacune d'entre elles apportent une information supplémentaire,

en termes de pourcentage de variance totale expliquée, un peu supérieure à 10%. Il n'y en a donc pas de direction prépondérante par rapport aux autres. Cela vient simplement illustrer plus pratiquement l'observation que nous avions faite : les variables sont globalement décorrélées. Dans la suite nous faisons le choix de ne pas appliquer de réduction de dimension aux variables.

4 MODELES

Pour la prédiction, nous avons testé plusieurs algorithmes, comparé les résultats puis optimisé les paramètres de ceux qui nous donnaient la meilleure accuracy. Les modèles utilisés sont présentés très rapidement ci-dessous par soucis de clarté.

4.1 Support Vector Machine (SVM)

Les machines à vecteurs de support pour la classification consistent à déterminer un hyperplan et une marge de décision afin de séparer les données en instances positives et en instances négatives. Il s'agit d'un problème dual de Lagrange.

4.2 Arbres de décision

Un arbre de décision représente un ensemble de choix. Il est construit en séparant à chaque étape l'espace selon un attribut et une valeur de cet attribut dite "de séparation". Le principal inconvénient est que les arbres de décision ne sont pas stables. C'est pourquoi nous emploierons plutôt les méthodes de bagging et de random forest ci-dessous.

4.3 Bagging

Le bagging est une méthode de bootstrapping où plusieurs datasets sont créés à partir de l'original. Un modèle est ensuite construit sur chaque dataset puis le modèle final est l'agrégat de l'ensemble des modèles construits (vote par majorité dans le cas de la classification).

Le bagging est traditionnellement utilisé sur les arbres de décisions mais il est applicable à toute autre méthode. Nous l'utiliserons aussi sur l'algorithme des k plus proches voisins (KNN).

4.4 Random Forest

La méthode de Random Forest repose sur la méthode de bagging sur arbres décisionnels. A chaque étape, les attributs pouvant séparer l'espace des instances sont pre-sélectionnés aléatoirement. Cela présente l'avantage que les différents arbres construits sont décorrélés les uns des autres (ce n'est pas le cas pour la méthode de bagging).

4.5 Boosting

L'algorithme de boosting utilisé dans ce projet est AdaBoost. Il consiste à construire progressivement des modèles dits faibles en les corrigeant avec les erreurs des modèles faibles précedemment construits. Il est donc applicable à une multitude de modèles.

Nous l'utiliserons ici sur les arbres de décision.

4.6 Stacking

Le principe de stacking consiste à créer plusieurs modèles ayant chacun bien modélisé une partie du problème et non sa totalité. En agrégeant les résultats de ces sous-modèles, on obtient un modèle final performant sur l'ensemble du problème.

5 EVALUATION

5.1 Evaluation Standard

On rassemble les résultats (accuracy) pour chaque modèle sur un set de test (Test_Score) et sur un set d'entraînement (Train_Score). Ils sont présentés dans la figure 7. On observe des scores élevés sur le set de test mais on observe aussi que certains modèles (RandomForest et Bagging with Three) ont un score de 100 sur le set d'entraînement. Cela traduit

	Models	Test_Score	Train_Score
9	Random Forest with Grid	83.38	100.00
2	Random Forest	83.23	100.00
8	Stacking	80.95	81.15
5	Bagging with Tree	80.03	100.00
6	Decision Tree with grid	78.81	81.34
1	Decision Tree	78.20	84.96
7	AdaBoost	76.68	77.94
4	Bagging with SVC	73.32	72.82
0	Support Vector Machine	73.17	72.29
3	Bagging with KNN	64.79	76.22

Figure 7: Résultats (Accuracy) de chaque modèle sans éviter le sur-apprentissage

	Models	Test_Score	Train_Score
9	Random Forest with Grid	84.60	92.67
2	Random Forest	82.47	100.00
5	Bagging with Tree	81.10	99.66
8	Stacking	80.95	81.15

Figure 8: Résultats (Accuracy) de chaque modèle après avoir réduit le sur-apprentissage

un sur-apprentissage de ces modèles. Pour réduire ce sur-apprentissage et étant donné que le dataset du projet est déterminé, on cherche à moins entraîner notre modèle.

Pour RandomForest with Grid, on avait comme nombres d'arbres 400 et dans nos choix entre 200 et 2000 (par pas de 200). On a donc réduit la recherche entre 0 et 200 (par pas de 20). On trouve alors de nouveaux paramètres avec un nombre d'arbres égal à 140. Pour Bagging with Tree, on reduit le nombre d'arbres également (au préalable 100). On observe cependant en itérant qu'il est impossible de non overfitter (avec deux arbres, le score sur le set d'entraînement est déjà proche de 100). Cependant, on remarque quand même une amélioriation du score sur le set de test.

5.2 Précision des résultats par Cross Valuation

Etant donné que nous ne disposons que d'un jeu d'entraînement ici, il nous a finalement semblé logique de ne pas distinguer un jeu de données test plutôt qu'un autre parmi l'historique. Nous avons donc décidé d'affiner notre évaluation des modèles en couplant un grid search à une validation croisée sur l'ensemble des données d'apprentissage. Nous avons donc été en mesure d'obtenir des résultats de précision moyennés et plus fidèles aux performances réelles des algorithmes. Afin de pouvoir comparer les imputers, nous avons aussi effectué une validation croisée par type d'imputer.

	Models	Score
3	RF, sep Mean imputer	0.8003
16	Stacking, sep Mean imputer	0.7979
11	AB, sep Mean imputer	0.7518
1	RF, sep KNN imputer	0.7084
2	RF, Mean imputer	0.6773
0	RF, KNN imputer	0.6758
13	GNB, sep KNN imputer	0.6266
8	AB, KNN imputer	0.6245
10	AB, Mean imputer	0.6239
12	GNB, KNN imputer	0.6230
14	GNB, Mean imputer	0.6211
9	AB, sep KNN imputer	0.6205
6	SVC, Mean imputer	0.6199
15	GNB, sep Mean imputer	0.6196
4	SVC, KNN imputer	0.6172
7	SVC, sep Mean imputer	0.6098
5	SVC, sep KNN imputer	0.6098

	Models	Score
0	RF, sep Mean imputer	0.8003
4	Stacking, sep Mean imputer	0.7979
2	AB, sep Mean imputer	0.7518
3	GNB, sep Mean imputer	0.6196
1	SVC, sep Mean imputer	0.6098

	Models	Score
0	RF, Mean imputer	0.6773
2	AB, Mean imputer	0.6239
3	GNB, Mean imputer	0.6211
1	SVC, Mean imputer	0.6199

	Models	Score
0	RF, sep KNN imputer	0.7084
3	GNB, sep KNN imputer	0.6266
2	AB, sep KNN imputer	0.6205
1	SVC, sep KNN imputer	0.6098

	Models	Score
0	RF, KNN imputer	0.6758
2	AB, KNN imputer	0.6245
3	GNB, KNN imputer	0.6230
1	SVC, KNN imputer	0.6172

Figure 9: Résultats (Accuracy) pour différents imputers, en validation croisée et grid search

Les résultats que nous avons obtenus étant assez proches de 60% (ce qui correspond à la taille de la classe non potable), nous avons pris le temps de vérifier que nous n'avions pas majoritairement construit des programmes qui renvoient "non potable" dans tous les cas. En calculant les scores de précision séparément sur les deux classes, nous avons balayé ce doute.

Il est intéressant de constater que globalement, c'est l'algorithme de Random Forest qui obtient les meilleurs résultats (ce qui vient appuyer les observations précédentes). D'une manière similaire, l'imputer en moyennes séparées par classe permet d'obtenir les meilleurs scores. L'approche que nous avons suivie jusqu'ici en sort donc justifiée.

L'extraction de l'importance relative des données dans le meilleur Random Forest que nous avons obtenu nous apprend que celui-ci regarde en priorité la variable sulfate (feature importance de 34%), puis la variable pH (22%), et enfin, de manière assez égale, les autres variables (entre 3.5% et 8%).

6 CONCLUSION

Tout d'abord, l'algorithme de Random Forest optimisé par exploration des meilleurs hyper paramètres donne les meilleurs résultats (83% d'accuracy). On observe plus globalement que les algorithmes constitués d'arbres de décision sont plus efficaces que les SVM ou encore le KNN (malgré le bagging). Il est assez rare que l'AdaBoost soit moins performant qu'un arbre de décision basique puisqu'il est considéré comme un des meilleurs classificateurs "déjà prêt".

Concernant le problème général du projet, on observe qu'on arrive assez bien à expliquer la potabilité de l'eau grâce à nos données (environ 85%). Cependant cette précision n'est pas suffisante pour une utilisation concrète. Il serait intéressant d'augmenter le jeu de données pour améliorer cette précision ou alors essayer de trouver d'autres variables pouvant expliquer la potabilité de l'eau.Les résultats trouvés sont tout de même cohérents avec la réalité puisque par exemple les sulfates, que nous trouvons comme variable la plus explicative, provoquent des effets laxatifs chez l'homme lorsqu'ils sont très concentrés dans l'eau. Ils peuvent alors provoquer une forte déshydration.

Nous avons ici observé les paramètres physico-chimiques de l'eau pour expliquer sa potabilité mais il serait également intéressant d'avoir des données sur la présence de nitrates et de fluors massivement utilisés dans les engrais et pesticides d'après le Centre d'information sur l'eau (CIEAU) (voir référence [2]). En effet, l'agriculture intensive est aujourd'hui un pollueur aquatique au moins au même titre que les usines ou autres utilisateurs de produits chimiques.

Les substances toxiques devraient aussi être prises en compte puisqu'elles sont nocives à très faible dose: les micropolluants tels que l'arsenic, le cyanure, le chrome, le nickel, le sélénium ainsi que certains hydrocarbures sont soumis à des normes très sévères à cause de leur toxicité.On aurait alors un problème beaucoup plus complexe avec plus de features et donc un besoin d'avoir beaucoup plus de données (n»d). On pourrait aussi implémenter des algorithmes de réduction de dimensions comme la Principal Components Analysis (PCA).

References

[1] https://www.kaggle.com/artimule/drinking-water-probability

[2] https://www.cieau.com/espace-enseignants-et-jeunes/les-enfants-et-si-on-en-apprenait-plus-sur-leau-du-robinet/la-definition-de-leau-potable/