Projet Semestriel GM4 : Reconnaissance de chiffres manuscrits

KHALBOUS Adnene - CHANIER Jean-Baptiste - BOUAISS Marwan

1^{er} Juin 2020

INSA Rouen
À l'attention de Monsieur Bruno PORTIER

Contents

1 Description des données **4**
- 1.1 Contenu du jeu de données . . . 4
- 1.2 Effectif de chaque classe . . . 4
- 1.3 ACP . . . 5
- 1.4 Matrice des Corrélations . . . 5
- 1.5 Classification non supervisée . . . 6

2 Présentation des méthodes **8**
- 2.1 LDA . . . 8
  - 2.1.1 Fondements Mathématiques de LDA . . . 8
  - 2.1.2 Implémentation dans les programmes . . . 8
- 2.2 One vs All . . . 9
  - 2.2.1 Fondements Mathématiques de la Régression Logistique . . . 9
  - 2.2.2 Implémentation dans les programmes . . . 10
- 2.3 Arbre CART . . . 10
  - 2.3.1 Arbre Maximal . . . 10
  - 2.3.2 Arbre Optimal . . . 11
  - 2.3.3 Application à nos données . . . 11
- 2.4 Random Forest . . . 12
  - 2.4.1 Définitions . . . 12
  - 2.4.2 Algorithme . . . 12
  - 2.4.3 Implémentation dans les programmes . . . 12
- 2.5 SVM . . . 13
  - 2.5.1 Principe d'utilisation . . . 14
  - 2.5.2 Application à nos données . . . 15
- 2.6 Réseau de neurones . . . 15
  - 2.6.1 Algorithme . . . 15
  - 2.6.2 Implementation dans les programmes . . . 16
- 2.7 Régression logistique polytomique . . . 16
  - 2.7.1 Fondements Mathématiques . . . 17
  - 2.7.2 Implementation dans les programmes . . . 17
  - 2.7.3 Construction d'un modèle réduit . . . 18

3 Résultats **19**
- 3.1 Explication du code . . . 19
- 3.2 Tableau récapitulatif des taux de bon classement . . . 19
  - 3.2.1 Données d'apprentissage . . . 19
  - 3.2.2 Données test . . . 20
- 3.3 Analyses . . . 20
- 3.4 Test supplémentaire . . . 20

Introduction

Dans le cadre du projet semestriel du semestre 8 de GM, nous nous sommes interessé à un problème de reconnaissance de chiffres. Nous disposions de 1000 images, chacune représentant un chiffre compris entre 0 et 9. Comme nous avions accès au chiffre associé à chaque image, le projet s'articulait autours de méthodes de classification supervisée, donc dans le but de retrouver au mieux l'affectation des images dans l'une des 10 classes disponibles. Nous avons également implémenté des méthodes pour mieux visualiser nos données ainsi que quelques méthodes de classification non supervisée. Nous avons tenté de faire varier au maximum les paramètres de l'étude pour les ajuster au mieux et ainsi obtenir l'analyse la plus complète possible.

1 Description des données

1.1 Contenu du jeu de données

Nous disposons de 1000 images, chacune comportant 256 valeurs correspondant à une mesure sur un pixel. La variable y est la variable correspondant aux chiffres représentés par les images. De plus, les variables x_1 à x_256 représentent chacune un pixel différent. Les valeurs possibles pour chaque pixel sont comprises entre 0 et 1. 0 pour un pixel blanc et 1 pour un pixel noir, les valeurs intermédiaires correspondent donc à des nuances de gris.

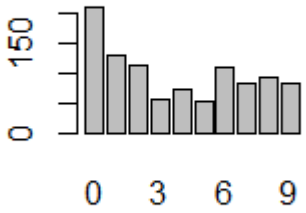
Exemple : Visualisation de la 100ème image du jeu de donnée, représentant un 6 (le choix des couleurs est arbitraire)

1.2 Effectif de chaque classe

Tableau de répartition des données par classes :

y	0	1	2	3	4	5	6	7	8	9
effectif (%)	20,9	12,9	11,2	5,7	7,2	5,3	11,1	8,2	9,3	8,2

Diagramme en bâton des effectifs de chaque classe :



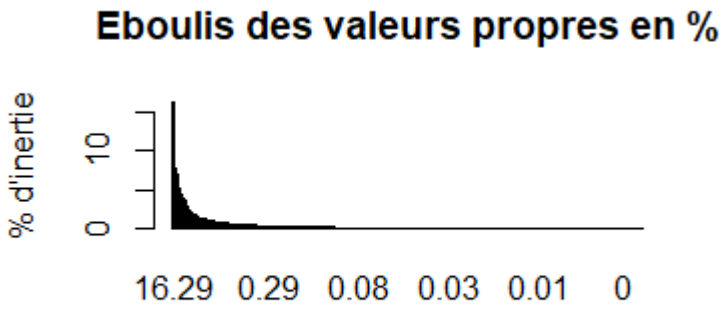
On remarque que les effectifs de la variable à expliquer ne sont pas équilibrés, par exemple 20,9% des images représentent des 0 tandis que seulement 5,3% sont

des 5. On se trouve donc dans le cas d'une classification déséquilibrée.

Cela peut influencer négativement l'étude, en effet, les chiffres en "sous-effectifs" seront plus difficiles à prévoir car ces classes auront moins de poids lors de la construction du modèle.

1.3 ACP

Nous avons effectué une ACP sur nos données complètes afin de mieux nous familiariser avec celles-ci: la première valeur propre ne représente qu'une inertie de 16.294% tandis que la seconde en représente 7.807% . Ainsi, la projection de nos variables sur le plan associé à ces deux valeurs propres ne permet pas d'avoir une vue d'ensemble de nos données. On note que, dans notre cas, le nombre important de variables explicatives empêche les analyses à partir de graphiques à cause de la surabondance d'information représentée.



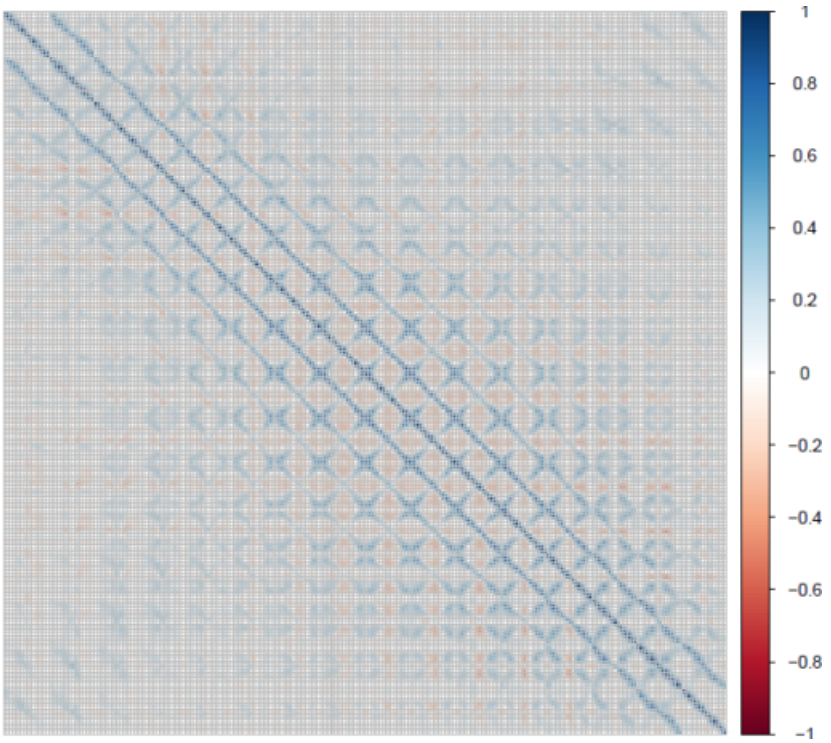
De plus, pour obtenir 95% de l'inertie totale, il est nécessaire d'utiliser les 92 valeurs propres les plus importantes. C'est une réduction de dimension intéressante puisque l'on passe de 256 variables initialement à 92. Cependant, ce nombre reste encore trop important pour pouvoir apporter une analyse sur notre jeu de données.

1.4 Matrice des Corrélations

Nous avons utilisé la matrice de corrélations entre toutes nos variables pour savoir si certaines d'entre elles étaient fortement corrélées. En premier lieu, nous avons pris comme critère une corrélation entre les 2 variables supérieure à 0.95 en valeur absolue. Avec un critère si restrictif, aucun des couples de variables n'est considéré comme corrélé, cependant, à titre informatif, en abaissant la limite à 0,90 en valeur absolue, cette fois, 6 couples distincts sortent du lot.

Nous choisissons tout de même de conserver ces couples dans le sens où une corrélation comprise entre 0,90 et 0,95 n'est pas suffisante pour pouvoir éliminer l'une des deux variables impliquée.

Nous avons cherché à afficher cette matrice de corrélations avec un code couleur distinctif.



On remarque alors qu'outre la diagonale logiquement corrélée à 1, des surdiagonales (et sous diagonales par symétrie de la matrice) proches de la diagonale dominante offrent des corrélations importantes, cela s'explique par le fait que sur une image, lorsqu'un pixel est colorisé, ses voisins ont plus de chance de l'être également car le "trait du dessin" est rarement large d'un seul pixel.

1.5 Classification non supervisée

Afin d'avoir une première idée de la difficulté de résolution de ce problème, nous avons étudié le nombre de classes préconisé naturellement dans le cadre de méthodes de classification non supervisées. Nous avons donc utilisé les méthodes CAH, k-means et modèle de mélange gaussien sur nos données. A chaque fois, le nombre de classe conservé pour la constitution du modèle était de 2 au lieu des 10 en réalité. Lorsque nous avons imposé le nombre de 10 classes, ces méthodes permettaient uniquement une bonne reconnaissance du chiffre 1 dont

les représentations étaient bien regroupés en une classe mais les autres chiffres n'étaient pas distingués et étaient distribués dans 9 autres classes sans suivre une tendance claire. Par exemple dans le cas de la CAH en ayant fixé le nombre de classes à 10 :

	y									
groupes.cah	0	1	2	3	4	5	6	7	8	9
1	1	0	2	0	0	1	80	0	1	0
2	2	0	58	8	0	6	3	0	2	0
3	1	0	12	0	26	3	1	1	0	2
4	0	4	3	0	37	1	0	17	19	73
5	20	2	32	48	0	26	5	1	71	0
6	0	123	0	0	0	0	6	0	0	0
7	91	0	3	0	0	0	10	0	0	0
8	1	0	1	1	7	16	1	12	0	7
9	93	0	0	0	0	0	5	0	0	0
10	0	0	1	0	2	0	0	51	0	0

On voit que les images représentant le chiffre 1 ont été rassemblées quasiment exclusivement (123 sur 129) dans une seule classe. À contrario, les autres chiffres voient leurs effectifs répartis entre 2 ou 3 classes à l'image du chiffre 0 réparti en 3 classes. Les deux autres méthodes cités precedemment donnent des résultats similaires.

2 Présentation des méthodes

2.1 LDA

Pour pouvoir classer nos données en 10 classes selons le chiffre correspondant nous avons choisi d'utiliser en premier lieu une méthode de type probabiliste: l'Analyse Discriminante Linéaire (LDA).

2.1.1 Fondements Mathématiques de LDA

- Soit Y notre variable aléatoire à expliquer, qui prend ses valeurs dans 0...9
- $X = (X_1, ..., X_{256})$ est notre vecteur de variables explicatives qui correspond aux pixels des images
- (π_0,...,π_9) la distribution de Y ou $\pi_k = P(Y = k)$ est la proportion théorique de G_k encore appelée probabilité à priori de G_k.
- f_k est la densité de probabilité du vecteur alétoire X dans le groupe G_k

$f(x) = \sum_{k=1}^{K} \pi_k f_k(x)$
L'affectaion d'un nouvel individu pour lequel on ne dispose que de la mesure x de$X = (X_1, X_2, ..., X_{256})^T$ va donc se faire sur la base de la régle de classement optimal de bayes et donc des valeurs des probabilités à posteriori estimées :

$$q = argmax_{k=1,2,..K} P[G_k|x] = argmax_{k=0,2,..9} \pi_k f_k(x)$$

avec dans le cas de LDA

$$f_k(x) = \frac{1}{\sqrt{(2\pi)^{256} det(\Gamma)}} exp(-\frac{1}{2}(x - \mu_k)^T \Gamma^{-1}(x - \mu_k))$$

Or on sait que LDA correspond au cas homoscédatique c'est à dire que : $\Gamma_1 = \Gamma_2 = ... = \Gamma_K = \Gamma$ alors on peut aussi trouver l'indice q tel que :

$$q = argmax_{k=0,...,9}(x^T \Gamma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Gamma^{-1} \mu_k + log(\pi_k))$$

$\Rightarrow$ On affectera le nouvel individu au groupe G_q selon la valeur de q.

2.1.2 Implémentation dans les programmes

En R, les packages disponibles pour créer des modèles de LDA sont les packages MASS. Dans nos programmes, on a utilisé le fonction lda pour construire nos modéles sur les données d'apprentissage en utilisant les paramètres par défaut et ensuite faire la prediction du modèle pour pouvoir comparer notre modèle à nos données de base en construisant la matrice de confusion et en calculant le taux de bon classement.

2.2 One vs All

Dans nos données, la variable y à expliquer est multinomiale, c'est à dire qu'elle comporte plus de 2 modalités. Cela interdit l'utilisation d'un régression logistique classique, qui est applicable uniquement dans le cas d'une variable à expliquer de type binaire.

Cependant, on peut contourner ce problème en appliquant une méthode de type One vs All : Dans notre cas, la variable y compte 10 modalités (valeurs allant de 0 à 9), on va alors effectuer 10 régressions logistique classiques après avoir rendu y binaire de 10 manières différentes, mais toujours en suivant le même procédé: à chaque fois, on prend la variable y telle qu'elle est sans modification (10 modalités) et on va choisir un chiffre entre 0 et 9, notons le a. On va construire le nouveau y avec la règle suivante : $y_i^a = 1$ si $y_i = a$ et $y_i^a = 0$ sinon. La nouvelle variable y^a est donc bien une variable binaire.

Les 10 modèles sont construits en cherchant à expliquer les y^a avec l'ensemble des variables x_k. Ils se présentent sous la forme de $k+1$ coefficients réels (voir partie sur les explications mathématiques). On peut alors appliquer ces 10 modèles à chaque image et obtenir une mesure de vraisemblance d'appartenance à chaque classe possible. Pour chaque image, elle sera affectée à la classe qui offre la mesure de vraisemblance la plus haute.

2.2.1 Fondements Mathématiques de la Régression Logistique

Soit y la variable binaire à expliquer et soit $x = (x_1, ..., x_N)$ un individu du jeu de données. On cherche à estimer la probabilité que x appartienne à chacune des deux classes, en connaissant les valeurs des pixels pour cette image, c'est à dire estimer :

$$p(x) = P(Y = 1|(X_1, ..., X_N) = x)$$

Dans ce cas, un modèle de régression logistique serait :

$$logit(p(x)) = \alpha_0 + \alpha_1 x_1 + ... + \alpha_N x_N$$

avec la fonction *logit* suivante : $logit(y) = log(\frac{y}{1-y})$, pour $y \in]0, 1[$et $logit(y) \in \mathbb{R}$.

On peut donc réécrire l'expression précédente :

$$p(x) = \frac{exp(\alpha_0 + \alpha_1 x_1 + ... + \alpha_N x_N)}{1 + exp(\alpha_0 + \alpha_1 x_1 + ... + \alpha_N x_N)}$$

Le problème se résume alors à estimer les α_i pour $i \in \{0, ..., 256\}$ à partir des données. On utilise en fait le maximum de l'expression de vraisemblance suivante :

$$L(\alpha, z) = \prod_{i=1}^{n} p(x_i)^{z_i}(1 - p(x_i))^{1-z_i}$$

avec $z_i \in \{0, 1\}$

On peut alors obtenir les estimateurs des coefficients α_i avec l'algorithme de Newton-Raphson. Ils seront notés $\hat{\alpha_i}$.

L'estimation de la probabilité d'appartenance de x à une classe est alors :

$$\hat{p(x)} = \frac{exp(\hat{\alpha_0} + \hat{\alpha_1} x_1 + ... + \hat{\alpha_N} x_N)}{1 + \sum_{k=2}^{m} exp(\hat{\alpha_0} + \hat{\alpha_1} x_1 + ... + \hat{\alpha_N} x_N)}$$

2.2.2 Implémentation dans les programmes

Des fonctions ont été spécialement implémentées en R pour effectuer une méthode de type One vs All sur des données. Il s'agit des fonctions cost.proportionate.classifier du package costsensitive. Cependant, dans l'optique de contrôler nous même toutes les étapes de cette méthode, nous avons pris le parti de la recoder intégralement. Ainsi, nous avons mécaniquement obtenu les 10 variables y^a, $a \in \{0, ..., 9\}$. Nous avons alors effectué les 10 regressions logistiques classiques avec la fonction glm du package MASS, en cherchant à expliquer les variables y^a avec les 256 variables x_j, $j \in \{1, ..., 256\}$. Nous avons ensuite récupéré les coefficients avec la commande "modele$coefficients". En faisant le produit scalaire entre le vecteur des valeurs de pixel de l'image (en ajoutant un 1 pour prendre en compte le premier coefficient qui correspond à l'ajout de la constante additive) et les coefficients du modèles, on peut obtenir les mesures de vraisemblance d'appartenance de chaque classe. On affecte alors l'image à la classe qui offre la plus haute mesure de vraisemblance d'appartenance. On répète ce processus pour chaque image du jeu de données jusqu'à les avoir toutes affectées.

2.3 Arbre CART

Un Arbre CART est un arbre binaire, la construction d'un arbre binaire équivaut à une partition de nos données. L'algorithme CART nous permet de construire un arbre binaire et nous fournit un modèle de classification supervisée. Le principe de construction d'un arbre CART est, dans un premier temps, de construire un arbre maximal, puis l'étape d'élagage qui va donner des sous arbres optimaux et la derniere étape consiste à sélectionner le meilleur sous arbre optimal.

2.3.1 Arbre Maximal

On commence d'abord par construire l'arbre maximal, on se place donc à la racine de l'arbre et il faut donc choisir la variable et la valeur de la variable où l'on va faire notre découpe : on a nos données X_i $i \in \{1, ..., p\}$, il nous faut donc un couple de valeur (i, d) tel que $\{X_i \leq d\} \cup \{X_i > d\}$ (notre découpe) avec des valeurs (i, d) qui satisfont un critère. On peut par exemple utiliser l'indice de

Gini. L'une des méthodes de recherche de la découpe optimale est la recherche de la réduction d'impureté :

$$s^* = arg\ max_s(\#Nd\phi(Nd) - (\#Fg\phi(Fg) + \#Fd\phi(Fd)))$$

Avec s^* la découpe optimale, s l'ensemble des découpes, $\#E$ le cardinal de l'ensemble E, Nd un noeud, Fg arbre fils gauche et Fd arbre fils droite.

Après la première découpe il faut réitérer cette action sur les 2 ensembles obtenus jusqu'à une condition d'arrêt. On obtient alors un arbre maximal qui présente quelques défaut (par exemple il est très touffu). On va alors construire un arbre Optimal.

2.3.2 Arbre Optimal

En partant de l'arbre maximal obtenu on peut construire un arbe optimal à l'aide de l'algorithme d'élagage de Breiman et al :

- On part de l'arbre T_{max} et on pose $\alpha_1 = 0$. On cherche alors les noeuds t (non terminaux) qui minimise $g_1(t) = \frac{Err(t) - Err(A_t^{(1)})}{|A_t^{(1)}| - 1}$. On appelera ce noeud $t^{(1)}$et on pose $\alpha_2 = g_1(t^{(1)})$ et $A^{(2)} = A^{(1)} - A_{t^{(1)}}^{(1)}$
- On effectue la même étape en partant de l'arbre $A^{(2)}$ jusqu'à obtenir $|A^{(q)}| = 1$

On a alors une suite de sous arbre optimaux $A^{(1)}, A^{(2)}, ..., A^{(q)}$. Pour obtenir le meilleur d'entre eux on peut choisir le sous arbre qui minimise l'erreur de validation ou on peut le choisir par validation croisée (dans le cas où l'échantillon est trop petit).

2.3.3 Application à nos données

Pour construire nos modèles d'arbre CART nous utilisons la fonction rpart du package du même nom. Avant l'utilisation de cette fonction nous devons transformer nos données en variables qualitatives à l'aide de la fonction "as.factor". Nous laissons le reste des paramètres par défaut à l'exception du paramétre Cp qui correspond à la complexité de l'arbre que nous laissons à 0 pour obtenir le modèle de l'arbre maximal.

Et pour la construction du modèle de l'arbre optimal on cherche dans le modèle de l'arbre maximal la valeur du Cp tel que l'erreur sur nos données d'entrées (la valeur de nos pixels) soit la plus petite et on rentre cette valeur de Cp à la place du 0 initialement prévu pour le modèle de l'arbre maximal.

Nous avons affiché nos 2 arbres à l'aide de la fonction prp du package rpart.plot.

De plus nous savons que les variables explicatives les plus importantes sont affichées sur les noeuds proches de la racine nous pouvons donc relever ces variables : x_213, x_115, x_77, x_75, x_24 (apparait 2 fois) et x_57.

2.4 Random Forest

La méthode des « forêts aléatoires » (ou Random Forest parfois aussi traduit par forêt d'arbres décisionnels) est une méthode de classification qui réduit la variance des prévisions d'un arbre de décision seul, améliorant ainsi leurs performances. Pour cela, il combine de nombreux arbres de décisions dans une approche de type bagging.

2.4.1 Définitions

Le bagging consiste à construire une suite d'arbres CART optimaux en faisant du bootstrap et agréger les prédictions fournies par les arbres, en faisant un vote majoritaire.

Le bootstrap est une méthode de rééchantillonnage qui permet de construire un nouvel échantillon de données à partir d'un échantillon observé.

2.4.2 Algorithme

Pour construire une forêt alétoire à N arbres on suit les étapes suivantes :
pour i de 1 à N

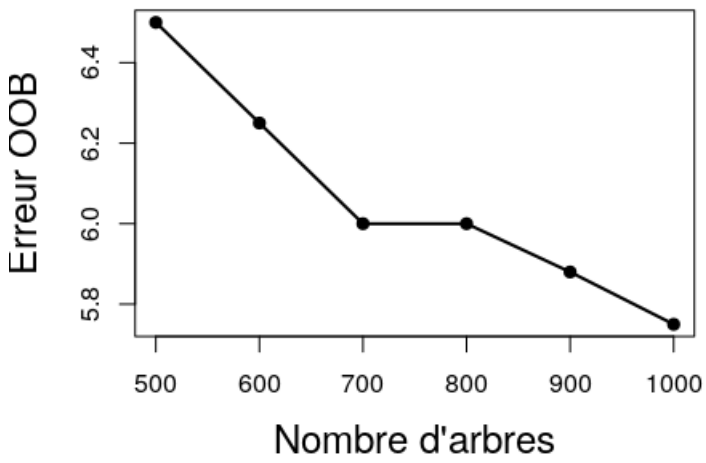
- construire un échantillon bootstrap Eapp* de Eapp
- construire un arbre CART maximal (pas d'élagage) à partir de cet échantillon bootstrap tel que :

→ à chaque noeud , on choisit la meilleure découpe sur la base de k variables choisis alétoirement parmi les p variables d'enrée.
fin pour.
On peut prédire notre Y en prenant la classe la plus populaire parmi les N arbres.

2.4.3 Implémentation dans les programmes

En R, les packages disponibles pour créer des modèles de random forest sont les packages randomforest.

Nous avons fait varier le paramètre nTree qui correspond au nombre d'arbres de la forêt RandomForest. Pour comparer la qualité des modèles, nous avons utilisé comme critère le taux d'erreur Out Of Bag. Nous avons donc fait varier le nombre d'arbre de 500 à 1000 et nous conservons le nombre qui permet d'avoir un taux d'erreur OOB le plus bas.



On peut voir que l'erreur OOB décroît lorsque le nombre d'arbres CART augmente. Ainsi, dans l'optique d'obtenir les meilleurs résultats possibles avec le modèle RF, nous avons choisi de fixer de manière permanente le paramètre nTree = 1000.

De plus nous avons pu voir nos variables "importantes" à l'aide de la fonction varImpPlot: c'est une méthode de sélection qui utilise en fait le critère de l'évolution de l'erreur quadratique moyenne lors de permutations des variables au hasard. Nous avons relevé les variables : x_24, x_120, x_105 ou encore x_213. (on peut remarquer qu'il y a des similitudes avec le modèle de l'arbre CART)

2.5 SVM

SVM (Support Vector Machine) est un algorithme d'apprentissage suppervisé, l'algorithme a pour but de séparer/classifier 2 classes différentes. Le principe consiste à estimer un hyperplan qui va séparer 2 classes, cet hyperplan sera lié à une fonction de décision qui permettra de classer nos données. La fonction est de la forme $f(x) = w^T x + b, w \in R^D, b \in R$.

Il arrive que cet hyperplan ne soit pas unique c'est là qu'intervient le terme de Marge :

- La marge géométrique représente la plus petite distance d'un point de nos donées à l'hyperplan.
- La marge numérique correspond à la plus petite valeur de la fonction de décision atteinte sur un point de nos données

On sait aussi [7] que l'hyperplan qui maximise la marge géométrique est un hyperplan qui minimise l'erreur. Donc on doit s'interesser à un problème de la

forme :

$$max_{w,b}\ min_{i\in\{1,...,N\}}\ distance(x_i, \Delta(w,b)) = \begin{cases} max_{w,b}\ m \\ \forall i \in \{1,...,N\} \quad \frac{|w^T x_i + b|}{||w||} \geq m \end{cases}$$

Aprés un changement de variable et en faisant attention aux contraintes on obtient le problème d'optimisation :

$$\begin{cases} min_{w,b} \frac{1}{2}||w||^2 \\ y_i(w^T x_i + b) \geq 1 \quad \forall i \in \{1,...,N\} \end{cases}$$

On peut résoudre ce problème en passant par son dual pour pouvoir généraliser au cas non linéaire, pour cela on passe par le Lagrangien et on tient compte des conditions de KKT. En effet on a à l'optimum : $w = \sum_{i=1}^{N} \alpha_i y_i x_i$, les α_i correspondent aux contraintes de Lagrange. Grace aux conditions de KKT (la condition de Complémentarité) w est défini tel que $y_i(w^T x_i + b) = 1$ et sont appelés vecteurs supports. Les vecteurs supports correspondent à un point par laquelle on peut tracer une droite parallèle à l'hyperplan et cette droite correspond à la marge géométrique.

En remplaçant le w dans le Lagrangien on se ramène alors au problème de Wolfe :

$$\begin{cases} min_{\alpha \in R^N} \frac{1}{2}\alpha^T G \alpha - e^T \alpha \\ \alpha_i \geq 0,\ \forall i = 1,...,N \\ y^T \alpha = 0 \end{cases}$$

avec G la matrice de Gram définie par $G = D_y X X^T D_y$ avec $D_y$correspondant à une matrice carré avec les y_i en diagonale et $e = (1,...,1)^T$. Ce résultat est vrai pour les données séparables, s'il ne l'est pas il faut introduire un facteur d'erreur donc le problème se réecrit :

$$\begin{cases} min_{\alpha \in R^N} \frac{1}{2}\alpha^T G \alpha - e^T \alpha \\ y^T \alpha = 0 \\ 0 \leq \alpha_i \leq C,\ \forall i = 1,...,N \end{cases}$$

C étant le facteur d'erreur que l'on doit nous même fixer.

2.5.1 Principe d'utilisation

Les étapes d'utilisation sont :

1. Centrer et réduire les données
2. Fixer le paramètre C
3. Résoudre le problème dual
4. En déduire les paramétre α_i et β_i

5. Calculer w et en déduire les points supports

6. Calculer la fonction de décision

2.5.2 Application à nos données

Sous R on utilise le package "e1071" pour l'utilisation des svm. Nous avons une utilisé une méthode de type One vs All en construisant nos modèles avec la fonction svm. On peut connaitre le nombre de points supports nécessaires pour construire la fonction de décision :

Chiffre	0	1	2	3	4	5	6	7	8	9
Nombre de points supports	773	749	672	441	535	445	798	718	658	426

On voit que selon la méthode des svm les chiffres les plus faciles à reconnaitre sont le 3, le 5 et le 9 (moins de la moitié des données en tant que points supports). Ensuite nous avons décidé de voir les cas linéaires et non linéaires.

La fonction svm du package "e1071" faisait tous les réglages tel que la séparation en multiclass entre autre, on a donc décidé de laisser les paramétres par defaut, sauf pour le cas linéaire où on a ajouté l'option "kernel=linear", pour construire nos modèles et les étudier.

2.6 Réseau de neurones

Les Réseaux de neurones permettent de faire de l'apprentissage supervisé non linéaire. Ils sont constitués d'une couche de neurones d'entrée, une couche cachée intermédiaire (ou parfois plusieurs) et d'une couche de sortie où l'on peut lire la variable prédite. On cherche à trouver les poids reliant les neurones de façon à minimiser une fonction de coût.
Le perceptron simple est dit simple parce qu'il ne dispose que de deux couches : la couche en entrée et la couche en sortie. Le réseau est déclenché par la réception d'une information en entrée. Le traitement de la donnée dans ce réseau se fait entre la couche d'entrée et la couche de sortie qui sont toutes les 2 reliées entre elles. Le réseau intégral ne dispose ainsi que d'une matrice de poids. Le fait de disposer d'une seule matrice de poids limite le perceptron simple à un classificateur linéaire permettant de diviser l'ensemble des informations obtenues en deux catégories distingues.
Le perceptron multicouche se structure de la même façon. À la différence du perceptron simple, le perceptron multicouche dispose, entre la couche d'entrée et la couche de sortie, d'une ou plusieurs couches dites « cachées ». Un perceptron multicouche est donc mieux adapté pour traiter les types de fonctions non-linéaires.

2.6.1 Algorithme

x correspond au vecteur des entrées et y au vecteur des sorties.
On initialise les poids W_i (les coefficients des arcs reliant les neurones) et le seuil T et on choisit une fonction de transfert ou d'activation qui permet de calculer

l'état du neurone. On initialise le parametre μ.
Pour chaque donnée d'apprentissage :
$((x_1, ..., x_{256}), y)$
$S = \sum_i^{256} W_i x_i - T$
$\hat{y} = f(S)$
Si $\hat{y} <> y$
alors pour tout $i \in \{1, , , 256\}$
$W_i = W_i + \mu(\hat{y} x_i)$
fin pour
Fin si
Fin pour
On peut remplacer un seuil non nul par une entrée supplémentaire constament égale à 1 avec un poids w_0=-T. On appelle ce poids le biais du neurone.
Il y a plusieurs fonctions d'activation: fonction gaussienne, fonction sinusoidale ....

2.6.2 Implementation dans les programmes

En R, les packages disponibles pour créer des modèles de réseaux de neurones sont les packages nnet et neuralnet, nous avons utilisé uniquement nnet. Dans nos programmes, on a joué sur les nombres de neurones dans la couche cachée:

- Pour le perceptron simple nous avons pris size = 0 et skip = True (pas de couche cachée donc connexion directe entre l'entrée et la sortie). La convergence est rapide. Ce sont des paramétres centraux dans ce package. Nous avons trouvé les résultats suivant : 100% de tbc pour yapp et 93% pour ytest. Nous avons estimés 3100 paramètres (poids) lors de l'uilisation de cette fonction avec ces paramètres.

- Pour le perceptron multicouche nous avons fait varier le nombre de neurones size afin d'obtenir le meilleur résultat . La convergence est cette fois moins rapide. Pour retrouver le résultat de perceptron simple il nous faut 20 neurones dans la couche cachée donc nous pouvons dire que le perceptron multicouche est plus couteux en terme de paramètre estimé. En effet nous avons estimés 5350 paramètres (poids) lors de l'uilisation de cette fonction avec ces paramètres.

$\rightarrow$ Pour les deux méthodes nous avons pris un decay (paramètre de régularisation du poids) faible (de l'ordre de 10^{-3}) et maxWTS (nombre maximal de poids estimés) grand (de l'ordre de 10^4).

2.7 Régression logistique polytomique

Lorsque la variable à expliquer n'est pas binaire, ce qui est notre cas ici, on peut adapter le modèle de régression logistique à un cadre polytomique. Dans le cas d'une variable y avec des modalités non ordonnées, on parle alors de modèle de régression multinomial.

2.7.1 Fondements Mathématiques

La construction d'un modèle de régression polytomique non-ordonnée se résume à rechercher les N+1 coefficients notés $\overset{k}{\alpha_0}$, ... , $\overset{k}{\alpha_N}$ tels que :

$$log(\frac{p_k(x)}{p_1(x)}) = \overset{k}{\alpha_0} + \overset{k}{\alpha_1} x_1 + ... + \overset{k}{\alpha_N} x_N$$

ce qui revient à

$$p_k(x) = \frac{exp(\overset{k}{\alpha_0} + \overset{k}{\alpha_1} x_1 + ... + \overset{k}{\alpha_N} x_N)}{1 + \sum_{k=2}^{m} exp(\overset{k}{\alpha_0} + \overset{k}{\alpha_1} x_1 + ... + \overset{k}{\alpha_N} x_N)}$$

On trouve l'expression suivante pour la vraisemblance du modèle :

$$L(\alpha, z) = \prod_{i=1}^{n} \prod_{k=1}^{m} p_k(x_i) \times 1_{\{x_i = u_k\}}$$

avec $z = (z_1, ..., z_n) \in \{u_1, ..., u_m\}^n$ et $\alpha = (\alpha_0^{(1)}, ..., \alpha_N^{(1)}, ..., \alpha_0^{(m)}, ..., \alpha_N^{(m)})$ et $x_i = (x_{1,i}, ..., x_{N,i})$.
On retrouve bien l'expression de la vraisemblance d'une loi multinomiale.

On va alors estimer les paramètres α avec la méthode du maximum de vraisemblance. L'algorithme de Newton-Raphson permet d'obtenir les estimateurs de tous les coefficients α_i, on les notera $\hat{\alpha_i}$.

Finalement, on peut obtenir l'estimation souhaitée de $p_k(x)$, $k \in \{2, ..., m\}$ avec la formule suivante :

$$\hat{p_k}(x) = \frac{exp(\overset{\hat{k}}{\alpha_0} + \overset{\hat{k}}{\alpha_1} x_1 + ... + \overset{\hat{k}}{\alpha_N} x_N)}{1 + \sum_{k=2}^{m} exp(\overset{\hat{k}}{\alpha_0} + \overset{\hat{k}}{\alpha_1} x_1 + ... + \overset{\hat{k}}{\alpha_N} x_N)}$$

Toutes les expressions précédentes correspondent à une généralisation des expressions de la régression logistique classique à une dimension plus importante.

2.7.2 Implementation dans les programmes

En R, les packages disponibles pour créer des modèles de régression polytomiques sont les packages nnet avec la fonction multinom ou vgam avec la fonction vglm. Comme il est possible en faisant varier quelques paramètres d'obtenir exactement les mêmes modèles avec les deux méthodes, nous avons choisi de privilégier celle qui serait la plus rapide en temps d'execution. Il s'agit de la fonction multinom, c'est donc celle qui nous a servi à créer nos modèles dans toute les parties du code liées à la régression polytomique.

2.7.3 Construction d'un modèle réduit

Pour sélectionner les variables les plus pertinentes à utiliser dans la construction d'un modèle réduit, on peut afficher leurs classement par ordre d'importance avec la commande "VarImplot". Cette méthode de sélection utilise en fait le critère de l'évolution de l'erreur quadratique moyenne lors de permutations des variables au hasard . On sélectionne alors celles qui sont affichées à l'écran. On vérifie bien que ce sont celles situées le plus haut lors de la construction de l'arbre optimal (voir paragraphe précédent). Pour utiliser ce modèle réduit, il suffira de préciser aux fonctions de R que l'on ne considère plus toutes les variables x_i (utilisation de la syntaxe .) comme variables explicatives mais plutôt les variables précédentes.

3 Résultats

3.1 Explication du code

Nous avons rassemblé l'implémentation des 10 méthodes précédentes de classification supervisée dans un unique fichier R. Pour éviter les fluctuations d'échantillonage par rapport au tirage des données d'apprentissage, nous avons effectué une boucle for permettant de tirer au hasard chaque fois un nouvel échantillon. Les données tests sont alors les données restantes parmi les données initiales. On va alors appliquer chaque méthode de manière répétée sur chaque set de données (apprentissage puis test) en retenant le taux de bon classement par rapport à la variable y. On obtient alors pour chaque méthode une suite de taux de bon classement chacun provenant d'un jeu de données différent. Il ne reste plus qu'à calculer la moyenne de ces taux de bon classement et on peut alors avoir une idée réaliste des performances de chaque méthode de manière globale, en terme de classification.

3.2 Tableau récapitulatif des taux de bon classement

Les méthodes testées sont les suivantes avec dans l'ordre d'apparition :

1. Regression Polytomique Complet
2. Regression Polytomique Reduit
3. One vs All (OvA)
4. LDA
5. Reseau de neurones (NNet)
6. SVM lineaire
7. SVM non lineaire
8. Arbre CART Maximal (TMax)
9. Arbre CART Optimal (TOpt)
10. Random Forest (RF)

Pour les résultats ci-dessous, chaque valeur numérique est une moyenne sur 100 valeurs. Il s'agit du taux de bon classement (en %) par rapport à la variable à expliquer y.

3.2.1 Données d'apprentissage

Modèle	Poly C	Poly R	OvA	LDA	NNet	SVM L	SVM NL	TMax	Topt	RF
TbC	100	100	100	98.63125	100	100	99.1025	83.26875	82.3225	100

3.2.2 Données test

Modèle	Poly C	Poly R	OvA	LDA	NNet	SVM L	SVM NL	TMax	Topt	RF
TbC	75.11	72.145	73.765	82.715	89.24	90.955	91.23	71.21	71.17	91.48

3.3 Analyses

On voit que toutes les méthodes comptent un fort taux de surapprentissage (compris entre 25 et 8%). Nous avons essayé de réduire ce phénomène en jouant sur le nombre de données d'apprentissage (par défaut à 800 donc avec 200 données test), en passant par exemple à 500 / 500 mais bien que cela ait eu un impact sur les résultats des méthodes, cela n'a pas permis de réduire le surapprentissage (voir prochaine partie).

D'autre part, on voit que les méthodes les plus performantes sont Random Forest et SVM. Les réseaux de neurone donnent aussi des bons résultats. On retrouve bien le fait que la méthode Random Forest donne de meilleurs résultats que les méthodes de l'arbre CART. En effet ce gain de performance provient de l'utilisation du boosting et de la selection aléatoire de k régresseurs à chaque noeud.

On remarque aussi que les méthodes Regression polytomique, One vs all et arbre CART qui sont les plus facilements interprétables (avec le moins de paramètre estimés) donnent les résultats les moins satisfaisants.

3.4 Test supplémentaire

Nous avons travaillé jusqu'ici avec 800 données d'apprentissage et 200 données de test car cela permet d'utiliser 80% des données lors de la construction des modèles. Cependant, ce ratio de 0.8 a été choisi de manière arbitraire. En effet, initialement nos données étaient séparées en deux fichiers .txt (un fichier d'apprentissage et un fichier de test) qui contenaient chacun les valeurs de pixels de 500 images. Autrement dit, le ratio entre données d'apprentissage et données test était de 50%. Comme nous avons observé un phénomène de surapprentissage généralisé entre toutes les méthodes avec 800 images d'apprentissage pour 200 images de test, nous avons pensé à faire varier ce taux.

Pour obtenir une vision globale de l'influence de ce paramètre, nous avons fait varier le nombre d'images d'apprentissage entre 500 et 800 (en augmentant à chaque fois de 25), tout en recalculant la moyenne des taux de surapprentissage (TBC sur App moins TBC sur Test) de toutes les méthodes sur des jeux de données tirés au hasard. Les résultats sont rassemblés dans le tableau suivant

	500	525	550	575	600	625	650
Reg Poly	26.2	22.94737	25.333333	23.294118	23.50000	23.200000	18.857143
LDA	23.6	21.95489	21.676768	19.304348	19.66667	18.773333	17.802198
Reseau Neurone	17.0	12.42105	15.111111	15.294118	13.00000	16.266667	14.000000
SVM Lin	11.0	11.36842	11.111111	9.411765	11.00000	10.666667	10.285714
Random Forest	9.8	10.10526	9.555556	8.705882	8.25000	7.200000	6.285714
OneVsAll	36.2	36.21053	35.111111	32.000000	30.00000	29.066667	30.285714
Tmax	17.4	18.39599	17.292929	15.959079	12.75000	12.320000	15.296703
Topt	16.8	17.29323	14.545455	11.120205	11.58333	11.893333	15.670330
Reg Poly Reduit	28.4	27.15789	26.888889	27.294118	28.75000	28.533333	27.428571
SVM Non Lin	11.8	11.42857	11.454545	10.598465	11.16667	9.066667	9.802198

675	700	725	750	775	800
23.692308	22.666667	24.000000	21.20000	21.777778	20.500
16.501425	15.857143	16.576803	14.40000	14.136201	15.375
15.076923	13.666667	12.363636	15.20000	13.333333	12.500
10.153846	9.333333	9.090909	9.20000	9.333333	10.500
6.461538	8.000000	7.636364	7.60000	7.555556	8.500
28.307692	26.000000	26.181818	24.40000	23.555556	24.500
13.629630	10.380952	10.896552	12.80000	12.315412	9.375
14.256410	11.285714	10.244514	12.93333	14.379928	10.500
28.615385	24.000000	24.727273	28.00000	25.333333	24.000
9.413105	9.142857	9.354232	8.80000	8.559140	9.250

Dans ce tableau on peut voir que les meilleurs taux de surapprentissage correspondent aux méthode de SVM linéaire (à 725 données d'apprentissage), SVM non linéaire (à 775 données d'apprentissage) et Random Forest (à 650 données d'apprentissage). On peut aussi voir que le taux de surapprentissage des méthodes CART varie aléatoirement.

On peut conclure que globalement l'augmentation du nombre des données d'apprentissage conduit à une réduction du taux de surapprentissage, cependant il y a des exceptions comme la méthode Random Forest dont le taux de surapprentissage remonte à partir de 675 données d'apprentissage. Cela confirme le choix de construire nos modèles avec 800 données d'apprentissage.

Conclusion

Pour ce problème de reconnaissance de chiffres, nous avons cherché au maximum à faire varier les paramètres de l'étude pour produire des analyses les plus complètes possibles. Nous avons ainsi entre autre rendu aléatoire et répété le tirage des données d'apprentissage, construit des modèles réduits en sélectionnant les variables les plus pertinentes ou encore cherché à optimiser le nombre de données d'apprentissage. Certaines de nos manipulations n'ont pas donné des résultats exploitables (ACP, Matrice de corrélation ...) à cause notamment du nombre de variables explicatives trop important. Pour chaque méthode à notre disposition, nous avons cherché à optimiser les paramètres sur lesquels nous pouvions influer comme le nombre de neurones pour les réseaux de neurones ou le nombre d'arbre CART dans le modèle de Random Forest. Nous avons finalement obtenu une comparaison en terme de taux de bon classement des différents modèles, ce qui a conduit à la sélection de méthodes dont les résultats sur les données tests avoisinent en moyenne 92%. On peut donc conclure que dans une optique de performance, il paraîtrait logique de sélectionner les modèles Random Forest et SVM non linéaire qui offrent les meilleurs résultats. Il est intéressant de noter que ces deux méthodes sont celles qui ont été mises au point le plus récemment.

Bibliographie

References

[1] https://cel.archives-ouvertes.fr/cel-01248297v2/document

[2] https://perso.univ-rennes2.fr/system/files/users/rouviere_l/chapitre4_glm.pdf

[3] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_SVM_R_Python.pdf

[4] https://medium.com/@ODSC/build-a-multi-class-support-vector-machine-in-r-abcdd4b7dab6

[5] https://www.rdocumentation.org/

[6] https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-rn.pdf

[7] cours machine learning GM4 INSA ROUEN