MASTER MIAGE 2ème année
Université Paris Nanterre

Mémoire de fin d'études

Étude comparative d'algorithmes pour le classement de chiffres manuscrits

Auteur :
Alexandre
PETIT-PAS

Tuteur :
PR. Marie-Pierre
GERVAIS

Résumé

Dans ce mémoire, nous présentons le classement, une méthode précise de l'apprentissage supervisé. Corriger des copies d'examen est une activité chronophage qui pourrait être automatisée. C'est pourquoi il est possible d'implémenter des outils permettant de corriger des questions, exercices ou QCM. Ce mémoire porte sur la partie reconnaissance de chiffres manuscrits permettant ainsi la reconnaissance des numéros étudiants. Trois algorithmes seront étudiés : CART, C4.5 et K-NN. Après analyse et implémentation, nous concluons que CART et K-NN sont deux solutions avec leurs avantages et leurs défauts. Finalement, K-NN sera retenu comme algorithme adéquat pour reconnaître les numéros étudiants efficacement dans un contexte de correction.

Remerciements

Je tiens à remercier Mme Marie-Pierre Gervais et M. Emmanuel Hyon pour l'aide et les conseils précieux qu'ils m'ont apportés durant la rédaction de ce mémoire.

Un grand merci à toute ma famille qui m'a soutenu jusqu'à la dernière ligne et qui m'a aidé à la relecture.

Enfin je tiens à remercier toutes les personnes qui m'ont aidé d'une quelconque manière à l'écriture de ce mémoire.

Sommaire

Introduction

Motivations

J'ai choisi ce sujet suite à trois événements. Durant l'un de mes stages, j'ai pu travailler avec un doctorant en informatique qui faisait des recherches sur de l'apprentissage. C'est aussi durant ce stage que j'ai approché pour la première fois l'informatique décisionnelle. Ce sujet m'a rapidement intéressé étant donné la grandeur et la popularité de celui-ci. Cependant, je ne voulais pas l'aborder sous le thème de la recommandation (qui est populaire mais a été traité de nombreuses fois), c'est pourquoi je me suis intéressé en profondeur à ce qui se faisait en terme d'arbre de décision.

C'est en découvrant l'existence d'algorithmes *lazy* et des *Lazy Decision Trees* que le sujet a été choisi. Les algorithmes paresseux, ou *lazy*, sont beaucoup utilisés dans le domaine du Web pour leur économie en ressource. Les ORM utilisent beaucoup cette technique qui leur permet de minimiser les requêtes vers la base de données. C'est pourquoi j'ai trouvé intéressant le fait d'apporter ce concept dans l'apprentissage supervisé.

Par ailleurs, durant mon stage de troisième année de licence, j'ai eu l'occasion de surveiller des partiels de première année (L1). Leurs épreuves étaient sous forme de QCM avec certaines réponses libres. Toutes les questions fermées étaient corrigées automatiquement alors que les réponses libres ne l'étaient pas. Un autre problème qui m'a marqué était l'écriture du numéro étudiant, composé de chiffres. Celui-ci n'était pas récupéré automatiquement car l'outil ne pouvait pas reconnaître les chiffres écrits à la main.

Ce sont ces trois idées qui m'ont permis d'écrire ce mémoire, afin de comprendre le domaine de l'apprentissage supervisé en répondant à un besoin d'enseignants universitaires.

Problématique

La reconnaissance d'images est un domaine populaire qui est devenu mature avec le temps. Des problématiques de performance continuent d'exister de différentes manières : l'image reconnue est-elle la bonne et l'algorithme est-il suffisamment performant pour supporter la charge ? Dans certains contextes, l'erreur peut coûter très cher et dans d'autres, il n'est pas possible de reconnaître efficacement l'image. Nous allons nous intéresser à la reconnaissance de chiffres provenant du MNIST. Cependant, il est possible par extension de l'appliquer sur d'autres jeux de données. La problématique est de savoir si les solutions recherchées sont suffisamment efficaces pour être

appliquées dans un cas concret de reconnaissance de numéros étudiants à des fins d'évaluation.

Objectifs

L'objectif de ce mémoire est de comprendre comment les techniques de classement fonctionnent et de les appliquer sur un ensemble de données connues. Les résultats permettront alors de connaître si un algorithme est plus performant qu'un autre et d'apporter une réflexion autour de l'amélioration de ceux-ci. Il est disponible à l'adresse suivante `https://github.com/AlexandrePetit-Pas/Memoire-M2`

Chapitre 1

Concepts formels

1.1 Introduction

L'objectif de ce chapitre est dans un premier temps de rappeler des notions de probabilités [1, 2, 3] qui serviront à la compréhension de ce document. Il s'agit ensuite d'expliquer dans sa généralité les concepts d'arbres décisionnels, afin d'introduire des principes d'apprentissage supervisé, de classement et d'optimisation.

1.2 Rappels de probabilité

1.2.1 Expérience aléatoire

Une expérience est dite aléatoire si les résultats possibles sont connus à l'avance sans vraiment savoir celui obtenu au préalable. On appelle univers, noté Ω, l'ensemble de toutes les issues possibles d'une expérience et ω une réalisation de l'expérience.

$$\Omega = \left\{\omega_1, \omega_2, \cdots, \omega_n\right\}, n \in \mathbb{N} \tag{1.1}$$

1.2.2 Événements

Un événement correspond à un ensemble de résultats possibles pour une expérience. Si cet ensemble est constitué d'un seul élément, on parle alors d'événement élémentaire. Si l'ensemble de résultats est égal à l'univers Ω, alors l'événement est dit certain. En revanche, si aucun résultat n'est présent (ensemble $\emptyset$), alors c'est un événement impossible.

1.2.3 Probabilités

Une probabilité est une fonction qui à un événement A, associe un poids.

$$\begin{cases} P(\Omega) = 1 \\ P(\emptyset) = 0 \\ 0 \leq P(A) \leq 1 \\ P(\bigcup_{i=1}^{n} A_i) = \sum_{i=1}^{n} P(A_i) \end{cases} \tag{1.2}$$

Plus la probabilité est proche de 1, plus il est possible que l'événement se réalise. Soit A et B deux événements quelconques, $P(A)$ est dite conditionnelle si son résultat est influencé par l'événement B :

$$\begin{cases} B \neq \emptyset \\ P(B) \neq 0 \\ A = (A \cap B) \cup (A \cap \overline{B}) \end{cases} \tag{1.3}$$

On peut alors déduire deux formules : le théorème de Bayes (1.4) permettant de calculer $P_B(A)$ et le théorème des probabilités totales (1.5) qui permet de connaitre la valeur de $P(A)$ à partir de deux événements A et B.

$$P_B(A) = \frac{P(A \cap B)}{P(B)} \tag{1.4}$$

$$P(A) = P_B(A)P(B) + P_{\overline{B}}(A)P(\overline{B}) \tag{1.5}$$

1.3 Arbre de décision

1.3.1 Définition

Un arbre de décision est une représentation structurelle qui permet d'aboutir à un choix. C'est un graphe acyclique orienté composé :

- d'un sommet sans parents appelé racine,
- de sommets appelés nœuds, correspondant à des tests,
- de sommets terminaux nommés feuilles,
- des arrêtes, ou branches, désignant chacune les résultats d'un test

Pour construire un arbre de décision, une approche *Top-Down* est utilisée, appelée *Top Down Induction of Decision Tree* [4] (*TDIDT*). Elle peut se décomposer en plusieurs parties :

1. Partir du jeu complet de données et construire la racine.
2. Réaliser un test afin de séparer les données.
3. Séparer le nœud actuel en fonction des résultats possibles.
4. Appliquer récursivement jusqu'à atteindre les feuilles.

Lorsque chaque nœud est composé exactement de deux descendants (hors feuilles), on parle alors d'arbre de décision binaire. C'est un cas très largement utilisé des algorithmes de construction d'arbres tels que CART [5], qui sera expliqué ultérieurement.

1.3.2 Exemple

Prenons comme exemple les données météorologiques présentes dans la table 1. Cette table est constituée de différentes colonnes concernant différentes informations ainsi qu'une colonne indiquant si la décision de sortir a été prise ou non. Dans

un premier temps, il est possible de déduire les tests à réaliser comme par exemple "La température est-elle élevée ?" ou encore "Quel est le temps ?". De ces questions découlent des réponses possibles, {oui, non} pour la première et {soleil, nuageux, pluie} pour la deuxième. Une fois l'approche *TDIDT* utilisée, un arbre de décision est alors obtenu (figure 1.1).

Jour	Temps	Température	Humidité	Vent	Sortie
1	soleil	élevée	haute	faible	N
2	soleil	élevée	haute	fort	N
3	nuageux	élevée	haute	faible	Y
4	pluie	moyenne	haute	faible	Y
5	pluie	basse	normale	faible	Y
6	pluie	basse	normale	fort	N
7	nuageux	basse	normale	fort	Y
8	soleil	moyenne	haute	faible	N
9	soleil	basse	normale	faible	Y
10	pluie	moyenne	normale	faible	Y
11	soleil	moyenne	normale	fort	Y
12	nuageux	moyenne	haute	fort	Y
13	nuageux	élevée	normale	faible	Y
14	pluie	moyenne	haute	fort	N

Table 1.1 – Table Météo

1.4 Arbre de décision et classement

1.4.1 Attributs et classes

Soit un ensemble d'observations S de taille n :

$$S = \{s_1, s_2, ..., s_n\}. \tag{1.6}$$

Chaque observation s_i est composée d'un ensemble de m attributs :

$$A_s = \{a_1, a_2, ..., a_m\}. \tag{1.7}$$

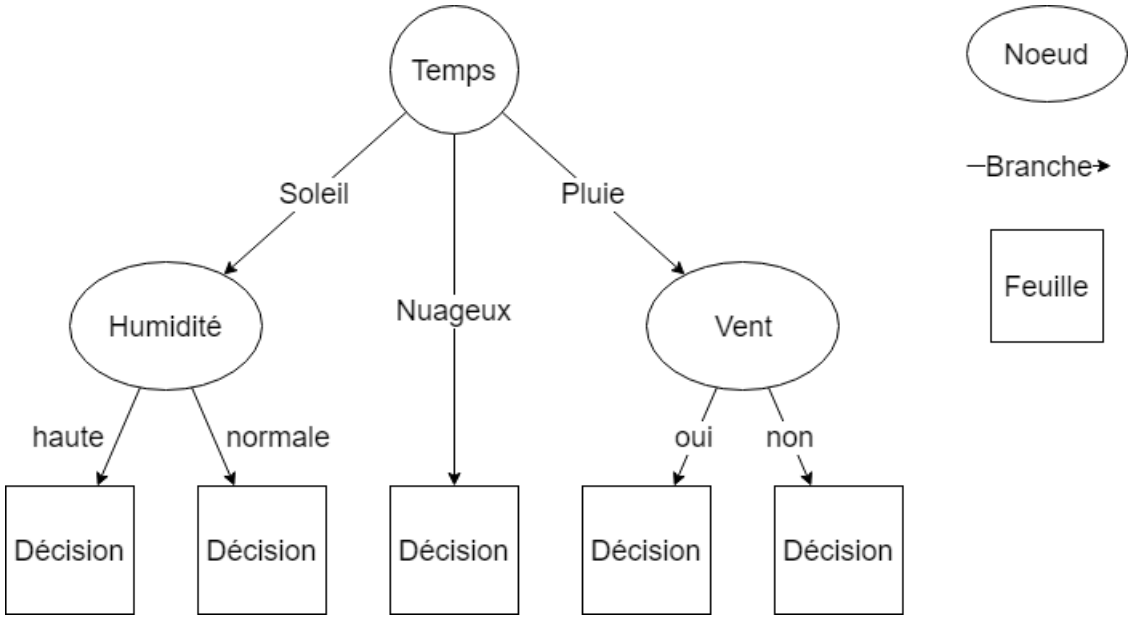

Figure 1.1 – Arbre de décision de la table Météo

Soit V_j l'ensemble de taille l des valeurs possibles de l'attribut a_j d'une observation s_i tel que :

$$V_j = \{v_1, v_2, ..., v_l\}. \tag{1.8}$$

Un attribut est dit qualitatif si l'ensemble des valeurs possibles est symbolique (non numérique), par exemple si V_j représente les couleurs d'écriture d'un mot. On obtiendrait alors $V_j = \{$ bleu, rouge, noir, vert $\}$:

- a_j est nominal si la notion d'ordre n'est pas présente dans l'ensemble des valeurs possibles, par exemple si $V_j = \{$ un, deux, trois, quatre, cinq, six, sept, huit $\}$ représente le nom d'un chiffre en toutes lettres.
- a_j est ordinal si les valeurs possibles contiennent la notion d'ordre. Cela peut être par exemple l'appréciation d'un client : $V_j = \{$ mauvais, bon, très bon $\}$

Une donnée est quantitative si l'ensemble des valeurs possibles est un ensemble numérique fini ou infini :

- Si un attribut peut prendre une infinité de valeurs dans son ensemble, alors celui-ci est qualifié de continu, par exemple le temps d'exécution d'un processus.
- Dans le cas contraire, une variable est dite discrète si l'ensemble des valeurs possible est fini. Elle est généralement liée à une énumération, comme par exemple le nombre de traits dans un caractère.

Qu'elle soit qualitative ou quantitative, une variable est qualifiée de binaire si l'ensemble V_j des valeurs possibles est de taille $l = 2$, par exemple $V_j = \{0, 1\}$ ou $V'_j = \{\text{vrai}, \text{faux}\}$

La classe d'une observation correspond à une "catégorie" et permet de se rapprocher ou de se différencier des autres observations. Elle correspond à une feuille dans un arbre décisionnel.

Reprenons la table 1.1 (Météo), l'attribut *Jour* correspond à l'identifiant d'une observation et donc n'est pas pris en compte. *Temps*, *Température*, *Humidité* et *Vent* sont des attributs qualitatifs. Parmi eux, deux sont binaires : $V_{Humidite} = \{haute, normale\}$ et $V_{Vent} = \{fort, faible\}$. Chaque observation possède une classe *Sortie* pouvant prendre Y ou N comme valeur.

1.4.2 Apprentissage supervisé

Le domaine de l'apprentissage peut se séparer en deux types : le supervisé et le non-supervisé.

Dans le cas de l'apprentissage non-supervisé, le but recherché est d'assembler les similitudes entre les observations. Une des méthodes les plus communes est le *clustering* [6].

Le second type d'apprentissage est dit supervisé. Dans ce cas, le but est d'apprendre du modèle pour arriver à déterminer la classe de l'observation. Ce modèle d'apprentissage est obtenu grâce à des données dont les classes sont connues. Il existe différentes applications de l'apprentissage supervisé, telles que les arbres de décision, les k plus proches voisins et les réseaux de neurones. Nous nous intéresserons uniquement aux arbres de décision et à la méthode des k plus proches voisins. Les jeux de données sont alors séparés en deux : une partie jeu d'entraînement (*training set*) et une partie jeu de test (*test set*). Le but est d'apprendre du jeu d'entraînement afin de classer les observations du jeu de test.

Soit C un ensemble de classes de taille n :

$$C = \{c_1, \cdots, c_n\} \tag{1.9}$$

Classer une observation s_i revient à trouver le c_j qui lui correspond le mieux par le biais d'une fonction de classement F, calculée grâce au jeu de données d'entrainement.

1.4.3 Quantification de l'information

La composante principale des arbres de décision est la quantification de l'information. "*Dans la théorie de l'information, les notions de quantité d'information et d'incertitude sont équivalentes*" [2, 7]. Partant de ces concepts, la quantité d'information $h(x)$ d'une probabilité $P(x)$ est une fonction croissante : si $P(x)$ augmente, $h(x)$ augmente. Par ailleurs, les événements certains et impossibles n'apportent aucune information

étant donné que le résultat est connu d'avance.

Afin de mesurer le gain d'information, il est nécessaire d'introduire le concept d'entropie correspondant à l'impureté d'une observation en théorie de l'information [8]. Soit S un ensemble d'observations de taille n où $s_i = \{\text{Y}, \text{N}\}$. L'entropie E de S, où p_Y correspond à la probabilité d'avoir Y et p_N d'avoir N [9], est définie par

$$E(S) = -p_Y log_2 p_Y - p_N log_2 p_N \tag{1.10}$$

La figure 1.2 représente la fonction d'entropie de S en fonction de p_Y. On observe que l'entropie est comprise entre 0 et 1 et vaut 0 lorsque p_Y vaut 0 ou 1. Cela confirme le fait que les événements impossibles et certains n'apportent aucun gain d'information.

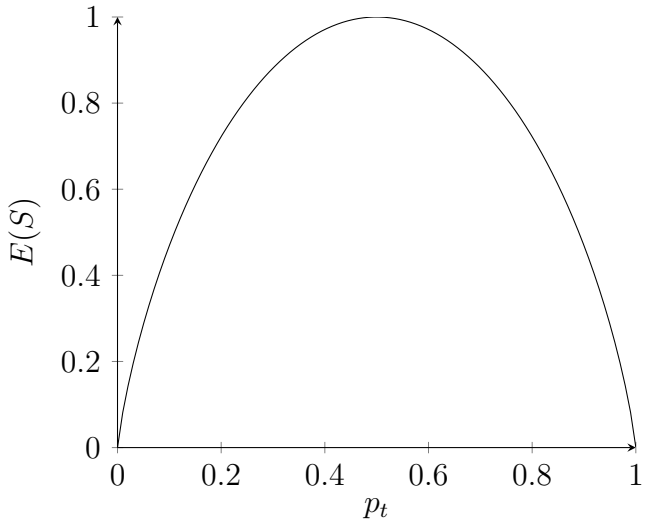

Figure 1.2 – Entropie de S en fonction de p_t

Nous venons de montrer l'entropie de S pour un attribut binaire. Par extension, si une observation s_i peut prendre l valeurs, alors

$$E(S) = \sum_{i=1}^{l} -p_i log_2 p_i \tag{1.11}$$

Par exemple, d'après la table 1.1, l'entropie de l'attribut *Sortie* est égale à

$$\begin{aligned} E(Sortie) &= -\frac{9}{14} log_2(\frac{9}{14}) - \frac{5}{14} log_2(\frac{5}{14}) \\ &= 0.4098 + 0.5305 \\ &= 0.940 \end{aligned} \tag{1.12}$$

A partir de cette entropie, il est possible d'obtenir une mesure appelée gain d'information. Celle-ci correspond à la réduction de l'entropie et permet de quantifier l'apport

en information lors du choix d'un test. Elle est définie [9] par

$$G(S, A) = E(S) - \sum_{v \in V_A} \frac{|S_v|}{|S|} E(S_v) \tag{1.13}$$

Par exemple, afin de créer l'arbre de décision de la table Météo, il faut déterminer l'attribut qui sera la racine de l'arbre. Pour cela, il faut déterminer quel attribut maximise le gain d'information :

$$\begin{aligned}
G(S, Temps) &= E(S) - \frac{5}{14} E(S_{T,Soleil}) - \frac{4}{14} E(S_{T,Nuageux}) - \frac{5}{14} E(S_{T,Pluie}) = 0.246 \\
G(S, T°) &= E(S) - \frac{4}{14} E(S_{Tp,Elev}) - \frac{6}{14} E(S_{Tp,Moy}) - \frac{4}{14} E(S_{Tp,Basse}) = 0.030 \\
G(S, Humidite) &= E(S) - \frac{7}{14} E(S_{H,Haute}) - \frac{7}{14} E(S_{H,Normale}) = 0.152 \\
G(S, Vent) &= E(S) - \frac{6}{14} E(S_{V,Fort}) - \frac{8}{14} E(S_{V,Faible}) = 0.048
\end{aligned} \tag{1.14}$$

D'après les résultats obtenus :
$G(S, Temps) > G(S, Humidite) > G(S, Vent) > G(S, T°)$. L'attribut *Temps* sera donc choisi comme racine de l'arbre de décision. Ce processus sera ensuite répété pour chaque nouveau nœud crée jusqu'à ce que toutes les feuilles soient atteintes (figure 1.1).

1.5 Optimisation d'arbre de décision

1.5.1 Problème d'*overfitting*

Un des principaux problèmes des arbres de décision est le sur-ajustement (*overfitting*) : la taille d'un arbre augmente linéairement avec la taille du jeu de données d'apprentissage. Ce principe d'*overfitting* peut être défini de la manière suivante : "*soit H un espace d'hypothèses et $h \in H$, h sur-ajuste le jeu d'apprentissage s'il existe une alternative $h' \in H$, tel que h ait une marge d'erreur plus petite que h' sur les jeux d'apprentissage, mais une marge d'erreur plus grande sur la totalité du jeux de données.*" [9] On peut voir sur la figure 1.3 que plus la précision du jeu d'apprentissage augmente, plus la précision du jeu de test diminue. C'est pourquoi des méthodes d'élagage et d'amélioration d'arbre sont mises en place afin de minimiser ce problème.

1.5.2 Élagage

Pré-élagage

Le pré-élagage consiste à arrêter prématurément la construction de l'arbre même si chaque feuille ne correspond pas à une et seulement une classe. Certains nœuds ne

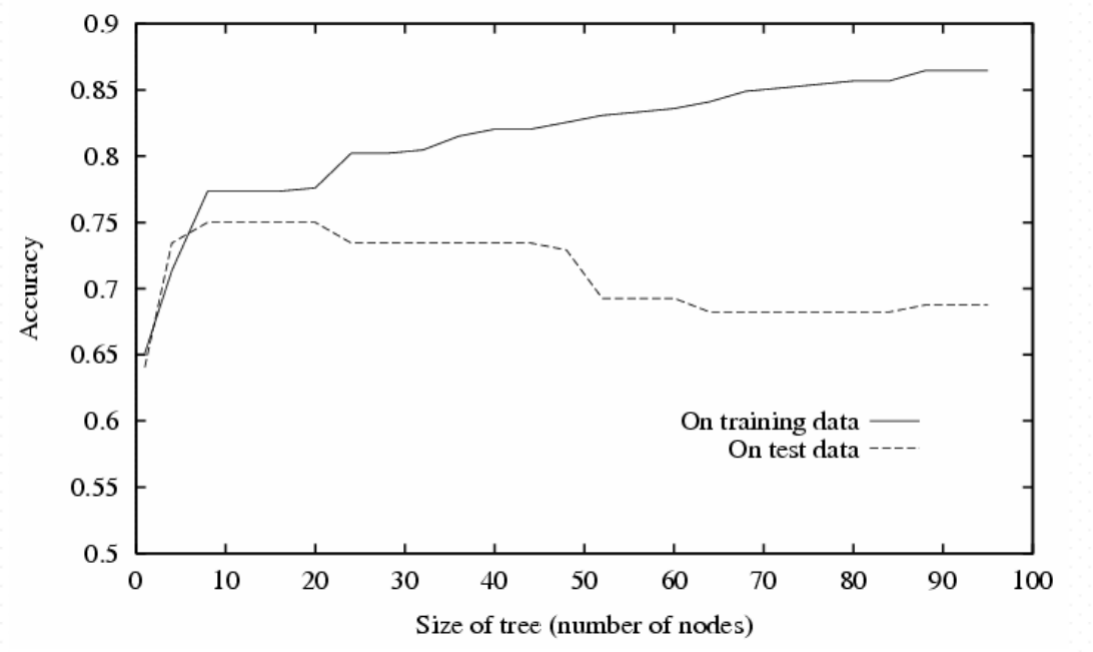

Figure 1.3 – *Overfitting* d'un arbre de décision sur une base de données médicales afin de déterminer si les patients sont atteints de diabète. [9]

seront alors pas plus développés, selon différents critères d'arrêt choisis au préalable.

Une première méthode consiste à définir un nombre d'objets minimum par nœud. Si lors de la création de nouveaux nœuds, l'un d'eux ne dépasse pas le seuil, alors ceux-ci ne sont pas créés et le nœud parent devient une feuille de l'arbre. Une deuxième méthode utilise le test du χ^2 afin de déterminer s'il est statistiquement pertinent de sélectionner un attribut pour construire la suite de l'arbre. Il existe par ailleurs d'autres méthodes qui consistent à définir des seuils pour arrêter la construction [2, 10] :

- le seuil maximum de feuilles a été atteint,
- le seuil maximum d'attributs a été franchi,
- le gain d'information est inférieur à celui fixé.

Post-élagage

Contrairement au pré-élagage, le post-élagage se place une fois la construction de l'arbre faite. L'objectif est de supprimer des sous-ensembles de l'arbre afin de les remplacer par des feuilles en minimisant l'impact sur le taux d'erreur, voire réduire le taux d'erreur. Il existe différentes manières d'élaguer un arbre de décision, quatre d'entre elles seront étudiées [11].

Une première méthode de post-élagage est le *reduced error pruning* [12]. Elle compare le nombre d'erreur de classement entre le sous-arbre T_t intact et le sous-arbre T'_t lorsque t est transformé en feuille. Si T'_t est plus performant, alors T_t est élagué et

t devient une feuille de l'arbre.

Le *pessimistic error pruning* est une méthode d'élagage proposée par Quinlan [12]. Elle remplace le nombre J d'observations mal classées sur une feuille par $J + \frac{1}{2}$. D'après lui, le rapport $\frac{J}{K}$ d'observations mal classées, où K est le nombre d'observations sur une feuille, est une estimation optimiste pas assez fiable, c'est pourquoi il a choisi de la remplacer par une distribution binomiale [9, 12].

La troisième méthode de post-élagage est le *cost complexity pruning*. Elle consiste à construire une séquence d'arbre $T_0, T_1, \ldots, T_L$ en minimisant la valeur de α correspondant à la moyenne du nombre d'observations mal classées par feuille, T_0 étant l'arbre initial et T_L l'arbre final. T_{i+1} est obtenu en élaguant tous les nœuds de T_i qui ont la plus petite valeur de α [11].

Enfin, le *minimum error pruning* permet de minimiser le taux d'erreur attendu d'un sous-arbre élagué. Si celui-ci est inférieur au taux d'erreur du sous-arbre sans élagage, alors le nœud sera remplacé par une feuille. Ce taux d'erreur est définit par

$$E_k = \frac{(n - n_j + k - 1)}{n + k} \tag{1.15}$$

où n correspond au nombre d'observations dans une feuille, n_j observations appartenant à la classe j et k le nombre de valeurs de la classe [11].

Chapitre 2

Classement supervisé

2.1 Introduction

L'objectif de ce chapitre est de mettre en avant les méthodes les plus utilisées dans l'apprentissage supervisé. Ainsi, deux méthodes de construction d'arbre de décisions, CART et C4.5, seront présentés ainsi qu'une autre méthode appelée *K-Nearest Neighbors*. Cela permettra ensuite de mieux comprendre le fonctionnement de l'apprentissage supervisé et pouvoir par la suite mettre en œuvre ces algorithmes.

2.2 K Nearest Neighbors

2.2.1 Concept

La méthode des k plus proches voisins (*K-Nearest Neighbors ou K-NN*) est une méthode paresseuse d'apprentissage supervisée principalement utilisée pour le classement. "*Le problème des K-NN est de créer une structure de données pour un ensemble d'objets qui, étant donné un objet q, son plus proche voisin dans l'ensemble peut être trouvé rapidement.*" [13] L'objectif est de placer les observations du jeu d'apprentissage sur un espace métrique. Naturellement, ces observations seront majoritairement regroupées vers un même espace. Lors de l'ajout d'une nouvelle instance, l'algorithme va analyser quels sont ses k voisins les plus proches afin de déterminer la classe à attribuer. C'est un algorithme *lazy* car celui-ci ne fait que très peu de traitement durant la phase d'apprentissage, les calculs sont faits durant la phase de classement des données. Afin de faciliter ce calcul de distance, les données sont normalisées, ce qui simplifie le traitement des données continues mais complexifie le traitement des données qualitatives.

La figure 2.1 montre un ensemble de données [1] contenant deux attributs $x \in [0, 30]$ et $y \in [0, 30]$ et deux classes, l'une rouge et l'autre jaune. Il est clairement visible que les points rouges se trouvent sur la partie supérieure gauche du graphique tandis que les points jaunes, sur la partie inférieure droite (figure 2.2). Si une nouvelle instance apparaissait, $\forall k \in \mathbb{N}$, celle-ci serait relativement simple à classer.

1. Disponible à l'adresse suivantes : `https://www.kaggle.com/alqamahjsr/topic-3-decision-trees-and-knn`

En revanche, avec *K-NN*, le choix du k est arbitraire et celui-ci peut influencer le résultat obtenu.

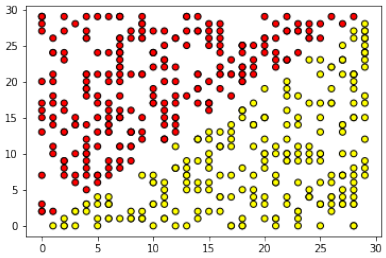

Figure 2.1 – Représentation graphique de points ayant pour classe "rouge" ou "jaune"

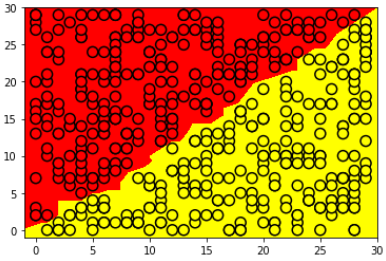

Figure 2.2 – Séparation des données par leur classe

2.2.2 Choix de k

k est une valeur arbitraire choisie avant le lancement de l'algorithme. Cette valeur permet de choisir le nombre de voisins à regarder afin de déterminer la classe d'une observation. Cela signifie que si k est trop petit, l'algorithme renverra la valeur de l'observation la plus proche. Un petit k générera alors un modèle trop précis, qui engendrera un problème d'*overfitting*. Dans le cas contraire, un trop grand k pourrait réduire la précision du modèle et rendre le comportement du modèle trop abstrait [6].

Par exemple, la figure 2.3 [2] met en avant le problème du choix de k. Cette figure possède une observation de test (cercle vert) à classer soit en carré bleu, soit en triangle rouge. Dans le premier cas, $k = 3$, l'algorithme va donc regarder les trois voisins les plus proche du cercle : deux triangles rouges et un carré bleu et déduire que l'observation est un triangle rouge. Dans le deuxième cas, $k = 5$, les cinq plus proches voisins sont deux triangles rouges et trois triangles bleus : l'observation sera alors un carré bleu. Dans cet exemple, un k trop grand a changer la classe en carré bleu alors que celle-ci semble être plus proche des triangles rouges.

Afin d'améliorer *K-NN*, il est possible de donner un poids aux voisins et aux attributs. Ainsi, un voisin proche aura plus d'importance qu'un voisin éloigné qui risque de ne pas être de la même classe. Par ailleurs, un attribut n'étant pas pertinent aura beaucoup moins d'importance qu'un attribut aidant au classement [6].

2. Antti Ajanki AnAj - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=2170282`

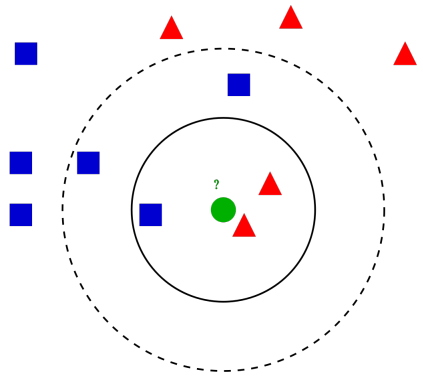

Figure 2.3 – Exemple de K-NN avec deux valeurs différentes de k

2.3 CART

Classification And Regression Trees (CART) est une méthode développée par Breiman et al. [14]. Cette approche permet de créer des arbres binaires (chaque nœud a deux descendants directs). Chaque test, ou *split*, correspond à la question binaire qui maximise la réduction de l'entropie. Pour cela, l'algorithme va parcourir tous les attributs de chaque nœud et garder le meilleur pour en dégager le *split* maximisant la réduction de l'entropie. Il va ensuite comparer les *splits* pour garder le meilleur. L'algorithme 1 montre le procédé suivant le choix du split. Les données sont alors séparées en deux : celles qui répondent "vrai" au test d'un côté et celles qui renvoient "faux" de l'autre. Ces actions sont répétées autant de fois que nécessaire jusqu'à atteindre les feuilles de l'arbre ou un autre critère d'arrêt.

Data : jeu de données
Result : arbre de décision
calcul du meilleur split et de son gain d'information ;
if *gain d'information = 0* **then**
    renvoie une feuille contenant les données ;
else
    récupérer les données "vraies" et "fausses" du split ;
    retour au début en donnant les données "vraies" comme paramètre ;
    retour au début en donnant les données "fausses" comme paramètre ;
end

Algorithm 1 : Algorithme CART

Dans le cas où les attributs ne sont pas binaires, les questions posées sont alors de la forme :

- $x \leq 0.9$?
- $v_1 \in V = \{v_1, v_2, v_n\}, \forall n \in \mathbb{N}, x = v_1$?
- $\forall i < n, x \in \{v_1, \ldots, v_i\}$?

Chaque possibilité sera étudiée ce qui peut augmenter rapidement la complexité de l'algorithme si les nombres d'attributs ou de valeurs sont grands. C'est pourquoi l'algorithme CART n'est pas adapté aux valeurs continues.

Afin d'améliorer la précision de l'arbre et réduire l'*overfitting*, il est possible d'utiliser des méthodes de pré-élagage et de post-élagage. Cependant, il n'existe pas de méthode prédéfinie. Il faut alors en choisir une (ou plusieurs) lors de l'implémentation de l'algorithme.

2.4 C4.5

C4.5 est un algorithme inventé par Quinlan [15]. Comme pour CART, l'algorithme va parcourir chaque nœud récursivement et trouver le meilleur *split* à chaque fois. Cependant, celui-ci ne se limite pas aux arbres binaires et peut donc avoir *n* nœuds descendants. Pour des attributs qualitatifs, une branche sera créée pour chaque valeur possible. Pour choisir le meilleur *split*, l'algorithme va utiliser le gain d'information (équation 1.13). Pour les valeur continues, l'algorithme utilise un seuil qui permet de diviser les valeurs en deux : le nœud de gauche comprendra toute valeur inférieur à ce seuil tandis que le nœud de droite, toutes les valeurs supérieures. C4.5 est par conséquent plus performant que CART dans la gestion des valeurs continues.

Une fois l'arbre construit, une fonction d'élagage (algorithme 2) est alors appelée afin d'améliorer les résultats. Cette méthode, appelée *error-based pruning*, est une amélioration du *pessimistic pruning* [15]. Selon Quinlan, E objets mal classés sur N observations amènent à un taux d'erreur optimiste de $\frac{E}{N}$. La borne supérieure $U_{CF}(E, N)$ de l'indice de confiance CF peut être trouvée selon les limites de confiance de la distribution binomiale [15], ce qui correspond au taux d'erreur prédit pour une feuille. Ce niveau de confiance est une valeur donnée et vaut 25%. Le but de l'algorithme est de comparer le taux d'erreur de chaque feuille avec les feuilles des sous-arbres correspondant. Si le taux d'erreur du sous-arbre (nœud parent) est inférieur à celui des feuilles, alors on élague le sous-arbre par une feuille.

Data : E objets mal classés, N observations
Result : arbre
while *sous-arbres à parcourir* **do**
  erreur feuilles $\leftarrow \sum N \times U_{25}(E, N)$;
  erreur nœud parent $\leftarrow N' \times U'_{25}(E', N')$;
  if *erreur feuilles > erreur nœud parent* **then**
    nœud parent $\leftarrow$ feuille ;
  else
    nœud parent inchangé ;
  end
end

Algorithm 2 : Élagage de l'algorithme C4.5

Chapitre 3

Apprentissage du MNIST

3.1 Introduction

L'objectif de ce chapitre est d'implémenter et analyser les méthodes décrites précédemment afin de déduire l'efficacité des algorithmes dans le cas de la base du MNIST. Il est nécessaire dans un premier temps d'expliquer la base de donnée MNIST et de situer la problématique du mémoire afin d'introduire par la suite la méthode de travail. Une fois les observations faites, une comparaison des résultats sera effectué pour en dégager la méthode la plus pertinente pour répondre au problème.

3.2 Définition du MNIST

MINST (*Modified National Institute of Standards and Technology*) est une base de données de chiffres écrits à la main produite par LeCun et al. [16]. Elle s'inspire d'une autre base de données, créée par le NIST (*National Institute of Standards and Technology*), qui possédait des caractères alphanumériques écrits à la main. La base MNIST est composée de 60 000 images d'apprentissage et 10 000 images de test, chacune en noir et blanc et d'une taille de 28x28 pixel. Une technique d'anticrénelage a été appliquée sur les images afin de leur donner un niveau de gris [16]. Une image donne alors un vecteur de dimension 784 dont chaque valeur est comprise entre 0 (blanc) et 255 (noir).

La base MNIST est une base qui sert de standard dans les domaines de l'apprentissage de part sa disponibilité, sa taille et le fait que les données soient des caractères écrits à la main [17]. La figure 3.1 montre un exemple d'une observation du jeu de données, qui est un chiffre 5.

Figure 3.1 – Exemple d'un chiffre de la base MNIST

3.3 Méthode de recherche

En utilisant MNIST, l'objectif est de déterminer quelle méthode d'apprentissage permettra d'obtenir un classement avec la meilleure précision tout en gardant un temps d'exécution relativement raisonnable.

Cette recherche se place dans un contexte de classement de chiffres afin de reconnaître les numéros étudiants écrits sur une feuille d'examen. Il est donc envisageable d'avoir une erreur lors de la prédiction car le correcteur peut manuellement faire les changements nécessaire dans ce cas. En revanche, plus la précision est importante, plus l'algorithme sera fiable et économisera du temps lors d'une correction. Par ailleurs, le temps d'exécution de l'algorithme n'est pas un facteur majeur mais si celui-ci devient trop conséquent alors la correction assistée sera plus longue qu'une correction manuelle. Dans ce cas, l'algorithme n'est pas pertinent.

Deux algorithmes ont été choisis afin de réaliser les tests : K-NN et CART.

Le premier, K-NN, a été sélectionné car il permet de tester l'approche paresseuse des classements. La problématique du choix de la valeur de k se résoudra par un choix arbitraire : l'algorithme sera testé avec $k = 1$, $k = 3$, $k = 5$, $k = 10$. Les explications précédentes (3.2.2) laissent à penser que $k = 10$ sera trop grand, ce qui réduira la précision des résultats. Au contraire, $k = 1$ aura une bonne précision mais une marge d'erreur plus grande à cause de l'*overfitting*.

Le deuxième algorithme sélectionné est CART. Celui-ci représentera la famille des arbres de décision. L'avantage du traitement des attributs de C4.5 n'est pas exploité avec les données du MNIST car celles-ci sont toutes quantitatives. CART sera par conséquent préféré.

Pour répondre à des problèmes de mémoire, le nombre de données d'apprentissage sera réduit 30 000 quant au jeu de test, celui-ci sera réduit à 5000. De plus, une autre série de test sera faite sur un échantillon de 30 000 puis 10 000 d'apprentissage pour 800 de test. Les jeux de test du MNSIT sont supportables par une machine mais ceux d'apprentissage sont trop lourds à charger en mémoire. La deuxième série de test sera réalisé dans le but de se rapprocher le plus possible du cas d'une promotion de cent étudiants en moyenne avec chacun un numéro étudiant composé de huit chiffres. Le choix d'un échantillon d'apprentissage plus petit (10 000 observations) et de voir si la perte de précision du au manque de cas est négligeable.

Parmi les outils utilisés, Scikit-learn [18] permet d'implémenter les algorithmes cités précédemment. Afin de faciliter son utilisation, il a fallu convertir les fichiers de la base MNIST du format IDX, qui est fait pour les vecteurs et les matrices multidimensionnelles, en format CSV.

3.4 Analyse des résultats

Les tableaux 3.1, 3.2, 3.3 et 3.4 montrent les résultats obtenus suite aux implémentations. La colonne ”Précision” correspond au quotient de réussite du classement de l'algorithme. La colonne ”Temps” correspond au temps d'exécution du programme, du chargement des données au calcul de la précision. Cette métrique est influencée par les performances de l'ordinateur et de l'allocation donnée au programme. Il faut donc regarder l'ordre de grandeur de celle-ci plutôt que la valeur exacte.

Quatre tests ont été réalisés pour chaque algorithme :

- ”Test 1” correspond à 30 000 données d'apprentissage pour 5 000 de test.
- ”Test 2” correspond à 30 000 données d'apprentissage pour 800 de test.
- Le ”Test 3” est composé de 10 000 observations d'apprentissage pour 5 000 de test.
- ”Test 4” est composé de 10 000 observations d'apprentissage pour 800 de test.

	Test 1		Test 2	
k	Précision	Temps	Précision	Temps
$k = 1$	0.9614 ($\pm$ 0.0079)	1 147s	0.9614 ($\pm$ 0.0079)	969s
$k = 3$	0.9608 ($\pm$ 0.0064)	1 122s	0.9608 ($\pm$ 0.0064)	965s
$k = 5$	0.9600 ($\pm$ 0.0075)	1 162s	0.9600 ($\pm$ 0.0075)	970s
$k = 10$	0.9561 ($\pm$ 0.0081)	1 130s	0.9561 ($\pm$ 0.0081)	980s

Table 3.1 – Résultats de K-NN avec 30 000 observations d'apprentissage

	Test 3		Test 4	
k	Précision	Temps	Précision	Temps
$k = 1$	0.9392 ($\pm$ 0.0146)	196s	0.9392 ($\pm$ 0.0146)	133s
$k = 3$	0.9395 ($\pm$ 0.0108)	188s	0.9395 ($\pm$ 0.0108)	132s
$k = 5$	0.9377 ($\pm$ 0.0133)	194s	0.9377 ($\pm$ 0.0133)	134s
$k = 10$	0.9339 ($\pm$ 0.0161)	190s	0.9339 ($\pm$ 0.0161)	134s

Table 3.2 – Résultats de K-NN avec 10 000 observations d'apprentissage

Test 1		Test 2	
Précision	Temps	Précision	Temps
0.8469 ($\pm$ 0.0179)	54s	0.8481 ($\pm$ 0.0157)	53s

Table 3.3 – Résultats de CART avec 30 000 observations d'apprentissage

Test 3		Test 4	
Précision	Temps	Précision	Temps
0.8180 ($\pm$ 0.0150)	17s	0.8125 ($\pm$ 0.0155)	17s

Table 3.4 – Résultats de CART avec 10 000 observations d'apprentissage

Les tableaux 3.1 et 3.2 montrent que K-NN a une précision comprise entre 0.93 et 0.96. Dans ce domaine, il peut être qualifié d'efficace car la probabilité de mal classer un chiffre reste faible. En revanche, le temps d'exécution de l'algorithme devient relativement long lorsque le nombre d'échantillon d'apprentissage est grand : il faut entre quinze et vingt minutes pour classer le jeu de données de test suivant la taille de celui d'apprentissage. Par ailleurs, il est intéressant de noter que la précision de l'algorithme ne diminue pas avec la taille de l'échantillon de test puisque celui se base principalement sur les attributs du jeu de données et non sa taille. Avec une distance fixe entre les voisins, la précision de l'algorithme ne varie donc pas en fonction de la taille de l'échantillon de test. Les résultats de CART montrent un temps d'exécution beaucoup plus court que K-NN, en revanche, la précision est plus faible. La plus haute précision est à 0.84 tandis que la plus basse, 0.81.

3.5 Choix de l'algorithme

Les deux algorithmes peuvent être fondamentalement préférés. L'un permet d'avoir une solution rapidement tandis que l'autre apporte une précision plus fine. Dans le cadre d'une détection de numéros étudiant d'une promotion, qui est composée de cent étudiants dans ce document, l'algorithme devrait être intégré à une solution qui permettrait de faire plus que reconnaître les chiffres. Que ce soit pour corriger automatiquement ou agréger des exercices entre eux, on peut considérer le fait qu'un tel outil est créé afin d'accélérer la correction des examens papiers. C'est pourquoi avoir un algorithme prenant un peu plus de temps qu'un autre mais qui est plus précis semble plus pertinent pour le correcteur. Dans ce cas, K-NN est l'algorithme à retenir avec $k = 3$. Cette valeur de k a été choisie car c'est avec celle-ci que la précision de l'algorithme est la plus grande tout en prenant en compte l'*overfitting*.

Conclusion

Le domaine de l'apprentissage est un champs d'étude particulièrement utilisé. De la reconnaissance d'image à la recommandation, il a fait l'objet de nombreuses recherches. Dans ce mémoire, nous avons abordé la partie supervisée de cet apprentissage et plus particulièrement, la notion de classement. Celle-ci est basée sur des concepts de probabilité, tels que le théorème de Bayes ou le théorème des probabilités totales. Par ailleurs, le cas le plus représenté est celui de l'arbre de décision. Nous avons vu qu'il était possible de quantifier l'information afin de déterminer une classe à une instance. Cependant le problème d'*overfitting* rend cette décision de moins en moins fiable. C'est pourquoi des solutions d'élagage sont alors apparus afin de réduire l'impact de l'*overfitting*.

Dans un deuxième temps, nous avons vu trois algorithmes de classement : *K-Nearest Neighbors*, *Classification And Regression Trees* et C4.5. Chacun possède ses avantages et ses inconvénients et suite à une mise en pratique, il a été possible de les mettre en valeur. Grâce au MNIST, il a été possible de montrer que K-NN est précis mais couteux tandis que CART est rapide mais pas assez fiable. Dans notre problématique de reconnaissance de numéros étudiant, il a été décidé de retenir K-NN. Celui-ci semble plus adéquat étant donné la qualité du classement et l'absence de nécessité à avoir une solution particulièrement rapide.

Il existe cependant de nombreuses autres techniques d'amélioration des algorithmes (*bagging*, *boosting*) ainsi que d'autres méthodes (*Lazy Decision Tree*, *Random Forest*, algorithmes génétiques, etc.) mais celles-ci font l'objet de recherches complémentaires. Il n'existe pas de solution universelle dans l'apprentissage supervisée. La réalité fait qu'il est souvent très compliqué d'appliquer toutes ces notions à des cas existants. Dans le nôtre, les recherches se sont basées sur le fait d'avoir toutes les données. K-NN pourrait ne pas être la meilleure solution si des données étaient manquantes ou si les données changeaient.

Bibliographie

[1] Hélène Guérin. *Qu'est ce qu'une probabilité ?* 2008.

[2] Lamis Hawarah. "Une approche probabiliste pour le classement d'objets incomplètement connus dans un arbre de décision". Thèse de doct. Université Joseph Fourier - Grenoble I, 2008.

[3] R. Gilleron F. Denis. *Apprentissage à partir d'exemples*. 2000.

[4] J. Ross Quinlan. "Induction of Decision Trees". In : *Machine Learning* 1.1 (1986), p. 81-106.

[5] Jan Kozak. *Decision tree and ensemble learning based on ant colony optimization*. Springer, 2019.

[6] Daniel T. Larose. *Discovering knowledge in data : an introduction to data mining*. Wiley-Interscience, 2005.

[7] Steven Roman. *Introduction to coding and information theory*. Undergraduate texts in mathematics. 1997.

[8] C. E. Shannon. "A Mathematical Theory of Communication". In : *The Bell System Technical Journal, Vol. 27*. 1948, p. 379-423, 623-656.

[9] Tom M. Mitchell. *Machine Learning, International Edition*. McGraw-Hill Series in Computer Science. 1997.

[10] Nikita Patel et Saurabh Upadhyay. "Study of Various Decision Tree Pruning Methods with their Empirical Comparison in WEKA". In : *International Journal of Computer Applications, Volume 60* (2012).

[11] Floriana Esposito, Donato Malerba et Giovanni Semeraro. "A Comparative Analysis of Methods for Pruning Decision Trees". In : *IEEE Trans. Pattern Anal. Mach. Intell.* 19.5 (1997), p. 476-491.

[12] J. Ross Quinlan. "Simplifying Decision Trees". In : *International Journal of Man-Machine Studies* 27.3 (1987), p. 221-234.

[13] Jorge Moraleda. "Gregory Shakhnarovich, Trevor Darrell and Piotr Indyk : Nearest-Neighbors Methods in Learning and Vision. Theory and Practice". In : *Pattern Anal. Appl.* 11.2 (2008), p. 221-222.

[14] Leo Breiman et al. *Classification and Regression Trees*. Wadsworth, 1984. isbn : 0-534-98053-8.

[15] J. Ross Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, 1993. isbn : 1-55860-238-0.

[16] Y. LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.

[17] L. Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In : *IEEE Signal Process. Mag.* 29.6 (2012), p. 141-142.

[18] F. Pedregosa et al. “Scikit-learn : Machine Learning in Python”. In : *Journal of Machine Learning Research* 12 (2011), p. 2825-2830.

Table des matières

Table des figures

Liste des tableaux