Projet :
Reconnaissance de chiffres manuscrits

Groupe : AMIRAULT TONY <> PHAM QUOC DAT

Enseignant : Claude Barras

Matière : Apprentissage, optimisation, fouille de données

I - Introduction

II – Prétraitement des données via ACP

III – Apprentissage avec simplification

IV – Classification bayésienne

V – Performances

I – Introduction

I.1 - But du projet

Le but de ce projet était d'implémenter puis d'évaluer les performances d'un système de reconnaissance de chiffres manuscrits basé sur un « Classifieur Bayésien ».

I.2 – Prérequis

Pour réaliser ce projet plusieurs prérequis étaient nécessaires :

Tout d'abord l'utilisation du langage « Python » nous a été demandée, il nous a donc fallut mettre en place un environnement permettant l'utilisation du langage « Python ».

Ensuite il nous a été mis à disposition un ensemble d'images représentant des chiffres manuscrits de 0 à 9 pour l'élaboration de notre projet. On y retrouve :

- Des images d'apprentissage : soit 10 000 images de taille 28*28
- Des images de tests : soit 5000 images de taille 28*28

Afin d'effectuer la correspondance de chaque image par rapport à ce qu'elle représente réellement, deux fichiers texte nous ont été fournis :

- Un fichier « train.txt » : contenant la correspondance entre le nom des images d'apprentissages et leur classe (le nombre qu'elle représente).
- Un fichier « test.txt » : contenant la correspondance entre le nom des images de tests et leur classe (le nombre qu'elle représente).

Une fois l'environnement mis en place et les fichiers acquis nous avons pu commencer à réfléchir aux différentes étapes.

I.3 – Etapes du projet

Les étapes suivies pour la réalisation du projet sont :

1 - Prétraitement des données afin de limiter le nombre de dimension à analyser.

2- Apprentissage des Gaussiennes sur des données d'entraînement afin de déterminer la probabilité d'appartenance à une classe.

3- Classification des données dans la classe ayant la plus forte probabilité de présence.

II – Prétraitement des données via ACP

L'objectif de ce module est d'obtenir après lecture de l'ensemble des images d'apprentissage une matrice dite de projection qui permettra par la suite de réduire le nombre de dimension des données utilisées et de garder les composantes les plus représentatives.

Nous avons suivi la méthode d'analyse des composantes (ACP) principales afin de réaliser ce module. Cette méthode consistait à créer une matrice contenant chaque image d'apprentissage afin d'ensuite obtenir les valeurs et vecteurs propres après avoir effectué la covariance de la matrice.

Explication détaillée :

Nous avons créé une matrice « MatriceVecteursImg » qui contenait les données de chaque image d'apprentissage de la façon suivante :

Lignes/Colonnes	1	2	3	…	d
1	Données image d'apprentissage numéro 1				
2	Données image d'apprentissage numéro 2				
NbImage	Données image d'apprentissage numéro i				

« NbImage » étant le nombre d'image d'apprentissage.
« d » étant la taille des données d'une image d'apprentissage sous forme de vecteur ligne, soit 28*28 = 748.

- Le code suivant nous a permis d'obtenir les valeurs propres et les vecteurs propres :

Code :

```
MatriceCov = ns.cov(MatriceVecteursImg.T) # calcul la matrice de covariance
valP,vectP = ns.linalg.eigh(MatriceCov) #récupération valeurs propres et vecteurs propres
```

- Nous avons ensuite trié ces valeurs afin d'avoir les vecteurs propres les plus significatifs en tête de liste dans le but que par la suite on ne récupère que les vecteurs propres les plus grands (avant la projection) afin d'obtenir les données les plus significatives (après la projection).

Code :

```
DicoValVectP= zip(listValP,listVectP ) #association valeurs propres et vecteurs propres
DicoValVectP.sort(reverse=True) #Tri dans l'ordre decroissant
vectP =[x[1] for x in DicoValVectP ]#récupère les vecteurs propre trier dans le bon ordre
```

Une fois la récupération des vecteurs propres triés le module de prétraitement est terminé, il est maintenant possible d'effectuer le module d'apprentissage.

III – Apprentissage avec simplification

L'objectif de ce module est d'obtenir une distribution de probabilité des observations pour chaque classe (0, 1, 2…9) afin de pouvoir par la suite choisir la classe qui minimise le risque d'erreur.

Pour faire cela il faut déterminer une matrice représentant contenant l'ensemble des données des images d'une classe donnée et cela pour chaque classe pour ensuite déterminer un vecteur moyen et une matrice de covariance pour chaque classe.

Explication détaillée :

A) Simplification

Dans notre cas, nous avons effectué une projection de chaque image (avant son apprentissage) dans la dimension choisie au lancement de l'apprentissage afin de réduire la dimension des données et de se limiter aux composantes les plus représentatives.

1 - Nous réduisons tout d'abord l'ensemble des vecteurs propres (déterminé dans le module prétraitement) à la dimension choisie « nb Dim » :

Code :

```
vectP = vectP[0:nbDim,:]### réduction du nombre de composante pour la projection
```

2 - Nous projetons l'ensemble des données d'une image dans la nouvelle dimension, et nous effectuons cela pour chaque image.

La formule utilisée est la suivante :

Y = P * X ; Avec P l'ensemble des vecteurs propres réduit et X les données de l'image.

On a donc le code suivant (pour une image) :

Code :

```
donneeImgModif = ns.dot(vectP,donneeImg) # projection des données dans la nouvelle dimension
```

B) Apprentissage

Nous avons créé une matrice qui contenait les données des images simplifiées pour chaque classe.

Lignes/Colonnes	1	2	3	…	nbDim
1	Données image d'apprentissage numéro 1				
2	Données image d'apprentissage numéro 2				
NbImageClasse	Données image d'apprentissage numéro i				

« NbImageClasse» étant le nombre d'images d'apprentissage correspondant à une classe donnée.
« nbDim » étant la taille des données d'une image d'apprentissage projetée dans la nouvelle dimension soit « nbDim »

- Le code suivant nous a permis d'obtenir une matrice de covariance et un vecteur moyen pour chaque classe:

Code :

```
Moyenne.append(MatriceVecteursImg.mean(0))# Moyenne ajouter dans la liste
MatriceCov.append(ns.cov(MatriceVecteursImg.T))#Covariance ajouter dans la liste
```

Ainsi avons-nous créer deux listes contenant un vecteur moyen et une matrice de covariance pour chaque classe (0, 1, 2, … 9).

Le module d'apprentissage terminé on dispose donc de toutes les données nécessaires pour effectuer une classification sur celle-ci.

IV - Classification Bayésienne

L'objectif de ce dernier module est de classifier différentes images de tests et de connaître le taux d'erreur global de notre classifieur.

Pour cela il faut déterminer la probabilité d'appartenance de chaque image de test pour chaque classe à l'aide de la loi normale et ensuite déterminer la classe ayant la probabilité maximale.

Explication détaillée :

Pour implémenter ce module nous avons donc utilisé les différentes données résultantes des modules précédents afin d'appliquer la formule de la loi normal, soit :

$$g_i(\mathbf{x}) = \log P(\omega_i) - \frac{1}{2}\log|\Sigma_i| - \frac{1}{2}(\mathbf{x}-\mu_i)^t \Sigma_i^{-1}(\mathbf{x}-\mu_i)$$

Avec :

- X : données de l'image de test que l'on souhaite classer
- μi : le vecteur moyen correspondant à la classe (déterminer dans l'apprentissage)
- Σi : la matrice de covariance correspondant à la classe (déterminer dans l'apprentissage)

Les classes sont équiprobables alors « LogP(wi) » est inutile dans notre cas. De plus nous utilisons cette équation avec les données de test simplifiées grâce à l'ACP effectuée dans le module de prétraitement des données.

Une fois la probabilité pour chaque classe obtenue nous avons déterminé la classe correspondant en prenant la classe ayant la probabilité maximale pour une image donnée. Soit le code suivant :

Code :

```
numClasse = resultLoiNormal.argmax()# récupération de la classe avec la plus haute probabilité
```

V – Performances

L'objectif de cette partie est d'analyser les résultats obtenus par le Classifieur Bayésien implémenté afin d'évaluer ses performances.

Pour effectuer l'analyse il nous a fallu comptabiliser l'erreur globale du système mais aussi établir une matrice de confusion.

A) Outil d'analyse

Calcul de l'erreur global :

Pour calculer l'erreur globale du système nous avons utilisé le fichier « test.txt » afin de connaître les classes réelles et nous avons comparé ces classes par celles obtenues par notre classifieur. Ensuite nous avons appliqué la formule suivante :

Taux Erreur Global = (nombre d'image mal classé / nombre d'image testé) * 100

Soit le code :

Code :

```
tauxErreurGlobal = (nbImageMalClasse/nbImgTest) * 100;#Calcul du taux d'erreur global
```

Matrice de confusion :

La matrice de confusion a été établie de la façon suivante :

Chaque colonne de la matrice représente le nombre d'occurrences d'une classe estimée, tandis que chaque ligne représente le nombre d'occurrences d'une classe réelle.

B) Analyse

1 – Test avec toutes les données et une dimension au choix

Nombre d'images d'apprentissage : 10 000
Nombre d'images de test : 5 000
Dimension choisie : 15

Au lancement du programme on choisit la partie 4 qui exécutera les fonctions suivantes successivement :

1 -> vectP,MatriceData = ModulePretraitementACP(10 000)
2 -> Moyenne,MatriceCov = ModuleApprentissageAvecAcp(10 000, 15,vectP,MatriceData)
3 -> ModuleRecoImg(5 000, 15,vectP,Moyenne,MatriceCov)

Matrice de confusion :

	Classe estimée									
Classe réelle		0	1	2	3	4	5	6	7	8	9
	0	448.	0	2	0	0	6	2	0	2	0
	1	0	543	6	13	2	2	0	0	5	0
	2	8	0	497	4	3	0	6	4	8	0
	3	1	0	3	458	1	14	0	9	10	4
	4	1	0	4	0	455	0	3	3	7	27
	5	2	0	2	22	1	403	0	4	14	8
	6	19	1	3	0	5	13	409	0	12	0
	7	1	2	29	8	6	0	0	421	9	36
	8	8	0	8	26	5	5	1	3	416	17
	9	2	1	6	15	28	3	1	6	14	444

Taux d'erreur par classe (en %):

0	1	2	3	4	5	6	7	8	9
2.6	4.9	6.2	8.4	9.0	11.6	11.47	17.7	14.9	14.6

On peut observer ici que la classe 7 est celle avec le plus grand taux d'erreur, tandis que la classe 0 est celle avec le plus petit taux d'erreur.

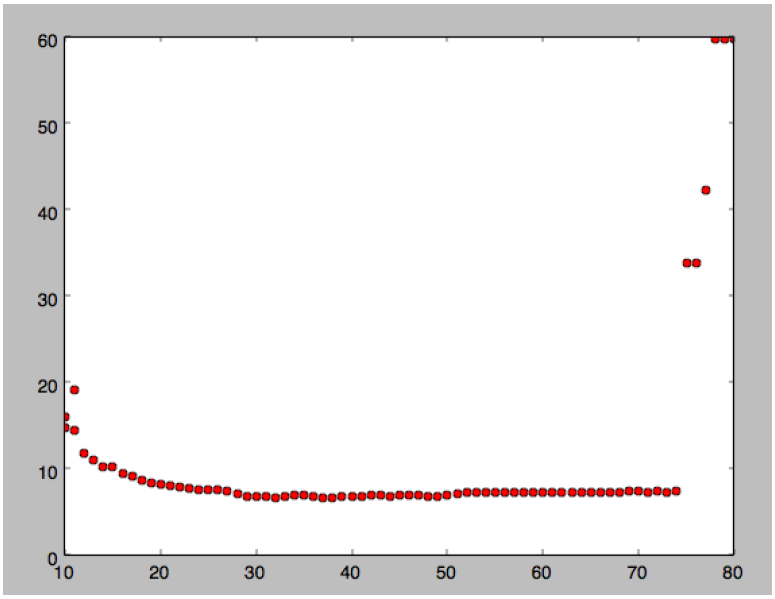
Taux d'erreur global (en %) : 10.12%

2 - Recherche de la meilleure dimension

Nombre d'image d'apprentissage : 10 000
Nombre d'image de test : 5 000
Nombre de dimension : ?

Au lancement du programme on choisit la partie 5 qui cherchera la meilleure dimension dans l'intervalle souhaité.

Le programme nous affiche la courbe du Taux d'erreur en fonction du nombre des dimensions soit :



On obtient 37 comme meilleure dimension pour la projection.

On remarquera que si le nombre de dimension est trop petit alors on perd des informations, ce qui augmente le taux d'erreur globale et que si le nombre de dimension est trop grand alors on peut penser que les données obtenues ne sont plus représentatives et donc cela augmente aussi le taux d'erreur globale.

3 – Test avec toutes les données et la meilleure dimension

Nombre d'images d'apprentissage : 10 000
Nombre d'images de test : 5 000
Dimension choisie : 37

Matrice de confusion :

	Classe estimée									
Classe réelle		0	1	2	3	4	5	6	7	8	9
	0	455	0	1	0	0	1	0	0	3	0
	1	0	538	9	4	0	0	0	0	20	0
	2	2	0	514	1	1	0	2	0	10	0
	3	0	0	6	478	0	6	0	1	7	2
	4	0	0	5	0	484	0	3	0	3	5
	5	1	0	0	14	1	429	0	0	12	0
	6	7	1	1	1	3	10	424	0	15	0
	7	0	1	39	4	2	2	0	436	12	16
	8	4	0	10	14	0	2	0	3	454	2
	9	3	0	4	9	18	4	0	4	17	461

Taux d'erreur par classe (en %):

0	1	2	3	4	5	6	7	8	9
1	5,77	3	4,4	3,2	5,9	8,22	14,8	7,15	11,34

Comme dans l'analyse précédente on observe que la classe 7 est celle avec le plus grand taux d'erreur et que la classe 0 est toujours celle avec le plus petit taux d'erreur.

Taux d'erreur global (en %) : 6.54%

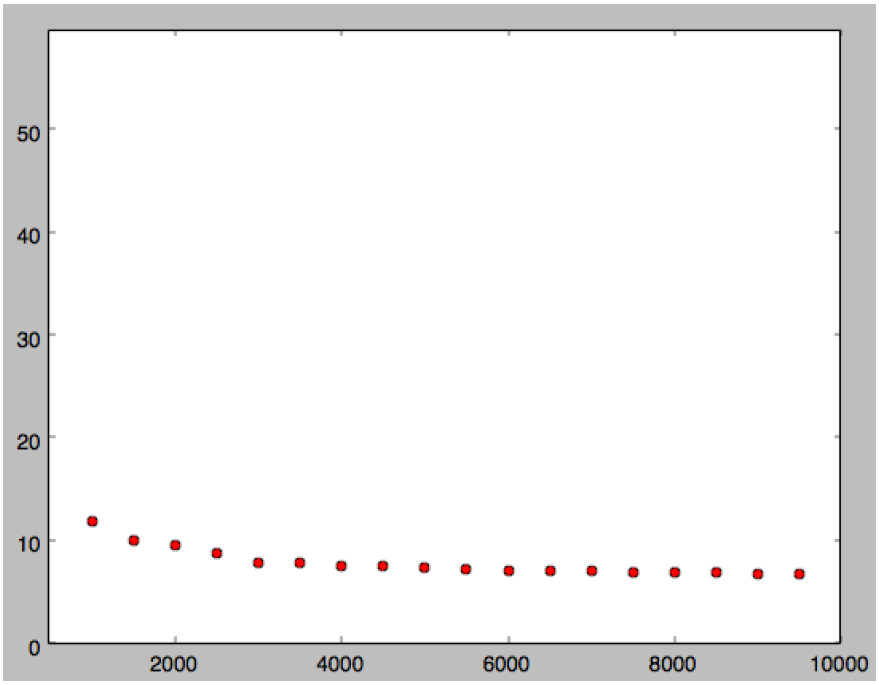
On peut observer ici l'intérêt de l'ACP, en effet cela nous permet de réduire la dimension de nos données d'apprentissage mais aussi d'obtenir un taux d'erreur plus faible même en réduisant la taille de nos données. Dans le cas de la dimension optimale ici 37 on peut remarquer que par rapport à notre test précédent on obtient presque un taux d'erreur divisé par deux.

4 - Recherche du meilleur nombre d'images d'apprentissage

Nombre d'images d'apprentissage : ?
Nombre d'images de test : 5 000
Nombre de dimension : 37

Au lancement du programme on choisit la partie 6 qui cherchera le meilleur nombre d'image d'apprentissage. La recherche est lancé de 1000 à 10 000 images d'apprentissage par pas de 500 images.

Le programme nous affiche la courbe du Taux d'erreur en fonction du nombre d'image d'apprentissage soit :



On obtient 10 000 comme meilleures dimensions pour la projection.

Si l'on regarde la courbe on peut remarquer que plus la quantité d'informations (nombre d'images) augmente plus notre classifieur devient précis, ce qui paraît normal car plus on a d'images différentes plus la classification devient facile.

5 – Analyse des erreurs sur la classe 7

Peu importe la dimension la classe 7 reste celle avec le plus haut taux d'erreur.

Analysons les images de test de cette classe afin de mieux comprendre ce taux d'erreur.

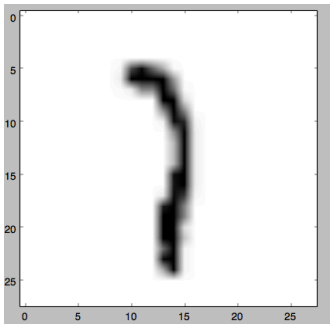
Exemple 1 : Le classifieur n'arrive pas à classer l'image de test n° 2695

Soit l'image ci-dessous :

Celle-ci devrait d'après le fichier test.txt être un « 7 » or, même à l'œil humain on a du mal à se dire que cette image représente un « 7 ». L'erreur du classifieur paraît alors légitime même s'il propose un « 5 ».

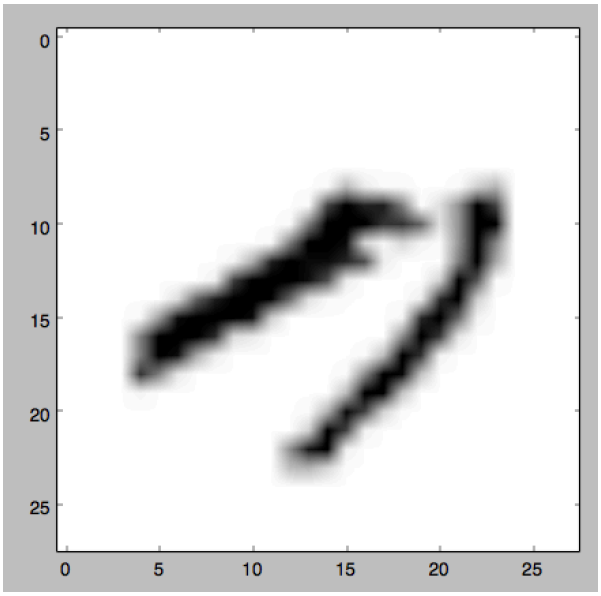
Exemple 2 : Le classifieur n'arrive pas à classer l'image de test n° 1260

Soit l'image ci-dessous :



Celle-ci devrait d'après le fichier test.txt être un « 7 », or encore une fois même à l'œil humain on a du mal à se dire que cette image représente un « 7 » mais plutôt un « 1 ». L'erreur du classifieur paraît alors légitime car il propose bien un « 1 ».

On peut conclure d'après ces deux exemples que les erreurs obtenues sont liées au fichier de test qui sont un peu ambigus. Pour les autres cas, nous conclurons que l'erreur est due aux erreurs contenues dans les fichiers d'apprentissage, car comme nous pouvons le voir ci-dessous il se trouve que dans nos fichiers d'apprentissage nous avons des images qui ne sont pas représentatives des classes (ici image de la classe 7).



La solution pour l'amélioration du classifieur serait peut-être alors d'avoir de meilleures images d'apprentissage, afin que les ces images d'apprentissage soient plus représentatives de la classe qui leur est attribuée. Mais on pourrait aussi augmenter considérablement le nombre d'images d'apprentissage ce qui permettrait aussi d'améliorer le taux d'erreur globale.

Annexe 1 : Comment ça marche le menu interactif ?

Menu :
1 - Prétraitement des données via ACP
2 - Apprentissage avec simplification
3 – Classification Bayésienne
4 - Prétraitement + Apprentissage + Classification
5 – Recherche de la meilleure dimension
6 – Recherche du meilleur nombre de données d'apprentissage
0 - Quitter
Sélectionner une action (0/1/2/3/4/5/6):

L'utilisateur peut choisir d'exécuter chaque module pas à pas ou d'exécuter tous les modules d'un coup.
Pour un fonctionnement total du programme (option 4), l'utilisateur doit saisir ensuite le nombre d'images pour l'apprentissage, la dimension de la projection, et le nombre d'images à reconnaître.
Pour un fonctionnement « pas à pas », les étapes 1, 2 et 3 sont séparables. C'est-à-dire qu'ils vont créer des fichiers de ressources pour que l'utilisateur puisse les réutiliser après.
Par exemple, si l'on veut prétraiter 10000 images puis faire l'apprentissage avec des différentes dimensions. Avec le fonctionnement total (option 4), on doit refaire le même prétraitement sur 10000 images pour chaque dimension. Alors qu'avec le fonctionnement « pas à pas », on peut faire le prétraitement une seule fois (option 1). Les ressources stockées dans un fichier vont être réutilisées directement quand on choisit ensuite l'apprentissage (option 2).
Par contre, pour la première exécution du programme, il n'y pas encore les fichiers de ressources, on doit forcément choisir l'option 4 ou exécuter les options 1, 2, 3 dans l'ordre.
À la fin d'exécution, le programme va afficher la matrice de confusion et le taux d'erreur globale.

Les options 5 et 6 effectuent respectivement en plus d'une recherche un affichage du taux d'erreur en fonction du nombre de dimension et un affichage du taux d'erreur en fonction du nombre de données d'apprentissage.

Annexe 2 : Les fonctions

- ModulePretraitementACP(nbImg)

nbImg = nombre d'images à prétraiter.
Objectif : Récupère les composantes principales de ces images.

- ModuleApprentissageAvecAcp(nbImg,nbDim,vectP,MatriceData)

nbImg : Nombre d'image à apprendre
nbDim : Dimension pour la projection
vectP : Ensemble de vecteur propre résultant de l'ACP
MatriceData : Ensemble des données d'apprentissage
Objectif : Faire l'apprentissage de chaque classe avec la simplification grâce à l' ACP.

- recupList(nbImg,nomFichier)

nbImg : Nombre d'images à lire
nomFichier : Nom du fichier texte (« train.txt » ou « test.txt »)
Objectif : Transformer le texte en liste des images associées à leurs classes.

- ModuleRecoImg(nbImgTest,nbDim,vectP,Moyenne,MatriceCov)

nbImgTest : Nombre d'image à tester
nbDim : Dimension pour la projection
vectP : Ensemble de vecteur propre résultant de l'ACP
Moyenne : Ensemble de vecteur moyen (par classe) résultant de l'apprentissage
MatriceCov : Ensemble de matrice de covariance (par classe) résultant de l'apprentissage
Objectif : Appliquer la loi normale et classifier les données.