Institut national des sciences appliquées de Toulouse

Génie Mathématiques et Modélisation
Spécialité Mathématiques Appliquées

Rapport de Projet

Machine Learning

Auteures du compte rendu :
Camusat Léa
Etheve Eva
Gonzalez Julie
Roig Lila

Groupe :
MA A/B

Année Universitaire :
2021/2022

20 juin 2022

TABLE DES MATIÈRES

I Introduction

Ce projet de Machine Learning a pour objectif de travailler sur un problème de classification appliqué à des observations météorologiques. Il s'agira de prédire pour un jour j la catégorie à laquelle appartiendra la quantité de pluie du jour suivant $j+1$ et ce, grâce aux variables explicatives. On vise à prédire les quantités de précipitations dans le cas de la régression, et la classe d'intensité des précipitations dans le cas de la classification. On vise aussi à trouver quelle variable permet de séparer au mieux nos données.

II Description des données

Il nous a été fourni un jeu de données de 688 observations météorologiques par *Météo France* utilisé dans le cadre du challenge *Défi IA 2022*. Cette base de données est accessible sur le Github de *Météo France* sous le nom de *rain.txt*. Les mesures ont été réalisées en 2016-2017 dans le Sud-Est de la France métropolitaine ainsi que la Corse.

Cette base de données comporte 16 variables explicatives.

7 variables concernent la mesure de paramètres du jour courant :

— *date* : date du jour courant
— *ff* : vitesse du vent en $m.s^{-1}$
— *t* : température en Kelvin K
— *td* : température du point de rosée en Kelvin K
— *hu* : taux d'humidité en %
— *dd* : direction du vent en degrés
— *precip* : taux de précipitation en $kg.m^{-2}$

Les 9 autres concernent la prédiction de ces paramètres du jour suivant selon le modèle *AROME* de *Météo France* :

— *ws_arome* : vitesse du vent en $m.s^{-1}$
— *p3031_arome* : direction du vent en degrés
— *u10_arome* : composante U du vent (vent zonal)
— *vu10_arome* : composante V du vent (vent méridien)
— *t2m_arome* : température à 2 m de hauteur en Kelvin K
— *d2m_arome* : température du point de rosée à 2m de hauteur en Kelvin K %
— *r_arome* : taux d'humidité en %
— *tp_arome* : quantité de précipitation en $kg.m^{-2}$
— *msl_arome* : pression au niveau de la mer en Pa

Dans ce projet, nous avons deux variables réponse de nature différente :

— *rain* : variable quantitative indiquant le taux de pluie du jour suivant $kg.m^{-2}$
— *rain_class* : variable qualitative composée de 3 classes :
  • *no_rain* : une observation est classée dans cette catégorie si la variable $rain = 0$
  • *low_rain* : une observation est classée dans cette catégorie si $0 < rain \leq 2$
  • *high_rain* : une observation est classée dans cette catégorie si $rain > 2$.

On notera qu'il n'y a pas de valeurs manquantes pour chacune des variables explicatives, c'est-à-dire que toutes les mesures ont bien pu être réalisées pour chaque variable.

III Analyse unidimensionnelle

L'analyse unidimensionnelle consiste à étudier séparément chaque variable du jeu de données. Pour cela, nous utilisons les boxplots (A.1) et histogrammes (A.2), disponibles en annexe. Cela nous donne une idée générale de la distribution des variables et nous permettra également de déterminer les transformations à effectuer pour les symétriser.

III.1 Analyse unidimensionnelle des variables d'entrée

1. Les variables relatives à la vitesse du vent :

- **ff** : La vitesse du vent ressemble à une distribution gaussienne, environ 175 individus ont une valeur de **ff** autour de la moyenne qui est de 4m.s^{-1}. L'étendue et l'écart interquartiles sont assez faibles. On observe quelques outliers dans les valeurs hautes. La force du vent est donc en général assez constante et basse sauf pour quelques dates où elle est plus importante.

- **ws_arome** : Globalement on peut faire les mêmes observations que sur **ff**, à la différence que **ws_arome** prend des valeurs un peu plus basses en terme de moyenne et de valeurs extrêmes (son maximum est par exemple de 10 $m.s^{-1}$ contre $12m.s^{-1}$ pour **ff**). On peut se poser se demander si le modèle AROME a tendance à sous-estimer la force du vent mais dans l'ensemble la prédiction semble correcte.

2. Les variables relatives aux précipitations :

- **precip** : En moyenne les précipitations sont très faibles, la médiane des observations est à 0.4 $kg.m^{-2}$. La distribution a une forme exponentielle. Elle est fortement impactée par des valeurs extrêmes, il y a un nombre important d'outliers et le maximum de précipitations est de 34.5 $kg.m^{-2}$

- **tp_arome** : En comparaison avec **precip**, l'étendue et l'écart interquartile sont plus resserrés. Il y a également beaucoup d'outliers et on observe des valeurs extrêmes (maximum à 1069 $kg.m^{-2}$) qui semblent aberrantes. Le modèle AROME semble avoir du mal à fournir des prédictions correctes en cas de fortes précipitations.

3. Les variables relatives à la température (t, t2m_arome) et au point de rosée (td et d2m_arome) le jour J et prédite pour le jour suivant :

Ces variables ont une distribution très similaire, qui ressemble à une gaussienne, sans valeurs extrêmes. Dans l'ensemble les données de température et point de rosée ne varient pas beaucoup.

4. Les variables relatives à la direction du vent :

- **dd** et **p3031_arome** : Ces deux variables ont une distribution très similaire qui ressemble à une gaussienne avec une moyenne (218 degrés) et une médiane (203 degrés) décalées vers les valeurs hautes. L'étendue est très importante et il n'y a pas d'outlier. La direction du vent change beaucoup d'un jour sur l'autre mais le modèle AROME semble en fournir une bonne prédiction.

- **u10_arome** et **vu10_arome** : La distribution ressemble à une gaussienne et l'étendue est faible. Les composantes U et V restent relativement constantes.

5. Les variables relatives à l'humidité :

- **hu** et **r_arome** : La distribution a une forme de gaussienne avec la moyenne (de 85%) décalée vers les fortes valeurs et une queue assez importante qui s'étend sur les faibles valeurs (minimum à 50%). L'étendue est importante, on a donc une forte variation de l'humidité au cours de l'année. La prédiction fournie par AROME semble très correcte.

6. La variable relative à la pression atmosphérique : msl_arome

- **msl_arome** : La distribution a une forme de gaussienne centrée. Il y a beaucoup d'outliers sur les valeurs basses, ces valeurs sont aberrantes au vu de la régularité du reste de la distribution.

III.2 Analyse unidimensionelle des variables de sortie

- **rain** : quantité totale de précipitation le jour suivant $kg.m^{-2}$: La distribution ressemble à une exponentielle décroissante. La médiane (de 0.4 $kg.m^{-2}$) et la moyenne (de 2.3 $kg.m^{-2}$) sont très basses et l'étendue est très faible. Mais on observe un très grand nombre d'outliers. On peut remettre en question l'efficacité du modèle AROME pour prédire la variable *rain* lorsque la quantité de pluie est importante.
- **rain_class** : Chaque modalité représente environ un tiers des observations. Au cours de l'année on a donc à peu près à parts égales des pluies fortes, moyennes et nulles.

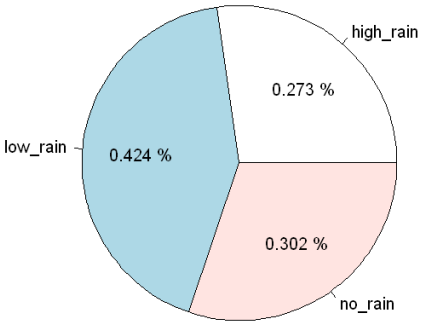


FIGURE 1 – Pie chart de la variable rain_class

Par la suite, nous appliquons des transformations sur nos données pour rendre certaines distributions plus symétriques, ce qui est nécessaire pour appliquer certains algorithmes. Nous ajoutons alors le suffixe **"_mod"** à la fin des variables modifiées. Nous référons le lecteur au Notebook pour connaître les transformations effectuées.

IV Analyse bidimensionnelle

Nous poursuivons avec une analyse bidimensionnelle qui permet de comprendre les corrélations entre les variables.

1. Corrélation entre deux variables quantitatives :

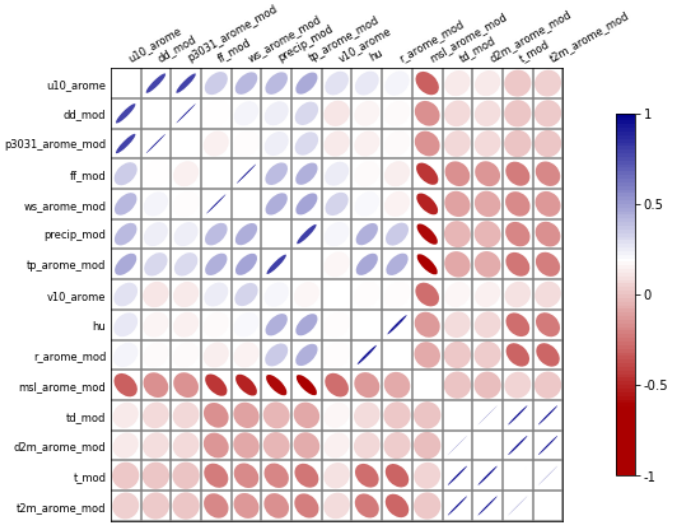


FIGURE 2 – Corrplot pour l'Analyse bidimensionnelle

On voit sur le corrplot présenté en figure 2 que les variables corrélées sont :
- La température aux jours j (t_mod) et j+1 (t2m_arome_mod), le point de rosée aux jours j (td_mod) et j+1 (d2m_arome_mod).
- La vitesse du vent aux jours j (ff_mod) et j+1 (ws_arome_mod).
- La direction du vent aux jours j (dd_mod) et j+1 (p3031_arome_mod), les composantes du vent d'ouest au jour j+1 (u10_arome).
- L'humidité aux jours j (hu) et j+1 (r_arome).
- Les précipitations au jour j (precip_mod) et la quantité totale de précipitations prévue pour j+1 (tp_arome).

2. Corrélation entre une variable quantitative et une variable qualitative

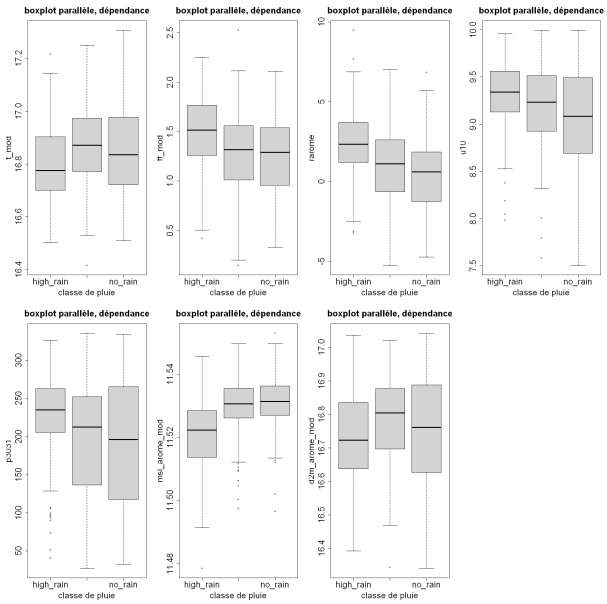


FIGURE 3 – Boxplot pour l'Analyse bidimensionnelle

Sur le boxplot (figure 3), on voit que les variables très liées à rain_class sont :
- msl_arome_mod : quand la pression est grande la pluie est forte, ce qui est normal d'un point de vue météorologique ; à l'inverse, quand la pression est plus faible il y a peu ou pas pluie,
- le vent d'ouest u10, ws_arome et ff : un vent fort est plus propice à un épisode pluvieux,
- la quantité de pluie prédite par le modèle tp_arome,
- la direction du vent dd.

Parmi les variables très peu liées et qui ont donc peu d'impact sur l'occurrence de pluie on retrouve : la température t_mod, le vent venant du sud v10 et le point de rosée td_mod.

3. Corrélation entre deux variables qualitatives

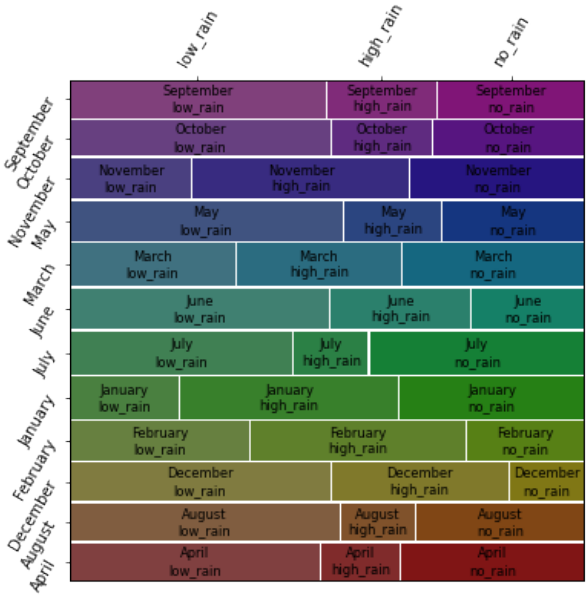


FIGURE 4 – Mosic plot pour l'Analyse bidimensionnelle

Le mosaic plot (figure 4) présente la corrélation entre les variables rain_class et Date. On voit que les séparations ne sont pas au même niveau ce qui signifie que les deux variables sont très liées : selon le mois, l'intensité de la pluie varie. En lisant le mosaic plot, on voit que l'on peut séparer les mois en deux groupes : septembre, octobre, mai, juin, juillet, août pour lesquels il y a peu de jours avec de fortes pluies et beaucoup de pluie faible et d'absence de pluie et novembre, décembre, mars, janvier, février correspondant à des mois avec beaucoup de fortes pluie en moyenne. Cela semble cohérent pour le climat en France.

V Analyse en Composantes Principales (ACP)

L'ACP est une méthode de **réduction de dimension** permettant de visualiser l'information en transformant des variables très corrélées en de nouvelles variables décorrélées les unes des autres appelées composantes principales. Pour ce faire, on trouve d'abord les axes principaux : le premier axe principal est celui qui **maximise la variance des données projetées**. Le deuxième axe principal, est l'axe orthogonal au premier qui maximise la variance des données projetées etc... Les composantes principales renferment les coordonnées des individus projetés sur chacun

des axes principaux. Il faut déterminer le nombre de composantes principales à conserver pour expliquer les données.

Nous effectuons une ACP centrée réduite sur les données météo en retirant la variable à expliquer rain_mod et traçons le graphe des pourcentages cumulés d'inertie en figure 5 (voir le Notebook pour davantage de graphes) :

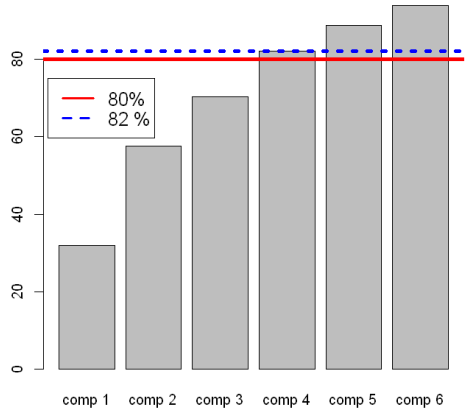


FIGURE 5 – Pourcentages cumulés des inerties

L'analyse de la figure 5 suggère de garder quatre composantes principales pour l'ACP afin de réduire la dimension sans perdre une quantité trop importante d'information. En effet, quatre dimensions permettent d'expliquer 82% de l'inertie des données.
Nous traçons à présent le graphe des variables donné en figure 6.

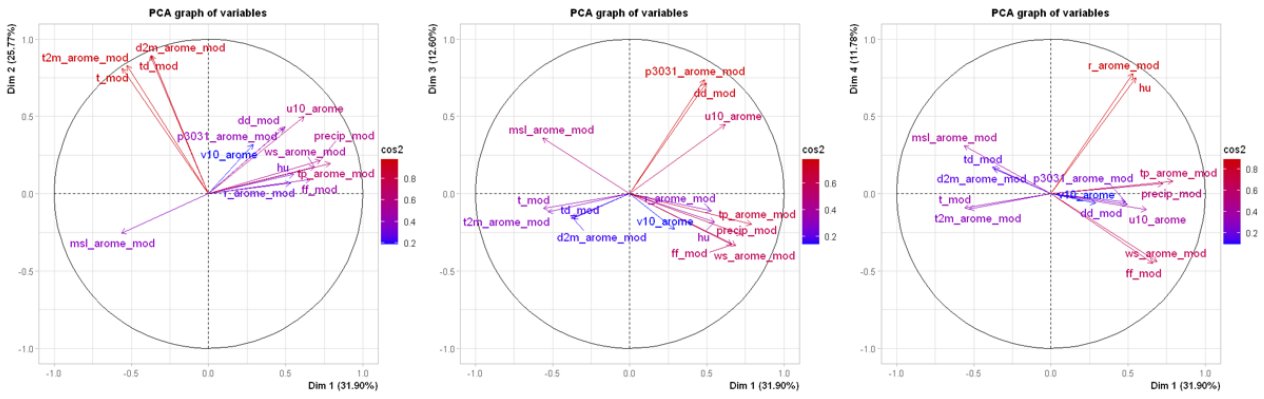


FIGURE 6 – Graphe des variables dim1-dim2, dim1-dim3, dim1-dim4

Pour analyser chaque graphe des corrélations de la figure 6, et expliquer les composantes, on ne peut considérer que les variables qui ont une flèche proches du cercle, c'est à dire qu'elles sont bien représentées dans le plan de l'ACP. Nous observons que les variables bien projetées sur la **dimension 1** sont les variables relatives à la vitesse du vent (ff_mod, ws_arome_mod) et celles relatives aux précipitations (precip_mod et tp_arome). Les variables bien projetées sur la **dimension 2** sont les variables relatives à la température et au point de rosée (t_mod, td_mod, t2m_arome_mod et d2m_arome_mod). Les variables bien projetées sur la **dimension 3** sont

les variables relatives à la direction du vent (dd_mod, p3031_arome_mod, u10_arome). Enfin les variables bien projetées sur la **dimension 4** sont les variables relatives à l'humidité (hu et r_arome).

Nous traçons à présent le graphe des individus (figure 7) où les trois modalités de la variable rain_class sont affichées.

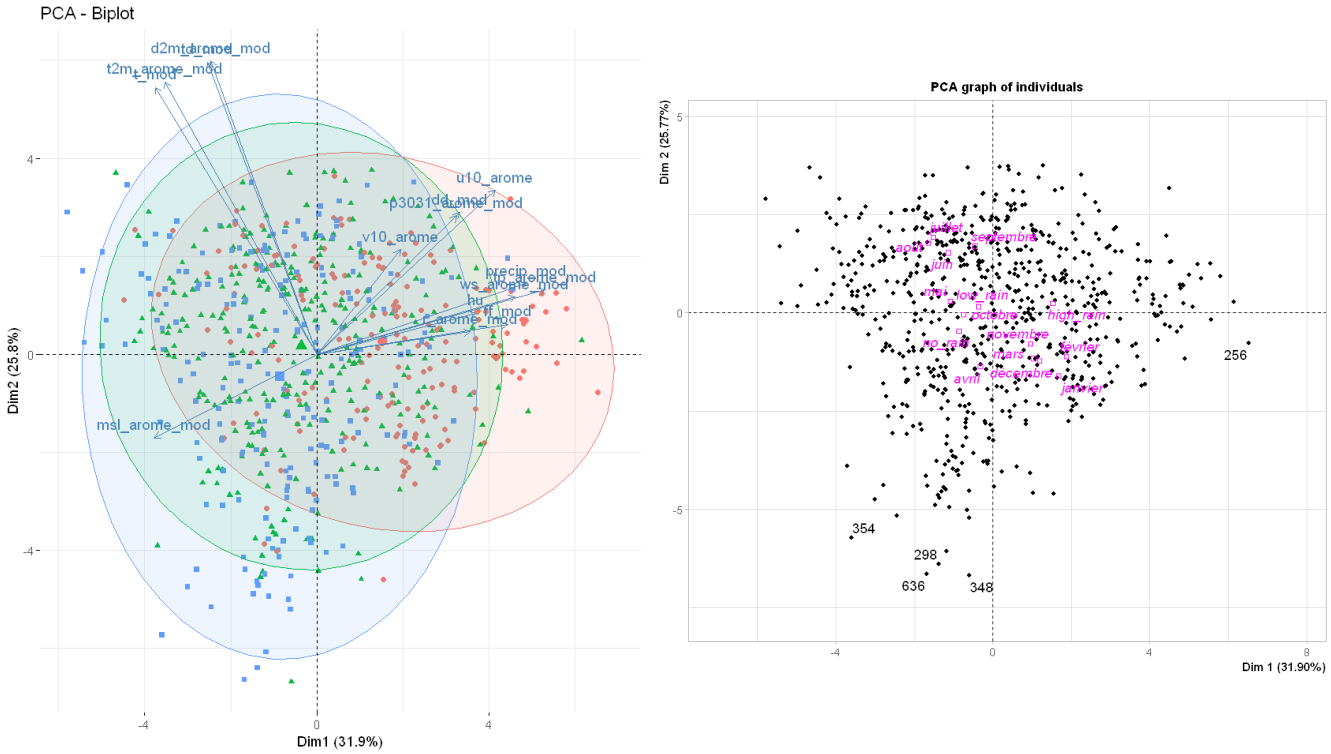


FIGURE 7 – Graphe des individus pour dim1-dim2

Avec la figure 7, il semblerait que les individus les plus à droite sur la **dimension 1** portent la modalité **high_rain**, tandis la **dimension 2** pourrait plutôt représenter les modalités **low_rain** et **no_rain** qui sont confondues. Ceci peut se vérifier avec l'analyse des outliers : les outliers 354, 298 et 636 ont de très faibles valeurs de précipitations (precip_mod et tp_arome_mod) et leurs valeurs sur ces variables correspondent au minimum. Ils présentent tous la modalité no_rain. Les outliers 348 et 256 présentent respectivement les modalités low_rain et high_rain.

Une fois que nous avons analysé les différentes variables et leurs corrélations dans les sections III et IV et que nous les avons représentées dans la section V, nous souhaitons appliquer différents algorithmes de Machine Learning pour prédire la quantité de pluie le jour suivant.

VI Protocole de Comparaison

Afin d'appliquer des algorithmes de Machine Learning, il est nécessaire de séparer aléatoirement les données en un échantillon d'apprentissage ($\approx$ 80% des données) et un échantillon de

test ($\approx 20\%$ des données). L'échantillon d'apprentissage sert à entraîner les algorithmes tandis que l'échantillon de test sert à évaluer leur performance.
Les variables explicatives du modèle $X = (X^1, .., X^p)$ sont les « paramètres météorologiques observés le jour j » et les « prévisions météos du jour suivant par le modèle AROME ». Les variables à expliquer Y sont : **rain_class** pour la classification ($Y \in \{1..K\}$) et **rain_mod** pour la régression ($Y \in \mathbb{R}$). Nous notons *datappr* et *datestr* pour l'échantillon d'entraînement et l'échantillon de test pour la régression et *datappq* et *datestq* pour la classification.

1. Métriques pour évaluer la régression

Pour comparer nos résultats en régression, nous définissons plusieurs erreurs que nous chercherons à minimiser. La variable à expliquer est **rain**.

- **L'erreur d'apprentissage** $MSE = \frac{1}{n}\sum_{1\leq i\leq n}(Y_{appr,i} - \hat{Y}_i)^2$ où $\hat{Y}_i$ sont les valeurs prédites et $Y_{appr,i}$ appartiennent à l'échantillon d'apprentissage
- **L'erreur de généralisation** $Err = \frac{1}{n}\sum_{1\leq i\leq n}(Y_{test,i} - \hat{Y}_i)^2$ où $\hat{Y}_i$ sont les valeurs prédites et $Y_{test,i}$ appartiennent à l'échantillon de test.
- **L'erreur de généralisation** définie par **Météo France** MAPE $= \frac{100}{n}\sum_{i=1}^{n}\frac{|Y_{test,i} - \hat{Y}_i|}{Y_{test,i}+1}$ où $\hat{Y}_i$ sont les valeurs prédites et $Y_{test,i}$ appartiennent à l'échantillon de test.
- **Le Score Python** qui correspond à R^2 pour la régression que nous souhaitons maximiser.

2. Métriques pour évaluer la classification

Nous distinguerons deux classifications. La première classification est la classification directe qui consiste à prédire la variable qualitative **rain_class**. La deuxième classification est obtenue en appliquant des seuils sur les résultats de la régression de manière à pouvoir passer du vecteur prédit en régression à un vecteur en classification. Ainsi, les seuils qui définissent les modalités sont rain $= 0$, $0 <$ rain ≤ 2 et rain > 2. Cependant, le problème est que si l'on définit la modalité no_rain par une valeur discrète, celle ci ne sera jamais observée dans la prédiction étant donné que la variable prédite est quantitative. Pour la transformation des résultats de regression, nous définissons ainsi la première modalité par rain ≤ 0, car certains modèles prédisent des valeurs négatives mais cela n'a pas de sens, nous considèrerons donc que cela correspond à l'absence de pluie. D'autre part, nous avions appliqué la transformation $\square^{1/3}$ à **rain** afin de la symétriser. Nous transformons aussi les seuils en classifiant comme no_rain les valeurs de rain en dessous de 0, comme high_rain les valeurs de rain qui sont en dessus de $2^{1/3}$ et comme low_rain les valeurs de rain entre 0 et $2^{1/3}$.

Pour pouvoir comparer les modèles de classification après seuillage de la régression et les modèles de classification, nous avons choisi d'utiliser le F1-score, le Mallow score et l'indice de Pureté.

- **L'indice de Pureté** $\mathcal{P}$ permet de mesurer le nombre de modalités correctement prédites dans la table de contingence : $\mathcal{P} = \frac{1}{N}\sum_{m\in M}\max_{d\in D}|m \cap d|$ où N est le nombre total d'observations, M correspond aux modalités prédites et D aux modalités observées. Plus $\mathcal{P}$ est proche de 1 et meilleure est la prédiction.

— On calcule le **F1-score** pour chaque classe le F1-score et en faisant la moyenne des trois F1-score : $F1 - score = \frac{TP}{TP+\frac{1}{2}(FN+FP)}$. Cette métrique est particulièrement efficace car elle permet de calculer à la fois le recall et la précision.

VII Prévision par modèle gaussien

Dans cette section, nous souhaitons expliquer la variable quantitative Y = **rain_mod**.

VII.1 Modèle linéaire

1. Sans sélection de variables

Un **modèle linéaire** suppose une relation linéaire entre les variables explicatives $X_i, \forall 1 \leq i \leq n$ et la variable à expliquer Y, c'est-à-dire de la forme : $Y = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n + \varepsilon$ où $\beta \in \mathbb{R}^p$ est un paramètre du modèle et $\varepsilon \sim \mathcal{N}_n(0_n, \sigma I_n)$. Nous commençons par vérifier si les résidus $\hat{\varepsilon}$ vérifient les hypothèses de la régression linéaire avec la figure 8.

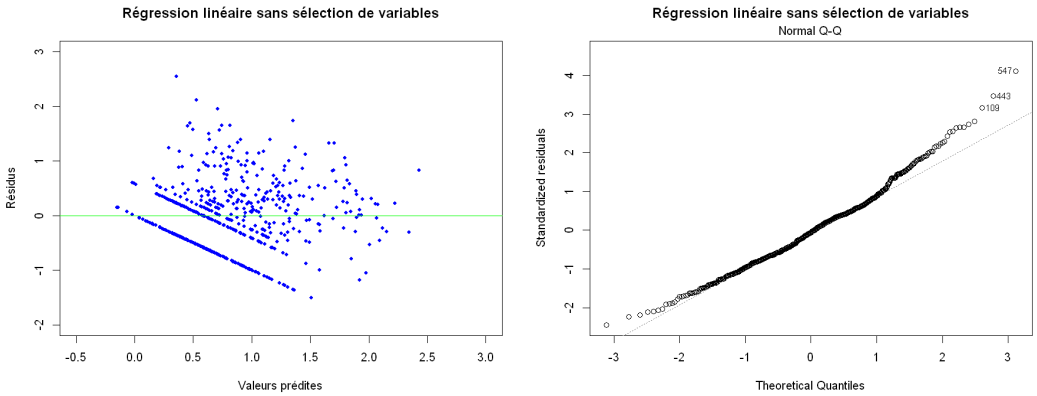


FIGURE 8 – Graphe des résidus de la régression linéaire

Sur le 1er graphe, on observe une tendance dans le nuage de points et les résidus ne sont pas dispersés de façon homogène. Ceci suggère que les ε_i ne sont pas indépendants et que leur variance dépend de Y (hétéroscédasticité). De plus, on a $\varepsilon = Y - \hat{Y}$ où $\hat{Y} = X\hat{\beta}$ sont les valeurs prédites. Ainsi, les droites observées traduisent une valeur constante de précipitations Y. Par exemple, la droite la plus basse $\hat{\epsilon} = -\hat{Y}$ passe par l'origine du repère donc $Y = 0$, c'est-à-dire qu'il n'y a pas de pluie. On observe en effet dans les données qu'il y a beaucoup de jours pour lesquels la pluie est nulle. Sur le 2ème graphe, on voit un décrochement rapide aux extrémités, les résidus ne sont donc pas gaussiens. En conclusion, les hypothèses de la régression linéaire ne sont pas vérifiées et nous ne pouvons donc pas l'appliquer pour expliquer nos données en théorie.

2. Sélection de variables par régularisation L1 (LASSO)

Dans un deuxième temps, nous faisons une régression **LASSO** qui consiste à pénaliser le modèle pour obtenir un estimateur $\hat{\beta}$ parcimonieux et rendre ainsi le modèle plus interprétable. Le problème considéré est : $\hat{\beta}^* = \underset{\beta \in \mathbb{R}^{p+1}}{argmin} ||Y - X\beta|| + \lambda ||\beta||_1$ où λ est le paramètre à optimiser qui permet de doser la force du terme de pénalisation. Comme nous pouvons le voir sur le tracé

des chemins de régularisation, plus λ augmente, plus il y a de coefficients nuls dans $\hat{\beta}^*$, ce qui permet d'effectuer de la sélection de variables.

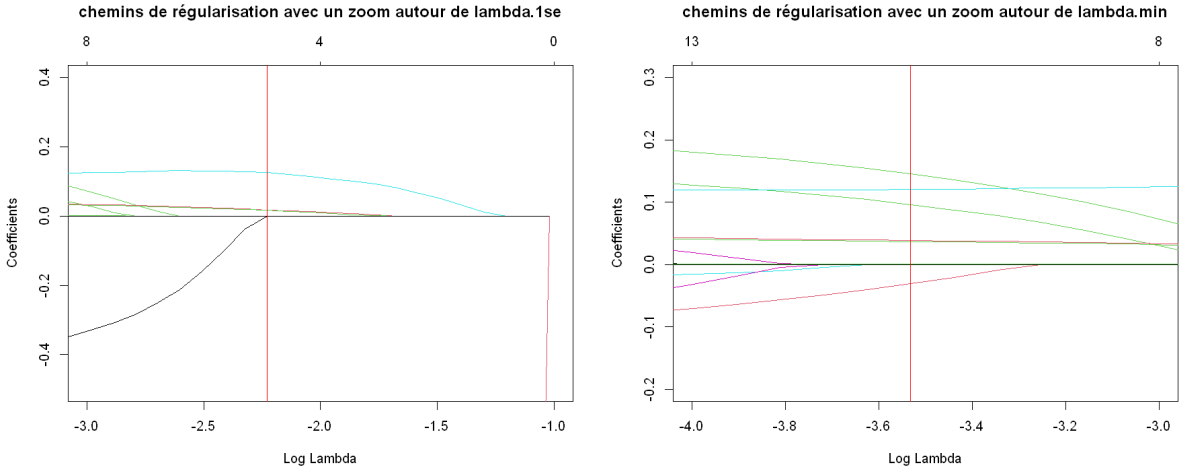


FIGURE 9 – Chemins de régularisation de LASSO

Avec un test de **10-fold cross validation**, on peut déterminer deux valeurs de λ qui permettent d'obtenir un estimateur efficace.
- λ_{min} est la valeur pour laquelle l'erreur de généralisation obtenue est la plus faible, la prédiction est donc précise. On obtient ici un estimateur $\hat{\beta}$ avec 9 paramètres non nuls (plus l'intercept).
- λ_{1se} est la plus grande valeur de λ tel que l'erreur de généralisation soit à une distance de une erreur standard de l'erreur minimale. Elle donne une erreur de généralisation faible tout en diminuant le nombre de variables et permet de rendre le modèle plus inteprétable. On obtient ici l'estimateur $\hat{\beta}$ avec 4 paramètres non nuls (plus l'intercept).

3. Comparaison des résultats

Paramètres sélectionnés : Avec λ_{min}, les paramètres qui semblent avoir le plus d'influence sur les précipitations sont precip_mod, u10_arome, v10_arome et msl_arome_mod. Cela est assez cohérent avec le corrplot obtenu en figure 2 car on avait que precip_mod et tp_arome étaient très corrélées, et la sélection de variables permet de ne garder que precip_mod. De même, on garde ws_arome qui était très corrélée avec precip_mod, tp_arome et ff_mod et on supprime ces dernières. Nous obtenons les résultats en figure 10.

	Avec R en régression :		Avec R en régression avec seuillage :		Avec Python en régression :		Avec Python en régression avec seuillage :		
Erreurs / Modèles	MSE	Erreur de généralisation	Pureté	F1-score	Erreur de généralisation	MAPE	Pureté	F1-score	Mallows score
Modèle déterministe AROME	-	4.1073			3.73	40.69	0.40	0.22	0.5
Linéaire simple	0.3796	0.4251	0.5217	0.4643	0.51	36.86	-	-	-
Linéaire Lasso λ_{min}	0.3973	0.4285	0.4928	0.4074	-	-	-	-	-
Linéaire Lasso λ_{1se}	0.4351	0.4464	0.4855	0.3265	-	-	-	-	-
Linéaire Lasso λ par CV 5-folds	-	0.4275	-	-	0.49	36.65	0.55	0.37	0.48

FIGURE 10 – MSE et erreur de généralisation des modèles linéaires

Dans la figure 10, nous observons que la MSE obtenue avec LASSO pour R est moins bonne car comme il s'agit d'un sous modèle, l'erreur commise est plus grande. Cependant, ce n'est pas très important car on souhaite surtout minimiser l'erreur de généralisation. Or, l'erreur de

généralisation est plus faible pour la régression sans sélection de variables pour R, ce qui n'est pas le cas en python. Pénaliser un modèle ne garantit donc pas toujours d'avoir des meilleurs résultats sur l'erreur de généralisation, mais évite le surapprentissage.

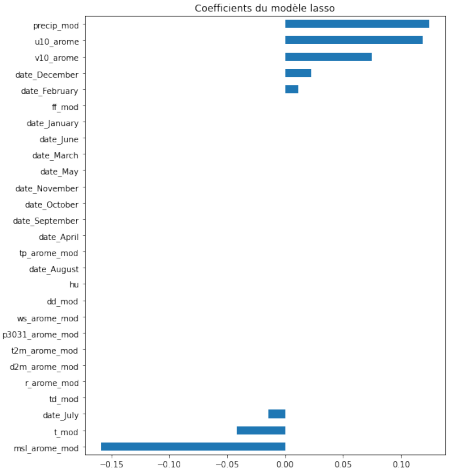


FIGURE 11 – Effet de la régression Lasso

En python, on obtient des résultats similaires à R pour Lasso, avec λ=0.3 obtenu par CV-5-folds, la pénalisation Lasso garde 8 variables et en supprime 19 (Figure 11), rendant le modèle plus interprétable. Les paramètres qui semblent avoir le plus d'influence sur les précipitations sont precip_mod, u10_arome, v10_arome et msl_arome_mod. On constate que les composantes du vent méridionale et zonale (u_10 , v_10) sont gardées par Python alors que R a gardé ff (vent).

Remarque : Sur Python, certaines fonctionnalités pour les modèles linéaires généralisés comme les stratégies classiques (forward, backward...) de sélection de variables par optimisation d'un critère (Cp, AIC, BIC) ne sont pas disponibles. La prise en compte des interactions et de leur sélection ne sont pas prévues. De plus l'interprétation est compliquée par la dissociation de chaque variable qualitative en paquets d'indicatrices. C'est encore compréhensible avec peu de variables mais devient rapidement inexploitable. C'est pourquoi nous avons uniquement les résultats en R.

VII.2 Modèle quadratique

Un modèle de **régression quadratique** suppose des interactions d'ordre 2 entre les variables.

1. Sélection de variables par le critère AIC

Dans un premier temps, nous effectuons une sélection par le critère $AIC(m) = -2log\mathcal{L} + 2k$ où $\mathcal{L}$ est le maximum de la log-vraisemblance du modèle m et k le nombre de paramètres. Le modèle choisi est celui qui minimise la valeur du AIC. En minimisant $-2log\mathcal{L}$, on favorise la qualité d'ajustement du modèle tout en réalisant un compromis avec la complexité en pénalisant les modèles ayant un trop grand nombre de paramètres avec le terme $2k$.

2. Sélection de variables par régularisation L1 (LASSO)

Nous effectuons une régression pour le modèle avec interactions. Comme nous l'avons vu précédemment, cela permet faire de la réduction de modèle.

3. Comparaison des résultats

	Avec R en régression :		Avec R en régression avec seuillage :	
Erreurs / Modèles	MSE échantillon train	Erreur de généralisation	Pureté	F1-score
Quadratique simple	0.1602	0.9586	0.5217	0.4416
Quadratique AIC	0.1968	0.7493	0.5072	0.4507
Quadratique Lasso λ_{min}	0.3858	0.4408	0.4855	0.3265

FIGURE 12 – Performance des modèles quadratiques

La figure 12 résume les performances obtenues. Nous trouvons que le modèle quadratique avec sélection de variables LASSO donne les meilleurs résultats.

Remarque : Python ne permet pas de modéliser les interactions.

Classification post régression après seuillage

Réel / Prédit	Avec R pour modèle quadratique simple : high_rain	low_rain	no_rain
high_rain	17	15	9
low_rain	17	42	17
no_rain	2	6	13

Réel / Prédit	Avec Python, lasso CV 5 folds : high_rain	low_rain	no_rain
high_rain	14	4	1
low_rain	21	52	46
no_rain	0	0	0

FIGURE 13 – Table de contingence, classification à partir de la régression après seuillage

Dans la Figure 13, on voit que pour Python, le modèle arrive bien prédire low_rain et high_rain, mais ne prédit aucun no_rain. Pour R en revanche, le modèle prédit no_rain.

VII.3 Comparaison entre modèle linéaire et quadratique

Pour comparer les résultats entre les différentes méthodes, on trace le graphe des résidus et les erreurs (figure 14).

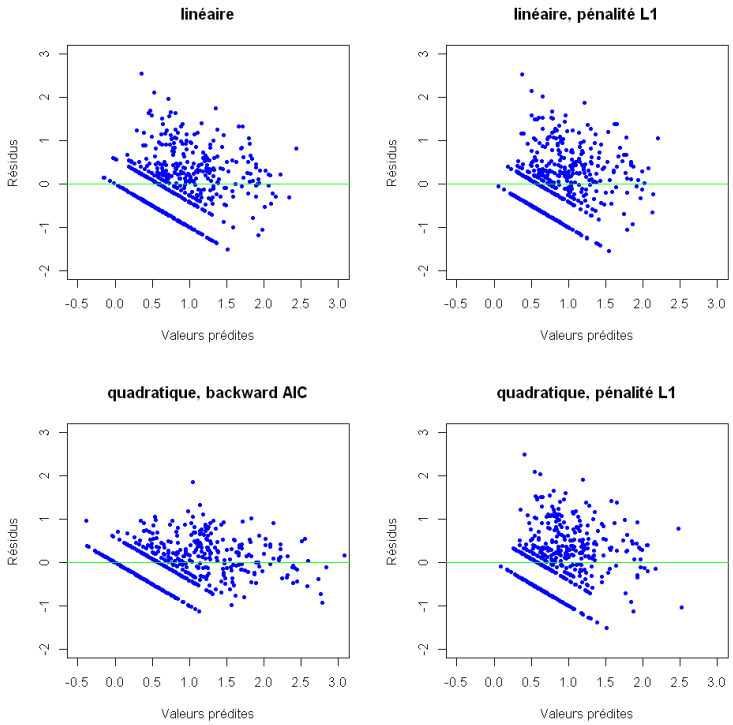


FIGURE 14 – graphe des résidus

Nous pouvons conclure que le modèle quadratique donne de meilleurs résultats que le modèle linéaire. De plus, l'erreur MSE du modèle AROME vaut $\boxed{MSE = 4.1073}$ donc le modèle statistique améliore la prédiction par rapport au modèle AROME dans tous les cas.

VIII Modèle binomial ou régression logistique

La **régression logistique** s'applique à la classification binaire lorsque $Y \in \{-1, 1\}$ où $\mathbb{P}_\theta(Y = 1|X = x) = \frac{e^{\langle\theta,x\rangle}}{1+e^{\langle\theta,x\rangle}}$ avec $\theta \in \mathbb{R}^p$. L'estimateur du maximum de vraisemblance $\hat{\theta}$ est approximé par l'algorithme de Newton-Raphson. Cependant, la variable qualitative à expliquer **rain_class** n'est pas binaire et possède trois modalités. On utilise alors la **regression logistique multiclasse** qui apprend tour à tour une classe contre toutes les autres.

1. Régression logistique sans interaction

Nous appliquons d'abord le modèle de **régression logistique sans interaction** et optimisons le nombre de paramètres avec le critère AIC. Pour chaque individu, nous obtenons les trois probabilités d'appartenir aux modalités high_rain, low_rain et no_rain. Pour déterminer les valeurs prédites $\hat{Y}$, il suffit d'attribuer à chaque individu la modalité ayant la plus forte probabilité. Les résultats sont présentés dans la table de contingence en figure 15.

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	18	11	3
low_rain	17	35	16
no_rain	1	17	20

Prédit \ Réel	Avec Python : high_rain	low_rain	no_rain
high_rain	20	9	4
low_rain	13	42	25
no_rain	2	5	18

FIGURE 15 – Table de contingence, régression logistique multiclasse sur échantillon de test

On observe que :
- Lorsque la valeur observée est **high_rain**, le modèle se trompe une fois sur deux et prédit low_rain.
- Lorsque la valeur observée est **low_rain**, le modèle est un peu meilleur et fournit une prédiction correcte dans 62% des cas et se trompe avec équiprobabilité entre high_rain et no_rain. - Lorsque la valeur observée est **no_rain**, le modèle se trompe une fois sur deux et prédit low_rain.
Afin de quantifier la performance du modèle, on calcule l'indice de **pureté** $\mathcal{P}$. On trouve $\boxed{\mathcal{P}_R = 0.5290}$ et $\boxed{\mathcal{P}_{Python} = 0.5797}$. De plus l'indice de Mallows vaut $\boxed{\text{M=0.46}}$

2. Régression logistique avec interactions

Nous appliquons à présent la **régression logistique multiclasse** avec interactions entre les variables et optimisons le nombre de paramètres par *stepwise sélection*. De la même façon que précédemment, nous déterminons $\hat{Y}$ et présentons les résultats dans la table de contingence suivante en figure 16

Réel / Prédit	Avec R : high_rain	low_rain	no_rain
high_rain	16	10	4
low_rain	18	40	20
no_rain	2	13	15

FIGURE 16 – Table de contingence, régression logistique multiclasse avec interactions sur échantillon de test

Nous pouvons observer des résultats similaires à ceux du modèle sans interactions. Néanmoins, nous trouvons une pureté $\boxed{\mathcal{P}_R = 0.5145}$, ce qui est légèrement meilleur que pour le modèle sans interactions.
Remarque : Python ne permet pas de modéliser les interactions.

IX Analyse discriminante

La LDA (Linear Discriminant Analysis) est une méthode de classification supervisée. Elle tient en effet compte de class_rain pour faire l'ACP. C'est donc une sorte d'ACP supervisée. Elle suppose deux hypothèses : que la structure de variance soit identique pour toutes les classes (frontière linéaire entre les classes), et que les données suivent une distribution gaussienne. En classification multiclasse, trois frontières (high/low, no/low, high/no) sont optimisées de manière à maximiser la fonction discriminante. Ensuite, nous prédisons pour chacune des paires le $\hat{Y}_{test}$. Nous choisissons la classe qui a été prédite le plus de fois.

1. Estimation des modèles par LDA

Ici on voit sur la figure 17 que les nuages ne sont pas forcément sphériques, car les variances ne sont pas forcément identiques dans les différentes activités. On voit qu'elle n'est pas vraiment discriminante, que les classes low_rain et no_rain sont presque confondues. Seule la classe

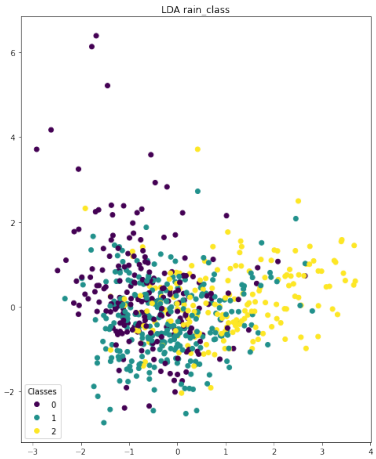


FIGURE 17 – LDA

high_rain est légèrement plus discriminée, mais cela reste encore relatif. On fait une cross table sur la figure 18, pour faire l'erreur de généralisation obtenue sur l'échantillon de test, on voit que cette matrice est presque symétrique. Le taux de réussite n'est pas très bon pour chaque classe.

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	82	53	17
low_rain	28	153	48
no_rain	17	77	75

Prédit \ Réel	Avec Python : high_rain	low_rain	no_rain
high_rain	17	10	4
low_rain	17	38	14
no_rain	2	15	21

FIGURE 18 – Table de contingence, LDA sur l'échantillon de test

2. Prévision de l'échantillon de test

Afin de quantifier la performance du modèle, on calcule l'indice de **pureté**. On obtient $\boxed{\text{Pureté}_R = 0.55}$ et $\boxed{\text{Pureté}_{Python} = 0.54}$.

X Arbre de décision binaire ou arbre CART

L'**arbre de décision binaire** est une méthode non paramétrique qui ne nécessite aucune hypothèse. L'arbre est très interprétable, mais instable de par sa construction.
A chaque noeud de l'arbre nous cherchons à minimiser le critère $\text{Crit}(A) = D(A) + \gamma \times |A|$ où $D(A)$ est la somme des hétérogénéités de chaque feuille de l'arbre, $|A|$ le nombre de feuilles de l'arbre et $\gamma > 0$ la force du terme de pénalisation à optimiser. Plus γ augmente et plus l'arbre obtenu est élagué.
La **première étape** consiste à calculer l'arbre maximal A_{max} (pour lequel chacune des feuilles ne contient que des valeurs Y homogènes). Pour ce faire, à chaque noeud, on divise selon une variable et un seuil choisis tels que le découpage maximise la décroissance en hétérogénéité.
La **deuxième étape** consiste à élaguer l'arbre maximal pour éviter le surajustement. Il s'agit de "couper" les feuilles issues d'une division pour laquelle l'amélioration en hétérogénéité est inférieure à γ. Après itération de cet algorithme, nous obtenons une séquence d'arbres imbriqués : $A_{max} \supset A_\kappa \supset A_{\kappa-1} \supset ... \supset A_1$.
Pour prédire la valeur d'une nouvelle observation, on fait passer cette observation dans l'arbre

et on observe dans quelle feuille elle tombe. Enfin, différents paramètres sont à optimiser : la pénalisation γ, la profondeur de l'arbre ou encore le nombre minimal d'observations par noeud.

1. Estimation et élagage de l'arbre de régression

En régression, l'hétérogénéité d'un noeud correspond à l'erreur quadratique. Nous souhaitons ici expliquer la variable **rain_mod**. Nous commençons par construire A_{max} puis nous optimisons par **validation croisée 10-folds** la valeur de γ. Nous obtenons les arbes élagués figure **??** (les résultats obtenus avec R sont disponibles en annexe A.3).

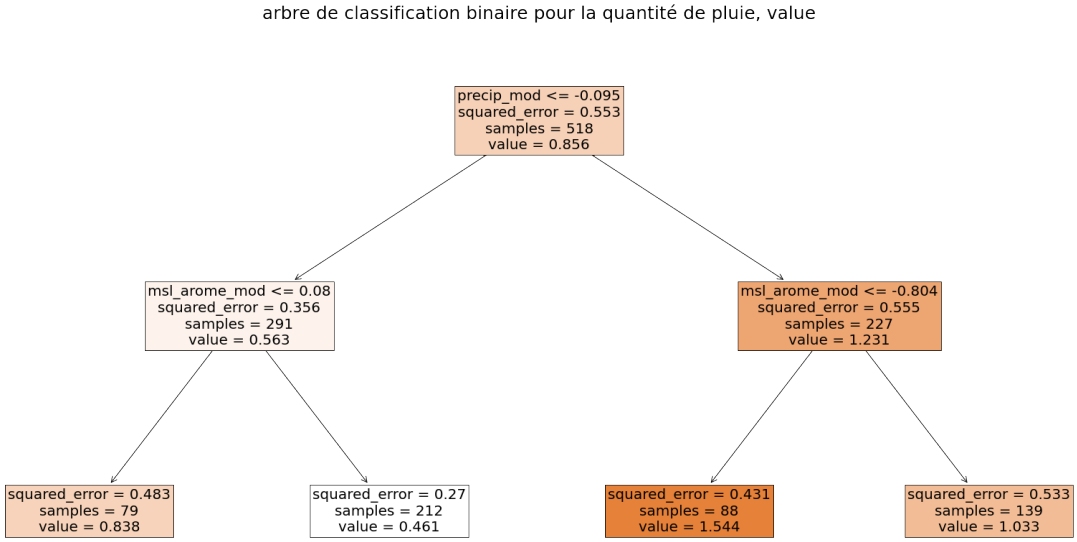


FIGURE 19 – CART obtenu avec Python

Les variables permettant la décision diffèrent entre les deux langages. Cela peut être du au fait que les arbres sont très instables, et donc dépendent de l'échantillon d'apprentissage choisi et qu'en python, il n'y a pas de découpage en termes de modalités pour les variables qualitatives. La variable qui contribue le plus à l'interprétation est msl_arome_mod (pression atmosphérique du jour j+1 par AROME). Ceci semble cohérent car le temps est à la pluie lorsque la pression atmosphérique est basse. Nous trouvons ensuite precip_mod (quantité totale des précipitation du jour j) et hu (humidité du jour j).

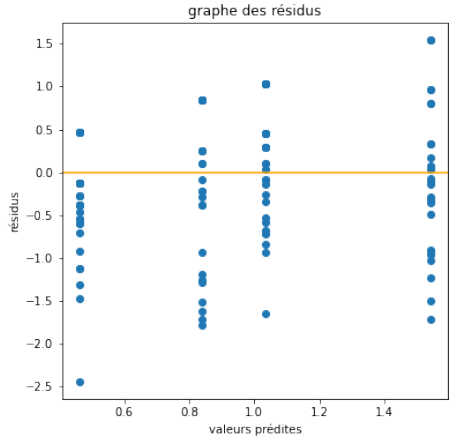


FIGURE 20 – Graphe des résidus CART

Notre arbre élagué présente 4 feuilles. En régression, la valeur prédite pour chaque feuille est constante et correspond à la moyenne des observations Y_i tombées dans la feuille pendant l'apprentissage. Il y a donc autant de prédictions que de feuilles. Cette prédiction constante par morceaux explique la forme striée du graphe des résidus Figure 20 (en abscisse chaque valeur correspond à la valeur d'un noeud).

	Avec R en régression :		Avec R en régression avec seuillage :		Avec Python en régression :			Avec Python en régression avec seuillage :		
Erreurs / Modèles	MSE	Erreur de généralisation	Pureté	F1-score	Erreur de généralisation	MAPE	Score python	Pureté	F1-score	Mallows score
Arbre de régression binaire	0.4114	0.50	0.4928	0.3529	0.52	36.65	0.75	0.49	0.39	0.56

FIGURE 21 – Performances de l'arbre de régression binaire

Classification post régression après seuillage

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	9	4	2
low_rain	27	59	37
no_rain	-	-	-

Prédit \ Réel	Avec Python, lasso CV 5 folds : high_rain	low_rain	no_rain
high_rain	17	6	0
low_rain	18	50	47
no_rain	0	0	0

FIGURE 22 – Table de contingence après seuillage des résultats obtenus post régression pour les arbres CART

On voit dans la Figure 22, que no_rain n'est pas prédit par le modèle CART en Python comme en R.

2. Estimation et élagage d'un arbre de discrimination

En classification, l'hétérogénéité d'un noeud correspond à la Gini-Concentration ou à la Cross Entropy. Nous souhaitons ici expliquer la variable **rain_class** en procédant de la même façon qu'en régression (cf. Arbres en annexe A.5 et A.4). Les variables permettant la décision diffèrent grandement entre les deux langages. Ceci peut s'expliquer de la même façon que pour la régression. On retrouve : la variable d'importance msl_arome_mod puis hu, date (mois) et u10_arome pour R et la variable d'importante precip_mod puis td_mod,d2m_arome_mod, msl_arome_mod et p3031_arome_mod pour Python.

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	15	10	4
low_rain	19	39	21
no_rain	2	14	14

Prédit \ Réel	Avec Python : high_rain	low_rain	no_rain
high_rain	19	17	7
low_rain	11	35	21
no_rain	5	4	19

FIGURE 23 – Table de contingence, Arbre binaire sur échantillon de test

Pour la pureté, nous obtenons $\boxed{\mathcal{P}_R = 0.4926}$ et $\boxed{\mathcal{P}_{Python} = 0.5289(1 - score = 0.47)}$. De plus :
- Lorsque la valeur observée est **high_rain** le modèle se trompe une fois sur deux et prédit low_rain.
- Lorsque la valeur observée est **low_rain** le modèle se trompe dans moins de 40% des cas et prédit high_rain et no_rain.
- Lorsque la valeur observée est **no_rain** le modèle se trompe dans plus de 60% des cas et prédit low_rain et high_rain avec la même probabilité.

XI Agrégation de modèles

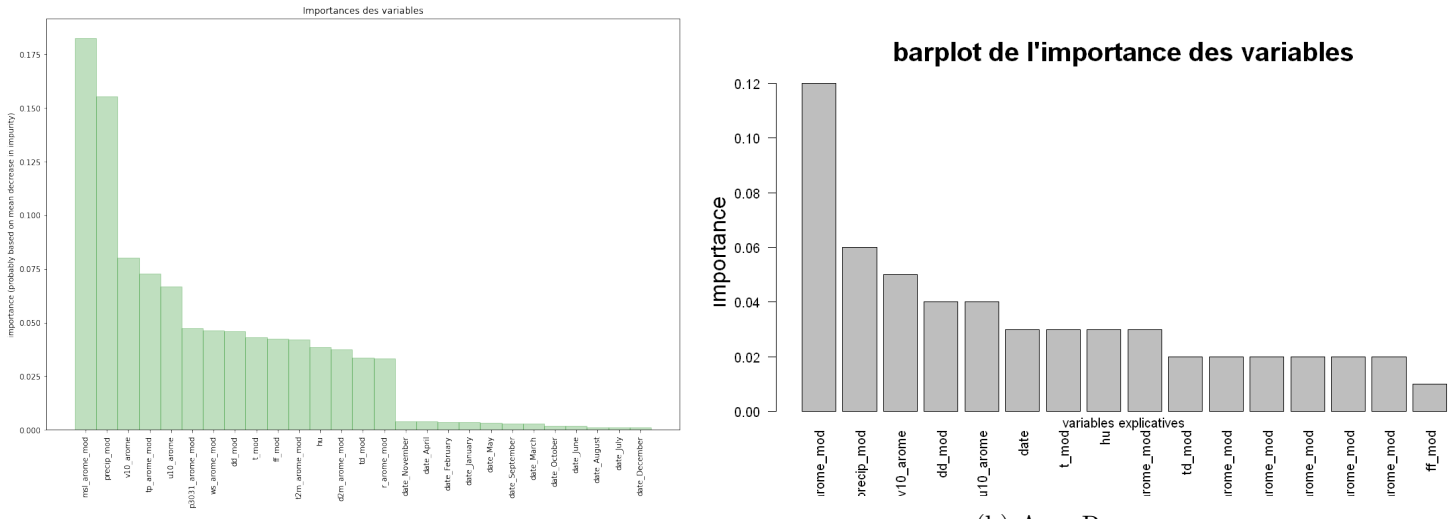
XI.1 Forêts aléatoires

Le **bagging** agrège plusieurs arbres CART quasiment indépendants entre eux. Ceci permet de réduire l'instabilité d'un seul arbre et d'augmenter la robustesse, en réduisant la variance.

Pour avoir des arbres aussi indépendants que possible, le bagging construit B arbres CART sur B échantillons **bootstrap** (un échantillon bootstrap est un échantillon de taille n tiré aléatoirement avec remise dans le dataset). L'algorithme **Random Forest** (RF) utilise un principe supplémentaire à l'algorithme du bagging. Pour augmenter la robustesse et l'indépendance des arbres de la forêt, l'algorithme RF choisit aléatoirement m variables parmi les p disponibles, pour effectuer un découpage à un noeud. On choisit alors le seuil et la variable donnant le meilleur découpage parmi les m variables tirées. Ceci permet de sélectionner des variables qui n'auraient jamais été sélectionnées autrement car de meilleures variables auraient été préférées. Il est à noter qu'il n'y a pas d'étape d'élagage car l'agrégation permet de contrebalancer le surajustement que peuvent produire des arbres maximaux et l'élagage produirait des arbres trop corrélés entre eux. Les paramètres à optimiser sont le nombre d'arbres dans la forêt *ntree*, la profondeur maximale des arbres, le nombre minimal et maximal d'individus dans une feuille et le nombre de variables m pouvant être choisies à chaque noeud.
Enfin, pour trouver la prédiction $\hat{Y}_0$ d'une nouvelle observation x_0, on fait passer cette observation dans chaque arbre de la forêt et on regarde dans quelle feuille elle tombe. En classification, on prédit pour x_0 la classe majoritaire (classe obtenue le plus grand nombre de fois pour $\hat{Y}_0$ parmi tous les arbres) et en régression, on prédit pour x_0 la moyenne de tous les $\hat{Y}_0$ prédits par tous les arbres.

1. Estimation d'une forêt aléatoire en régression

Dans un premier temps, nous souhaitons expliquer **rain_mod**. Nous optimisons les valeurs de m et de *ntree* en cherchant à minimiser l'erreur sur l'échantillon de test et l'erreur sur l'échantillon **Out-Of-Bag** (données non tirées dans la construction des échantillons bootstraps, ce qui représente environ $\frac{1}{3}n$) et trouvons $m = 5$ et $ntree = 650$. Comme la forêt n'est pas interprétable, on calcule l'**importance** des variables pour savoir quelles variables ont eu le plus d'influence sur la prédiction obtenue. Contrairement au cas de l'arbre CART, le critère d'importance (Gini en Python et MDA en R) est calculé après construction de la forêt et n'est pas nécessairement la première variable des arbres.



(a) Avec Python (b) Avec R

FIGURE 24 – Importance des variables explicatives dans le modèle Random Forest

Nous observons (figure 24) que msl_arome_mod (pression atmosphérique prédite par AROME qui n'avait pas été choisie pour l'arbre CART) est ici la variable la plus importante en R et en Python. Ceci peut s'expliquer car msl_arome_mod est une variable qui est très corrélée avec un grand nombre de variables explicatives (comme precip_mod) comme on l'avait vu précédemment dans le graphe des corrélations. C'est aussi une variable qui a été conservée dans le modèle de régression pénalisée avec Lasso. La variable ayant le plus d'impact sur les précipitations du lendemain est donc la pression atmosphérique prédite pour le jour suivant. Cependant, l'importance des autres variables est assez différente entre les deux modèles.

Dans la classification post régression, on voit que la variable low_rain n'est encore pas prédite.

	Avec R en régression :		Avec R en régression avec seuillage :		Avec Python en régression :			Avec Python en régression avec seuillage :		
Erreurs / Modèles	MSE	Erreur de généralisation	Pureté	F1-score	Erreur de généralisation	MAPE	Score python	Pureté	F1-score	Mallows score
Random Forest en régression	0.4121	0.4356	0.51	0.4483	0.4646	34.54	0.6695	0.49	0.38	0.42

FIGURE 25 – Performances de Random Forest en régression

Classification post régression après seuillage

Réel / Prédit	Avec R : high_rain	low_rain	no_rain
high_rain	13	6	3
low_rain	23	57	36
no_rain	-	-	-

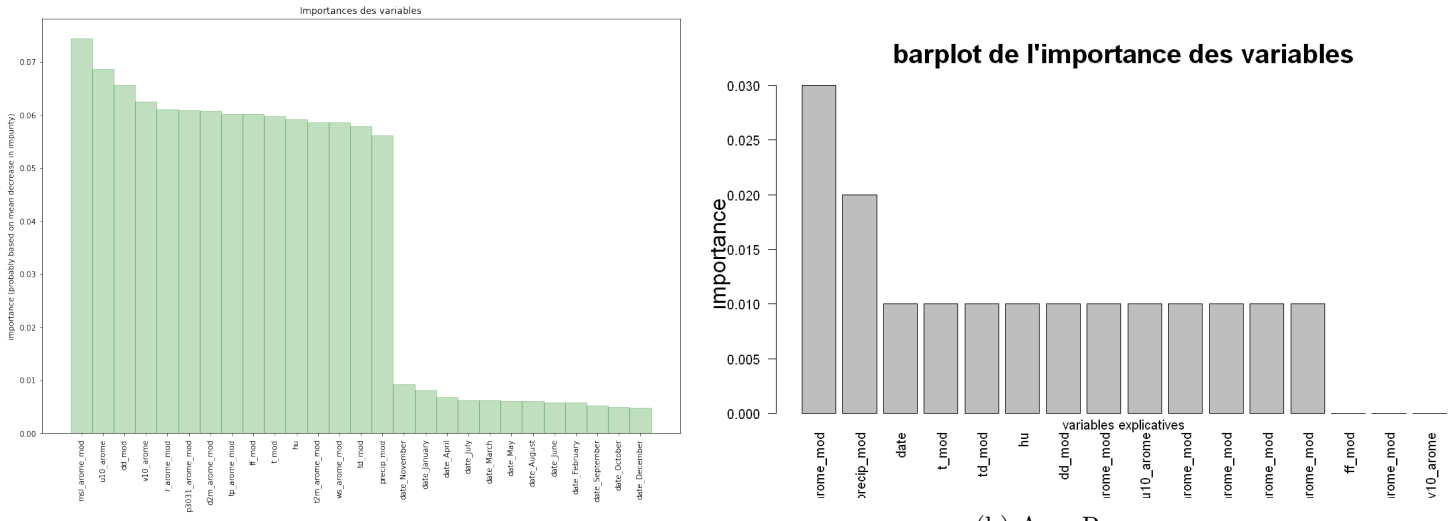
Réel / Prédit	Avec Python, lasso CV 5 folds : high_rain	low_rain	no_rain
high_rain	15	4	1
low_rain	20	52	46
no_rain	0	0	0

FIGURE 26 – Table de contingence après seuillage des résultats obtenus post régression pour les random forest

On voit que dans la Figure 26, on voit que no_rain n'est pas prédit par le modèle Random Forest en python et en R.

2. Estimation d'une forêt aléatoire en classification

Nous procédons de la même façon qu'en régression et trouvons les paramètres optimaux $m = 4$ et $ntree = 700$.



(a) Avec Python

(b) Avec R

FIGURE 27 – Importance des variables explicatives dans le modèle Random Forest

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	17	15	5
low_rain	17	31	13
no_rain	2	17	21

Prédit \ Réel	Avec Python : high_rain	low_rain	no_rain
high_rain	20	11	5
low_rain	12	37	22
no_rain	3	8	20

FIGURE 28 – Table de contingence, Random Forest sur échantillon de test

En Python, contrairement à R, il n'est pas possible de faire les découpages aux noeuds selon les modalités d'une variable qualitative. A l'inverse, chaque modalité est vue comme une variable à part entière selon laquelle se fait le découpage. Cela pourrait expliquer certaines différences sur les graphiques d'importance (figure 27).
Pour la pureté, nous obtenons $\boxed{\mathcal{P}_R = 0.5000}$ et $\boxed{\mathcal{P}_{Python} = 0.5434(1 - score = 0.4420)}$.

XI.2 Boosting

Une autre manière de faire de l'agrégation de modèles est d'utiliser le **boosting**. Il s'agit de réaliser une combinaison linéaire d'une séquence récursive de prédicteurs faibles où chaque prédicteur, pondéré par un certain poids, est une version adaptée du prédicteur précédent. L'aggrégation permet de réduire la variance et le biais des prédicteurs. A chaque étape n, des poids plus importants sont attribués aux données qui ont été mal prédites par le prédicteur à l'étape $n-1$.
L'algorithme historique *Adaboost* est dédié à la classification binaire et minimise une fonction perte exponentielle. Le *Gradient Boosting Models (GBM)* est une amélioration d' *Adaboost*. Cet algorithme repose toujours sur une agrégation de prédicteurs où chaque prédicteur agrégé constitue une étape conduisant à une amélioration de la prédiction. La différence réside dans le fait que cette étape est faite en direction du gradient de la fonction perte, le gradient étant lui-même approximé par un arbre de régression. Cette méthode s'applique aussi bien en régression qu'en classification.
Pour l'algorithme GBM, les paramètres à calibrer sont :
- La valeur du *shrinkage* (en R) ou *learning rate* (en Python) qui permettent d'éviter le surajustement en redimensionnant la contribution de chaque arbre intervenant dans *GBM* par un facteur ν.
- La profondeur de l'arbre à calibrer par cross-validation.
- Le nombre d'itérations M. En effet, chaque itération de l'algorithme de boosting réduit la perte empirique et une valeur de M trop importante mène à du surajustement. Une façon de calibrer ce paramètre est d'estimer l'erreur de généralisation sur un échantillon de validation (par exemple en utilisant la validation croisée) et de choisir la valeur de M qui minimise cette erreur. Une autre façon de procéder est de faire de *l'early stopping* c'est-à-dire interrompre l'entraînement d'un modèle quand la perte d'un ensemble de données de validation commence à augmenter et que les performances sur le test se dégradent.

1. Cas de la régression

Nous commençons par appliquer le boosting dans le cas de la régression pour expliquer **rain_mod**. La fonction *gbm* est utilisée en R et suppose que la distribution des variables est

gaussienne. Dans un premier temps, on optimise le nombre d'itérations M à l'aide du graphique figure 29.

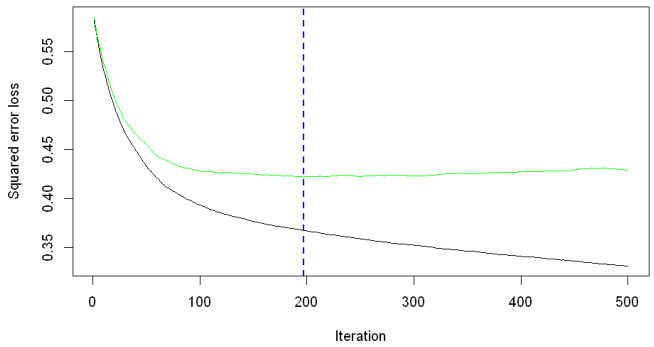


FIGURE 29 – Erreur en fonction du nombre d'itérations

Sur le graphique figure 29, la courbe verte représente l'erreur de validation croisée 10-fold (calculée avec l'échantillon d'entraînement) en fonction du nombre d'itérations. La courbe noire représente l'erreur de généralisation (calculée avec l'échantillon de test) en fonction du nombre d'itérations. On trouve que $M = 197$ itérations minimise ces deux erreurs (pour une valeur de *shrinkage* fixée).
On peut également optimiser la valeur du *shrinkage*. Pour ce faire, on calcule à une valeur de M fixée, l'erreur de généralisation pour différents *shrinkage*. On trouve que *shrinkage* $= 0.05$ minimise l'erreur. Enfin, on redéfinit le boosting avec les paramètres trouvés et on obtient les résultats figure 30.

	Avec R en régression :		Avec R en régression avec seuillage :		Avec Python en régression :			Avec Python en régression avec seuillage :		
Erreurs / Modèles	MSE	Erreur de généralisation	Pureté	F1-score	Erreur de généralisation	MAPE	Score python	Pureté	F1-score	Mallows score
Boosting en régression	0.4326	0.3335	0.5	0.4483	0.4606	29.04	0.6637	0.49	0.39	0.47

FIGURE 30 – Performances de Boosting en régression

Classification post régression après seuillage

Réel / Prédit	Avec R : high_rain	low_rain	no_rain
high_rain	13	7	2
low_rain	23	55	36
no_rain	0	1	1

Réel / Prédit	Avec Python, lasso CV 5 folds : high_rain	low_rain	no_rain
high_rain	17	5	1
low_rain	18	51	46
no_rain	0	0	0

FIGURE 31 – Table de contingence après seuillage des résultats obtenus post régression pour la méthode boosting

On voit dans la Figure 31 que no_rain n'est jamais prédite pour les résultats en Python tandis qu'il est très faiblement prédit en R.

2. Cas de la discrimination

Nous poursuivons en appliquant le boosting dans le cas de la classification multiclasse pour expliquer **rain_class** (qui prend les modalités high_rain, low_rain et no_rain). La fonction

gbm utilisée en R suppose une distribution multinomiale. De la même façon que précédemment, nous déterminons que le nombre d'itérations optimal est $M = 94$ à l'aide du graphique 32.

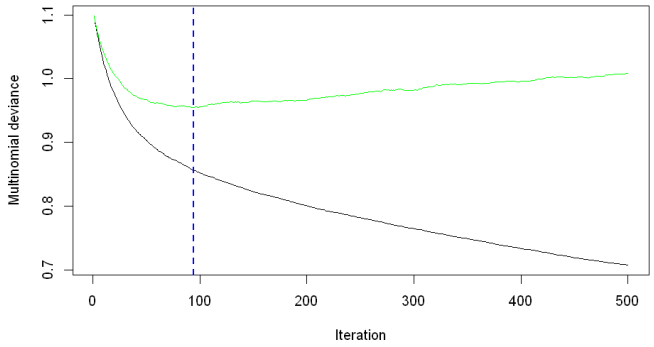


FIGURE 32 – Erreur en fonction du nombre d'itérations

Afin de déterminer le *schrinkage* optimal, on calcule l'indice de **pureté** pour différentes valeurs du *schrinkage* à M fixé. On trouve alors *schrinkage* $= 0.03$. On redéfinit le boosting avec les paramètres optimaux et on obtient la table de contingence figure 33.

Réel / Prédit	Avec R : high_rain	low_rain	no_rain
high_rain	16	11	4
low_rain	19	20	15
no_rain	1	12	20

Réel / Prédit	Avec Python : high_rain	low_rain	no_rain
high_rain	19	7	4
low_rain	13	44	22
no_rain	3	5	21

FIGURE 33 – Table de contingence pour la classification avec le boosting

Pour la pureté, nous obtenons $\boxed{\mathcal{P}_R = 0.5507}$ et $\boxed{\mathcal{P}_{Python} = 0.6086 (1 - score = 0.3913)}$. Nous observons que :

- Lorsque la valeur observée est **high_rain** le modèle se trompe une fois sur deux et prédit low_rain.
- Lorsque la valeur observée est **low_rain** le modèle se trompe aussi une fois sur deux et prédit high_rain et no_rain avec même probabilité environ.
- Lorsque la valeur observée est **no_rain** le modèle se trompe encore une fois sur deux et prédit low_rain.

XII Réseau de neurones

Un **réseau de neurones** ou **perceptron multicouche** est une application non linéaire f par rapport à ses paramètres θ qui associe à une entrée x une sortie $y = f(x, \theta)$. Le réseau est composé de plusieurs couches cachées de neurones où la sortie d'une couche constitue l'entrée de la couche suivante.

Un neurone est une fonction définie par $y_j = f_j(x) = \phi(\langle \omega_j, x \rangle + b_j)$ où $x = (x_1, ..., x_n)$ est l'entrée, ϕ une fonction d'activation et $\omega_j = (\omega_{j,1}, ..., \omega_{j,d})$ (les poids) et b_j (les biais) constituent les paramètres θ. L'optimisation des paramètres θ se fait en minimisant une perte empirique non convexe. Un algorithme de rétro-propagation du gradient permet de trouver un minimum local.

Enfin, les réseaux de neurones s'appliquent en régression comme en classification.
Différents paramètres du réseau sont à déterminer :
- Le paramètre **decay** ou α (valeur par défaut à 0) est le terme de pénalisation λ dans l'expression de la perte empirique $L_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(Y_i, f(X_i, \theta)) + \lambda\Omega(\theta)$ que l'on cherche à minimiser. Il s'agit d'un paramètre de régularisation pénalisant les modèles trop complexes afin d'éviter le surajustement. Aujourd'hui, la méthode du **drop-out** est aussi beaucoup utilisée. Cela consiste à mettre à 0 certains poids du réseau avec une probabilité p et indépendemment des autres.
- Le paramètre **size** détermine le nombre de neurones par couche.

1. Cas de la régression

En régression, la fonction d'activation $\psi(x) = Identité$ est utilisée pour la couche de sortie et la fonction sigmoïde $\phi(x) = \frac{1}{1+e^{-x}}$ ou RELU $\phi(x) = \max(x, 0) + \alpha\min(x, 0)$ est implémentée par défaut pour les couches cachées.
Nous commençons par appliquer un perceptron à une couche cachée pour expliquer la variable **rain_mod**. En utilisant une validation croisée 10-fold (fonction tune sur R), nous trouvons **decay** = 4 et **size** = 3. Avec ces paramètres optimisés, le réseau converge en 140 itérations et comprend 85 paramètres de poids. La table 34 répertorie les résultats obtenus.

	Avec R en régression :		Avec R en régression avec seuillage :		Avec Python en régression :			Avec Python en régression avec seuillage :		
Erreurs / Modèles	MSE	Erreur de généralisation	Pureté	F1-score	Erreur de généralisation	MAPE	Score python	Pureté	F1-score	Mallows score
Réseaux de neurones en régression	0.4280	0.4266	0.5072	0.4407	0.4852	30.31	0.6991	0.485	0.535	0.436

FIGURE 34 – Performances des Réseaux de neurones à une couche en régression

Classification post régression après seuillage

Réel / Prédit	Avec R : high_rain	low_rain	no_rain
high_rain	13	6	4
low_rain	23	57	35
no_rain	-	-	-

Réel / Prédit	Avec Python, lasso CV 5 folds : high_rain	low_rain	no_rain
high_rain	15	5	1
low_rain	20	51	45
no_rain	0	0	1

FIGURE 35 – Table de contingence après seuillage des résultats obtenus post régression pour la méthode réseaux de neurones

Dans la figure 35, on voit que la modalité no_rain n'est pas bien prédite dans les résultats Python et R.

2. Cas de la discrimination

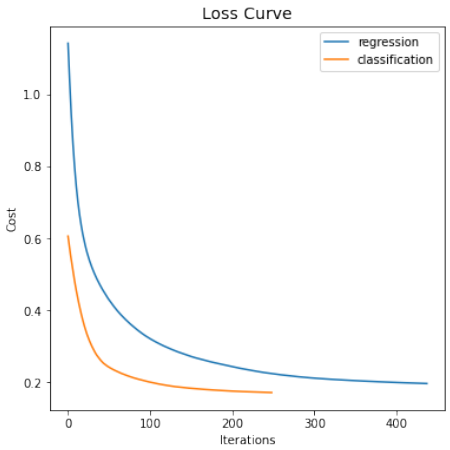


FIGURE 36 – Fonction perte en fonction du nombre d'itérations

En classification binaire, la fonction d'activation sigmoïde $\Psi(x) = \frac{1}{1+e^{-x}}$ est utilisée pour la couche de sortie. En classification multiclasse il s'agit de la fonction softmax multidimensionnelle : $\Psi(x)_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$.
Nous appliquons aussi un perceptron à une couche cachée pour expliquer la variable **rain_class**. Avec la validation croisée 10-fold, **decay** = 5 et **size** = 3. Le réseau converge en 100 itérations et comprend 153 poids. On obtient la table de contingence figure 37.

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	19	12	6
low_rain	16	37	19
no_rain	1	14	14

Prédit \ Réel	Avec Python : high_rain	low_rain	no_rain
high_rain	18	9	4
low_rain	14	37	22
no_rain	3	10	21

FIGURE 37 – Table de contingence pour la classification avec perceptron une couche

On remarque que quelque soit le modèle statistique choisi, on améliore les prédictions par rapport au modèle AROME. Pour **R**, le meilleur algorithme en **régression** et en **classification** est le **Boosting**. Pour **Python**, le meilleur algorithme en **régression** est l'algorithme de **réseaux de neurones** et en **classification** c'est le **Boosting**. Pour la pureté, nous obtenons $\boxed{\mathcal{P}_R = 0.5072}$ et $\boxed{\mathcal{P}_{Python} = 0.5507}$. Nous obtenons quasiment les mêmes résultats que pour CART ce qui n'est pas très concluant.

XIII Support Vector Machine (SVM)

Les **SVM**s sont des des méthodes d'apprentissage supervisé qui permettent de prédire $\hat{Y}$ en classification et en régression. Les paramètres du modèle sont :
- Le coût C qui détermine le nombre de slack variables $\xi_i > 0$ et donc le nombre de vecteurs supports. Plus C augmente et plus il y a un risque de faire du surapprentissage. L'algorithme deviendra alors peu robuste à de nouvelles observations.
- La largeur des marges ϵ.
- Les paramètres liés au choix du noyau : γ (valeur par défaut : $\frac{1}{taille\ donnees}$) pour les noyaux 'rbf', 'poly' ou 'sigmoid' et coef0 (valeur par défaut : 1) correspondant au degré du polynôme des noyaux 'poly' ou 'sigmoid'. Ces paramètres sont optimisés par cross validation de manière à minimiser l'erreur de généralisation.

1. Cas de la régression

Dans le cas de la régression, SVM essaie de trouver une fonction f qui approche au mieux les données (la différence entre $f(x_i)$ et y_i est au plus de ϵ) et qui est la plus "plate" possible. C'est-à-dire que l'on essaye de trouver une fonction qui s'apparente à une droite et qui approxime au mieux nos données envoyées dans un certain espace de Hilbert (RKHS - Reproducing Kernel Hilbert Space). Pour envoyer nos données dans le RKHS, on utilise un *kernel* (ici le *kernel gaussien*, qui donne généralement de bons résultats).
Nous optimisons les paramètres de coût C et γ par validation croisée 10-folds et redéfinissons la SVM avec ces paramètres optimaux $C = 0.5$ et $\gamma = 0.021$.

	Avec R en régression :		Avec R en régression avec seuillage :		Avec Python en régression :			Avec Python en régression avec seuillage :		
Erreurs / Modèles	MSE	Erreur de généralisation	Pureté	F1-score	Erreur de généralisation	MAPE	Score python	Pureté	F1-score	Mallows score
SVM en régression	0.3670	0.4142	0.5072	0.4643	0.4671	33.31	0.50	0.471	0.538	0.446

FIGURE 38 – Performances de la SVM en régression

Nous traçons également le graphe des résidus $\hat{\epsilon}$ en fonction des valeurs prédites $\hat{Y}$ (figure 39) et faisons varier les paramètres C et ϵ pour observer leur effet.

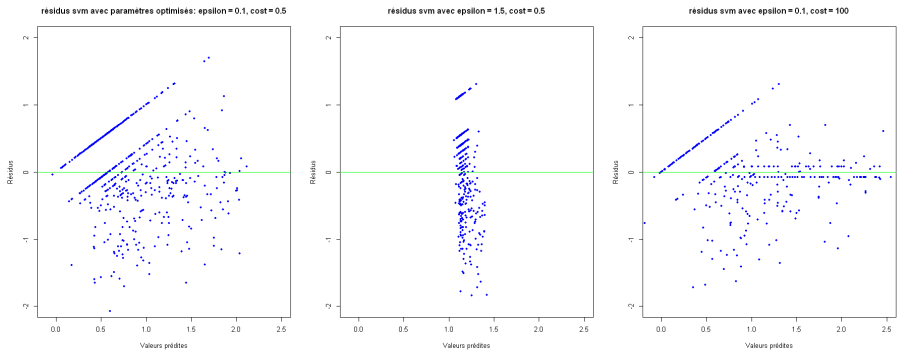


FIGURE 39 – Graphe des résidus pour plusieurs valeurs de C et ϵ

Pour **Python**, le meilleur algorithme en **régression** est l'algorithme de **réseaux de neurones** et en **classification** c'est le **Boosting**. On observe que lorsque ϵ augmente les résidus ont tendance à tendre vers une droite verticale. Rappelons que ϵ calibre la fonction perte l_ϵ qui est insensible aux erreurs plus petites que ϵ. Ainsi, plus ϵ augmente et plus les erreurs sont importantes et donc moins la fonction f s'ajuste aux données. Dans le cas extrême d'un ϵ très grand, la prédiction $\hat{Y}$ devient la moyenne de toutes les données.
Par la suite, on observe que lorsque C augmente nos données forment un tube de taille ϵ autour de 0. En effet, si C augmente, alors les slack variables ξ_i diminuent et donc on n'autorise pas d'erreurs plus grandes que ϵ. Ainsi, lorsqu'une donnée à prédire est hors du tube les valeurs de $\hat{Y}$ prédites vont être constantes et valoir $\pm\epsilon$ pour tenter d'approcher au mieux la donnée sans sortir du tube.

Classification post régression après seuillage

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	13	6	1
low_rain	23	56	37
no_rain	0	1	1

Prédit \ Réel	Avec Python, lasso CV 5 folds : high_rain	low_rain	no_rain
high_rain	15	7	1
low_rain	20	49	45
no_rain	0	0	1

FIGURE 40 – Table de contingence après seuillage des résultats obtenus post régression pour la méthode SVM

2. Cas de la discrimination

Pour la classification binaire on suppose que les données sont linéairement séparables et $Y \in \{-1, 1\}$. La séparation consiste à maximiser une marge de part et d'autre de l'hyperplan séparant les données. Il est aussi possible d'autoriser des mauvaises classifications avec des marges plus flexibles en introduisant des slack variables ξ_i. Dans le cas où les données ne sont pas linéairement séparables, on choisit un espace de Hilbert (RKHS) dans lequel les données sont séparables linéairement, ce qui nous ramène à la résolution du problème dans le cas linéaire. La classification multi-classe, utilisée pour modéliser les trois classes de rain_class, est similaire à la classification binaire. Il s'agit d'apprendre une classe contre toutes les autres.
Remarque : En Python, la stratégie One-vs-One Class est utilisée dans la classification multiclasse (les frontières sont optimisées 2 à 2, puis un vote à la majorité à lieu).

Nous optimisons les paramètres cost C et γ et trouvons $C = 0.75$ et $\gamma = 0.06$. Les résultats sont présentés dans la figure 41.

Prédit \ Réel	Avec R : high_rain	low_rain	no_rain
high_rain	14	11	4
low_rain	22	47	24
no_rain	0	5	11

Prédit \ Réel	Avec Python (RBF) : high_rain	low_rain	no_rain
high_rain	16	7	4
low_rain	15	43	25
no_rain	4	6	19

FIGURE 41 – Table de contingence pour la SVM classification

Modèles \ Erreurs	Avec Python : best params	Pureté
Noyau linéaire	*C :0.03*	0.5072
Noyau polynomial λ_{min}	*C : 0.03, γ : 0.1*	0.5362
Noyau RBF λ_{1se}	C : 10, γ : 0.005	0.5579

FIGURE 42 – Tests sur Python avec différents noyaux

- Pour la prédiction de **low_rain** on voit que le modèle est plutôt bon, il ne se trompe pas dans plus de 70% des cas.
- La prédiction de **high_rain** n'est pas très bonne. Le modèle se trompe plus d'une fois sur deux et prédit low_rain au lieu de high_rain
- Enfin, pour la prédiction de **no_rain** le modèle est médiocre, il se trompe dans 70%.

Pour la pureté, nous obtenons $\boxed{\mathcal{P}_R = 0.5217}$ et $\boxed{\mathcal{P}_{Python} = 0.5579}$.

	Erreur en régression			Erreur en classification	
	Avec R	Avec Python		Avec R	Avec Python
	Erreur Généralisation	Erreur Généralisation	MAPE	Pureté	Pureté
Modèle Arome	4.1073	3.73	40.69	-	-
Modèle linéaire simple	0.4251	0.51	36.86	-	-
Modèle linéaire pénalisation Lasso λ_{min}	0.4285	-	-	-	
Modèle linéaire pénalisation Lasso λ_{1SE}	0.4464	-	-	-	-
Modèle linéaire pénalisation Lasso $\lambda_{CV-5-folds}$	0.4275	0.49	36.65	-	-
Modèle quadratique simple	0.9586				
Modèle quadratique pénalisation AIC	0.7493				
Modèle quadratique pénalisation Lasso λ_{min}	0.4408				
Régression logistique sans interaction	-	-	-	0.5290	0.5797
Régression logistique avec interaction	-	-	-	0.5145	-
Arbres CART	0.5014	0.52	36.65	0.4926	0.5289
Random Forest	0.4356	0.4646	34.54	0.5	0.5434
Boosting	0.3335	0.4606	29.0392	0.5507	0.6086
Réseaux de neurones	0.4066	0.4852	30.31	0.5072	0.5507
SVM	0.4142	0.4671	33.31	0.5217	0.5579

TABLE 1 – Tableau comparatif final

XIV Comparaison finale des résultats

On remarque quelque soit le modèle statistique choisi, on améliore les prédictions par rapport au modèle AROME.On remarque que dans tous les cas, les résultats sont meilleurs qu'avec un modèle aléatoire avec 3 classes.

Pour **R**, le meilleur algorithme en **régression** et en **classification** est le **Boosting**. Pour **Python**, le meilleur algorithme en **régression** est l'algorithme de **réseaux de neurones** et en **classification** c'est le **Boosting**.

Au sein des modèles linéaires celui à privilégier semble être le modèle linéaire simple en R, ce qui semble peu cohérent car par exemple le modèle LASSO est censé faire moins de sur-apprentissage sur les résultats et donc favoriser les erreurs de généralisation plus basses. Pour Python, le modèle LASSO avec λ obtenu par validation croisée est à favoriser permettant une erreur de généralisation plus faible, ce qui semble plus cohérent.

On remarque que les erreurs sont moins importantes avec les modèles agrégés qu'avec les modèles d'arbre. Cela s'explique car les arbres sont particulièrement instables, alors que les Random Forests et les méthodes de Boosting, grâce à techniques de boostrapping et de sélection de variables aléatoires à chaque noeud pour les Random Forests et la combinaison séquentielle de classifieurs pour le Boosting permettent de mieux prendre en compte l'ensemble des variables

et de faire diminuer la variance.

	Erreur de classification (après seuillage en régression)				Erreur en classification (discrimination)		
	Avec Python						
	Pureté	F1-score	Mallows	variable non prédite	Pureté	F1-score	Mallows
Modèle Arome	0.5	0.225	0.395	-	-	-	-
régression logistique/lasso	0.478	0.370	0.554	no_rain	0.579	0.568	0.456
Arbres CART	0.486	0.390	0.410	no rain	0.529	0.521	0.616
Random Forest	0.486	0.381	0.422	no rain	0.558	0.545	0.658
Boosting	0.493	0.394	0.472	no rain	0.61	0.596	0.653
Réseaux de neurones	0.486	0.390	0.546	-	0.551	0.535	0.728
SVM	0.471	0.378	0.447	-	0.558	0.538	0.726

TABLE 2 – Comparaison entre la classification/classification seuillée post régression. La régression logistique/lasso a été comparée avec la régression logistique sans interactions en Python

	Erreur de Classification (après seuillage en régression)			Erreur de Classification (discrimination)	
	Avec R				
	Pureté	F1-Score	variable non prédite	Pureté	F1-Score
Modèle Arome	0.4348	0.5225	no_rain	-	-
Régression Linéaire simple	0.5217	0.4643	-	0.5797	0.5294
Régression Lasso λ_{min}	0.4928	0.4074	-	0.5797	0.5294
Régression Lasso λ_{1se}	0.4855	0.3265	-	0.5797	0.5294
Régression quadratique simple	0.5217	0.4416	-	0.5797	0.5294
Régression quadratique pénalité AIC	0.5072	0.4507	-	0.5797	0.5294
Régression Lasso $_{min}$	0.4855	0.3265	-	0.5797	0.5294
Arbre binaire	0.4928	0.3529	no_rain	0.4926	0.4615
Random Forest	0.51	0.4483	no_rain	0.50	0.4657
Boosting	0.50	0.4483	-	0.5507	0.4776
Réseaux de neurones	0.5072	0.4407	no_rain	0.5072	0.5205
SVM	0.5072	0.4643	-	0.5217	0.4308

TABLE 3 – Comparaison entre la classification et classification seuillée post régression. La régression logistique et quadratique a été comparée avec la régression logistique sans interactions en R.

On vise maintenant avec les tableaux 2 et 3 à comparer les résultats en classification et en régression avec seuillage pour classifier. On choisit à chaque fois la méthode qui permet d'obtenir le meilleur score. On voit que les résultats sont toujours meilleurs en classification sans faire de seuillage avec Python. En R, les résultats sont meilleurs en régression avec seuillage pour Random Forest et pour l'arbre de régression binaire. On voit que la meilleure méthode en classification avec seuillage ainsi qu'en classification sans seuillage est le **Boosting**.

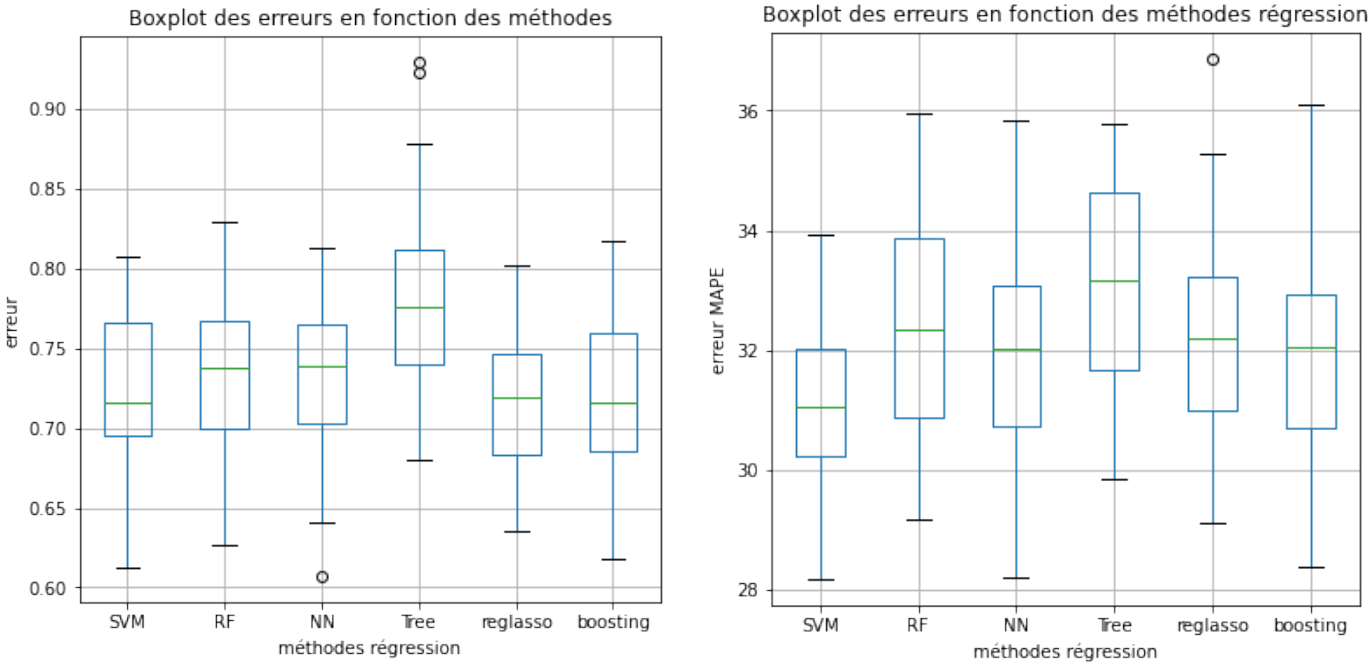
On a constaté que, lorsque l'on a voulu faire de la classification en utilisant les résultats de la régression après seuillage, les résultats étaient très mauvais. Tout d'abord, la variable rain a été considérée comme étant une variable continue, ce qui n'est pas le cas car elle a une masse en 0 comme nous avions pu le voir avec la figure 8. La modalité no_rain est très rarement prédite avec les modèles de classification seuillée, et on a vu que quand c'était le cas, en général très peu de données étaient considérées (moins de 2).

De plus, on peut faire l'hypothèse que certaines variables provenant du modèle de Météo France étant éloignées des vraies prédictions, contribuent aux mauvais résultats des modèles.

XIV.1 Validation Croisée Monte-Carlo

L'échantillon de test est de taille modeste et donc l'estimation de l'erreur de prévision peut présenter une variance importante. Celle-ci est réduite en opérant une forme de validation croisée

(Monte Carlo) en tirant plusieurs couples d'échantillons d'apprentissage et de test pour itérer les traitements précédents. Ici, on pose le nombre d'itérations B=30.



(a) Erreur de généralisation

(b) MAPE (erreur de généralisation)

FIGURE 43 – Boxplot des erreurs obtenues par Monte Carlo avec B=30 en régression

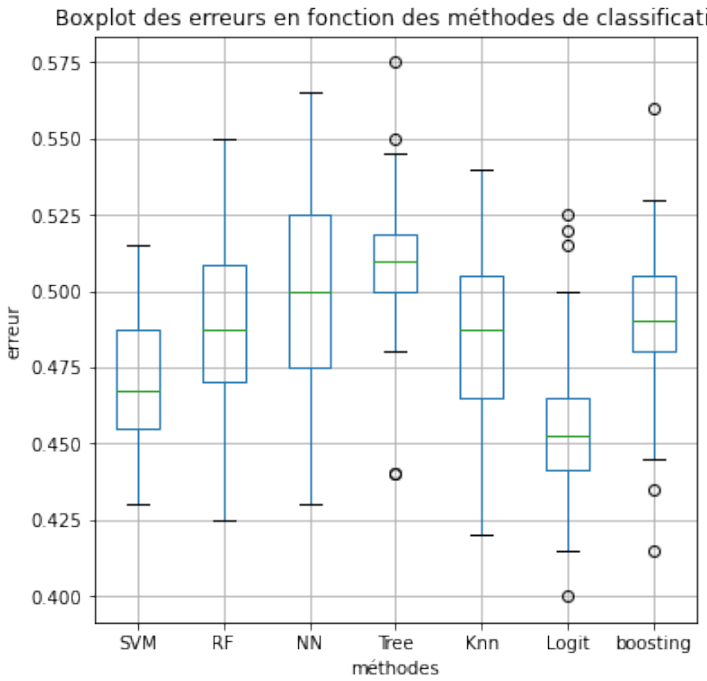


FIGURE 44 – Boxplot des erreurs en classification des différentes méthodes

Sur la Figure 43a et 43b, en régression, on constate que l'erreur de généralisation obtenue par Monte Carlo avec B=30 la plus basse est obtenue avec le modèle SVM permettant de minimiser le MAPE. Cela est différent de précédemment où on obtenait le boosting comme algorithme le plus performant. Pour la figure 44, pour l'erreur de généralisation obtenue par Monte Carlo en classification, on voit que la méthode la plus performante correspond à la méthode de régression logistique. Auparavant, la méthode avec les meilleurs résultats correspondait à celle du boosting. Ce changement peut s'expliquer car la méthode de Monte Carlo est plus robuste. L'échantillon étant de petite taille (environ 688 observations), l'erreur de généralisation avait plus de biais. Faire 30 validations croisées a donc donné des résultats différents.

XV Conclusion

Nous avons travaillé sur un problème de classification appliqué à des observations météorologiques. L'objectif est de trouver le meilleur modèle possible pour prédire la quantité de pluie du jour suivant $j+1$. Nous disposons de plusieurs variables explicatives, sept d'entre elles sont les données météorologique du jour j et les neuf autres sont des données prédites par le modèle *AROME* de Météo France. Nous améliorons donc ce modèle déterministe à l'aide d'un modèle statistique et nous déterminons quelles variables ont le plus d'influence. Nous avons considéré d'une part un modèle de régression pour lequel nous devons prédire la quantité de pluie, et d'autre part un modèle de classification pour lequel nous devons prédire le niveau d'intensité de la pluie. Nous avons appliqué différents modèles à ces deux problèmes. Nous avons commencé par une ACP, puis des modèles simples, comme le modèle linéaire ou quadratique (avec ou sans sélection de variables par Lasso), le modèle binomial ou la regression logistique, l'analyse discriminante. Nous avons ensuite implémenté des arbres de classification et régression, que nous avons amélioré par agrégation avec les forêts aléatoires et le Boosting. Enfin, nous avons testé les réseaux de neurones et les SVM. Nous avons pu remarquer des résultats cohérents entre plusieurs analyses (en particulier l'ACP, la régression de LASSO et les forêts aléatoires) concernant les variables les plus importantes. Ces variables sont **msl_arome** (pression atmosphérique), **precip_mod** (taux de précipitations du jour j) et les composantes du vent **u10_arome** et **v10_arome**.

Pour R, le meilleur algorithme en régression et en classification est le Boosting. Pour Python, le meilleur algorithme en régression est l'algorithme de réseaux de neurones et en classification c'est le Boosting. Les modèles fournissent une meilleur prédiction que le modèle déterministe *AROME* seul, mais les résultats restent faibles, le meilleur indice de pureté obtenu est de 0.6 (obtenu pour la classification avec le Boosting).

On pourrait penser à des pistes d'amélioration de nos modèles : supprimer certaines variables explicatives obtenues par le modèle de Météo France, pour voir si cela permettrait d'obtenir de meilleures résultats, essayer d'obtenir plus de données de manière à pouvoir mieux apprendre nos modèles, changer les seuils de manière à ne pas à avoir à prédire une valeur ponctuelle.

ANNEXE A

ANNEXES

I Analyse unidimensionnelle

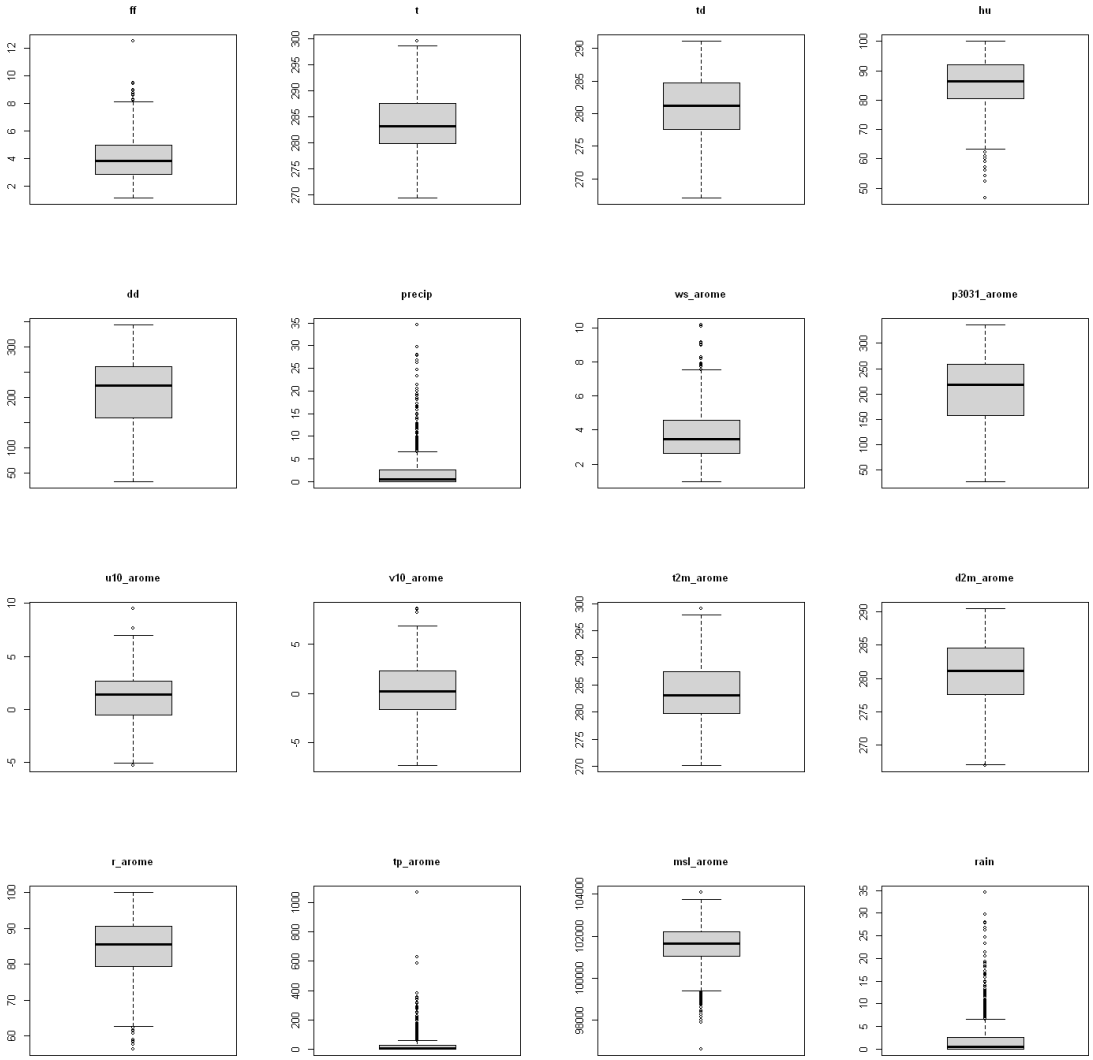


FIGURE A.1 – Boxplots analyse unidimensionnelle

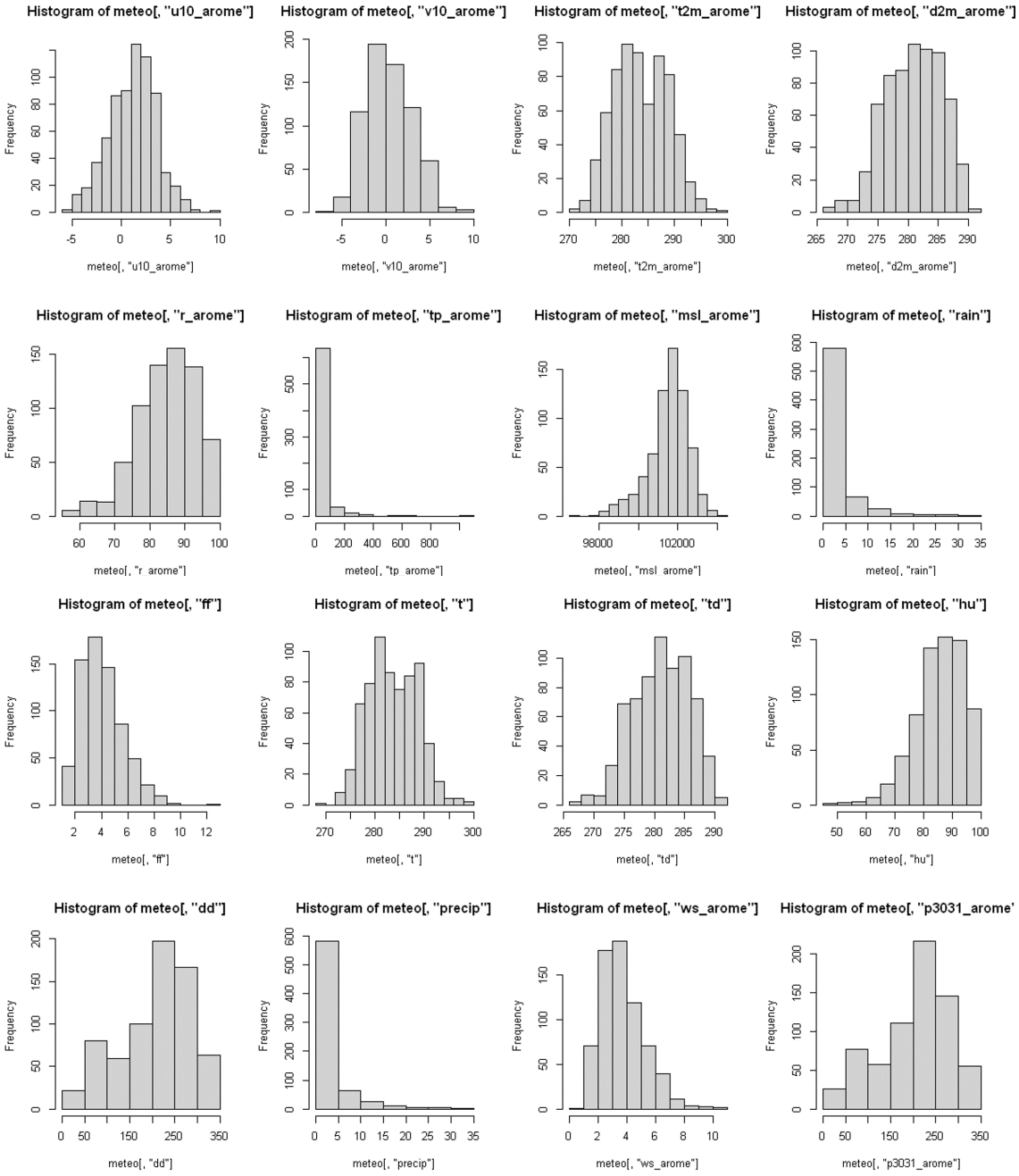


FIGURE A.2 – Histogrammes analyse unidimensionnelle

II CART

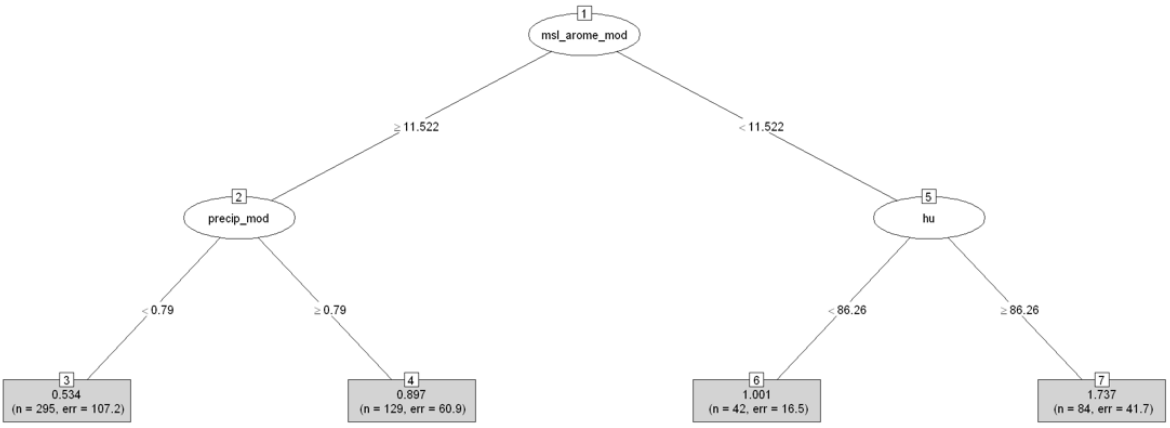


FIGURE A.3 – CART obtenu avec Python en régression

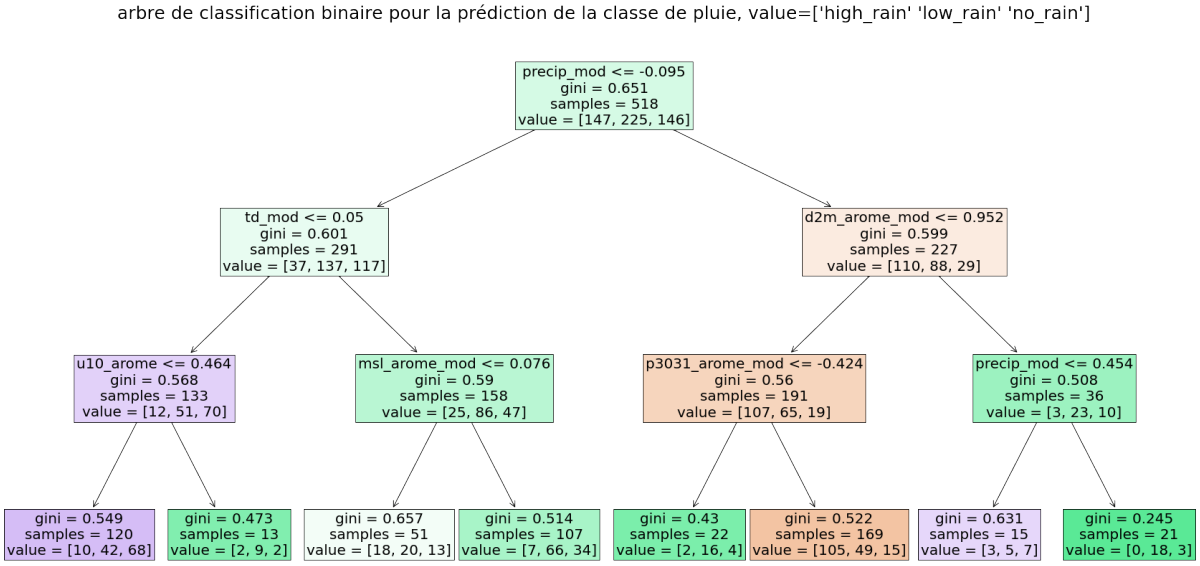


FIGURE A.4 – Arbre de classification pour prédire la classe de précipitations avec Python

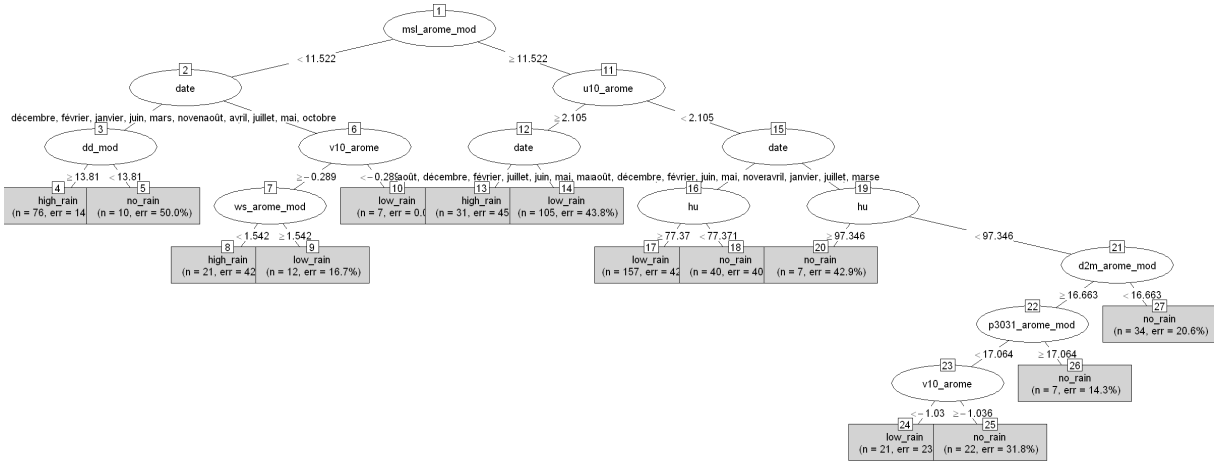


FIGURE A.5 – Arbre de classification pour prédire la classe de précipitations avec R