Le génie pour l'industrie

École de technologie Supérieure

SYS800

Reconnaissance de formes et inspection

Laboratoire 2: Algorithmes de classification

Auteur :
M'Hand Kedjar

Code ÉTS :
KEDM09088002

Montréal, le 02 décembre 2016

I Introduction

Le but de ce deuxième laboratoire est d'appliquer 3 différents algorithmes de classification sur les vecteurs caractéristiques des bases de données d'apprentissage et de test. Nous rappelons que nous avons obtenu après application de la projection ACP, une base de données d'apprentissage contenant 6000 vecteurs caractéristiques de 44 éléments chacun, qui permettaient de garder plus de 95% de la variabilité des données. La base de test contient quant à elle 1000 vecteurs caractéristiques de 44 éléments chacun. Nous avons en tout 600 exemples pour chaque chiffre dans la base d'apprentissage et 100 dans la base de test. Le tableau 1 résume les caractéristiques des données utilisées.

Base de données	n_e	n_c	n_{e_c}
Totale	7000	10	700
Apprentissage	6000	10	600
Test	1000	10	100

TABLE 1 – Caractéristiques des bases de données utilisées

II Bayes Quadratique

C'est une des approches paramétriques qui consiste à faire une hypothèse sur la distribution des données, puis d'estimer les paramètres de cette distribution en utilisant la base d'apprentissage. Le classificateur de Bayes Quadratique suppose que les données suivent une une distribution Gaussienne multivariée de la forme suivante :

$$p(x|\omega_j) = (2\pi)^{-d/2}|\Sigma_j|^{-1/2}exp(-\tfrac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)) \tag{1}$$

où d est la dimension des vecteurs de caractéristiques. Σ_j et μ_j représentent, respectivement, la matrice de covariance et le vecteur moyen (centroïde) des données de la classe ω_j. Le terme $p(x|\omega_j)$ représente la vraisemblance qui sera utilisée dans la phase de classification pour estimer les probabilités *a posteriori* en utilisant la règle de Bayes :

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)} \tag{2}$$

où $P(\omega_j)$ est la probabilité *a priori* de la classe ω_j et $p(x)$ l'évidence qui est définie par :

$$p(x) = \sum_{j=1}^{c} p(x|\omega_j)P(\omega_j) \tag{3}$$

où c est le nombre de classes du problème. Nous avons donc :

$$\begin{aligned} P(\omega_j|x) &= p(x|\omega_j)P(\omega_j)/p(x) \\ &= (2\pi)^{-d/2}|\Sigma_j|^{-1/2}exp(-\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j))P(\omega_j)/p(x) \end{aligned} \tag{4}$$

La règle de décision consiste à classer x dans la classe qui possède la plus grande probabilité *a posteriori*, c'est-à-dire :

$$P(\omega_i|x) > P(\omega_j|x),\ \forall\ j \neq i \implies \omega = \omega_i \tag{5}$$

Le terme $p(x)$ est un terme de normalisation et n'intervient pas dans la décision. En éliminant ce terme de l'équation (4) et en prenant le logarithme naturel, on obtient les fonctions discriminantes associées à chacune des classes qui sont données par :

$$\begin{aligned} d_j(x) &= ln[(2\pi)^{-d/2}|\Sigma_j|^{-1/2}exp(-\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j))P(\omega_j)] \\ &= ln((2\pi)^{-d/2}) + ln(|\Sigma_j|^{-1/2}) + (-\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)) + ln(P(\omega_j)) \end{aligned} \tag{6}$$

Le logarithme naturel $ln(.)$ est une fonction monotone croissante, la décision reste inchangée si à la place de maximiser $P(\omega_j|x)$, on maximise : $ln(P(\omega_j|x))$.

Dans notre cas, les classes $w_j,\ j = 1, 2, ..., 10$ sont équiprobables car dans la base d'apprentissage chacune des 10 classes contient exactement 600 éléments, c'est-à-dire : $P(\omega_j) = (1/10),\ \forall j$. Ce terme n'intervient donc pas dans la règle de décision. Au final, en éliminant les termes constants de l'équation (6), nous obtenons :

$$d_j(x) = -\tfrac{1}{2}(|\Sigma_j|) - \tfrac{1}{2}(x - \mu_j)^T \Sigma_j^{-1}(x - \mu_j) \tag{7}$$

Enfin, la décision est prise en cherchant la valeur maximale parmi les fonctions discriminantes associées à chacune des classes. Ainsi la règle de décision (5) devient :

$$d_i(x) > d_j(x),\ \forall\ j \neq i \implies \omega = \omega_i \tag{8}$$

II.1 Codage de la fonction Bayes Quadratique

Dans cette section, nous allons utiliser le langage Matlab pour coder la fonction ***Classify_QBayes.m*** et évaluer le taux d'erreur. Avant de passer à l'algorithme proprement dit, nous allons détailler les bases de données d'entrainement et de test utilisées qui sont détaillées en figure (1).

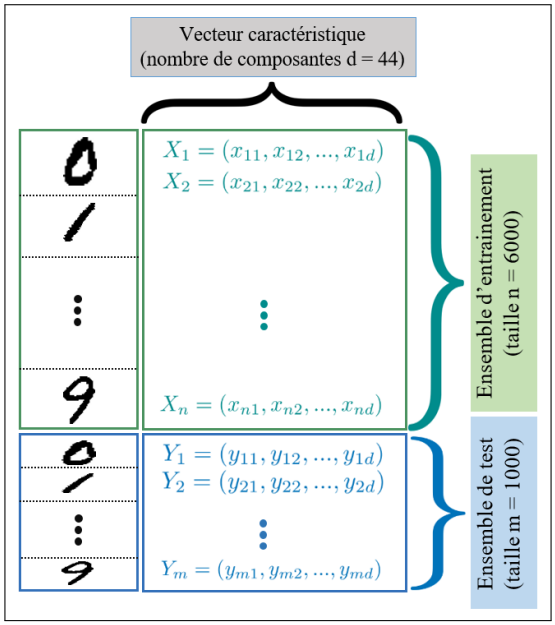

FIGURE 1 – Bases de données d'entrainement et de test

La base d'entrainement est composée de $n = 6000$ éléments répartis en $c = 10$ classes de $n_{e_c} = 600$ éléments chacune. Chaque élément est représenté par un vecteur caractéristique $X_n = (x_{n1}, x_{n2}, ..., x_{nd})$, $n = 1, 2, ..., 6000$ et $d = 44$. La base de test est composée de $m = 1000$ éléments répartis en $c = 10$ classes de $m_{e_c} = 100$ éléments chacune. Chaque élément est représenté par un vecteur caractéristique $Y_m = (y_{m1}, y_{m2}, ..., y_{md})$, $m = 1, 2, ..., 1000$ et $d = 44$.
Nous utiliserons les éléments de la base d'apprentissage pour estimer les paramètres de la fonction de décision (8) pour chacune des classes $j = 1, 2, ..., 10$:

- **Vecteur moyen** $\mu_j = \frac{1}{n_{e_c}} \sum_i^{n_{e_c}} X_i$: de taille $d = 44$ éléments.
- **Matrice de covariance** Σ_j, son inverse : Σ_j^{-1} et son déterminant : $|\Sigma_j|$

a. Évaluation du taux d'erreur

En évaluant les équations (7) et (8) pour la base de test, nous trouvons une erreur de **5.4%**, c'est-à-dire 54 éléments incorrectement classés parmi les 1000 que contient la base de test.

b. Évaluation de la matrice de confusion

Pour évaluer en détail les performances de l'algorithme et sa qualité de classification, nous avons calculé la matrice de confusion. Le résultat trouvé est résumé en tableau (2). En analysant les résultats du tableau, nous pouvons faire les quelques remarques suivantes :

- Le chiffre 5 est celui qui possède l'erreur de classification la plus élevée : parmi les 100 éléments dans la base de test qui représentent le chiffre 5, 18 ont été mal classés ; parmi eux 16 classés comme 3 et 2 comme un 8.
- Le chiffre 6 est celui qui possède l'erreur de classification la plus petite : uniquement un seul 6 a été incorrectement classé en 8.
- Le chiffre 8 a été incorrectement classé 5 fois comme le chiffre 3, et le chiffre 3 incorrectement classé 6 fois comme le chiffre 2

		Classe estimée										Σe
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Classe actuelle	**0**	**96**	**2**	**1**	0	0	0	0	0	**1**	0	**4**
	1	0	**98**	0	0	0	0	**1**	0	**1**	0	**2**
	2	0	0	**96**	0	0	0	0	0	**4**	0	**4**
	3	0	0	**6**	**92**	0	0	0	0	**2**	0	**8**
	4	**1**	0	**2**	0	**97**	0	0	0	0	0	**3**
	5	0	0	0	**16**	0	**82**	0	0	**2**	0	**18**
	6	0	0	0	0	0	0	**99**	0	**1**	0	**1**
	7	0	0	**3**	0	0	0	0	**96**	0	**1**	**4**
	8	0	0	**1**	**5**	**1**	0	0	0	**93**	0	**7**
	9	0	0	**1**	0	0	0	0	**1**	**1**	**97**	**3**
		1	**2**	**14**	**21**	**1**	**0**	**1**	**1**	**12**	**1**	**54**

TABLE 2 – Matrice de confusion obtenue avec le Bayes Quadratique

Ainsi, nous pouvons faire une première conclusion : Les chiffres 5, 3 et 8 sont ceux qui ont l'erreur de classification la plus élevée. Pour comprendre un peu plus en détail ces erreurs de classification, nous allons afficher les chiffres mentionnés. Ainsi, la figure 2 montre les chiffres mal-classés avec la classe estimée associée.

FIGURE 2 – Exemples de quelques chiffres mal-classés avec Quadratique Bayes. La classe prédite est affichée en haut à gauche (en rouge) de chaque élément. Le nom du fichier original est aussi inscrit en bas à gauche (en bleu).

En analysant ces résultats, nous constatons que visuellement et à l'exception d'un seul cas, le chiffre 5 qui a été mal classé 18 fois en tout et 16 fois en 3, est très facile à reconnaitre. Autrement dit, en le voyant, il n'y aucune ambiguïté pour dire que c'est un 5. Le même constat peut être fait pour le chiffre 8, à l'exception du cas

où il a été mal classé en 4 où effectivement nous constatons qu'il est mal écrit. Pour le chiffre 3, nous voyons que visuellement, nous pouvons effectivement le confondre avec un 8 ou un 2. Pour les autres chiffres pris en exemple (0, 4, 1, 7, 2, 9 et 6), visuellement, ils sont parfaitement reconnaissables. Ces erreurs de classifications du chiffre 5 particulièrement peuvent surprendre à première vue, surtout que visuellement, ils sont parfaitement reconnaissable. C'est extrêmement difficile de trouver l'exacte raison de ces erreurs. Peut être que ça peut s'expliquer par la méthode d'extraction de caractéristiques que nous avons choisie où les zones de projections entre quelques éléments représentant le 3 et le 5 avec des vecteurs caractéristiques proches. Ces erreurs de projections peuvent survenir d'une rotation, d'une translation ou d'un ajout de bruit sur l'image représentant l'élément.
La classification Bayes Quadratique qui repose sur la maximisation de la probabilité *a posteriori* pour établir la classe de sortie fait donc moins de **6%** d'erreur sur la base de test. Pour une étude plus poussée, nous avons décidé d'analyser ces 54 erreurs en regardant pour chaque classe actuelle la classe prédite si nous prenons la deuxième valeur de la fonction fonction discriminante $d_j(x)$ pour $j = 1, ..., 10$

Chiffre	Classe	Σe_1	Σe_2	Σe_3	Σe_4	$j \vert \Sigma e_j = 0$
0	C_1	**4**	**1**	0	0	3
1	C_2	**2**	**1**	1	1	5
2	C_3	**4**	**0**	0	0	2
3	C_4	**8**	**1**	1	0	3
4	C_5	**3**	**1**	1	1	7
5	C_6	**18**	**6**	0	0	3
6	C_7	**1**	**1**	1	1	6
7	C_8	**4**	**1**	0	0	3
8	C_9	**7**	**1**	1	0	4
9	C_{10}	**3**	**1**	1	1	5

TABLE 3 – Évolution de l'erreur en fonction du choix de d_j

Dans le tableau 3 sont résumées les erreurs de prédiction pour les différentes classes $C_i, i = 1, ..., 10$, si nous décidons de nous servir des probabilités *a posteriori* (ou les fonctions discriminantes associées) $d_j, j = 1, ..., 10$ en les classant par ordre décroissant. Par exemple, pour la classe C_1 et pour le fichier ***mnist7000\Test\0\000618.tif***, nous avons les classes suivantes par ordre de probabilité *a posteriori* croissante : **3**, **1**, 4, 9, 5, 2, 7, 10, 8, 6. C'est-à-dire que l'exemple qui appartient à la classe C_1 est classé en C_3, classe qui possède la probabilité *a posteriori* la plus élevée pour cet exemple. La prochaine classe qui possède la probabilité la plus élevée est C_1.

Ainsi, nous arrivons à bien classer ce chiffre si nous prenons en compte la probabilité *a posteriori* pour la classe arrivant en deuxième position. En suivant ce même principe, nous remarquons que si nous prenons en compte la probabilité *a posteriori* pour la classe arrivant en deuxième position pour le chiffre 5, le nombre d'erreurs descend de 18 à 6, et à 0 si nous rajoutons celle à la troisième position. Le tableau 3 donne la position j pour laquelle l'erreur de la classe C_i descent à 0. Nous pouvons conclure donc que dans la majorité des cas, si nous prenons en compte la probabilité pour la classe arrivant en deuxième position, l'erreur globale descend de 5.4% à 1.4%.

Dans la figure 3, nous rappelons le taux de recouvrement entre les classes pour les bases d'apprentissage et de test obtenu après l'application de l'ACP. En analysant ces deux matrices, nous remarquons, pour la base de test, que

$$\begin{bmatrix} 598 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 595 & 0 & 1 & 0 & 0 & 0 & 0 & 4 & 0 \\ 2 & 1 & 584 & 1 & 1 & 1 & 1 & 1 & 7 & 1 \\ 0 & 0 & 2 & 592 & 0 & 0 & 1 & 1 & 4 & 0 \\ 0 & 3 & 1 & 0 & 589 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 1 & 3 & 0 & 593 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 599 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 587 & 2 & 7 \\ 0 & 8 & 1 & 3 & 1 & 5 & 0 & 1 & 573 & 8 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 5 & 0 & 593 \end{bmatrix} \quad \begin{bmatrix} 98 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 99 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 96 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 98 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 94 & 0 & 1 & 1 & 1 & 2 \\ 0 & 0 & 0 & 3 & 0 & 97 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 98 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 \\ 1 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 94 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 98 \end{bmatrix}$$

FIGURE 3 – Matrices de confusion pour le recouvrement des classes pour les bases d'apprentissage (à gauche) et test (à droite) après ACP

le chiffre 5 a un recouvrement de 3 avec le chiffre 3, 8 a un recouvrement de 2 avec 1, 2 avec 3 et 1 avec 9. Ainsi, les erreurs trouvées avec la classification Bayes Quadratique, ne peuvent s'expliquer qu'en partie, par la mesure du taux de recouvrement. Une analyse statistique plus approfondie est nécessaire pour trouver la cause de ces erreurs.

III k Plus Proches Voisins (k-PPV)

Dans le domaine de la reconnaissance des formes, l'algorithme des k plus proches voisins (k-PPV) ou k-NN en anglais (k-nearest neighbors) est une méthode non-paramétrique utilisée pour la classification. Ainsi, nous n'avons pas besoin de faire une hypothèse sur la nature de distribution de nos données. Le principe de l'algorithme des k-PPV est le suivant :

Nous disposons d'un ensemble d'apprentissage représenté par l'ensemble des vecteurs caractéristiques $X_i, i = 1, ..., n$ et un ensemble de test représenté par $Y_j, j = 1, ..., m$. Pour déterminer la classe C_j d'un élément de l'ensemble test, il suffit de calculer sa distance $d(Y_j, X_i)$ avec tous les éléments de l'ensemble d'apprentissage puis d'effectuer un vote majoritaire parmi les k données les plus proches ; chaque donnée votant pour la classe à laquelle elle appartient.

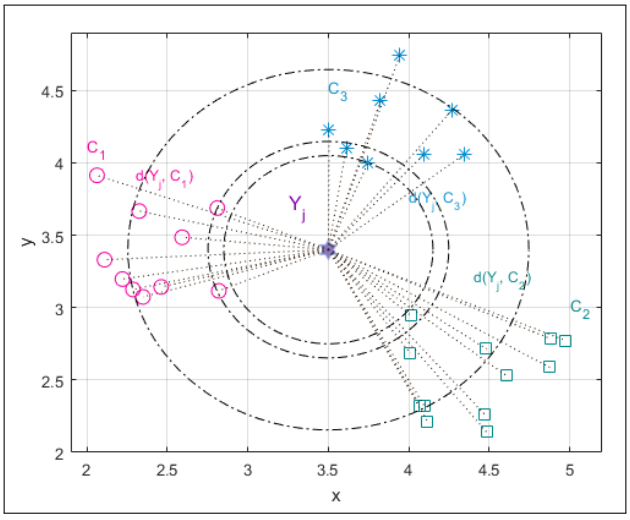

FIGURE 4 – Illustration de l'algorithme k-PPV

La figure 4 présente une illustration de l'algorithme. Nous voulons classifier le nouvel élément Y_j dans une des 3 classes C_1, C_2 et C_3. Un cercle de rayon r_k est tracé pour inclure les k éléments les plus proches, provenant des classes C_1, C_2 et C_3. Pour classer l'élément Y_j, nous comptons le nombre d'éléments majoritaires qui appartiennent à une des 3 classes et qui sont à l'intérieur du cercle. Le seul paramètre que nous avons besoin de fixer pour prendre la décision est le nombre k. Habituellement, et pour éviter des cas ambiguës, k est un nombre naturel, positif et impair. La distance Euclidienne est la mesure de distance la plus courante utilisée et elle définie par :

$$d_i(x) = (x - x_i)^T (x - x_i) \quad i = 1, 2, ..., n \tag{9}$$

Ici, x représente l'élément à classifier, et x_i l'un des n exemples de l'ensemble d'apprentissage.

III.1 Codage du classificateur k-PPV

Dans cette section, nous allons utiliser le langage Matlab pour coder la fonction ***Classify_KNN.m*** et évaluer le taux d'erreur. Comme nous l'avons mentionné auparavant, le seul paramètre que nous avons besoin de fixer est le nombre des plus proches voisins k. Dans la figure 5, est illustrée la méthode que nous allons utiliser pour trouver le k optimal.

La base de données d'entrainement après ACP contient 6000 chiffres répartis en 10 classes (chiffres 0 à 9) égales de 600 éléments chacune. Chaque chiffre est représenté par un vecteur caractéristiques à $d = 44$ éléments. Pour chaque classe, on sélectionne, d'une manière aléatoire, $r_a = 75\%$ pour la base d'apprentissage, et $r_v = 25\%$ pour la base de validation. On aura donc la base d'apprentissage globale composée de 4500 chiffres répartis en 10 classes (chiffres 0 à 9) égales de 450 éléments chacune. La base de validation globale sera composée de 1500 chiffres répartis en 10 classes égales de 150 éléments chacune. En procédant ainsi, nous aurons toujours le même nombre d'éléments par classe :450 pour la base d'apprentissage et 150 pour la base de validation.

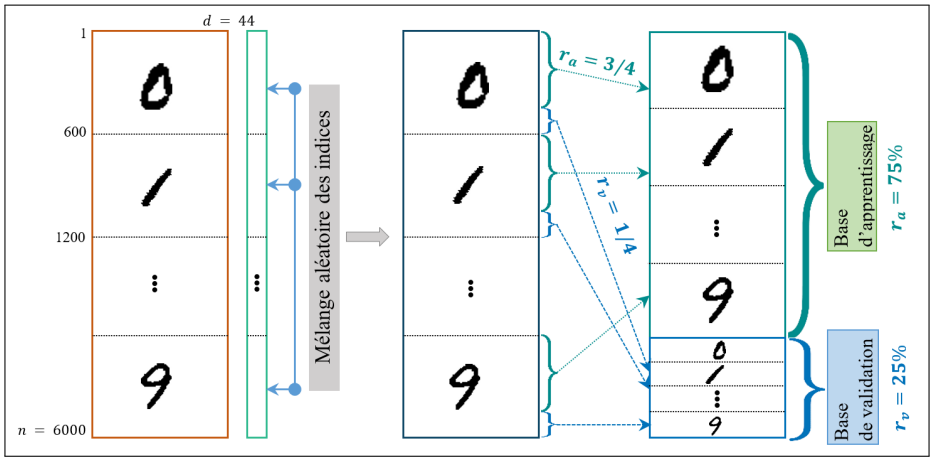

FIGURE 5 – Construction des bases d'apprentissage et de validation

a. Détermination du nombre k de voisins

Nous allons faire varier le nombre des plus proches voisins, et à chaque fois évaluer le taux d'erreur sur la base de validation τ_{ev}.

Nous rappelons que pour évaluer le taux d'erreur, pour chaque élément de la base de validation, nous devons calculer sa distance avec tous les éléments de la base d'apprentissage, et effecteur un vote majoritaire pour déterminer sa classification. Le meilleur taux d'erreur sur la base de validation trouvé est égal à $\tau_{ev} = \mathbf{1.93\%}$, c'est-à-dire **29** éléments mal-classés sur les 1500 que contient la base de validation. Cette valeur a été atteinte pour $k = 1$. Dans un premier temps, nous pouvons conclure donc que la valeur optimale de k est égale à 1. Dans les prochaines sections, nous allons tenter d'aller plus en détail pour vérifier l'influence de quelques paramètres sur la valeur optimale de k.

b. Erreur d'entrainement en fonction de k

Dans cette section, nous allons calculer le taux d'erreur sur la base d'apprentissage τ_{ea} en fonction du nombre de plus proches voisins pris. Dans le but d'effectuer une comparaison, nous allons également calculer le taux d'erreur sur la base de validation τ_{ev} et de test τ_{et} pour différentes valeurs de k. La figure 6 montre le résultat obtenu sur le taux d'erreur en faisant varier k de 1 à 51, pour une sélection aléatoire de ($r_a = 3/4$: apprentissage , $r_v = 1/4$: test). Nous remarquons tout d'abord que l'erreur sur la base d'apprentissage est de 0 pour $k = 1$, ce qui compréhensible

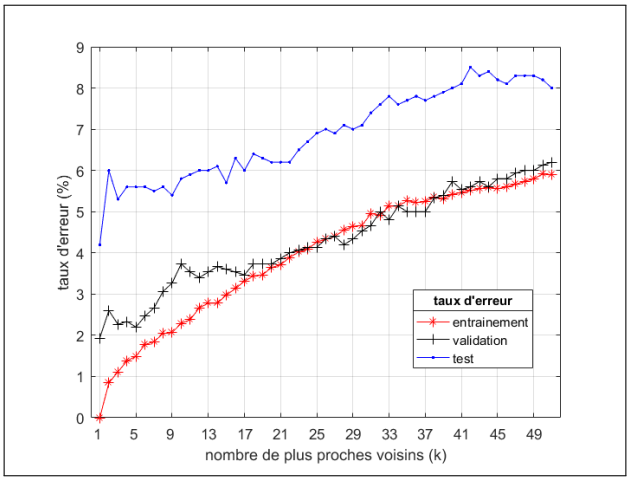

FIGURE 6 – Taux d'erreur en fonction de k

vu que l'élément le plus proche d'un élément X_i est l'élément lui-même. À mesure que k augmente, le taux d'erreur

augmente aussi. Pour la base de validation, nous remarquons également que la valeur $k = 1$ est celle qui obtient le taux d'erreur le plus faible qui est égal à 1.93%.
Ce résultat peut paraitre surprenant, et pour vérifier plus en détail, nous avons décider de modifier quelques paramètres pour voir leur effet sur la valeur de k optimale.

- **Mesure de distance utilisée** : nous avons expérimenté avec la distance euclidienne normalisée et la distance de Mahalanobis, et à chaque fois le taux d'erreur obtenu est plus grand que celui obtenu avec la distance euclidienne. Ceci peut être expliqué par le fait que nous avons aussi utilisé la distance euclidienne pour calculer le taux de recouvrement, et prendre la configuration qui le minimise.
- **Nombre de composantes principales après ACP** Nous avons fait varier le nombre de composantes principales pour voir leur influence sur la valeur de k optimal, le tableau 4 résume quelques résultats :

N_c	s	r_a	r_t	k_{opt}	τ_{ev}	τ_{et}	τ_{eQB}	Commentaires
5	47.45	17.90	19.20	9	14.27	21.70	23.50	Pour une petite valeur de $N_c(\leq 10)$, les performances de k-PPV et QBayes sont trop basses
10	67.10	4.32	5.20	3	4.07	7.50	9.10	
18	80.33	2.20	**2.50**	1	2.13	5.20	4.60	$min(r_t)$ pour la base de test
25	86.76	1.68	3.10	1	1.87	4.30	4.90	
30	89.81	1.55	3.10	1	1.93	**4.00**	4.80	$min(\tau_{ev})$pour k-PPV sur la base de test
33	91.25	1.63	3.20	1	1.80	4.40	**4.40**	$min(\tau_{eQB})$pour QBayes sur la base de test
37	92.88	**1.53**	2.90	1	1.67	4.50	4.50	$min(r_a)$ pour la base d'apprentissage
39	93.59	1.58	2.60	1	**1.60**	4.40	4.70	$min(\tau_{ev})$ pour k-PPV sur la base d'évaluation
44	**95.11**	**1.62**	**2.80**	**1**	**1.93**	**4.20**	**5.40**	$s \geq 95\%$: seuil fixé (variabilité expliquée)
50	96.51	1.70	2.80	1	1.93	4.60	5.10	La valeur optimale de k reste constante et égale à 1 à partir de $N_c = 18$. Lorsque le nombre de composantes principales (N_c) devient assez élevé ($N_c \geq 50$), τ_{et} reste constant, τ_{ev} reste bas, tandis que τ_{eQB} augmente rapidement avec la valeur de N_c
60	98.23	1.63	2.70	1	1.73	4.60	5.60	
65	98.86	1.67	2.80	1	1.73	4.60	5.90	
70	99.37	1.73	2.80	1	1.67	4.60	6.40	
80	100	1.72	2.90	1	1.80	4.60	7.10	
90	100	1.72	2.90	1	1.80	4.60	71.10	
98	100	1.72	2.90	1	1.80	4.60	90.00	

TABLE 4 – Performances des classifieurs K-PPV et QBayes en fonction du nombre de composantes principales. N_c : nombre de composantes principales de l'ACP. s : seuil sur la variabilité expliquée. r_a : taux de recouvrement pour la base d'apprentissage. r_t : taux de recouvrement pour la base de test. k_{opt} : valeur optimal de k. τ_{ev} : taux d'erreur sur la base de validation en utilisant la valeur k_{opt}. τ_{et} : taux d'erreur sur la base de test en utilisant la valeur k_{opt}. τ_{eQB} : taux d'erreur donnée par Bayes Quadratique (pour comparaison).

En analysant les résultats du tableau 4, nous pouvons faire les remarques suivantes :

- À partir d'une certaine valeur du nombre de composantes principales considérées $N_c = 18$, la valeur optimale de k (celle qui donne l'erreur de classification la plus basse sur la base de validation) reste constante et est égale à 1.
- L'erreur sur la base de test τ_{et} en utilisant la valeur optimale de k est en général meilleure que celle trouvée en utilisant Bayes Quadratique τ_{eQB}
- Les performances de l'algorithme K-PPV restent, dans une certaine mesure, stables si on augmente le nombre de composantes principales. Contrairement à Bayes Quadratique dont les performances se dégradent rapidement à partir d'une certaine valeur de $N_c \geq 60$
- Si nous prenons les indicateurs de performances individuellement, chacun d'eux atteint sa valeur optimale pour un nombre différent de N_c ; par exemple $min(\tau_{ev})$ est atteint pour $N_c = 39$, tandis que $min(\tau_{et})$ est atteint pour 30. Une similaire observation peut être faite pour les autres mesures : r_a, r_t, et τ_{eQB}.
- L'erreur k-PPV sur la base de test (τ_{et}) est tout le temps plus grande que celle sur la base de validation (τ_{ev}).

- Nous avons aussi effectué une validation croisée avec 4 blocs, et la valeur optimale de k est également égale à 1. Ainsi, nous pouvons conclure que $k_{opt} = 1$ et l'utiliser pour la suite du laboratoire.

c. Évaluation du taux d'erreur sur la base de test

Nous allons donc utiliser $k = 1$ pour calculer l'erreur sur la base de test. En prenant toujours le nombre de composantes principales $N_c = 44$, le taux d'erreur trouvé sur la base de test en utilisant la totalité de la base d'apprentissage est égal à **4.20%** ; c'est-à-dire 42 éléments mal classés sur les 1000 que contient la base de test. Nous constatons que l'erreur trouvée avec le classifieur k-PPV est plus faible que celle trouvée avec Bayes Quadratique (qui était de **5.4%**).

d. Évaluation de la matrice de confusion

Nous avons donc montré que le k optimal est égal à 1. C'est cette valeur que nous allons utiliser par la suite pour évaluer le taux d'erreur sur la base de de test et le calcul de la matrice de confusion donnée en figure 5.

		Classe estimée										Σe
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Classe actuelle	**0**	**97**	2	0	0	0	0	1	0	0	0	**3**
	1	0	**100**	0	0	0	0	0	0	0	0	**0**
	2	1	0	**91**	0	0	0	2	0	6	0	**9**
	3	0	1	1	**96**	0	0	0	0	2	0	**4**
	4	0	0	1	0	**95**	0	2	1	0	1	**5**
	5	0	0	0	2	0	**97**	0	1	0	0	**3**
	6	0	0	0	0	0	1	**99**	0	0	0	**1**
	7	0	0	0	0	0	0	0	**100**	0	0	**0**
	8	0	2	0	4	1	2	0	1	**88**	2	**12**
	9	0	0	0	1	2	0	0	2	0	**95**	**5**
		1	**5**	**2**	**7**	**3**	**3**	**6**	**5**	**8**	**3**	**42**

TABLE 5 – Matrice de confusion obtenue avec le k-PPV

En analysant les résultats du tableau 5, nous pouvons faires les remarques suivantes :

- Le chiffre 8 est celui qui possède l'erreur de classification la plus élevée : parmi les 100 éléments dans la base de test qui représentent le chiffre 8, 12 ont été mal classés ; parmi eux quatre classés en 3, deux en 1, deux en 5, deux en 9, un en 4 et un comme un 7.
- Les chiffres 1 et 7 n'ont enregistré aucune erreur de classification.
- Le chiffre 2 a été incorrectement classé six fois comme un 8, et le chiffre 3 incorrectement classé 2 fois comme le chiffre 8.

Ainsi, nous pouvons conclure que Les chiffres 8 et 2 sont ceux qui ont l'erreur de classification la plus grande.
Pour voir en détail ces éléments mal classés, dans la figure 7 sont affichés l'image originale associée au chiffre et la classe de sortie attribuée par le classificateur k-PPV.
Nous remarquons qu'au moins 6 éléments (par exemple : **8/000629.tif**, **8/000698.tif**, etc.) qui représentent le chiffre 8, sont effectivement très difficile à reconnaitre visuellement. Nous pouvons faire la même remarque pour quelques autres éléments : **2/000668.tif**, **3/000608.tif**, **4/000693.tif**, **0/000693.tif**, etc. D'un autre coté, nous voyons certains éléments mal classés parfaitement identifiables visuellement, par exemple : **5/000631.tif**, **8/000611.tif**, **4/000606.tif**, etc.
Avec l'algorithme k-PPV, nous trouvons moins d'éléments représentant le chiffre 5 qui sont mal classés en 3 qu'avec l'algorithme Bayes Quadratique. Par contre, pour le chiffre 2 est plus souvent confondu avec un 8. L'algorithme k-PPV de par son principe de mesure de distance et de vote majoritaire, peut effectivement trouver deux éléments de deux classes différentes plus proches que ceux de la même classe. Plusieurs raisons peuvent expliquer cette situation. Si l'image a subi une rotation, une translation ou l'ajout d'un bruit peuvent avoir un impact sur la distance mesurée. Aussi, les performances de k-PPV dépendent grandement de la méthode choisie pour l'extraction des caractéristiques.

FIGURE 7 – Exemples de quelques chiffres mal-classés avec k-PPV. La classe prédite est affichée en haut à gauche (en rouge) de chaque élément. Le nom du fichier original est aussi inscrit en bas à gauche (en bleu).

En comparant les résultats de la figure 7 avec ceux de la figure 2, nous remarquons qu'ils existent certaines éléments qui ont été mal classés par les deux algorithmes, Quadratique Bayes et k plus proches voisins. Parmi ces éléments, nous pouvons citer : **0/000693.tif**, **2/000607.tif**, **3/000629.tif**, etc. Pour certains de ces éléments, les deux algorithmes attribuent la même classe de sortie (exemple : l'élément **8/000665.tif** classé comme un 3), mais pour d'autres éléments, les classes de sortie attribuées par les deux algorithmes sont différentes (exemple : **3/000608.tif** classé comme un 8 avec Quadratique Bayes, et comme un 1 par k-PPV).
Nous avons résumé ces éléments dans la figure 8.

FIGURE 8 – Exemples de quelques chiffres mal-classés avec Bayes Quadratique et k-PPV. Chaque élément illustré est accompagné de sa classe prédite par Bayes Quadratique (premier chiffre entre parenthèses, en rouge) et par k-PPV (deuxième chiffre entre parenthèses, en vert). Le nom du fichier original est aussi inscrit en bas à gauche (en bleu).

Nous pouvons faire une première remarque importante : Les classifieurs Quadratique Bayes et k-PPV ont fait des erreurs de classification communes sur uniquement **17** éléments de la base de test. Ainsi, si nous pouvons combiner les performances de chacun de ces classifieurs, nous pouvons atteindre un taux d'erreur globale de **1.70%**, ce qui

Classe	k_{opt}	τ_{ev}	τ_{et}
0	1	0.67	3.00
1	**2**	0	0
2	1	2.00	9.00
3	1	2.67	4.00
4	1	0.67	5.00
5	1	2.00	3.00
6	1	0	1.00
7	**3**	0.67	0
8	1	2.67	12.00
9	1	0.67	5.00
	τ_e	**1.20**	**4.20**

TABLE 6 – k optimal pour chacune des classes

représente un très grand gain par rapport au **4.20%** du k-PPV seul, et plus encore par rapport au **5.40%** du Quadratique Bayes.

e. Détermination du k optimal pour chacune des classes.

Dans cette section, dans le but d'améliorer le taux d'erreur sur la base de test, nous allons essayer d'évaluer la valeur optimale de k pour chacune des classes. Nous allons utiliser la même stratégie pour décomposer la classe d'entrainement en deux sous-ensembles : 75% pour l'apprentissage et 25% pour la validation.
L'algorithme qui sera implémenté est détaillé comme suit :

Pour chaque classe $C_i, i = 1, 2, .., 10$;

- Nous allons prendre 75% pour la base d'apprentissage, ce qui donne 4500 éléments, et 25% pour la base de validation en limitant les éléments sélectionnés à la classe correspondante, ce qui donne 150 éléments.
- Nous allons calculer la distance euclidienne entre chaque élément de la base de validation avec tous les éléments de la base d'apprentissage, et nous allons effectuer un vote majoritaire en fonction de la valeur de k en prenant les k plus proches voisins pour déterminer la classe de sortie.
- Nous allons évaluer l'erreur sur la base de validation, et la valeur de k qui donne l'erreur minimale sera le k optimal pour cette classe.

Une fois qu'on aura déterminé le k optimal pour chacune des classes, nous allons utiliser les valeurs correspondantes pour évaluer le taux d'erreur sur la base de test. Les valeurs k trouvées sont résumées dans le tableau 6.

Nous pouvons remarquer que le k optimal est différent pour les classes représentant les chiffres 1 et 7, où il est égal à $k_{opt} = 2$ et 3, respectivement. Pour les autres classes, la valeur de $k = 1$ reste inchangée. Pour ce qui est de l'erreur sur la base de validation, elle s'améliore de beaucoup ($\tau_{ev} = 1.20\%$) par rapport à la valeur précédente ($\tau_{ev} = 1.93\%$). Par contre, le taux d'erreur sur la base de test est resté inchangé ($\tau_{et} = 4.20\%$), et nous retrouvons exactement les mêmes erreurs pour chacune des classes que celles trouvées précédemment en prenant $k = 1$ pour toutes les classes.

IV Machines à vecteurs de support

Dans cette troisième partie, nous allons illustrer l'utilisation de l'algorithme basé sur les machines à vecteurs de support (Support Vector Machine (SVM) en anglais) pour la reconnaissance de formes. Le SVM fait partie de la famille de l'ensemble des techniques d'apprentissage supervisé. C'est un classifieur binaire dont l'objectif est de trouver la frontière optimale qui sépare les données de l'ensemble d'apprentissage en deux classes. Cette frontière est donnée par l'équation suivante :

$$\sum_{i=1}^{n} y_i a_i K(x_i, x) + b = 0 \tag{10}$$

$y_i \in \{1, -1\}$ correspondent aux étiquettes des données d'apprentissage x_i. K est la fonction noyau qui permet de projeter les données dans un espace de plus grande dimension où une séparation linéaire est éventuellement possible. Le but de la phase d'apprentissage est de déterminer les coefficients a_i et le biais b qui maximisent une certaine quantité qu'on appelle marge de séparation m. Cette marge représente la distance entre les données de la classe avec l'étiquette $y_i = 1$ (classe positive) des données de la classe avec l'étiquette $y_i = -1$ (classe négative) les plus proches de la frontière de séparation. La détermination des paramètres a_i et b se fait en résolvant un problème d'optimisation sous contraintes, équivalent à l'équation suivante :

$$\max_{a} \left[\sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_i y_j K(x_i, y_j) \right] \tag{11}$$

avec $\sum_{i=1}^{n} a_i y_i = 0$ et $0 \leq a_i \leq \mathbf{C}$, $i = 1, ..., n$. C est un paramètre dit *de régularisation*

Les données d'apprentissage x_i associées à des coefficients $a_i \neq 0$ sont nommées vecteurs de support.
La sortie d'un classifieur SVM est donné par :

$$f(x) = \sum_{i=1}^{m} y_i a_i K(x_i, x) + b \tag{12}$$

où la décision sur la classe de sortie est prise en regardant le signe de $f(x)$. Dans le cas de l'utilisation d'un noyau linéaire $K(x_i, x) = x_i.x$, la frontière de séparation est alors donnée par l'équation d'un hyperplan défini par :

$$w.x + b = 0, \quad \text{avec} \quad w = \sum_{i=1}^{m} y_i a_i x_i$$

Le vecteur w est le vecteur déterminant la direction de l'hyperplan, et b est appelé le **biais**. La figure 9 illustre un cas à deux dimensions pour le cas simple de 2 classes linéairement séparables.

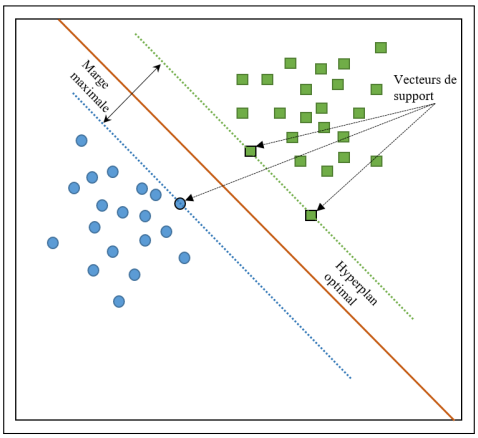

FIGURE 9 – Exemple d'un SVM en 2 dimensions : classes linéairement séparables

Dans la plupart des applications pratiques, les données ne sont pas linéairement séparables. Dans ce cas, nous utilisons d'autres types de noyaux (Ce qu'on appelle souvent *Kernel Trick* en anglais)pour réaliser une séparation linéaire en passant à un espace de plus grande dimensions. Parmi les noyaux utilisés, nous pouvons citer :

- **Le noyau polynomial** $K(x_i, x_j) = (ax_i.x_j + b)^d$

- **Le noyau Gaussien** $K(x_i, x_j) = exp\left(-\gamma \left\|x_i - x_j\right\|^2\right)$

La figure 10 montre un exemple de classes non linéairement séparables, et comment l'application d'un noyau permet de réaliser une séparation linéaire.

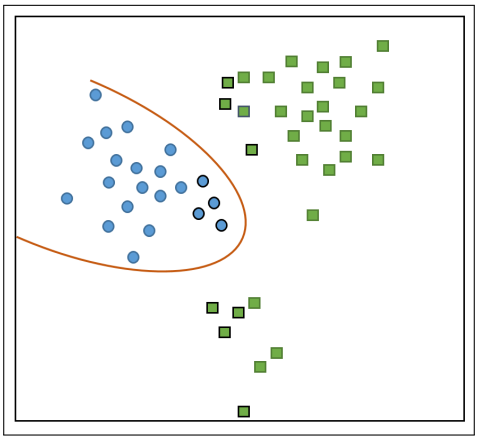

FIGURE 10 – Exemple d'un SVM en 2 dimensions : classes non linéairement séparables

Comme nous venons de le voir, le SVM dans sa forme de base ne permet de séparer que 2 classes. Mais dans notre problème, comme dans la plupart des cas pratiques, le nombre de classes est généralement plus grand que 2. Dans ce cas, il faudrait trouver une stratégie pour décomposer le problème globale de séparation entre les classes en plusieurs sous-problèmes de séparation binaire en 2 classes. Deux grandes approches sont à notre portée :

- la stratégie **un-contre-un** (one-against-one) qui consiste à construire un SVM pour chaque paire de classes. Ainsi, pour un problème à c classes, nous aurons $\frac{c(c-1)}{2}$ SVM qui seront entrainés pour distinguer les éléments d'une classe des éléments d'une autre classe. Habituellement, la classification d'un élément inconnu se fait au vote majoritaire, où chaque SVM vote pour une classe.
- la stratégie **un-contre-tous** (one-against-all) qui consiste à construire autant de SVM qu'il y'a de classes de sortie. Chaque SVM est alors entrainé à distinguer entre une classe C_i, et toutes les autres classes $C_j, j = 1, ..., c, j \neq i$. Dans ce cas, la décision de classification est prise en cherchant la valeur maximale parmi les sorties des différents SVM.

IV.1 Codage du classifieur SVM

Dans cette section, nous allons utiliser le langage Matlab pour coder la fonction ***Classify_SVM.m*** en utilisant la stratégie **un-contre-tous**, et évaluer le taux d'erreur sur la base de test. Ils existent plusieurs implémentations du SVM avec Matlab, et dans notre cas, nous allons nous servir de la librairie SVMlight développée par Thorsten Joachims, et dont le code est disponible en ligne [1].
Les valeurs de quelques paramètres du modèle doivent être fixés afin de minimiser l'erreur sur la base de validation. Pour cette partie, nous allons utiliser le noyau Gaussien : $K(x_i, x_j) = exp\left(-\gamma \left\|x_i - x_j\right\|^2\right)$. Nous allons avoir donc 3 paramètres à fixer : γ, qui peut être vu comme l'inverse de la variance d'une gaussienne, et C qui s'appelle le facteur de régularisation Dans la figure 11 est résumée la méthode que nous allons utiliser pour trouver les valeurs optimales de ces paramètres.

a.Détermination des hyperparamètres optimaux

Nous disposons de la base d'entrainement globale qui contient $n_e = 6000$ éléments, à partir de laquelle nous allons prendre $r_a = 75\%$ pour construire la base d'apprentissage, ce qui donne $n_a = r_a \times n_e = 4500$ éléments, et $r_v = 25\%$ pour la base de validation, ce qui donne $n_v = r_v \times n_e = 1500$ éléments. La sélection des éléments se fera d'une

1. http ://svmlight.joachims.org/

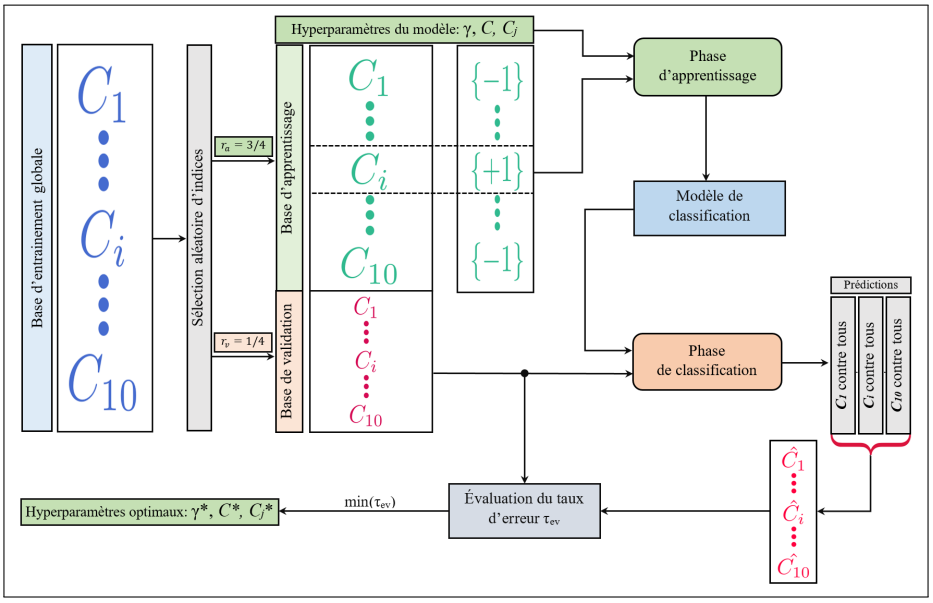

FIGURE 11 – Détermination des hyperparamètres en utilisant la stratégie un-contre-tous

manière aléatoire, et le nombre d'éléments par classe dans la base d'apprentissage sera gardé le même pour chaque classe et sera égal à $n_{a_c} = 450 = (6000 \times 75\%)/10$. Nous allons désigner les données de cette base par T_x et ses étiquettes par T_y.
Le nombre d'éléments par classe pour la base de validation sera égal à $n_{v_c} = 150 = (6000 \times 25\%)/10$, et sera le même pour toutes les classes. Nous allons désigner les données de cette base par V_x et ses étiquettes par V_y.

Nous disposons de 3 hyperparamètres à fixer : g, C et C_j, dont les valeurs sont affichées dans le tableau 7.
Pour cette partie de l'algorithme, nous allons prendre $C_j = C$ pour toutes les configurations. Nous allons faire varier

g	0.1	0.01	0.001	0.0001	
C	1	10	100	1000	10000
C_j	1	10	100	1000	10000

TABLE 7 – Valeurs des hyperparamètres considérés

les valeurs des hyperparamètres g, C et C_j, et nous allons mesurer le taux d'erreur τ_{ev} sur la base de validation. Les valeurs de g, et C qui donnent le taux d'erreur le plus bas seront nos hyperparamètres optimaux : g^* et C^* que nous allons utiliser pour calculer le taux d'erreur sur la base de test τ_{et}.

- Pour chaque classe $C_i, i = 1, 2, ..., 10$, nous allons créer un vecteur qui va contenir les étiquettes des classes de sortie, en attribuant l'étiquette $\{+1\}$ pour la classe actuelle C_i, et $\{-1\}$ pour les autres classes. Nous allons désigner ce vecteur par Y_x
- Nous allons entrainer un classifieur SVM en lui donnant comme paramètres d'entrée les données d'entrée de la base d'apprentissage T_x , le vecteur des étiquettes défini précédemment Y_x, le type de noyau utilisé (Gaussien dans notre cas), ainsi que les valeurs des hyperparamètres g, C et C_j.
- Le classifieur SVM nous retournera un modèle de classification, que nous allons utiliser pour prédire les étiquettes des classes de la base de validation en lui donnant comme paramètres d'entrée les données V_x . Le résultat de cette étape sera un vecteur de la même taille que V_y qui contient les "prédictions" du SVM pour la classe C_i contre toutes les autres classes.
- Une fois qu'on aura terminé ce processus pour toutes les classes, $C_i, i = 1, 2, ..., 10$, on aura une matrice M à 10 colonnes et le même nombre de lignes que V_y. Chaque ligne $i, i = 1, 2, ..., 10$ de M contient les prédictions du SVM pour la classe C_i contre toutes les autres classes. La classe de sortie finale sera calculée en prenant le

numéro de la colonne ayant la plus grand valeur. Le résultat de cette étape sera un vecteur P_y contenant les étiquettes prédites que nous allons comparer aux étiquettes "**ground-truth**" V_y pour calculer le taux d'erreur sur la base de validation : $\tau_{ev} = \frac{1}{n_v}\sum_{i=1}^{n_v}[P_y(i) \neq V_y(i)]$

- Nous allons répéter les étapes précédentes pour les toutes valeurs des hyperparamètres que nous voudrons tester, et ceux qui donnent la valeur minimum pour τ_{ev} seront nos hyperparamètres optimaux g^* et C^*
- Nous allons utiliser g^* et C^* pour évaluer le taux d'erreur sur la base de test τ_{et}

Nous avons utilisé la stratégie décrite précédente pour trouver les hyperparamètres optimaux. Les valeurs de l'erreur τ_{ev} sont données dans le tableau 8 En analysant les résultats du tableau 8, nous trouvons que l'erreur minimale sur

C / g	1	10	100	1000	10000
0.1	81.27	82.73	82.73	82.73	82.73
0.01	5.13	6.53	6.53	6.53	6.53
0.001	1.47	**1.13**	**1.13**	**1.13**	**1.13**
0.0001	5.07	2.4	1.73	1.73	1.73

TABLE 8 – Taux d'erreur τ_{ev} (%) en fonction des hyperparamètres g et c

la base de validation est : $min(\tau_{ev}) = 1.13\%$ ou, c'est-à-dire 17 éléments mal classés sur les 1500. Nous remarquons que la valeur optimale de $g^* = 0.001$ est unique, tandis que pour C^*, quatre valeurs sont possibles : $10, 100, 1000$ ou 10000

Nous allons donc fixer $g = 0.001$, et essayer d'autres valeurs plus précises de $C = 10^p, p = 0$ à 5, par pas de 0.5. Nous avons trouvé que $min(\tau_{ev}) = 1.13\%$ reste inchangée. Nous allons donc prendre la valeur de $C^* = 100$ pour la suite.

b.Évaluation du taux d'erreur sur la base de test

Dans cette section, nous allons évaluer le taux d'erreur sur la base de test τ_{et}. Nous allons utiliser la totalité de la base d'entrainement, c'est-à-dire les 6000 éléments obtenus après ACP. Les valeurs optimales des hyperparamètres de $g^* = 0.001$ et $C^* = 100$ déterminés précédemment seront utilisées pour entrainer le SVM. Les paramètres d'entrée de l'algorithme seront donc : La base d'entrainement globale, le type du noyau (toujours gaussien), $g = 0.001, c = 100$ et $C_j = 100$.
Le taux d'erreur trouvé sur la base de test en utilisant la totalité de la base d'entrainement est égal à **3.00%** ; c'est-à-dire 30 éléments mal classés sur les 1000 que contient la base de test. Cette erreur est plus faible que celle trouvée avec k-PPV qui était de **4.20%**, et encore plus que celle trouvée avec Bayes Quadratique (**5.4%**).
Ainsi, nous pouvons faire une première conclusions que l'algorithme basé sur les SVM possède une meilleure généralisation parmi les trois algorithmes étudiés dans ce laboratoire.
La matrice de confusion est donnée en figure 9

En analysant les résultats du tableau 9, nous pouvons faire les remarques suivantes :

- Le chiffre 2 est celui qui possède l'erreur de classification la plus élevée ; parmi les 100 éléments de la base de test qui représentent le chiffre 2, neuf ont été mal classés ; parmi eux huit classés comme 8 et un comme un 9.
- Les chiffres 1, 6, et 7 n'ont enregistré aucune erreur de classification.
- Le chiffre 5 a été incorrectement classé trois fois comme un 3, et le chiffre 8 a été incorrectement classé deux fois comme un 3

Ainsi, nous pouvons conclure que les chiffres 2, 8 et 5 sont ceux qui ont l'erreur de classification la plus grande.
Pour voir en détail ces éléments mal classés, nous avons affiché dans la figure 12 l'image originale associée au chiffre et la classe de sortie attribuée par le classifieur SVM.

Nous pouvons remarquer que les éléments mal classés qui représentent le chiffre 2 sont faciles à reconnaitre visuellement. Nous pouvons citer par exemple : **2/000609.tif**, **2/000611.tif**, etc. Quelques autres éléments sont aussi facilement reconnaissable visuellement (par exemple :**0/000619.tif**, **4/000619.tif**, **5/000661.tif**, etc.). En

Classe actuelle \ Classe estimée	0	1	2	3	4	5	6	7	8	9	Σe
0	97	2	1	0	0	0	0	0	0	0	**3**
1	0	100	0	0	0	0	0	0	0	0	**0**
2	0	0	91	0	0	0	0	0	8	1	**9**
3	0	1	0	98	0	0	0	0	1	0	**2**
4	0	0	1	0	97	0	2	0	0	0	**3**
5	0	0	0	3	1	95	0	1	0	0	**5**
6	0	0	0	0	0	0	100	0	0	0	**0**
7	0	0	0	0	0	0	0	100	0	0	**0**
8	0	1	0	2	1	0	0	1	94	1	**6**
9	0	0	1	0	0	1	0	0	0	98	**2**
	0	**4**	**3**	**5**	**2**	**1**	**2**	**2**	**9**	**2**	**30**

TABLE 9 – Matrice de confusion obtenue avec le SVM

FIGURE 12 – Exemples de quelques chiffres mal-classés avec SVM. La classe prédite est affichée en haut à gauche (en rouge) de chaque élément. Le nom du fichier original est aussi inscrit en bas à gauche (en bleu).

revanche, d'autres éléments mal classés sont effectivement assez difficile à identifier, même pour un humain. On peut citer les éléments suivants : **8/000634.tif**, **8/000698.tif**, **4/000693.tif**, etc.

Ainsi le chiffre 2 est souvent confondu avec un 8, malgré que visuellement, nous pouvons affirmer sans grande hésitation que c'est vraiment un 2. Quelques pistes de raisonnements peuvent expliquer cette situation. Nous pouvons par exemple penser aux transformations géométriques qu'une image peut subir comme la translation, la rotation, et l'ajout de bruit, qui peuvent avoir un impact sur la classification finale. Il y'a aussi la méthode d'extraction de caractéristiques qui n'est pas à négliger et qui peut aussi influencer le résultat final.

En comparant les résultats des figures 2, 7 et 12, nous retrouvons certains qui ont été mal classés par les trois algorithmes étudiés dans ce laboratoire, Quadratique Bayes, k-PPV et SVM. Parmi ces éléments, nous pouvons citer : 0/000693.tif, 3/000608.tif, 9/000618.tif, etc. Pour certains de ces éléments, les trois algorithmes attribuent la même classe de sortie (par exemple : 0/000694.tif classé comme un 1, 8/000698.tif classé comme un 4, etc.), mais pour d'autres éléments, il attribuent des classes de sorties différentes (exemple : 4/000621.tif classé comme un 2 par Quadratique Bayes et k-PPV, et comme un 6 par SVM, etc.)

La liste de ses éléments est affichée en figure 13

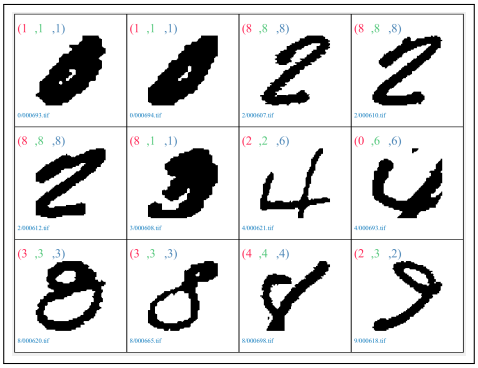

FIGURE 13 – Exemples de quelques chiffres mal-classés avec Bayes Quadratique, k-PPV et SVM. Chaque élément illustré est accompagné de sa classe prédite par Bayes Quadratique (premier chiffre entre parenthèses, en rouge), par k-PPV (deuxième chiffre entre parenthèses, en vert) et par SVM (troisième chiffre entre parenthèse, en bleu). Le nom du fichier original est aussi inscrit en bas à gauche (en bleu).

En effectuant cette première comparaison du taux d'erreur sur la base de test pour les trois algorithmes, nous pouvons conclure que les classifieurs Quadratique Bayes, k-PPV et SVM ont fait des erreurs communes sur 12 éléments de la base de test. C'est-à-dire, si nous pouvons réaliser une combinaison idéale des trois modèles, nous aurions atteint une erreur de classification de 1.2%, ce qui représente une excellente performance.

c. Graphe 3D de l'erreur sur la base de test

Dans cette section, nous faire évoluer les valeur des hyperparamètres C et g et calculer le taux d'erreur sur la base de test en utilisant la totalité de la base d'entrainement.

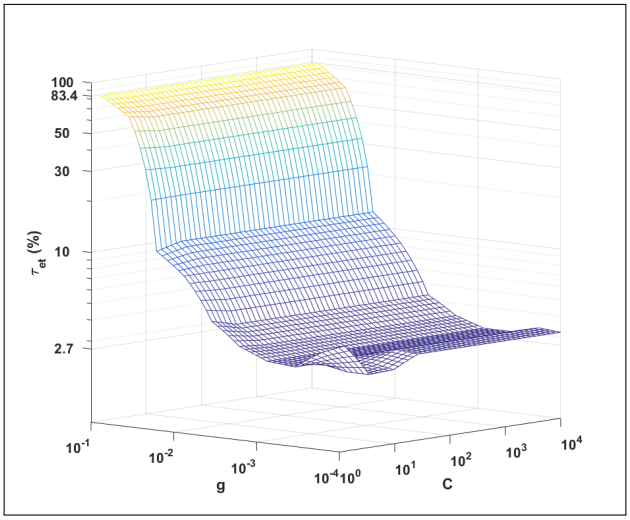

FIGURE 14 – Graphe 3D représentant la variation de l'erreur τ_{et} sur la base de test en fonction des hyperparamètres C et g.

Les valeurs prises pour C et g sont :

- $C \in [1, 5, 10^1, 3 \times 10^1, 7 \times 10^1, 10^2, 2 \times 10^2, 5 \times 10^2, 10^3, 3 \times 10^3, 6 \times 10^3, 10^4]$
- $g \in [10^{-4}, 2 \times 10^{-4}, 3, 5 \times 10^{-4}, 5 \times 10^{-4}, 7, 5 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, 5 \times 10^{-3}, 7.5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}]$

Le résultat trouvé est affiché en figure 14.

En analysant ce graphe, nous constatons :

- Pour des valeurs élevées de g, τ_{et} est assez grand (τ_{et} autour de $83.4 - 82.9\%$) quelque soit la valeur de C.
- Pour des plus petites valeurs de g, τ_{et} diminue pour atteindre le minimum $\min(\tau_{et}) = 2.7\%$, pour $g = 5 \times 10^{-3}$ et $C = 5$, valeur qui est aussi atteinte pour $g = 3.5 \times 10^{-3}$ et $C = 5$ ou $C = 10$.
- Quand g devient encore plus petit $\leq 10^{-4}$, τ_{et} est inversement proportionnel par rapport à la valeur de C, pour $C \leq 200$ et reste ensuite constant pour $C \geq 500$.

Nous constatons que nous obtenons une bien meilleure erreur sur la base de test en ayant essayé d'autres valeurs des hyperparamètres g et C. Nous pouvons dire aussi que le paramètre g du noyau Gaussien est celui qui a la plus grande influence sur le taux d'erreur dans notre exemple. Ceci est compréhensible vu que le noyau Gaussien est défini comme une fonction de la distance (ou de similarité) en les éléments x_i et x_j avec le paramètre g qui contrôle le seuil 'partir duquel nous pouvons dire si les deux éléments x_i et x_j appartiennent au non à la même classe.

V Comparaison des trois algorithmes

Dans cette section, nous allons comparé les trois algorithmes étudiés sur les critères suivants :

- **La capacité de généralisation** : voir si le modèle se généralise bien à de nouvelles données. Ceci sera principalement mesuré en comparant les erreur sur la base de test.
- **Le temps de la phase d'apprentissage** : Certains classifieurs, comme k-PPV ne possèdent pas de phase d'apprentissage proprement dite, donc on va intégrer la phase de validation pour la sélection des paramètres optimaux.
- **Le temps de la phase de test** : Le temps qu'il faut à l'algorithme pour classifier tous les éléments de la base de test.
- **La capacité de mémoire de stockage des paramètres du modèle** : La mémoire physique nécessaire pour stocker les paramètres du modèle.
- **L'influence du nombre de composantes principales** : Le nombre de vecteurs sur lesquels nous projetons nos données
- **Influence du bruit** : Évaluer les performances des trois modèles si du bruit est rajouté aux images de départ.

Pour l'environnement informatique, nous avons utilisé le langage de programmation Matlab version 2016a avec les bibliothèques de traitement d'image et de vision par ordinateur. Pour le classifieur SVM, nous avons utilisé la librairie open source SVMlight. Le système d'exploitation est Windows 10, avec un processeur Intel Core i5-4770 @ 3.20 GHz et 24 Go de RAM.

V.1 Capacité de généralisation

Dans un problème de classification, ce qui nous intéresse plus particulièrement est d'avoir une bonne capacité de généralisation du classifieur. Nous savons que par le principe de minimisation du risque empirique, nous entrainons un modèle, c'est-à-dire nous adaptons ses paramètres pour avoir au final le taux d'erreur le plus faible sur l'ensemble d'entrainement. Mais, ce qui nous importe vraiment est de bien généraliser sur de nouveaux exemple que le classifieur n'a jamais vu auparavant. Ainsi, le fait que nous choisissant en quelque sorte les paramètres du modèle pour minimiser l'erreur sur l'ensemble d'entrainement, celle-ci sous-estime l'erreur de généralisation. De ce fait, l'erreur d'entrainement n'est pas un bon estimé de l'erreur de généralisation. Nous disons dans ce cas que c'est un estimateur biaisé.
Dans ce laboratoire, nous avons donc étudié trois modèles de classification : le Bayes Quadratique, le k-PPV (k plus proches voisins) et la Machine à vecteur de support (SVM). Dans le tableau 10 sont résumé le taux d'erreur obtenu sur la base de test.

Modèle	τ_{ev}	τ_{et}
Bayes Quadratique	N.A.	5.4%
k-PPV	1.93% (avec $k = k_{opt} = 1$ pour toutes les classes) 1.20% (avec $k = k_{opt}$ pour chaque classe)	4.20%
SVM	1.73%	3.00%

Table 10 – Capacité de généralisation des modèles étudiés

Nous devons faire une première remarque : Comme nous avons supposé que les données suivent une certaine distribution pour le modèle Bayes Quadratique, celui-ci ne comporte pas d'étape de validation. Pour les deux autres algorithmes, k-PPV et SVM, nous avons effectué une validation simple, ainsi qu'une validation croisée avec 4 blocs, pour estimer les hyper-paramètres qui minimisent l'erreur sur la base d'entrainement. Nous constatons que l'erreur la plus élevée obtenue sur la base de test ($\tau_{et} = 5.4\%$) est celle du modèle de Bayes Quadratique. Avec l'algorithme k-PPV, une erreur de $\tau_{ev} = 1.20\%$ a été obtenue à l'étape de validation, mais en utilisant la même valeur de k_{opt} sur la base de test, l'erreur a augmenté pour atteindre $\tau_{et} = 4.20\%$, c'est-à-dire près de 3 fois l'erreur de validation. Pour le modèle SVM, l'erreur de validation obtenue avec les hyper-paramètres optimaux a été de $\tau_{ev} = 1.73\%$, et

celle sur la base de test est de $\tau_{et} = 3.00\%$, moins de 2 fois l'erreur de validation.

Nous pouvons constater que l'erreur sur la base de test τ_{et} est tout le temps plus élevée que l'erreur de validation τ_{et}. Ceci est probablement du à la nature de la base de test dont les éléments ont été choisis de telle manière pour que ça soit extrêmement difficile d'atteindre un taux de reconnaissance de 100% et ainsi mieux tester la robustesse de nouveaux algorithmes et encourager la communauté scientifique pour améliorer les modèles existants. Nous pouvons donc conclure que le modèle SVM est celui qui possède la plus grande capacité de généralisation puisque ses performances de classifications ne se dégradent pas beaucoup en l'appliquant à de nouveaux éléments.

V.2 Temps de calcul

Le temps de calcul, ou pour parler d'un terme plus appropriée, la complexité d'un algorithme est toujours d'une importante cruciale pour tout problème de classification. L'aspect temps réel est considéré dans plusieurs applications pratiques, et ainsi c'est primordial d'évaluer les performances des modèles en termes de complexité et de temps de calcul. Pour notre cas, nous avons résumé le temps d'exécution des trois modèles étudiés dans ce laboratoire, en séparant les données par phases : la phase d'entrainement qui peut être composée d'une phase d'apprentissage et une phase de validation, où nous entrainons un modèle pour minimiser l'erreur sur les éléments d'entrainement. Et une phase de test pour évaluer l'algorithme sur de nouveaux éléments. Le tableau 15 et la figure 16 montrent les résultats obtenus.

		Entrainement		Test
		Apprentissage	Validation	
Bayes Quadratique		0.0038		0.0362
k-PPV	Validation simple	1.6244	0.4866	4.7599
	Validation croisée	6.6531	1.9464	
SVM	Validation simple	3207.2	2578.1	26.1998
	Validation croisée	12789	10292	

TABLE 15 – Temps de calcul des trois modèles étudiés

Si nous prenons le temps d'exécution du modèle Bayes Quadratique comme référence, nous constatons que l'algorithme k-PPV est plus de 2 ordres de grandeur plus lent que Bayes Quadratique. Le modèle SVM est de près de 3 ordres de grandeurs plus long que k-PPV et de plus de 5 ordres de grandeur plus lent que Bayes Quadratique. Il ne faut que $3,8$ ms pour le modèle Bayes Quadratique pour l'entrainement, et 36.2 ms pour la phase de test. Un temps pareil est extrêmement petit, et pour des applications temps réel ou l'aspect performance optimale n'est pas très important, Bayes Quadratique est tout-à-fait adapté. Pour le modèle k-PPV, le temps de la phase d'apprentissage et validation est de 2.111, et 4.7599 s pour le test. Ce temps reste très intéressant surtout que pour k-PPV, nous n'avons besoin de faire une hypothèse sur la nature des données. Ainsi, k-PPV reste un algorithme de choix pour de nombreux problèmes de classification. Pour ce qui est du SVM, les temps d'exécution pour la phase d'apprentissage et validation est assez conséquent ; en utilisant seulement la validation simple, il faut environ 5785.3 s pour entrainer le modèle et obtenir les hyperparamètres optimaux, c'est-à=dire plus de 1 heure et 36 minutes ! ! c'est une vitesse d'exécution extrêmement lente qui ne permet de considérer aucune application temps réel. C'est vrai que nous parlons ici que du temps d'apprentissage et validation, phase qui est en général, exécutée une seule fois. Le temps de la phase de test est lui beaucoup plus intéressant, il n'est "que"de 26.1998 s pour classifier 1000 éléments, ce qui peut être considéré acceptable pour certaines applications.

C'est important de mentionner que si nous utilisons la validation croisée avec seulement 4 blocs, le temps pour la phase d'apprentissage et de validation augmente considérablement. Par exemple, il atteint autour de 6 heures et 25 minutes. À la lumière de ces résultats, nous pouvons conclure que l'algorithme de Bayes Quadratique est de loin le plus rapide des trois modèles étudiés. Il possède des performances de calcul adaptées aux applications temps réelles les plus exigeantes. Les performances de k-PPV sont également intéressantes, surtout qu'une hypothèse sur la distribution des données n'est pas requise. Des trois algorithmes, le SVM est de loin le plus lent des trois, spécialement pour la phase d'apprentissage et validation. Vu que cette dernière étape est généralement exécutée une seule fois, et les paramètres du modèles sont sauvegardés pour une utilisation ultérieure, c'est un algorithme assez intéressant au niveau du taux d'erreur de la classification.

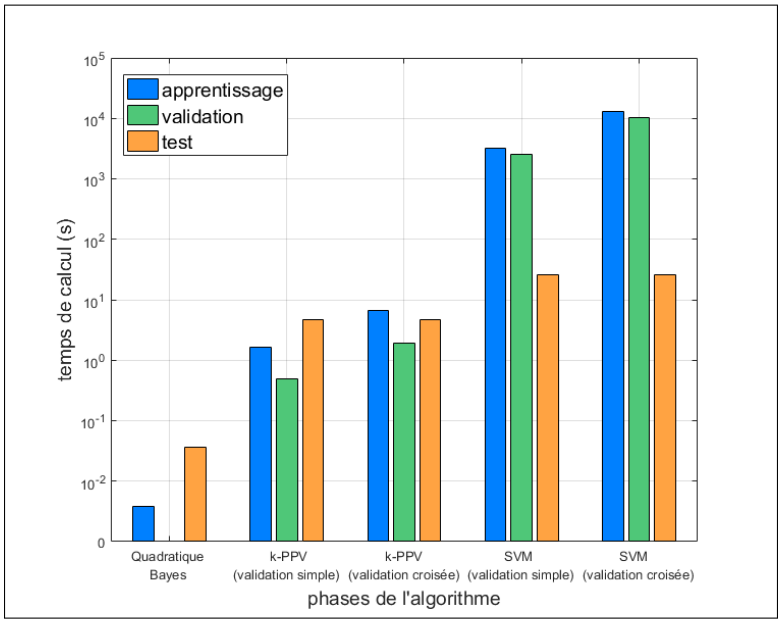

FIGURE 16 – Temps de calcul des trois modèles étudiés

V.3 Stockage des paramètres

L'autre aspect important dans un problème de classification est l'espace mémoire disponible pour stocker les données d'entrainement, de test, de calcul et les paramètres du modèle. Pour des applications sur des systèmes embarqués où l'espace de stockage est souvent limité, trouver des modèles qui ne nécessitent pas beaucoup de mémoire est primordial. Pour évaluer les performance des trois algorithmes pour cet aspect, nous avons afficher dans le tableau 17 le nombre et la tailles des éléments en octets nécessaire au bon fonctionnement des algorithmes. Avec le logiciel Matlab, toute variable peut être considérée comme une matrice, mais aussi possède plusieurs structures de données utilisées pour stocker l'information. Les type de variables les plus courantes que nous avons utilisé pour stocker nos données sont les matrices, les vecteurs, les cellules, les structures (struct) et les variables simples. En Matlab, une variable simple double précision occupe en général 8 octets (Bytes en anglais). Un vecteur à n nombres double précision occupent $8 \times n$ octets.

	nombre et taille des éléments (octets)											
	matrices		**vecteurs**		**cellules**		**structures**		**valeurs singulières**		**nombre d'éléments**	**Taille totale**
Bayes Quadratique	4	2,468,320	4	16,160	3	313,200	0	0	10	80	21	**2,797,760**
k-PPV Validation	8	60072000 (59928000)	7	132,144	3	83,056	0	0	12	96	30	**60287296 (60143296)**
k-PPV Test	5	52,576,800	4	112,000	1	912	0	0	10	80	20	**52,689,792**
SVM Validation	6	4536000 (4392000)	10	165,160	3	12,856	1	1,575,818	19	144	39	**6289978 (6145978)**
SVM Test	4	2,544,800	9	132,160	2	744	1	287,946	13	124	29	**2,965,774**

TABLE 17 – Mémoire de stockage des trois modèles étudiés - Détails par phases et types de données

Nous pouvons constater en analysant les résultat du tableau 17, que le modèle de Bayes Quadratique est celui qui nécessite l'espace mémoire le moins élevé des trois algorithmes. Il nécessite environs 21 éléments de stockage pour un espace total de près de 2.8 Mo (Mega octets). Il faudrait environ 113 Mo pour stocker les données et les paramètres avec le modèle k-PPV, et 9.3 Mo pour l'algorithme SVM.

Si nous nous intéressons au stockage en fonction des phases de classification, nous pouvons remarquer à partir des résultats du tableau 19 qu'une fois de plus, le Bayes Quadratique est celui qui nécessite la mémoire de stockage

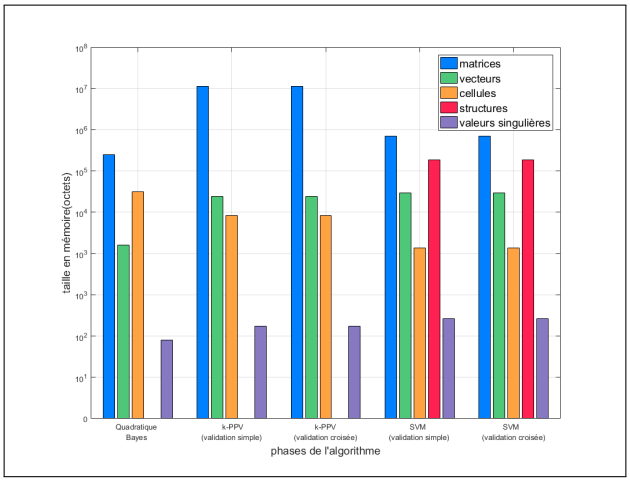

FIGURE 18 – Mémoire de stockage des trois modèles étudiés

la plus petite pour la phase de test qu'est uniquement 9 Ko. Le SVM nécessite quant à lui environ 2.97 Mo et le k-PPV, autour de 52.69 Mo.

	Entrainement	Test	Total
Qbayes	2,503,712	9,000	2,512,712
KNN	60,287,296	52,689,792	112,977,088
SVM	6,289,978	2,965,774	9,255,752

TABLE 19 – Mémoire de stockage des trois modèles étudiés

V.4 Influence du nombre de composantes principales de l'ACP

Dans cette section, nous allons analyser les performances des 3 algorithmes en fonction du nombre de composantes principales N_c choisies avec PCA. Nous avons fait varier les valeurs de $N_c = 10, 18, 33, 44, 60, 80, 98$, et nous avons mesurer le taux d'erreur, le temps de calcul pour les 3 modèles, ainsi que la valeurs des paramètres k pour le k-PPV, et g et C pour le SVM. Les résultats trouvés sont résumés dans le tableau 21 et aussi tracés dans la figure 20. Il est important de mentionner que dans cette section, nous avons utiliser une autre librairie SVM, qui s'appelle LIBSVM. Elle est disponible gratuitement, et une interface est également fournie. Nous n'avons pas pu continuer d'utiliser la librairie SVMLight car le temps de calcul pris pour une seule configuration est beaucoup trop grand. Aussi, elle avait un autre souci ; c'est que Matlab crash souvent en invoquant un problème d'accès à la mémoire.

Nous remarquons que le taux d'erreur d'erreur avec le modèle de Bayes quadratique augmente considérablement quand le nombre de composantes dépasse environ 80. Ainsi, nous pouvons voir l'intérêt de l'application de l'ACP. l'ACP ne permet pas seulement de réduire la dimension des vecteurs caractéristiques, mais elle permet aussi d'éliminer les composantes qui peuvent influencer sur la nature Gaussienne des données, essentielle pour que Quadratique Bayes fonctionne bien. De l'autre coté, les performances de k-PPV et SVM sont assez similaires, quoique le k-PPV est légèrement meilleur. Nous voyons là aussi un autre bénéfice de l'ACP, le taux d'erreur final sur la base de test est meilleur lorsque le nombre de composantes principales est situé autour de la valeur $N_c = 44$ que nous avions choisie dès le départ.
Comme nous l'avons mentionné précédemment, dans cette partie, nous avons utilisé une autre libraire qui s'appelle LIBSVM, et c'est pour ça que nous voyons que le taux d'erreur (4.30%) sur la base de test pour $N_c = 44$ est différent de celui trouvé avec la version SVMLight qui était de 3.00%.

Pour voir l'influence de N_c sur le temps de calcul, nous avons estimé le temps de calcul de deux phases, l'entrainement et le test. Les résultats sont affichés dans la figure 22.

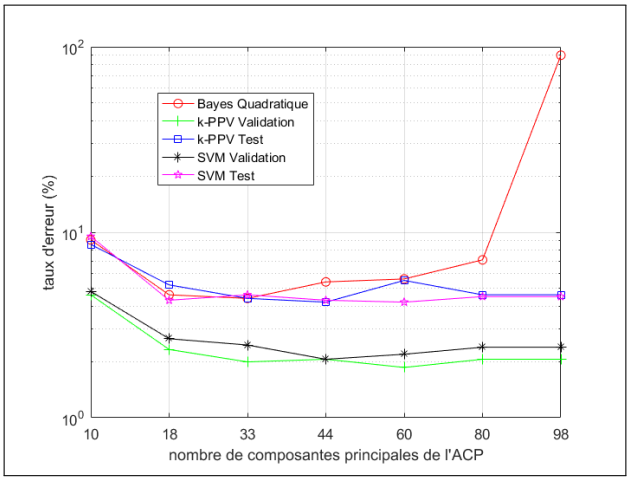

FIGURE 20 – Taux d'erreur en fonction du nombre de composantes principales.

	Nombre de composantes principales						
	10	**18**	**33**	**44**	**60**	**80**	**98**
Quadratique Bayes	9.1	4.6	4.4	5.4	5.6	7.1	90
k-PPV Validation	4.6	2.33	2	2.07	1.87	2.07	2.07
k-PPV Test	8.6	5.2	4.4	4.2	5.5	4.6	4.6
SVM Validation	4.8	2.67	2.46	2.07	2.2	2.4	2.4
SVM Test	9.5	4.3	4.6	4.3 (3)	4.2	4.5	4.5

TABLE 21 – Valeurs du taux d'erreur en fonction du nombre de composantes principales.

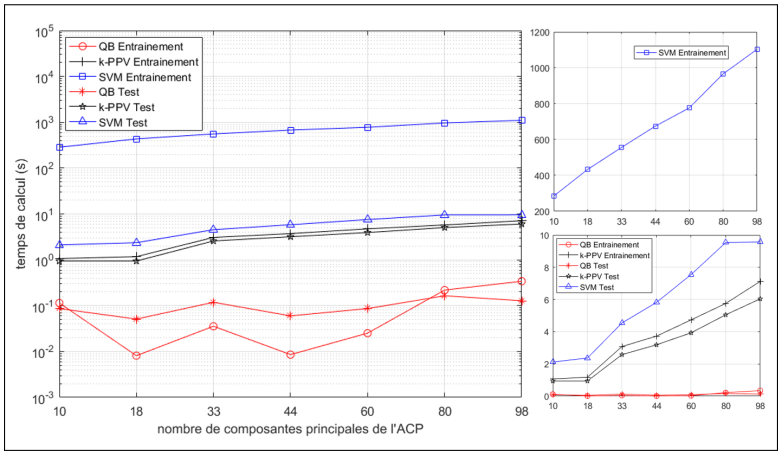

FIGURE 22 – Taux d'erreur en fonction du nombre de composantes principales.

En analysant ces résultats, nous pouvons voir que Quadratique Bayes est de loin le plus rapide, et son temps de calcul est relativement stable en fonction de N_c. Le temps de calcul pour les deux autres algorithmes, k-PPV et SVM, et pour les deux phases, est proportionnel au nombre de composantes principales prises. Par exemple, quand nous passons de $N_c = 44$ à $N_c = 98$, le temps de calcul est presque multiplié par 2. Nous constatons aussi que la phase d'entrainement-validation du SVM est celle qui prend le plus de temps, avec des temps de calcul de près de 2 à 3 ordre de magnitude que le k-PPV, et jusqu'à 5 ordres de grandeur que Bayes Quadratique.
Nous pouvons conclure que pour le temps de calcul, Bayes Quadratique est de loin le plus intéressant, surtout que sont de calcul ne dépend pas, à priori, du nombre de composantes principales utilisées.
Pour ce qui est l'influence du nombre de composantes principales sur la valeur optimale du paramètre k dans le k-PPV et des deux hyperparamètres, g et C pour le SVM, nous avons trouvé que les valeurs de g^* et C^* sont

toujours les mêmes quelque soit la valeur de N_c. Pour k, nous avons trouvé $k_{opt} = 1$ pour toutes les configurations, excepté pour $N_c = 60$, où $k_{opt} = 3$. Ainsi, comme conclusion partielle pour cette sous-section, nous pouvons dire que le nombre de composantes principales N_c a une influence considérable sur le temps de calcul. Doubler N_c revient à peu près à doubler le temps de calcul du k-PPV et du SVM. L'erreur de classification aussi est largement dépendante de N_c. Quand N_c est trop petit, l'erreur de classification des trois algorithmes est assez élevée. À mesure que N_c croit à partir d'une certaine valeur, l'erreur commence à diminuer pour atteindre une valeur minimale pour N_c autour de $40 - 60$ dépendamment des configurations. À partir de N_c autour de 80, l'erreur de k-PPV et SVM se stabilise, tandis que celle de Bayes Quadratique croit en exponentielle. Ainsi, lorsque N_c est trop grand, la nature gaussienne des données n'est plus vérifiée. Avant de terminer cette section, il est important de mentionner que chaque composante de l'ACP prise dans les données, va nécessiter un espace de stockage supplémentaire pour les paramètres du modèle. Nous avons analysé les paramètres des modèles qui dépendent du nombre N_c, et nous avons trouvé les résultats suivants :
Pour chaque composante principale supplémentaire, il faudrait pour :

- Quadratique Bayes : 56 Kilo-octets
- k-PPV : 232 Kilo-octets
- SVM : 156 Kilo-octets

Nous voyons donc que le nombre de composantes principales a une plus influence au niveau du stockage pour le k-PPV que les deux autres modèles. C'est un aspect important à prendre en compte pour la sélection du nombre adéquat de N_c.

V.5 Influence du bruit

Dans cette section, nous allons essayer de rajouter du bruit aux images de départ, et voir son influence sur les performances des trois modèle. La figure 23 montre un exemple du chiffre 0 avec du bruit de différentes variances. Nous voyons, que lorsque la valeur de la variance est petite, le chiffre est parfaitement reconnaissable. À mesure que la variance du bruit augmente, le chiffre devient de plus en plus difficile à reconnaitre. Avec une variance de 1, l'image départ est à presque 100% que du bruit.

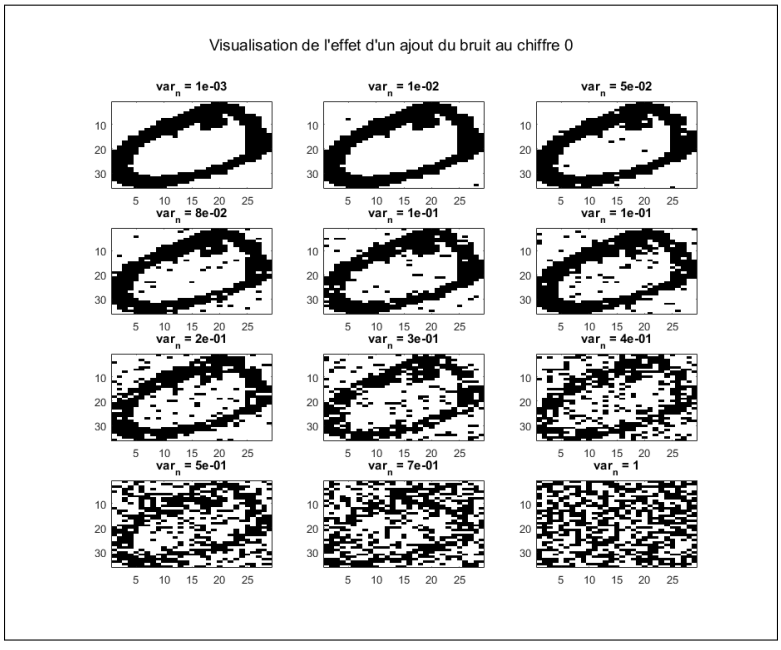

FIGURE 23 – Exemple d'un ajout de bruit sur les images de départ

Dans cette optique, et pour voir l'influence du bruit uniquement, nous avons gardé les paramètres que nous avons pris pour les sections précédentes : nombre de composantes principales $N_c = 44$, valeur de $k = 1, 3, 5, 7, 9, 11$ pour k-PPV, $g = 0.0001, 0.001, 0.01, 0.1$ et $C = 1, 10, 100, 1000, 10000$ pour le SVM. La figure 24 montre l'erreur pour les

différentes phases des trois algorithmes en fonction de la variance du bruit rajoutée. Nous constatons que quand la valeur de la variance est petite (≤ 0.2), l'erreur de test pour les 3 algorithmes est relativement stable. À partir d'une variance d'environ 0.5, les performances des 3 algorithmes se dégradent rapidement avec une fonction exponentielle, pour presque atteindre l'erreur 100% pour une variance de bruit proche de 1. Nous remarquons que les performances du SVM se dégradent plus vite que les autres à partir de la valeur de variance égale à 0.5.
Il est important de noter que l'erreur sur la base de validation est toujours plus petite pour la plupart des valeurs de la variance, ce qui compréhensible vu que nous avons optimiser les paramètres du modèle pour justement minimiser l'erreur de classification sur les données d'entrainement.

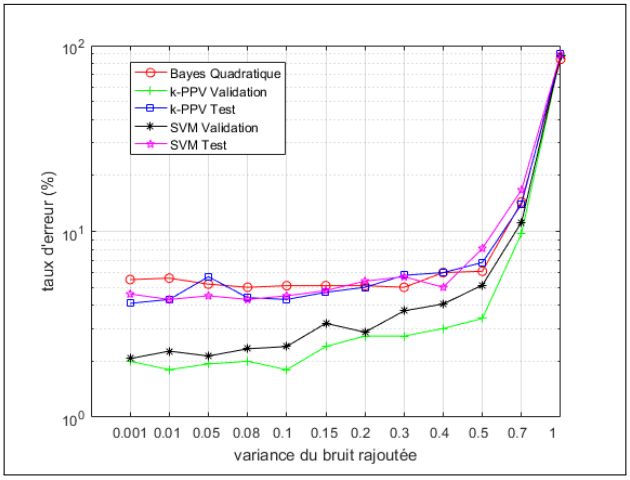

FIGURE 24 – Taux d'erreur en fonction de la variance du bruit

Pour bien comprendre le résultats de le l'ajout du bruit aux images d'entrée, nous avons calculé le taux de recouvrement des deux bases d'apprentissage et de test en fonction de la variance du bruit. La figure 25 montre le résultat obtenu.

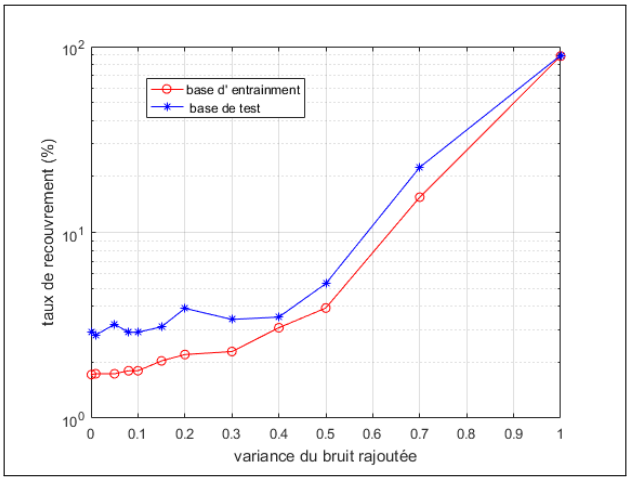

FIGURE 25 – Taux de recouvrement en fonction de la variance du bruit

Comme nous pouvons s'y attendre, le recouvrement entre les classes dépend fortement de la variance du bruit. Plus la variance augmente, plus le recouvrement devient conséquent. Ce que nous pouvons constater aussi, est que le recouvrement de la base de test est tout le temps plus grand que le recouvrement de la base d'apprentissage. Nous pouvons expliquer ça par le fait que la base de test a été mise en œuvre pour qu'elle contienne des éléments plus difficiles à classer, ce qui permet de mieux évaluer la robustesse des algorithmes de classification.

Nous avons également estimé le temps de calcul en fonction de la variance du bruit. La figure 26 montre le résultat

obtenu.

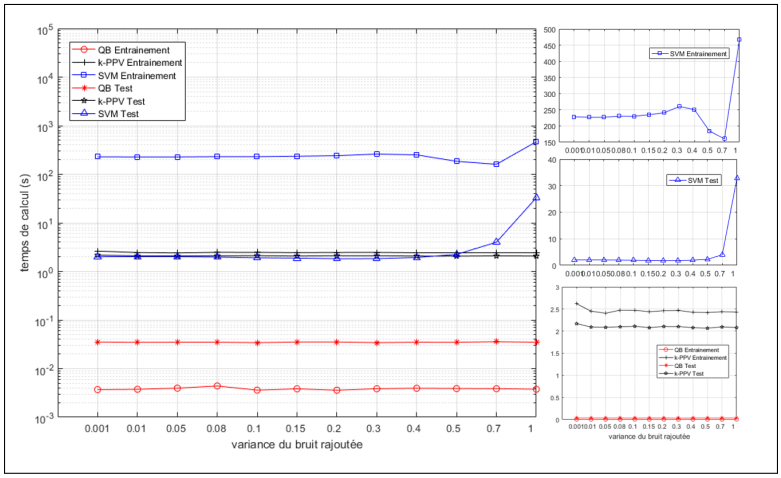

FIGURE 26 – Temps de calcul en fonction de la variance du bruit

Nous remarquons que le temps de calcul des 3 algorithmes dans leurs différentes phases, est relativement constant. L'influence de la variance est très faible, excepté pour le SVM où nous voyons une brusque augmentation à partir de la valeur 0.7 pour la variance, pour la phase de test. Pour le SVM au niveau de la phase d'entrainement-validation, nous constatons une diminution du temps en les valeurs de 0.3 et 0.7, et après ça, une forte augmentation. Ceci peut s'expliquer par le choix des valeurs de g et C qui ont une influence sur le temps de calcul et qui sont influencé par la quantité de bruit.

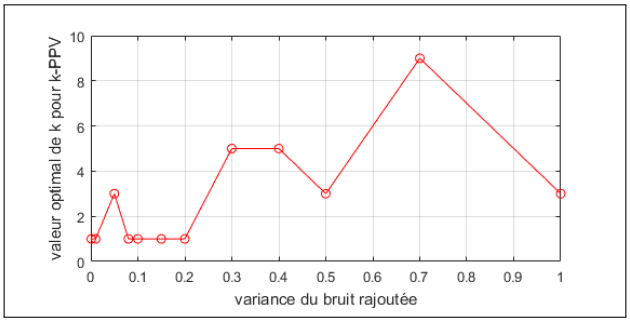

FIGURE 27 – Valeur optimale de k pour k-PPV en fonction de la variance du bruit

Pour ce qui est des valeurs optimales de g et C pour le SVM, nous avons trouvé exactement les mêmes valeurs précédentes, et ce quelque soit la valeur de la variance. Cependant, pour k, la valeur optimale est fortement dépendante de la variance. La figure 27 montre la valeur de k_{opt} en fonction de la variance du bruit. Nous pouvons constater que pour les petites valeurs de la variance, k optimal est proche de la valeur trouvée sans le bruit ($k_{opt} = 1$). Quand la variance du bruit augmente, la valeur de k_{opt} augmente aussi. Ceci peut s'expliquer par le fait que l'ajout du bruit à l'image fait en sorte que les éléments appartenant à des classes différentes peuvent se rapprocher, et le k-PPV a besoin de trouver plus de voisins avant de décider de la classe de sortie.
Nous pouvons donc conclure que le rajout du bruit a une certaine influence sur les performances de l'algorithme. Si la valeur de la variance est petite, les algorithmes étudiés arrivent à résister. Quand la variance dépasse un certain seuil, les performance des 3 algorithmes se dégradent brusquement. Enfin, notons que le SVM est le plus influencé quand la variance du bruit dépasse un certain seuil (autour de $0.4 - 0.5$ environ).

VI Conclusion

Dans cette deuxième partie de laboratoire, nous avons étudié trois algorithmes de classification : le Bayes Quadratique, le k plus proches voisins et la Machine à vecteurs de support (SVM). Dans la première partie du laboratoire, nos avons extrait les vecteurs caractéristiques des données, et réduit leur dimension en appliquant l'ACP. Pour évaluer les performances des algorithmes, nous avons considéré plusieurs aspects. Parmi les plus importants dans la littérature est le taux d'erreur sur la base de test, qui mesure la capacité de généralisation des modèles de classification. Pour être plus complet et aussi montrer d'autres aspects aussi important que le taux d'erreur et qui sont souvent rarement mentionné dans les papiers scientifiques, le temps de calcul, la complexité de l'algorithme ainsi que l'espace de stockage nécessaire pour son bon fonctionnement et l'influence du rajout de bruit aux images de départ sur les performances des 3 modèles. Dans les applications pratiques où l'aspect temps réel est important, négliger la complexité de l'algorithme et se focaliser uniquement sur le taux d'erreur est souvent contre productif. Nous vivons dans une époque où la taille des disques durs d'ordinateur est souvent mesurée en Tera-octets, mais ils existent certains domaines nécessitant des solutions embarquées où souvent l'espace de stockage est très limité. Dans ce cas, le critère à ne pas négliger est l'espace mémoire nécessaire pour stocker les paramètres.

Pour mesurer les performances des algorithmes au niveau de l'erreur de classification, nous avons opté pour l'apprentissage supervisé. Nous disposons donc des étiquettes "**ground-truth**" pour les données de test qui nous permettent de comparer aux prédictions des trois modèles. Pour certains algorithmes, le modèle de classification possède des paramètres qu'il faudrait estimer. C'est le cas par exemple pour le k-PPV où le nombre des plus proches voisins k doit être estimé pour minimiser l'erreur sur la base d'entrainement. Cette phase de validation doit être fait en décomposant la base d'entrainement en deux sous-ensembles : un pour l'apprentissage des paramètres, et un autre pour la validation. Nous avons opté pour une décomposition de 75% pour la base d'apprentissage et 25% pour la base de validation. Nous avons aussi expérimenter avec l'approche de validation croisée en utilisant 4 blocs de taille égale, 3 pour la l'apprentissage et le quatrième pour la validation. Une permutation des blocs est réalisée pour les utiliser tous les éléments dans la phase de validation. De cette étape, les valeurs optimales des paramètres ont été obtenues et qui seront utilisées pour la phase de test.

Au niveau du temps d'erreur sur la base de test, le modèle SVM est celui qui possède l'erreur de classification la moins élevé. Par contre, c'est aussi le plus lent des trois algorithmes au niveau du temps de calcul. Le modèle basé sur Bayes Quadratique est celui qui est le plus rapide des trois algorithmes. Il 2 et 5 ordre de grandeur plus rapide que le k-PPV et SVM, respectivement. Cependant, c'est aussi l'algorithme qui possède l'erreur de classification la plus élevée. Pour le modèle k-PPV, ses performances niveau de l'erreur de classification et le temps de calcul sont en quelque sort entre celles des deux autres algorithmes. Par contre, l'espace de mémoire nécessaire pour stocker les paramètres est de loin le plus élevé des trois algorithmes.

Nous avons aussi analyser l'importance de l'étape de réduction de la dimension de l'espace des caractéristiques. L'analyse en composantes principales (ACP) ne permet pas seulement de gagner en temps de calcul et en mémoire de stockage, mais également d'améliorer le taux d'erreur sur la base de test. Également, son influence est particulièrement visible sur le taux d'erreur du modèle Bayes Quadratique que nous avons vu augmenté à plus de 80% quand le nombre de composantes principales est élevé. Ainsi, nous pouvons constater que chaque algorithme possède ses forces et ses faiblesses. Le choix du modèle de classification dépend grandement de la nature des données qu'on a, des exigences de l'application au niveau de l'aspect temps réel, au niveau des solutions embarquées et du taux de classification exigée. Bayes Quadratique convient pour les deux aspects, temps réel et solution embarquée, tandis que SVM est le modèle qui possède la meilleure capacité de généralisation.

Nous avons aussi constaté en analysant les éléments mal classifiés par les 3, certaines sont effectivement difficilement reconnaissables visuellement, par contre d'autres, comme les chiffre 5 et 2 sont parfaitement identifiables. Construire un modèle de classification automatique est un travail extrêmement difficile, et doit débuter impérativement par une analyse exhaustive des données ; leur compréhension est une des clés de la réussite. Plusieurs étapes sont nécessaires pour avoir un modèle robuste et qui se généralise bien aux nouvelles données, et des erreurs peuvent se produisent à chacune de ces étapes. La nature des données, les transformations géométriques, l'ajout de bruit et les exigences du problème peuvent tous affecter les performances de la classification.