



**UNIVERSITÉ
DE LORRAINE**



Projet de Fouille de Données et Extraction de Connaissances :

L'analyse prédictive au service du
développement durable



Cyprien MANGEOT - Eugénie ROGER - Victor TOMMASINI

M2 IMSD 2022-2023

Table des matières

1	Introduction	2
2	Présentation des données	3
3	Prétraitement des données	3
3.1	Suppression de variables	3
3.2	Imputation des données manquantes	4
3.3	Réduction du nombre de modalités	6
3.4	Transformation des variables catégoriques en variables numériques	8
4	Défi 1 : Prédiction de la variable 'DEFAULT'	9
4.1	Tâche supervisée 1 : Classification uni-label	9
4.2	Tâche supervisée 2 : Classification multi-label	10
4.2.1	Prédiction de la variable 'Collet'	10
4.2.2	Prédiction de la variable 'Tronc'	11
4.2.3	Prédiction de la variable 'Houppier'	12
4.2.4	Prédiction de la variable 'Racine'	12
5	Défi 2 : État du parc végétal	13
5.1	Développer une carte interactive	13
5.2	Ajouter de nouvelles données	13
5.3	Modéliser le vieillissement	13
6	Conclusion	14
7	Annexe	15
8	Bibliographie	52

1 Introduction

A l'occasion de la conférence nationale EGC 2017, Big Datest, une entreprise grenobloise spécialisée dans l'analyse prédictive, s'est associée à la mairie de Grenoble pour présenter une base de données ainsi que deux défis concernant les arbres des espaces verts de la ville.

Le premier défi consiste en deux tâches de prédiction.

- La première vise à déterminer, à partir des données disponibles, si un arbre possède ou non un défaut. C'est de la classification uni-label.
- La deuxième vise quant à elle à déterminer la nature du défaut. Dans l'hypothèse où un arbre possède un défaut, il s'agira de prédire si le(s) défaut(s) se trouvent sur le tronc, sur le collet, sur le houppier et/ou sur la racine. C'est de la classification multi-label.

Pour cela nous procéderons à un nettoyage du jeu de données à travers différentes méthodes : suppression de variables, imputation de valeurs manquantes, réduction du nombre de modalités d'une variable, encodage des variables catégoriques. Nous allons ensuite construire différents modèles de Machine Learning pour tenter de trouver la meilleure prédiction.

Nous utiliserons les modèles suivants : DecisionTreeClassifier, RandomForestClassifier, RIPPER, BaggingClassifier (avec RIPPER), StackingClassifier, VotingClassifier, et AdaBoostClassifier. Certains de ces modèles seront boostés grâce à l'utilisation d'un Grid-Search.

Le second défi a pour objectif de mieux connaître l'état du parc végétal de Grenoble. Le but est de mieux comprendre son évolution, et de fournir des préconisations pour faciliter son entretien.

Pour cela, nous proposerons plusieurs pistes : notamment, le développement d'une carte interactive. Nous pourrions alors visualiser facilement des zones où les arbres semblent fortement touchés par des maladies. Nous pourrions aussi ajouter des données liées au climat pour voir leur influence sur la santé des arbres. Nous aborderons aussi la modélisation du vieillissement des arbres.

2 Présentation des données

La base de données présente les informations de 15375 arbres et contient 34 attributs :

- 27 variables explicatives décrivant l'état de l'arbre, son identification, sa localisation, ses caractéristiques, etc.
- 1 variable binaire 'DEFAULT' qui indique si l'arbre possède ou non un défaut
- 4 variables binaires 'Collet', 'Houppier', 'Racine' et 'Tronc' qui indiquent si l'arbre possède ou non un défaut aux endroits indiqués.

3 Prétraitement des données

Dans un premier temps, nous nous intéressons aux valeurs manquantes de notre base de données. La figure 1 présente le nom de chaque attribut et le nombre de valeurs manquantes qu'il contient.

ADR_SECTEUR	0	SOUS_CATEGORIE_DESC	0
ANNEEDEPLANTATION	0	STADEDEVELOPPEMENT	51
ANNEEREALISATIONDIAGNOSTIC	8	STADEDEVELOPPEMENTDIAG	13
ANNEETRAVAUXPRECONISESDIAG	4511	TRAITEMENTCHENILLES	14287
CODE	0	TRAVAUXPRECONISESDIAG	4525
CODE_PARENT	0	TROTTOIR	0
CODE_PARENT_DESC	0	TYPEIMPLANTATIONPLU	15014
DIAMETREARBREAUNMETRE	67	VARIETE	13212
ESPECE	1018	VIGUEUR	11
FREQUENTATIONCIBLE	1	coord_x	0
GENRE_BOTA	0	coord_y	0
IDENTIFIANTPLU	15014	DEFAULT	0
INTITULEPROTECTIONPLU	15014	Collet	0
NOTEDIAGNOSTIC	40	Houppier	0
PRIORITEDERENOUVELLEMENT	127	Racine	0
RAISONDEPLANTATION	15145	Tronc	0
REMARQUES	11176	dtype: int64	
SOUS_CATEGORIE	0		

FIGURE 1 – Nombre de valeurs manquantes pour chaque variable

3.1 Suppression de variables

Nous remarquons que certains attributs contiennent plus de 10000 valeurs manquantes, ce qui représente plus de deux tiers de l'effectif total. C'est le cas des variables 'IDENTIFIANTPLU', 'INTITULEPROTECTIONPLU', 'RAISONDEPLANTATION', 'REMARQUES', 'TRAITEMENTCHENILLES', 'TYPEIMPLANTATIONPLU' et 'VARIETE'. Nous choisissons de supprimer ces variables car leur taux de valeurs manquantes est trop élevé.

Nous pourrions conserver la variable 'REMARQUES' qui contient des commentaires d'experts. Cependant celle-ci pose plusieurs difficultés : elle contient énormément de valeurs manquantes et les valeurs disponibles peuvent être toutes différentes et par conséquent difficiles à traiter globalement.

Ensuite, certains attributs ne seront pas utiles à la prédiction des défauts de l'arbre. Nous supprimons donc les variables 'CODE', 'CODE_PARENT' et 'CODE_PARENT_DESC' car elles présentent respectivement le code de l'arbre, le code et l'adresse de son parent. De même, nous supprimons la variable 'SOUS_CATEGORIE' car elle est inutile à la prédiction du fait qu'elle contient exactement les mêmes informations que la variable 'SOUS_CATEGORIE_DESC'.

Dans la taxonomie des arbres et des plantes, le genre et l'espèce correspondent respectivement aux 6^{ème} et 7^{ème} niveaux de classification. Nous avons à notre disposition un attribut 'ESPECE' qui contient 1018 valeurs manquantes et un attribut 'GENRE_BOTA' qui n'en contient aucune. Nous choisissons alors de supprimer 'ESPECE' car le second contient des informations légèrement moins précises mais ne possède aucune valeur manquante.

Vous trouverez le code ayant permis de supprimer toutes ces variables en annexe . Grâce à ce processus, nous avons pu alléger notre dataset. Il ne nous reste alors plus que les variables suivantes :

		STADEDEVELOPPEMENTDIAG	13
ADR_SECTEUR	0	TRAVAUXPRECONISESDIAG	4525
ANNEEPLANTATION	0	TROTTOIR	0
ANNEEREALISATIONDIAGNOSTIC	8	VIGUEUR	11
ANNEETRAVAUXPRECONISESDIAG	4511	coord_x	0
DIAMETREARBREAUNMETRE	67	coord_y	0
FREQUENTATIONCIBLE	1	DEFAULT	0
GENRE_BOTA	0	Collet	0
NOTEDIAGNOSTIC	40	Houppier	0
PRIORITEDERENOUVELLEMENT	127	Racine	0
SOUS_CATEGORIE_DESC	0	Tronc	0
STADEDEVELOPPEMENT	51	dtype: int64	

FIGURE 2 – Nombre de valeurs manquantes pour chaque variable

3.2 Imputation des données manquantes

L'attribut 'TRAVAUXPRECONISESDIAG' indique le type de travaux qu'il faudra effectuer à l'année indiquée par l'attribut 'ANNEETRAVAUXPRECONISESDIAG'. Ces deux variables ont respectivement 4525 et 4511 valeurs manquantes et nous remarquons que la plupart des arbres présentant une valeur manquante pour la première variable présentent également une valeur manquante pour la seconde, et inversement. Nous remplaçons donc les valeurs manquantes de la manière suivante :

- Si un arbre possède une valeur manquante pour les attributs 'ANNEETRAVAUX-PRECONISESDIAG' et 'TRAVAUXPRECONISESDIAG' simultanément, nous attribuons la valeur 'Pas prévu' pour la première variable et la valeur 'Pas de travaux' pour la deuxième.
En effet, s'il n'y a aucune année préconisée pour des travaux ni de diagnostic des travaux à effectuer, nous considérons qu'il n'y a pas de travaux prévus.
- Si un arbre possède une valeur manquante uniquement pour l'attribut 'TRAVAUX-PRECONISESDIAG', nous remplaçons celle-ci par la valeur 'Contrôle'.
En effet, si une année est préconisée pour des travaux dont nous ne connaissons pas la nature, nous considérons qu'il faudra faire un contrôle.
- Si un arbre possède une valeur manquante uniquement pour l'attribut 'ANNEETRAVAUXPRECONISESDIAG', ce qui concerne seulement 12 arbres, nous choisissons de supprimer ces arbres plutôt que d'inventer une année de travaux préconisés.

Vous trouverez le code ayant permis cette imputation en annexe . Après traitement des variables détaillées précédemment, nous obtenons la figure 2 qui présente le nom de chaque variable conservée et le nombre de valeurs manquantes qu'elle contient.

ADR_SECTEUR	0	TRAVAUXPRECONISESDIAG	0
ANNEEDEPLANTATION	0	TROTTOIR	0
ANNEEREALISATIONDIAGNOSTIC	8	VIGUEUR	11
ANNEETRAVAUXPRECONISESDIAG	12	coord_x	0
DIAMETREARBREAUNMETRE	67	coord_y	0
FREQUENTATIONCIBLE	1	DEFAULT	0
GENRE_BOTA	0	Collet	0
NOTEDIAGNOSTIC	40	Houppier	0
PRIORITEDERENOUVELLEMENT	127	Racine	0
SOUS_CATEGORIE_DESC	0	Tronc	0
STADEDEVELOPPEMENT	51	dtype: int64	
STADEDEVELOPPEMENTDIAG	13		

FIGURE 3 – Nombre de valeurs manquantes après suppression et complétion de variables

Nous remarquons que les variables possédant des valeurs manquantes sont peu nombreuses. Nous regardons alors le nombre d'arbres qui possèdent des valeurs manquantes.

```

data1 = data1.dropna()

data1.isnull().sum()

```

ADR_SECTEUR	0	SOUS_CATEGORIE_DESC	0
ANNEEPLANTATION	0	STADEDEVELOPPEMENT	0
ANNEEREALISATIONDIAGNOSTIC	0	STADEDEVELOPPEMENTDIAG	0
ANNEETRAVAUXPRECONISESDIAG	0	TRAVAUXPRECONISESDIAG	0
DIAMETREARBREAUNMETRE	0	TROTTOIR	0
FREQUENTATIONCIBLE	0	VIGUEUR	0
GENRE_BOTA	0	coord_x	0
NOTEDIAGNOSTIC	0	coord_y	0
PRIORITEDERENOUVELLEMENT	0	DEFAULT	0
		Collet	0
		Houppier	0
		Racine	0
		Tronc	0
		dtype: int64	

FIGURE 4 – Nombre de valeurs manquantes pour chaque variable

Nous avons constaté que seulement 273 arbres (sur les 15375 de départ) possèdent au moins une valeur manquante. Nous décidons alors de les supprimer et nous nous retrouvons comme le montre la figure ci-dessus avec un dataset sans aucune valeur manquante. Notre base de données présente désormais les informations de 15102 arbres et contient 22 attributs.

3.3 Réduction du nombre de modalités

Nous voudrions pouvoir convertir les variables catégoriques en numériques. Mais avant ça, nous regardons le nombre de modalités par variables pour s'assurer qu'elles n'ont pas un nombre de modalités trop grand. La figure ci-dessous nous renseigne :

```

Nombre de modalités de ANNEEPLANTATION : 12
Nombre de modalités de ANNEEREALISATIONDIAGNOSTIC : 5
Nombre de modalités de ANNEETRAVAUXPRECONISESDIAG : 12
Nombre de modalités de DIAMETREARBREAUNMETRE : 19
Nombre de modalités de FREQUENTATIONCIBLE : 3
Nombre de modalités de GENRE_BOTA : 107
Nombre de modalités de NOTEDIAGNOSTIC : 5
Nombre de modalités de PRIORITEDERENOUVELLEMENT : 4
Nombre de modalités de SOUS_CATEGORIE_DESC : 4
Nombre de modalités de STADEDEVELOPPEMENT : 3
Nombre de modalités de STADEDEVELOPPEMENTDIAG : 3
Nombre de modalités de TRAVAUXPRECONISESDIAG : 16
Nombre de modalités de TROTTOIR : 2
Nombre de modalités de VIGUEUR : 3

```

FIGURE 5 – Nombre de modalités pour chaque variable

Nous remarquons que les attributs 'DIAMETREARBREAUNMETRE' et 'GENRE_BOTA' possèdent beaucoup de valeurs différentes. Nous modifions alors certaines de leurs modalités de la manière suivante :

Pour la variable 'DIAMETREARBREAUNMETRE' :

Nous décidons de regarder la répartition des modalités car elle en possède 19. La figure ci-dessous nous renseigne sur ce point :

```
data2['DIAMETREARBREAUNMETRE'].value_counts()
```

```
10 à 20 cm      4173
20 à 30 cm      3146
30 à 40 cm      2369
40 à 50 cm      1865
50 à 60 cm      1108
0 à 10 cm       921
60 à 70 cm      645
70 à 80 cm      379
80 à 90 cm      233
90 à 100 cm     137
100 à 110 cm    63
110 à 120 cm    26
120 à 130 cm    14
130 à 140 cm    11
140 à 150 cm     4
150 à 160 cm     4
160 à 170 cm     2
180 à 190 cm     1
170 à 180 cm     1
Name: DIAMETREARBREAUNMETRE, dtype: int64
```

Nous décidons de créer une modalité '60 à 80 cm' regroupant '60 à 70 cm' et '70 à 80 cm', et une autre '80 à 180 cm' remplaçant toutes les modalités de cet intervalle. Après cette transformation, l'attribut 'DIAMETREARBREAUMETRE' ne contient que 8 modalités. Vous trouverez le code ayant permis cette manipulation en annexe .

Voici la répartition des modalités après réduction du nombre de modalités :

```
10 à 20 cm      4173
20 à 30 cm      3146
30 à 40 cm      2369
40 à 50 cm      1865
50 à 60 cm      1108
60 à 80 cm      1024
0 à 10 cm       921
80 à 180 cm     496
Name: DIAMETREARBREAUNMETRE, dtype: int64
```

Pour la variable 'GENRE_BOTA' :

La variable possède 107 modalités, c'est beaucoup trop. Nous choisissons de conserver les modalités qui concernent plus de 100 arbres dans notre base de données et de changer les autres catégories en 'Autre' pour ne garder que 25 modalités pour l'attribut 'GENRE_BOTA'. Vous trouverez le code ayant permis de faire cette modification en annexe .

Voici la répartition des modalités après modification :

```
data2['GENRE_BOTA'].value_counts()
```

```
Platanus      2956
Acer           2280
Autre         1635
Tilia         988
Pinus         969
Betula        685
Populus       668
Fraxinus      641
Liquidambar   470
Carpinus      408
Prunus        366
Quercus       305
Alnus         291
Liriodendron  288
Sophora       269
Chamaecyparis 242
Aesculus      240
Magnolia      238
Pyrus         231
Cedrus        219
Cercis        169
Gleditsia     165
Robinia       154
Picea         114
Cupressus     111
Name: GENRE_BOTA, dtype: int64
```

Nous pourrions également remplacer chaque modalité par la famille correspondante. En effet dans la taxonomie des arbres et des plantes, le genre et la famille sont respectivement les 6^{ème} et 5^{ème} niveaux de classification, il y aurait donc moins de modalités car certains genres appartiennent à la même famille.

Nous pourrions aussi modifier quelques unes des 16 modalités de l'attribut 'TRAVAUX-PRECONISESDIAG' mais celles-ci sont assez spécifiques et sont certainement déterminantes dans l'étude des défauts de l'arbre. Nous choisissons donc de les conserver totalement.

Maintenant que chaque variable catégorique contient un nombre raisonné de modalités, nous allons donc pouvoir passer à l'étape d'encodage.

3.4 Transformation des variables catégoriques en variables numériques

Certaines variables sont catégoriques. Nous les transformons en variables numériques. C'est le cas des attributs 'ANNEEDEPLANTATION', 'ANNEEREALISATIONDIAGNOSTIC', 'ANNEETRAVAUXPRECONISESDIAG', 'DIAMETREARBREAUMETRE', 'FREQUENTATIONCIBLE', 'GENRE_BOTA', 'NOTEDIAGNOSTIC', 'PRIORITEDERENOUVELLEMENT', 'SOUS_CATEGORIE_DESC', 'STADEDEDEVELOPPEMENT', 'STADEDEVELOPPEMENTDIAG', 'TRAVAUXPRECONISESDIAG', 'TROTTOIR' et 'VIGUEUR'. Vous trouverez le code permettant d'encoder ces variables en numérique en annexe .

4 Défi 1 : Prédications de la variable 'DEFAULT'

4.1 Tâche supervisée 1 : Classification uni-label

Pour réaliser cette tâche de prédiction, nous commençons par construire X qui est l'ensemble de nos variables descriptives. Toutes nos variables précédentes sauf 'Houppier', 'Collet', 'Tronc' et 'Racine', qui sont inutiles pour cette prédiction. Et surtout elles contiennent déjà l'information de la présence d'un défaut.

```
data3 = data2.drop(columns=["Collet", "Houppier", "Racine", "Tronc"])
```

	ADE	SECTEUR	ANNEESPLANIFICATION	ANNEEREAISATIONDIAGNOSTIC	ANNEETRAVAUXPRECONSEISSDIAG	DIAMETREARBREALIMETRE	FREQUENTRIENOCIBLE	GENIE_BOIA	NOTEDIAGNOSTIC	PRIORITYDERENOUVELLEMENT	SOUS_CATEGORIE_DESC	STADEDEVELOPPEMENT	STADEDEVELOPPEMENTDIAG	TRAVAUXPRECONSEISSDIAG	TROTTORIE	VIGILEUR	coord_x	coord_y	
0	5	100	30	70	10	10	20	10	30	00	10	10	30	00	10	30	00	1.0	1.0
1	3	00	20	110	30	00	170	10	30	00	00	00	00	80	00	10	1.0	1.0	
2	2	00	30	20	40	00	230	10	30	00	00	00	00	140	10	10	1.0	1.0	
3	1	00	20	110	20	00	100	10	30	00	00	00	00	00	10	10	1.0	1.0	
4	3	00	40	60	40	00	150	30	00	00	00	00	00	40	00	10	1.0	1.0	
...
15097	3	00	20	110	00	00	160	10	30	00	00	00	10	80	00	10	1.0	1.0	
15098	4	00	30	20	40	00	170	10	30	00	00	00	00	110	00	10	1.0	1.0	
15099	1	00	20	110	10	00	100	10	30	00	00	00	00	00	00	10	1.0	1.0	
15100	4	00	30	110	10	00	120	10	30	00	00	00	00	80	10	10	1.0	1.0	
15101	1	100	30	70	10	10	100	10	30	00	00	10	10	30	00	10	1.0	1.0	

Nous construisons également y comme étant notre variable à prédire, ici c'est DEFAULT.

Nous utilisons ensuite la fonction `train_test_split` pour séparer nos données en deux : un set d'apprentissage et un set de test. Nous décidons de construire plusieurs modèles de prédiction afin de pouvoir comparer les performances des modèles pour avoir la meilleure prédiction. Pour pouvoir fiabiliser nos comparaisons, nous fixerons l'aléatoire en posant un `random_state = 0` dans chaque modèle.

```
import sklearn
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, shuffle=True, random_state=100)
```

Nous allons construire les modèles suivants :

- Un modèle d'Arbre de décision (`max_depth = 6`).
- Une modèle de Forêt aléatoire (optimisée grâce à un GridSearchCV)
- Un modèle RIPPER
- Un modèle de Bagging-RIPPER
- Un modèle Stacking Classifier (optimisée grâce à un GridSearchCV)
- Un modèle Voting Classifier
- Un modèle AdaBoost Classifier (optimisée grâce à un GridSearchCV)

Vous trouverez le code et des explications concernant la construction de chaque modèle en cliquant sur leur nom.

Nous affichons dans le tableau ci-dessous plusieurs métriques afin d'évaluer nos différents modèles de prédiction :

Modèle	Accuracy	Score Cross-validation	Précision	Rappel	F1-score
Arbre de décision	85.6%	85.1%	89.3%	62.8%	73.7%
Forêt aléatoire	84.4%	84.0%	85.0%	62.5%	72.0%
JRip	82.1%	83.9%	88.8%	50.9%	64.7%
Bagging (JRip)	80.7%	80.8%	85.6%	48.3%	61.8%
Stacking	76.3%	75.1%	73.9%	40.9%	52.7%
Voting	84.8%	83.5%	75.2%	78.9%	77.0%
AdaBoost	84.8%	84.3%	85.6%	63.3%	72.8%

Pour la prédiction de la variable 'DEFAULT', nous voudrions essayer de choisir un modèle qui sort du lot. Nous pourrions penser que l'arbre de décision est le meilleur modèle car il a une très bonne accuracy et une très bonne précision. Cependant, les résultats du Rappel sont décevants. En fait l'arbre de décision aura tendance à dire trop facilement qu'un arbre possède un défaut. Ce qui explique que nous ayons une bonne précision.

Nous allons donc nous pencher sur le F1-Score qui est un bon compromis entre la précision et le rappel. Nous remarquons que le modèle de Voting est plutôt bon. L'accuracy est quasiment la même et c'est le modèle qui nous offre le meilleur F1-score. Nous aurons donc tendance à privilégier ce modèle.

Notre modèle de Voting est composé de trois modèles : Un SGDClassifier, un DecisionTreeClassifier, un RIPPER. Nous avons choisi un *voting='hard'*, ce qui veut dire que nous choisirons la prédiction majoritaire.

4.2 Tâche supervisée 2 : Classification multi-label

Nous commençons par créer un nouveau X et des nouveaux y (y_houppier, y_racine, y_tronc et y_collet). Ces variables nous serviront pour construire les modèles de prédiction. Voici comment nous avons procédé :

```
data_part2=data2[data2.DEFAULT == 1]

data_part2.shape

Xbis = data_part2.drop(["Houppier", "Racine", "Collet", "Tronc", "DEFAULT"], axis=1)
y_houppier = data_part2["Houppier"]
y_racine = data_part2["Racine"]
y_tronc = data_part2["Collet"]
y_collet = data_part2["Tronc"]
```

4.2.1 Prédiction de la variable 'Collet'

Nous commençons par utiliser la fonction *train_test_split* avec *Xbis* et *y_collet*.

```
X_train_collet, X_test_collet, y_train_collet, y_test_collet = train_test_split(Xbis, y_collet, test_size=0.2, shuffle=True, random_state=100)
```

Nous construisons les mêmes modèles en suivant les mêmes procédés qu'avant. Vous trouverez le code annexe .

Modèle	Accuracy	Score Cross-validation	Précision	Rappel	F1-score
Arbre de décision	66.6%	67.0%	60.4%	60.4%	60.4%
Forêt aléatoire	63.7%	65.2%	69.8%	24.8%	36.6%
JRip	61.2%	62.8%	72.3%	13.4%	22.6%
Bagging (JRip)	61.2%	63.5%	65.4%	17.5%	27.6%
Stacking	64.3%	64.2%	55.0%	83.1%	66.2%
Voting	66.8%	64.3%	60.3%	62.6%	61.4%
AdaBoost	64.7%	68.1%	61.6%	43.9%	51.2%

Nous constatons directement que trois modèles sortent du lot : L'arbre de décision, le Stacking, et le Voting.

Nous pouvons d'ores et déjà éliminer l'arbre de décision car il a des résultats très similaires au modèle Voting, mais ils sont en globalité un peu moins bon.

Si nous voulons un modèle qui détecte absolument les arbres possédant un défaut, quitte à se tromper (avoir des Faux-Négatifs), nous privilégierons le modèle de Stacking. Ce modèle a un très bon rappel, bien que sa précision en soit affectée. En revanche, si nous cherchons plutôt un modèle équilibré, alors nous choisirons plutôt le modèle de Voting.

4.2.2 Prédiction de la variable 'Tronc'

Nous commençons par utiliser la fonction `train_test_split` avec `Xbis` et `y_tronc`.

```
X_train_tronc, X_test_tronc, y_train_tronc, y_test_tronc = train_test_split(Xbis, y_tronc, test_size=0.2, shuffle=True, random_state=100)
```

Nous construisons les mêmes modèles en suivant les mêmes procédés qu'avant. Vous trouverez le code en annexe .

Modèle	Accuracy	Score Cross-validation	Précision	Rappel	F1-score
Arbre de décision	85.1%	84.3%	65.4%	31.3%	42.3%
Forêt aléatoire	83.5%	83.2%	80.0%	7.1%	13.0%
JRip	84.0%	83.9%	68.4%	15.3%	25.1%
Bagging (JRip)	83.0%	82.9%	64.2%	5.3%	9.8%
Stacking	82.5%	82.1%	0%	0%	0%
Voting	84.0%	84.1%	71.8%	13.6%	22.8%
AdaBoost	83.6%	83.7%	62.5%	14.7%	23.9%

Nous choisirons le modèle d'Arbre de décision qui se démarque à travers le rappel et le F1-score. Nous pourrions tenir le même raisonnement qu'avant mais les rappels/f1-score sont vraiment trop mauvais.

4.2.3 Prédiction de la variable 'Houppier'

Nous commençons par utiliser la fonction `train_test_split` avec `Xbis` et `y_houppier`.

```
X_train_houppier, X_test_houppier, y_train_houppier, y_test_houppier = train_test_split(Xbis, y_houppier, test_size=0.2, shuffle=True, random_state=100)
```

Nous construisons les mêmes modèles en suivant les mêmes procédés qu'avant. Vous trouverez le code en annexe .

Modèle	Accuracy	Score Cross-validation	Précision	Rappel	F1-score
Arbre de décision	71.8%	72.8%	74.7%	86.7%	80.2%
Forêt aléatoire	69.3%	70.8%	69.5%	95.3%	80.3%
JRip	53.2%	55.5%	94.3%	31.0%	46.7%
Bagging (JRip)	66.0%	66.5%	66.0%	100%	79.5%
Stacking	74.2%	66.8%	74.7%	92.0%	82.5%
Voting	59.2%	71.1%	88.8%	43.6%	58.5%
AdaBoost	74.0%	74.0%	75.6%	89.5%	82.0%

Nous constatons directement que deux modèles sortent du lot : le Stacking et AdaBoost.

Si nous voulons un modèle qui détecte absolument les arbres possédant un défaut, quitte à se tromper (avoir des Faux-Négatifs), nous privilégierons le modèle de Stacking qui a un rappel de 92%. Sinon, nous choisirons le modèle AdaBoost.

Dans tous les cas, ces deux modèles ont de très bonnes performances, surtout le Stacking.

4.2.4 Prédiction de la variable 'Racine'

Nous commençons par utiliser la fonction `train_test_split` avec `Xbis` et `y_racine`.

```
X_train_racine, X_test_racine, y_train_racine, y_test_racine = train_test_split(Xbis, y_racine, test_size=0.2, shuffle=True, random_state=100)
```

Nous construisons les mêmes modèles en suivant les mêmes procédés qu'avant. Vous trouverez le code en annexe .

Modèle	Accuracy	Score Cross-validation	Précision	Rappel	F1-score
Arbre de décision	87.7%	85.5%	62.5%	8.1%	14.3%
Forêt aléatoire	87.3%	85.5%	0%	0%	0%
JRip	86.4%	85.8%	28.5%	4.8%	8.3%
Bagging (JRip)	87.2%	85.7%	0%	0%	0%
Stacking	87.3%	85.5%	0%	0%	0%
Voting	83.1%	85.8%	37.8%	52.0%	43.8%
AdaBoost	88.0%	85.9%	76.9%	8.1%	14.7%

Nous choisirons le modèle de Voting qui se démarque à travers le rappel et le F1-score. Certes, nous avons une précision amoindrie, mais les autres modèles nous fournissent des rappels et des f1-score de très mauvaise qualité.

5 Défi 2 : État du parc végétal

La seconde tâche, plus ouverte, vise à mieux connaître l'état du « parc végétal » de Grenoble, mieux comprendre son évolution et fournir des préconisations pour faciliter son entretien.

Pour cette partie, nous présenterons quelques idées qui pourront être développées et approfondies.

5.1 Développer une carte interactive

Les données, contenant les coordonnées de chaque arbre, pourraient être visualisées sur une carte interactive.

Cette carte pourra servir aux employés de la ville de Grenoble, et permettrait de visualiser certains arbres selon certains critères, en particulier la présence, ou le nombre, de défauts. Ainsi, il pourra être plus facile de repérer des zones où les arbres sont en mauvaise santé, ce qui permettrait de localiser les traitements.

Un tel outil servirait également dans la sélection de lieux pour la plantation de nouveaux arbres, par exemple en choisissant une zone où les arbres développent moins de défauts, ou une zone encore peu végétalisée.

5.2 Ajouter de nouvelles données

Pour améliorer les modèles de prédiction développés dans la première partie, il pourrait être utile d'utiliser des données externes supplémentaires. D'abord, des données climatiques de précipitations, d'ensoleillement, de vent, depuis la plantation des arbres, jusqu'au diagnostic.

En plus d'une éventuelle amélioration des modèles, ces informations donneraient des indications sur les effets du climat, notamment la sécheresse, sur l'état du parc végétal. De même, des données sur la qualité de l'air apporteraient un éclairage sur l'impact de la pollution sur la santé des arbres en ville.

5.3 Modéliser le vieillissement

Les arbres les plus anciens sont plus susceptibles de développer un défaut. En effet, près de 90% des arbres décrits comme "vieillissants" dans la base de données présentent un défaut contre seulement 12% des arbres "jeunes".

Ainsi, la modélisation du vieillissement des arbres serait particulièrement utile dans la gestion des espaces verts de la ville. Cette modélisation prendrait en compte les variables affectées par le vieillissement, comme son stade de développement, sa vigueur, le diamètre de son tronc, ou encore sa note de diagnostic.

En simulant le vieillissement de la base de données de n années (pour $n = 3, 5, 10$ par exemple), nous pourrions estimer le nombre d'arbres actuellement plantés qui présenteront un défaut dans n années.

6 Conclusion

Après avoir nettoyé la base de données, nous avons appliqué différents algorithmes de classification supervisés, nous avons réalisé les tâches de prédiction de défauts sur les arbres de la ville de Grenoble. Les résultats obtenus nous semblent corrects pour la prédiction d'un défaut et satisfaisantes pour les parties du houppier et du collet.

Le modèle de Voting Classifier nous permet d'atteindre une accuracy de 84.8% et un F1-score de 77.0% pour la prédiction de la variable 'DEFAULT'.

Le modèle Stacking Classifier semble être le meilleur pour prédire les variables 'Collet' et 'Houppier'. Pour la prédiction de la variable 'Collet', il nous donne une accuracy de 64.3% et un F1-score de 66.2%. Pour la prédiction de la variable 'Houppier', nous obtenons une accuracy de 74.2% et un F1-score de 82.5%.

Nous pouvons rencontrer des difficultés à détecter un défaut sur certaines parties de l'arbre : sur le tronc ou sur les racines. L'arbre de décision nous offre une accuracy de 85.1% et un F1-score de 42.3% pour la prédiction de la variable 'Tronc'. La variable 'Racine' sera quant à elle mieux prédite avec un modèle Voting Classifier. Ce modèle nous donne une accuracy de 83.1% et un F1-score de 43.8%.

Dans certains cas, le choix du modèle pourra résulter d'un choix entre la volonté de ne pas laisser passer un arbre présentant un défaut (rappel plus important) ou celle d'avoir une prédiction plus fiable (précision plus forte).

Pour cette partie de prédiction, des améliorations pourraient être obtenues avec une analyse textuelle poussée de certaines variables de diagnostic, comme celle des "remarques". D'autres algorithmes de classification pourraient également être essayés, en particulier une classification multi-label pour la localisation de défauts.

Dans un second temps, nous avons développé des idées pour aider à mieux connaître l'état actuel du parc végétal de la ville, sous la forme d'une carte interactive, l'analyse de l'influence du climat et de la pollution ; et enfin son état futur avec une modélisation du vieillissement des arbres.

7 Annexe

Code concernant la suppression des variables inutiles : Retour au document.

```
data1 = data1.drop(columns=["IDENTIFIANTPLU", "INTITULEPROTECTIONPLU", "RAISONDEPLANTATION",  
                           "REMARQUES", "TRAITEMENTCHENILLES", "TYPEIMPLANTATIONPLU", "VARIETE"])
```

```
data1 = data1.drop(columns=["CODE", "CODE_PARENT", "CODE_PARENT_DESC", "SOUS_CATEGORIE"])
```

```
data1 = data1.drop(columns=["ESPECE"])
```

Code concernant l'imputation des valeurs manquantes : Retour au document.

```
annee_travaux_null = data1.ANNEETRAVAUXPRECONISESDIAG.isnull()  
travaux_prec_null = data1.TRAVAUXPRECONISESDIAG.isnull()  
  
for i in range(len(data1.ANNEETRAVAUXPRECONISESDIAG)):  
    if annee_travaux_null[i] and travaux_prec_null[i]:  
        data1.ANNEETRAVAUXPRECONISESDIAG[i] = "Pas prévu"  
        data1.TRAVAUXPRECONISESDIAG[i] = "Pas de travaux"  
data1.loc[:, ['TRAVAUXPRECONISESDIAG']] = data1.loc[:, ['TRAVAUXPRECONISESDIAG']].replace(np.nan, "Controle")
```

Code concernant la réduction du nombre de modalités de la variable 'DIAMETREARBREAUNMETRE' Retour au document.

```
data2.DIAMETREARBREAUNMETRE.replace(['60 à 70 cm', '70 à 80 cm'], '60 à 80 cm', inplace=True)  
data2['DIAMETREARBREAUNMETRE'].value_counts()
```

```
10 à 20 cm      4173  
20 à 30 cm      3146  
30 à 40 cm      2369  
40 à 50 cm      1865  
50 à 60 cm      1108  
60 à 80 cm      1024  
0 à 10 cm       921  
80 à 90 cm      233  
90 à 100 cm     137  
100 à 110 cm    63  
110 à 120 cm    26  
120 à 130 cm    14  
130 à 140 cm    11  
140 à 150 cm     4  
150 à 160 cm     4  
160 à 170 cm     2  
180 à 190 cm     1  
170 à 180 cm     1  
Name: DIAMETREARBREAUNMETRE, dtype: int64
```

```
data2.DIAMETREARBREAUNMETRE.replace(['80 à 90 cm', '90 à 100 cm', '100 à 110 cm', '110 à 120 cm',
                                     '130 à 140 cm', '140 à 150 cm', '150 à 160 cm', '160 à 170',
                                     '170 à 180 cm'], '80 à 180 cm', inplace=True)
data2['DIAMETREARBREAUNMETRE'].value_counts()
```

```
10 à 20 cm    4173
20 à 30 cm    3146
30 à 40 cm    2369
40 à 50 cm    1865
50 à 60 cm    1108
60 à 80 cm    1024
0 à 10 cm     921
80 à 180 cm   496
```

```
Name: DIAMETREARBREAUNMETRE, dtype: int64
```

Code concernant la réduction du nombre de modalités de la variable 'GENRE_BOTA' :

Retour au document. Nous créons une liste Genre_new contenant les modalités qui concernent plus de 100 arbres.

```
Genre_new=[]
Genre=data2.GENRE_BOTA.to_list()
while Genre!=[]:
    ref=Genre[0]
    occ=Genre.count(ref)
    if occ<100:
        Genre=list(filter((ref).__ne__,Genre))
    else:
        Genre_new.append(ref)
        Genre=list(filter((ref).__ne__,Genre))
```

Nous parcourons la colonne GENRE_BOTA de notre dataset, si la modalité n'appartient pas à Genre_new, on la remplace par 'Autre'.

```
for i in range(len(data2.GENRE_BOTA)):
    if (data2.GENRE_BOTA[i] not in Genre_new):
        data2['GENRE_BOTA'][i]="Autre"
```

Code concernant la transformation des variables catégoriques en variables numériques : Retour au document.

```
from sklearn.preprocessing import OrdinalEncoder

enc = OrdinalEncoder()
features_cat = ["ANNEEPLANTATION", "ANNEEREALISATIONDIAGNOSTIC", "ANNEETRAVAUXPRECONISESDIAG",
               "DIAMETREARBREAUNMETRE", "FREQUENTATIONCIBLE", "GENRE_BOTA", "NOTEDIAGNOSTIC",
               "PRIORITEDERENOUVELLEMENT", "SOUS_CATEGORIE_DESC", "STADEDEVELOPPEMENT",
               "STADEDEVELOPPEMENTDIAG", "TRAVAUXPRECONISESDIAG", "TROTTOIR", "VIGUEUR"]

for i in features_cat :
    data2[i]=enc.fit_transform(data2[[i]])
```

Code concernant le modèle d'Arbre de décision pour la prédiction de la variable **DEFAULT** : Retour au document.

```
from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(max_depth=6, random_state=100)
model_tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=100)
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_tree = cross_val_score(model_tree, X, y, cv=10)
print(res_cv_tree)
print(mean(res_cv_tree))
```

```
[0.85175381 0.85175381 0.86423841 0.85231788 0.84834437 0.84900662
 0.85298013 0.84437086 0.85165563 0.84503311]
0.8511454630721289
```

```
from sklearn.metrics import accuracy_score
y_pred_tree = model_tree.predict(X_test)
print("Accuracy du modèle :",accuracy_score(y_test, y_pred_tree))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_tree))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test, y_pred_tree))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test, y_pred_tree))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test, y_pred_tree))
```

```
Accuracy du modèle : 0.8563389606090699
Matrice de confusion du modèle :
[[1976  73]
 [ 361 611]]
Précision du modèle : 0.8932748538011696
Rappel du modèle : 0.6286008230452675
F1 Score du modèle : 0.7379227053140096
```

Code concernant le modèle de Forêt aléatoire pour la prédiction de la variable
DEFAUT : Retour au document.

```
from sklearn.model_selection import GridSearchCV
param_grid_random_forest = {'n_estimators':[100, 200,500,1000], 'criterion':['gini','entropy'],'max_depth':[2,3]}

from sklearn.ensemble import RandomForestClassifier
grid_random_forest = GridSearchCV(RandomForestClassifier(random_state=100),param_grid_random_forest,cv=10)
grid_random_forest.fit(X_train,y_train)

print(grid_random_forest.best_params_)

model_random_forest = grid_random_forest.best_estimator_
print(model_random_forest.score(X_test,y_test))

{'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 500}
0.8440913604766633
```

Evaluation du modèle par cross-validation :

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_random_forest = cross_val_score(model_random_forest, X, y, cv=10)
print(res_cv_random_forest)
print(mean(res_cv_random_forest))

[0.8365321 0.84315023 0.84503311 0.8410596 0.84238411 0.84503311
 0.84304636 0.82715232 0.84900662 0.83245033]
0.8404847892496965
```

Différentes métriques :

```
y_pred_random_forest = model_random_forest.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test, y_pred_random_forest))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_random_forest))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test, y_pred_random_forest))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test, y_pred_random_forest))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test, y_pred_random_forest))
```

```
Accuracy du modèle : 0.8440913604766633
Matrice de confusion du modèle :
[[1942 107]
 [ 364 608]]
Précision du modèle : 0.8503496503496504
Rappel du modèle : 0.6255144032921811
F1 Score du modèle : 0.7208061647895674
```

Code concernant le modèle Ripper pour la prédiction de la variable DEFAUT :
Retour au document.

```
import wittgenstein as lw
```

```
model_ripper = lw.RIPPER(random_state=100)  
model_ripper.fit(X_train, y_train)  
print(model_ripper.score(X_test, y_test))
```

```
0.821582257530619
```

```
from sklearn.model_selection import cross_val_score  
from statistics import mean  
res_cv_ripper = cross_val_score(model_ripper, X, y, cv=10)  
print(res_cv_ripper)  
print(mean(res_cv_ripper))
```

```
[0.85307743 0.83851754 0.84834437 0.84172185 0.83708609 0.84039735  
 0.83576159 0.8218543 0.84238411 0.83178808]  
0.8390932718562769
```

```
y_pred_ripper = model_ripper.predict(X_test)
```

```
from sklearn.metrics import accuracy_score  
print("Accuracy du modèle :",accuracy_score(y_test, y_pred_ripper))
```

```
from sklearn.metrics import confusion_matrix  
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_ripper))
```

```
from sklearn.metrics import precision_score  
print("Précision du modèle :",precision_score(y_test, y_pred_ripper))
```

```
from sklearn.metrics import recall_score  
print("Rappel du modèle :",recall_score(y_test, y_pred_ripper))
```

```
from sklearn.metrics import f1_score  
print("F1 Score du modèle :",f1_score(y_test, y_pred_ripper))
```

```
Accuracy du modèle : 0.821582257530619
```

```
Matrice de confusion du modèle :
```

```
[[1987  62]  
 [ 477 495]]
```

```
Précision du modèle : 0.8886894075403949
```

```
Rappel du modèle : 0.5092592592592593
```

```
F1 Score du modèle : 0.6474820143884893
```

Code concernant le modèle de Bagging-Ripper pour la prédiction de la variable
DEFAUT : Retour au document.

```
from sklearn.ensemble import BaggingClassifier

model_bagging_ripper = BaggingClassifier(base_estimator = lw.RIPPER(max_rules=3, random_state=100), n_estimators = 10, random_state=100)
model_bagging_ripper.fit(X_train,y_train)

print(model_bagging_ripper.score(X_test,y_test))

0.8076795762992387
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_bagging_ripper = cross_val_score(model_bagging_ripper, X, y, cv=10)
print(res_cv_bagging_ripper)
print(mean(res_cv_bagging_ripper))
```

```
[0.80344143 0.80741231 0.81324503 0.80264901 0.81390728 0.81456954
 0.80993377 0.80066225 0.81125828 0.80596026]
0.8083039169709109
```

```
y_pred_bag_rip = model_bagging_ripper.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test, y_pred_bag_rip))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_bag_rip))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test, y_pred_bag_rip))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test, y_pred_bag_rip))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test, y_pred_bag_rip))
```

```
Accuraciy du modèle : 0.8076795762992387
Matrice de confusion du modèle :
[[1970  79]
 [ 502 470]]
Précision du modèle : 0.8561020036429873
Rappel du modèle : 0.4835390946502058
F1 Score du modèle : 0.6180144641683104
```

Code concernant le modèle Stacking Classifier pour la prédiction de la variable DEFAUT : Retour au document.

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = KNeighborsClassifier(n_neighbors=2)

from sklearn.model_selection import GridSearchCV
param_grid_stacking = {'final_estimator':[SGDClassifier(random_state=100),DecisionTreeClassifier(random_state=100), KNeighborsClassifier()]}

from sklearn.ensemble import StackingClassifier
grid_stacking = GridSearchCV(StackingClassifier([('SGD',model_1),('Tree',model_2),('KNN',model_3)]),param_grid_stacking,cv=10)
grid_stacking.fit(X_train,y_train)

print(grid_stacking.best_params_)
model_stacking = grid_stacking.best_estimator_
print(model_stacking.score(X_test,y_test))

{'final_estimator': DecisionTreeClassifier(random_state=100)}
0.7636544190665343
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_stacking = cross_val_score(model_stacking, X, y, cv=10)
print(res_cv_stacking)
print(mean(res_cv_stacking))
```

```
[0.82859034 0.8365321 0.84503311 0.71788079 0.75231788 0.56887417
 0.82847682 0.73907285 0.69403974 0.70794702]
0.7518764819579157
```

```
y_pred_stacking = model_stacking.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test, y_pred_stacking))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_stacking))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test, y_pred_stacking))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test, y_pred_stacking))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test, y_pred_stacking))
```

```
Accuraciy du modèle : 0.7636544190665343
Matrice de confusion du modèle :
[[1909 140]
 [ 574 398]]
Précision du modèle : 0.7397769516728625
Rappel du modèle : 0.4094650205761317
F1 Score du modèle : 0.5271523178807948
```

Code concernant le modèle Voting Classifier pour la prédiction de la variable
DEFAUT : Retour au document.

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
import wittgenstein as lw

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = lw.RIPPER(random_state=0)

from sklearn.ensemble import VotingClassifier

model_voting = VotingClassifier([('SGD',model_1),('Tree',model_2),('Ripper',model_3)],voting='hard')
model_voting.fit(X_train,y_train)
print(model_voting.score(X_test,y_test))
```

0.8487255875537901

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_voting = cross_val_score(model_voting, X, y, cv=10)
print(res_cv_voting)
print(mean(res_cv_voting))
```

[0.83917935 0.84976837 0.8397351 0.8192053 0.82781457 0.8397351
0.84370861 0.82649007 0.83178808 0.83509934]
0.8352523875684276

```
y_pred_voting = model_voting.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test, y_pred_voting))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_voting))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test, y_pred_voting))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test, y_pred_voting))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test, y_pred_voting))
```

Accuraciy du modèle : 0.8487255875537901
Matrice de confusion du modèle :
[[1797 252]
 [205 767]]
Précision du modèle : 0.7526987242394504
Rappel du modèle : 0.7890946502057613
F1 Score du modèle : 0.7704671019588147

Code concernant le modèle AdaBoost Classifier pour la prédiction de la variable DEF AUT :Retour au document.

```
from sklearn.model_selection import GridSearchCV
param_grid_adaboost = {'learning_rate':[0.01,0.1],'n_estimators': [100, 200,500],'algorithm':['SAMME','SAMME.R']}

from sklearn.ensemble import AdaBoostClassifier
grid_adaboost = GridSearchCV(AdaBoostClassifier(random_state=100),param_grid_adaboost,cv=10)
grid_adaboost.fit(X_train,y_train)

print(grid_adaboost.best_params_)
model_adaboost = grid_adaboost.best_estimator_
print(model_adaboost.score(X_test,y_test))

{'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 500}
0.8480635551142006
```

Evaluation du modèle par cross-validation :

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_adaboost = cross_val_score(model_adaboost, X, y, cv=10)
print(res_cv_adaboost)
print(mean(res_cv_adaboost))

[0.84116479 0.8563865 0.84370861 0.84569536 0.84768212 0.84701987
 0.8410596 0.83774834 0.84370861 0.83576159]
0.8439935396496333
```

Différentes métriques :

```
from sklearn.metrics import accuracy_score
y_pred_adaboost = model_adaboost.predict(X_test)
print("Accuracy du modèle :",accuracy_score(y_test, y_pred_adaboost))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test,y_pred_adaboost))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test, y_pred_adaboost))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test, y_pred_adaboost))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test, y_pred_adaboost))
```

```
Accuracy du modèle : 0.8480635551142006
Matrice de confusion du modèle :
[[1946 103]
 [ 356 616]]
Précision du modèle : 0.8567454798331016
Rappel du modèle : 0.6337448559670782
F1 Score du modèle : 0.7285629804849202
```

Code concernant la construction des modèles servant à prédire la variable 'Collet' : Retour au document.

Arbre de décision :

```
from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(max_depth=6, random_state=100)
model_tree.fit(X_train_collet, y_train_collet)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=100)
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_tree = cross_val_score(model_tree, Xbis, y_collet, cv=10)
print(res_cv_tree)
print(mean(res_cv_tree))
```

```
[0.65226337 0.65020576 0.68518519 0.65843621 0.66185567 0.67216495
 0.67216495 0.68865979 0.68659794 0.67835052]
0.6705884349412414
```

```
from sklearn.metrics import accuracy_score
y_pred_tree = model_tree.predict(X_test_collet)
print("Accuracy du modèle :",accuracy_score(y_test_collet, y_pred_tree))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_collet,y_pred_tree))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_collet, y_pred_tree))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_collet, y_pred_tree))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_collet, y_pred_tree))
```

```
Accuracy du modèle : 0.666323377960865
Matrice de confusion du modèle :
[[399 162]
 [162 248]]
Précision du modèle : 0.6048780487804878
Rappel du modèle : 0.6048780487804878
F1 Score du modèle : 0.6048780487804878
```

Forêt aléatoire :

```
from sklearn.model_selection import GridSearchCV
param_grid_random_forest = {'n_estimators':[100, 200,500], 'criterion':['ginni', 'entropy'], 'max_depth':[2,3]}

from sklearn.ensemble import RandomForestClassifier
grid_random_forest = GridSearchCV(RandomForestClassifier(random_state=100),param_grid_random_forest,cv=10)
grid_random_forest.fit(X_train_collet, y_train_collet)

print(grid_random_forest.best_params_)

model_random_forest = grid_random_forest.best_estimator_
print(model_random_forest.score(X_test_collet, y_test_collet))

{'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 200}
0.6374871266735325
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_random_forest = cross_val_score(model_random_forest, Xbis, y_collet, cv=10)
print(res_cv_random_forest)
print(mean(res_cv_random_forest))

[0.62962963 0.66255144 0.63168724 0.64197531 0.66597938 0.66804124
 0.65360825 0.64948454 0.67835052 0.64536082]
0.6526668363667218
```

```
y_pred_random_forest = model_random_forest.predict(X_test_collet)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_collet, y_pred_random_forest))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_collet,y_pred_random_forest))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_collet, y_pred_random_forest))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_collet, y_pred_random_forest))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_collet, y_pred_random_forest))

Accuracy du modèle : 0.6374871266735325
Matrice de confusion du modèle :
[[517  44]
 [308 102]]
Précision du modèle : 0.6986301369863014
Rappel du modèle : 0.24878048780487805
F1 Score du modèle : 0.36690647482014394
```

Ripper :

```
import wittgenstein as lw

model_ripper = lw.RIPPER(random_state=100)
model_ripper.fit(X_train_collet, y_train_collet)
print(model_ripper.score(X_test_collet, y_test_collet))
```

0.6127703398558187

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_ripper = cross_val_score(model_ripper, Xbis, y_collet, cv=10)
print(res_cv_ripper)
print(mean(res_cv_ripper))
```

[0.65020576 0.62757202 0.60699588 0.61728395 0.62474227 0.64536082
0.62061856 0.60618557 0.64329897 0.64742268]
0.6289686479148106

```
y_pred_ripper = model_ripper.predict(X_test_collet)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy du modèle :", accuracy_score(y_test_collet, y_pred_ripper))
```

```
from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n", confusion_matrix(y_test_collet, y_pred_ripper))
```

```
from sklearn.metrics import precision_score
print("Précision du modèle :", precision_score(y_test_collet, y_pred_ripper))
```

```
from sklearn.metrics import recall_score
print("Rappel du modèle :", recall_score(y_test_collet, y_pred_ripper))
```

```
from sklearn.metrics import f1_score
print("F1 Score du modèle :", f1_score(y_test_collet, y_pred_ripper))
```

Accuracy du modèle : 0.6127703398558187

Matrice de confusion du modèle :

```
[[540 21]
 [355 55]]
```

Précision du modèle : 0.7236842105263158

Rappel du modèle : 0.13414634146341464

F1 Score du modèle : 0.2263374485596708

Bagging-Ripper :

```
from sklearn.ensemble import BaggingClassifier

model_bagging_ripper = BaggingClassifier(base_estimator = lw.RIPPER(max_rules=3, random_state=100), n_estimators = 10)
model_bagging_ripper.fit(X_train_collet,y_train_collet)

print(model_bagging_ripper.score(X_test_collet,y_test_collet))

0.6127703398558187
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_bagging_ripper = cross_val_score(model_bagging_ripper, Xbis, y_collet, cv=10)
print(res_cv_bagging_ripper)
print(mean(res_cv_bagging_ripper))
```

```
[0.6563786  0.63168724 0.61316872 0.64609053 0.65773196 0.62886598
 0.62886598 0.60824742 0.64742268 0.6371134 ]
0.6355572525561071
```

```
y_pred_bag_rip = model_bagging_ripper.predict(X_test_collet)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_collet, y_pred_bag_rip))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_collet,y_pred_bag_rip))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_collet, y_pred_bag_rip))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_collet, y_pred_bag_rip))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_collet, y_pred_bag_rip))
```

```
Accuracy du modèle : 0.6127703398558187
Matrice de confusion du modèle :
[[523  38]
 [338  72]]
Précision du modèle : 0.6545454545454545
Rappel du modèle : 0.17560975609756097
F1 Score du modèle : 0.27692307692307694
```

Stacking Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = KNeighborsClassifier(n_neighbors=2)

from sklearn.model_selection import GridSearchCV
param_grid_stacking = {'final_estimator':[SGDClassifier(random_state=100),DecisionTreeClassifier(random_state=100), KNeighborsClassifier()]}

from sklearn.ensemble import StackingClassifier
grid_stacking = GridSearchCV(StackingClassifier([('SGD',model_1),('Tree',model_2),('KNN',model_3)]),param_grid_stacking,cv=10)
grid_stacking.fit(X_train_collet,y_train_collet)

print(grid_stacking.best_params_)
model_stacking = grid_stacking.best_estimator_
print(model_stacking.score(X_test_collet,y_test_collet))

{'final_estimator': DecisionTreeClassifier(random_state=100)}
0.6426364572605562
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_stacking = cross_val_score(model_stacking, Xbis, y_collet, cv=10)
print(res_cv_stacking)
print(mean(res_cv_stacking))
```

```
[0.67901235 0.53703704 0.69958848 0.62757202 0.68453608 0.69896907
 0.68041237 0.6371134 0.59587629 0.59175258]
0.6431869670357643
```

```
y_pred_stacking = model_stacking.predict(X_test_collet)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_collet, y_pred_stacking))
```

```
from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_collet,y_pred_stacking))
```

```
from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_collet, y_pred_stacking))
```

```
from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_collet, y_pred_stacking))
```

```
from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_collet, y_pred_stacking))
```

```
Accuracy du modèle : 0.6426364572605562
Matrice de confusion du modèle :
[[283 278]
 [ 69 341]]
Précision du modèle : 0.5508885298869144
Rappel du modèle : 0.8317073170731707
F1 Score du modèle : 0.6627793974732751
```

Voting Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
import wittgenstein as lw

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = lw.RIPPER()

from sklearn.ensemble import VotingClassifier

model_voting = VotingClassifier([('SGD',model_1),('Tree',model_2),('Ripper',model_3)],voting='hard')
model_voting.fit(X_train_collet,y_train_collet)
print(model_voting.score(X_test_collet,y_test_collet))

0.6683831101956745
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_voting = cross_val_score(model_voting, Xbis, y_collet, cv=10)
print(res_cv_voting)
print(mean(res_cv_voting))

[0.63374486 0.6399177 0.67489712 0.67078189 0.63505155 0.6556701
 0.62268041 0.63505155 0.62268041 0.64123711]
0.6431712697806626
```

```
y_pred_voting = model_voting.predict(X_test_collet)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_collet, y_pred_voting))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_collet,y_pred_voting))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_collet, y_pred_voting))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_collet, y_pred_voting))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_collet, y_pred_voting))

Accuracy du modèle : 0.6683831101956745
Matrice de confusion du modèle :
[[392 169]
 [153 257]]
Précision du modèle : 0.6032863849765259
Rappel du modèle : 0.6268292682926829
F1 Score du modèle : 0.6148325358851675
```

AdaBoost Classifier :

```
from sklearn.model_selection import GridSearchCV
param_grid_adaboost = {'learning_rate':[0.01,0.1], 'n_estimators': [100, 200,500], 'algorithm':['SAMME', 'SAMME.R']}

from sklearn.ensemble import AdaBoostClassifier
grid_adaboost = GridSearchCV(AdaBoostClassifier(random_state=100),param_grid_adaboost,cv=10)
grid_adaboost.fit(X_train_collet,y_train_collet)

print(grid_adaboost.best_params_)
model_adaboost = grid_adaboost.best_estimator_
print(model_adaboost.score(X_test_collet,y_test_collet))

{'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 500}
0.6477857878475798
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_adaboost = cross_val_score(model_adaboost, Xbis, y_collet, cv=10)
print(res_cv_adaboost)
print(mean(res_cv_adaboost))
```

```
[0.68518519 0.66460905 0.68106996 0.68106996 0.67216495 0.69278351
 0.68247423 0.67216495 0.71752577 0.66185567]
0.6810903228543549
```

```
from sklearn.metrics import accuracy_score
y_pred_adaboost = model_adaboost.predict(X_test_collet)
print("Accuracy du modèle :",accuracy_score(y_test_collet, y_pred_adaboost))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_collet,y_pred_adaboost))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_collet, y_pred_adaboost))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_collet, y_pred_adaboost))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_collet, y_pred_adaboost))
```

```
Accuracy du modèle : 0.6477857878475798
Matrice de confusion du modèle :
[[449 112]
 [230 180]]
Précision du modèle : 0.6164383561643836
Rappel du modèle : 0.43902439024390244
F1 Score du modèle : 0.5128205128205129
```

Code concernant la construction des modèles servant à prédire la variable 'Tronc' : Retour au document.

Arbre de décision :

```
from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(max_depth=6, random_state=100)
model_tree.fit(X_train_tronc, y_train_tronc)
```

```
DecisionTreeClassifier(max_depth=6, random_state=100)
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_tree = cross_val_score(model_tree, Xbis, y_tronc, cv=10)
print(res_cv_tree)
print(mean(res_cv_tree))
```

```
[0.83950617 0.83127572 0.85596708 0.84156379 0.84123711 0.8371134
 0.84329897 0.8556701 0.85360825 0.83092784]
0.8430168427304738
```

```
from sklearn.metrics import accuracy_score
y_pred_tree = model_tree.predict(X_test_tronc)
print("Accuracy du modèle :",accuracy_score(y_test_tronc, y_pred_tree))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_tree))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_tree))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_tree))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_tree))
```

```
Accuracy du modèle : 0.8516992790937178
Matrice de confusion du modèle :
[[774 28]
 [116 53]]
Précision du modèle : 0.654320987654321
Rappel du modèle : 0.3136094674556213
F1 Score du modèle : 0.42399999999999993
```

Forêt aléatoire :

```
from sklearn.model_selection import GridSearchCV
param_grid_random_forest = {'n_estimators':[100, 200,500], 'criterion':['ginni','entropy'], 'max_depth':[2,3]}

from sklearn.ensemble import RandomForestClassifier
grid_random_forest = GridSearchCV(RandomForestClassifier(random_state=100),param_grid_random_forest,cv=10)
grid_random_forest.fit(X_train_tronc, y_train_tronc)

print(grid_random_forest.best_params_)

model_random_forest = grid_random_forest.best_estimator_
print(model_random_forest.score(X_test_tronc, y_test_tronc))

{'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 500}
0.835221421215242
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_random_forest = cross_val_score(model_random_forest, Xbis, y_tronc, cv=10)
print(res_cv_random_forest)
print(mean(res_cv_random_forest))

[0.83333333 0.82716049 0.82510288 0.83333333 0.84123711 0.84123711
 0.82474227 0.84948454 0.82886598 0.82474227]
0.8329239319502779
```

```
from sklearn.metrics import accuracy_score
y_pred_random_forest = model_random_forest.predict(X_test_tronc)
print("Accuracy du modèle :",accuracy_score(y_test_tronc, y_pred_random_forest))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_random_forest))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_random_forest))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_random_forest))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_random_forest))
```

```
Accuracy du modèle : 0.835221421215242
Matrice de confusion du modèle :
[[799  3]
 [157 12]]
Précision du modèle : 0.8
Rappel du modèle : 0.07100591715976332
F1 Score du modèle : 0.13043478260869565
```

Ripper :

```
import wittgenstein as lw

model_ripper = lw.RIPPER(random_state=100)
model_ripper.fit(X_train_tronc, y_train_tronc)
print(model_ripper.score(X_test_tronc, y_test_tronc))
```

0.8403707518022657

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_ripper = cross_val_score(model_ripper, Xbis, y_tronc, cv=10)
print(res_cv_ripper)
print(mean(res_cv_ripper))
```

[0.82510288 0.83744856 0.8436214 0.83950617 0.84123711 0.84329897
0.84948454 0.84329897 0.84948454 0.82680412]
0.839928725976836

```
y_pred_ripper = model_ripper.predict(X_test_tronc)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_tronc, y_pred_ripper))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_ripper))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_ripper))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_ripper))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_ripper))
```

Accuracy du modèle : 0.8403707518022657
Matrice de confusion du modèle :
[[790 12]
[143 26]]
Précision du modèle : 0.6842105263157895
Rappel du modèle : 0.15384615384615385
F1 Score du modèle : 0.25120772946859904

Bagging-Ripper :

```
from sklearn.ensemble import BaggingClassifier

model_bagging_ripper = BaggingClassifier(base_estimator = lw.RIPPER(max_rules=3, random_state=100), n_estimators = 10)
model_bagging_ripper.fit(X_train_tronc,y_train_tronc)

print(model_bagging_ripper.score(X_test_tronc,y_test_tronc))
```

0.8300720906282183

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_bagging_ripper = cross_val_score(model_bagging_ripper, Xbis, y_tronc, cv=10)
print(res_cv_bagging_ripper)
print(mean(res_cv_bagging_ripper))
```

[0.82510288 0.82921811 0.82716049 0.82921811 0.82474227 0.83505155
0.82886598 0.83917526 0.83298969 0.82061856]
0.8292142887446439

```
y_pred_bag_rip = model_bagging_ripper.predict(X_test_tronc)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test_tronc, y_pred_bag_rip))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_bag_rip))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_bag_rip))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_bag_rip))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_bag_rip))
```

Accuraciy du modèle : 0.8300720906282183
Matrice de confusion du modèle :
[[797 5]
 [160 9]]
Précision du modèle : 0.6428571428571429
Rappel du modèle : 0.05325443786982249
F1 Score du modèle : 0.09836065573770492

Stacking Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = KNeighborsClassifier(n_neighbors=2)

from sklearn.model_selection import GridSearchCV
param_grid_stacking = {'final_estimator':[SGDClassifier(random_state=100),DecisionTreeClassifier(random_state=100), KNeighborsClassifier()]}

from sklearn.ensemble import StackingClassifier
grid_stacking = GridSearchCV(StackingClassifier([('SGD',model_1),('Tree',model_2),('KNN',model_3)]),param_grid_stacking,cv=10)
grid_stacking.fit(X_train_tronc,y_train_tronc)

print(grid_stacking.best_params_)
model_stacking = grid_stacking.best_estimator_
print(model_stacking.score(X_test_tronc,y_test_tronc))

{'final_estimator': KNeighborsClassifier()}
0.8259526261585994
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_stacking = cross_val_score(model_stacking, Xbis, y_tronc, cv=10)
print(res_cv_stacking)
print(mean(res_cv_stacking))
```

```
[0.82098765 0.82098765 0.82098765 0.82098765 0.82268041 0.82268041
 0.82061856 0.82061856 0.82061856 0.82061856]
0.8211785668830343
```

```
y_pred_stacking = model_stacking.predict(X_test_tronc)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_tronc, y_pred_stacking))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_stacking))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_stacking))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_stacking))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_stacking))
```

```
Accuracy du modèle : 0.8259526261585994
Matrice de confusion du modèle :
[[802  0]
 [169  0]]
Précision du modèle : 0.0
Rappel du modèle : 0.0
F1 Score du modèle : 0.0
```

Voting Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
import wittgenstein as lw

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = lw.RIPPER()

from sklearn.ensemble import VotingClassifier

model_voting = VotingClassifier([('SGD',model_1),('Tree',model_2),('Ripper',model_3)],voting='hard')
model_voting.fit(X_train_tronc,y_train_tronc)
print(model_voting.score(X_test_tronc,y_test_tronc))
```

0.8403707518022657

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_voting = cross_val_score(model_voting, Xbis, y_tronc, cv=10)
print(res_cv_voting)
print(mean(res_cv_voting))
```

[0.80864198 0.84567901 0.84156379 0.84773663 0.84536082 0.8371134
0.84742268 0.8556701 0.84742268 0.83917526]
0.8415786347630563

```
y_pred_voting = model_voting.predict(X_test_tronc)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test_tronc, y_pred_voting))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_voting))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_voting))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_voting))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_voting))
```

Accuraciy du modèle : 0.8403707518022657
Matrice de confusion du modèle :
[[793 9]
 [146 23]]
Précision du modèle : 0.71875
Rappel du modèle : 0.13609467455621302
F1 Score du modèle : 0.22885572139303484

AdaBoost Classifier :

```
from sklearn.model_selection import GridSearchCV
param_grid_adaboost = {'learning_rate':[0.01,0.1], 'n_estimators': [100, 200,500], 'algorithm':['SAMME', 'SAMME.R']}

from sklearn.ensemble import AdaBoostClassifier
grid_adaboost = GridSearchCV(AdaBoostClassifier(random_state=100),param_grid_adaboost,cv=10)
grid_adaboost.fit(X_train_tronc,y_train_tronc)

print(grid_adaboost.best_params_)
model_adaboost = grid_adaboost.best_estimator_
print(model_adaboost.score(X_test_tronc,y_test_tronc))

{'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 500}
0.8362512873326468
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_adaboost = cross_val_score(model_adaboost, Xbis, y_tronc, cv=10)
print(res_cv_adaboost)
print(mean(res_cv_adaboost))
```

```
[0.83539095 0.83744856 0.83333333 0.82921811 0.8371134 0.84123711
 0.83298969 0.8556701 0.84123711 0.83505155]
0.8378689915574222
```

```
from sklearn.metrics import accuracy_score
y_pred_adaboost = model_adaboost.predict(X_test_tronc)
print("Accuracy du modèle :",accuracy_score(y_test_tronc, y_pred_adaboost))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_tronc,y_pred_adaboost))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_tronc, y_pred_adaboost))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_tronc, y_pred_adaboost))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_tronc, y_pred_adaboost))
```

```
Accuracy du modèle : 0.8362512873326468
Matrice de confusion du modèle :
[[787 15]
 [144 25]]
Précision du modèle : 0.625
Rappel du modèle : 0.14792899408284024
F1 Score du modèle : 0.23923444976076555
```

Code concernant la construction des modèles servant à prédire la variable 'Houppier' : Retour au document.

Arbre de décision :

```
from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(max_depth=6, random_state=100)
model_tree.fit(X_train_houppier, y_train_houppier)
```

```
DecisionTreeClassifier(max_depth=6, random_state=100)
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_tree = cross_val_score(model_tree, Xbis, y_houppier, cv=10)
print(res_cv_tree)
print(mean(res_cv_tree))
```

```
[0.718107  0.69753086 0.72016461 0.74074074 0.71752577 0.75463918
 0.73195876 0.76494845 0.70515464 0.73402062]
0.7284790632556956
```

```
y_pred_tree = model_tree.predict(X_test_houppier)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test_houppier, y_pred_tree))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_tree))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_tree))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_tree))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_tree))
```

```
Accuraciy du modèle : 0.7188465499485067
Matrice de confusion du modèle :
[[142 188]
 [ 85 556]]
Précision du modèle : 0.7473118279569892
Rappel du modèle : 0.8673946957878315
F1 Score du modèle : 0.8028880866425991
```

Forêt aléatoire :

```
from sklearn.model_selection import GridSearchCV
param_grid_random_forest = {'n_estimators':[100, 200,500], 'criterion':['gini','entropy'], 'max_depth':[2,3]}

from sklearn.ensemble import RandomForestClassifier
grid_random_forest = GridSearchCV(RandomForestClassifier(random_state=100),param_grid_random_forest,cv=10)
grid_random_forest.fit(X_train_houppier, y_train_houppier)

print(grid_random_forest.best_params_)

model_random_forest = grid_random_forest.best_estimator_
print(model_random_forest.score(X_test_houppier, y_test_houppier))

{'criterion': 'entropy', 'max_depth': 3, 'n_estimators': 100}
0.6930998970133883
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_random_forest = cross_val_score(model_random_forest, Xbis, y_houppier, cv=10)
print(res_cv_random_forest)
print(mean(res_cv_random_forest))

[0.71193416 0.70164609 0.69547325 0.72222222 0.70721649 0.71958763
 0.70721649 0.70927835 0.70927835 0.69896907]
0.7082822111917186
```

```
y_pred_random_forest = model_random_forest.predict(X_test_houppier)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_houppier, y_pred_random_forest))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_random_forest))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_random_forest))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_random_forest))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_random_forest))

Accuracy du modèle : 0.6930998970133883
Matrice de confusion du modèle :
[[ 62 268]
 [ 30 611]]
Précision du modèle : 0.6951080773606371
Rappel du modèle : 0.953198127925117
F1 Score du modèle : 0.8039473684210526
```

Ripper :

```
import wittgenstein as lw

model_ripper = lw.RIPPER(random_state=100)
model_ripper.fit(X_train_houppier, y_train_houppier)
print(model_ripper.score(X_test_houppier, y_test_houppier))
```

0.5324407826982492

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_ripper = cross_val_score(model_ripper, Xbis, y_houppier, cv=10)
print(res_cv_ripper)
print(mean(res_cv_ripper))
```

[0.5781893 0.48971193 0.56378601 0.54526749 0.56701031 0.54845361
0.55257732 0.59175258 0.57319588 0.54226804]
0.5552212464469051

```
y_pred_ripper = model_ripper.predict(X_test_houppier)

from sklearn.metrics import accuracy_score
print("Accuraciy du modèle :",accuracy_score(y_test_houppier, y_pred_ripper))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_ripper))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_ripper))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_ripper))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_ripper))
```

Accuraciy du modèle : 0.5324407826982492
Matrice de confusion du modèle :
[[318 12]
[442 199]]
Précision du modèle : 0.943127962085308
Rappel du modèle : 0.31045241809672386
F1 Score du modèle : 0.46713615023474175

Bagging-Ripper :

```
from sklearn.ensemble import BaggingClassifier

model_bagging_ripper = BaggingClassifier(base_estimator = lw.RIPPER(max_rules=3, random_state=100), n_estimators = 10)
model_bagging_ripper.fit(X_train_houppier,y_train_houppier)

print(model_bagging_ripper.score(X_test_houppier,y_test_houppier))
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_bagging_ripper = cross_val_score(model_bagging_ripper, Xbis, y_houppier, cv=10)
print(res_cv_bagging_ripper)
print(mean(res_cv_bagging_ripper))
```

```
[0.66460905 0.66460905 0.66460905 0.66460905 0.66597938 0.66597938
 0.66597938 0.66597938 0.66597938 0.66597938]
0.6654312502651564
```

```
y_pred_bag_rip = model_bagging_ripper.predict(X_test_collet)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_houppier, y_pred_bag_rip))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_bag_rip))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_bag_rip))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_bag_rip))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_bag_rip))
```

```
Accuracy du modèle : 0.6601441812564367
Matrice de confusion du modèle :
[[ 0 330]
 [ 0 641]]
Précision du modèle : 0.6601441812564367
Rappel du modèle : 1.0
F1 Score du modèle : 0.7952853598014888
```

Stacking Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = KNeighborsClassifier(n_neighbors=2)

from sklearn.model_selection import GridSearchCV
param_grid_stacking = {'final_estimator':[SGDClassifier(random_state=100),DecisionTreeClassifier(random_state=100), KNeighborsClassifier()]}

from sklearn.ensemble import StackingClassifier
grid_stacking = GridSearchCV(StackingClassifier([('SGD',model_1),('Tree',model_2),('KNN',model_3)]),param_grid_stacking,cv=10)
grid_stacking.fit(X_train_houppier,y_train_houppier)

print(grid_stacking.best_params_)
model_stacking = grid_stacking.best_estimator_
print(model_stacking.score(X_test_houppier,y_test_houppier))

{'final_estimator': DecisionTreeClassifier(random_state=100)}
0.7425334706488157
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_stacking = cross_val_score(model_stacking, Xbis, y_houppier, cv=10)
print(res_cv_stacking)
print(mean(res_cv_stacking))
```

```
[0.7345679 0.66872428 0.70576132 0.66460905 0.66597938 0.3814433
 0.73195876 0.76907216 0.66597938 0.70103093]
0.6689126468966102
```

```
y_pred_stacking = model_stacking.predict(X_test_houppier)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_houppier, y_pred_stacking))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_stacking))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_stacking))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_stacking))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_stacking))
```

```
Accuracy du modèle : 0.7425334706488157
Matrice de confusion du modèle :
[[131 199]
 [ 51 590]]
Précision du modèle : 0.7477820025348543
Rappel du modèle : 0.9204368174726989
F1 Score du modèle : 0.8251748251748251
```

Voting Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
import wittgenstein as lw

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = lw.RIPPER()

from sklearn.ensemble import VotingClassifier

model_voting = VotingClassifier([('SGD',model_1),('Tree',model_2),('Ripper',model_3)],voting='hard')
model_voting.fit(X_train_houppier,y_train_houppier)
print(model_voting.score(X_test_houppier,y_test_houppier))

0.592173017507724
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_voting = cross_val_score(model_voting, Xbis, y_houppier, cv=10)
print(res_cv_voting)
print(mean(res_cv_voting))

[0.50617284 0.74691358 0.77572016 0.55349794 0.76082474 0.73814433
 0.76494845 0.77525773 0.7443299 0.74845361]
0.711426328963557
```

```
y_pred_voting = model_voting.predict(X_test_houppier)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_houppier, y_pred_voting))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_voting))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_voting))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_voting))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_voting))

Accuracy du modèle : 0.592173017507724
Matrice de confusion du modèle :
[[295 35]
 [361 280]]
Précision du modèle : 0.8888888888888888
Rappel du modèle : 0.43681747269890797
F1 Score du modèle : 0.5857740585774058
```

AdaBoost Classifier :

```
from sklearn.model_selection import GridSearchCV
param_grid_adaboost = {'learning_rate':[0.01,0.1], 'n_estimators': [100, 200,500], 'algorithm':['SAMME', 'SAMME.R']}

from sklearn.ensemble import AdaBoostClassifier
grid_adaboost = GridSearchCV(AdaBoostClassifier(random_state=100),param_grid_adaboost,cv=10)
grid_adaboost.fit(X_train_houppier,y_train_houppier)

print(grid_adaboost.best_params_)
model_adaboost = grid_adaboost.best_estimator_
print(model_adaboost.score(X_test_houppier,y_test_houppier))

{'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 500}
0.7404737384140062
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_adaboost = cross_val_score(model_adaboost, Xbis, y_houppier, cv=10)
print(res_cv_adaboost)
print(mean(res_cv_adaboost))
```

```
[0.70576132 0.75308642 0.71604938 0.74485597 0.7257732 0.77319588
 0.75670103 0.75051546 0.74226804 0.73814433]
0.7406351024564083
```

```
from sklearn.metrics import accuracy_score
y_pred_adaboost = model_adaboost.predict(X_test_houppier)
print("Accuracy du modèle :",accuracy_score(y_test_houppier, y_pred_adaboost))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_houppier,y_pred_adaboost))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_houppier, y_pred_adaboost))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_houppier, y_pred_adaboost))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_houppier, y_pred_adaboost))
```

```
Accuracy du modèle : 0.7404737384140062
Matrice de confusion du modèle :
[[145 185]
 [ 67 574]]
Précision du modèle : 0.7562582345191041
Rappel du modèle : 0.8954758190327613
F1 Score du modèle : 0.8200000000000001
```

Code concernant la construction des modèles servant à prédire la variable 'Racine' : Retour au document.

Arbre de décision :

```
from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(max_depth=6, random_state=100)
model_tree.fit(X_train_racine, y_train_racine)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, random_state=100)
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_tree = cross_val_score(model_tree, Xbis, y_racine, cv=10)
print(res_cv_tree)
print(mean(res_cv_tree))
```

```
[0.85185185 0.84773663 0.86213992 0.8436214 0.85154639 0.85360825
 0.85979381 0.87010309 0.85979381 0.8556701 ]
0.8555865258156209
```

```
from sklearn.metrics import accuracy_score
y_pred_tree = model_tree.predict(X_test_racine)
print("Accuracy du modèle :",accuracy_score(y_test_racine, y_pred_tree))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_racine,y_pred_tree))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_racine, y_pred_tree))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_racine, y_pred_tree))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_racine, y_pred_tree))
```

```
Accuracy du modèle : 0.8774459320288363
Matrice de confusion du modèle :
[[842  6]
 [113 10]]
Précision du modèle : 0.625
Rappel du modèle : 0.08130081300813008
F1 Score du modèle : 0.14388489208633093
```

Forêt aléatoire :

```
from sklearn.model_selection import GridSearchCV
param_grid_random_forest = {'n_estimators':[100, 200,500], 'criterion':['ginni', 'entropy'], 'max_depth':[2,3]}

from sklearn.ensemble import RandomForestClassifier
grid_random_forest = GridSearchCV(RandomForestClassifier(random_state=100),param_grid_random_forest,cv=10)
grid_random_forest.fit(X_train_racine, y_train_racine)

print(grid_random_forest.best_params_)

model_random_forest = grid_random_forest.best_estimator_
print(model_random_forest.score(X_test_racine, y_test_racine))

{'criterion': 'entropy', 'max_depth': 2, 'n_estimators': 100}
0.8733264675592173
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_random_forest = cross_val_score(model_random_forest, Xbis, y_racine, cv=10)
print(res_cv_random_forest)
print(mean(res_cv_random_forest))

[0.85596708 0.85596708 0.85596708 0.85596708 0.8556701 0.8556701
 0.8556701 0.8556701 0.8556701 0.85773196]
0.8559950786984005
```

```
from sklearn.metrics import accuracy_score
y_pred_random_forest = model_random_forest.predict(X_test_racine)
print("Accuracy du modèle :",accuracy_score(y_test_racine, y_pred_random_forest))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_racine,y_pred_random_forest))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_racine, y_pred_random_forest))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_racine, y_pred_random_forest))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_racine, y_pred_random_forest))

Accuracy du modèle : 0.8733264675592173
Matrice de confusion du modèle :
[[848  0]
 [123  0]]
Précision du modèle : 0.0
Rappel du modèle : 0.0
F1 Score du modèle : 0.0
```

Ripper :

```
import wittgenstein as lw

model_ripper = lw.RIPPER(random_state=100)
model_ripper.fit(X_train_racine, y_train_racine)
print(model_ripper.score(X_test_racine, y_test_racine))
```

0.8640576725025747

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_ripper = cross_val_score(model_ripper, Xbis, y_racine, cv=10)
print(res_cv_ripper)
print(mean(res_cv_ripper))
```

[0.8600823 0.86213992 0.8600823 0.86831276 0.85773196 0.84742268
0.85154639 0.86391753 0.8556701 0.85773196]
0.8584637902507318

```
y_pred_ripper = model_ripper.predict(X_test_racine)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :", accuracy_score(y_test_racine, y_pred_ripper))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n", confusion_matrix(y_test_racine, y_pred_ripper))

from sklearn.metrics import precision_score
print("Précision du modèle :", precision_score(y_test_racine, y_pred_ripper))

from sklearn.metrics import recall_score
print("Rappel du modèle :", recall_score(y_test_racine, y_pred_ripper))

from sklearn.metrics import f1_score
print("F1 Score du modèle :", f1_score(y_test_racine, y_pred_ripper))
```

Accuracy du modèle : 0.8640576725025747
Matrice de confusion du modèle :
[[833 15]
[117 6]]
Précision du modèle : 0.2857142857142857
Rappel du modèle : 0.04878048780487805
F1 Score du modèle : 0.08333333333333334

Bagging-Ripper :

```
from sklearn.ensemble import BaggingClassifier

model_bagging_ripper = BaggingClassifier(base_estimator = lw.RIPPER(max_rules=3, random_state=100), n_estimators = 10)
model_bagging_ripper.fit(X_train_racine,y_train_racine)

print(model_bagging_ripper.score(X_test_racine,y_test_racine))

0.8722966014418125
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_bagging_ripper = cross_val_score(model_bagging_ripper, Xbis, y_racine, cv=10)
print(res_cv_bagging_ripper)
print(mean(res_cv_bagging_ripper))
```

```
[0.85596708 0.85802469 0.8600823  0.85802469 0.85360825 0.8556701
 0.85773196 0.85773196 0.85979381 0.85360825]
0.8570243095329005
```

```
y_pred_bag_rip = model_bagging_ripper.predict(X_test_racine)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_racine, y_pred_bag_rip))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_racine,y_pred_bag_rip))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_racine, y_pred_bag_rip))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_racine, y_pred_bag_rip))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_racine, y_pred_bag_rip))
```

```
Accuracy du modèle : 0.8722966014418125
Matrice de confusion du modèle :
[[847  1]
 [123  0]]
Précision du modèle : 0.0
Rappel du modèle : 0.0
F1 Score du modèle : 0.0
```

Stacking Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = KNeighborsClassifier(n_neighbors=2)

from sklearn.model_selection import GridSearchCV
param_grid_stacking = {'final_estimator': [SGDClassifier(random_state=100), DecisionTreeClassifier(random_state=100), KNeighborsClassifier()]}

from sklearn.ensemble import StackingClassifier
grid_stacking = GridSearchCV(StackingClassifier([('SGD', model_1), ('Tree', model_2), ('KNN', model_3)]), param_grid_stacking, cv=10)
grid_stacking.fit(X_train_racine, y_train_racine)

print(grid_stacking.best_params_)
model_stacking = grid_stacking.best_estimator_
print(model_stacking.score(X_test_racine, y_test_racine))

{'final_estimator': KNeighborsClassifier()}
0.8733264675592173
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_stacking = cross_val_score(model_stacking, Xbis, y_racine, cv=10)
print(res_cv_stacking)
print(mean(res_cv_stacking))
```

```
[0.85596708 0.85596708 0.85596708 0.85596708 0.8556701 0.8556701
 0.8556701 0.8556701 0.8556701 0.85773196]
0.8559950786984005
```

```
y_pred_stacking = model_stacking.predict(X_test_racine)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy du modèle :", accuracy_score(y_test_racine, y_pred_stacking))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n", confusion_matrix(y_test_racine, y_pred_stacking))

from sklearn.metrics import precision_score
print("Précision du modèle :", precision_score(y_test_racine, y_pred_stacking))

from sklearn.metrics import recall_score
print("Rappel du modèle :", recall_score(y_test_racine, y_pred_stacking))

from sklearn.metrics import f1_score
print("F1 Score du modèle :", f1_score(y_test_racine, y_pred_stacking))
```

```
Accuracy du modèle : 0.8733264675592173
```

```
Matrice de confusion du modèle :
```

```
[[848  0]
 [123  0]]
```

```
Précision du modèle : 0.0
```

```
Rappel du modèle : 0.0
```

```
F1 Score du modèle : 0.0
```

Voting Classifier :

```
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
import wittgenstein as lw

model_1 = SGDClassifier(random_state=0)
model_2 = DecisionTreeClassifier(random_state=0)
model_3 = lw.RIPPER()

from sklearn.ensemble import VotingClassifier

model_voting = VotingClassifier([('SGD',model_1),('Tree',model_2),('Ripper',model_3)],voting='hard')
model_voting.fit(X_train_racine,y_train_racine)
print(model_voting.score(X_test_racine,y_test_racine))

0.831101956745623
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_voting = cross_val_score(model_voting, Xbis, y_racine, cv=10)
print(res_cv_voting)
print(mean(res_cv_voting))

[0.87242798 0.85802469 0.85390947 0.85802469 0.86185567 0.85773196
 0.85979381 0.85979381 0.87010309 0.82886598]
0.8580531161172628
```

```
y_pred_voting = model_voting.predict(X_test_racine)

from sklearn.metrics import accuracy_score
print("Accuracy du modèle :",accuracy_score(y_test_racine, y_pred_voting))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_racine,y_pred_voting))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_racine, y_pred_voting))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_racine, y_pred_voting))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_racine, y_pred_voting))
```

```
Accuracy du modèle : 0.831101956745623
Matrice de confusion du modèle :
[[743 105]
 [ 59  64]]
Précision du modèle : 0.378698224852071
Rappel du modèle : 0.5203252032520326
F1 Score du modèle : 0.4383561643835616
```

AdaBoost Classifier :

```
from sklearn.model_selection import GridSearchCV
param_grid_adaboost = {'learning_rate':[0.01,0.1], 'n_estimators': [100, 200,500], 'algorithm':['SAMME', 'SAMME.R']}

from sklearn.ensemble import AdaBoostClassifier
grid_adaboost = GridSearchCV(AdaBoostClassifier(random_state=100),param_grid_adaboost,cv=10)
grid_adaboost.fit(X_train_racine,y_train_racine)

print(grid_adaboost.best_params_)
model_adaboost = grid_adaboost.best_estimator_
print(model_adaboost.score(X_test_racine,y_test_racine))

{'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 500}
0.8805355303810505
```

```
from sklearn.model_selection import cross_val_score
from statistics import mean
res_cv_adaboost = cross_val_score(model_adaboost, Xbis, y_racine, cv=10)
print(res_cv_adaboost)
print(mean(res_cv_adaboost))
```

```
[0.8600823  0.86213992 0.85802469 0.85802469 0.86185567 0.85979381
 0.8556701  0.86391753 0.85979381 0.85979381]
0.8599096347206313
```

```
from sklearn.metrics import accuracy_score
y_pred_adaboost = model_adaboost.predict(X_test_racine)
print("Accuracy du modèle :",accuracy_score(y_test_racine, y_pred_adaboost))

from sklearn.metrics import confusion_matrix
print("Matrice de confusion du modèle :\n",confusion_matrix(y_test_racine,y_pred_adaboost))

from sklearn.metrics import precision_score
print("Précision du modèle :",precision_score(y_test_racine, y_pred_adaboost))

from sklearn.metrics import recall_score
print("Rappel du modèle :",recall_score(y_test_racine, y_pred_adaboost))

from sklearn.metrics import f1_score
print("F1 Score du modèle :",f1_score(y_test_racine, y_pred_adaboost))
```

```
Accuracy du modèle : 0.8805355303810505
Matrice de confusion du modèle :
[[845  3]
 [113 10]]
Précision du modèle : 0.7692307692307693
Rappel du modèle : 0.08130081300813008
F1 Score du modèle : 0.14705882352941174
```

8 Bibliographie

Documentation Scikit-Learn. *Supervised learning*.

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Sabeur ARIDHI, *Fouille de Données et Extraction de Connaissance*. Université de Lorraine, 2022