Reconnaissance des Formes
Travaux Pratiques, 1ère séance /4
- Corrigé partiel -

Modalités de déroulement

- Séance en salle Linux
- Compte-rendu à remettre à l'issue de la séance, avec possibilité de compléter ultérieurement. Les sources et la rédaction doivent figurer dans un document unique, dans un format lisible par MsWord. Le fichier peut être envoyé par mail à :
  decoux@efrei.fr
  Si sa taille dépasse quelques centaine de Ko, il doit être compressé.
- Pour ce TP, vous aurez besoin d'informations disponibles sur le site RDF (données, programmes, corrigés des TP des années précédentes, résumé de cours, etc) :
  http://www-rdf.efrei.fr

Objectif de la séance

- Affichage de données par la méthode de Sammon ; étude des méthodes de classification kPPV, nuées dynamiques, VQ, LVQ
- Cas traités : données de fleurs Iris et images de caractères manuscrits pré-traités
- Langages et environnements de programmation : C et Scilab

1) Affichage de Sammon

1.1) On cherche à visualiser les données Iris. Pour cela on peut utiliser l'utilitaire *sammon* du LVQ_PAK. Commencer par installer cet outil (fichier *lvq_paq-3.1.tar*) à partir du site RDF, en effectuant les différentes étapes suivantes :
- sauvegarder le fichier sur le disque ;
- décompresser le fichier avec la commande "tar xvf lvq_paq-3.1.tar" ;
- effacer les exécutables et les fichiers objets ;
- compiler en renommant le fichier *makefile.unix* en *Makefile* (à l'aide de la commande *cp* d'UNIX, puis en tapant "make".

Remarques importantes :

Les différents utilitaires du LVQ_PAK peuvent traiter des fichiers de données portant l'extension *.dat*, un format un peu différent du format *.don* (voir document *lvq_pak.pdf*). Il faut :
- ajouter le nombre de caractéristiques des données au début des fichiers d'extension *.don*
- vérifier qu'en fin de fichier, il n'y a pas de retour à la ligne (sinon les utilitaires du LVQ_PAK plantent)

Appliquer l'utilitaire *sammon* du LVQ_PAK aux données du fichier *iris.don*. Déterminer des paramètres adéquats.

Syntaxe de la commande :

```
> sammon -cin iris.don -cout iris.sam -rlen 10000 -ps 1
```

"rlen" est le nombre d'itérations de l'algorithme (rappel : cet algorithme est basé sur la descente de gradient d'un critère d'erreur). L'inconvénient d'un nombre fixe d'itérations est que l'on risque d'arrêter l'algorithme alors que le critère n'est pas suffisamment faible pour que le résultat soit révélateur de la réalité.

1.2) Visualiser ces résultats également avec Scilab, en utilisant le fichier d'extension *.sam* généré par l'utilitaire *sammon*, en utilisant les fonctions :
- `fscanfMat` pour charger une matrice à partir d'un fichier,
- `plot2d` pour l'affichage,
- `xnumb` pour afficher des nombres,
- `xs2gif` pour générer un fichier graphique de type *.gif* et l'insérer dans le compte-rendu.

On écrit un programme pour afficher les points à 2D contenus dans le fichier d'extension .sam généré par l'utilitaire sammon du LVQ_PAK (programme *xnumb_sammon.sce*) :

```
a1=fscanfMat('iris.sam');
plot2d([-100,500],[-100,600],[-1,-1],"022")
xnumb(a1(:,1)/10.+200,a1(:,2)/10.+200,a1(:,3),0)
xs2gif(0, "tmp1.gif", 1)
```

Pour les données du fichier "iris.don", le fichier .gif généré est le suivant :

1.3) Interpréter ces résultats du point de vue du problème de classification posé (classes entremêlées, classes bien distinctes des autres, etc). Quelles sont les 2 espèces d'iris qui se ressemblent le plus ?

Dans le cas des données Iris, la classe Setosa est bien distincte des 2 autres, et devrait pouvoir être classifiée avec un taux proche de 100%. Par contre, les classes Versicolor et Virginica se chevauchent partiellement, par conséquent leur taux de classification ne devrait pas atteindre 100%.

1.4) Lancer une nouvelle fois le programme et comparer les résultats avec ceux obtenus précédemment. Les interpréter par rapport au principe de fonctionnement de l'algorithme de Sammon.

La position absolue des points n'est pas la même que précédemment, mais leur disposition relative (les uns par rapport aux autres) est la même.

La raison est que les coordonnées des points dans l'espace d'arrivée à 2 dimensions, sont initialisés aléatoirement dans l'algorithme.

1.5) Lancer la commande avec une valeur de *rlen* plut petite, égale à 100. Visualiser les résultats et les interpréter.

Avec une valeur plus petite pour le paramètre *rlen*, le critère d'erreur de l'agorithme n'a pas le temps de diminuer suffisamment, donc il y a encore une part d'aléatoire non négligeable dans la position des points en 2D.

2) Classification par kPPV

2.1) Pour le test des algorithmes de classification, nous allons utiliser les données du fichier *iris.don* puis celle du fichier *optdigits.don.* Décrire succintement le contenu de ces fichiers de données (nombre de classes, nature des éléments caractéristiques, nombre approximatif d'exemples par classe, etc).

Les données *Iris* sont consituées par 4 mesures effectuées sur 3 espèces d'Iris différentes. Ces mesures sont : la longueur et la largeur de pétale, et la longueur et la largeur de sépale. Il y a 50 exemples pour chacune des 3 classes.

Les données *Optdigits* sont constituées par 3 fichiers : *optdigits_tra.don*, *optdigits_tes.don* et *optdigits_all.don*. Le 3^{e} est la réunion des 2 premiers. Ils ont été convertis au format .don à partir des fichiers originaux (on peut trouver ces derniers par exemple à l'adress ……). Les informations sur le contenu de ces fichiers sont présentes dans le fichier .txt les accompagnant. Ils comportent des petite images, de taille 8x8 pixels, de chiffres manuscrits. *optdigits_tra.don* comporte les contributions de 30 personnes, et *optdigits_tes.don* celles de 13 personnes. Chaque pixel de ces images est un entier compris entre 0 et 16. Elles ont été obtenues à partir d'images de taille 32x32 pixels, dans lesquelles des moyennes des blocs de 4x4 pixels ont été calculées. Le fichier *optdigits_tra.don* comporte *3823* images et le fichier *optdigits_tes.don* 1797 images. Le nombre de classes est 10 (chiffres de 0 à 9), et le nombre approximatif d'exemples par classe est 380 pour le fichier d'apprentissage et 180 pour le fichier de test. De plus, il est indiqué dans le fichier d'information que la méthode des kPPV donne environ 98% de reconnaissance avec des données.

2.2) Appliquer l'algorithme des kPPV du programme *kppv.c* à ces deux types de données. Mettre en évidence l'influence des paramètres de l'algorithme sur le résultat de classification en testant quelques valeurs différentes des paramètres :
- *k* : 1, 3, 5 et 7 ;
- nombre de vecteurs initialement classés : 5, 7, 10 et 20.

On s'assurera que les données utilisées pour comme références initiales (vecteurs initialement classés) sont différentes des données de test, par exemple en examinant l'algorithme du programme kppv.c décrit dans le résumé de cours.
Remarque : pour effectuer tous ces tests de manière automatique, il suffit d'entrer toutes les commandes dans un fichier texte, de le rendre exécutable (à l'aide de la commande UNIX *chmod*), et d'exécuter ce fichier.

La syntaxe de la commande, par exemple pour les données Iris, k=3 et i=5 (i=nombre de vecteurs déjà classés en début d'algorithme), est la suivante :

```
> kppv iris.don k=3 i=5
```

On peut utiliser le fichier complet, car le programme se charge de répartir les vecteurs en vecteurs de référence et en vecteurs de test. De même, pour les données *Optdigits*, on utilise le fichier complet *optdigits_all*.don.

Avec différentes valeurs des paramètres *i* et *k*, les résultats sont les suivants :

iris.don :

I	k=1	k=3	k=5	k=7
5	94,1	94,8	74,8	96,3
10	95	95,8	95,8	95
20	93,3	93,3	94,4	93,3

optdigits.don :

i	k=1	k=3	k=5	k=7
5	85,4	85,4	87,5	89,7
10	91,3	95,4	94,4	95,3
20	94,1	94,7	94,9	95,4

2.3) Mêmes tests avec l'utilitaire *knntest* du LVQ_PAK. Déterminer les paramètres adéquats et préciser la syntaxe de la commande. Là encore, il faudra s'assurer que les vecteurs de référence sont différents des vecteurs de test.

L'étape préalable à la classification est la recherche de vecteurs de référence, par l'utilitaire *eveninit*. Le critère utilisé par cet utilitaire pour dire si un vecteur est représentatif ou non, est qu'il soit correctement classé par la méthode des kPPV (voir doc du LVQ_PAK).

Par exemple, si l'on souhaite 5 exemples initiaux pour chaque classe, cela signifie un paramètre *noc* égal à 3×5=15.

Les options à indiquer au programme sont :

- *noc*= 3×5 = 15 ;
- *din* indique le fichier de données d'entrée ;
- *cout* indique le fichier des vecteurs de référence, avec lesquels seront comparés les vecteurs à classer.

Pour s'assurer que les données de référence et les données de test sont différentes, il faut utiliser 2 fichiers différents avec les utilitaires *eveninit* et *knntest*. Il faut découper le fichier de départ en 2 parties, par exemple de tailles égales.

La ligne de commande correspondante est donc :

```
> eveninit -din iris1.dat -cout iris.cod -noc 15
```

La classification proprement dite peut alors avoir lieu :

```
> knntest -din iris2.dat -cin iris.cod -knn 3
```

Pour les données Optdigits, on utilise le fichier *optdigits_tar.don* pour eveninit et *optdigits_tes.don* pour *knntest*.

Les taux de reconnaissance globaux obtenus avec différentes valeurs de *k*, et différentes valeurs du nombre de vecteurs représentatifs (*noc*) sont :

iris.don :

noc	k=1	k=3	k=5	k=7
5	94,7	93,3	92	92
10	94,7	94,7	94,7	92
20	94,7	93,3	94,7	94,7

optdigits.don :

noc	k=1	k=3	k=5	k=7
5	61,3	54,5	49,1	45,2
10	70,3	69,1	66,5	62,2
20	81,8	78,8	75,2	73,9

2.4) Essayer d'améliorer les résultats obtenus avec les données *Optdigits* en augmentant le nombre de vecteurs de référence. Il suffira pour cela d'utiliser une valeur du paramètre *noc* suffisamment grande pour que la plupart des vecteurs du fichier utilisé par *eveninit* soient utilisés comme vecteurs de référence.

Le fichier *optdigits_tra.don* comporte environ 280 vecteurs par classe. On peut donc prendre une valeur de 300×10=3000 pour *noc*. Si l'effectif d'une classe est inférieur à ce nombre, le programme se charge d'utiliser plusieurs fois le même vecteur de référence. Le résultat atteint environ 98 %.

2.5) Interpréter les résultats de ces tests, du point de vue :
- de la comparaison entre les programmes *kppv.c* et *knntest* ;
- des paramètres optimaux pour les programmes ;
- des deux types de données utilisés ;
- des résultats similaires obtenus avec les données *caract_holland.don* et *chiffres.don* (voir corrigé de la 1ère séance de TP de l'année dernière).

D'après les résultats des 3 exercices précédents, on peut constater que les programmes *knntest* et *kppv* donnent à peu près les mêmes résultats, ce qui est logique, même si les 2 implémentations ne sont pas tout à fait identiques. Les résultats obtenus avec *kppv.c* sont meilleurs d'environ 5% de ceux obtenus avec le *LVQ_PAK*. Le premier utilise le fichier complet et le second les fichiers d'apprentissage et de test (ce qui revient au même puisque les données utilisées en référence, c'est à dire les vecteurs dont on indique la classe, ne sont plus utilisés dans la suite de l'algorithme).

Avec le programme *knntest* et les données *Iris*, le taux de reconnaissance reste à peu près constant quelles que soient les valeurs de *noc* et de *k*. On retrouve à peu près les mêmes résultats qu'avec *kppv.c*.

Dans le cas des données *Optdigits*, le taux de reconnaissance augmente quand *noc* augmente, et quand *k* diminue. Le premier constat est normal, car plus on utilise de vecteurs de référence, plus grande est la densité de points et plus riche est l'information sur les classes. Le deuxième constat (amélioration des résultats quand *k* diminue) pourrait vouloir dire que les frontières entre les classes sont très découpées, et donc que le lissage apporté par des grandes valeurs de *k* ne convient pas.

Avec les données Iris, les taux de reconnaissance reste à peu près constant en fonction de ces deux paramètres.

2.6) En apportant une modification au programme *kppv.c*, et en l'appliquant aux données *wine.don* (par exemple avec *i*=10 et *k*=3), montrer que la normalisation des données améliore les résultats de manière significative. Expliquer brièvement pourquoi. Répéter la même opération avec les données Iris.

Avec le LVQ_PAK, il n'est pas possible de normaliser les données. Il faut donc le faire comme pré-traitement.

Dans l'implémentation de l'algorithme de *kppv.c*, la fonction *normalis_donnees()* est utilisée. Son prototype est :

```
void normalis_donnees(vec, nb_vec, taille);
```

où :
- *vec* est un pointeur vers une liste de vecteurs (voir la structure *Vec* dans le code),
- *nb_vec* le nombre de vecteurs ;
- *taille* leur taille en nombre de composantes.

Il suffit de mettre cette fonction en commentaires pour la désactiver.

Avec les données Wine, après plusieurs exécutions de l'algorithme on constate un taux de reconnaissance de l'ordre de 70%, au lieu de 96% environ. On pourrait vérifier qu'avec les données *Iris*, le gain apporté par la normalisation est beaucoup moins important.

2.7) Essayer d'améliorer les résultats en utilisant la méthode de test *leave-k-out* (pour la mettre en œuvre, il suffit d'utiliser un nombre de vecteurs initiaux adéquat, et de lancer la méthode un nombre de fois suffisant pour pouvoir calculer des taux d'erreurs/de reconnaissance significatifs).

Cette méthode n'est pas applicable à la classification par les kPPV : il n'y a pas vraiment de base d'apprentissage et de base de test. Par contre, elle pourra être utilisée avec les méthodes d'aprentissage compétitif.

3) Classification par apprentissage compétitif : nuées dynamiques, VQ et LVQ

3.1) Le programme *compet.c* permet de tester, entre autres, la méthode des nuées dynamiques. L'appliquer aux données des fichiers *iris.don* puis *optdigits.don*. Tester dans les deux cas différentes valeurs de paramètres de l'algorithme, et leur influence sur les résultats.

L'algorithme des nuées dynamiques est le numéro 1 parmi les 3 méthodes implémentées dans ce programme (les 2 autres sont VQ et LVQ). Le paramètre *m* doit donc être égal à 1.

Le nombre de présentation de la base d'apprentissage peut être égal à 1. En effet les vecteurs de références sont obtenus par calcul de moyennes, donc ils sont complètement déterminés après la 1ère présentation. Le paramètre *n* peut donc être égal à 1.

Le paramètre restant est le nombre de vecteurs utilisés pour calculer le vecteur moyen initial. Il s'agit du paramètre *i*.

Les résultats sont rassemblés dans le tableau de la question suivante.

La syntaxe de la commande, par exemple pour i=5 (nombre de vecteurs utilisés pour calculer le vecteur représentant initial), est la suivante :

```
> compet iris.don i=5 m=1
```

m est le numéro de méthode (=1 pour les nuées dynamiques).

3.2) Même chose avec les versions VQ et LVQ de *compet.c*.

Les résultats obtenus sont :

iris.don :

noc (*)	nuees	vq	lvq
5			

10			
20			

optdigits.don :

noc (*)	nuees	vq	lvq
5			
10			
20			

(*) *noc* : nombre de vecteurs représentatifs ; *m* : méthode de classification

On voit que les résultats de classification s'améliorent quand le nombre de vecteurs représentants augmentent. Les meilleurs résultats seraient sans doute obtenus avec la méthode *leave-1-out*, constistant à utiliser tous les vecteurs disponibles sauf un, pour l'apprentissage. Le test est réalisé sur ces vecteurs non utilisés. Le test ne peut être valable que si cette opération (apprentissage+test) est répétée un grand nombre de fois, les vecteurs non-utlisés étant à chaque fois différents.

Cette méthode prend pas mal de temps. Pour aller plus vite, on peut utiliser la version *leave-k-out*, consistant à laisser de côté non plus un mais *k* vecteurs, de côté pour le test. Pour l'implémenter avec les utilitaires du LVQ_PAK, il suffit d'utiliser le fichier de données complet optdigits_all.don, avec par exemple un paramètre noc égal à 520x10, soit 5200. En effet, ce fichier comporte environ 560 vecteurs par classe. Avec ce paramètre il en resterait environ 560-520=40 par classe. On peut par exemple exécuter 10 fois de suite la même commande, et calculer le taux de reconnaissance moyen.

Les résulats obtenus sont les suivants :

3.3) Que peut-on conclure de ces résultats du point de vue de l'aspect supervisé ou non-supervisé des méthodes ?

Les résultats de classification sont globalement meilleurs avec la version supervisée de l'algorithme qu'avec la version non-supervisée, ce qui est normal dans la mesure où l'on corrige les erreurs de l'algorithme lors de l'apprentissage.

3.4) Essayer d'obtenir des résultats encore meilleurs avec quelques variantes de cette même méthode du LVQ_PAK.

Les différentes versions de l'algorithme LVQ du LVQ_PAK sont : LVQ1, LVQ2.1, LVQ3 et OLVQ1.

Les paramètres communs à toutes ces versions sont :

Le nombre d'exemples représentatifs par classe (*noc* pour "number of codebook vectors") ;

On applique l'apprentissage avec la moitié des données et la reconnaissance avec l'autre moitié.

Les autres paramètres sont :

- Pour LVQ2.1 et LVQ3, la taille de la fenêtre (*win*) ;

- Pour LVQ3 : epsilon (voir règle d'apprentissage).

Pour générer le fichier contenant les vecteurs représentatifs des classes, on peut utiliser l'utilitaire *eveninit*. Le fichier généré porte l'extension ".cod".

Avec les données du fichier *optdigits.don*, pour avoir 1 exemple représentatif par classe, on écrit "–noc 26" dans la ligne de commande ; en effet, il y a 10 classes et *eveninit* répartit les exemples représentatifs équitablement entre toutes les classes. Pour 5 exemples représentatifs par classe on écrit simplement "–noc 130".

Pour la génération des exemples représentatifs par classe, on tape la commande :

```
eveninit -din iris1.don -cout iris.cod -noc 30
```

Le taux d'apprentissage (paramètre *alpha*) est pris égal à 0,02 et le nombre de cycles d'apprentissage ("rlen") à 10000, mais une étude complète nécessiterait de tester plusieurs valeurs différentes. La reconnaissance est testée avec le 2e fichier.

```
lvq1 -din iris2.don -cin iris.cod -cout iris2.cod -rlen 10000 -alpha 0.02
```

Pour la version 2.1 il faut ajouter un paramètre de largeur de fenêtre : par exemple "–win 0.3" :

```
lvq2 -din iris2.don -cin iris.cod -cout iris2.cod -rlen 10000 -win 0.3
```

Pour lvq3 il y a un paramètre en plus par rapport à lvq2.1 : par exemple "–epsilon 0.1".

```
lvq3 -din iris2.don -cin iris.cod -cout iris2.cod -rlen 10000 -win 0.3 -epsilon 0.02
```

Pour *olvq1* (version à taux d'apprentissage optimisé), il faut d'abord utiliser l'utilitaire *balance*, qui détermine des taux d'apprentissage différents pour chaque classe, et les place dans un fichier ".cod". Le programme *balance* calcule les médianes des distances les plus courtes entre les prototypes de chaque classe et les égalise. Elle ajuste le nombre de prototypes par classe, pour qu'il soit optimal.

```
balance -din iris.don -cin iris.cod -cout iris2.cod
```

puis

```
olvq1 -din iris2.don -cin iris2.cod -cout iris3.cod -rlen 10000
```

A chaque fois il faut visualiser le résultat de la classification à l'aide de l'utilitaire "classify". Les arguments sont :

- le fichier des vecteurs représentatifs modifiés par apprentissage ("holland2.cod" dans l'exemple) ;
- le fichier de données initial.

```
accuracy -din iris2.don -cin iris2.cod
```

Pour les données *Optdigits* le principe est le même. Les taux de reconnaissance obtenus avec ces différentes méthodes et les différents paramètres sont les suivants :

iris.don :

noc (*)	Lvq1	lvq2.1	lvq3	olvq1
1				
5				

10				
20				
30				

optdigits.don :

noc	Lvq1	lvq2.1	lvq3	olvq1
1				
5				
10				
20				
30				

(*) *noc* : nombre de vecteurs représentatifs ; *m* : méthode de classification

3.4) Confronter ces résultats avec ceux de l'année dernière, portant sur d'autres données.