CORRÉLATION ENTRE TOPONYMIE ET GÉOGRAPHIE DES COMMUNES FRANÇAISES : À LA RECHERCHE DE TRIFOULLIS-LES-OIES (SUJET 9)

2019-2020

Louis Lesueur - Lucas Perrin - Bryan Delamour

Encadré par Robin Ryder

Sujet

Vous ne connaissez sans doute pas les communes françaises de Kersaint-Plabennec, Adamswiller et Bézenac, mais vous ne serez pas surpris d'apprendre que la première est en Bretagne, la seconde en Alsace et la troisième en Aquitaine.

On se propose d'étudier comment un toponyme (nom de lieu) donne de l'information sur la localisation géographique. Les données prennent la forme de 34000 noms de communes de France métropolitaine, avec la longitude, la latitude et le code postal (dont on peut facilement déduire les département et région administrative d'appartenance). Il faudra d'abord compiler ou apprendre automatiquement une liste de propriétés intéressantes (par exemple : longueur du toponyme ; présence d'un "w" ou d'un trait d'union ; dernière lettre...). On appliquera des méthodes classiques et de Machine Learning pour classifier les communes entre régions administratives ou départements. On pourra aussi appliquer des méthodes de régression pour expliquer et prédire la longitude et la latitude.

Dans un second temps, on testera le pouvoir prédictif de cette analyse (validation croisée). On pourra ensuite s'intéresser spécifiquement aux noms des communes nouvelles, et mesurer si les nouveaux noms sont plus typiques, ou plus originaux, que la moyenne. On pourra également utiliser ces résultats pour proposer un emplacement pour quelques communes de fiction, comme Triffouillis-les-Oies, Beauxbâtons, Champignac ou Framboisy.

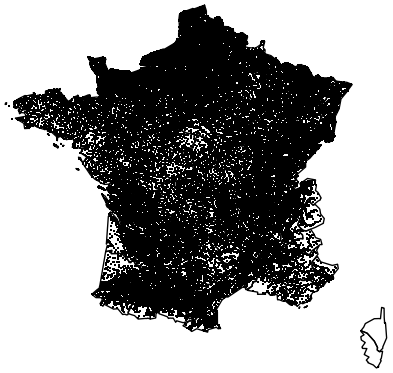
Table des matières

Introduction

Qui n'a jamais entendu parler de *Trifouillis-les-Oies*, *Perpète-les-Oies* ou encore *Trou-la-Ville* ? Ce sont autant de noms fantaisistes censés imiter un nom localité rurale française, sans attrait particulier. Historiquement, on trouve dès 1866 "*Trifouilly-les-Canards*" dans la pièce comique *Nos bonnes Villageoises* de M. A. de Jallais. Il est indéniable que la langue française est riche en jeux de mots sur ces noms de communes imaginées, sensés ironiquement représenter leur région, leur ruralité, ou leur dialecte local.

Nous appelons toponymie la *discipline linguistique qui étudie les toponymes, c'est-à-dire les noms propres désignant un lieu.* Ce mémoire a donc pour sujet l'étude du rapport entre les noms des communes françaises et leur localisation sur le territoire.

La toponymie des communes françaises est particulièrement intéressante à étudier par sa richesse et sa diversité. En effet, par son histoire, la France à vu passer sur ses terres de nombreux peuples tels que les Celtes, Grecs, Latins, Germains ou encore Flamands qui ont, au cours des siècles, marqué leurs territoires et leur régions. C'est aussi l'une des nations comptant le plus de communes à ce jour. Malgré les politiques récentes de réduction du nombre de communes aux échelles européennes et nationales, seul notre pays d'irréductibles gaulois résiste : on compte environ 40% des communes de l'union européennes sur le territoire Français. Passée sous la barre des 35 000 en 2018, Nous restons loin devant notre dauphine, l'Allemagne, qui en compte à ce jour seulement environ 11 000. Voir Figure 1



h !h !

FIGURE 1

Loin de l'idée de critiquer les politiques territoriales, cela rend au contraire le sujet des plus intéressants. Nous allons donc nous intéresser à la base de données datée de 2011 de 36 209 localités françaises, présentée de la manière suivante :

nom	altitude	pays	population	longitude	latitude	surface	departement	region	indicatif

Les données présentées comme ci-dessus seront divisées en deux jeux, d'une manière similaire à chaque fois, entre un jeu de données d'apprentissage, et une base de données de tests. Cela nous permettra donc d'effectuer notre travail sur la base d'apprentissage de taille n, pour ensuite tester nos résultats sur la base de test (de taille n/10). Nous choisissons alors une séparation aléatoire de 10% des données.

Bien que de nombreux travaux sur la toponymie française aient été réalisés depuis 1920, nous avons décidé d'aborder le sujet d'une manière exploratoire. Nous essaierons donc de trouver une procédure

"automatique" pour estimer la région (méthodes de classification) ou les coordonnées (méthodes de régression) des toponymes de communes françaises. Par la suite, il sera intéressant d'analyser les résultats de ces prédictions, et peut être de constater effectivement une certaine répartition des toponymes sur le territoire national en fonction des histoires régionales.

Il faut noter que la problématique mathématique est ici en fait double :

- Nous devons tout d'abord trouver des façons de transformer des chaînes de caractères (toponymes) en données numériques utilisables pour faire de l'apprentissage statistique.
- Nous devons aussi essayer différents modèles explicatifs et les comparer pour trouver le meilleur modèle prédictif.

Finalement, pour suivre la lancée incitée en 1866 par M.A. de Jallais, nous utiliserons les différents modèles crées pour chercher sur la carte de France où se trouve la commune fictive de "Trifouillis-les-oies" afin de constater si oui ou non les différents modèles s'accordent.

Remerciements

Nous tenons à remercier Robin Ryder notre tuteur de mémoire de Master 1, maître de conférence au CEREMADE à Dauphine. M. Ryder nous a proposé le sujet, a supervisé l'implémentation du code et l'interprétation des résultats et a enfin relu le rapport et corrigé les imprécisions. Il a en outre pris le temps tout au long de l'année de discuter avec nous des différents problèmes rencontrés dans le sujet et de répondre à toutes nos questions ; et ce dès que nous en avions besoin. Le soutien que nous a apporté M. Ryder, ainsi que son expertise en mathématiques, nous ont été indispensables pour mener à bien ce projet, et pour cela nous lui sommes infiniment reconnaissants.

Code et Résultats

Nous avons utilisées pour l'implémentation des différentes méthodes les langages *R* ainsi que *Python*. Toues les implémentations ainsi que l'ensemble des résultats sont disponibles sur le GitHub suivant : Codes et sorties du mémoire.

Première partie

Préparation des données

Chapitre 1

Matrice d'occurence des lettres

Dans un premier lieu nous allons traiter une chaîne de caractères comme un vecteur numérique de la façon suivante : pour chacun des caractères (majuscules, lettres accentuées, espaces et - inclus) nous allons comprendre un mot comme une suite de modalités représentant le nombre d'itérations des lettres du mot.

Ainsi 'Paris' s'écrira comme un vecteur (de taille 66-26 lettres de l'alphabet, 26 majuscules, -, _ et 12 caractères accentués) avec 1 aux modalités P,a,r,i,s et 0 ailleurs. La ville de Lille aura 1 aux modalités L, i et e et 2 à la modalité l.

Si ceci est une compréhension extrêmement naïve des chaînes de caractères cela a quand même un sens car certaines lettres comme le k ou le w portent beaucoup d'information sur la localisation de la ville juste par leur présence/ absence.

Dans un second temps, nous ajouterons les chaînes des trois et quatre caractères les plus fréquentes dans le nom des communes de la base d'apprentissage, à savoir les caractères présents dans Tableau 1.1 ; ainsi que le nombre total de lettre dans le nom d'une ville.

Voir Figure 7.3 et Figure 1.1

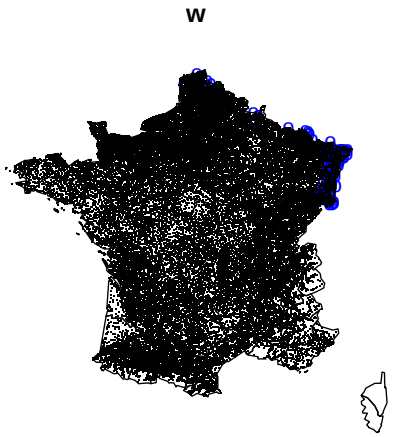


FIGURE 1.1 – Bleu : la commune contient un w

ain sai int ill lle vil les ont sur our mon cha cou ign res mar ier che ère ell sain aint ille vill mont cour ourt elle lles ière ères igny evil ntma ller erre inte illa cham

TABLE 1.1 – Syllabes particulièrement fréquentes

Enfin en regardant les fréquences d'apparition de chaque lettre on remarque que certaines lettres se démarquent par leur rareté c'est le cas de : j,k,w,ç,à,ê,ô,ë,ÿ,î,â,û,ü,Z,Y,U,I,Q,K,W,X.

Nous avons alors, pour chaque commune i, un vecteur associé :

$$X_i^{occ} = (\mathrm{a}_i, \mathrm{b}_i, \dots)$$

donnant ainsi une matrice de taille $n \times 66$, que nous nommerons X^{occ}.

$$X^{occ} = \begin{bmatrix} \mathrm{a}_1 & & \cdots & & \ddot{\mathrm{u}}_1 \\ \vdots & \ddots & & & \vdots \\ \mathrm{a}_i & & \mathrm{G}_i & & \ddot{\mathrm{u}}_i \\ \vdots & & & \ddots & \vdots \\ \mathrm{a}_n & & \cdots & & \ddot{\mathrm{u}}_n \end{bmatrix}$$

Chapitre 2

Distances d'édition

Nous allons ici raisonner d'une manière différente : on va s'intéresser aux distances d'éditions. Regardons une première définition :

Définition 1. *la **distance d'édition** est une distance qui évalue, en termes de nombre de transformations élémentaires, le nombre d'opérations nécessaires et leur coût pour passer d'une chaine de caractères à un autre. On considère 3 transformations élémentaires :*

- *insertion d'un caractère (si $a = sr$, insérer t donne str) ;*
- *suppression d'un caractère (si $a = str$, supprimer t donne sr) ;*
- *substitution d'un caractère (dans str, la substitution $(t \to x), t \neq x$ donne sxr).*

La notion de distance d'édition est souvent confondue avec la **distance de Levenshtein** qui est en fait une distance particulière parmi d'autres. Les principales distances d'éditions sont :

- la **distance de Levenshtein** (autorise les insertions, les suppressions et les substitutions) ;
- la **plus longue sous-séquence commune** (autorise les insertions et les suppressions) ;
- la **distance de Hamming** (autorise uniquement les substitutions, et ne s'applique qu'aux chaines de même longueur) ;
- la **distance de Jaro** (autorise uniquement les transpositions).

En ce qui nous concerne, nous cherchons à établir une distance entre des noms de communes. Nous n'allons donc pas nous intéresser à la distance de Hamming, la taille des noms étant variante. La plus longue sous-séquence étant similaire à la distance de Levenshtein, avec une transformation autorisée en moins, nous allons aussi ne pas l'utiliser. Nous nous attarderons donc à expliquer et utiliser la distance de Lenvenshtein et la distance de Jaro, plus particulièrement une variante de cette dernière : la distance de Jaro-Winkler.

Nous remarquerons que toutes ces distances sont des "distances" au sens mathématique du terme. Cependant, nous allons utiliser plus tard des fonction qui découlent de ces distances qui elles, ne sont pas des distances au sens mathématique du terme.

2.1 Distance de Levenshtein

Voici une première définition de la distance de Lenvenshtein :

Définition 2. *La distance de Levenshtein entre deux chaines de caractères s_1, s_2 (de taille $|s_1|$ et $|s_2|$ respectives) est donnée par* $\text{lev}_{s_1,s_2}(|s_1|,|s_2|)$*, où :*

$$\text{lev}_{s_1,s_2}(i,j) = \begin{cases} \max(i,j) & \text{si } \min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{s_1,s_2}(i-1,j)+1 \\ \text{lev}_{s_1,s_2}(i,j-1)+1 \\ \text{lev}_{s_1,s_2}(i-1,j-1)+\mathbb{1}_{((s_1)_i \neq (s_2)_j)} \end{cases} & \text{autrement.} \end{cases}$$

où $\mathbb{1}_{((s_1)_i \neq (s_2)_j)}$ est la fonction indicatrice et $\text{lev}_{s_1,s_2}(i,j)$ est la distance entre le premier $i$ème caractère de s_1 et le premier $j$ème caractère de s_2.

Nous allons nous intéresser au ratio de similarité, donné lui par :

$$\frac{(|s_1|+|s_2|) - lev_{s_1,s_2}(i,j)}{|s_1|+|s_2|}$$

Il est compris entre 0 et 1, et n'est pas une distance au sens mathématique du terme. Plus ce ratio est proche de 1, plus les chaines de caractères sont similaires.

2.2 Distance de Jaro-Winkler

La distance de Jaro-Winkler est une modification de la distance de Jaro classique, celle ci est donnée par :

Définition 3. *La **distance de Jaro** entre deux chaines de caractères s_1, s_2, est donnée par :*

$$d_{jaro}(s_1,s_2) = \begin{cases} 0 & \text{si } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{sinon.} \end{cases}$$

où :

- *$|s_i|$ est la longueur de la chaine s_i ;*
- *m est le nombre de caractères correspondants ;*
- *t est le nombre de transpositions.*

Nous définissions le nombre de caractères correspondants, en appelant *caractères correspondants*, deux caractères identiques éloingnés d'au plus $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} - 1 \right\rfloor$.

La définition ci-dessus utilise la notion de transposition. Il s'agit de compter le nombre de caractères correspondants différents, et de diviser ce compte par deux.

Winkler introduit un coefficient de préfixe p qui favorise les chaînes commençant par un préfixe de longueur ℓ (avec $\ell \leq 4$). Nous pouvons alors définir la distance de Jaro-Winkler de la manière suivante :

Définition 4. *Deux chaînes s_1 et s_2 ont une distance de Jaro-Winkler d_w définie comme suit :*

$$d_{winkler}(s_1,s_2) = d_{jaro}(s_1,s_2) + (\ell p(1 - d_{jaro}(s_1,s_2)))$$

Nous prendrons un préfixe $p = 0.1$ par la suite.

L'intéret de se servir de cette pseudo-distance, est, en plus de comparer d'une manière différente des chaines de caractères, de donner une importance au début (préfixe) des chaines de caractères. Il sera alors intéressant de constater Nous définirons la création des jeux de données et des matrices qui les concernent en seconde partie.

Deuxième partie

Méthodes et résultats

Chapitre 3

Modèles Linéaires

Dans ce chapitre nous allons créer un modèle linéaire. Nous allons ici utiliser la matrice d'occurence des lettres pour les covaraibles.

Notations On notera dans toute cette section :

- $(longitude_i, latitude_i)$ les coordonnées de la commune i.
- $latitude := (latitude_i)_{i=1\dots n}^T$
- $latitude := (longitude_i)_{i=1\dots n}^T$
- $Y_{\in}\{latitude, longitude\}$
- β le régresseur associé à Y (selon que Y désigne la latitude ou la longitude)
- ϵ le bruit blanc gaussien associé à Y.
- **L** l'ensemble des caractères présents dans les noms
- $X_i = (1, a_i, b_i, \dots)$
- $X = [X_i]_{i=1,\dots,n}$
- $X_{i,\alpha}$ nombre d'apparitions de la lettre $\alpha \in \mathbf{L}$ dans le nom de la commune i.

Nous rappellerons aussi :

- n le nombre de communes de la base d'apprentissage
- n' le nombre de communes dans la base de tests

3.1 Estimation des coordonnées

3.1.1 Modèles réguliers

Informations sur le modèle Nous considérons ici que **la latitude et la longitude d'une commune sont des variables indépendantes** (et peuvent donc être estimées séparément).

Pour toute commune i de la base de données de coordonnées $(longitude_i, latitude_i)$:

$$Y_i = X_i\beta + \epsilon_i$$

Y désigne aussi bien ici la latitude que la longitude ; et β le régresseur associé à Y selon qu'il désigne la latitude ou la longitude.

$X_i = (1, a_i, b_i, \dots)$ déterministe ; β inconnu.

On veut $(C) : \{\beta \in argmin\{||Y - X\beta||\}\}$.

Résultats des tests de Fisher	latitude	longitude
p-value 10^{-16}	a z e r y u i o p s d f g h j k l m v n é è _ ë A Z E U P S F H K L W C V B	a e t y u i q s d f h k m w x b n é – è _ ô ë A Z E R T Y P Q S D G H J L M W X C V B N
p-value<5%	B I J â ü T	l I K o c F
p-value>5%	t q x c – ç à ê ô ÿ î û R Y O Q D G M X N	z r p g j v çà ê ÿ î â û ü U O

TABLE 3.1 – Résultats des tests de Fisher pour la pertinence des variables pour la variabilité de la latitude et de la longitude

Un estimateur de β solution de ce problème d'optimisation (C), unique si X est de rang plein est donné par :

$$\hat{\beta} := (X'X)^{-1}X'Y$$

Pour plus de détails sur ces estimateurs, merci de consulter CORNILLON et al., 2019, chap.1-4 et DONNET, 2019, ch.1.

Tests sur les covariables Nous allons nous servir de ce modèle pour tester l'impact du nombre d'itérations de chaque lettre sur la latitude et la longitude pour voir quelles lettres auront eu le plus d'impact sur les variables d'intérêt.

Pour cela nous allons faire des tests de Fisher : c'est à dire nous allons regarder si la rapport des variances entre le modèle linéaire comprenant toutes les covariables moins celle que nous testons d'une part et d'autre part le modèle linéaire complet.

Pour $\delta \in \mathbf{L}$ on teste :

$$H_0 : Y_i = \beta_0 + \sum_{\alpha \in L, \alpha \neq \delta} \beta_\alpha X_{i,\alpha} \text{ vs } H_1 : Y_i = \beta_0 + \sum_{\alpha \in L} \beta_\alpha X_{i,\alpha}$$

avec :

- β_α régresseur associé à la lettre α :
- $X_{i,\alpha}$ nombre d'apparitions de la lettre alpha dans le nom de la commune i.

Pour plus de détails sur la statistique de test et sur la zone de rejet, consulter DONNET, 2019, section 3.3.4.

Il s'agit sous R de la fonction 'anova'. Les résultats de ces tests de Fisher sont décrits ci-dessous Tableau 3.1 :

- La plupart des lettres ont une p-value <5% (c'est à dire que les tests simples sont extrêmement significatifs) : le plupart des lettres ont un impact statistiquement significatif sur la latitude. Attention néanmoins aux tests multiples : si on fixe le niveau des tests à 5% (tests multiples sans ajustement du niveau des tests) : on arrive à un niveau global des tests de 100%...
- Cependant, il y a beaucoup de p-values à 10^{-16} c'est à dire que beaucoup de lettres semblent avoir un impact extrêmement significatif sur les coordonnées d'une commune. Cela va compliquer la procédure de sélection des variables. En effet, comme nous faisons des tests multiples on veut ajuster le niveau des tests pour éviter de rejeter à tort des hypothèses nulles ; pour cela nous allons appliquer la procédure de Bonferroni-Holmes.

Algorithme 1. ***Bonferroni-Holmes***
(Initialisation) On se donne q tests simples ; on calcule leur p-values et on classe ces tests par ordre croissant en p-values. On se donne de plus un niveau global de confiance α. p=0.
(A l'étape $1 \leq k \leq q$) $p = p + p_k$ où p_k désigne la k-ième p-value des tests simples. Si $p \leq$ alpha on rejette H_0 l'hypothèse nulle du k-ième test. Sinon, on ne rejette pas H_0, ainsi que les hypothèses nulles de tous les tests suivants et on sort de la boucle.

Cependant, les p-values de ces tests êtant de l'ordre 10^{-16} ou supérieurs à 5%, on peut se permettre de rejeter tous les tests de p-values plus petites que 5% et donc de ne pas ajuster le niveau des tests ; et cela sans être en excès de confiance.

Hypothèses du modèle Dans les modèles linéaires nous faisons l'hypothèse que les bruits $\epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2); \forall i \in (1, \ldots, n)$. Il y a en fait 4 hypothèses : l'indépendance des bruits, la normalité des bruits, leur moyenne constante et nulle (on ne néglige pas de tendance) ; et leur variance constante. Nous testerons graphiquement ces hypothèses. Nous accédons au bruit via les résidus :

$$Y = X\beta + \epsilon; \hat{Y} = X\hat{\beta}$$

donc,

$$\epsilon \sim Y - \hat{Y} \text{ avec } Y \in \{latitude, longitude\}; \beta \in \{\beta_1, \beta_2\}$$

Les graphes de diagnostiques sont en annexes : Figure 7.4 pour la latitude, Figure 7.5 pour la longitude.

- *Residuals vs fitted* : Ces graphiques nous permettent de tester si les résidus sont de moyenne nulle. Nous ne sommes pas sensés observer de tendance particulière mais ici, nous voyons bien une tendance (décroissante) pour la longitude comme pour la latitude ; l'espérance des résidus n'est pas nulle.
- *QQ-plot* : Ces graphiques nous permettent de tester la normalité de la latitude et de la longitude. Si ces variables suivaient une loi normale nous devrions observer des droites ce qui n'est pas le cas.
- *Scale Location* : Sous l'hypothèse d'homoscédasticité (variances des résidus constantes) nous ne devrions pas observer de tendance particulière sur ce graphe. Ce n'est pas le cas en particulier pour la longitude.
- *Residuals vs leverage* : Quelques points sont très loin sur la droite. Si aucun point ne semble aberrant il y a l'air d'y avoir un effet de classe sur les covariables. En regardant les points extrêmes on se rend compte que les noms des communes comprennent de nombreux caractères rares : il est possible que ces villes soient seules dans leurs combinaisons de modalités.

Nous observons finalement qu'aucune des hypothèses du modèle linéaire n'est vérifiée. Pour tenter d'améliorer le modèle (le faire se conformer à ces hypothèses) nous avons essayé d'appliquer les fonctions $log(|.|)$ et $\sqrt{(|.|)}$ à la latitude et la longitude sans que cela ait amélioré ces hypothèses (les graphes de diagnostique correspondant sont en annexe Figure 7.6, Figure 7.7). On en déduit que les coordonnées des communes ne dépendent pas linéairement des covariables choisies.

Sélection des variables Il convient maintenant de sélectionner les lettres ayant le plus d'impact sur les coordonnées des communes françaises. Pour cela nous avons déjà fait la procédure de type anova1. Nous allons tester 2 autres procédures : AIC backward et BIC backward. En d'autres termes pour sélectionner les covariables pertinentes nous allons appliquer l'algorithme suivant :

Algorithme 2. ***Sélection des variables (procédure backward)***
(Initialisation) : On part du modèle plein ; comprenant l'ensemble des covariables. On choisit de plus une fonction f d'utilité BIC ou AIC.
(Itération k) : On a sélectionné M_k modèle à k variables et on crée l'ensemble des modèles à k-1 variables ; et on calcule l'utilité de ces modèles : puis on prend le modèle M_{k-1} à k-1 variables ayant maximisé l'utilité. Si $f(M_k) \geq f(M_{k-1})$ on s'arrête et on garde le modèle M_k Sinon on itère avec le modèle M_{k-1}.

$AIC(\xi) := nlog\frac{SCR(\xi)}{n} + n(1 + log2\pi) + 2|\xi + 1|$
$BIC(\xi) := n(1 + log2\pi) + nlog\frac{SCR(\xi)}{n} + |\xi + 1|log(n)$

Score des modèles	Erreur sur la latitude	Erreur sur la longitude
Modèle trivial	0.001461842	0.002055721
Modèle complet	0.001217323	0.001861603
Modèle avec anova type 1	0.001224906	0.001881928
Modèle AIC backward	0.001218195	0.001862060
Modèle BIC backward	**0.001218928**	**0.001864996**
Modele Elastic-net	**0.001217612**	**0.001861504**

TABLE 3.2 – Erreur quadratique moyenne des modèles réguliers

n le nombre de variables.
$|\xi|$ nombre de variables explicatives sélectionnées.
$SCR(\xi) := ||Y - X\xi||$.
Nous avons finalement crée 4 modèles : le modèle linéaire complet ; le modèle après anova1, le modèle après AIC et le modèle après BIC. Leurs performances (sur la base de tests) sont dans Tableau 3.2.
Les performances ont été évaluées en calculant l'erreur quadratique moyenne sur la base de tests ; ie :

$$\frac{1}{n'}\sum_{i=1}^{n'}((longitude_i - \hat{longitude_i})^2, (latitude_i - \hat{latitude_i}^2))$$

On préférera finalement le modèle BIC.

Elastic-net Pour sélectionner des variables et pour éviter le problème des coefficients proches de 0 mais dont les tests de Student nous font rejeter l'hyphothèse de nullité des coefficients nous avons décidé de finalement créer un dernier modèle de type Elastic-net. Il s'agit en fait des mêmes modèles $Y_i = X_i\beta + \epsilon_i$ mais nous rajoutons une contrainte sur β du type :
$\beta \in argmin\{||X\beta - Y||_2^2 + \lambda||\beta||_2 + \mu||\beta||_1\}$ pour$\lambda, \mu > 0$ (le ML simple correspondait au cas $\mu = \lambda = 0$).
Pour choisir efficacement λ et μ nous allons appliquer la procédure de validation croisée. Pour plus de détails, consulter CORNILLON et al., 2019, section 8.3.4.
On voit que les paramètres ont bien convergé (cf Figure 7.10,Figure 7.11) pour toutes les covariables. Lorsque nous calculons l'erreur quadratique de prévision par le modèle Elastic-net et que nous le comparons aux autres erreurs quadratiques nous voyons qu'Elastic-net fait finalement aussi bien que le modèle BIC backward voir Tableau 3.2.

3.1.2 Modèles singuliers

Informations sur le modèle Dans cette partie nous allons rajouter un niveau d'interaction dans le modèle (nous intéresser en plus à la présence/absence d'une seule lettre nous allons aussi regarder la présence simultanée de 2 lettres).
Voici 2 exemples pour montrer de quoi il s'agit en pratique :
Pour Dax cela donne :
$Y_{Dax} = \beta_0 + \beta_D + \beta_a + \beta_x + \beta_{D,a} + \beta_{D,x} + \beta_{a,x} + \epsilon_{Dax}$
Pour Lille cela donne :
$Y_{Lille} = \beta_0 + \beta_L + \beta_i + \beta_{2l} + \beta_e + \beta_{L,i} + \beta_{L,2l} + \beta_{L,e} + \beta_{i,2l} + \beta_{i,e} + \beta_{2l,e} + \epsilon_{Lille}$
Pour plus de détails sur la forme des estimateurs ; consulter DONNET, 2019, chap.3

Hypothèses du modèle et sorties Nous allons vérifier si les hypothèses du modèle (qui sont formellement les mêmes que pour le modèle régulier) sont vérifiées.
Ces graphiques sont aux annexes : Figure 7.8, Figure 7.9.

- *Residuals vs fitted* : la tendance observée dans le cas régulier est encore visible.
- *Normal Q-Q* : Le QQ plot ressemble presque à une droite, le fait de rajouter des interactions a permis d'applatir la droite.

- *Scale-location* : La tendance qui était percevable dans le modèle régulier est nettement moins visible.
- *Residuals vs leverage* : il y a moins d'outliers que dans le modèle régulier.

Les hypothèses du modèle linéaire ne sont toujours pas graphiquement validées ceci dit on est maintenant beaucoup plus tentés de les accepter que dans le cas du modèle singulier. On pourrait même se dire que si on pouvait ajouter autant d'interactions qu'on voudrait sans prendre en compte les temps de calcul on pourrait finir par accepter l'hypothèse selon laquelle les coordonnées d'une ville suivent des lois normales (c'est à dire que chaque ville aurait une combinaison unique de modalités et d'interactions entre les modalités et chaque coordonnée serait alors estimées par elles-mêmes). Nous allons maintenant commenter les sorties du modèle :

- A propos des régresseurs : On remarque qu'il y a beaucoup de régresseurs ps non nuls (c'est à dire avec des tests de nullité avec une p-value au moins $<5\%$) le modèle est donc très gros. La sélection des variables risque aussi d'être compliquée à cause de la taille des modèles.
- A propos des R^2 on obtient les scores 18% pour la longitude et 20% pour la latitude. Ce serait assez faible dans le cadre modèle gaussien mais c'est très intéressant pour notre sujet de remarquer qu'avec des modèles simples et faux on arrive quand même à expliquer environ $1/5$ de la variabilité des positions des villes juste à partir de leur nom.
- A propos de la statistique de Fisher : on obtient une p-value de pertinence du modèle proche de 0 c'est à dire qu'il est fort probable que la présence d'au moins un des caractères étudiés a eu un impact sur la latitude (resp. longitude) des villes en France : c'est très encourageant !
- A propos des NA (not a number) :Nous observons dans les sorties des tests sur le modèle un certain nombre de NA. Ils viennent sans doute du fait que nous avons considéré des interactions qui n'existent pas. Il est possible qu'aucune ville n'ait à la fois les lettres a et ü dans leur nom (les NA sont les mêmes pour la latitude et la longitude). Il serait pertinent de ne pas regarder alors ces interactions... Une autre hypothèse moins plausible serait que X la matrice d'apprentissage ne soit pas de rang plein.

Sélection des variables Pour la sélection des variables avec 1 niveau d'interactions on a plus de 2000 coefficients à tester il n'y a donc pas moyen de faire BIC ou AIC, même avec une procédure car les temps de calcul deviennent énormes. En effet avec une procédure forward (par exemple) : à l'itération 1000 nous devrions créer 1000 modèles à 1001 coefficients... De plus comme beaucoup (plus de 200) de ces coefficients sont avec une forte probabilité non nuls il y a de fortes chances pour que nous ayons effectivement à considérer ces modèles. Pour la sélection des variables nous allons utiliser encore une fois une procédure de type anova1.

Nous allons donc tester l'influence de chacune des covariables ainsi que leurs interactions sur sur la variance des coordonnées. Pour visualiser ces tests nous allons garder l'exemple de la ville de Dax.

Pour la 1ère série de tests il s'agit des tests de Fisher ou nous testons la pertinence du modèle sans inclure une covariable (ici a par exemple a) contre le modèle complet.
$H_0 : Y_{Dax} = \beta_0 + \beta_D + \beta_x + \beta_{D,x} + \epsilon_{Dax}$ vs $H_1 : Y_{Dax} = \beta_0 + \beta_D + \beta_a + \beta_x + \beta_{D,a} + \beta_{D,x} + \beta_{a,x} + \epsilon_{Dax}$.

En d'autres termes on fait un rapport de variances entre le modèle moins la variable (ici a)que l'on cherche à tester et ses interactions avec les autres variables ; et le modèle complet.

Pour la seconde série (si on a rejeté H_0 dans le 1er test) ; on s'intéressera à la pertinence de ses interactions avec les autre variables (pour Dax mettons qu'on garde le a à l'issue du 1er test et que nous nous interrogeons sur la pertinence de l'interaction a :x) on testera :
$H_0 : Y_{Dax} = \beta_0 + \beta_D + \beta_a + \beta_x + \beta_{D,a} + \beta_{D,x} + \epsilon_{Dax}$ vs $H_1 : Y_{Dax} = \beta_0 + \beta_D + \beta_a + \beta_x + \beta_{D,a} + \beta_{D,x} + \beta_{a,x} + \epsilon_{Dax}$

Score des modèles	Erreur sur la latitude	Erreur sur la longitude
Modèle trivial	0.001461842	0.002055721
Modèle singulier	0.001193490	0.001934674
Modèle singulier anova de type1	**0.001136769**	**0.001788715**
Modèle régulier BIC backward	0.001218928	0.001864996
Modele Elastic-net	0.001217612	0.001861504

TABLE 3.3 – Erreur quadratique moyenne des modèles singuliers

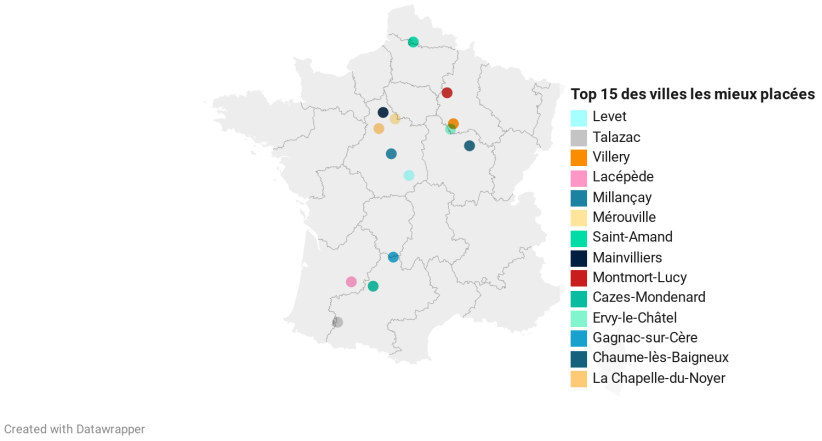


FIGURE 3.1 – Top 15 des villes les mieux placées par le modèle singulier après anova.

La statistique de tests, ainsi que l'ensemble de la procédure anova 1 sont décrits dans la section 'tests statistiques des différents effets' du chap.3 de DONNET, 2019

Nous fixons le niveau de risque à 5% et nous réduisons ainsi les modèles à environ 400 paramètres pour chacune des coordonnées.

3.1.3 Conclusions

Performances des modèles Nous avons crée nos modèles en utilisant une base d'apprentissage comprenant 90% de la base de données. Nous nous servons maintenant des 10% des communes restantes rassemblées dans une base de tests pour calculer l'erreur (quadratique) d'apprentissage. Pour chacun des modèles nous calculons :

$$\frac{1}{n'}\sum_{i=1}^{n'}((longitude_i - \hat{longitude}_i)^2, (latitude_i - \hat{latitude}_i^{\,2}))$$

Et nous regardons le modèle minimisant la moyenne des erreurs quadratiques pour voir lequel de nos modèles prédit le mieux.

Le meilleur modèle que nous ayons crée est **le modèle singulier après sélection des variables via l'anova de type 1**,d'après Tableau 3.3.

Sélection des individus Il est maintenant intéressant de voir pour le meilleur modèle quelles villes ont été les mieux placées. Le top 15 de ces villes est dans Figure 3.1.

Nous observons que pour l'ensemble des villes les mieux placées, peu importe le critère ; il n'y a pas spécialement de points communs entre ces villes. Elles ne sont pas dans la même région ni proches du centre. En revanche leurs noms rappellent le nom des villes ayant l'air d'avoir eu un fort leverage :

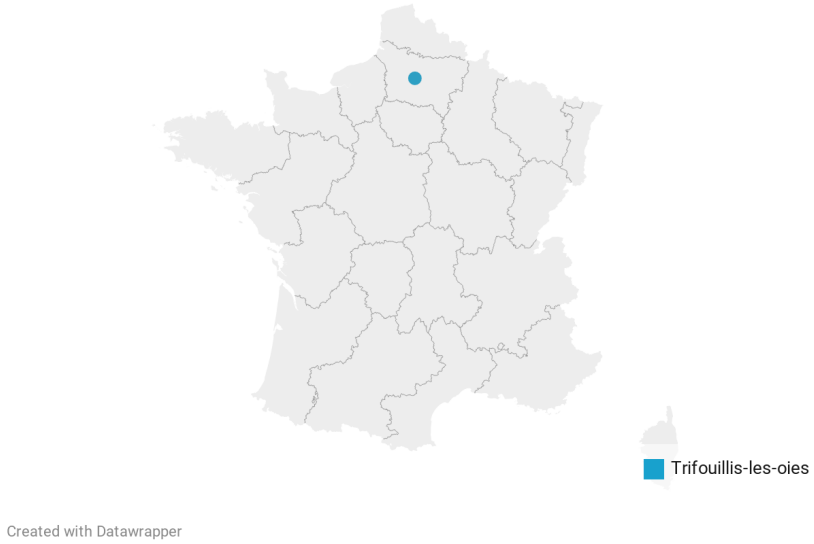


FIGURE 3.2 – Coordonnées estimées pour Trifouillis les oies via ANOVA

leurs coordonnées sont sans doutes estimées par elles-mêmes. Ces villes ont peut-être une combinaison de modalités unique.

Trifouillis-les-oies La question naturelle est de placer Trifouillis-les-oies avec notre modèle linéaire optimal Figure 3.2

3.2 Modèles linéaires généralisés et classification

3.2.1 Régressions logistiques binaires

Notre première partie (Estimation des coordonnées) nous a montré que la toponymie porte de l'information sur la localisation des villes, en revanche jusqu'ici cette information ne semble pas assez précise pour pouvoir situer les communes avec une grande précision. Notre jeu de données, en plus des longitudes et latitudes de chaque ville, comporte également leur région, il nous a semblé intéressant de vouloir expliquer la variable région à l'aide d'un modèle statistique. Notre approche de cette nouvelle problématique consistera pour une région donnée, à modéliser l'appartenance ou non d'une ville à cette région en fonction de sa toponymie.

Informations sur le modèle Nous créons un modèle de régression logistique binaire pour chaque région. Notre variable d'intérêt est désormais R_i une variable qualitative et ne peut prendre que deux valeurs 0 (la ville i n'appartient pas à la région) ou 1 (la ville i appartient à la région). Les variables explicatives sont celles de la première partie, c'est-à-dire, le nombre d'itérations des lettres dans le nom des villes en tant que vecteurs, auxquelles nous avons ajouté les 20 plus fréquentes chaînes de trois et quatre caractères dans les noms des communes de la base d'apprentissage, ainsi que le nombre total de lettre dans le nom de la ville.

Le modèle logistique considère une modélisation de la variable aléatoire $R_i \mid X_i = x_i$, de loi de Bernoulli $p_\beta(x_i) = \mathbf{P}(R_i = 1 \mid X_i = x_i)$.

L'hypothèse du modèle est $\log \frac{p_\beta(x_i)}{1-p_\beta(x_i)} = X_i.\beta$ où le paramètre β est estimé par maximum de vraisemblance.

Évaluation du modèle Pour évaluer la qualité du modèle il nous faut définir une règle d'affectation, effectuer un matrice de confusion et mesurer l'erreur de prédiction.

a z e r t y u i o p q s d f g h j k l m w x c v b n é è ç à ê ô ë î â û ü A Z E R T Y U I O P Q S D F G H J K L M W X C V B N ain sai int ill lle vil les ont sur our mon cha cou ign res mar ier che ère ell sain aint ille vill mont cour ourt elle lles ière ères igny evil ntma ller erre inte illa cham nbChar

TABLE 3.4 – Variables des régressions logistiques binaires

Une approche naïve aurait été de définir la règle d'affectation $t = 0,5$. Ainsi, si la prédiction $\hat{p}_\beta(x_i)$, est telle que $\hat{p}_\beta(x_i) > 0,5$, le modèle place la ville i dans la région. Et de définir l'erreur comme la somme des faux positifs et faux négatifs sur l'effectif total. Dans notre cas, cette méthode aurait été grandement inefficace. En effet, pour de nombreuses régions, cette règle d'affection n'a classé aucunes communes dans la région. Et étant donné que le nombre de villes dans une région est faible comparé à l'effectif total, nous aurions eu un nombre de faux négatifs très faible et aucun faux positif, donc une erreur très faible.

Pour cela, nous avons décidé de définir l'erreur de prédiction avec les coefficients $1/20$ pour les faux positifs et $19/20$ pour les faux négatifs. Ces coefficients ont été choisi arbitrairement mais ne semblent pas déraisonnables, dans le sens où nous avons 21 régions. La règle d'affectation t^* (à l'aide des données d'apprentissage), et le taux de succès sont définis :

$$Erreur(t) = (\frac{19}{20}.FauxPositif + \frac{1}{20}.FauxNegatif)/EffectifTotal$$

$$t^* = argmin_{t \in T} Erreur(t)$$
$$T = \{0.01, 0.02, ..., 0.99\}$$

$$Succes(t) = (VraiPositif + VraiNegatif)/EffectifTotal$$

Sélection des variables Nos modèles pleins possèdent 103 covariables,contenues dans le tableau Tableau 3.4 ; nbChar désignant le nombre de lettres dans un toponyme.

Nous remarquons que certaines covariables sont fortement corrélées entre elles, notamment "ain", "sai" et "int". Une procédure AIC, avec autant de covariables, n'a pas été faisable avec nos ordinateurs personnels, ainsi, nous avons opté pour simplement supprimer les covariables n'étant pas significatives au regard des p-values. Chaque région ayant son propre modèle, la sélection des variables sera donc différente d'une région à l'autre et donc les modèles finaux auront des covariables distinctes.

Résultats Nous avons, pour chacune des 21 régions du jeu de données, un modèle. La base d'apprentissage et la base test sont identique à la partie précédente. L'erreur moyenne des 21 modèles des prédictions sur la base de tests est de 3,03%, le succès moyen de 68,36% ; le succès et l'erreur aux sens définis ci-dessus. Les résultats sont représentés par les deux cartes ci-dessous.

Les modèles ayant les meilleurs performances, en termes d'erreur et de succès, sont ceux des régions Alsace (AL), Limousin (LI), et Provence-Alpes-Côte d'Azur (PA). Les régions Rhône-Alpes (RA) et Midi-Pyrénées (MP) ont eu des performances intéressantes, dans le sens où les modèles ont placés 93,75% et 86,71% des villes de leur régions correctement. Cependant, nous remarquons que pour ces deux régions le nombre de faux positif est bien plus important, plus 50% des villes hors région ont été placées à tort dans la région.

- *Alsace(AL)* : le taux d'erreur est de 0,56% et succès de 95,69%. Dans notre base test, parmi les 88 villes présentes en Alsace, 74 ont été correctement placées dans la région. Parmi les 3532 villes hors de la région, 3390 ont bien été placées en dehors de la région.

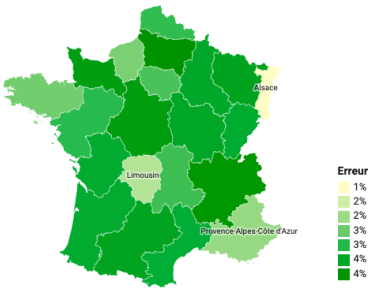


FIGURE 3.3 – Erreur - régression logistique

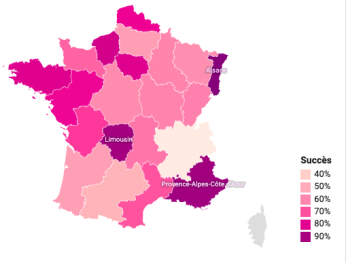


FIGURE 3.4 – Succès - régression logistique

AL	Négatif	Positif
Négatif	3390	14
Positif	142	74

p-value	Odd ratio>1	Odd ratio<1	Total
<0,1%	r p n l s g z d t H m f W b K k ller h w	C ont che A é y	25
<1%	x Z	cha è nbChar T	6
<5%	u U V	e i ier P q res sur	10
Total	24	17	41

TABLE 3.5 – Alsace - Test de Wald et odd ratio - régression logistique

Le modèle Alsace possède 41 variables significatives (Table 3.4), qui composent donc le modèle final pour cette région. La majorité d'entre elles ont une p-value strictement inférieure à 0,1%. Les odd ratios nous permettent de comprendre l'impact de ces variables sur les chances pour une ville d'appartenir à la région. Notamment, la lettre "w" possède un odd ratio de 105, ainsi les villes avec la lettre "w" ont plus de chance de se situer en Alsace selon notre modèle. De même, pour "ller" et "h" qui ont des odd ratios de 25. À l'inverse, "C", "ont" et "che" ont des odd ratios inférieurs à 0,1. Ainsi, la présence de ces caractères diminue les chances d'appartenir à la région pour notre modèle.

- *Limousin (LI)* : le taux d'erreur est de 1,77% et succès de 90,86%. Parmi les 75 villes présentes dans le Limousin, 22 ont été correctement placées dans la région. Parmi les 3545 villes hors de la région, 3267 ont bien été placées en dehors de la région.

LI	Négatif	Positif
Négatif	3267	53
Positif	278	22

TABLE 3.6 – Limousin - Matrice de confusion - régression logistique

p-value	Odd ratio>1	Odd ratio<1	Total
<0,1%	r t p a g z c les â x	lles nbChar	12
<1%	s o i l D J N Y	ont	9
<5%	e d f L our M C S sur B P à	ères	13
Total	30	4	34

TABLE 3.7 – Limousin - Test de Wald et odd ratio - régression logistique

- *Provence-Alpes-Côte d'Azur (PA)* : le taux d'erreur est de 2,07% et succès de 91,33%. Parmi les 88 villes présentes dans le Limousin, 22 ont été correctement placées dans la région. Parmi les 3532 villes hors de la région, 3284 ont bien été placées en dehors de la région.

PA	Négatif	Positif
Négatif	3284	66
Positif	248	22

TABLE 3.8 – Provence-Alpes-Côte d'Azur - Matrice de confusion - régression logistique

p-value	Odd ratio>1	Odd ratio<1	Total
<0,1%	a è C les V A R E â	y h	11
<1%	S M P T	ère ain lles z	8
<5%	u e o ier L che J illa ères	elle	10
Total	22	7	29

TABLE 3.9 – Provence-Alpes-Côte d'Azur - Test de Wald et odd ratio - régression logistique

Modèle multi-régions Les modèles précédemment créés étant indépendants les uns des autres, il y a donc de fortes chances que certaines communes furent placées dans plusieurs régions à la fois. Ainsi, nous avons créé un dernier modèle réunissant l'ensemble des modèles par région, qui assigne à une commune une unique région.

Désignons $\hat{p}_i^j$ la prédiction pour la commune i du modèle de la région j. La prédiction de notre modèle multi-région pour la commune i, $\hat{p}_i$, est défini : $\hat{p}_i = max_j(\hat{p}_i^j)$.

Résultats Le modèle a correctement placé 19,5% des communes de la base test. Notamment, il a placé correctement 72,73% des villes d'Alsace, et 51,83% des villes de Midi-Pyrénées. Pour une carte des performances par région regarder Figure 3.5

Triffouillis-les-Oies Par nos modèles indépendants, notre ville fictive Triffouillis-les-Oies a été placée dans 6 régions : Rhône-Alpes, Champagne-Ardenne, Picardie, Basse-Normandie, Île-de-France, et Pays de la Loire. Le modèle multi-région la place en Pays de la Loire.

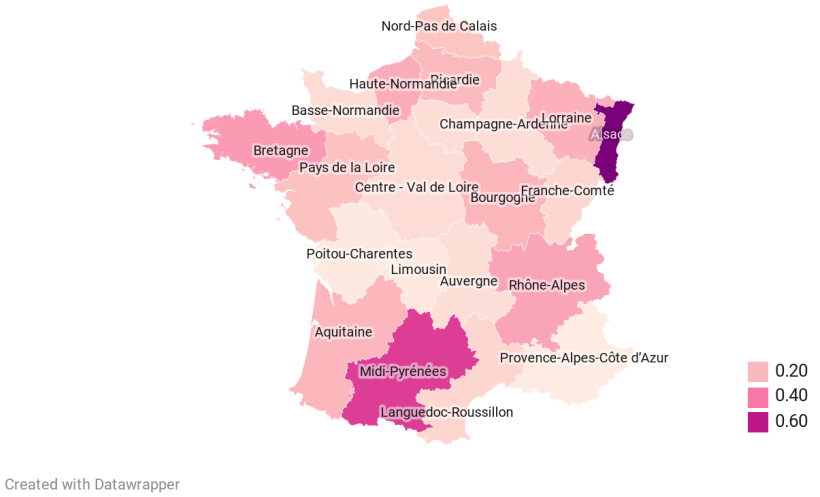


FIGURE 3.5 – Scores de prédiction par région par le modèle multi-régions

Région	Règle d'affectation t*	Prédiction $\hat{p}_\beta(x_i)$	Crédence
Limousin	0,05	0,0000	0,00%
Auvergne	0,04	0,0041	0,48%
Aquitaine	0,05	0,0047	0,55%
Franche-Comté	0,05	0,0068	0,80%
Midi-Pyrénées	0,05	0,0090	1,06%
Languedoc-Roussillon	0,05	0,0129	1,53%
Bretagne	0,05	0,0131	1,55%
Poitou-Charentes	0,05	0,0162	1,91%
Provence-Alpes	0,06	0,0187	2,21%
Lorraine	0,05	0,0259	3,06%
Alsace	0,07	0,0335	3,96%
Centre	0,05	0,0347	4,09%
Bourgogne	0,05	0,0369	4,36%
Haute-Normandie	0,05	0,0403	4,76%
Nord-Pas-de-Calais	0,05	0,0406	4,80%
Rhône-Alpes	0,05	0,0510	6,02%
Champagne-Ardenne	0,05	0,0562	6,63%
Picardie	0,05	0,0834	9,85%
Basse-Normandie	0,05	0,0914	10,79%
Île-de-France	0,05	0,1263	14,92%
Pays de la Loire	0,05	0,1410	16,66%

TABLE 3.10 – Estimation - Triffouillis-les-Oies - régression logistique

Ici la crédence est définie comme :

$$Credence_i := \frac{\hat{p}_\beta(x_i)}{\sum_{i,region} \hat{p}_\beta(x_i)}$$

3.2.2 Régression logistique multinomiale

Nous allons ici regarder les résultats sur un modèle de régression logistique multinomiale. Il s'agit d'une généralisation à plusieurs classes du modèle de régression logistique binaire. La loi d'intérêt n'étant

pas une loi binomiale mais la loi multinomiale. Cela fonctionne comme la création du modèle précédent, avec une logique *"one versus all"*, mais l'algorithme implémenté dans *R* est plus performant, et nous donne une sortie de type *softmax* avec donc les probabilités d'appartenance à chaque classe. La théorie ayant été expliqué précedemment, nous regarderons donc uniquement les sorties de ce modèle implémenté sur *R*.

On notera qu'après plusieurs tentatives, le step downward de cette régression multinomiale n'a pas pu être effectué car il prenait trop de mémoire lors des différents essais de calcul.

Nous obtenons un score global de 17.96%, avec les résultats par régions suivants :

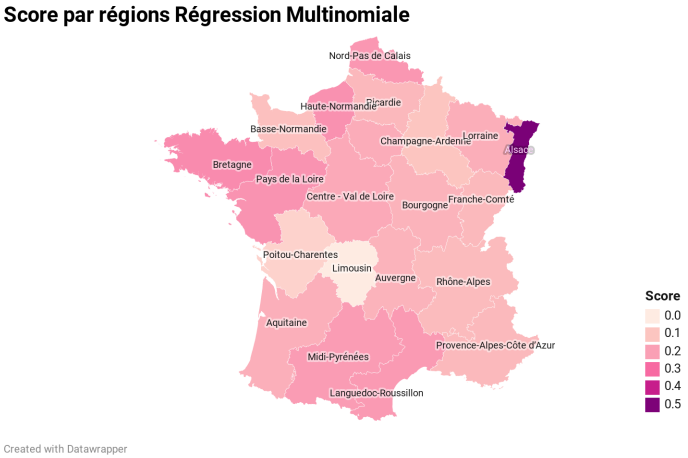


FIGURE 3.6 – Résultats de la régression multinomiale.

Placement de *Trifouillis-Les-Oies*

Par ce modèle, la commune de *Trifouillis-Les-Oies* se trouve en Basse Normandie.

Chapitre 4

k-NN modifié

Nous cherchons dans ce cas à prédire la région avec un implémentation personnalisée d'un algorithme de type k-NN. On utilise comme base de données d'apprentissage les différentes pseudo-distances d'éditions présentées dans la première partie. Nous ne nous intéresserons donc pas aux données sous forme de table d'occurence pour ce modèle. L'algorithme de classification que nous proposons est le suivant :

Algorithme 3. ***k-nn modifié***
(Initialisation) : On se donne d une distance ou un ratio de similarité entre 2 toponymes ; C, un ensemble de toponymes de références (ie la base d'apprentissage).
(Pour chaque commune de la base de tests c_i) : On calcule $d(c_i, c_j), \forall j \in C$ puis on prend les n communes les plus proches au sens de d ; et enfin on estime la région de c_i comme la région majoritaire au sein de ces n communes. En cas d'égalité on choisit arbitrairement la 1ère région réalisant le maximum d'occurences.
(Finalement) Pour conclure on calcule le score de classification en comparant les régions estimées des communes de la base de tests et leurs vraies régions.

Nous allons alors, pour les deux pseudo-distances (ratio de Levenshtein et distance de Jaro-Winkler) observer la précision pour différents k, puis nous nous attarderons alors sur le détail de la meilleur prédiction, région par région. Cela permettra aussi de comparer les deux pseudo-distances choisies dans un algorithme qui les utilise directement.

4.1 Avec distance de Levenshtein

En utilisant le ratio de Levenshtein comme pseudo-distance entre les noms de communes, nous obtenons les résultats suivants :

k	précision	temps d'exécution (secondes)
5 nearest-neighbours	24.36 %	380.95
10 nearest-neighbours	25.03 %	380.72
20 nearest-neighbours	**25.72 %**	**380.93**
50 nearest-neighbours	25.24 %	382.76
100 nearest-neighbours	24.50 %	383.32

La précision est donc maximale pour 20 plus proches voisins. Ce nombre semble totalement raisonnable et logique. En effet, un trop petit nombre de voisins ne permettrait pas une bonne précision, et un trop grand nombre amènerait à un "bruit" trop grand.

Cependant, on remarquera que le score est très similaire pour les différents k, et que le temps d'exécution n'est que très peu affecté par l'augmentation de k : même un facteur de 10 ne change pas grand chose.

4.2 Avec pseudo-distance de Jaro-Winkler

En utilisant cette fois la pseudo-distance de Jaro-Winkler (on rappelle $p = 0.1$ ici), on obtient les résultats suivants :

k	précision	temps d'exécution (secondes)
10 nearest-neighbours	21.51 %	64.94
20 nearest-neighbours	22.04 %	64.88
50 nearest-neighbours	**22.79 %**	**66.04**
100 nearest-neighbours	21.40 %	67.95

La précision est donc maximale pour 50 plus porches voisins, et nous pouvons tout à fait appliquer les remarques faites pour la pseudo-distance déterminée par le ratio de Levenshtein à celle de Jaro-Winkler.

Nous ferons tout de même la réflexion suivante : la précision est moindre pour cette pseudo-distance, mais nous y reviendront plus tard.

4.3 Résultats

Nous allons maintenant nous intéresser aux résultats régions par région pour un k optimal dans les deux cas. Nous obtenons les résultats suivants (visible à la Figure 7.4) :

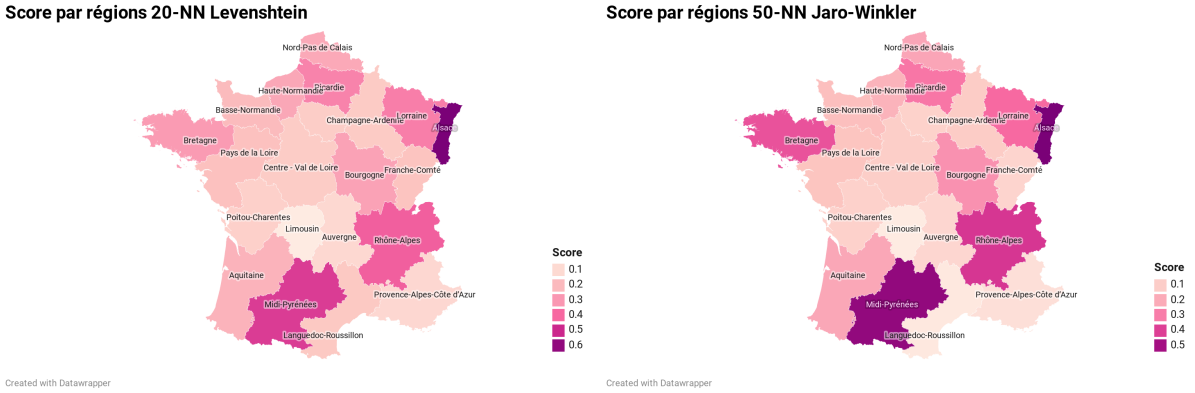


FIGURE 4.1 – Score par régions pour la méthode k-NN avec les deux pseudo-mesures

Nous remarquons tout d'abord bien la supériorité pour l'algorithme avec le ratio de Levenshtein par rapport à la pseudo-distance de Jaro-Winkler. Ensuite, le détail par région permet de donner une indication des régions possédant des noms de communes très spécifique à la zone géographique dans laquelle elles se situent.

Nous pouvons d'abord considérer les régions Alsace, Lorraine, Midi-Pyrénnées, Rhône-Alpes et Bretagne. Le score y est assez élevé dépassant même les 55% pour la région Alsace. Il est naturel de se dire que ces régions sont les régions où le nom de communes y est le plus significatif. Ce sont en effet des régions marquées durant l'histoire par un dialecte particulier.

D'autres régions, telles que le Limousin ou Provence-Alpes Côte d'Azur possèdent un score très faible. Cela est peu surprenant. En effet, ce sont des régions où les noms de communes sont variés et qui recouvrent une partie assez importante du territoire.

Nous pouvons surtout effectuer ici une comparaison entre les différentes distances. En effet, comme expliqué en première partie la pseudo-distance exprimée avec le ration de Levenshtein nous donne une similarité assez "souple" entre les différentes chaines de caractères. Celle de Jaro-Winkler est non seulement plus "rigide", mais elle favorise les similarités entre les entrées commençant de la même manière.

On considèrera donc, à partir de maintenant, uniquement le ratio de Levenshtein pour calculer la pseudo-distance entre deux noms de communes.

Note : Certaines méthodes qui suivent ont étés aussi réalisées avec la pseudo-distance de Jaro-Winkler, mais ont, dans tous les cas, corroboré l'observation faite ci-dessus : les résultats sont moins bons qu'avec le ratio de Levenshtein. Il est possible néanmoins d'observer ces résultats dans le code associé à ce mémoire.

4.4 Placement de Trifouillis-Les-Oies

Nous regardons alors où cette méthode place la communes de *Trifouilllis-Les-Oies*. On trouve la Basse Normandie pour la pseudo-distance avec le ratio de Levesnhtein, et la Haute Normandie pour la pseudo-distance de Jaro-Winkler (voir : Figure 4.2).

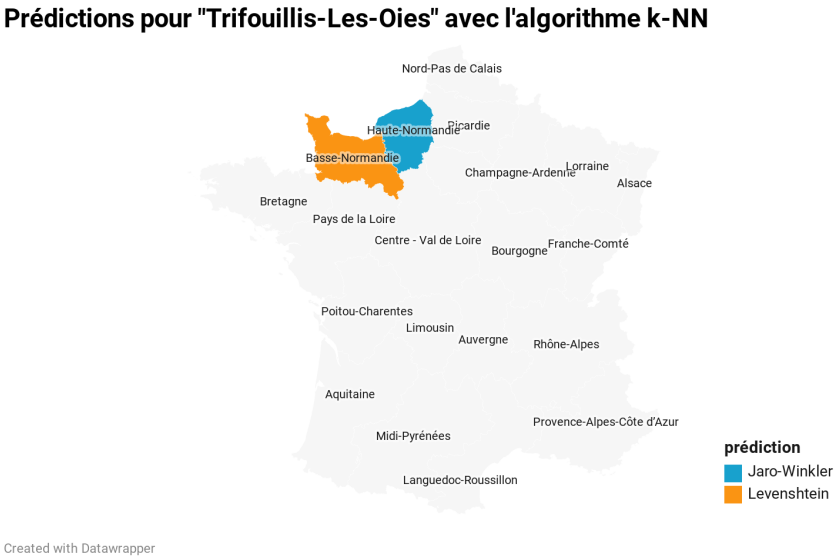


FIGURE 4.2 – *Prédictions pour Trifouillis-les-Oies*

Chapitre 5

Forêts Aléatoires

Dans ce chapitre, nous allons nous intéresser aux forêts aléatoires (Random Forest en anglais). On va tout d'abord expliquer la construction d'une foret aléatoire, puis tester cette méthode dans un premier lieu pour de la régression, puis pour de la classification. Nous utiliserons différent jeux de données :

- pour une **régression**, nous utiliserons les données présentées sous forme de matrice d'occurence des lettres ;
- pour une **classification**, nous regarderons les résultats avec la matrice d'occurence des lettres, ainsi que différentes manipulations des distances d'éditions (comme annoncé précédemment, nous ne continuons qu'avec la pseudo-distance formulée par le ratio de Levenshtein).

5.1 Construction de forêts aléatoires

5.1.1 Notations

- Les données sont des couples (X_i, Y_i) pour $i \in \{1, \dots, n\}$; On note $D_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$
- On suppose $\forall i \in \{1, \dots, n\}$, $X_i \in \mathbf{R}^p$ l'ensemble des covariables.
- $Y_1, \dots, Y_n$ Variables d'intérêt. Dans le cadre de la classification (ie Y_i à valeurs discrètes) ; on notera $\mathbf{Y}$ son ensemble de valeurs et $K := card(\mathbf{Y})$
- On notera (R) le problème d'optimisation de la variance inter-classes.

$$argmin_{j,d_j} \left\{ \sum_{i, X_{i,j} \leq d_j} (Y_i - Y_l)^2 + \sum_{i, X_{i,j} > d_j} (Y_i - Y_r)^2 \right\}$$

avec :

$$Y_l := \frac{1}{card(i : X_{i,j} \leq d_j)} \sum_{i, X_{i,j} \leq d_j} Y_i$$

$$Y_r := \frac{1}{card(i : X_{i,j} > d_j)} \sum_{i, X_{i,j} > d_j} Y_i$$

- On notera (C) le problème d'optimisation du critère de GINI :

$$argmin_{j,d_j} card\{i, X_{i,j} \leq d_j\} \sum_{k=1}^{K} \hat{p}^l_{j,k}(1 - \hat{p}^l_{j,k}) + card\{i, X_{i,j} > d_j\} \sum_{k=1}^{K} \hat{p}^r_{j,k}(1 - \hat{p}^r_{j,k})$$

avec :

$$\hat{p}^l_{j,k} := \frac{card\{i, X_{i,j} \leq d_j, Y_i = k\}}{card\{i, X_{i,j} \leq d_j\}}$$

$$\hat{p}^r_{j,k} := \frac{card\{i, X_{i,j} > d_j, Y_i = k\}}{card\{i, X_{i,j} > d_j\}}$$

Nous nous proposons de résumer d'abord l'algorithme de création d'arbre de décision de type CART, puis d'une forêt aléatoire de CART. Ceci est un résumé de Roche, 2020, dont le cours est inspiré de Genuer et Poggi, 2017.

5.1.2 Construction d'un Classification and regression tree (CART) :

Algorithme 4. ***CART***
Etape 1*(Initialisation de l'arbre) : On se donne un nombre de feuilles ou une profondeur maximale et on construit T_{max} l'arbre binaire correspondant.*

Etape 2 *(Partition de la racine) : On veut se servir de l'arbre T_{max} pour faire une partition de D_n.*

- *On agrège le nœud par la moyenne des Y_i pour la régression et la classe majoritaire pour la classification.*
- *On va associer à ce nœud un test de la forme $\mathbf{1}_{X_{i,j} \leq d_j}$ Il faut trouver j, d_j solution de : (R) pour la régression, solution de (C) pour la classification.*
- *On fait ce test pour D_n et on obtient 2 sous ensembles l et d. Il suffit d'appliquer récursivement cette même étape avec comme racines l'enfant à gauche et l'enfant à droite de la racine avec comme sous-échantillons d'apprentissage respectivement l et r ; et ce jusqu'à ce que le nœud courant n'ait pas d'enfants.*

Voici un exemple de CART : Figure 7.12

5.1.3 Construction de la forêt aléatoire

On se fixe n_{trees} un nombre d'arbres.

Pour $b \in \{1, \ldots, n_{trees}\}$ On de tire uniformément avec remise M échantillons parmi D_n ; souvent $M = n/3$. On construit ensuite T_b le CART engendré par cet échantillon.

On a ainsi construit la forêt $\{T_1, \ldots, T_{n_{trees}}\}$

Pour estimer Y_{n+1} à partir de X_{n+1} et de la forêt : pour chaque arbre $b \in \{1, \ldots, n_{trees}\}$ on obtient $\hat{Y}^b$ prenant l'agrégation du nœud terminal de l'arbre (celui auquel on aboutit après la série de tests sur X_{n+1}) ; puis dans le cadre de la régression $\hat{Y}_{n+1}$ est la moyenne des $\hat{Y}^b$, dans le cadre de la classification $\hat{Y}_{n+1}$ est la classe majoritaire parmi les $\hat{Y}^b$.

5.2 Régression

Nous avons réalisé deux forêts aléatoires, une pour la longitude et une pour la latitude. Chacune des forêts comportent 500 arbres et le nombre de variables tirées aléatoirement pour sélectionner la variable de segmentation de chaque noeud est de 35. Les covariables des modèles sont identiques à celles de la régression logistique. L'erreur moyenne de prédiction pour la latitude est de 0,0010 et 0,0017 pour la longitude, avec un R^2 de 17,69% et 28,41% respectivement. La longitude est plus difficilement prévisible avec une erreur 1,7 fois supérieure à celle de la latitude. Les forêts aléatoires ont une performance proche de celle des modèles linéaires des parties précédentes. En revanche il semblerait que les modèles agissent différemment au regard des covariables.
Nous remarquons que pour les deux modèles le nombre de lettres dans les noms des communes est la variable avec la plus forte importance, c'est-à-dire, celle qui diminue le mieux l'erreur de prédiction. Cela pourrait expliquer le fait que les villes les mieux prédites par les forêts aléatoires ont un nom plus long que le top 15 des modèles linéaires.

Nous remarquons également que les villes les mieux prédites pour la latitude, ont parfois une prédiction pour la longitude moins précise et inversement. Cela s'explique par le fait que les deux variables, longitude

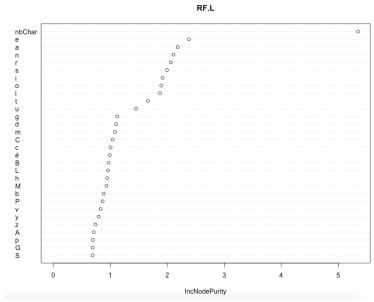


FIGURE 5.1 – Importance longitude

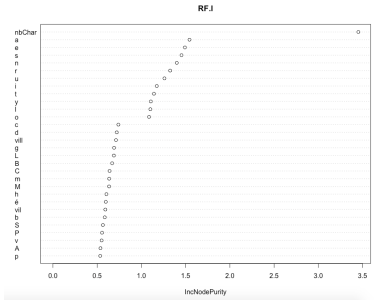


FIGURE 5.2 – Importance latitude

et latitude, ont été supposées indépendantes dans notre modèle. On peut constater ce phénomène avec l'exemple de Figure 7.13

Finalement, les dix villes les mieux prédites sont : Saint-Aubin-d'Écrosville, Cosswiller, Grisy-sur-Seine, Rancourt-sur-Ornain, Maisonnais, Lacaussade, Oinville-sur-Montcient, Maucourt, Vielmanay, Flacy.

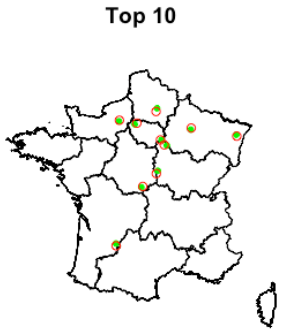


FIGURE 5.3 – Prédiction - Top 10 - forêts aléatoires

5.2.1 Placement de Triffouillis-les-Oies

Triffouillis-les-Oies a été située dans la région du centre. Cela diffère de nos prédictions en régression logistique, qui était de placer la ville en Pays de la Loire. En revanche, on remarque que la latitude pourrait correspondre à celle d'une ville de la région Pays de la Loire mais la longitude ne correspond pas. Cela diffère aussi de la classification par forêt aléatoire qui place la ville en Île-de-France. Encore une fois on peut remarquer que la longitude pourrait correspondre à celle d'une ville d'Île-de-France, mais cette fois c'est la latitude qui ne correspond pas.

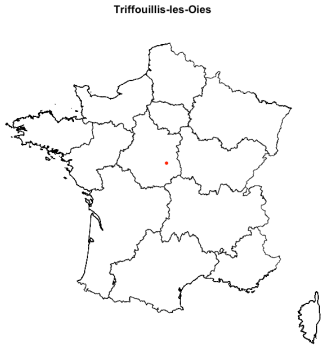


FIGURE 5.4 – Prédiction - Triffouillis-les-Oies - forêts aléatoires

5.3 Classification

5.3.1 Les données

Nous nous sommes servis de 2 jeux de données pour classifier les toponymes dans les régions françaises.

Nous avons utilisé d'abord la matrice des occurrences des lettres dans les noms de communes (les mêmes données que pour les modèles linéaires ; qui nous donnent ainsi un point de comparaison). Nous appellerons dans le reste de la section cette matrice $X^{(1)}$.

Nous avons aussi utilisé la distance de Levenshtein de deux manières différentes. If faut noter que calculer une matrice des distances sur la base de test demande un trop grande mémoire vive. En effet, il s'agit de calculer une matrice de taille environ 33000×33000 nécessitant plus de 10^9 opérations. Nous choisissons donc, de trouver une ou plusieurs communes représentant une région, et ce, pour toute les régions. Nous avons alors expérimenté deux stratégies différentes :

- **Commune médiane** : pour chaque région nous avons calculé le toponyme médian selon la métrique de Levenshtein ; afin d'obtenir un ensemble de 22 communes de référence. Autrement dit, pour chaque région, on calcule la distance de chaque ville avec toutes les autres ; puis les moyennes correspondantes. Nous regardons ensuite le toponyme associé à la médiane de ces moyennes. Enfin, pour chaque médiane ; on calcule les distances de la médiane avec l'ensemble des villes. Cette matrice sera notée :
$$X_{med}^{(2)}$$
- **$\mathbf{R = 20}$ représentants par régions basés sur les moyennes max** : Le point de départ est identique : nous calculons pour chaque commune sa pseudo-distance via le ratio de Lenvehstein avec toutes les autres de sa région. Nous faisons ensuite la moyenne de l'ensemble de ce vecteur de pseudo-distance. Dans ce cas là, nous sélectionnons les $R(= 20)$ communes ayant les moyennes les plus élevées. Nous obtenons donc finalement 20×21 communes dites "de référence". Nous allons alors nous servir de la matrice qui sur chaque ligne i contient les distances de la commune i à toutes

les communes de référence. Nous n'avons plus qu'environ 33000×420 calculs à effectuer ce qui est largement plus abordable. Il s'agit ici d'une sorte de barycentre avec les 20 communes qui ont les moyennes les plus élevées. Cette matrice sera elle notée :

$$X_{moy}^{(2)}$$

Note : les matrices $X_{med}^{(2)}$ et $X_{moy}^{(2)}$ n'ont pas les même dimensions, elles sont respectivement de taille $(n \times 21)$ et $(n \times 420)$.

Exemples de médianes

Voici quelques médianes pertinentes :

Alsace	Nord-Pas de Calais	Ile de France
Weitbruch (W)	Bourbourg (bourg)	Reuil-en-Brie (en-Brie)

Exemples et remarques sur les $R = 20$ représentants

Voici quelques exemples intéressant de faire remarquer :

- **Haute-Normandie** : Barneville-sur-Seine, Bernouville, Bourneville, Conteville, Foucrainville, Franqueville, Guernanville, Pinterville, Tourneville, Toutainville, Bertheauville, Bourdainville, Bourville, Canouville, Heurteauville, Maniquerville, Rainfreville, Sainneville, Sorquainville, Vatierville ;
- **Alsace** : Beinheim, Bernolsheim, Bindernheim, Friesenheim, Gerstheim, Hilsenheim, Landersheim, Limersheim, Maennolsheim, Marlenheim, Wingersheim, Witternheim, Wittersheim, Wiwersheim, Zittersheim, Baldersheim, Bartenheim, Biesheim, Kingersheim, Waltenheim ;

Il est difficile d'être plus explicite sur la particularité toponymique de certaines régions. Il est très clair ici que le suffixe "ville" est représentatif de la (Haute) Normandie, et le suffixe "heim", ou même "ersheim" ainsi que le préfixe "Wi" sont représentatifs de l'Alsace. On retrouve donc à nouveau ici l'histoire et les dialectes ayant marqués certaines régions françaises.

5.3.2 Création des forêts par validation croisée

Paramètres La création de forêts aléatoires dépend au moins de 2 paramètres : le nombre d'arbres ($n_{estimators}$) et la profondeur maximum des arbres (max_{depth}). Ces paramètres dépendant du jeu de données nous avons pensé à les choisir par validation croisée. En d'autres termes pour tester un couple de valeurs possibles $(n_{estimators}, max_{depth})$ on coupe la base d'apprentissage aléatoirement en 2, on crée la forêt sur le 1er sous-échantillon et on teste ses performances sur le 2ème ; on répète cette même opération m fois (ici 3 seulement pour limiter les temps de calculs) et on fait la moyenne des performances sur les m modèles pour obtenir finalement le score du couple $(n_{estimators}, max_{depth})$.

Pour le modèle engendré par $X_{occ}^{(1)}$ on trouvera finalement une profondeur maximale de 23 (à peu près $card(\mathbf{Y})$) et une taille de forêt de 450. Pour celui engendré par $X_{med}^{(2)}$, nous obtenons une profondeur de 23 et une taille de forêt de 1000. Pour celui engendré par $X_{moy}^{(2)}$, nous obtenons une profondeur de 35 et une taille de forêt de 1000 .

Note : Cette méthode de validation croisée sera réutilisée plus tard pour les réseaux de neurones et les SVM.

5.3.3 Performances

Le score de classification global de la 1ère forêt aléatoire est d'environ 22% tandis que celui de la seconde n'est que de 16%. Celui de la troisième est, lui, de 20%. Il semble que l'estimateur de la forêt aléatoire se comporte mieux avec les bases de données $X_{occ}^{(1)}$ et $X_{moy}^{(2)}$ c'est pourquoi nous nous intéresserons exclusivement à ces deux forêts jusqu'à la fin du chapitre.

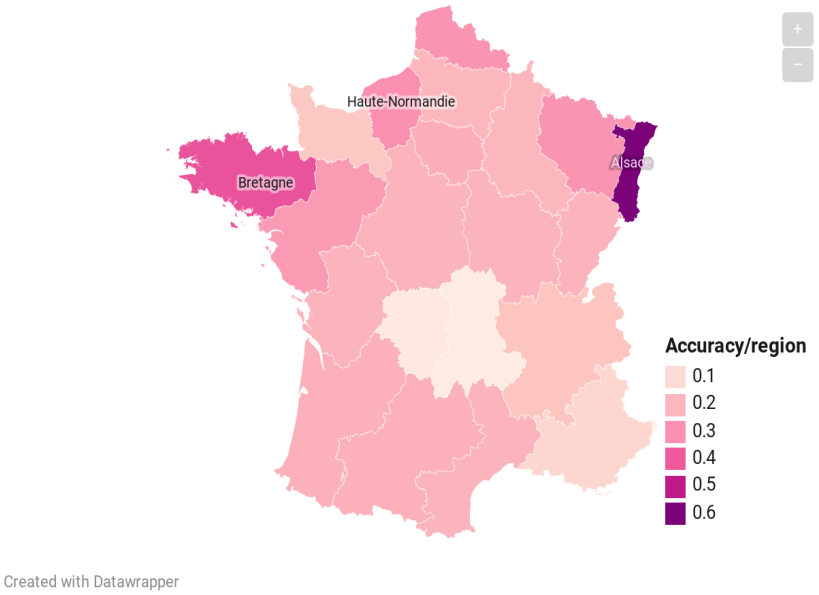


FIGURE 5.5 – Carte de France montrant le score de classification par région de la forêt aléatoire pour la base de données X^1

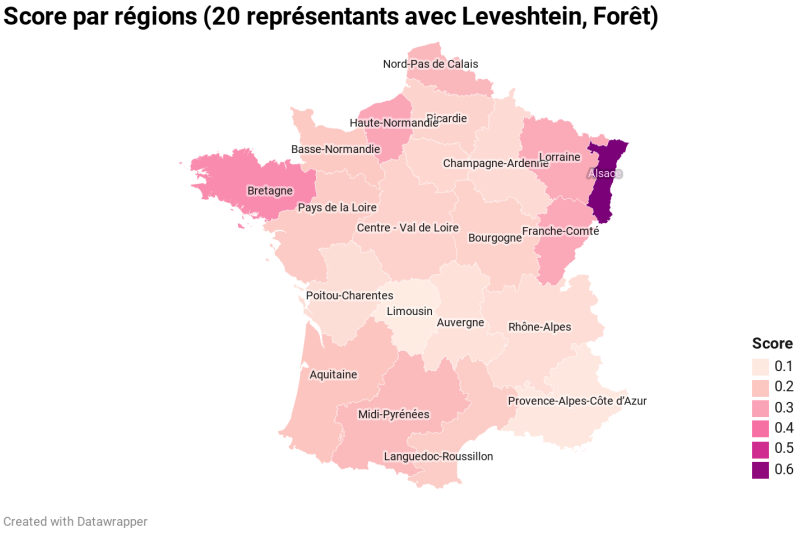


FIGURE 5.6 – Carte de France montrant le score de classification par région de la forêt aléatoire pour la base de données $X^{(2)}_{moy}$

Il est aussi pertinent de regarder le score de classification par label : c'est à dire le classificateur se comporte-t-il de manière équitable entre les régions ou est-il meilleur sur certaines régions que d'autres ? La réponse est donnée dans le graphique suivant pour l'entrée $X^{(1)}$: Figure 5.5

Elle est donnée ici pour l'entrée $X^{(2)}_{moy}$: Figure 5.6

Il est intéressant de remarquer que la crédence d'une prévision par le modèle dépend fortement du label de cette prévision. En effet, si la sortie est 'Alsace' ; la ville a effectivement 62% de chance de se trouver effectivement en Alsace. Tandis que si le label de la sortie est 'Auvergne' ; la ville n'a que 7% de chances de se trouver vraiment en Auvergne.

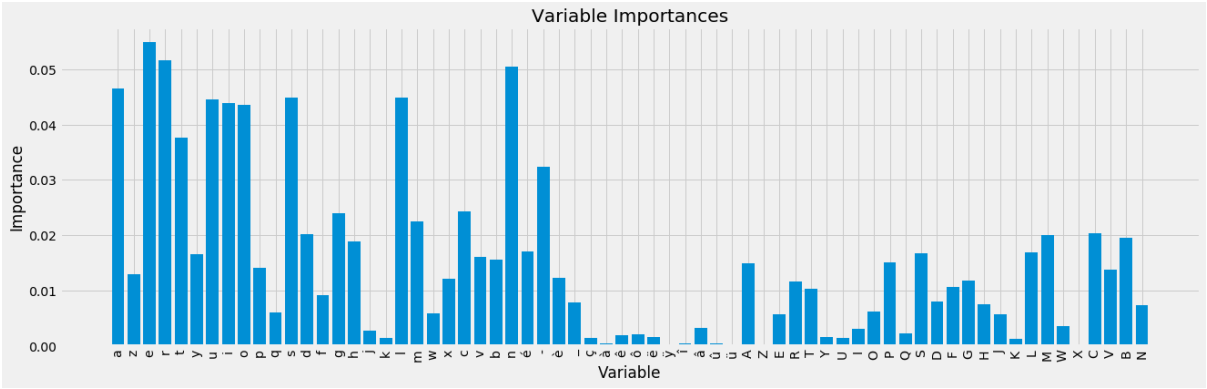


FIGURE 5.7 – Importance du nombre d'occurences des lettre dans le toponyme dans la création des CART de la forêt

5.3.4 Importance des variables et élagage

Nous regardons dans cette section la forêt engendrée par X^1 uniquement.

importance des variables Nous regardons maintenant quelles variables ont pesé sur la création de la forêt aléatoire : voir le graphique Figure 5.7

Nous remarquons que certaines des covariables n'ont pas ou peu d'impact sur les arbres de décision de la forêt. C'est le cas notamment de j,k,ç,à,ê,ô,ë,ÿ,î,â,û,ü,Z ou X. Dans les arbres de décision les lettres les moins importantes sont en fait les lettres les moins fréquentes. Nous pouvons ainsi nous demander si le score de classification ne peut être améliorer en construisant des arbres sans ces variables qui peuvent amener l'estimateur à sur-apprendre les données d'apprentissage.

Sélection des variables Nous créons ainsi une nouvelle forêt aléatoire en imposant que les arbres ne peuvent utiliser que $\sqrt{p}$ covariables pour la décision ; et en gardant les mêmes $(n_{estimators}, max_{depth})$. Le score global après élagage est de 23% ; on a donc gagné 1% de performance en élaguant les arbres.

Trifouillis les oies Nous plaçons Trifouillis-les-oies, selon la forêt engendrée par X^1 en **Île-de-France**.

Chapitre 6

Multi-Layer Peceptron (MLP)

6.1 Principe

Nous nous attardons ici sur le principe du *perceptron*, un réseau de neurones à une couche cachée. Nous utiliserons par la suite un réseau de neurones de type *perceptron* avec plusieurs couches cachées, mais le principe reste exactement identique.

Un **Perceptron** est un réseau de neurones des plus classiques, a une seule couche cachée : Il dispose d'une **fonction d'activation** et d'une **fonction de sortie**.

- Chaque neurone prend en entrée toutes les variables fournies (par l'entrée *Input*), et les fait passer dans sa fonction d'activation.
- La fonction de sortie prend donc les valeurs de tous les neurones pour donner une sortie qui donne, dans un cas général, une probabilité d'appartenir à une classe (**softmax** dans la plupart des cas de classification).

Nous avons ensuite une **fonction de propagation**, (*Forward*), qui permet de faire traverser notre *Input* au réseau de neurones jusqu'à la sortie. Elle utilise donc les fonctions d'activation et de sortie citées ci-dessus.

Puis, on possède aussi une **fonction de perte**, qui va permettre de savoir, une fois que notre information a traversé le réseau, à quel point ont est loin du résultat attendu.

- Elle est génralement soit logarithmique $\left(-\frac{1}{M}\sum_{n\in M}\sum_{c\in C} y_{n,i} log(\tilde{y}_{n,i})\right)$ soit équivalent aux carrés de la distance $\left(\frac{1}{2}\sum_{0=1}^{O}\left(d_o^k - s_o^k\right)^2\right)$ entre la prédiction et le résultat exact.
- On remarquera que ces fonction sont toutes deux convexes (condiation obligatoire pour optimiser la sortie)

Enfin, la partie la plus importante repose sur le concept de **rétropropagation** (*Backpropagation*). Il s'agit de faire remonter l'information de la perte calculée à la sortie jusqu'à à l'entrée du réseau, puis de la corriger. Corrigée une première fois, nous recommençons la propagation, et ainsi de suite jusqu'à ce que la correction sur la perte soit minimale. Cette méthode se base donc sur une descente de gradient (classique, ou stochastique généralement) pour minimiser la fonction de perte. Le calcul de ce résultat de *Backpropagation* est fondé mathématiquement sur des *chain-rules* de dérivées partielles.

Dans la plupart des cas, pendant la **rétropropagation** une pénalité est infligée, ce qui empêche une convergence totalement optimale de la descente de gradient. Cela permet d'empêcher le sur-apprentissage, assez fréquent lorsque le nombre de neurones est élevé. Une autre technique pour éviter ce sur-apprentissage est de faire aléatoirement sauter des neurones dans le réseau.

Nous utiliserons un réseau de neurone de type **Perceptron** à plusieurs couches, ou "**MLP**". Le principe est exactement le même, mais nous avons donc plusieurs couches, renforçant drastiquement l'efficacité du réseau, surtout sur de grandes dimensions.

Note : Pour un détail des mathématiques derrière un réseau de neurones, un exemple en notebook réalisé en 2018 par Lucas Perrin et David Gontier Phd. est disponible sur le GitHub.

Une illustration très connue d'un réseau de neurones est représentée en Figure 6.1.

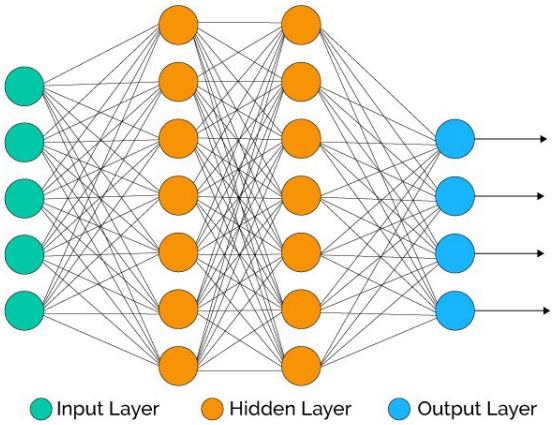


FIGURE 6.1 – Illustration classique d'un réseau de neurones à plusieurs (ici 2) couches cachées.

Nous cherchons donc à prédire la région avec un réseau de neurones à plusieurs couches. On prendra ici en entrée la matrice $X^{(2)}_{moy}$ définie dans la section sur les forêts aléatoires pour la classification :

- Nous possédons un input qui peut être lu par le MLP ;
- l'*Output* est de type classification (région) par rapport à une sortie *softmax* qui prédit avec une certaine probabilité la région ;
- nous regardons les résultats après avoir *optimisé* les paramètres via une *cross-validation* (comme expliquée dans le chapitre précédent) ;
- nous comparons la prédiction du MLP avec la base de test ;
- nous concluons, ou non de l'efficacité de cette méthode.

6.2 Optimisation des paramètres(Tuning)

Nous allons alors tenter de trouver les paramètres optimaux du réseau de neurones.
Il faut avoir conscience que trouver les paramètre optimaux d'un réseau de neurones relève d'une grande complexité, il est possible de jouer sur un certain nombre de variables :

- α : le taux d'apprentissage présent dans la *Backpropagation* ;
- # couches ;
- # représentants par couches.

Nous choisissons ici de ne jouer que sur la profondeur et la taille de chaque couche du réseau. Nous allons, tout comme pour la forêt aléatoire, effectuer une sélection avec validation croisée. On trouve que la configuration d'un réseau de deux couches à 100 neurones est la plus optimale.

6.3 Performances et résultats

Nous obtenons alors un score global de 19.59%, légèrement inférieur à ceux obtenus avec la classification par forêt aléatoire. Nous pouvons nous tourner vers les résultats par labels : voir Figure 6.2.

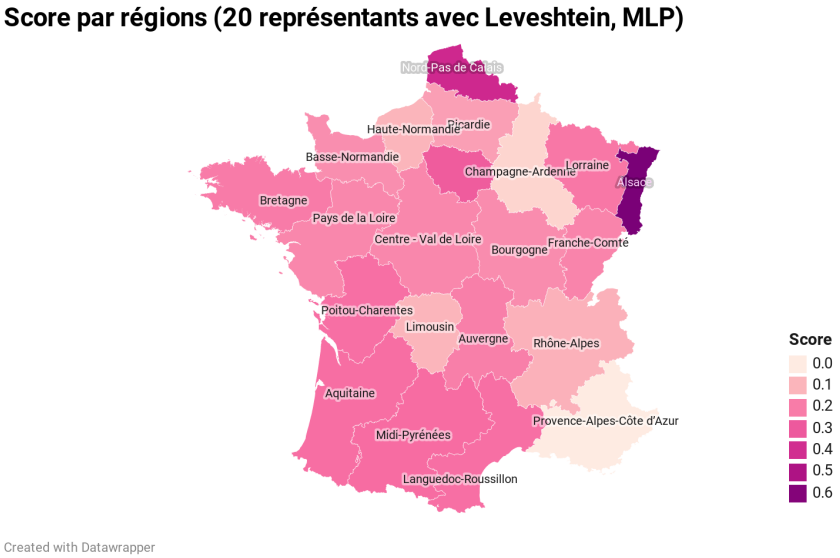


FIGURE 6.2 – résultats d'un MLP à deux couches de 100 neurones, pour une entrée avec 20 représentants par régions pour la pseudo-distance associée au ratio de Levenshtein.

6.4 Placement de *Trifouillis-Les-Oies*

Par ce MLP, la commune de *Trifouillis-Les-Oies* se retrouve en Haute-Normandie.

Chapitre 7

Support-Vector Machines

7.1 Principe

Les Support-Vector Machines (SVM) ont été vus en cours d'apprentissage statistiques, on ne reviendra donc que peu sur la théorie, voir Roche, 2020, polycopié sur les SVM. Il s'agit donc de trouver l'hyperplan qui maximise la marge (distance entre l'hyperplan et les échantillons à séparer (*support vectors*). Ce problème peut être formulé comme un problème d'optimisation quadratique pour lequel il existe des méthodes de résolutions numériques connues.

Cependant, nous sommes ici dans un cas assez spécifique : le cas multi-classe à marge souple.

Nous rappelons tout de même certaines notions avec les notation suivantes :

- $x = (x_1, \cdots, x_p)^t$ un vecteur d'entrées ;
- $w = (w_1, \cdots, w_p)^t$ un vecteur de poids ;
- $h : x \to w^t x + w_0$ le classifieur linéaire du problème ;
- $\mathbf{l} = (l_1, \cdots, l_n) \in \{-1, 1\}$ le vecteur de labels associées aux entrées $x_1, ..., x_n$.

Nous cherchons alors :

$$\arg\max_{w,w_0} \left\{ \frac{1}{\|w\|} \min_k \left[l_k \left(w^t x_k + w_0 \right) \right] \right\}$$

ou encore avec mise à l'échelle :

$$\min \left\{ \frac{1}{2} \|w\|^2 \right\} \quad \text{sous contraintes : } l_k \left(w^t x_k + w_0 \right) \geq 1$$

qu'on résout avec le *Lagrangien* suivant (minimisé sur w et w_0 et maximisé sur α

$$\mathcal{L}(w, w_0, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{k=1}^{p} \alpha_k \left\{ l_k \left(w^t x_k + w_0 \right) - 1 \right\}$$

et donc le problème dual suivant :

$$\max \tilde{L}(\alpha) = \sum_{k=1}^{p} \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j l_i l_j x_i^t x_j \quad \text{sous les contraintes : } \alpha_k \geq 0, \sum_{k=1}^{p} \alpha_k l_k = 0$$

7.1.1 Marge souple

Nous sortons alors de la théorie observée en cours qui consistait à trouver un hyperplan séparateur dans un cas où la condition de séparabilité était vérifiée. Il est évident que nos données sont beaucoup trop nombreuses pour vérifier la condition de séparabilités, même avec le *kernel trick*.

Kernel trick : Le *kernel trick* consiste à, dans un cas de non-séparabilité, appliquer une transofrmation non linéaires ϕ aux données d'entrée dans le but de reconsidérer le problème dans un espace de dimension supérieur. Cet espace d'arrivée est appellé *redescription space*. Cela revient en fait à chercher l'hyperplan :

$$h(x) = w^t \phi(x) + w_0$$

On veut alors maximiser :

$$\tilde{L}(\alpha) = \sum_{k=1}^{p} \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j l_i l_j \phi\left(x_i\right)^t \phi\left(x_j\right) \quad \text{sous les contraintes : } \alpha_i \geq 0, \sum_{k=1}^{p} \alpha_k l_k = 0$$

ce qui induit un produit scalaire dans l'espace de redescription. On utilise alors le *kernel trick* qui consiste à passer par un noyau K vérifiant $K\left(x_i, x_j\right) = \phi\left(x_i\right)^t \cdot \phi\left(x_j\right)$. Cela permet de rester dans l'espace d'origine et ne nécessite pas de connaitre explicitement la transformation non linéaire mais uniquement la fonction noyau. Nous utiliseront alors le noyau RBF *Radial basis function* suivant :

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Pour en revenir au principe de marge souple, nous définissons donc une marge qui autorise les mauvaises classifications en introduisant des *slack varaibles* η dans les contraintes, ainsi qu'une constante C qui permet de contrôler le rapport entre l'erreur de classement et la taille de la marge. Le problème d'optimisation devient alors la minimisation de :

$$\frac{1}{2}\|w\|^2 + C \sum_{k=1}^{p} \eta_k \quad \text{sous les contraintes : } l_k\left(w^t x_k + w_0\right) \geq 1 - \eta_k \quad \eta_k \geq 0$$

Nous regarderons plusieurs valeurs pour C lors du *tuning* par validation croisée.

7.1.2 Multi-classe

La méthode des SVM ne s'applique uniquement qu'à un problème à deux classes. On peut néanmoins contourner cette contrainte de deux manières lorsque l'on possède m classes :

- méthode **ovr** (*one versus rest*) : Nous construisons m machines qui attribuent 1 à la classe actuelle et -1 à toutes les autres. C'est la machine qui donne la meilleur marge à l'entrée fournie qui est retenue ;
- méthode **ovo** (*one versus one*) : Nous construisons $m(m-1)/2$ machines qui confrontent chacunes des m classes. Lors de l'analyse de l'entrée, c'est la classe qui apparaît le plus souvent qui sera retenue.

  Nous confronterons les deux méthodes lors de la validation croisée.

7.2 Choix des hyperparamètres (Tuning)

Comme indiqué précedemment, nous choisissons ici de jouer sur la méthode de multi-class et sur la constance C lors de la validation croisée. On notera qu'on doit ici créer au moins 21 machines, sinon 210 ce qui représente un coût en opérations et en temps beaucoup plus élevé que pour d'autres méthodes. On obtient une configuration optimale dans le cas d'une méthode *ovo* pour une constance $C = 2$.

7.3 Performances et Résultats

La performance globale est de 21.53 % de précision. Les résultats régions par régions sont obesrvables en Figure 7.1.

Il est possible de remarquer certaines choses. Tout d'abord, le score impressionnant de l'Alsace est toujours présent, et d'une manière générale, les régions telles que la Bretagne ou le Languedoc-Rousillon ont des scores aussi très élevés. Le Limousin se retrouve lui avec un score nul.

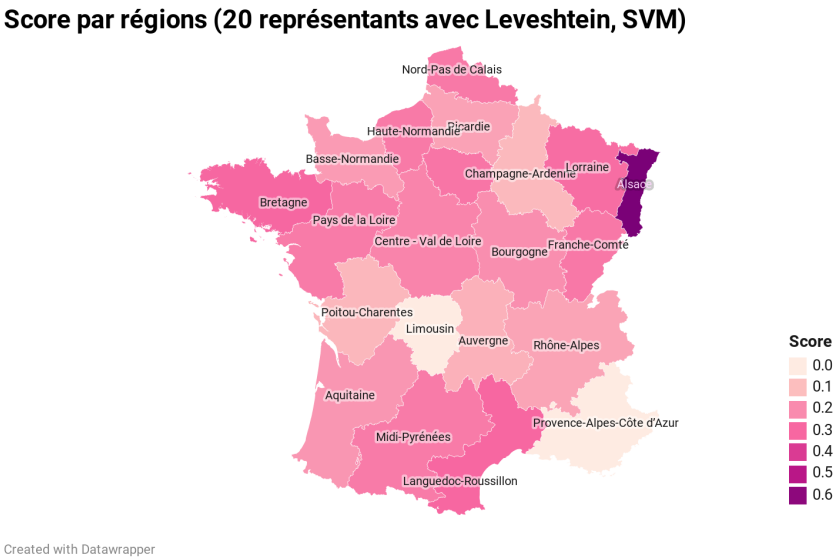


FIGURE 7.1 – résultats d'un SVM avec méthode *ovo* et $C = 2$ pour une entrée avec 20 représentants par régions pour la distance associée au ratio de Levenshtein.

7.4 Placement de *Trifouillis-Les-Oies*

Par ce SVM, la commune de *Trifouillis-Les-Oies* se retrouve placée en Haute-Normandie. Cela a toujours été le cas lorsque la pseudo-distance utilisée était celle de Levenshtein. Nous pouvons alors supposer, à la vue des différents résultats de régression basées sur la matrice d'occurences des lettres que le choix de la manière de présenter les données y est pour quelque chose.

Modèles de régression	Données	Erreur moyenne latitude	Erreur moyenne longitude
ANOVA	Nombre d'occurrences des lettres	0.0011	0.0017
Forêt aléatoire	Nombre d'occurrences des lettres + Syllabes fréquentes	0.0010	0.0017

TABLE 7.1 – Performance des modèles de régression

Méthode	Données	Score de classification
Régressions logistiques binaires	Nombre d'occurences des lettres+ syllabes fréquentes	19.5%
Régression logistique multinomiale	Nombre d'occurences des lettres	17.96%
k-nn modifié	**Matrice des ratios de Levenshtein**	**25.72**%
k-nn modifié	Matrice des distances de Jaro-Winkler	22.79%
Forêt aléatoire	Matrice d'occurence des lettres	22.87%
MLP	Matrice des ratios de Levenshtein avec communes de références	19.59%
SVM	Matrice des ratios de Levenshtein avec communes de références	21.53%

TABLE 7.2 – Performances des modèles de classification

Conclusion

Performances générales des modèles Nous allons récapituler ici les performances générales des modèles implémentés voir Tableau 7.1 et Tableau **??**

Nous avons vu au cours de notre exploration de l'apprentissage statistique, plusieurs manières d'interpréter les données et plusieurs algorithmes de machine-learning différents. En régression si nous avons tenté de faire un modèle linéaire et une forêt aléatoire pour modéliser la latitude et la longitude nous avons vite été arrêtés par le problème d'estimation jointe de coordonnées : nous avons dû supposer la latitude et la longitude indépendantes. De plus ces modèles placent les villes au centre de la France (les erreurs quadratiques sont proches de la variance des données). C'est pourquoi nous nous sommes principalement concentrés sur les méthodes de classification. En classification supervisée nous avons vu des modèles linéaires, des forêts aléatoires, des MLP et des SVM et en non supervisée nous avons proposé une méthode des k-plus proches voisins. Finalement, c'est cette dernière méthode qui finalement a fait ses preuves et qui fait le meilleur score de classification parmi tous les modèles étudiées. Si nous devions continuer à travailler sur ce mémoire nous aurions sans doute continuer dans la voie de l'apprentissage non supervisé pour essayer d'autres méthodes.

Où est Trifouillis-les-oies ? Nous pouvons supposer à la vue de Figure 7.2 que Trifouillis-les-oies se situerait dans le quart Nord-Ouest de la France (mais pas en Bretagne) ; notre meilleur modèle le place en Haute-Normandie.

Pour aller plus loin... Si nous voulions conclure d'une manière simple, nous répondrons au problème posé : "il n'est pas possible de prédire facilement l'emplacement d'une commune sur la carte française seulement à partir de son nom". Mais une réponse plus détaillée est tout aussi complexe que le problème

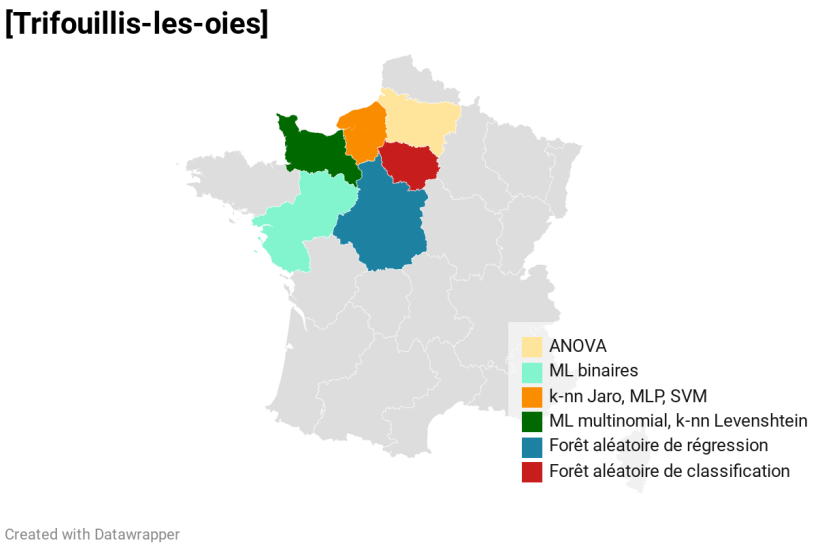


FIGURE 7.2 – Placement de Trifouillis-les-oies selon différents modèles

en lui même. Aucun des modèles et méthodes testée n'a permis de dépasser la barre des 25 %. Nous pouvons donc affirmer que, mathématiquement parlant, plus d'une ville sur quatre ne peut pas correctement se placer sur la carte de France, rien qu'avec son nom. Cependant, lorsque nous rentrons dans le détail, on s'aperçoit que certaines régions de France sont facilement prévisible, ou encore que certaines lettres sont très représentatives d'une zone géographique particulière. Les cartes crées grâce aux scores de nos modèles sont toutes plus ou moins similaires (dans leur globalités) et assez bien corrélées avec des analyses plus littéraires du sujet. Les communes aux consonances germaniques se trouvent à l'Est, en Alsace ou en Lorraine par exemple, les communes aux consonances celtes à l'Ouest, en Bretagne notamment. Le suffixe "ville" est très représentatif de la Normandie et les noms basques se trouvent au Sud-Ouest, en Aquitaine ou Midi-Pyrénées principalement. Ce n'est donc pas un échec, mais plutôt une belle découverte que celle d'avoir pu faire ressortir à l'aide d'outils statistiques et mathématiques les particularités du territoire Français.

Cependant, nous n'avons qu'effleurée la surface de l'apprentissage statistiques, et d'autres méthodes que nous apprendrons par la suite, avec d'autres supports, pourront, peut-être, voir sûrement, nous faire revenir sur ce sujet des plus intéressants.