CentraleSupélec

Rapport projet
Water Potability

Apprentissage Automatique

Equipe Projet :
Martin Vey
Sylvain Muller

Enseignants :
Myriam Tami
myriam.tami@centralesupelec.fr
Gherbi elies
gherbi.elies@centralesupelec.fr

Table des matières

1 Introduction

Lors de ce cours d'apprentissage automatique, nous avons réalisé un projet qui consiste à prédire si une eau est potable ou non en fonction de certaines informations sur celle-ci. Il s'agit donc d'un problème de classification entre 2 classes, "eau potable" et "eau non potable".

Ce problème outre son intérêt académique est un problème réel, en effet l'accès à l'eau potable est une problématique pour de nombreuses personnes encore aujourd'hui. Dans de nombreux pays, l'accès à l'eau potable, qui est essentielle à la santé humaine n'est pas garanti.

Le problème ici est de pouvoir prédire si une autre est potable ou non, en ayant connaissance de 9 variables explicatives. Ce problème est un problème qui peut se résoudre en machine learning car même s'il existe une norme précise pour l'eau potable, les 9 variables explicatives ne sont pas suffisantes pour valider cette norme, il est donc bon d'utiliser un algorithme d'apprentissage automatique qui permettra de déduire du jeu de donnée une règle pour prédire si une eau est potable. Cela peut être utile pour savoir assez rapidement si une eau d'un cours d'eau est potable et en déduire s'il est utile d'investir dans un réseau d'assainissement des eaux.

Dans ce cadre, nous avons analysé les données misent à disposition et entraîné 3 algorithmes de classification pour nous permettre de prédire si une eau est potable grâce à ces 9 variables explicatives.

2 Les données

Les données mises à disposition pour ce problème sont issues du site de challenge en machine learning "Kaggle". La base de données est composée de 9 variables explicatives et d'une variable à expliquer qui est la potabilité de l'eau. Ces informations sont disponibles pour 3276 eaux différentes. Les 9 variables explicatives sont toutes des données numériques et sont les suivantes :

- Le Ph de l'eau
- La dureté de l'eau
- La solvabilité des solides dans l'eau
- La concentration en chloramine
- La concentration en sulfates
- la conductivité
- La concentration en carbone organique
- La concentration en trihalométhane
- La turbidité de l'eau.

2.1 Analyse des données

La première chose que l'on remarque lorsque l'on regarde les données plus attentivement est la présence de données manquantes pour les variables du Ph, de la concentration en sulfates et en trihalométhanes. Cela est un souci qu'il faudra résoudre pour traiter les données.

Une autre chose nous interpelle, c'est le nombre relativement faible des données que nous avons à notre disposition pour ce problème. En effet, pour ce type de problème il faut en général plus de 3000 échantillons de données pour pouvoir entraîner un algorithme d'apprentissage automatique. Il faudra donc veiller lors du pré-traitement des données de ne pas en laisser de côté au vu du faible nombre de donnée à notre disposition.

De plus, l'analyse de la matrice de corrélation des variables explicatives nous indique qu'il n'y a aucune corrélation évidente entre les features. Les valeurs absolues des coefficients de corrélation ne dépassent pas les 0,1. Les variables apportent donc seulement l'information qu'elle contiennent et ne viennent pas enrichir l'information des autres variables.

2.2 Pré-traitement des données

Afin de pouvoir entraîner des modèles sur nos données plusieurs actions doivent être mises en oeuvre sur celle-ci. Les principales actions que l'on a mises en place sont : la gestion des données aberrantes et des données manquantes, le rééquilibrage des classes et la mise à l'échelle des différents features. D'autres actions ont été envisagées mais n'ont pas été mises en place comme le "feature engineering". Les prétraitements ont été appliqués aux données dans l'ordre suivant :

— Gestion des données aberrantes
— Séparation données d'entrainement et de test (85%/15%)
— Standardisation
— Gestion des données manquantes
— Rééquilibrage des classes dans le jeu d'entrainement

2.2.1 Gestion des données aberrantes

Dans le but d'entraîner un modèle sur les données les plus représentatives de la réalité nous avons cherché à éliminer les données aberrantes de notre base de données. Pour détecter la présence de données aberrante nous avons utilisé la méthode de l'intervalle inter-quartile. Nous remarquons ainsi un certain nombre de données qui apparaissent comme "aberrantes", cependant en affichant les distributions des données selon chaque feature nous remarquons que celles-ci sont gaussiennes. Il n'était donc pas évident que ces données constituaient une aberration. Par ailleurs, la suppression d'outliers n'avait pas d'effet significatif sur la performance de nos algorithme. Nous avons donc décidé de ne pas supprimer les outliers.

Néanmoins, lors d'analyses plus avancées nous avons remarqué que certaines de nos données étaient sûrement mal labellisées. En effet, certaines données sont labellisées comme potable alors que leur pH est inférieur à 6,5 qui est la limite basse de la norme du pH pour de l'eau potable établie par l'OMS. Nous n'avons pas relabellisé ces données pour autant, en estimant que nous souhaitions apprendre la définition de l'eau potable selon ce jeu de données sans se préoccuper de la définition utilisée par l'OMS, et sans risquer de corrompre le jeu de données original en changeant des labels qui ne devaient pas être changés.

2.2.2 Mise à l'échelle des données

Nous avons considéré un StandardScaler et un MinMaxScaler. Chaque feature étant normalement distribué, le StandardScaler nous semblait le plus adapté à première vue. Il s'est avéré qu'en effet, c'est celui avec lequel nous avons obtenu les meilleurs résultats.

2.2.3 Gestion des données manquantes

Les données manquantes de ce data set sont des valeurs des 3 variables explicatives suivantes :

— Le Ph de l'eau : 491 valeurs manquantes 15% des données
— La concentration en sulfates : 781 valeurs manquantes 24% des données
— La concentration en trihalométhane : 162 valeurs manquantes 5% des données

Les données manquantes sont réparties de manière équitable entre l'eau potable et sur l'eau non potable, et il n'apparaît pas de relation entre ces données manquantes et les autres données. On va donc supposer que ces données manquantes sont du type MCAR (Missing Completely at Random).

La solution retenue pour régler le problème des données manquantes est la procédure d'imputation multiple. En effet, la suppression par liste (ou listwise deletion) même si elle était envisageable ne va pas être privilégiée car le jeu de données est déjà petit, réduire encore sa taille n'est pas une bonne idée. L'imputation simple a été envisagée également, avec un remplacement par la moyenne par exemple, mais cela nous donnait des résultats légèrement moins bons qu'avec une imputation multiple.

Plusieurs solutions existent pour effectuer une procédure d'imputation multiple, nous avons choisi d'utiliser une méthode utilisant l'algorithme des k plus proche voisins. Pour cela, nous avons utilisé la méthode "KNNImputer" de la bibliothèque "sklearn.impute". Une imputation utilisant la méthode des K plus proche voisins fonctionne de la même manière que l'algorithme des K plus proche voisins. Ici, pour chaque instance avec une valeur manquante, l'algorithme cherche les k plus proches voisins en utilisant seulement les données non manquantes (i.e. dans le sous espace n'incluant pas la dimension de la donnée manquante). La valeur qui remplacera la donnée manquante est alors la moyenne, pondérée ou non, des valeurs pour cette variable des K plus proches voisins trouvés.

2.2.4 Rééquilibrage des classes

Le jeu de données tel qu'il nous l'est transmis est un peu déséquilibré. Ici 60% de instances sont non potables. Nous avons un temps pensé à rééquilibrer ce jeu de donnée en faisant de l'upscalling. L'upscalling est une méthode qui permet de rééquilibrer un jeu de donnée en créant des nouvelles données synthétiques pour le label le moins représenté. Nous avons finalement abandonné cette idée car cela altérait grandement les prédictions et diminuait les performances de nos algorithmes.

3 Apprentissage de modèles

Il s'agit d'un problème de classification, nous avons donc utilisé des modèles de machine learning pour la classification. Les modèles que nous avons regardés viennent de la bibliothèque scikit-learn. L'approche globale pour l'évaluation et la sélection des modèles est basée sur des grid search et validations croisées sur le jeu d'entrainement, avec 5 splits à chaque fois.

3.1 Modèles utilisés

Nous avons regardé plusieurs modèles : Régression logistique, kNN, SVC, RandomForest, AdaBoost, XGBoost, Bagging... Finalement nous n'avons retenu que 3 modèles : SVC, Random Forest et XGBoost. Ce sont les 3 modèles qui nous donnaient les meilleures performances.

3.1.1 SVM

Le Support Vector Classifier (SVC) est un classificateur qui autorise des hypothèses assez flexibles sur la répartition des instances appartenant à chaque classe, et permet d'avoir des instances mal classifiées dans le cas des soft margin et kernel svm. Cela nous permet donc en théorie d'avoir un classificateur assez robuste à l'overfitting. Le principe du kernel svm est, à partir d'un jeu d'instances non séparable linéairement, d'utiliser une application ϕ à valeurs dans un nouvel espace, dans lequel on va tenter de trouver une frontière linéaire qui sépare au mieux l'ensemble des images du jeu d'instances par ϕ. En principe, on ne détermine pas directement ϕ, on utilise le kernel trick : au lieu de calculer $\phi(x_i)$ et $\phi(x_j)$, on calcule directement le produit scalaire entre $\phi(x_i)$ et $\phi(x_j)$ via la fonction noyau $k(x_i, x_j)$.

Les principaux hyperparamètres du SVC sont C (paramètre de régularisation), *kernel* (le noyau) et *gamma* (coefficient des noyaux rbf, poly et sigmoid). Il y a également *degree* (degré du noyau poly), $coef0$ (terme indépendant dans les noyaux poly et sigmoid), *class_weight* (hyperparamètre qui donne des poids différents à la régularisation selon la classe) , et *tol* (seuil qui définit le critère d'arrêt de l'algorithme d'apprentissage). Nous avons réglé tous ces hyperparamètres, à l'exception de $coef0$.

Les grid search ont été effectuées en utilisant neg_log_loss comme métrique pour la sélection des meilleurs hyperparamètres.

Nous avons tout d'abord effectué une première grid search sur C et $kernel$. Les meilleurs hyperparamètres étant $C = 1$ et $kernel = rbf$, nous avons effectué un second grid search avec $kernel = rbf$ et C autour de 1, en ajoutant les hyperparamètres $gamma$ et $class\_weight$. Nous avons de nouveau obtenu, pour les meilleurs hyperparamètres, $C = 1$, puis $gamma = auto$ et $class\_weight = None$. Finalement, nous avons fait un grid search sur tol, et la meilleur neg_log_loss moyenne de validation croisée a été obtenue pour $tol = 10^{-5}$.

Nous avons donc sélectionné l'estimateur $SVC(C = 1,\ kernel = rbf,\ gamma = auto,\ class\_weight = None,\ tol = 10^{-5})$ et obtenu une neg_log_loss moyenne de validation croisée de -0.61. Plus de détails sur les performances de cet estimateur sont présentés dans la section 3.2.

3.1.2 Random Forest

Une Random Forest est un classificateur se basant sur le principe de Bagging. L'estimateur de base utilisé est un arbre de décision, d'où le nom de Random Forest. On entraine chaque estimateur de base sur un sous ensemble de l'ensemble des instances d'entrainement, construit par tirage aléatoire avec remise (si $bootstrap = True$). A chaque split d'un noeud, $m \leq n\_features$ features sélectionnés aléatoirement sont considérés pour trouver le meilleur split. m est fixé (i.e. le même pour chaque split). La prédiction fournie par la Random Forest est celle qui est majoritaire parmi les prédictions des estimateurs de base.

Les principaux hyperparamètres sont $n\_estimators$ (le nombre d'arbres), $criterion$ (le critère utilisé pour réaliser les split), max_depth (la profondeur maximale des arbres), $max\_features$ (le nombre m évoqué précédemment) et $min\_samples\_leaf$ (le nombre minimal d'instances dans une feuille lors de la phase d'entrainement sur chaque bootstrap sample). Nous n'avons pas fait de grid search sur les autres hyperparamètres. Nous avons utilisé encore une fois la métrique neg_log_loss pour la sélection des meilleurs hyperparamètres.

Nous avons d'abord fait une grid search sur $(n\_estimators,\ criterion,\ max\_features)$, puis une seconde grid search en fixant les meilleurs hyperparamètres de la première, cette fois-ci sur $(max\_depth,\ min\_samples\_leaf)$.

Finalement, l'estimateur avec la meilleure neg_log_loss moyenne de validation croisée (-0.612) est :

$RandomForestClassifier(n\_estimators = 750,\ max\_features = 7,\ criterion = entropy,\ max\_depth = 15,\ min\_samples\_leaf = 1)$

3.1.3 XGBoost

Extreme gradient Boosting (ou XGBoost) est aussi un algorithme de décision basé sur des apprenants faibles (weak learner) qui sont des arbres de décision comme

pour la forêt d'arbres aléatoire. Cependant, cet algorithme n'utilise pas le principe de bagging mais celui de boosting. En d'autres termes les arbres de décision ne sont pas entraînés de manière parallèle mais de manière séquentielle, l'un après l'autre pour que les nouveaux arbres réduisent l'erreur de prédiction des précédents arbres. Pour cela la méthode de descente de gradient est appliquée pour réduire le plus vite possible cette erreur.

Pour cet algorithme nous avons utilisé la fonction XGBoostClassifier de la bibliothèque XGBoost. Cette bibliothèque est connu pour implémenter de manière efficace l'algorithme de descente de gradient. Ces algorithmes sont souvent les vainqueurs des compétitions d'apprentissage automatique en ligne.

Les principaux hyperparamètres sont $n\_estimators$ (le nombre d'arbres), max_depth (la profondeur maximale des arbres), $learning\_rate$ (le paramètre de la descente de gradient). Nous n'avons pas fait de grid search sur les autres hyperparamètres. La métrique accuracy a été utilisée pour la grid search.

3.2 Résultat des apprentissages

3.2.1 SVM

mean cv accuracy : 0.673

val classification report :

	precision	recall	f1-score	support
0	0.70	0.90	0.79	264
1	0.67	0.35	0.46	154
accuracy			0.70	418
macro avg	0.68	0.62	0.62	418
weighted avg	0.69	0.70	0.67	418

3.2.2 Random Forest

mean cv accuracy : 0.667

val classification report :

	precision	recall	f1-score	support
0	0.71	0.86	0.77	264
1	0.61	0.39	0.48	154
accuracy			0.68	418
macro avg	0.66	0.62	0.63	418
weighted avg	0.67	0.68	0.66	418

3.2.3 XGBoost

mean cv accuracy : 0.652

val classification report :

	precision	recall	f1-score	support
0	0.69	0.85	0.76	264
1	0.57	0.35	0.44	154
accuracy			0.67	418
macro avg	0.63	0.60	0.60	418
weighted avg	0.65	0.67	0.64	418

3.3 Selection du modèle

Le SVC est celui qui a la meilleure cross validation accuracy. Par ailleurs, en comparant les rapports de classification de validation des 3 estimateurs, nous voyons que le recall de la classe 0 (non potable) du SVC est meilleur que celui des deux autres estimateurs. Ce recall est un indicateur important pour notre problème car nous souhaitons en priorité prédire, quand l'eau est non potable, qu'elle est effectivement non potable. Plus le recall de l'eau non potable est faible, plus grande sera la proportion d'eau non potable qui "passe entre les mailles du filet". Ce que l'on veut bien évidemment éviter. Enfin, on voit que sur les autres mesures de performance, le SVC est soit meilleur soit sensiblement de même performance que les deux autres estimateurs. Nous choisissons donc l'estimateur SVC :

$SVC(C = 1,\ kernel = rbf,\ gamma = auto,\ class\_weight = None,\ tol = 10^{-5})$

3.4 Performances en test

test classification report :

	precision	recall	f1-score	support
0	0.69	0.94	0.79	308
1	0.74	0.28	0.41	184
accuracy			0.70	492
macro avg	0.72	0.61	0.60	492
weighted avg	0.71	0.70	0.65	492

4 Conclusion

Le modèle choisi pour prédire la potabilité de l'eau est donc le SVC mentionné à la fin de la section 3.3. D'après les performances en test (section 3.4), on peut attendre de cet estimateur une accuracy de 70%. Concernant les precision scores : quand l'estimateur prédit que l'eau est potable (respectivement non potable), dans 70% des cas l'eau est effectivement potable (respectivement non potable). En termes de recall de l'eau non potable, on peut espérer que 90% à 95% des eaux non potables soient détectées. C'est un relativement bon score et c'était notre priorité. En revanche, seulement environ 30% des eaux potables seront détectées comme étant potables.

Si l'on se fie aux résultats de cet estimateur, on risque donc d'investir dans un système d'assainissement de l'eau alors qu'il n'y en aurait pas besoin... En revanche, il est assez improbable que l'on n'investisse pas dans un tel système alors que l'eau est en réalité non potable. On risque donc d'investir trop, mais toujours moins que si l'on investissait partout, et le niveau de sécurité des populations concernant la qualité de l'eau courante sera correct : $100\% - proportion\_non\_potable \times (1 - recall\_non\_potable) = 100\% - 60\% \times 5\% = 97\%$ des cours d'eau testés puis assainis si besoin d'après la prédiction seront potables. Mais on aura investi dans un système d'assainissement de l'eau pour $95\% \times 60\% + (1 - 30\%) \times 40\% = 85\%$ des cours d'eau alors qu'il n'aurait fallu investir que pour 60% des cours d'eau.

5 Code source

Lien Gitlab du code source : `https://gitlab-student.centralesupelec.fr/2019veym/ml-project-water-potability`