



Reconnaissance de formes et inspection SYS800

Reconnaissance optique de caractères

Labo 2 : Algorithmes de classification

0 1 2 3 4 5 6 7 8 9

Rémy Carpentier

École de technologie supérieure

Montréal, Canada

Automne 2017

1. Introduction

L'objectif de ce laboratoire est de nous faire coder un programme capable de reconnaître l'écriture manuscrite de chiffres. Ce sujet se décompose en deux étapes : premièrement, à partir des images, extraire des caractéristiques discriminantes entre les classes. Deuxièmement, coder et comparer les performances de différents algorithmes de classification.

Comme nous venons de le dire, ce deuxième laboratoire sera consacré à la mise en place de trois algorithmes de classification qui sont le classificateur Bayes quadratique, le K-nn et le SVM. Dans un second temps, nous allons comparer les différents résultats obtenus avec ses trois classifieurs en ce qui concerne la rapidité, l'efficacité et les ressources utilisées.

La suite du rapport est composée comme suit, brièvement expliquer les données à notre disposition, détailler le fonctionnement de chacune des trois méthodes et commenter nos codes correspondants. Nous allons vous expliquer comment notre programme s'articule. Ensuite, les résultats obtenus seront présentés et discutés. Pour finir, nous vous présenterons notre méthode pour combiner les classificateurs et ainsi améliorer la précision.

2. Bases de données

Nous disposons donc deux bases de données : *ACPFeatures_Rt* et *ACPFeatures_Pr*, pour les méthodes respectives rétine et profils développées dans le premier laboratoire. La méthode ACP a été appliquée aux deux bases de données pour réduire leurs dimensionnalités et ainsi réduire la malédiction de la dimensionnalité. Ces bases de données sont composées de 6000 images pour l'entraînement et 1000 images pour le test. Le nombre de classes est 10, soit les chiffres entre 0 et 9. Pour la méthode rétine, il y a 37 caractéristiques par images et pour la méthode profils, il y en a 27. On retrouve toutes ses informations dans les quatre matrices suivantes :

- *train_ACP*, matrice de 6000 x (Nb de caractéristiques)
- *trainlabels*, matrice de 1 x 6000
- *test_ACP*, matrice de 1000 x (Nb de caractéristiques)
- *testlabels*, matrice de 1 x 1000.

À noter que nous ne réutiliserons pas nos bases de données obtenues lors du premier laboratoire. En effet, pour que tous les groupes aient le même point de départ, nous avons tous reçu les mêmes données acquises avec un seul code. Ces nouvelles données sont les caractéristiques extraites des images selon les deux méthodes vues précédemment, à savoir la méthode profile et la méthode rétine, puis d'un ACP appliqué aux deux méthodes.

Deuxième remarque, les trois méthodes que nous allons utiliser (Bayes, K-nn et SVM) ont besoin de données labélisées, des données dont on connaît la classe au préalable. Ce type d'apprentissage est dit supervisé.

3. Algorithmes de classification

3.1. Bayes Quadratique

Notre premier algorithme de classification est le Bayes Quadratique. C'est une approche paramétrique qui repose sur l'hypothèse que la distribution des données de chaque classe suit une loi normale multivariable. L'ensemble des équations nécessaire pour réaliser ce classificateur nous ont été données dans l'énoncé du deuxième laboratoire. Nous avons donc modifié la dernière équation, l'équation (4), en supprimant le premier terme étant donné que toutes les classes sont équiprobables. L'ensemble du classificateur Bayes quadratique se résume dans l'équation suivante pour notre cas :

$$d_j(x) = -\frac{1}{2} \ln(|\hat{\Sigma}_j|) - \frac{1}{2} (x - \hat{\mu}_j)^T \hat{\Sigma}_j^{-1} (x - \hat{\mu}_j)$$

Le premier terme $\hat{\Sigma}_j$ correspond à la covariance de chaque classe, soit 10 fois la covariance de matrice de 600 individus par leur nombre de caractéristique (27 ou 37 selon le choix de la méthode, profile ou rétine). Le terme x correspond au nouvel individu testé. Il est représenté par un vecteur de la taille du nombre de caractéristiques. Enfin, le troisième terme $\hat{\mu}_j$ est la valeur moyenne de chaque caractéristique pour chacune des 10 classes. Ce classificateur n'a pas besoin d'être entraîné au préalable puisqu'il n'a pas de paramètres à fixer.

3.2. K-nn

Le K-nn, pour K-nearest neighbors, est notre deuxième algorithme de classification. Ce classificateur ne fait aucune hypothèse sur la distribution des données. Il ne fait qu'observer une donnée parmi toutes les données. On nomme les classificateurs de ce type, non paramétrique.

Son fonctionnement est le suivant : soit Y un individu de la base de test. On calcule la distance euclidienne (eq. 2) entre Y et tous les individus X de la base d'entraînement. On attribue à Y le label majoritaire parmi ceux de ses k voisins les plus proches, autrement dit avec la distance la plus faible. On remarque ici que le paramètre k, le nombre de voisins pris en compte, est un paramètre à fixer par l'utilisateur.

$$d_{eucl}(X, Y) = (X_i - Y_i)^T (X_i - Y_i) \quad , i = 1..nb \text{ caract.}$$

Pour fixer le paramètre k de façon intelligente, nous avons découpé la base d'entraînement en deux. Un tiers de la base d'entraînement est maintenant la base d'évaluation de notre K-nn. Cela nous permet de tester plusieurs valeurs de k sans pour autant l'entraîner sur notre base de test. Notre démarche est la suivante :

- Calcul des distances pour chaque Y de la base d'évaluation avec l'ensemble des valeurs X de la base d'entraînement, soit une matrice de 2000 x 4000 distances.
- Pour chaque Y, classer les distances en ordre croissant et retenir les k plus petits.
- Donner à Y la classe la plus fréquente des k voisins les plus proches.

- Comparer tous les labels des Y obtenus avec les véritables labels pour en déduire l'erreur moyenne.

Cette méthode permet d'avoir le taux d'erreur pour un k fixé. Nous avons donc refait cette boucle pour des valeurs de k allant de 1 à 25. Pour la suite de cet algorithme, nous avons fixé k à sa valeur qui minimise le taux d'erreur sur la base d'évaluation.

Une fois que k est fixé, nous avons refait les instructions ci-dessus, mais avec la base de test cette fois-ci. On a obtenu une matrice de 1000 x 6000 distances. On a sélectionné les labels des k plus petites distances pour chaque individu de la base de test et attribué les plus fréquents pour chacun.

3.3. SVM

Le SVM, support machine vecteur en anglais, est notre troisième algorithme de classification. À la différence des deux premiers, cet algorithme a pour but de déterminer des frontières entre les classes et non de trouver leurs distributions. Ce type de classificateur est dit séparateur, en opposition de modélisation.

L'objectif d'un SVM est maximisé l'espace, la marge entre deux classes. Cela revient à trouver une solution optimale à l'équation suivante :

$$Max_{\alpha} \left[\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right] \quad avec \quad \sum_{i,j=1}^n \alpha_i y_i = 0 \quad et \quad 0 \leq \alpha_i \leq C, \quad i = 1..n$$

Pour notre application, nous avons des données qui ne sont pas linéairement séparables malgré l'ACP. Pour contourner ce problème, nous allons utiliser un noyau gaussien qui a la forme suivante :

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

Nous avons donc deux paramètres à fixer manuellement, γ et C. Le paramètre C va impacter le nombre de vecteurs support, plus il est faible et plus le nombre de vecteurs support est grand, et le paramètre gamma va influencer sur la variance des classes, plus il est faible et plus la variance est forte.

Comme pour le K-nn, nous avons séparé notre base d'entraînement en deux. Cela nous a permis de fixer C et gamma de façon à minimiser l'erreur obtenue avec la base d'évaluation. Nous avons gardé le ratio un tiers, deux tiers.

Dernier point, le SVM est un classificateur binaire. C'est-à-dire qu'il ne peut comparer que deux classes à la fois. Étant donné que nous avons un problème avec dix classes, nous avons dû utiliser une stratégie permettant d'avoir plus que deux classes. Cette stratégie est « un contre tous », c'est-à-dire qu'il faut un SVM par classe et que ce SVM va donner une probabilité d'appartenance par rapport à sa classe uniquement. Il faut donc, pour obtenir une prédiction, comparer la probabilité d'appartenance de chaque individu pour chaque classe et prendre celle qui a la valeur la plus forte.

4. Programmation

Nous avons organisé notre travail de programmation autour du script `Main_labo2.m`. C'est à partir de ce script qu'on va appeler différentes fonctions :

- `ReadDatabase.m`, permet de charger la base de données sélectionnée, rétine ou profiles.
- `Choose.m`, permet de choisir l'algorithme de classification, Bayes, Knn, Knn2 ou SVM.
- `Bayes.m`, réalise la classification selon l'algorithme de Bayes quadratique.
- `Knn.m`, réalise l'algorithme K-nn en faisant étape par étape.
- `Knn2.m`, réalise l'algorithme avec les fonctions Matlab `fitcknn` et `predict`.
- `SVM.m`, réalise l'algorithme SVM avec les fonctions Matlab `fitcsvm` et `predict`.
- `SVM2.m`, réalise une crossvalidation sur le SVM.
- `Resultats.m`, affiche la matrice de confusion et des exemples de mauvaise classification.

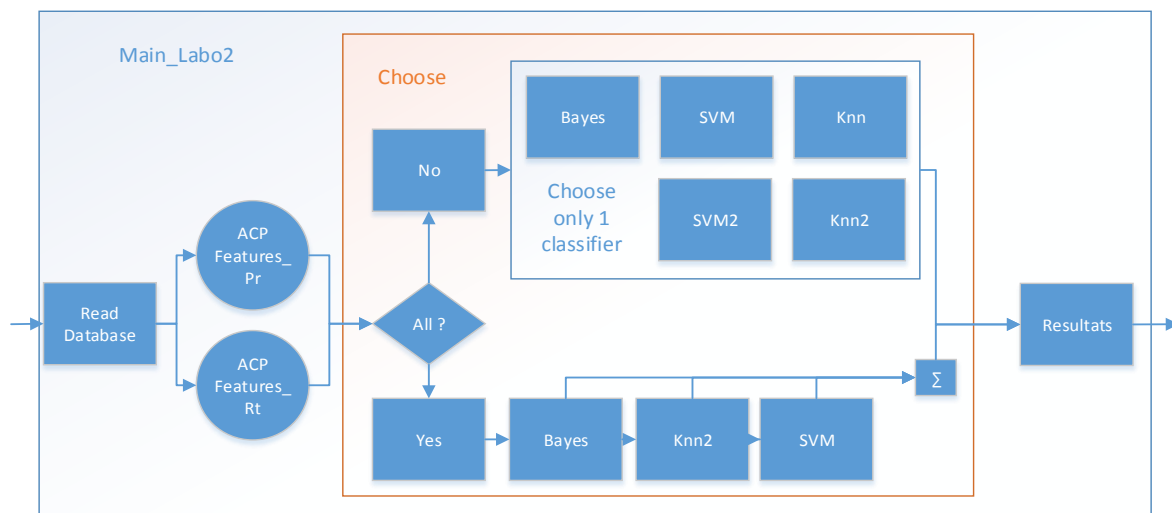


Figure 1 : Organisation des fonctions

L'avantage d'avoir réalisé le programme de cette façon est de pouvoir rapidement changer de base de données et d'algorithmes. Autre avantage, les quatre classificateurs ont la même sortie, c'est-à-dire qu'ils renvoient tous une matrice *result* de dimension 1000 x 2 avec dans la première colonne les vrais labels et dans la deuxième, les prédictions de l'algorithme. Grâce à ça, la fonction *Resultats* est la même pour tous. Il n'y a donc pas de facteurs extérieurs qui rentrent parasiter les résultats.

La fonction *Resultats* nécessite que vous ayez les images de test dans le même répertoire que là où vous lancé le code. Sinon il risque d'y avoir une erreur et de ne pas voir les images.

On peut voir ici comment nous avons fait pour essayer d'améliorer les performances de chaque classificateur. Pour cela, dans la fonction *choose*, on peut spécifier la variable `all='Yes'`. Dans ce cas, on va faire les trois classificateurs suivants, Bayes, Knn2 et SVM, à la suite. Les matrices *result* des trois algorithmes seront stocké dans *result1*, *result2* et *result3*. Par ligne, on prend la valeur la plus fréquente, voir ci-dessous. En cas d'égalité, la valeur la plus faible sera conservée. La section 5.5 traite plus en détail du fonctionnement de la fonction « All ». Attention, sur la figure ci-dessus, nous avons mis un sigma majuscule mais nous ne faisons pas la somme.

5. Résultats

La section suivante est consacrée aux résultats. Pour l'ensemble des trois méthodes, nous avons comparé la vitesse d'entraînement, la vitesse de test, le taux d'erreur et l'utilisation de la mémoire de l'ordinateur lors du calcul. Pour réaliser les mesures de temps, nous avons utilisé les fonctions tic et toc de Matlab. Pour connaître la mémoire utilisée, nous avons utilisé la fonction memory, plus précisément, la valeur renvoyée dans l'utilisation de la mémoire par Matlab (voir ci-dessous). Pour les algorithmes K-nn et SVM, nous avons des données supplémentaires, comme l'évolution du taux d'erreur en fonction de k, et de C et gamma. Enfin, notre code permet de renvoyer directement des exemples d'images mal classées. Nous les avons aussi affichés dans ce rapport.

```
Maximum possible array:    12230 MB (1.282e+10 bytes) *
Memory available for all arrays:    12230 MB (1.282e+10 bytes) *
Memory used by MATLAB:    1130 MB (1.185e+09 bytes)
Physical Memory (RAM):    8089 MB (8.482e+09 bytes)

* Limited by System Memory (physical + swap file) available.
```

Figure 2 : Capture d'écran sur ce que renvoie la fonction memory de Matlab

5.1. Bayes Quadratique

Comme nous l'avons dit plus haut, l'algorithme Bayes quadratique repose sur un seul calcul matriciel. Notre code est assez court. La première partie, appelé entraînement, permet de récupérer les valeurs de $\hat{\Sigma}_j$ et $\hat{\mu}_j$. La deuxième partie est justement l'équation présentée précédemment. On fait ce calcul sur le nombre de classes. On donne ensuite le label de la classe ayant la plus grande distance.

Ci-dessous, les matrices de confusions obtenues ainsi que des exemples d'images mal classifiées. Le reste des indicateurs se trouvent dans la section comparaison.

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	87	0	2	3	1	1	0	0	6	0
	1	0	92	2	0	3	0	1	0	2	0
	2	0	0	96	1	0	0	0	0	3	0
	3	0	0	2	92	0	3	0	0	3	0
	4	0	0	4	0	95	0	0	0	1	0
	5	0	0	1	18	0	79	0	0	2	0
	6	0	0	3	1	0	0	96	0	0	0
	7	0	0	7	1	1	0	0	84	4	3
	8	1	0	0	5	2	0	0	1	90	1
	9	0	0	1	0	2	0	0	11	3	83

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	98	0	1	0	0	0	0	0	1	0
	1	0	92	0	0	2	0	1	2	3	0
	2	0	0	97	0	0	0	0	0	3	0
	3	0	0	5	91	0	0	0	0	4	0
	4	1	0	3	0	95	0	0	0	1	0
	5	0	0	4	7	0	87	0	0	2	0
	6	1	0	0	0	0	0	98	0	1	0
	7	0	0	3	0	0	0	0	93	4	0
	8	0	0	0	6	1	0	0	0	93	0
	9	0	0	3	1	1	0	0	1	2	92

Figure 3 : matrice de confusion pour les méthodes Bayes (Profiles à gauche, Rétine à droite)

Image										
Réel	7	7	0	1	9	8	9	3	8	9
Prédiction	9	2	3	4	4	3	7	8	3	7

Image										
Réel	5	9	5	1	9	5	4	1	4	4
Prédiction	8	4	8	7	7	3	2	8	8	2

Figure 4 : Exemples de chiffres mal classifiés (Profiles en haut, Rétine en bas)

5.2. K-nn

Pour le code du K-nn, nous avons vraiment deux étapes, l'entraînement et le test. L'entraînement sert à fixer la valeur de k, le paramètre qui correspond au nombre de voisins pris en compte.

Pour cela, nous avons coupé notre base de données en 67% pour l'entraînement et 33% pour la validation. Nous avons calculé la distance pour chaque individu de validation sa distance euclidienne avec tous les individus d'entraînement et nous avons pris le label le plus fréquent sur les k plus proches voisins.

Nous avons pris k allant de 1 à 25. Nous avons testé k allant de 1 à 4000 (taille maximale). Le résultat est sans appel, plus k augmente, plus l'erreur augmente ainsi que le temps de calcul... Nous avons donc fixé k à 25 au maximum, sachant qu'il n'a jamais pris de valeur plus grande que 5 tout au long de nos tests. La sélection de k se fait en prenant celui qui minimise l'erreur sur la base de validation. Ci-dessous les courbes d'évolution de l'erreur en fonction de k. Le meilleur k est 1 pour les deux méthodes profile et rétine.

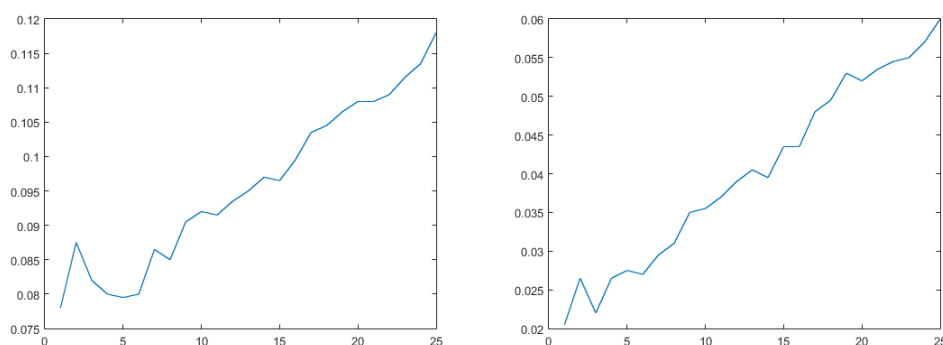


Figure 5 : Evolution de l'erreur en fonction de k avec la méthode Knn.m (Profiles à gauche, Rétine à droite)

La deuxième étape consiste à refaire le calcul mais cette fois avec le k obtenu et sur l'ensemble des données d'entraînement (on refusionne les matrices d'entraînement et de validation). On fait les distances des 1000 individus avec tous ceux de la base de test et on en prenant en compte les labels

des k plus proches voisins, on sélectionne le plus fréquent. En comparant avec les vrais labels, on obtient les matrices de confusions ci-dessous :

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	99	0	0	0	0	0	0	0	1	0
	1	0	100	0	0	0	0	0	0	0	0
	2	1	0	86	4	1	0	0	0	8	0
	3	5	1	0	90	0	0	1	1	2	0
	4	3	0	0	0	88	0	2	4	2	1
	5	0	0	1	8	0	84	2	0	5	0
	6	3	0	0	0	0	0	97	0	0	0
	7	0	0	2	2	0	0	0	82	4	10
	8	4	1	0	2	2	0	1	0	89	1
	9	1	0	2	0	1	0	0	8	0	88

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	98	0	1	0	0	0	1	0	0	0
	1	0	100	0	0	0	0	0	0	0	0
	2	2	0	92	0	0	0	0	1	5	0
	3	0	2	4	89	0	0	0	0	5	0
	4	1	2	1	0	95	0	0	0	0	1
	5	0	1	0	5	0	94	0	0	0	0
	6	0	0	0	0	0	0	100	0	0	0
	7	0	0	0	0	1	0	0	97	0	2
	8	0	0	0	6	2	0	0	0	90	2
	9	0	0	0	0	1	0	0	3	0	96

Figure 6 : Matrice de confusion pour la méthode Knn.m (Profils à gauche, Rétine à droite)

Ci-dessous des exemples d'images mal classifiées. On peut remarquer que, comme pour le Bayes, les 7 et 9 sont souvent confondus ainsi que le triplet 3,5 et 8. On peut remarquer aussi que le premier et troisième chiffre de la deuxième ligne sont mal segmentés

Image										
Réal	9	9	7	3	5	7	2	2	3	3
Prédiction	7	7	9	0	3	8	8	8	6	7

Image										
Réal	8	4	7	8	9	3	4	8	3	3
Prédiction	4	9	4	3	7	8	0	9	1	1

Figure 7 : Exemples de chiffres mal classifiés avec la méthode Knn.m (Profils en haut, Rétine en bas)

L'algorithme est assez lent. Il faut calculer 4000×2000 plus 6000×1000 distances, soit 14 millions de distances. Nous trouver la fonction Matlab *fitcknn* qui permet de refaire exactement la méthode ci-dessus. Nous l'avons comparé à notre premier code, on obtient sensiblement les mêmes taux d'erreur pour 30 fois moins de temps (voir la section comparaison).

Comme pour le premier K-nn, nous avons l'évolution de l'erreur en fonction de k. encore une fois, le meilleur résultat est obtenu pour $k=1$ dans les deux cas. À noter que k est fixé en fonction de la base d'entraînement et la base d'évaluation et que la séparation un tiers, deux tiers est faite au hasard. Il se peut que si l'on relance l'algorithme, on obtienne un k différent et donc une précision différente.

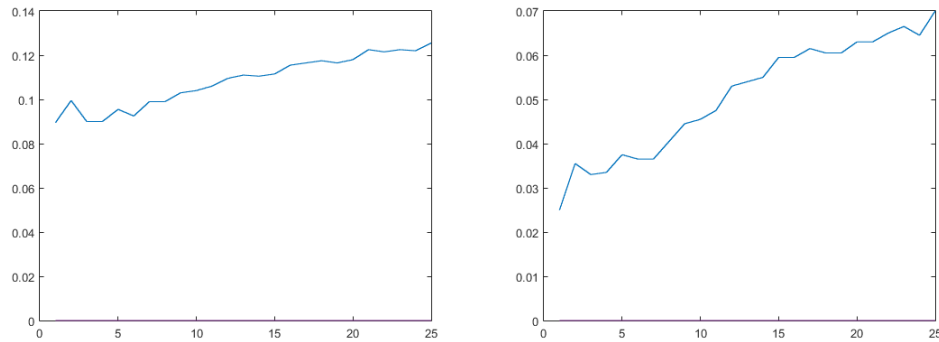


Figure 8 : Évolution de l'erreur en fonction de k avec la méthode Knn2.m
(Profils à gauche, Rétine à droite)

Comme pour le premier K-nn et Bayes, on a les matrices de confusions et des exemples d'images mal classifiées. On retrouve encore le même pattern, 7 et 9 sont souvent confondus surtout avec la méthode profile. Alors que la méthode rétine va plus confondre les 3,5 et 8.

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	99	0	0	0	0	0	0	0	1	0
	1	0	100	0	0	0	0	0	0	0	0
	2	1	0	86	4	1	0	0	0	8	0
	3	5	1	0	90	0	0	1	1	2	0
	4	3	0	0	0	88	0	2	4	2	1
	5	0	0	1	8	0	84	2	0	5	0
	6	3	0	0	0	0	0	97	0	0	0
	7	0	0	2	2	0	0	0	82	4	10
	8	4	1	0	2	2	0	1	0	89	1
	9	1	0	2	0	1	0	0	8	0	88

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	98	0	1	0	0	0	1	0	0	0
	1	0	100	0	0	0	0	0	0	0	0
	2	2	0	92	0	0	0	0	1	5	0
	3	0	2	4	89	0	0	0	0	5	0
	4	1	2	1	0	95	0	0	0	0	1
	5	0	1	0	5	0	94	0	0	0	0
	6	0	0	0	0	0	0	100	0	0	0
	7	0	0	0	0	1	0	0	97	0	2
	8	0	0	0	6	2	0	0	0	90	2
	9	0	0	0	0	1	0	0	3	0	96

Figure 9 : Matrice de confusion pour la méthode Knn2.m (Profils à gauche, Rétine à droite)

On remarque que le chiffre sept ici est fait avec une barre descendante à gauche sur la première ligne. À cause de cela, il est mal classifié. De la même façon, les 8 non fermés en haut sont souvent considérés comme des 4.

Image										
Réal	5	5	9	9	3	7	8	3	9	7
Prédiction	3	8	7	7	0	9	4	0	7	9








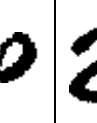
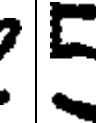

Image										
Réel	2	3	9	8	2	0	7	0	2	5
Prédiction	8	2	7	3	8	6	9	2	8	3

Figure 10 : Exemples de chiffres mal classifiés avec le Knn2.m (Profils en haut, Rétine en bas)

5.3. SVM

Comme pour le K-nn, nous avons coupé la base d'entraînement en deux (67% entraînement et 33% évaluation). Cela nous permet de faire varier les deux variables du SVM, à savoir C et gamma. Nous avons utilisé les propriétés *BoxConstraint* et *KernelScale* pour respectivement le C et le gamma. On a fait varier C et gamma entre 1 et 1 000 000 (en multipliant par 10). Nous avons testé avec des valeurs inférieures à 1 (0.001, 0.01 et 0.1) pour C et gamma. Les erreurs obtenues avec ces valeurs étaient d'environ 80-90%. Nous les avons enlevées pour gagner un peu de vitesse et des résultats cohérents. Ci-dessous l'erreur obtenue en fonction de C et gamma pour profils et rétine.

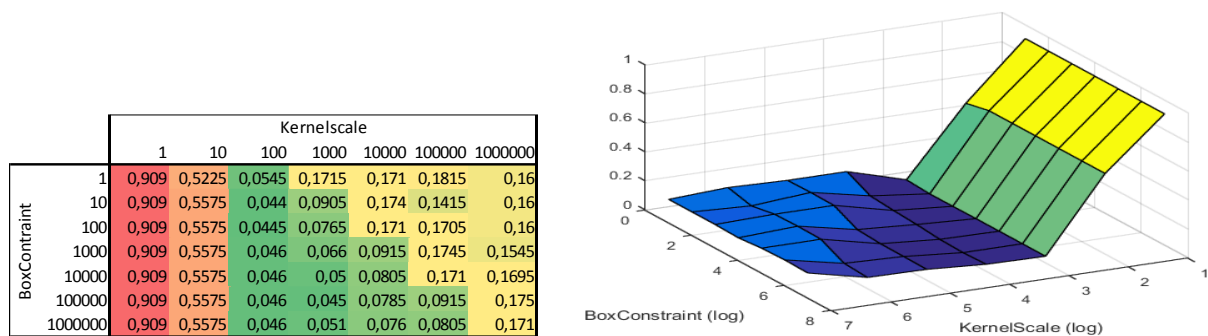


Figure 11 : Évolution de l'erreur en fonction de BoxConstraint et KernelScale pour le SVM.m (Profil)

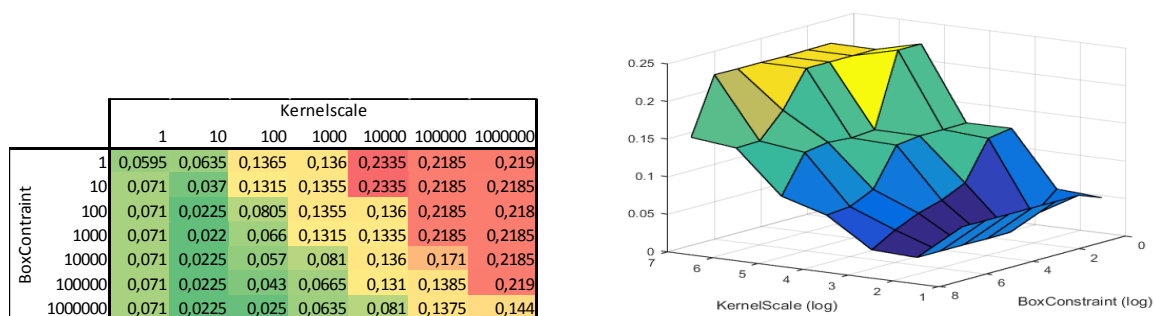


Figure 12 : Évolution de l'erreur en fonction de BoxConstraint et KernelScale pour le SVM.m (Rétine)

Les meilleurs résultats sont obtenus pour C = 10 et gamma = 100 pour la méthode profil et C = 1000 et gamma = 10 pour la méthode rétine. Ces valeurs sont conservées pour la partie test de l'algorithme. Comme pour le K-nn, on reprend l'ensemble des données d'entraînement et on teste sur des données jamais encore vues (pour éviter toute sorte de surapprentissage). De même, nous n'avons pas voulu chercher les valeurs de C et gamma avec précision pour éviter le surapprentissage.

		Prediction									
		0	1	2	3	4	5	6	7	8	9
R��l	0	99	0	0	0	0	0	0	0	1	0
	1	0	99	0	0	1	0	0	0	0	0
	2	2	0	82	2	2	0	2	0	10	0
	3	1	3	3	89	0	0	0	0	2	2
	4	3	1	0	0	88	0	4	0	1	3
	5	0	1	0	3	0	90	4	0	2	0
	6	6	0	0	0	0	0	94	0	0	0
	7	0	1	1	1	1	0	0	88	1	7
	8	3	0	1	7	2	0	2	0	83	2
	9	1	0	1	0	2	2	0	4	1	89

		Prediction									
		0	1	2	3	4	5	6	7	8	9
R��l	0	97	0	2	0	0	0	0	0	1	0
	1	0	96	0	1	0	1	0	1	1	0
	2	0	0	83	0	5	1	2	0	9	0
	3	0	1	4	91	0	0	0	0	4	0
	4	1	1	0	0	88	2	6	0	1	1
	5	1	0	0	7	1	82	4	2	1	2
	6	0	0	1	0	0	0	98	0	1	0
	7	0	0	0	1	1	0	0	91	7	0
	8	2	1	1	5	1	0	1	0	87	2
	9	0	0	1	0	2	1	0	0	1	95

Figure 13 : Matrice de confusion pour la m  thode SVM.m (Profiles    gauche, R  tine    droite)

Ci-dessus, les matrices de confusions obtenues et ci-dessous, des exemples de mauvaises classifications. On retrouve les 8 ouverts en haut. Les images bruit  es ou parasit  es comme le deuxi  me et septi  me chiffre de la deuxi  me ligne. Aussi, le chiffre 4 de la deuxi  me ligne est ferm  . C'est une autre fa  on d'  crire les 4 mais cela pose probl  me pour le reconnaitre.

Image										
R��l	8	5	5	3	8	6	0	7	8	9
Pr��diction	4	6	6	8	3	0	8	3	4	7

Image										
R��l	5	1	4	0	4	5	5	8	4	6
Pr��diction	0	7	5	8	6	6	8	3	5	2

Figure 14 : Exemples de chiffres mal classifi  s avec le SVM.m (Profiles en haut, R  tine en bas)

5.4. SVM2

Le majeur changement entre SVM.m et SVM2.m est que nous avons rajout   la crossvalidation au second. Nous n'avons pas refait de double boucle pour trouver C et gamma optimaux du fait de la lenteur du calcul (>2h). Nous avons donc choisi de reprendre les valeurs optimales obtenues dans la premi  re fonction SVM.m et de les hardcoder dans SVM2.m.

Les r  sultats obtenus sont affich  s dans la section comparaison. Nous n'avons pas continu   la recherche dans cet axe-l   puisque SVM2 nous donne des r  sultats moins bons (taux d'erreur un peu plus   lev  ) pour de temps de calcul plus longs. Exemple, pour faire la phase de test avec C et gamma fix  , la premi  re fonction SVM met entre 3 et 17 secondes ; alors que la deuxi  me fonction, SVM2, met entre 30 et 155 secondes.

5.5. « All »

La fonction « All » va essayer de tirer le meilleur de chaque classificateur. Pour cela, elle va faire rouler les fonctions Bayes, Knn2 et SVM l'un après l'autre, indépendamment. La matrice *result* issue de Bayes est stockée dans *result1*, celle de Knn2 dans *result2* et celle du SVM dans *result3*. Nous avons donc trois matrices 1000 x 2.

On isole la deuxième colonne de chaque matrice *result(i)*, celle qui contient les prédictions du classificateur correspondant. Ensuite, nous avons utilisé la fonction *mode* de Matlab pour chacune des lignes pour les deuxième colonnes des matrices *result(i)*. Cette fonction permet de renvoyer la valeur la plus fréquente.

Sur l'exemple ci-dessous, nous avons représenté tous les cas de figure possibles avec la fonction *mode* pour 3 valeurs :

- Soit les trois valeurs sont identiques, on prend cette valeur.
- Soit deux valeurs identiques et une quelconque, on prend la valeur présente deux fois.
- Soit on a trois valeurs différentes, dans ce cas-là, la fonction renvoie la valeur la plus faible.

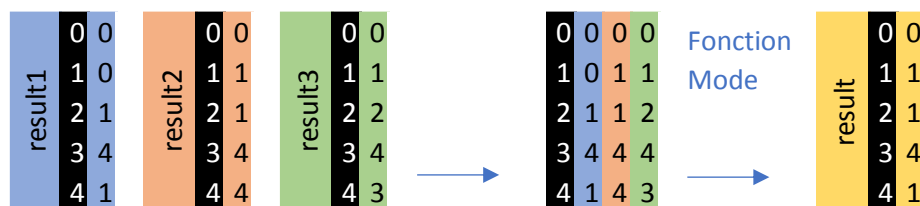


Figure 15 : Fonctionnement de la fonction « All »

Remarque, comme la fonction lance les trois classificateurs de façon séquentielle, on a un temps de calcul égal à la somme des temps de calcul de chaque classificateur.

Au niveau du taux d'erreur obtenu, nous avons de très bons résultats puisque le taux d'erreur est inférieur à 1% pour les deux méthodes, profile et rétine. Nous avons dû même modifier la fonction *Resultats* car il n'y a pas assez d'erreurs pour en afficher 10. Nous en avons 9 sur 1000 dans les deux cas, soit 0,9% d'erreur. Ci-dessous, les matrices de confusion qui confirment nos résultats.

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	100	0	0	0	0	0	0	0	0	0
	1	0	100	0	0	0	0	0	0	0	0
	2	0	0	100	0	0	0	0	0	0	0
	3	0	0	0	100	0	0	0	0	0	0
	4	0	0	0	0	100	0	0	0	0	0
	5	0	0	0	2	0	98	0	0	0	0
	6	0	0	0	0	0	0	100	0	0	0
	7	0	0	0	0	1	0	0	99	0	0
	8	1	0	0	0	2	0	0	0	97	0
	9	0	0	1	0	1	0	0	1	0	97

		Prediction									
		0	1	2	3	4	5	6	7	8	9
Réal	0	100	0	0	0	0	0	0	0	0	0
	1	0	100	0	0	0	0	0	0	0	0
	2	0	0	100	0	0	0	0	0	0	0
	3	0	0	2	98	0	0	0	0	0	0
	4	0	0	0	0	100	0	0	0	0	0
	5	0	0	0	2	0	98	0	0	0	0
	6	0	0	0	0	0	0	100	0	0	0
	7	0	0	0	0	0	0	0	100	0	0
	8	0	0	0	5	0	0	0	0	95	0
	9	0	0	0	0	0	0	0	0	0	100

Figure 16 : Matrice de confusion pour la méthode « All » (Profiles à gauche, Rétine à droite)

Enfin, toutes les erreurs de la fonction « All » pour profile et rétine ci-dessous. Pour la méthode profile, on des erreurs dues aux chiffres non fermés, comme les 8 et le 9. On aussi des problèmes liés au pré-processing et justement à la segmentation des chiffres comme le 7 et le premier 8. Pour la méthode rétine, les erreurs n'arrivent quasiment que sur notre triplet 3,5 et 8.









Image									
Réel	7	8	9	9	5	8	9	8	5
Prédiction	4	4	7	4	3	4	2	0	3










Image									
Réel	5	8	3	8	5	8	3	8	8
Prédiction	3	3	2	3	3	3	2	3	3

Figure 17 : Exemples de chiffres mal classifiés par la méthode « All » (Profiles en haut, Rétine en bas)

5.6. Comparaison

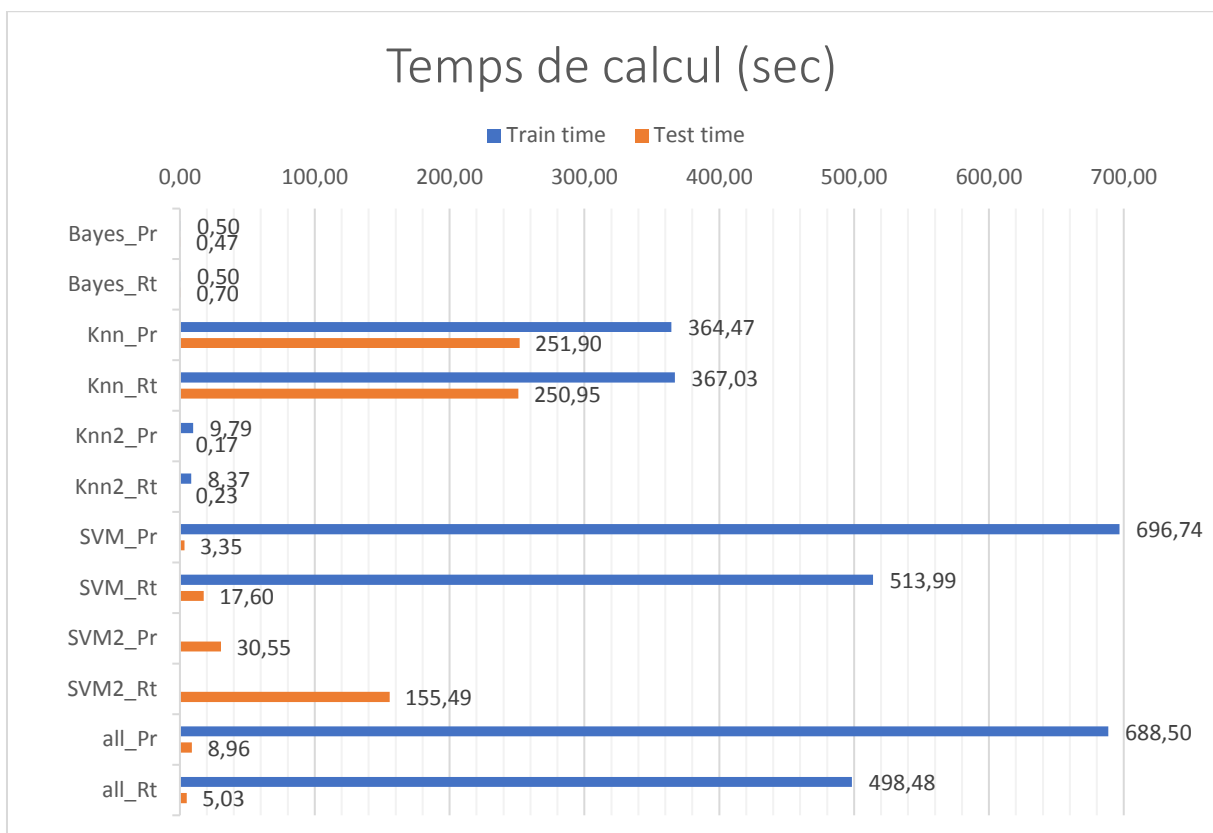


Figure 18 : Comparaison des temps de calcul

Le graphique ci-dessus est le résumé de tous les temps d'entraînement (en bleu) et de test (en orange) pour chaque classificateur pour les deux méthodes (profile et rétine).

Tout d'abord, on voit que le classificateur le plus rapide est Bayes quadratique. Cela peut s'expliquer dû fait qu'il n'y a qu'un seul calcul pour chaque classe, pour chaque individu testé, soit seulement 10 000 exécutions. À l'opposé, le premier K-nn doit faire 14 000 000 de calcul de distance et il y a de nombreuses boucles imbriquées dans ce code, d'où son temps très long.

En parlant du K-nn, on voit très clairement la différence de temps entre le K-nn codé par nous-mêmes et celui utilisant le code de Matlab.

Le SVM est très long à entrainer, mais est ensuite très rapide pour les temps de tests. On peut aussi remarquer le temps de test du SVM2 bien supérieur au premier SVM qui est sans la crossvalidation.

Enfin, comme nous l'avons dit, la fonction « All » est la somme des temps des fonctions Bayes, Knn2 et SVM, pour l'entraînement et pour le test.

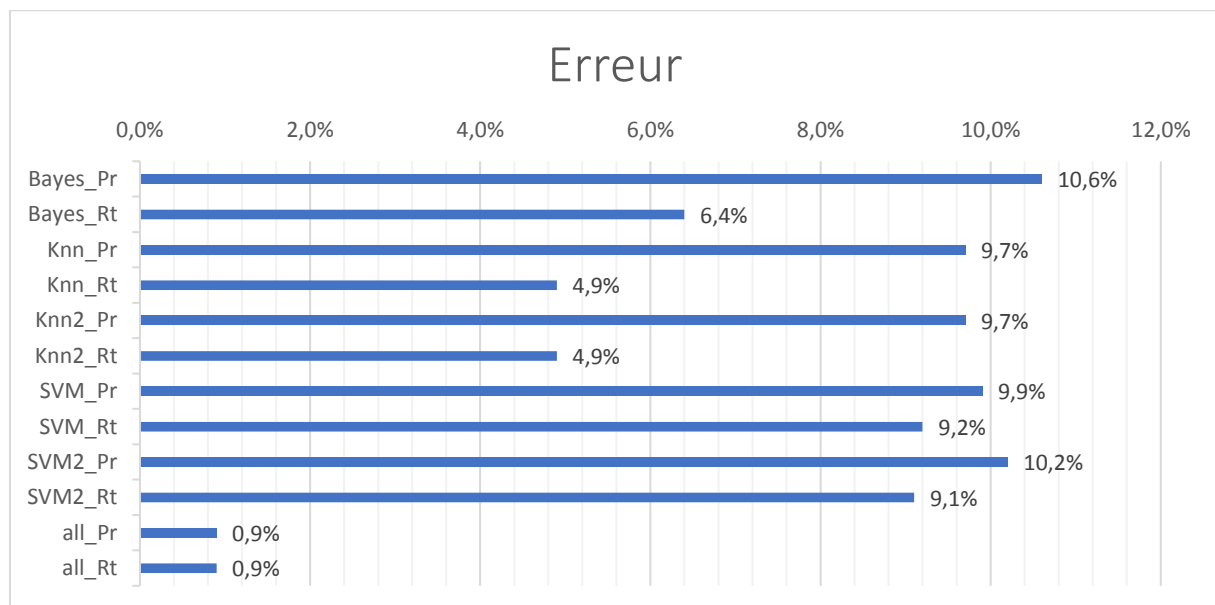


Figure 19 : Comparaison du taux d'erreur

Ci-dessus, le graphique représentant le taux d'erreur de nos classificateurs. L'erreur est calculée de la façon suivante : $a = \text{mean}(\text{result}(:,1) \sim \text{result}(:,2))$; ce qui revient à l'équation suivante :

$$e = \frac{\sum_{i=1}^{Nb \text{ ind. Test}} \text{predictlabel}(i) - \text{truelabel}(i)}{Nb \text{ ind. Test}}$$

Les résultats obtenus montrent qu'en général, la méthode rétine donne de meilleurs résultats que la méthode profile et cela peut-importe le classificateur.

Le résultat le plus remarquable ici est que la fonction « All » est nettement plus performante que tous les autres classificateurs, entre 5 à 10 fois mieux.

On peut remarquer aussi que le taux d'erreur des deux K-nn est légèrement plus faible que les autres classificateurs (hormis le « All ») et qu'ils donnent les mêmes résultats, ce qui est un bon point.

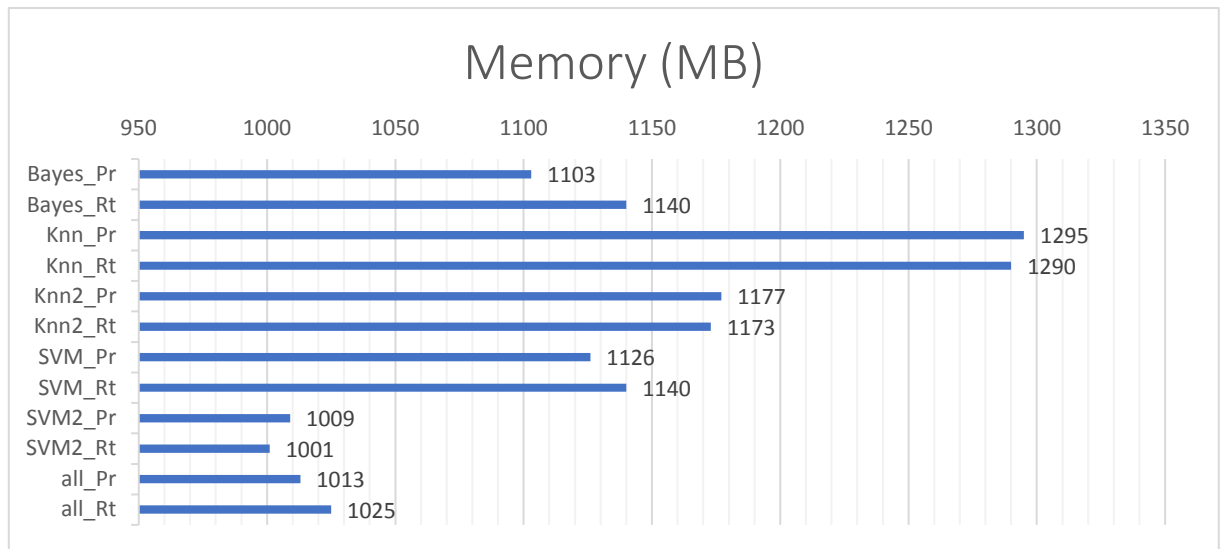


Figure 20 : Comparaison de l'utilisation mémoire

Le facteur d'utilisation mémoire est, selon nous, un aspect non important pour notre application actuelle. La capacité mémoire utilisée est sensiblement la même pour tous les algorithmes, entre 1000 Mb et 1300 Mb. Cette différence est négligeable sur les ordinateurs actuels ayant en moyenne plus de 8000 Mb de mémoire vive disponible.

6. Conclusion

Pour revenir sur le travail effectué, nous pouvons dire que nous avons réalisé trois catégories de classificateurs (Bayes, K-nn et SVM). Nous avons aussi essayé d'explorer au maximum chacun de ses trois algorithmes pour essayer de les optimiser comme c'est le cas avec la fonction Knn2 qui est plus rapide de 6 fois que la fonction que nous avons codée personnellement. Nous avons aussi pu voir que certaines optimisations n'étaient pas appliquées pour notre problème, comme le crossvalidation du SVM qui augmente énormément les temps de calcul pour des résultats similaires.

Nous avons pu aussi mettre en place l'algorithme « All » qui reprend tout notre travail et nous donne de très bons résultats en termes d'erreur obtenus.

Nous avons listé ici quelques améliorations à ce travail pour le futur : premièrement, étendre notre base de données à l'ensemble des données du MNIST pour gommer les erreurs de segmentation d'image. Nous pouvons aussi coder de nouveaux classificateurs pour encore faire descendre le taux d'erreur. Les classificateurs de type K-means sont une catégorie que nous n'avons pas utilisée (non supervisé). Il serait intéressant de les rajouter. De plus, nous rajouter ce K-means à la fonction « All ». Nous pouvons aussi revoir la façon dont sont extraites les données. Nous avons vu qu'avec la méthode rétine, on était toujours plus performant jusqu'à presque 50% pour le K-nn. Cela pourrait permettre de mieux gérer les chiffres bruités, pixel n'appartenant pas à la forme, et de mieux prendre en compte les défauts d'écritures, comme les 8 maux fermés. On pourrait aussi rajouter des 1 et des 7 écrits à l'« européenne » et voir comment nos classificateurs performant face à ces ressemblances.

Enfin, nous tenons à dire que nous avons vraiment apprécié faire ce projet. Il nous a vraiment permis de comprendre le fonctionnement interne des classificateurs à travers un projet intéressant.