05/11/2021

Projet Machine Learning

Modèle de prédiction de la potabilité d'une eau

Maxime Zennaro
Wallerand Peugeot
Rayan Smaoui

I. Table des matières

II. Introduction

Le projet porte sur l'étude d'un jeu de données sur la qualité de l'eau. Aujourd'hui, l'accès à l'eau est un des problèmes majeurs de certains pays en voie de développement mais aussi dans certaines régions des pays développés. C'est aussi un enjeu important pour le développement durable notamment du point de vue environnemental et social mais aussi économique dans une perspective d'aménagement d'infrastructures d'assainissement de l'eau.

Notre problématique sur ce sujet est la suivante : Trouver un modèle de prédiction nous permettant de savoir si une eau est propre à la consommation ou non.

L'objectif est de trouver un modèle de classification binaire permettant de maximiser la précision. En effet, nous avons jugé que ce critère d'évaluation comme étant le plus pertinent du fait des nombreux risques sanitaires qui peuvent découler d'un faux diagnostic (maladies…). Quant au choix de la classification binaire, le but est de prédire si une eau est potable (résultat 1) ou ne l'est pas (résultat 0).

Pour accéder à notre projet sur git, cliquer ici.

III. Travail

A. Analyse des données

Lors de notre étape d'analyse des données, nous avons repéré 3 features qui possédaient des données manquantes : le pH, le sulfate et le trihalométhane. Toutes les variables explicatives avaient des distributions de type gaussienne.

Nous avons regardé les corrélations qui pouvaient exister entre les différentes features et la variable à expliquer "Potabilité". Celles-ci apparaissent assez faibles (3% maximum).

Dans un premier temps nous avons donc décidé d'ignorer et d'exclure les features ayant des valeurs manquantes du dataset pour faire un premier test naïf. Les résultats obtenus en termes d'accuracy et de précision n'étaient pas très élevés, autour de 55% en moyenne.

Nous avons ensuite décidé d'adopter une stratégie d'imputation par la moyenne des valeurs mesurées pour chaque feature. Puis une stratégie d'imputation KNN i.e. d'estimer les valeurs manquantes en se basant sur des instances similaires pour les données connues.

Enfin, une dernière approche a été proposée celle de l'imputation itérative (multivariate imputer). Elle consiste à estimer les valeurs manquantes à partir des valeurs obtenues pour les autres features. C'est cette méthode qui sera retenue par la suite.

B. Stratégie adoptée et choix des modèles

Stratégie : Pour y parvenir, nous allons mettre en place la stratégie suivante : Dans un premier temps, nous allons travailler sur le jeu de données et appliquer des étapes de mise à l'échelle et d'imputation des données manquantes afin d'obtenir un DataSet le plus exploitable possible. Etant donné que le nombre de features est petit comparé au nombre d'instances nous avons jugé qu'il n'était pas nécessaire d'appliquer une stratégie de réduction de dimension (telle que la méthode PCA par exemple).

Ensuite, nous allons appliquer différents modèles de Machine Learning qui permettent de faire de la classification binaire. Enfin, nous optimiserons les hyperparamètres de chaque algorithme via une étape de Grid Search pour comparer les résultats obtenus et ainsi choisir l'algorithme le plus précis.

Les différents modèles retenus sont des modèles permettant de faire de la classification binaire comme la Régression Logistique, les Arbres de Décisions, le Random Forest, SVM (support vector machine), GPC (Gaussian Process Classifier), MLP (Multi Layer Perceptron) ou encore l'algorithme KNN. Une approche d'apprentissage de vote de majorité (Voting classifier) sera utilisée pour observer si une agrégation de plusieurs des modèles cités précédemment pourrait améliorer la performance du modèle.

Enfin, nous allons tester des modèles supplémentaires tels que le Gradient Boosting (xgboost) et l'Adaptative Boosting (Adaboost).

C. Modèles utilisés et leur paramétrage

1. Logistic Regression

C'est un algorithme supervisé de classification. C'est un modèle linéaire qui repose sur une fonction d'hypothèse appelé aussi fonction de score (sigmoïd, LOGIT…) qui permet de classer les instances selon si elles sont positives (1) ou négatives (0).

Les réglages optimaux sont ceux par défaut. Autrement dit, notre solveur est LBFGS avec une pénalité quadratique.

2. Decision Tree

Cet algorithme permet de représenter un ensemble de choix sous forme d'une structure d'arbres. Les décisions finales sont atteintes en fonction des différentes décisions intermédiaires prises lors du parcours de l'arbre.

Les réglages optimaux sont aussi ceux par défaut. Autrement dit, notre critère de segmentation est « gini » (il mesure avec quelle fréquence un élément aléatoire l'ensemble serait mal classé) avec un splitter de type best.

3. Random Forest

C'est un algorithme de classification qui réduit la variance des prévisions d'un arbre de décision seul en ne se reposant sur le principe de bagging (Bootstrap + aggregating) d'un grand nombre d'arbres de décisions construits avec un certains nombres de caractéristiques aléatoires.

À la suite de notre étape de Grid Search, les réglages optimaux sont les suiavnts : le nombre d'estimateur est de 100, notre critère de segmentation est « entropy », une profondeur maximale de 5 et nous prenons un nombre maximal de features lors de la segmentation de l'ordre de $\sqrt{n} = 3$ (avec n le nombre total de features).

4. SVM

Appelé « Machine à vecteurs de support », c'est un algorithme d'apprentissage supervisé où les données sont séparées en plusieurs classes en recourant à la "marge maximale", avec une frontière de séparation choisie pour maximiser la distance entre les groupes de données. L'objectif est de déterminer la frontière optimale.

À la suite de notre étape de Grid Search, les réglages optimaux sont les suivants : le noyau utilisé est de type polynomial de degré 3, notre coefficient gamma est « auto » et un coefficient C de 0.1.

5. KNN

L'algorithme des K plus proches voisins (KNN) est un algorithme très simple de mise en place qui permet de classer les instances en se basant sur les caractéristiques des voisins les plus proches autour de l'instance. La difficulté réside dans le nombre de voisins à considérer pour pouvoir être robuste au bruit ou éviter de biaiser nos résultats.

À la suite de notre étape de Grid Search, les réglages optimaux sont les suivants : le nombre de voisins est 25 et la fonction de poids utilisée pour la prédiction est uniforme.

6. Gaussian Classifier

Cette une méthode probabiliste permet d'obtenir une classification des instances. Une probabilité d'appartenance à chacune des classes est calculée et une classification est généralement obtenue en affectant chacune des observations à la classe la plus probable. Cet algorithme est basé sur l'approximation de Laplace (Sklearn).

À la suite de notre étape de Grid Search, les réglages optimaux sont ceux par défaut. Autrement dit, le noyau est gaussien avec comme optimiseur « fmin_l_bfgs_b ».

7. MLP

Le Perceptron multi-couches pour la classification est un réseau de neurones organisé en plusieurs couches et dit réseau de propagation directe (les informations circulent de l'entrée vers la sortie uniquement).

À la suite de notre étape de Grid Search, les réglages optimaux les suivants : la taille des couches intermédiaires est (50, 50, 50) la fonction d'activation est la tangente hyperbolique, le solver « sgd » (optimisation par descente de gradient stochastique) et un coefficient alpha (d'apprentissage) de 0.001.

8. Voting Classifier

Cette approche consiste à assembler plusieurs classificateurs classiques, qui pour chaque instance vont retourner une probabilité d'appartenance aux différentes classes, et le retour final est la classe dont la somme des probabilités est la plus haute.

Nous avons testé toutes les combinaisons possibles entre les 5 algorithmes suivants : Random Forest, SVM, Gaussian Classifier, KNN et le MLP. Dans l'optique d'optimiser la précision, nous avons trouvé la combinaison suivante : Random Forest, SVC et KNN.

9. Stacking Classifier

Le stacking classifier est un procédé qui consiste à appliquer un algorithme de machine learning classique (ici une régression logistique) aux sorties de plusieurs autres classificateurs. C'est une autre méthode d'agrégation de modèles que celle vu précédemment.

Nous avons testé toutes les combinaisons possibles entre les 5 algorithmes suivants : Random Forest, SVM, Gaussian Classifier, KNN et le MLP. La combinaison la plus optimale pour la précision est : SVC et MLP.

10. XGBoost

Le Boosting de Gradient est un algorithme d'apprentissage supervisé dont le principe et de combiner les résultats d'un ensemble de modèles plus simple et plus faibles afin de fournir une meilleure prédiction de manière séquentielle.

Les réglages optimaux sont aussi ceux par défaut. Autrement dit, le booster est de type « Gradient boosting tree » (la somme des résidus quadratique est plus faible qu'un modèle linéaire) et un objectif de classification binaire.

11. AdaBoost

Cette méthode de boosting s'appuie sur la sélection itérative de classifieur faible en fonction d'une distribution des exemples d'apprentissage. Chaque exemple est pondéré en fonction de sa difficulté avec le classifieur courant.

Nous avons choisi un nombre d'estimateurs de 100 et un taux d'apprentissage de 1.

IV. Présentation des résultats

	Model Accuracy	Model Precision	Model Recall	Model Name
0	60.989015	30.494508	50.000000	LogisticRegression
1	53.937535	51.753883	50.985281	DecisionTree
2	64.591324	64.177714	56.816641	KNN
3	64.132424	69.868316	54.909842	RandomForest
4	61.385915	71.368133	50.565379	SVM
5	67.888056	67.914450	61.745430	GaussianClassifier
6	63.827406	70.712934	53.090583	MLPClassfier
7	63.217325	74.167605	53.118899	VotingClassifier
8	63.430693	73.166481	52.540333	StackingClassifier
9	64.987479	62.588656	60.933945	XGBoost
10	59.553761	54.303736	52.561319	AdaBoost

Figure 1 : Tableau des différentes métriques (accuracy, précision et rappel) pour nos différents modèles

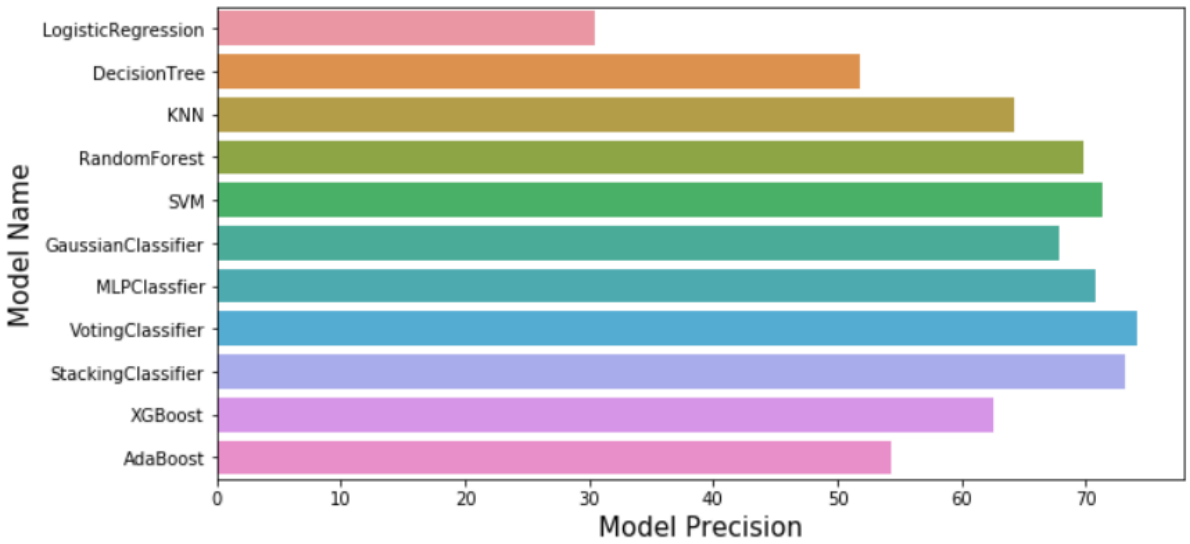

Figure 2 : Visualisation de la précision pour nos différents modèles

Le modèle le plus performant en termes de précision est donc le Voting Classifier (regroupant un Random Forest, un SVC et un KNN) avec un score de 74.2%, suivi de près par le Stacking Classifier (regroupant un SVC et un MLP) avec un score de 73.2 %.

V. Conclusion et perspectives

Concernant cette étude de cas, nous avons déduit que l'algorithme qui était le plus adapté à nos recherches était de type Random Forest ou SVC (tous 2 utilisés dans le voting classifier). En effet, dans un Random Forest, les choix se font en comparant les diverses valeurs des features par rapport à des seuils donnés par les normes gouvernementales (par exemple, le pH doit être compris entre 6.5 et 9) et c'est ce qui correspondait le mieux selon nous à ce problème.

Nous n'avons pas pu utiliser l'approche PCA (Principal Component Analysis) vue en cours sur ce projet car on a déjà assez peu de features et d'exemple à notre disposition et il n'est donc pas nécessaire de réduire davantage la dimension. Concernant le feature engineering, nous n'avons pas trouvé des relations pertinentes pouvant mettre en évidence une corrélation supplémentaire. Par exemple faire un rapport entre 2 features, nous fournissait au plus une corrélation de cette nouvelle feature de 3%.

Enfin concernant les résultats obtenus après Grid Search, pour la majorité des algorithmes, les hyperparamètres par défaut donnaient les meilleurs résultats étant donné que nos données sont très classiques (elles suivent une répartition de type gaussienne).

En conclusion, nous avons pu utiliser ce projet pour nous entraîner à implémenter différentes méthodes vues en cours. Cela nous a aussi permis de mieux les comprendre grâce à leur mise en application et l'étude des résultats qu'elles nous fournissent.