UNIVERSITÉ DE LORRAINE

PROJET DE FOUILLE DES DONNÉES

Master 2 Ingénierie Mathématique pour la Science des Données

2022-2023

Fouille de données au service du développement durable

Réalisé par :
Marcel MOUDILA
Abdoullah LATRECHE

Enseignant :
Sabeur ARIDHI
Sebastien DA SILVA

6 décembre 2022

Introduction

La fouille des données ou data mining, d'après Wikipédia, se propose d'utiliser un ensemble d'algorithmes issus de disciplines scientifiques diverses telles que les statistiques, l'intelligence artificielle ou l'informatique, pour construire des modèles à partir des données, c'est-à-dire trouver des structures intéressantes ou des motifs selon des critères fixés au préalable, et d'en extraire un maximum de connaissances.

Le projet, faisant l'objet de ce rapport, est la résultante d'un défi initié par la mairie de la ville de Grenoble et l'entreprise Big Datext. Il consiste, à partir d'une base de données opensource sur les arbres Grenoblois, d'extraire des connaissances et fournir des préconisations pour faciliter son entretien.

Les outils que nous avons utilisés sont Python (jupyter notebook) et Overleaf. Le premier pour l'analyse, le traitement, et l'aide à la décision. Le second pour la rédaction de ce rapport.

Le notebook jupyter ainsi que le jeu des données sont disponibles sur nos sites github (github.com/marcel-moudila/projet-fouille-des-donnees) ou (github.com/Nindo16)

Nous présentons le plan suivant :
Dans le chapitre 1, il sera question de faire l'analyse exploratoire des données (dimension du corpus des données, types des variables, description des variables, tests statistiques). Dans le chapitre 2, nous présentons le prétraitement en montrant quelques étapes de l'algorithme de prétraitement que nous avons créé. Dans le chapitre 3, nous entraînons différents modèles de machine-learning (Random Forest, Adaboost, SVM, KNN, et Xgboost), et présentons pour chacun d'eux les scores des métriques utilisés (recall, précision, et accuracy) pour résoudre la tâche de classification uni-label. Dans le chapitre 4, de même que nous présentons la classification multi-label sur ces données. Un dernier chapitre, plus ouverte, présente les suggestions dans le but de réduire au mieux la présente de défauts pour les prochaines années. L'annexe présente les codes, nous donnons également au professeur le notebook jupyter en HTML à ouvrir sur un navigateur.

Bonne Lecture.

Table des matières

Chapitre 1

Analyse exploratoire des données

1.1 Une première analyse de nos données

L'objectif de ce chapitre est de comprendre au mieux possible notre de jeu de données en vue d'établir des modèles cohérents pour nos prédictions.

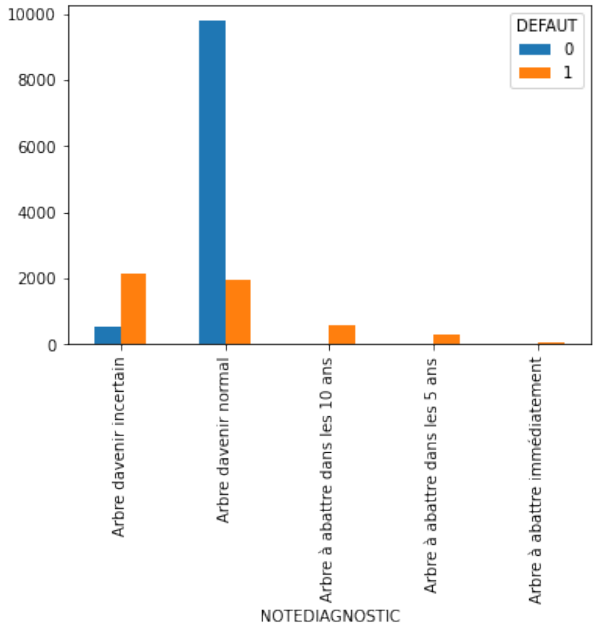
Jetons un coup d'œil aux différentes variables décrivant nos arbres :

- ANNEEDEPLANTATION
- DIAMETREARBREAUNMETRE
- ESPECE
- GENRE_BOTA
- STADEDEDEVELOPPEMENT
- VARIETE
- VIGUEUR

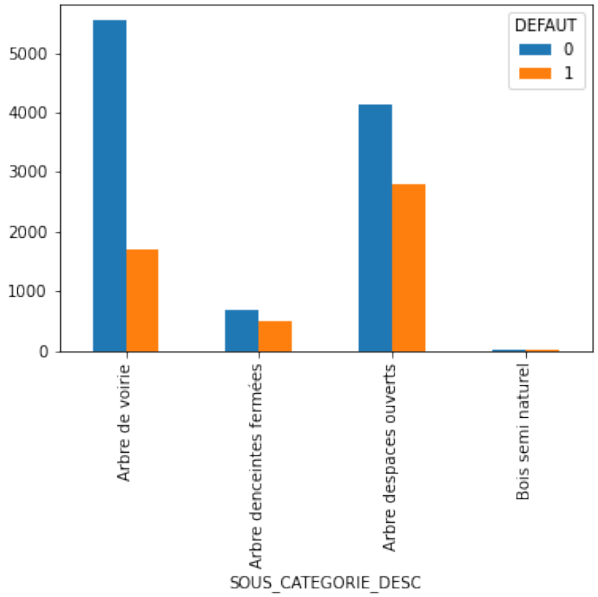
Les arbres présentent différents attributs les caractérisant.

- ADR_SECTEUR
- CODE_PARENT
- CODE_PARENT_DESC
- FREQUENTATIONCIBLE
- SOUS_CATEGORIE
- SOUS_CATEGORIE_DESC
- TROTTOIR

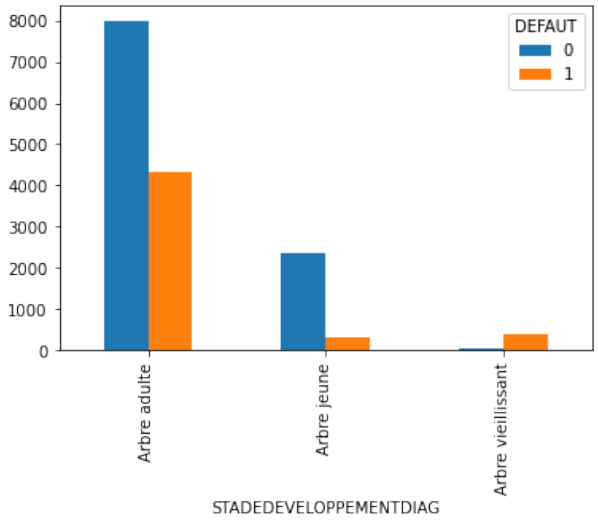
Des informations spatiales nous sont donc également fournies. Une première approche dans notre quête de prédiction avant toute considération de valeurs manquantes ou autre aspect technique peut être d'aller regarder, naïvement, quelles sont les variables de notre jeu de données ayant potentiellement un impact sur la présence de défaut ou non sur un arbre.



Cette visualisation nous permet d'étudier l'influence de l'attribut NOTEDIAGNOSTIC sur la présence de Défaut. On y remarque par exemple qu'environ un cinquième des arbres d'avenir normal présente une anomalie. Cette proportion est prépondérante chez les arbres d'avenir incertain, aux alentours des trois quarts. Il est donc possible que cette variable joue un rôle sur la présence ou non d'un défaut.



Malgré notre manque d'expertise en dendrologie, nous remarquons tout de même que les arbres d'espaces ouverts sont plus à même de posséder une anomalie que les arbres de voiries



De la même manière que sur les précédentes représentations, il semble y avoir une certaine cohérence. Les arbres vieillissants sont plus propices à la présence de défaut que les jeunes arbres.

Après nous être quelque peu familiarisés avec ces variables et après avoir émis quelques hypothèses, il convient tout de même de faire preuve de rigueur et quantifier les liens possibles entre notre variable cible et nos attributs. Nous ferons donc appel aux outils développés en ce sens, les tests statistiques. Nous n'allons pas ici faire le travail sur toutes les variables afin de ne pas alourdir la présentation, nous présenterons juste la méthode, le travail sur le reste des variables ayant été fait sur Jupyter Notebook.

1.2 Comment allons-nous choisir nos variables ?

Afin d'établir des modèles simples et efficaces de machine learning, nous sommes confrontés à certaines problématiques, notamment celle de la sélection de variables. Il nous faut garder les variables ayant un impact important, tout en gardant une certaine souplesse de généralisation.

Revenons à notre variable NOTEDIAGNOSTIC dont le tableau de contingence est présenté ci-dessous

DEFAUT NOTEDIAGNOSTIC	0	1
Arbre davenir incertain	511	2140
Arbre davenir normal	9784	1932
Arbre à abattre dans les 10 ans	25	567
Arbre à abattre dans les 5 ans	31	292
Arbre à abattre immédiatement	7	46

Nous avons toujours plus de 5 observations par case sur le tableau, on est donc en droit d'effectuer un test du Khi-deux qui répondra parfaitement à nos besoins.

	Chi2 stattistic	p-value
NOTEDIAGNOSTIC	50.834074	1.005135e-12

Notre p-value étant très faible, nous rejetons l'hypothèse d'indépendance et soupçonnons ainsi un certain lien entre la variable et cible et NoteDiagnostique. Il en est de même pour plusieurs autres

attributs décrivant l'arbre dans ce jeu de données.

Après avoir choisi soigneusement nos variables après ces tests statistiques, il subsiste toujours un aspect non traité actuellement, la corrélation. Il est possible que parmi nos variables certaines soient corrélées entre elles, on se trouverait alors dans un cas de données redondantes. Regardons alors de plus près nos modalités.

Profitons du fichier excel fourni décrivant toutes les variables de notre dataset. Voici la description de la variable

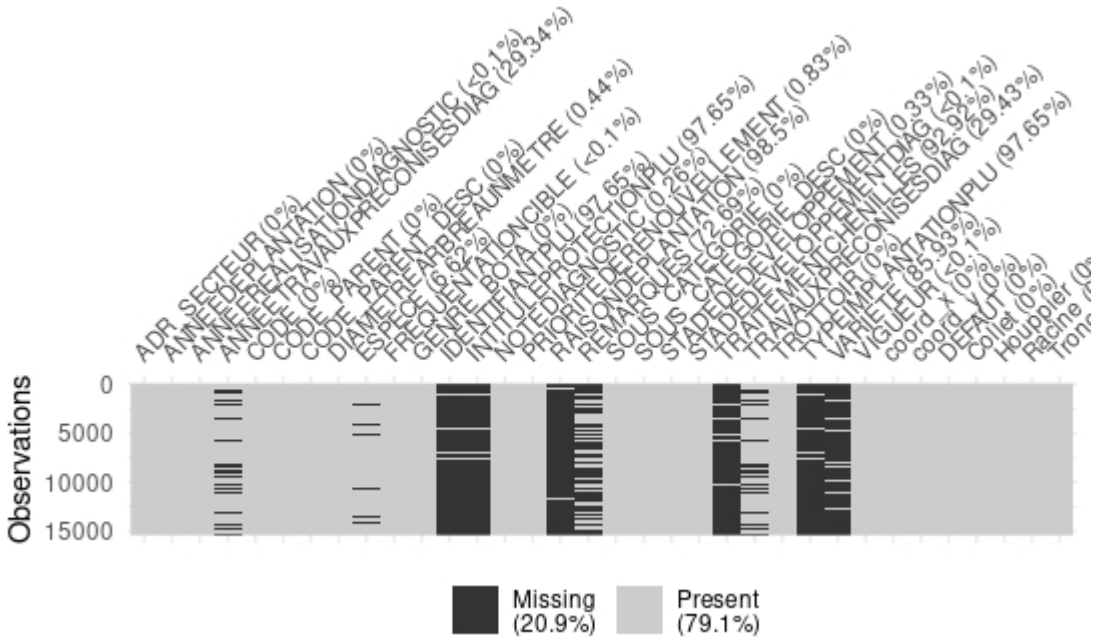
SOUS_CATEGORIE : Identifiant de la sous-catégorie
SOUS_CATEGORIE_DESC : Description des sous-catégories

On se doute que deux arbres appartenant à la même catégorie possèdent la même description de cette catégorie. Cela se révèlera être effectivement le cas, nous ferons le choix de supprimer la variable SOUS_CATEGORIE.

De la même manière en retournant sur la description des modalités.

STADEDEVELOPPEMENT : Stade de développement de l'arbre
STADEDEVELOPPEMENTDIAG : Stade de développement lors du diagnostique

Nous ferons le choix de ne garder simplement que la deuxième variable mentionnée afin de garder les informations les plus à jour possible.
Après avoir sélectionné ces quelques variables, nous sommes obligés de revenir sur un point omis jusqu'à présent, les données manquantes. Réelle problématique dans le monde de la Data, nous essayerons de faire au plus simple dans notre cas. Jetons donc un coup d'œil à la répartition de celle-ci au sein de notre jeu de données.



En vue d'établir un premier modèle, on se limitera simplement aux variables ayant moins de 70% de valeurs manquantes. Dans le cas où notre modèle ferait preuve de maigre performance, nous reconsidérerions cette approche.

1.3 À quoi ressemblent finalement nos arbres ?

Après avoir appliqué les changements précédents. On se retrouve avec 6343 arbres sains ainsi que 3570 arbres présentant une quelconque anomalie. Ce rapport n'est pas à négliger, il influencera forcément notre modèle. Se pose alors la question plus générale, quel est l'arbre type de notre jeu de données ? Voici ce que nous dit notre analyse de celui-ci.

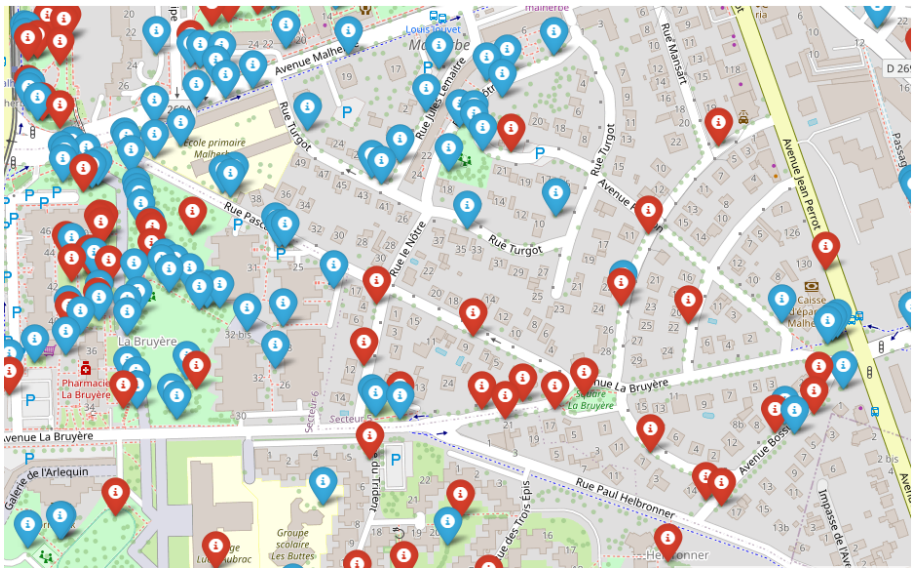
Les arbres les plus représentés dans ce jeu de données appartiennent aux catégories suivantes

- arbres entre 10 et 20 centimètres de diamètre
- d'avenir normal
- étant dans des espaces ouverts
- non positionnés sur le trottoir
- adulte
- vigoureux

Voici l'arbre type de notre forêt, cela reste de plus cohérent (pour nous amateurs en la matière) avec le fait qu'il ne présente pas de défaut. On a en effet un arbre diagnostiqué par les experts comme étant de bonne santé, non positionné sur un passage très fréquenté, vigoureux. Aucun problème à priori. Lors de la création de notre modèle, nous avons été confrontés à ce "problème", en effet, un modèle d'apprentissage aura tendance à prédire en tenant compte du personnage le plus représenté.

(petite digression) Nous avons donc considéré la bibliothèque SMOTE, qui semblait être la solution miracle. Elle permet de créer des échantillons synthétiques en utilisant l'algorithme des k plus proches voisins, c'est-à-dire dans notre cas, rajouter des arbres ayant un défaut afin de rééquilibrer les proportions. On se retrouverait donc avec environ une moitié d'arbre ayant un défaut et l'autre n'en ayant pas. Finalement, nous avons décidé de ne pas utiliser cette méthode. Il est possible de créer des corrélations entre variables non voulues, d'avoir l'effet inverse qu'escompter, à savoir un surapprentissage, etc. Nous avons voulu faire au plus simple et éviter une mauvaise utilisation des méthodes possibles.

Visualisation d'un échantillon de la forêt de Grenoble



Les arbres présentant un défaut ont été affichés en rouge tandis que ceux en bleu n'en possèdent pas. Dans un projet d'amélioration de la qualité de vie d'un arbre, il peut être intéressant d'étudier ces données spatiales. L'environnement dans lequel évolue l'arbre peut être un facteur d'aggravation ou non de son état. Nous avions vu tout juste précédemment que les arbres à proximité d'un trottoir semblaient plus sains.

Chapitre 2

Prétraitement

L'objectif de cette section est de présenter les étapes de pré-traitement effectuées.

2.1 conversion des types de certaines variables

La variable ADR_SECTEUR a été convertie en type "object", de même que les variables ANNEEDEPLANTATION, ANNEEREALISATIONDIAGNOSTIC, ANNEETRAVAUXPRECONISESDIAG.
La raison qui le justifie est que nous nous sommes intéressés aux effectifs de chaque modalité de ces variables.
Les variables de prédiction également ont été converties en type "object", car nous devons effectuer la classification.

2.2 Éliminations de certaines variables

1. CODE : car ses modalités (au nombre total des arbres) sont uniques pour chaque observation
2. CODE_PARENT : car a trop de modalités (au nombre de 1141) distinctes, et est redondant avec CODE_PARENT_DESC
3. IDENTIFIANTPLU : car a seulement 361 observations, soit environ 97 % de valeurs manquantes, et est redondant avec INTITULEPROTECTIONPLU, et aussi redondant avec TYPEIMPLANTATIONPLU
4. RAISONDEPLANTATION : car a seulement 230 observations, soit environ 98 % de valeurs manquantes
5. REMARQUES : car a trop de modalités (au nombre de 1684) distinctes
6. SOUS_CATEGORIE : car redondante avec SOUS_CATEGORIE_DESC (qui est conservée par rapport à elle, car les noms des modalités sont plus faciles à comprendre)
7. TRAITEMENTCHENILLES : car test de Khi-deux avec DEFAUT n'est pas significatif (p-value > 0.05)
8. VARIETE : car a seulement 2163 observations, soit environ 85 % de valeurs manquantes

	Chi2 stattistic	p-value
TRAITEMENTCHENILLES	3.220853	0.072706

Table 2.1 – Test de Khi-2 (TRAITEMENTCHENILLES, DEFAUT)

Cette tâche nous a permis de ne conserver que 9913 observations et 23 variables dont les 5 variables de prédiction. En principe, pour les valeurs manquantes, les algorithmes que nous utiliserons par la suite fonctionnent avec les valeurs manquantes, mais comme nous le verrons par la suite, certains algorithmes dans certains problèmes tirent profits de la quantité des données.

Chapitre 3

Prédiction uni-label

L'objectif de cette section est de présenter la réponse au problème de la classification uni-label de ce projet.

Après le prétraitement, on entraine les algorithmes de classification de notre choix sur le corpus train et puis on les évalue sur le corpus test à l'aide des métriques d'évaluation des problèmes de classification (précision, rappel, accuracy) et pour valider notre modèle, on fait la validation croisée.

Actual class (observation)	1	0
Predicted class (expectation)		
1	tp (true positive)	fp (false positive)
0	fn (false negative)	tn (true negative)

$$\text{precision} = \frac{tp}{tp + fp},$$

$$\text{recall} = \frac{tp}{tp + fn},$$

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}.$$

L'accuracy est la somme des valeurs sur la diagonale (true positive + true negative) divisé par le nombre d'observations (tp + fp + fn + tn)

Tous les algorithmes ont été utilisés avec les paramètres par défaut, sauf :

- Random Forest : n_estimators (300), max_depth (10),
- Xgboost : n_estimators (300), max_depth (10)

Tous les algorithmes ont travaillé sur les données grâce au paramètre random_state mis à 42 de la commande train_test_split de Scikit-learn qui sépare les données en corpus d'apprentissage et en corpus de test.

Le tableau suivant représente les valeurs de l'accuracy, le rappel (ou recall) et la précision des 5 modèles utilisés (Random Forest, Adaboost, k-Narest Neighbors(KNN), et eXtreme Gradient Boosting (Xgboost))

	accuracy	precision	recall
Random Forest	**0.88**	**0.91**	**0.76**
Adaboost	0.84	**0.86**	0.69
Support Vector Machine (SVM)	**0.87**	**0.87**	**0.74**
k-Narest Neighbors(KNN)	0.85	**0.83**	**0.74**
eXtreme Gradient Boosting (Xgboost)	**0.89**	**0.89**	**0.81**

TABLE 3.1 – scores des modèles

Les valeurs en gras sont au-dessus du classifieur Baseline 86% pour l'exactitude (accuracy), 82% pour la précision et 72% de rappel.

D'après les résultats, c'est Xgboost qui est l'alogorithme à priviligier pour l'accuracy , et le rappel, par contre Random Forest est le meilleur en précision.

3.1 Validation croisée

Ici, sont présentées la courbe d'apprentissage et la courbe de validation croisée pour notre meilleur algorithme Xgboost, mais aussi pour Random Forest (car comme on le verra dans la suite, les deux modèles ont la même valeur AUC). Le score est l'accuracy, et le nombre de folds est 10.

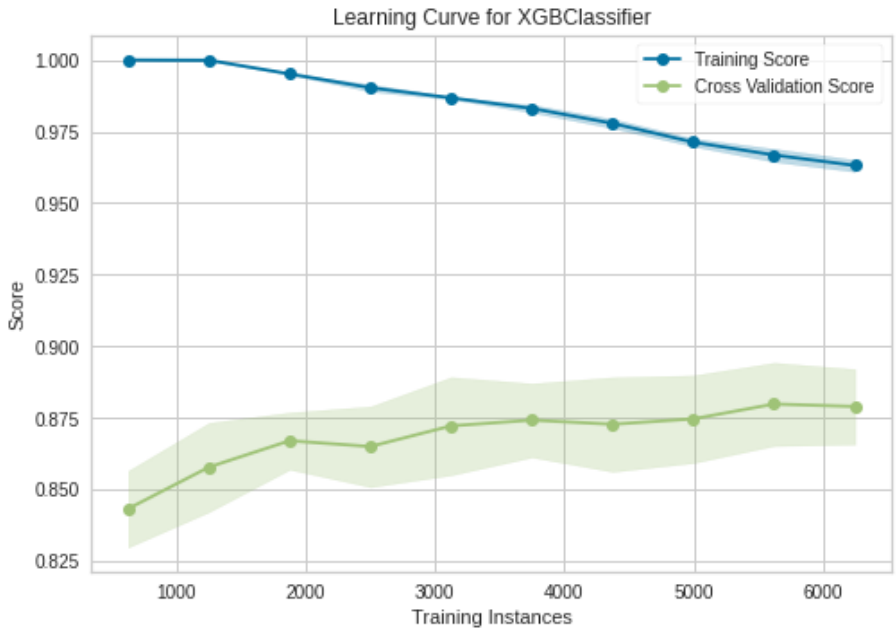


FIGURE 3.1 – Validation croisée Xgboost

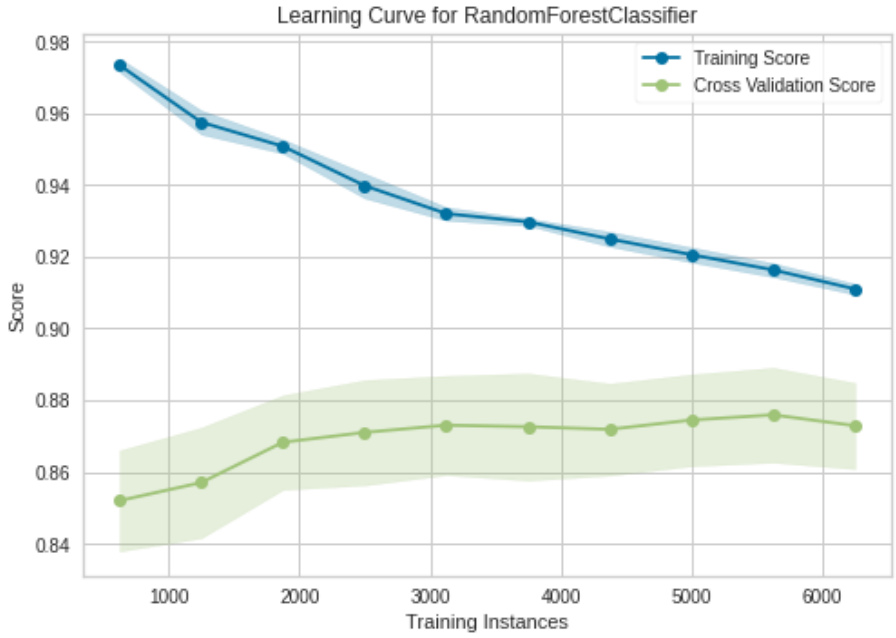


FIGURE 3.2 – Validation croisée Random Forest

Ces graphiques montrent la relation entre le score (accuracy) pour le corpus d'apprentissage et le score pour le corpus de validation croisée avec un nombre variable d'échantillons d'apprentissage.

Si les deux scores convergent à mesure que davantage de données sont ajoutées, le modèle ne bénéficiera probablement pas de plus de données.

la variabilité pendant la validation croisée est indiquée avec les zones ombrées qui représentent un écart type au-dessus et au-dessous de la moyenne pour toutes les validations croisées.

Si le modèle souffre d'une erreur due à un biais, il y aura probablement plus de variabilité autour de la courbe des scores d'apprentissage. Si le modèle souffre d'une erreur due à la variance, il y aura alors plus de variabilité autour du score de validation croisée.

Nous pouvons voir que les scores d'apprentissage et de test n'ont pas encore convergé, donc potentiellement les deux modèles bénéficieraient de plus de données d'apprentissage.
Les deux modèles souffrent tous les deux d'une erreur due à la variance (dans l'analyse exploratoire, on avait repéré des données déséquilibrées (ANNEEDEPLANTATION, FREQUENTATION-CIBLE, et NOTEDIAGNOSTIC)

Les scores CV pour les données de test sont plus variables que pour les données d'apprentissage, il est donc possible que le modèle soit surajusté.

Potentiellement, avec plus de données et une réduction des profondeurs des arbres dans les hyperparamètres des modèles, ces modèles deviendront beaucoup moins variables dans les données de test.

3.2 Courbe ROC

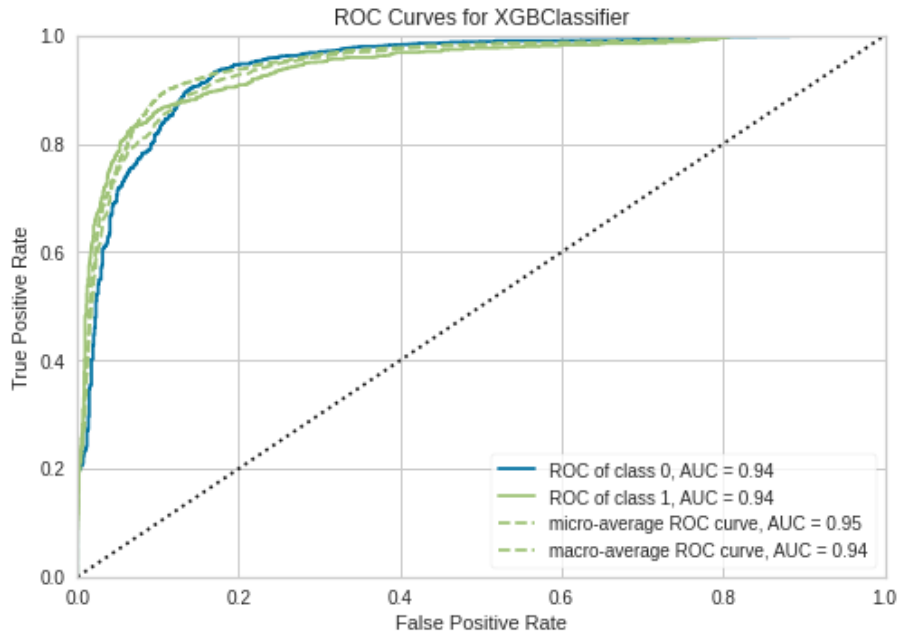


FIGURE 3.3 – Courbe ROC Xgboost

L'aire sous la courbe ROC (Receiver Operating Characteristic) appélée AUC (Area Under the Curve) est une autre métrique d'évaluation dans les problèmes de classification. L'AUC pour Xgboost était la même que pour Random Forest et meilleur que celles des trois autres modèles testés. En général, le meilleur modèle a la plus grande valeur AUC. Donc, on revient à la même conclusion de retenir Xgboost comme modèle pour prédire les défauts des arbres de test data dont les vrais labels sont auprès des enseignants.

3.3 Importance des variables

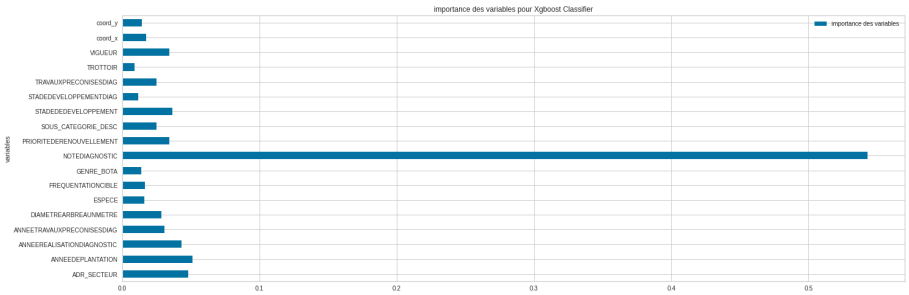


FIGURE 3.4 – Importance de l'impureté de Gini donnée par Xgboost

Ce graphique nous indique les variables importantes (ayant le plus faible gain de Gini) pour choisir les nœuds des apprenants faibles d'arbres de décision. On est presque sûr que la variable NOTEDIAGNOSTIC constitue le premier nœud (la racine) de ces apprenants faibles (300 arbres de décision puisque nous avons mis n_estimators à 300)

Les 10 variables les plus importantes pour prédire si l'arbre a un défaut sont les suivants : ADR_SECTEUR, ANNEEDEPLANTATION, ANNEEREALISATIONDIAGNOSTIC, ANNEETRAVAUXPRECONISESDIAG, DIAMETREARBREAUNMETRE, NOTEDIAGNOSTIC, PRIORITEDERENOUVELLEMENT, STADEDEDEVELOPPEMENT, TRAVAUXPRECONISESDIAG, et VIGUEUR.

Chapitre 4

Prédiction multi-label

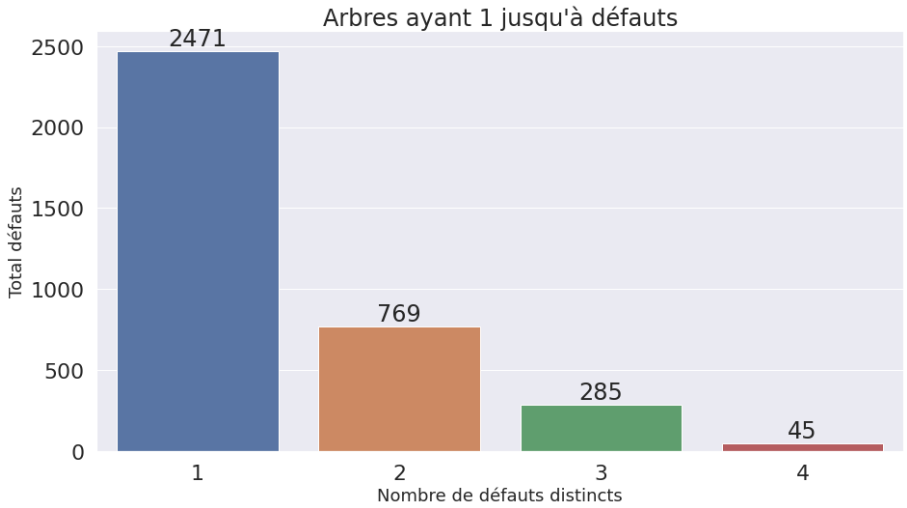


FIGURE 4.1 – après prétraitement
les arbres selon leur nombre de défauts

L'objectif de cette section est de présenter la réponse au problème de la classification multi-label de ce projet.

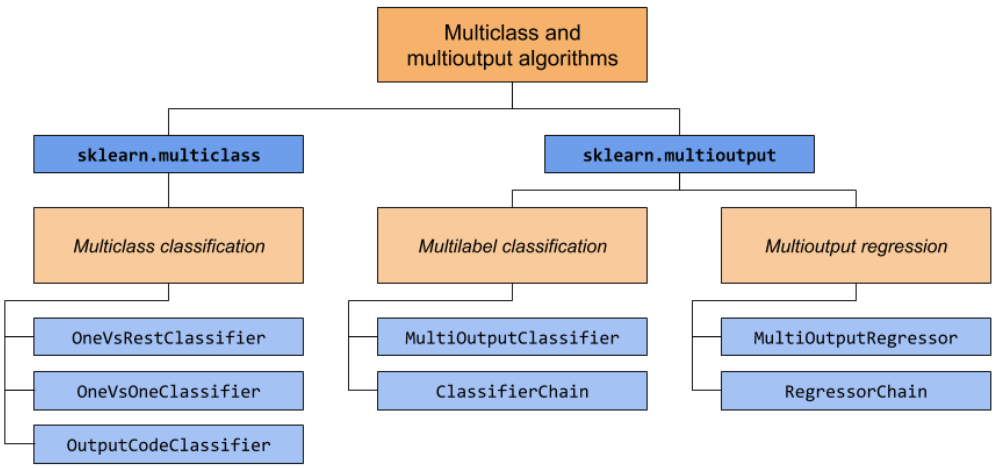


FIGURE 4.2 – Source : Scikit-learn

Nous allons considérer ce problème comme 4 problèmes de prédictions uni-label. L'un des deux algorithmes proposés par Scikit-learn a été mis en œuvre (le MultiOutputClassifier). Nous avons utilisé

trois classifieurs : Random Forest, Adaboost, et Xgboost. Ce choix car les méthodes d'ensemble profitent bien d'un grand nombre de données. Nous serions tentés par un classifieur probabiliste bayésien (GaussianNB, MultinomialNB, ou BernoulliNB) mais cela suppose que les caractéristiques des données soient indépendantes (livre Machine learning - les fondamentaux, de Matt Harrison, page 88), or, nous avons des corrélations dans nos données, avant et après prétraitement. De plus, nous l'avions testé pour MultinomialNB pour expérimenter, et l'accuracy était faible (en dessous de celle de la Base de référence). Nous gardons donc nos trois classifieurs choisis au départ.
Les métriques pour comparer les classifieurs sont toujours l'accuracy, la précision, et le rappel.

Remarques : comme il s'agit d'un problème multi-label, le paramètre average de la fonction precision_score, et de la fonction recall_score de Scikit-learn ne doit pas être par défaut (average = 'binary'). Nous avons donc modifié ce paramètre pour donc le choix de average = 'macro'. Ainsi, La précision est calculée comme la moyenne des précisions des classifieurs dédiés à chaque classe (Collet, Houppier, Racine, Tronc).

	accuracy	precision	recall
Random Forest	**0.75**	**0.78**	0.34
Adaboost	**0.71**	**0.56**	0.29
eXtreme Gradient Boosting (Xgboost)	**0.74**	**0.63**	**0.46**

TABLE 4.1 – scores des modèles multi-label

Résultats : Les valeurs en gras sont au-dessus du classifieur Baseline multi-label 64% pour l'exactitude (accuracy), 37% pour la précision et 37% de rappel.

D'après les résultats, pour le problème de la classification multi-label de ce projet, c'est Random Forest qui est l'alogorithme à priviligier pour l'accuracy et la précision, par contre Xgboost est le meilleur en rappel.

Chapitre 5

Conclusion

Nous avons à présent terminé les défis propres à ce jeu de donnée en utilisant différentes méthodes d'analyse de données, ainsi que de création de modèle pour un problème spécifique. Nous obtenons des précisions très correctes. avec les différentes méthodes de classifications unies et multi label utilisées. L'information présente dans ce document nous semble pouvoir aider les spécialistes en la matière afin de prendre des mesures.

```
inProj = pyproj.Proj("+init=EPSG:3945")
outProj = pyproj.Proj("+init=EPSG:4326")

data_coord = df.iloc[:,27:29]
for m in range(data_coord.shape[0]):
    data_coord['coord_x'][m],data_coord['coord_y'][m] =  pyproj.transform(inProj, outProj,
                                                        float(df.iloc[m,27]),
                                                        float(df.iloc[m,28]))
data_coord
```

	coord_x	coord_y
0	5.740495	45.172728
1	5.713406	45.170622
2	5.711751	45.179604
3	5.726057	45.186023
4	5.711043	45.194799
...	...	...
15370	5.709891	45.167862
15371	5.717215	45.174753
15372	5.711792	45.193785
15373	5.731925	45.173410
15374	5.698016	45.207024

15375 rows × 2 columns

FIGURE 5.1 – Conversion de coord x et coord x en longitude et latitude

il faut installer Pyproj, et folium sur Python, puis suivre le code du dessus et de la page suivante

```python
# coord y : latitude
# coord x : longitude

y_centre = data_coord['coord_y'][0]
x_centre = data_coord['coord_x'][0]
c = folium.Map(location=[y_centre, x_centre])

p = data_coord.shape[0]

for m in range(1,100):
    y = data_coord['coord_y'][m]
    x = data_coord['coord_x'][m]
    if df['DEFAUT'][m] == 1:
        folium.Marker(location=[y,x],icon=folium.Icon(color="red")).add_to(c)
    if df['DEFAUT'][m] == 0:
        folium.Marker(location=[y,x],icon=folium.Icon(color="blue")).add_to(c)

c.save('cartesArbres.html')
c
# visulaisation des 100 premiers arbres, en rouge (avec défaut), en bleu (sans défaut)
```

FIGURE 5.2 – Obtention de la carte géopgraphique

Chapitre 6

Annexe

6.1 logiciels à installer

```
# conda install -c conda-forge yellowbrick
# conda install -c conda-forge pyproj
# conda install -c conda-forge folium
```

6.2 bibliothèques importées

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import chi2, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
from xgboost import XGBClassifier
from yellowbrick.model_selection import LearningCurve
from yellowbrick.classifier import ROCAUC
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import recall_score, precision_score
import pyproj
import folium
import warnings
warnings.filterwarnings('ignore')
warnings.warn('DelftStack')
warnings.warn('Do not show this message')

```

6.3 code pour l'analyse exploratoire

```
def testQuant(num):
    # entrée : num (numéro de la colonne de la variable quantitative étudiée)
```

```
    # sortie : le résultat du test d'Anova de cette variable avec la cible
    data = pd.DataFrame({"x": df.iloc[:,num].values,"y" : df["DEFAUT"].astype("category").values})
    data = data.dropna()
    x, y = data.iloc[:,0].values.reshape(-1,1), data.iloc[:,1].values
    test_anova = f_classif(x,y)
    return pd.DataFrame({"F-stattistic" : test_anova[0], "P-value" : test_anova[1]},
                        index = [df.columns[num]])


def testQuali(num):
    # entrée : num (numéro de la colonne de la variable qualitative étudiée)
    # sortie : le résultat du test de chi2 de cette variable avec la cible
    data = pd.DataFrame({"x": LabelEncoder().fit_transform(df.iloc[:,num].values),"y" : df["DEFAUT"].values})
    data = data.dropna()
    x, y = data["x"].astype("category").values.reshape(-1,1), data["y"].astype("category").values
    test = chi2(x,y)
    return pd.DataFrame({"Chi2 stattistic" : test[0], "P-value" : test[1]},
                        index = [df.columns[num]])

def contingence(num):
    # entrée : num (numéro de la colonne de la variable qualitative étudiée)
    # sortie : table de contingence avec la cible

    return pd.crosstab(df.iloc[:,num].astype("category"),df["DEFAUT"].astype("category"),margins= False)

def descripQuali(num) :
    # entrée : num (numéro de la colonne de la variable qualitative étudiée)
    # sortie : le résultat de la description de cette variable
    return df.iloc[:,num].astype("category").describe()

def descripQuant(num) :
    # entrée : num (numéro de la colonne de la variable quantitative étudiée)
    # sortie : le résultat de la description de cette variable
    return df.iloc[:,num].describe()

def graphique(num) :
    # entrée : num (numéro de la colonne de la variable qualitative étudiée)
    # sortie : effectif de chaque modalité dans les arbres avec ou sans défaut
    return pd.crosstab(df[df.columns[num]].astype("category"),df["DEFAUT"].astype("category")).plot(
                        kind='bar',ylabel ='effectif')
```

6.4 codes du prétraitement

```
def changementTypeDonnees(df):

    df['ADR_SECTEUR'] = df['ADR_SECTEUR'].astype('object')
    df['ANNEEDEPLANTATION'] = df['ANNEEDEPLANTATION'].astype('object')
    df['ANNEEREALISATIONDIAGNOSTIC'] = df['ANNEEREALISATIONDIAGNOSTIC'].astype('object')
    df['ANNEETRAVAUXPRECONISESDIAG'] = df['ANNEETRAVAUXPRECONISESDIAG'].astype('object')
    df['DEFAUT'] = df['DEFAUT'].astype('object')
    df['Collet'] = df['Collet'].astype('object')
    df['Houppier'] = df['Houppier'].astype('object')
    df['Racine'] = df['Racine'].astype('object')
    df['Tronc'] = df['Tronc'].astype('object')
    return df


def supprimeVariable(df) :

    liste_variable = ['CODE','CODE_PARENT','CODE_PARENT_DESC','SOUS_CATEGORIE',
```

```
                        'RAISONDEPLANTATION', 'INTITULEPROTECTIONPLU','IDENTIFIANTPLU','TYPEIMPLANTATIONPLU',
                       'TRAITEMENTCHENILLES','VARIETE','REMARQUES']
    df = df.drop(liste_variable, axis=1)
    return df


def encodage(df):
    enconder =  LabelEncoder()
    for colonnes in df.select_dtypes('object').columns:
        df[colonnes] = enconder.fit_transform(df[colonnes])
    return df


def preprocessing(df):
    df = changementTypeDonnees(df)
    df = supprimeVariable(df)
    df = df.dropna()
    df = encodage(df)
    return df
```

6.5 codes classification uni-label

```
def evaluation(model) :
    model = model.fit(X = X_train,y = y_train)
    ypred = model.predict(X_test)
    print("matrice de confusion ",confusion_matrix(y_test,ypred), sep="\n")
    print("rapport de classification ", classification_report(y_test,ypred),sep="\n")


def graphiqueModel(model):
    visualizer = ROCAUC(model, classes=["0", "1"])
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    visualizer.show()


def graphiqueValidation(model):
    visualizer = LearningCurve(model, scoring='accuracy',cv = 10,train_sizes=np.linspace(0.1,1,10))
    visualizer.fit(X_train, y_train)
    visualizer.show()
```

6.6 codes classification multi-label avec MultiOutputClalssifier et Random forest

```
RF = MultiOutputClassifier(RandomForestClassifier(n_estimators = 500,max_depth=10))
RF.fit(X_train,y_train)
ypred_RF = RF.predict(X_test)
ypred_RF
# accuracy
RF.score(X_test,y_test)
# precision
precision_score(y_test,ypred_RF,average='macro')
# rappel
recall_score(y_test,ypred_RF,average='macro')
```

Bibliographie

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1) :5–32, 2001.

[Cho21] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021.

[FSA99] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780) :1612, 1999.

[Pet09] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2) :1883, 2009.