Stage L3 : Synthetic Visual Reasoning Test Challenge
ENS Cachan

Denoncin David, Lahouel Kamel, Destagnol Kévin
Maître de stage:
Alain Trouvé

23 juin 2010

Table des matières

Résumé

L'objectif de ce stage est de résoudre le Synthetic Visual Reasoning Test Challenge (svrt) qui consiste en la donnée de 23 problèmes binaires de classification concernant le machine learning et la reconnaissance de formes. On nous présente des arrangements de formes à classer selon une règle que l'on ne connaît pas a priori, on dispose alors de vingt exemples positifs et négatifs afin d'apprendre cette règle et l'algorithme que l'on réalise doit alors être capable de classer les autres arrangements.

La seule chose que l'on connaît a priori est que l'erreur de Bayes de chaque problème est nulle, autrement dit qu'on peut résoudre chaque problème avec un taux d'erreur nulle, que les deux classes de formes sont disjointes.

L'enjeu de ce challenge est de pallier aux algorithmes de classificaiton actuels tels qu'AdaBoost, kNN ou SVM qui, bien que donnant parfois de bons résultats, restent médiocres voires mauvais sur certains problèmes où ils ont des scores proches de ceux obtenus en classant par un tirage au hasard, alors que la plupart des problèmes peuvent aisèment être résolus par des êtres humains et ce avec un nombre très faibles d'exemples d'apprentissage.

Première partie

Présentation du problème

Chapitre 1

Exemples

Il s'agit d'un challenge proposé par Donald Geman et François Fleuret sur le site http ://www.idiap.ch/ fleuret/svrt/challenge.html concernant la reconnaissance de formes et le machine learning visant à démontrer certaines limites des algorithmes de classifications actuels tels qu'Adaboost et ensuite dans un second temps à chercher à développer des algorithmes plus performants capables de gérer ce types de problèmes. Ici, on a un exemple très simple en figure 1.1 où la règle à identifier est que l'on a deux formes identiques, l'une étant la translatée de l'autre. On constate que les algorithmes tels qu'AdaBoost parviennent assez difficilement à venir à bout de ce genre de problèmes. Mais on a d'autres exemples simples où on a des invariances entre les formes par similitudes qui ne sont pas du tout vus par ces algorithmes, à l'instar du problème suivant en figure 1.2. On a également des problèmes plus géométriques comme celui figure 1.3 où la règle à déterminer est qu'on a trois formes identiques plus une quatrième située à égale distance des trois autres.

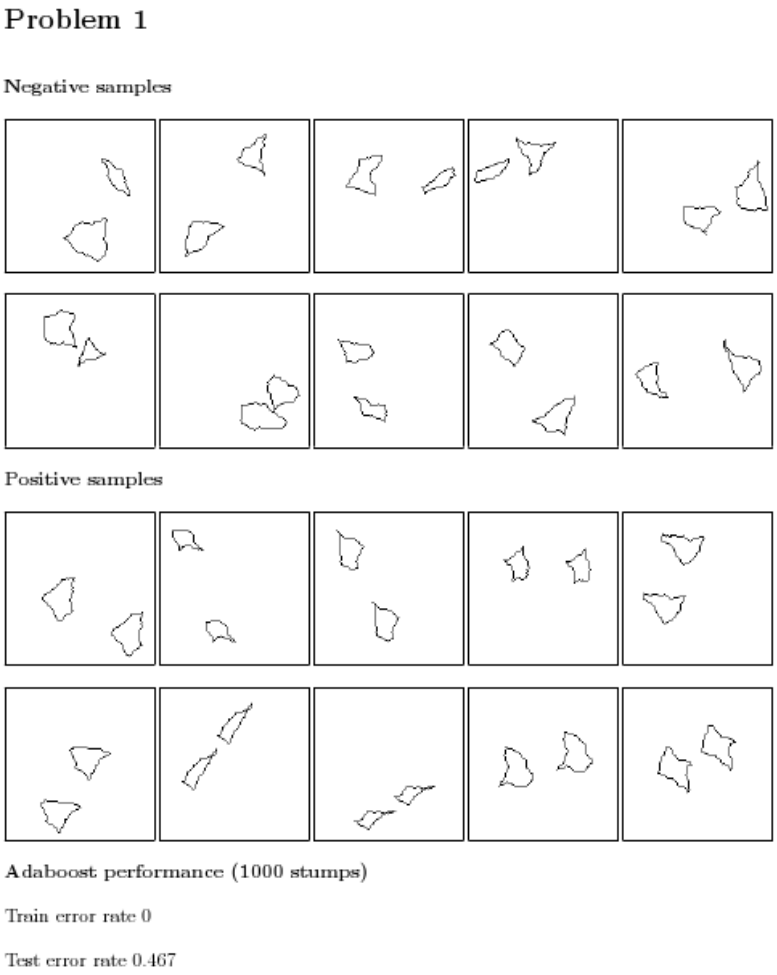

FIGURE 1.1 – Exemple simple

Et enfin, on a une dernière classe de problèmes comme celui figure 1.4 (parmi les treize dévoilés à ce jour) qui consiste en des problèmes de contacts et d'intérieur des formes comme le montre le problème suivant où la règle à identifier est que les deux formes se touchent. On constate ici en revanche que les algorithmes tels qu'Adaboost donnent d'assez bons résultats, car ils prennent très bien en compte les problèmes de densités de points et donc de contacts entre formes.

On peut donc constater que bien qu'efficaces sur certains problèmes, essentiellement ceux de contacts, les algorithmes tels qu'Adaboost ne sont pas efficaces du tout concernant des problèmes plus géométriques et concernant des invariances entre formes, qui sont pourtant des problèmes qui se résolvent très facilement et rapidemant à l'oeil nu, on observe en effet qu'il classe avec le même taux d'erreur que s'il classait en utilisant une règle du pile ou face. On voit donc bien que ces algorithmes peuvent être perfectionnés de manière à mieux répondre à ce type de problèmes, et c'est l'objet de ce stage.
Pour attaquer ce problème, M. Trouvé nous a suggéré de commencer par nous outiller pour traiter des problèmes d'invariances géométriques : savoir repérer que deux formes sont semblables ou homothétiques par exemple. Nous avons donc commencé tout trois par travailler notre géométrie différentielle pour arriver assez rapidement à résoudre ces problèmes d'invariances. La deuxième partie expose nos travaux mathématiques sur ces problèmes. Ensuite nous nous sommes, avec l'aide de M. Trouvé, attaqué à la programmation en Matlab d'un algorithme de classification qui fait l'objet de la troisième partie où nous y présentons nos idées, les difficultés que nous avons eu et commentons nos résultats.

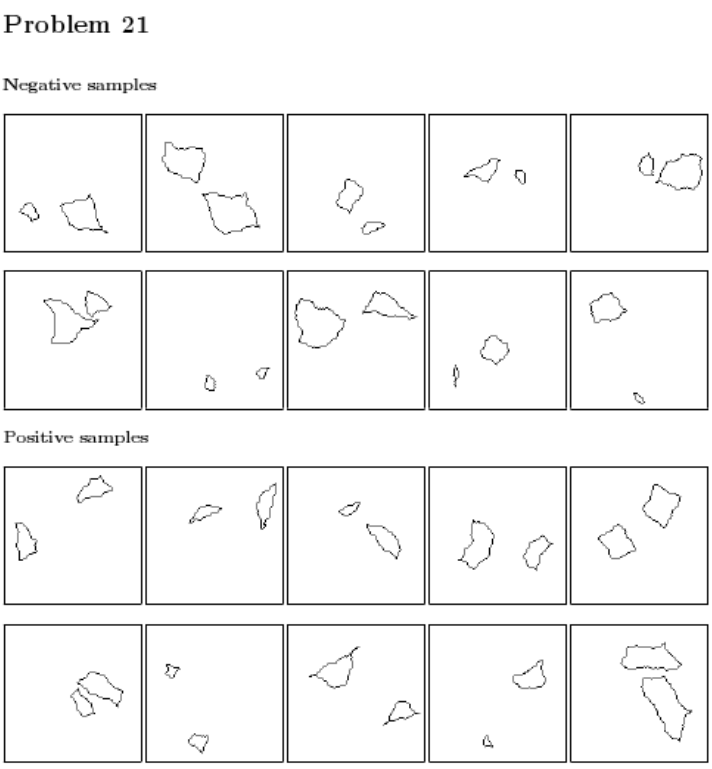

FIGURE 1.2 – Exemple avec invariance

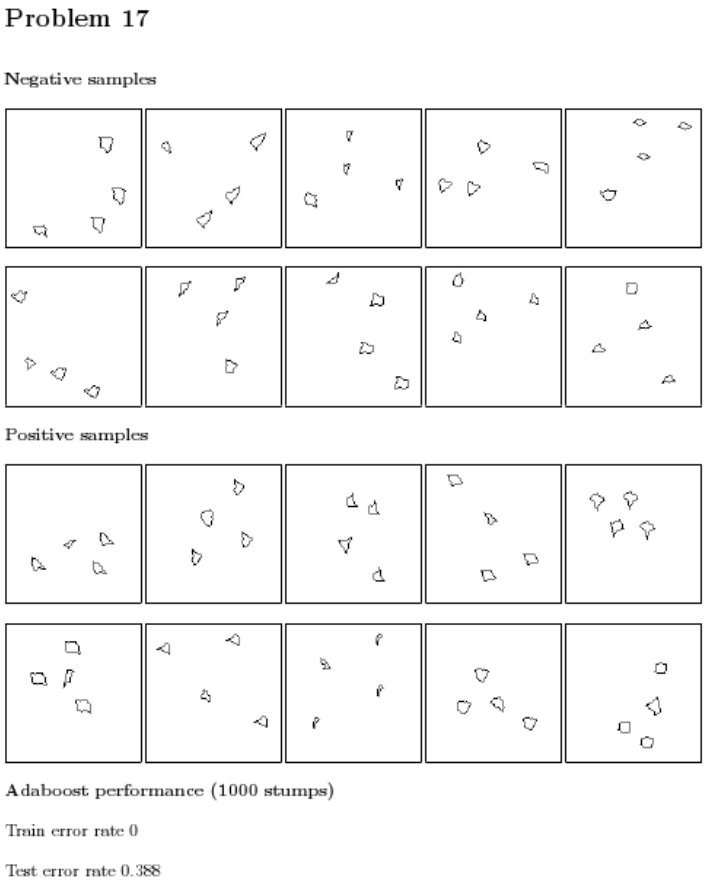

FIGURE 1.3 – Exemple géométrique

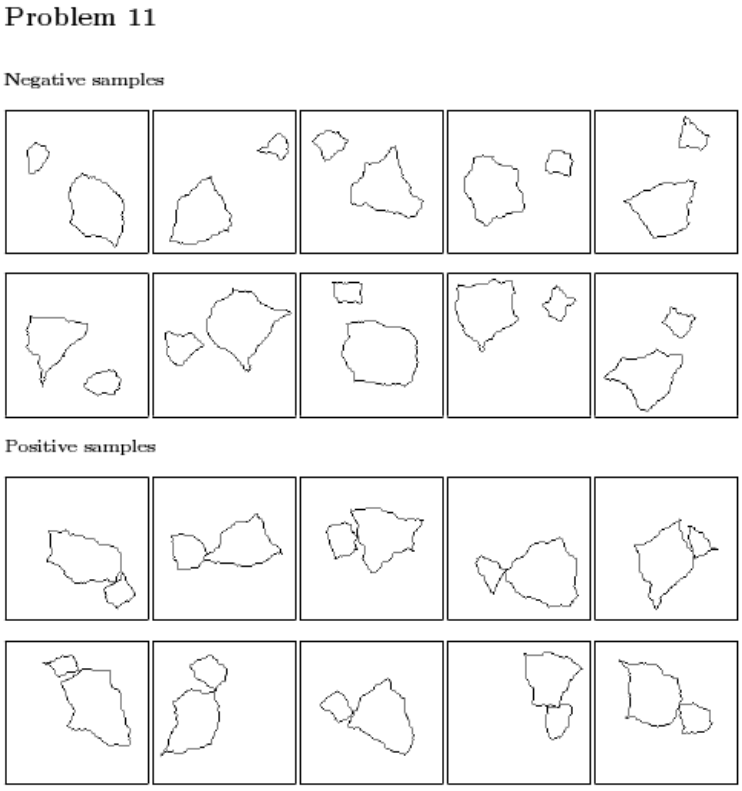

FIGURE 1.4 – Exemple avec contact

Deuxième partie

Étude mathématique des problèmes d'invariance

Chapitre 2

Préliminaires

On ne travaille pas directement sur une figure mais sur sa matrice de covariance, que nous définissons. C'est une représentation qui tient compte de la répartition des points d'une figure autour de son barycentre. Soit $(X_i)_{i\in I}$ une famille finie de points de $\mathbb{R}^2$. On pose $\overline{X} = \frac{1}{|I|}\sum_{i\in I} X_i$. On définit sa matrice de covariance par $S_X = \frac{1}{|I|}\sum_{i\in I}(X_i - \overline{X})^t(X_i - \overline{X})$. Cette représentation est une grande approximation. À chaque matrice de covariance, on associe de manière naturelle une conique de la façon suivante, à $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, on associe la conique $f(x,y) = [x \quad y]M^t[x \quad y] = ax^2 + (b+c)xy + dy^2$. C'est cette conique qui tracée donne une idée de la distribution des points autour de leur barycentre.
Cette matrice est symétrique positive en tant que combinaison linéaire convexe de telles matrices et le plus souvent pour notre problème, elle sera définie. Elle est également clairement invariante par translation des points.
De plus, on constate que si l'on applique une transformation linéaire bijective à un ensemble de points c'est-à-dire si on se donne $G \in GL_n(\mathbb{R})$, autrement dit si on perturbe quelque peu nos points, la matrice de covariance devient $S_{G.X} = G \times S_X \times^t G$ où $G.X = (G \times X_i)_{i\in I}$.
Il est donc naturel de considérer l'action de $GL_n(\mathbb{R})$ sur $S_n(\mathbb{R})$ (l'espace des matrices symétriques carrées sur $\mathbb{R}$ de taille n) définie par $A \cdot S = A \times S \times^t A$ où $A \in GL_n(\mathbb{R})$ et $S \in S_n(\mathbb{R})$, que l'on notera $\cdot$ contrairement au produit matriciel noté $\times$. On dispose donc d'un morphisme

$$\begin{aligned} GL_n(\mathbb{R}) &\xrightarrow{\Psi} \mathfrak{S}(S_n(\mathbb{R})) \\ G &\longmapsto \Psi_G, \end{aligned}$$

avec

$$\begin{aligned} S_n(\mathbb{R}) &\xrightarrow{\Psi_G} S_n(\mathbb{R}) \\ A &\longmapsto A \cdot G \end{aligned}$$

De plus, comme il s'agit d'une congruence on ne change pas la forme quadratique associée, cela définit donc également une action sur $S_n^{+\star}(\mathbb{R})$ (l'ensemble des matrices symétriques réelles définies positives).

Chapitre 3

Recherche de métriques riemanniennes invariantes

Notre objectif est alors de déterminer des métriques riemanniennes invariantes par rotations, homothéties et similitudes. On cherche donc une métrique g sur $S_n^\star(\mathbb{R})$ invariante sous l'action de $GL_n(\mathbb{R})$ ou équivariante, autrement dit telle que

$$\forall S, S' \in S_n(\mathbb{R}), \quad M \in S_n^\star(\mathbb{R}), \quad et \quad A \in GL_n(\mathbb{R}), \quad g_{A\cdot M}(A\cdot S, A\cdot S') = g_M(S, S').$$

La manière la plus naturelle de procèder est alors de choisir pour g_{I_n} le produit scalaire canonique sur $M_n(\mathbb{R})$. On obtient ainsi par transitivité de l'action, que $g_M(S, S') = g_{A\cdot M=I_n}(A\cdot S, A\cdot S')$ pour toutes matrices S et S' dans $S_n(\mathbb{R})$, M dans $S_n^\star(\mathbb{R})$ et A dans $GL_n(\mathbb{R})$ telle que $A\cdot M = I_n$.
Or, M appartient à $S_n^\star(\mathbb{R})$ donc il existe une unique matrice B dans $S_n^\star(\mathbb{R})$ telle que $B^2 = M$. On pose alors $A = B^{-1}$, où B est bien inversible. De plus, $A \in S_n^\star(\mathbb{R})$ donc ${}^tA = A$. Ainsi, on obtient bien que $A\cdot M = I_n$.
D'où, par un calcul élémentaire, on obtient que

$$\forall M \in S_n^\star(\mathbb{R}), \quad S, S' \in S_n(\mathbb{R}), \quad g_M(S, S') = Tr(M^{-1}SM^{-1}S')$$

On vérifie alors aisément qu'il s'agit bien d'une métrique vérifiant la condition imposée. Elle permet alors de définir une distance sur $S_n^\star(\mathbb{R})$ par la formule

$$\forall S, S' \in S_n^\star(\mathbb{R}), \quad d(S, S') = \inf_{\gamma:[0,1]\to S_n^\star(\mathbb{R}), \gamma\in C_m^1, \gamma(0)=S, \gamma(1)=S'} \left\{ \int_0^1 \sqrt{g_{\gamma(t)}(\gamma'(t), \gamma'(t))} \right\}.$$

Chapitre 4

Recherche de distances invariantes

On note $\Gamma_n(\mathbb{R})$ l'ensemble des similitudes de $\mathbb{R}^n$. L'objectif est maintenant de déteminer des distances invariantes sous l'action des similitudes pour l'action qui a été définie précedemment. Cela revient en réalité à déterminer une distance sur le quotient $S_n^\star(\mathbb{R})/\Gamma_n(\mathbb{R})$. On notera dans la suite $[S]$ l'orbite de S pour cette action.
Une méthode classique pour obtenir une telle distance est de poser

$$\forall S, S' \in S_n^\star(\mathbb{R}), \quad d([S],[S']) = \inf_{A,A' \in \Gamma_n(\mathbb{R})} \big\{ d(A \cdot S, A' \cdot S') \big\},$$

ce qui revient dans notre cas puisque l'action est transitive à prendre $d([S],[S']) = \inf_{A \in \Gamma_n(\mathbb{R})} \big\{ d(S, A \cdot S') \big\}$.
Le problème qui se pose alors est de calculer des géodésiques.

Chapitre 5

Calcul variationnel

5.1 Généralités

On souhaite donc déterminer le chemin γ (géodésique à vitesse constante) qui minimise la quantité $\int_0^1 |\frac{\partial \gamma}{\partial t}|_\gamma dt$.
On cherche donc à minimiser une quantité de la forme $\int_0^1 L(q, \frac{\partial q}{\partial t})dt$, où q est une fonction de t et de ϵ (par exemple $q(t,\epsilon) = q(t) + \epsilon\delta q(t)$) et où l'on impose que pour tout ϵ,

$$q(0,\epsilon) = a \, et \, q(1,\epsilon) = b$$

La quantité $J(\epsilon) = \int_0^1 L(q, \frac{\partial q}{\partial t})dt$ admet donc un point critique (un minimum) en $\epsilon = 0$.
L'idée est alors de dériver J par rapport à ϵ. On obtient alors

$$\frac{dJ}{d\epsilon} = \int_0^1 \left(\frac{\partial L}{\partial q}\frac{\partial q}{\partial \epsilon} + \frac{\partial L}{\partial \dot{q}}\frac{\partial \dot{q}}{\partial \epsilon} \right) dt \, .$$

Par intégration par parties, on obtient

$$\frac{dJ}{d\epsilon} = \int_0^1 \left(\frac{\partial L}{\partial q}\frac{\partial q}{\partial \epsilon} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right)\frac{\partial q}{\partial \epsilon} \right) dt + \left[\frac{\partial L}{\partial \dot{q}}\frac{\partial q}{\partial \epsilon} \right]_0^1 .$$

Or, le crochet $\left[\frac{\partial L}{\partial \dot{q}}\frac{\partial q}{\partial \epsilon} \right]_0^1$ est nul et un peu de théorie permet de justifier que l'intégrande est nulle donc on aboutit à l'équation d'Euler-Lagrange

$$\frac{\partial L}{\partial q} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = 0.$$

Une manière de résoudre cette équation est d'utiliser un point de vue hamiltonien en effectuant le changement de variable suivant $(q, \dot{q}) \mapsto (q, p)$ où $p = \frac{\partial L}{\partial \dot{q}}$. On pose alors $H(p,q) = \langle p, \dot{q} \rangle - L(q, \dot{q})$. On aboutit finalement au système suivant

$$\begin{cases} \frac{dp}{dt} = -\frac{\partial H}{\partial p} \\ \frac{dq}{dt} = -\frac{\partial H}{\partial q} \end{cases}$$

5.2 Application au cas de notre problème

Dans le problème introduit dans la première partie, on a $L(q, \dot{q}) = \frac{1}{2} g_q(\dot{q}, \dot{q})$. Le système précédent devient, après calculs

$$\begin{cases} \dot{S} = SPS \\ \dot{P} = -PSP \end{cases} ,$$

où S est notre géodésique et $P = \frac{\partial L}{\partial \dot{S}} = S^{-1}\dot{S}S^{-1}$.
On peut alors faire plusieurs remarques très importantes à ce stade des calculs. Tout d'abord le système auquel on aboutit ne fournit qu'une condition nécesaire que satisfait la géodésique et ne nous fournit pas

directement cette dernière. Il faut aller chercher d'autres infomations sur cette géodésique et les combiner avec le système, ce qui sera fait en détail dans la section "Espaces tangents et orthogonalité". Enfin, on remarque que le produit PS est constant d'après notre système (il suffit pour le voir de dériver le produit par rapport au temps ou d'appliquer le thèroème de Noether), ce qui permet de voir que toutes les géodésiques sont des exponentielles et la résolution du système est alors aisée.

Chapitre 6

Géométrie Différentielle

6.1 Espaces tangents et orthogonalité

Puisque la géodésique (à vitesse constante) est le plus court chemin entre 2 classes de similitude de 2 matrices symétriques définies positives, la géodésique doit être orthogonale aux fibres,c'est à dire aux classes de $S(t)$. Autrement dit, pour tout t dans $[0,1]$, $\dot{S}$ est orthogonal à T_S où T_S est l'espace tangent de la fibre au point $S(t)$. On peut alors remarquer que pour une matrice S fixée, et en ne considérant que les matrices de rotation de notre classe, la fibre peut être paramétrée par l'application suivante

$$\Gamma_n(\mathbb{R}) \xrightarrow{\Phi} [S]$$

$$R \longmapsto R.S,$$

où $\Gamma_n(\mathbb{R})$ est l'ensemble des similitudes réelles et $S = \Phi(Id)$. On déduit de cette paramétrisation que l'espace tangent en S dans ce cas est $Im(d_{Id}\Phi)$. On obtient donc l'ensemble des $HS - SH$ où $H \in \mathcal{A}_n(\mathbb{R})$ où $\mathcal{A}_n(\mathbb{R})$ est l'ensemble des matrices antisymétiques réelles de taille n.
On obtient donc que

$$\forall R \in \mathcal{A}_n(\mathbb{R}), \quad \forall t \in [0,1], \quad Tr(S^{-1}(t)(RS(t) - S(t)R)S^{-1}(t)\dot{S}(t)) = 0$$

soit

$$Tr(R(\dot{S}(t)S^{-1}(t) - S^{-1}(t)\dot{S}(t))) = 0$$

où $\dot{S}(t)S^{-1}(t) - S^{-1}(t)\dot{S}(t)$ est une matrice antisymétrique, donc en prenant $R =^t (\dot{S}(t)S^{-1}(t) - S^{-1}(t)\dot{S}(t))$, on obtient que

$$\dot{S}(t)S^{-1}(t) - S^{-1}(t)\dot{S}(t) = 0$$

soit que S et $\dot{S}$ commutent. Elles sont donc codiagonalisables et on pourra supposer que S est diagonale (donc P et $\dot{S}$ le sont aussi).
De même, en ne considérant que l'action des matrices d'homothéties, on obtient que

$$\forall \lambda \in \mathbb{R}, \quad \forall t \in [0,1], \quad \langle \lambda Id S - S\lambda Id \rangle_S = 2\lambda Tr(S^{-1}(t)S(t)S^{-1}(t)\dot{S}(t)) = 0$$

soit $Tr(PS) = 0$ donc on obtient une nouvelle condition $Tr(PS) = 0$.
On peut alors remarquer comme conséquence de la propriété $Tr(PS) = 0$ que le déterminant de S reste constant (il suffit pour cela de différentier le déterminant de S) et puisque nous pouvons partir de n'importe quel point en se déplacant sur la fibre correpondant à la matrice de départ, il suffit donc de partir de la forme diagonalisée et telle que le produit de ses valeurs propres soit égal a 1, dans la classe de $S(0)$. Et puisque nous savons que $S(t)$ va rester diagonale et de déterminant constant, nous sommes sûr d'arriver sur le même type de matrices pour la classe de $S(1)$.

$S(t)$ a donc la forme suivante :

$$\begin{bmatrix} \lambda_1(t) & & 0 \\ & \ddots & \\ 0 & & \lambda_n(t) \end{bmatrix},$$

Avec $\prod_{i=1}^{n} \lambda_i(t) = 1$

6.1.1 Distance invariante par similitudes

On rappelle que l'on a le système à résoudre est le suivant

$$\begin{cases} \dot{S} = SPS \\ \dot{P} = -PSP \end{cases}$$

Comme le déterminant est constant, on renormalise par le déterminant, c'est-à-dire que l'on part de la forme de détermiant 1. Comme P, S et $\dot{S}$ sont codiagonalisables on peut considérer que toutes les matrices sont diagonales. On utilise alors la proprtiété que PS est constante. Le système devient alors $\dot{S} = SA$ où A est une matrice constante. Or, l'action étant transitive, on peut se ramener à $S(0) = Id$ en effectuant le changement de variable $S \mapsto S(0)^{-\frac{1}{2}} S S(0)^{-\frac{1}{2}}$. Le système devient alors $\dot{U} = UB$ où $U = S(0)^{-\frac{1}{2}} S(t) S(0)^{-\frac{1}{2}}$ et $B = S(0)^{\frac{1}{2}} PSS(0)^{-\frac{1}{2}}$. On obtient donc que $U(t) = \exp(tB)$ car $U(0) = Id$ et donc on cherche à déterminer B. Pour se faire, on évalue en 1, et on obtient $S(0)^{-\frac{1}{2}} S(1) S(0)^{-\frac{1}{2}} = \exp(B)$. Or, $S(0)^{-\frac{1}{2}} S(1) S(0)^{-\frac{1}{2}}$ est symétrique définie positive, donc on a un résultat d'unicité du logarithme matriciel et on en déduit donc que B est semblable à $Diag(\ln(\mu_1), ..., \ln(\mu_n))$ où les μ_i désignent les valeurs propres normalisées de la matrice $S(0)^{-\frac{1}{2}} S(1) S(0)^{-\frac{1}{2}}$.

Le calcul de la distance est alors élémentaire et l'on obtient :

$$d([S],[S']) = \sqrt{\sum_{i=1}^{n} \ln^2 \mu_i}$$

où μ_i désigne les valeurs propres de $S(0)^{-1/2} S(1) S(0)^{-1/2}$ où $S(0)$ et $S(1)$ sont normalisées par leur déterminant.

On constate que cette distance est bien invariante par homothétie mais ne l'est malheureusement pas par rotation. On doit donc la modifier pour qu'elle le devienne, or, l'action d'une rotation sur une matrice diagonale est de permuter les valeurs propres, ainsi, on pose $d([S],[S']) = \inf_{\sigma \in \mathfrak{S}_n} \sqrt{\sum_{i=1}^{n} \ln^2 \mu_i}$

Dans le problème qui nous intéresse, en dimension 2, on obtient, en considérant les valeur propres normalisées de $S(0)$ et $S(1)$ $d([S],[S']) = \inf(\sqrt{2} \left|\ln(\frac{\lambda_2(0)}{\lambda_1(1)})\right|, \sqrt{2} \left|\ln(\frac{\lambda_1(0)}{\lambda_2(1)})\right|)$.

6.1.2 Distance invariante par rotations uniquement

C'est la même chose mis à part que l'on n'a plus la condition $Tr(PS) = 0$, et donc le déterminant n'est plus constant, la résolution du système est donc la même sauf que l'on ne considère plus les valeurs propres normalisées mais les véritables valeurs propres de $S(0)$ et $S(1)$. On obtient donc la même expression de la distance, seules les notations changent puisque les λ_i désignent les valeurs propres de $S(0)^{-\frac{1}{2}} S(1) S(0)^{-\frac{1}{2}}$ avec $S(0)$ et $S(1)$ non renormalisées :

$$d([S],[S']) = \inf_{\sigma \in \mathfrak{S}_n} \sqrt{\sum_{i=1}^{n} \ln^2 \mu_i}$$

Ce qui, dans le cadre de notre problème donne, en considérant les valeur propres de $S(0)$ et $S(1)$ $\inf\left(\sqrt{\ln(\frac{\lambda_1(0)}{\lambda_1(1)})^2 + \ln(\frac{\lambda_2(0)}{\lambda_2(1)})^2}, \sqrt{\ln(\frac{\lambda_2(0)}{\lambda_1(1)})^2 + \ln(\frac{\lambda_1(0)}{\lambda_2(1)})^2}\right)$

6.1.3 Distance invariante par homothéties uniquement

S et $\dot{S}$ ne commutent plus et donc on n'a plus la codiagonalisation et le système n'est plus diagonal, mais en revanche on a bien $Tr(PS) = 0$, donc le déterminant de S est constant. On résoud le système de la même manière : on utilise la propriété que PS est constante. Le système devient $\dot{S} = SA$ où A est une matrice constante. L'action est transitive, on peut se ramener à $S(0) = Id$ en effectuant le changement de variable $S \mapsto S(0)^{-\frac{1}{2}} S S(0)^{-\frac{1}{2}}$. Le système devient $\dot{U} = UB$ où $U = S(0)^{-\frac{1}{2}} S(t) S(0)^{-\frac{1}{2}}$ et $B = S(0)^{\frac{1}{2}} S S(0)^{-\frac{1}{2}}$. On obtient dons que $U(t) = \exp(tB)$ car $U(0) = Id$ et donc on cherche à déterminer B. Pour se faire, on évalue en 1, et on obtient $S(0)^{-\frac{1}{2}} S(1) S(0)^{-\frac{1}{2}} = \exp(B)$. Or, $S(0)^{-\frac{1}{2}} S(1) S(0)^{-\frac{1}{2}}$ est symétrique définie positive, donc on a toujours le même résultat d'unicité du logarithme matriciel et on en

déduit que B est semblable à $Diag(\ln(\mu_1), ..., \ln(\mu_n))$ où les μ_i désignent les valeurs propres normalisées de la matrice $S(0)^{-\frac{1}{2}}S(1)S(0)^{-\frac{1}{2}}$. On en déduit alors que $d([S],[S']) = \sqrt{\sum_{i=1}^{n} \ln^2(\mu_i)}$

6.2 Pour aller plus loin : carte exponentielle

Pour aller plus loin, on s'est intéressé à la représentation de la carte exponentielle de notre espace au voisinage de l'identité. On rappelle qu'à chaque matrice de covariance, on associe de manière naturelle une conique de la façon suivante, à $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, on associe la conique $f(x,y) = [x \quad y]M^t[x \quad y] = ax^2 + (b+c)xy + dy^2$. On cherche alors à représenter la carte exponentielle en l'identité. Celle-ci est en réalité l'exponentielle classique et pour se faire on considère la décomposition de notre espace suivant les trois matrices

$$e_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} e_2 = \begin{pmatrix} 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 \end{pmatrix} e_3 = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{pmatrix}$$

On calcule alors l'exponentielle de $te_1, \quad se_2$ et re_3 pour $t, s, r \in \mathbb{R}$ et on les représente alors sur le graphique suivant où l'axe des abscisses est dirigé par e_1 et l'axe des ordonnées par e_2, tandis que le troisième est de manière à ce que (e_1, e_2, e_3) soit un triplet direct.
On obtient alors aisément (la matrice étant diagonale) que

$$\forall t \in \mathbb{R} \exp(te_1) = \begin{pmatrix} \exp(\frac{t}{\sqrt{2}}) & 0 \\ 0 & \exp(-\frac{t}{\sqrt{2}}) \end{pmatrix}$$

et que

$$\forall t \in \mathbb{R} \exp(te_3) = \begin{pmatrix} \exp(\frac{t}{\sqrt{2}}) & 0 \\ 0 & \exp(\frac{t}{\sqrt{2}}) \end{pmatrix}$$

pour les mêmes raisons. On montre pour e_2, par une récurrence triviale sur $k \in \mathbb{N}$ que $e_2^{2k} = \frac{1}{2^k} Id$ et que $e_2^{2k+1} = \frac{1}{2^k} e_2$. D'où, on en déduit que

$$\forall t \in \mathbb{R}, \exp(te_2) = \sum_{i=0}^{+\infty} \frac{t^i e_2^i}{i!} = \sum_{i=0}^{+\infty} \frac{t^{2i} e_2^{2i}}{(2i)!} + \sum_{i=0}^{+\infty} \frac{t^{2i+1} e_2^{2i+1}}{(2i+1)!} = \sum_{i=0}^{+\infty} \frac{t^{2i} Id}{2^i (2i)!} + \sum_{i=0}^{+\infty} \frac{t^{2i+1} e_2}{2^i (2i+1)!}$$

$$= \sum_{i=0}^{+\infty} \frac{(\frac{t}{\sqrt{2}})^{2i} Id}{(2i)!} + \sqrt{(2)} \sum_{i=0}^{+\infty} \frac{(\frac{t}{\sqrt{2}})^{2i+1} e_2}{(2i+1)!}$$

On déduit alors finalement de tout cela que

$$\forall t \in \mathbb{R}, \exp(te_2) = \begin{pmatrix} \cosh(\frac{t}{\sqrt{2}}) & \sinh(\frac{t}{\sqrt{2}}) \\ \sinh(\frac{t}{\sqrt{2}}) & \cosh(\frac{t}{\sqrt{2}}) \end{pmatrix}$$

On obtient alors le graphe ci-dessous figure 6.1 où lorsque les ellipses viennent vers le lecteur elles sont homothétiques d'un rapport plus grand que 1 et inversement lorsqu'elles plongent à travers le document elles sont homothétiques d'un rapport plus petit que 1. L'axe horizontal (axe des x) correspond à e_1, l'axe vertical (axe des y) à e_2, et l'axe traversant le document (axe des z) à e_3. On identifie alors aisément les quotients par les différents sous-espaces considérés : en prenant le quotient par les homothéties on obtient exactement la figure 6.1, en quotientant par les rotations on obtient le demi-plan $y = 0, x > 0$, et en quotientant par les similitudes on obtient la demi-droite $y = 0, z = 0, x > 0$.

On a alors un espace où les singularités sont envoyées à l'infini et difféomorphe à notre espace tangent à notre variété en l'identité.

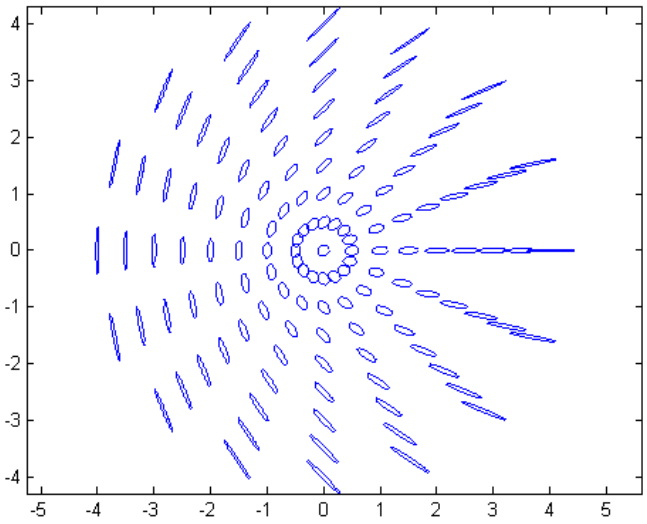

FIGURE 6.1 – carte exponentielle en l'identité

Troisième partie

Programmation d'un algorithme de classification

Chapitre 7

Aspect apprentissage

7.1 Principe de base

Un classifieur de Bayes Naif est un classifieur (basé sur des statistiques Bayésiennes) qui classe un objet en faisant un test statistique qui prend en compte un certain nombre des propriétés de l'objet en connaissant les propriétés en question pour les classes à choisir. Nous faisons également l'hypothèse que ces propriétés **sont indépendantes**. Nous choisissons ce classsifieur vu qu'il est simple, robuste et souple. Nous souhaitons également, en choisissant ce classifieur, mettre l'accent sur l'importance du problème de la représentation des données dans la performance finale.

7.2 Présentation de la base de données

Pour chaque problème, nous disposons d'un nombre n d'images dans une base d'apprentissage (nous connaissons la classe à laquelle appartient chacune des images). Pour chaque image, nous extrayons un nombre de caractéristiques (149).Nous disposons alors d'une matrice réelle où chaque ligne correspond à une image et chaque colonne à une caractéristique (sauf la première qui correspond à la classe elle même 0 ou 1). En notant, à caractéristique fixée et pour une classe donnée, x_i la valeur que prend cette caractéristique pour une image i appartenant à cette classe, on suppose que la caractéristique en question est une variable aléatoire $X \sim \mathcal{N}(m, \sigma^2)$ où on estime la variance et la moyenne par :

$$\widehat{m} = \frac{1}{n_{classe}} \sum_{i=1}^{n_{classe}} x_i$$

et

$$\widehat{\sigma}^2 = \frac{1}{n_{classe} - 1} \sum_{i=1}^{n_{classe}} (x_i - m)^2$$

où n_{classe} est le nombre d'images dans la base d'apprentissage appartenant à la classe en question.

7.3 Utilisation de la base d'apprentissage et premier test de vraisemblance

7.3.1 Cadre théorique

Notons X_k^0 resp (X_k^1) la variable aléatoire correspondant à la $k$ième caractéristique dans la classe 0 (resp1) et p le nombre de caractéristiques. Notons $Y^0 = (X_1^0, X_2^0, ..., X_p^0)$, $Y^1 = (X_1^1, X_2^1, ..., X_p^1)$ et, pour une image pour laquelle on veut effectuer le test "classe 1 contre classe 0", $(x_1, x_2, ..., x_p)$ les observations correspondant aux p caractéristiques. Le tout est de décider si $(x_1, x_2, ..., x_p)$ est une observation de Y^0 et donc l'image sur laquelle le test est effectué appartient à la classe 0 ou si $(x_1, x_2, ..., x_p)$ est une observation de Y^1 et dans ce cas l'image est à mettre dans la classe 1. Nous avons donc l'hypothèse H_1 : l'image est dans la classe 1 et l'hypothèse H_0 : l'image est dans la classe 0. Nous effectuons alors un test de Neymann-Pearson pour prendre une décision.

Notons $L_1(x_1, x_2, ..., x_p) = \prod_{i=1}^{p} f_i^1(x_i)$ où les f_i^1. sont les densités des gaussiennes de la classe 1 correspondant aux "p" caractéristiques, et $L_0(x_1, x_2, ..., x_p) = \prod_{i=1}^{p} f_i^0(x_i)$ où les f_i^0 sont les densités des gaussiennes de la classe 1 correspondant aux "p" caractéristiques. Notre rapport de vraisemblance est alors égal à :

$$\frac{L_1(x_1, x_2, ..., x_p)}{L_0(x_1, x_2, ..., x_p)} = \frac{\prod_{i=1}^{p} f_i^1(x_i)}{\prod_{i=1}^{p} f_i^0(x_i)}$$

Etant donné que le nombre d'images dans la classe 0 est égal à celui de la classe 1 i.e $\mathbf{P}(classe_1) = \mathbf{P}(classe_2) = \frac{1}{2}$, nous choisissons 1 comme seuil pour le rapport de vraisemblance, c'est à dire :

- Si $\frac{\prod_{i=1}^{p} f_i^1(x_i)}{\prod_{i=1}^{p} f_i^0(x_i)} \geq 1$, alors l'image est classée dans la classe 1.
- Sinon l'image est classée dans la classe 0

Nous avons donc

$$d(x_1, ..., x_p) = \mathbf{1}_{\frac{\prod_{i=1}^{p} f_i^1(x_i)}{\prod_{i=1}^{p} f_i^0(x_i)} \geq 1}$$

Intuitivement,ce que nous faisons c'est : On tire un vecteur x (ce tirage correspond à l'observation des caractéristiques de l'image), on regarde, parmi les deux densités des deux classes, celle pour laquelle x "serait dans une région de l'espace plus dense",c'est à dire qui donne le maximum de la probabilité à posteriori de la classe sachant les caractéristiques. Cette tache d'apprentissage est faite par la fonction intitulée naivebayes.

Le gros inconvénient de cette méthode est que nous prenons en compte toutes les caractéristiques dans le rapport de vraisemblance. Un grand nombre de caractéristiques vient alors "bruiter" notre rapport de vraisemblance et donc notre décision.

Un autre inconvénient est que toutes les caractéristiques ne sont pas définies pour une image donnée et que nous mettons un 0 pour la case correspondant à cette caractéristique (exemple : une caractéristique correspondant à une comparaison inter-classes de similitudes alors que l'image ne possède qu'une seule classe de similitude).

D'où la nécessité de :

- Trouver un critère qui classe les caractéristiques selon leur pertinence dans le problème grâce à la phase d'apprentissage.
- Introduire un objet correspondant à "case vide" à la place des zeros evoqués ci-dessus.

Chapitre 8

Choix des caractéristiques extraites d'une image

8.1 Idées de base

L'idée générale est la suivante : on prend en compte des invariances qui amènent à considérer des relations d'équivalence entre formes qui décrivent la géométrie des figures que l'on regarde.
En pratique, les relations ne sont pas des relations d'équivalences puisque l'on n'a pas toujours des distances égales à 0, on introduit alors des seuils et le problème de repérer les classes d'équivalences se transforme en un problème de clustering (repérer des paquets) dans le quotient. Cela permet d'être robuste au bruit. De manière plus précise, à chaque fois que l'on choisit une relation d'équivalence on en décrit de manière générique le quotient correspondant : on inscrit le nombre de regroupements que permet cette relation (clusters), ainsi que les cardinaux des trois plus gros clusters en ordre décroissant. Puis l'on décrit l'organisation intracluster, intercluster. L'étude est étagée en 3 niveaux qui correspondent à des vues de plus en plus globales de l'image, avec un niveau intermédiaire : la description intracluster qui est elle un peu particulière.

8.1.1 Niveau 0 : prise en compte d'invariances, description intracluster

Au niveau le plus élémentaire on a choisi plusieurs relations d'équivalences. Nous étudions les relations "être semblable à", "être une rotation de", "être homothétique à", "être le translaté de", "être de même aire que", "être en contact avec". Nous étudions aussi la relation triviale, dont l'unique cluster est formé de toutes les figures de l'image. Pour chacune de ces relations on stocke alors le nombre de clusters et les cardinaux des trois plus gros en ordre décroissant.
On essaie alors de décrire géométriquement l'organisation de chaque cluster dans l'image. A l'intérieur d'un cluster on introduit alors la relation "être à même distance de". C'est-à-dire que l'on calcule toutes les interdistances entre barycentres de figures d'un même cluster, et qu'on stocke alors pour chaque clusters distincts 4 caractéristiques : le nombre d'interdistances différentes, et les trois plus grandes distances.

On stocke aussi le centre d'intertie des trois plus gros clusters.

8.1.2 Niveau 0.5 : description intercluster

On essaie alors de décrire l'organisation des clusters entre eux : on prend les trois plus gros clusters, et on étudie la relation "être à même distance de" (qui n'est ici plus une relation d'équivalence, mais permet tout de même des regroupements) entre éléments de clusters distincts. C'est-à-dire que l'on calcule toutes les interdistances entre barycentres de figures de clusters distincts, et qu'on stocke alors pour deux clusters distincts 4 caractéristiques : le nombre d'interdistances différentes, et les trois plus grandes distances.

On fait de même en remplaçant les clusters par leur centre d'inertie.

8.1.3 Niveau 1 : description générale de l'image

Les trois plus gros clusters du niveau 0 sont réduits à la matrice d'inertie des barycentres des figures qui les composent, et on étudie la nouvelle image à partir du niveau 0. Cela pourrait donner lieu à un algorithme récursif mais on ne le fait qu'une fois, on se retrouve rapidement avec des matrices d'inerties non inversibles.

8.1.4 Niveau 2 : description "abstraite" de l'image

Pour les trois plus gros clusters du niveau 0 : on les réduit à leur centre d'inertie, et l'on prend la matrice d'inertie que l'on projette sur la demi-droite correspondant au quotient de l'espace par les similitudes, et on prend la distance à l'identité (voir figure 8.1 pour avoir une idée de la manière dont varient les ellipses associées aux matrices de covariances avec la distance à l'identité)

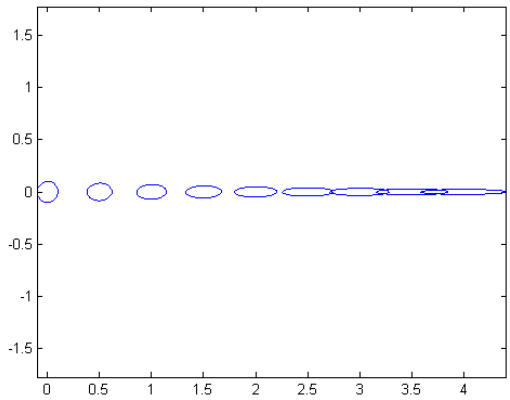

FIGURE 8.1 – La demi-droite correspondant au quotient par les similitudes

8.1.5 Commentaires

Ces choix de caractéristiques sont sujets à beaucoup de questions et remarques. On peut objecter qu'il n'y a pas de relation d'équivalence intercluster, que nous avons peut-être choisi ces caractéristiques pour résoudre le SVRT et qu'elles ne sont pas forcément très intéressantes. On peut aussi se demander comment choisir un seuil pour dire si deux formes sont dans le même cluster ou non, ou lorsque l'on est au niveau 1 et que l'on repart au niveau 0, quelle relation d'équivalence choisit-on ? Ces questions seront abordées dans la partie suivante. Avant cela, attachons-nous à développer un exemple qui nous a guidé dans le choix des caractéristiques.

8.2 Exemple

Condidérons l'image de la figure 8.2, et prenons l'exemple de la relation de similitude pour la suite. Au niveau 0 on voit donc trois clusters ayant 4, 3 et 2 éléments. On stocke les abscisses et ordonnées des barycentres des trois clusters. Pour le plus gros cluster, la figure formée est presque un carré, on aura donc 2 vecteurs de translation de normes différentes dans ce cluster (côté et diagonale voir figure 8.3). On fait de même pour les deux autres clusters.

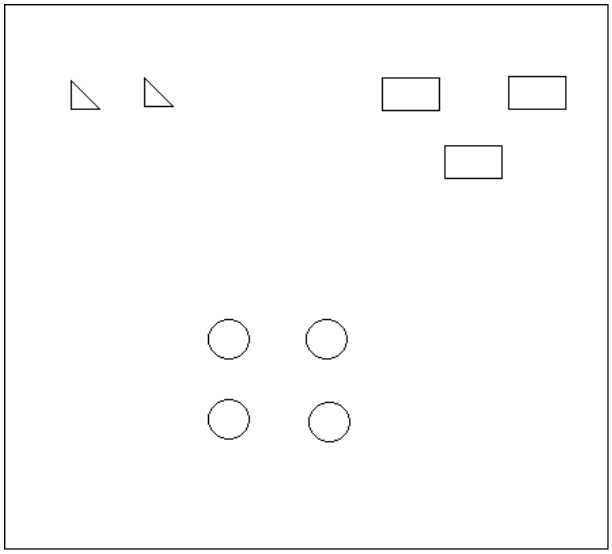

FIGURE 8.2 – Exemple

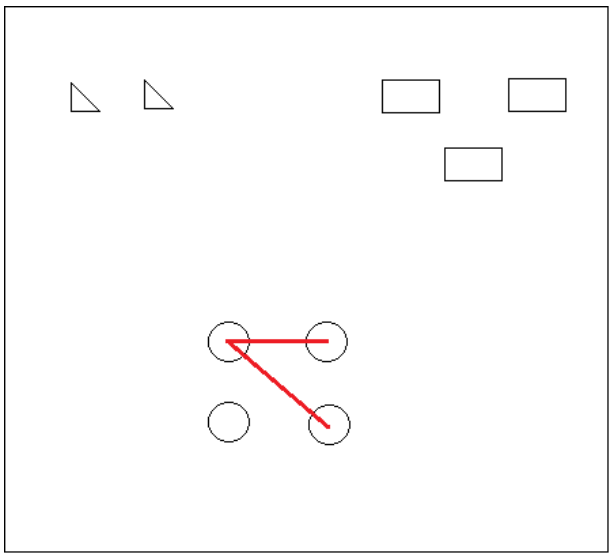

FIGURE 8.3 – Niveau 0 : description intracluster

Au niveau 0.5, on calcule toutes les distances entre les cercles et rectangles, cercles et triangles, triangles et rectangles. Sur la figure 3 on montre l'exemple des interdistances cercles-triangles. On peut voir que l'on va stocker le nombre 6 correspondant aux 6 traits qui sont de longueur différentes et 3 nombres correspondant au normes des trois plus grands vecteurs rouges ou oranges (voir figure 8.4).

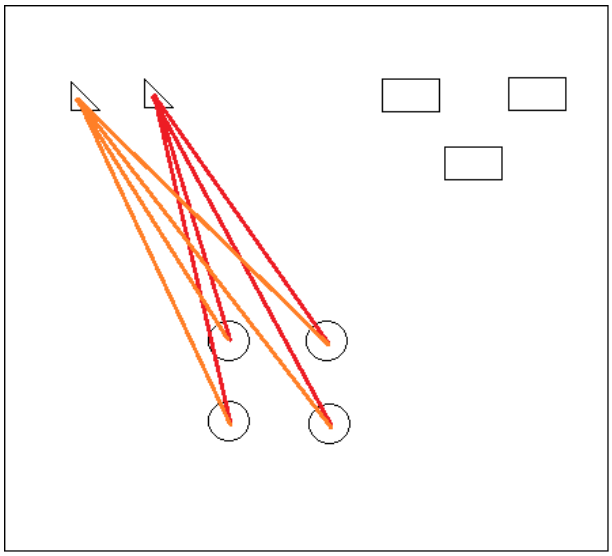

FIGURE 8.4 – Niveau 0.5

On recommence avec les centres d'inertie des clusters comme le montre la figure 8.5. Les distances sont à peu près les mêmes on stockera le nombre 1 et la distance correspondante.

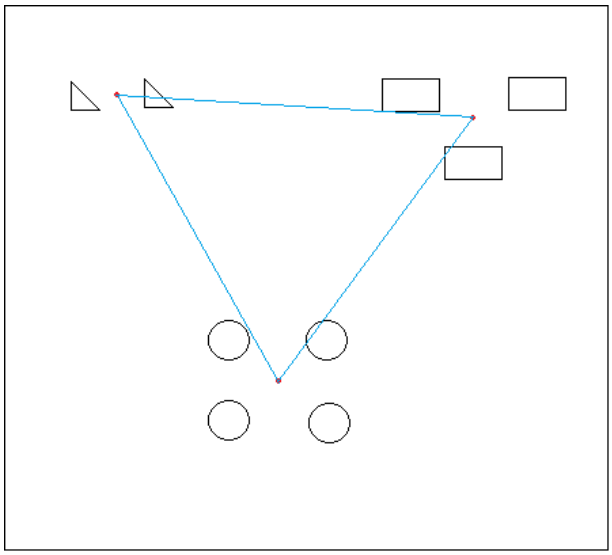

FIGURE 8.5 – Niveau 0.5

Au niveau 1 on remplace chaque cluster par la matrice d'inertie des barycentres des formes qui les composent (voir figure 8.6), et on étudie la nouvelle image comme précédemment. Lorsqu'il n'y a que deux formes dans un cluster, on rajoute quelques points près de barycentres pour que la matrice d'inertie soit inversible.

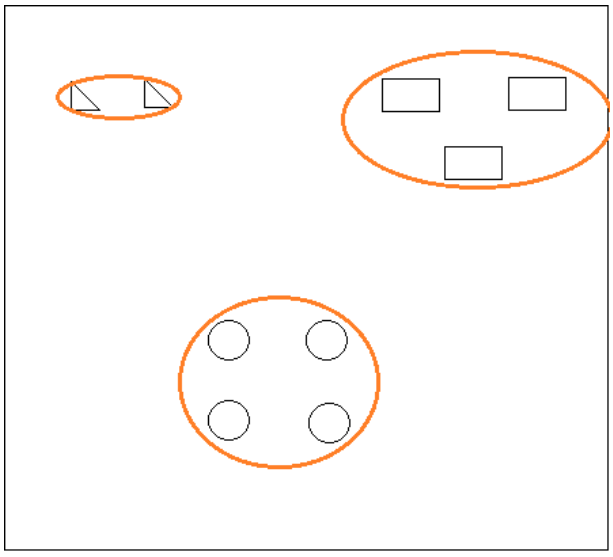

FIGURE 8.6 – Niveau 1

Au niveau 2 on remplace chaque cluster par son centre d'inertie (c'est un peu la continuité du niveau 1 dans le sens où cela revient presque à remplacer la matrice de covariance par son centre), on stocke le nouveau centre d'inertie de cette figure et l'on en prend alors la matrice de covariance, dont on stocke la distance à l'identité sur la demi-droite. (voir figure 8.7)

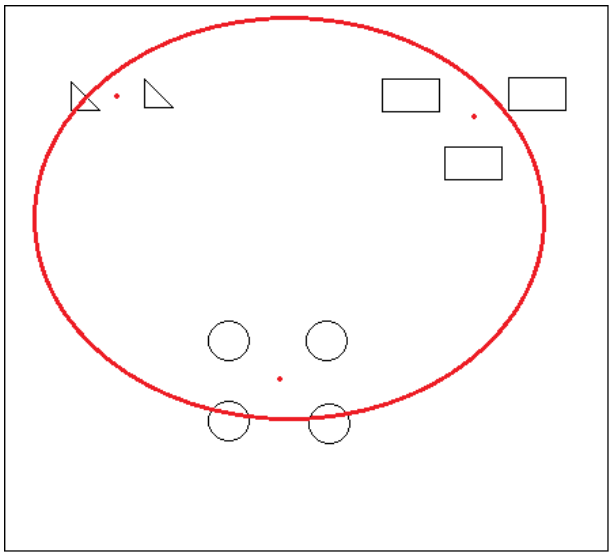

FIGURE 8.7 – Niveau 2

Après avoir étudié les problèmes d'invariances d'un point de vue théorique, décidé d'un algorithme de classification et des caractéristiques à prendre en compte, nous nous sommes attaqué à la programmation. Plusieurs problèmes nous sont apparus, ainsi que des solutions que nous présentons dans la partie suivante.

Chapitre 9

Problèmes d'ordre théoriques : Amélioration du test de Bayes naïf

9.1 Critère de pertinence d'une caractéristique et première idée : Distance de Kullback

9.1.1 Présentation de l'idée

L'idée est d'utiliser une sorte de distance entre 2 mesures de probabilité appelée information de Kullback (ce n'est pas une vraie distance, parce qu'elle ne vérifie pas l'inégalité triangulaire ni la symétrie). Pour chaque caractéristique, on regarde la distance de Kullback entre les 2 probabilités correspondant aux deux classes différentes (0 *et* 1). Plus la distance est grande, plus les deux gaussiennes correspondant aux deux densités des deux classses sont détachées et donc plus la caractéristique correspondante "constitue un point de divergence entre les 2 classes" : c'est à dire plus la caractéristique est pertinente (ou "classante"). Les 2 figures suivantes illustrent un peu l'ordre de grandeur et la manière avec laquelle varie cette distance.

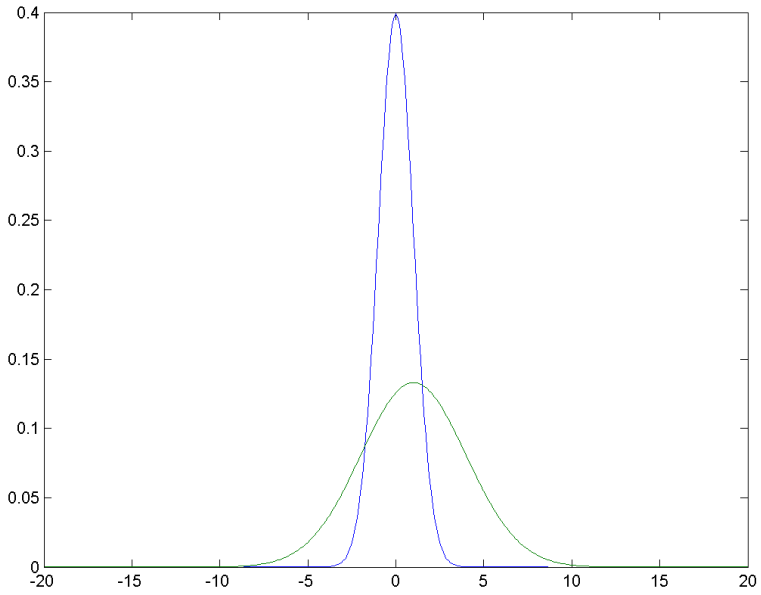

FIGURE 9.1 – gaussiennes "détachées" une $\mathcal{N}(0,1)$ et une $\mathcal{N}(1,3)$ et $K(\mathcal{N}(1,3),\mathcal{N}(0,1)) = 5.04$

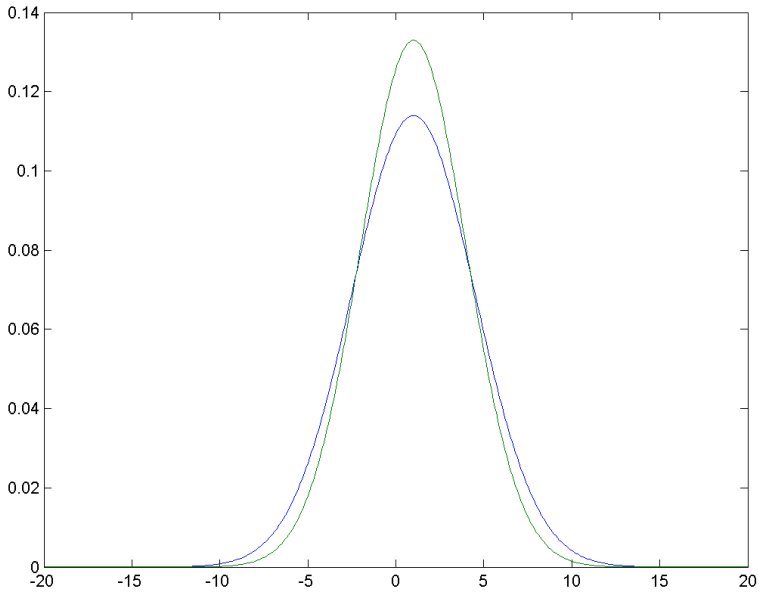

FIGURE 9.2 – gaussiennes " moins détachées" une $\mathcal{N}(1,3)$ et une $\mathcal{N}(1,3.5)$ et $K(\mathcal{N}(1,3.5),\mathcal{N}(1,3)) = 0.2$

9.1.2 Cadre théorique de la distance de Kullback

Si on prend 2 mesures de probabilité définies sur un espace mesurable $(\mathcal{E}, \mathcal{A})$ qu'on note $\mathbf{P}$ *et* $\mathbf{Q}$,et si on suppose que $\mathbf{P}$ est **absolument continue** par rapport à $\mathbf{Q}$,on définit la distance de Kullback de $\mathbf{P}$ par rapport à $\mathbf{Q}$ par :

$$\int_{\mathcal{E}} \log(\frac{\mathbf{dP}}{\mathbf{dQ}})\mathbf{dP} \; note \; d(P||Q)$$

Nous pouvons aussi écrire $d(\mathbf{P}||\mathbf{Q})$ sous la forme :

$$\int_{\mathcal{E}} \log(\frac{\mathbf{dP}}{\mathbf{dQ}})\frac{\mathbf{dP}}{\mathbf{dQ}}\mathbf{dQ}$$

Nous remarquons donc que puisque $x \to xlog(x)$ est convexe sur $\mathbf{R}_*^+$, et en utilisant l'inégalité de Jensen, l'information de Kullback est bien positive et que $d(\mathbf{P}||\mathbf{Q}) = 0$ si et seulement si $\mathbf{P} = \mathbf{Q}$.

Dans notre cas, pour chaque caractéristique, les lois sont des gaussiennes qui ont des variances non nulles (même lorsqu'on trouve une variance nulle (i.e masse de Dirac) dans la phase d'apprentissage,on rajoute un "epsilon" : c'est à dire $2.22 * 10^{-16}$). Donc les deux lois sont toujours absolument continues l'une par rapport à l'autre, et en notant $\mathbf{P}_i^0$ la loi correspondant à la ième caractéristique dans la classe 0 et $\mathbf{P}_i^1$ la loi correspondant à la ième caractéristique de la classe 1, on a :

$$\frac{\mathbf{dP}_i^0}{\mathbf{dP}_i^1}(x) = \frac{f_i^0(x)}{f_i^1(x)}$$

où f_i^0 est une $\mathcal{N}(m_i^0, \sigma_i^0)$ et f_i^1 est une $\mathcal{N}(m_i^1, \sigma_i^1)$.

Et on a donc :

$$d(\mathbf{P}_i^0||\mathbf{P}_i^1) = \int_{\mathcal{R}} log(\frac{f_i^0(x)}{f_i^1(x)})f_i^1(x)\mathbf{dx}$$

c'est à dire

$$\frac{1}{2}(log(\frac{{\sigma_i^1}^2}{{\sigma_i^0}^2}) + \frac{{\sigma_i^0}^2}{{\sigma_i^1}^2} + (\frac{m_i^0 - m_i^1}{\sigma_i^1})^2 - 1).$$

Remarque :On se retrouve des fois dans un cas où $m_i^0 - m_i^1 >> \sigma_i^1$ ce qui donne parfois des distances de Kullback de l'ordre de 10^{30}.

Nous reviendrons à cette remarque dans la prochaine section.

Une fois que nous avons cette formule explicite, nous ne prenons dans le test de rapport de vraisemblance que les n caractéristiques qui ont les n distances de Kullback les plus grandes (cette tâche est faite par la fonction matlab "naivebayes4"). D'après les test effectués sur les scores, une bonne valeur de n est 10. Par exemple, pour 50 images d'apprentissage et 50 de test, on se trompe 9 fois sur 100 (contre 22 sur 100 pour Adaboost). Remarque : Adaboost fait les tests sur 1000.

9.1.3 Les limites de cette approche

Le premier inconvénient est que cette approche utilise des mesures de probabilité sur $(\mathcal{R}, \mathcal{B}(\mathcal{R}))$ donc nous sommes encore obligés d'attribuer le nombre 0 à la caractéristique de l'image quand celle-ci n'est pas définie. Une solution assez naturelle proposée à ce problème fera l'objet de la prochaine section...

Le second inconvénient est que nous nous retrouvons dans des cas (assez rares) où une caractéristique peut avoir une distance de Kullback très grande alors qu'elle n'est pas du tout pertinente à cause de petits objets qui peuvent être pris en compte au moment de la lecture d'une image et de l'extraction des composantes connexes présentes dans l'image.

En effet, cela arrive quand la caractéristique devrait avoir la même distribution pour les 2 classes et que cette distribution est pratiquement une **dirac***, mais qu'en pratique nous nous retrouvons avec une gaussienne proche d'une dirac pour une classe et une gaussienne plus étalée pour l'autre classe à cause de petits objets pris en compte sur une image appartenant à cette classe, et qui font que pour cette image là, le nombre correspondant à la caractéristique en question soit assez loin de la moyenne. Ceci donne une distance de kullback assez grande.

Un exemple typique est celui rencontré dans le problème 20 avec la caractéristique la plus simple, c'est à dire le nombre de composantes connexes. On devrait avoir pratiquement une dirac en 2 dans les deux classes, mais à cause d'images dans la classe 1 comme celle ci-dessous on se retrouve avec une gaussienne centrée en 2 mais étalée pour la distribution correspondant à la classe 1.

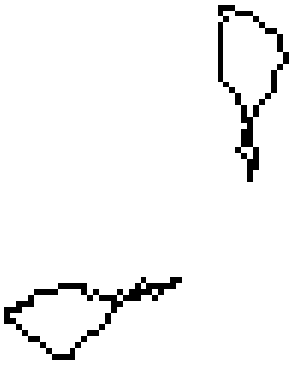

FIGURE 9.3 – Le nombre de composantes connexes de cette image pris en compte par la fonction matlab est **7** au lieu de 2, ce qui étale la distribution

9.2 Une nouvelle mesure de probabilité sur l'ensemble R $\bigcup \{*\}$

9.2.1 Présentation de l'idée

Comme nous l'avons evoqué précédemment, nous nous retrouvons parfois dans des cas où nous sommes obligés d'attribuer à la caractéristique un nombre (0) même quand celle-ci n'est pas définie puisque nous utilisons des mesures de probabilités sur $(\mathcal{R}, \mathcal{B}(\mathcal{R}))$.

Nous introduisons alors un objet $\{*\}$(sur Matlab on utilise "Inf") qu'on attribue à la case réservée à la caractéristique (au lieu de zéro) si celle-ci n'est pas définie.

Nous supposons alors que pour une caractéristique j fixée, la valeur que prend cette caractéristique si on se place dans la classe 0 suit une loi $p_j^0\delta_* + (1-p_j^0)\mathcal{N}(m_j^0, \sigma_j^0)$ où (en reprenant les notations de la section apprentissage de base) :

$$m_j^0 = \frac{1}{n_{classe0\neq *}} \sum_{i \ tq \ x_i \neq *}^{n_{classe0}} x_i$$

$$\sigma_j^{0\,2} = \frac{1}{n_{classe0\neq *} - 1} \sum_{i \ tq \ x_i \neq *}^{n_{classe0}} (x_i - m)^2$$

et enfin :

$$p_j^0 = \frac{1}{n_{classe0=*} + 2}(1 + \sum_{i \ tq \ x_i = *}^{n_{classe0}} \mathbf{1})$$

où $n_{classe0\neq *}$ désigne le nombre de fois, dans la classe 0 où la caractéristique admet un réel comme valeur et $n_{classe0=*}$ désigne le nombre de fois, dans la classe 0 où la caractéristique admet $*$ comme "valeur" et x_i étant bien sûr la "valeur" de la cractéristique correspondant à l'ième image de la classe 0 dans la base d'apprentissage.

Notons également que lorsque $n_{classe0\neq *} = 0$, on impose une gaussienne de moyenne 0 et de variance 10^{-10}.

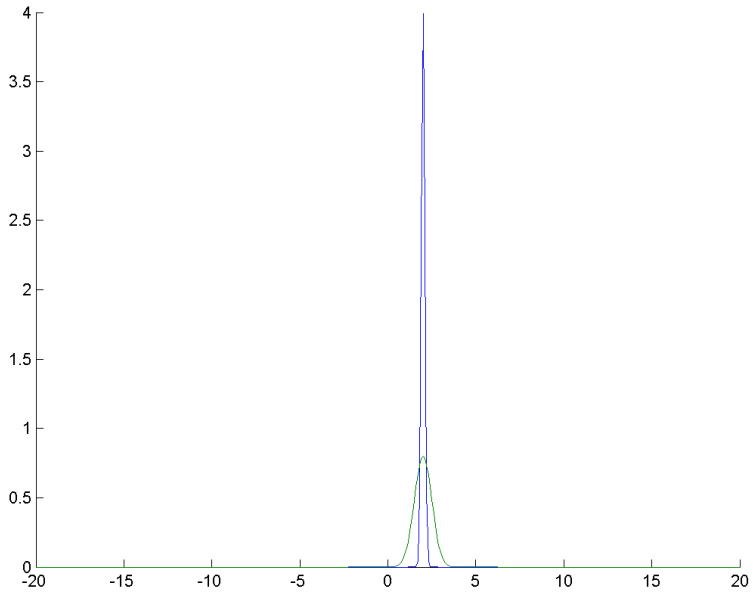

FIGURE 9.4 – Nous nous retrouvons dans le cas suivant :la distance de Kullback est très grande

Pour résoudre ce problème,nous avons créé la fonction "seuilaires" qui élimine toutes les composantes connexes d'une image dont la taille est 30 fois plus petite que la plus grande composante connexe de l'image.

En réalité, le but est d'éviter le cas (rare) où nous nous retrouvons avec que des "cases vides" dans la phase d'apprentissage (pour la caractéristique en question) et avoir ensuite une image sur laquelle on fait le test et qui contient un nombre et non "case vide" pour cette même caractéristique. Donc si on impose pas une moyenne et une variance au départ le rapport de vraisemblance ne serait pas bien défini dans ce cas particulier.

On procède de même pour la classe 1.

Remarquons également que nous avons fait le choix d'imposer aux p_i^0 et p_i^1 une valeur appartenant à $(0,1)$ pour avoir la continuité absolue des deux mesures $p_j^0\delta_* + (1-p_j^0)\mathcal{N}(m_j^0,\sigma_j^0)$ et $p_j^1\delta_* + (1-p_j^1)\mathcal{N}(m_j^1,\sigma_j^1)$ l'une par rapport à l'autre.

On a donc

$$\frac{\mathbf{dP}_j^0}{\mathbf{dP}_j^1}(x) = \frac{p_j^0\delta_*(x) + (1-p_j^0)f_j^0(x)}{p_j^1\delta_*(x) + (1-p_j^1)f_j^1(x)}.$$

où $f_i^0(x)$ désigne une densité $\mathcal{N}(m_j^0,\sigma_j^0)$ si $x \in \mathbf{R}$ et égale à 0 sinon.

On notera le rapport

$$\frac{p_j^0\delta_*(x) + (1-p_j^0)f_i^0(x)}{p_j^1\delta_*(x) + (1-p_j^1)f_i^1(x)} = \frac{g_i^0(x)}{g_i^1(x)}$$

où les g_i désignent donc les nouvelles densités des nouvelles mesures de probabilité par rapport à **la nouvelle mesure de référence** sur $(\mathcal{R}\bigcup\{*\},\mathcal{A})$ où $\mathcal{A}$ désigne les ensembles (borélien de $\mathbf{R}$)$\bigcup(\emptyset\ ou\ \{*\})$. qui est

$$\mu = \lambda + \delta.$$

où λ désigne la mesure qui est egale à la mesure de Lebesgue de la partie borélienne de $\mathbf{R}$ de l'ensemble, et δ désigne la mesure de dirac en $*$.

9.2.2 La nouvelle distance de Kullback et le nouveau rapport de vraisemblance

Les mesures de probabilité ont changé dans cette nouvelle approche, donc la distance de Kullback évoquée précedement change aussi et, pour chaque caractéristique, nous avons :

$$d(P^0||P^1) = \int_{\mathbf{R}\cup\{*\}} \log(\frac{g_0(x)}{g_1(x)}) g_0(x) d\mu(x)$$

où les g désignent les densités évoquées dans la section précedente (par rapport à la mesure μ). Nous obtenons alors :

$$d(P^0||P^1) = p_0 \log(\frac{p_0}{p_1}) + (1-p_0)\log(\frac{1-p_0}{1-p_1}) + (1-p_0) \underbrace{\frac{1}{2}(\log(\frac{{\sigma_i^1}^2}{{\sigma_i^0}^2}) + \frac{{\sigma_i^0}^2}{{\sigma_i^1}^2} + (\frac{m_i^0 - m_i^1}{\sigma_i^1})^2 - 1)}_{distance\ entre\ deux\ gaussiennes}$$

Cette nouvelle distance est calculée grâce à la fonction Matlab "Kullback1".
En prenant donc les n caractéristiques correspondant aux n caractéristiques les plus classantes, le nouveau rapport de vraisemblance est donc :

$$\prod_{i=1}^{n} \frac{g_i^1(x)}{g_i^0(x)}$$

c'est à dire

$$\prod_{i=1}^{n} \frac{p_i^1 \delta_*(x) + (1-p_i^1) f_i^1(x)}{p_i^0 \delta_*(x) + (1-p_j^0) f_i^0(x)} \ .$$

9.2.3 Limites de cette approche

En prenant comme pour notre première approche les 10 meilleures caractéristiques, nous obtenons un score d'environ 23 erreurs sur 100 en moyenne (sur 50 images pour la base d'apprentissage et 50 images à tester) avec des problèmes ou nous faisons un score de 50 sur 100 (comme le problème numéro 8), ce qui revient à simuler un pile ou face.

Et le problème n'est pas résolu en changeant le nombre de caractéristiques qu'on décide de prendre.

En réalité, ce qui se passe c'est que comme nous l'avons signalé précédement, la partie "distance entre deux gaussiennes" dans la nouvelle distance de Kullback peut être très grande (elle peut aller jusqu'à 10^{33}). On se retrouve dans le cas où cette partie est la partie prédominante dans la distance de Kullback, même lorsque le coefficient $(1-p_j^0)$ est assez petit (de l'orde de 2/100 par exemple), ce qui donne parfois une caractéristique qui n'est pas pertinente une distance de Kullback qui est très grande.

En effet, on se retrouve par exemple dans le problème numéro 8 avec une caractéristique où :

- p_0 est égale à 86/100
- p_1 est égale à 98/100
- La partie "distance entre deux gaussiennes" de l'ordre de 10^{23}

Donc la caractéristique en question admet une distance de Kullback qui est très grande, mais notons que dans la grande majorité des images, et dans les deux classes, la case réservée à cette caractéristique est **vide**. Donc elle n'est pas pertinente.

Il faut donc trouver un nouveau critère de pertinence puisque la distance de Kullback nous donne avec cette nouvelle approche, un classement qui n'est pas toujours représentatif de la pertinence des caractéristiques.

9.2.4 Un nouveau critère de pertinence : l'erreur de Bayes

L'idée est la suivante :

Nous fixons une caractéristique, nous observons la "valeur" de cette caractéristique pour une image dont on ne connait pas la classe. En gardant les mêmes notations de la section précédente, l'observation de la valeur de la caractéristique en question est alors celle d'une variable aléatoire $\mathbf{X}$ à valeurs dans $\mathbf{R} \cup \{*\}$ et qui admet une loi dont la densité est $\frac{1}{2}(g_0(x) + g_1(x))$ par rapport à la mesure de référence μ.

Le test de rapport de vraisemblance est alors effectué en n'utilisant que cette caractéristique. En d'autre termes, et en notant $d(X)$ la décision de mettre l'image dans la classe 1 ou 0, on a :

- Si $g_1(X) \geq g_0(X)$ alors $d(X) = 1$
- Sinon $d(X) = 0$

Notons C la variable aléatoire qui désigne la classe de l'image (0 ou 1)
Le critère qui remplace la distance de Kullback est alors l'espérance suivante :

$$\mathbf{P}(d(X) \neq C).$$

Plus cette quantité est petite, plus la caractéristique est pertinente.On peut réécrire cette espérance sous la forme :

$$\mathbf{E}(\mathbf{P}(d(X) \neq C|X)),$$

ou encore

$$\mathbf{E}(\mathbf{E}(\mathbf{1}_{d(X)\neq C}|X))$$

avec

$$d(X) = argmax_{c\in\{0,1\}}\mathbf{P}(C = c|X).$$

Décomposons encore plus cette expresssion :

$$\mathbf{E}(\mathbf{E}(\mathbf{1}_{d(X)\neq C}|X)) = \mathbf{E}(\mathbf{E}(\mathbf{1}_{d(X)=0}\mathbf{1}_{C=1}|X) + \mathbf{E}(\mathbf{1}_{d(X)=1}\mathbf{1}_{C=0}|X)).$$

Comme $\mathbf{1}_{d(X)=0}$ et $\mathbf{1}_{d(X)=1}$ sont mesurables par rapport à X, nous obtenons :

$$\mathbf{E}(\mathbf{E}(\mathbf{1}_{d(X)\neq C}|X)) = \mathbf{E}\left(\mathbf{1}_{d(X)=0}\mathbf{E}(\mathbf{1}_{C=1}|X) + \mathbf{1}_{d(X)=1}\mathbf{E}(\mathbf{1}_{C=0}|X)\right)$$

c'est à dire

$$\mathbf{E}(\mathbf{E}(\mathbf{1}_{d(X)\neq C}|X)) = \mathbf{E}(\mathbf{1}_{g_0(X)\geq g_0(X)}\mathbf{E}(\mathbf{1}_{C=1}|X) + \mathbf{1}_{g_1(X)>g_0(X)}\mathbf{E}(\mathbf{1}_{C=0}|X)).$$

Enfin :

$$\mathbf{E}(\mathbf{E}(\mathbf{1}_{d(X)\neq C}|X)) = \mathbf{E}(\mathbf{1}_{g_0(X)\geq g_0(X)}\mathbf{P}(C = 1|X) + \mathbf{1}_{g_1(X)>g_0(X)}\mathbf{P}(C = 0|X))$$

Maintenant, en utilisant la formule de Bayes pour $\mathbf{P}(C = 0|X)$ et $\mathbf{P}(C = 1|X)$., nous obtenons une espérance égale à :

$$\frac{1}{2}\int_{\mathbf{R}\cup\{*\}}(f_0(x) \wedge f_1(x))d\mu(x),$$

Ou encore

$$\frac{1}{2}(p_0 \wedge p_1) + \frac{1}{2}\int_{\mathbf{R}} (1-p_0)f_0(x) \wedge (1-p_1)f_1(x)\mathbf{dx}.$$

Ici f_0 et f_1 sont les gaussiennes de la caractéristique que nous avons fixée au départ.

Nous avons donc une formule théorique pour l'espérance de l'erreur Bayesienne. Mais l'intégrale que nous obtenons n'est pas simple à calculer. Nous utilisons alors une autre manière d'exprimer notre espérance de départ (afin de pouvoir l'estimer numériquement) laquelle était

$$\mathbf{P}(d(X) \neq C|X)$$

qu'on peut réécrire sous la forme

$$\mathbf{P}(d(X) = 0, C = 1) + \mathbf{P}(d(X) = 1, C = 0)$$

ou encore (puisqu'on suppose que $\mathbf{P}(C = 1) = \mathbf{P}(C = 0) = \frac{1}{2}$)

$$\frac{1}{2}(\mathbf{E}(\mathbf{1}_{f_0(X)\geq f_1(X)}|C = 1) + \mathbf{E}(\mathbf{1}_{f_0(X)<f_1(X)}|C = 0)).$$

Notons que lorsqu'on se place sur l'évènement $C = 1$,$\mathbf{E}(\mathbf{1}_{f_0(X)\geq f_1(X)}|C = 1)$ n'est autre que $\int \mathbf{1}_{f_0(x)\geq f_1(x)} d\mu_1(x)$ où $\mu_1 = p_1\delta_* + (1-p_1)f_1$ la loi de X sachant que $C = 1$.
De même, lorsqu'on se place sur l'évènement $C = 0$,$\mathbf{E}(\mathbf{1}_{f_0(X)<f_1(X)}|C = 0)$ n'est autre que $\int \mathbf{1}_{f_0(x)<f_1(x)} d\mu_0(x)$ où $\mu_0 = p_0\delta_* + (1-p_0)f_0$ la loi de X sachant que $C = 0$.
Pour estimer $\int \mathbf{1}_{f_0(x)\geq f_1(x)} d\mu_1(x)$, on simule un vecteur de taille N
où chacune de ses cases i est l'observation d'une variable aléatoire $Y_i = \mathbf{1}_{f_0(X_i)\geq f_1(X_i)}$ avec
$X_i \sim p_1\delta_* + (1-p_1)\mathcal{N}(m_1, \sigma_1)$ et les X_i sont **i.i.d**.
On compte alors le nombre d'apparitions de 1 dans le vecteur et on divise par sa taille N du tableau.
(Cette tâche est effectuée dans la fonction matlab "beest") où nous avons estimé (avec $N = 1000$) juste le nombre d'erreurs (donc sans diviser par la taille du tableau).

Notons que nous avons utilisé la fonction de matlab **randn** pour simuler la "partie gaussienne des variables aléatoires", et que pour simuler la partie "loi discrète", nous avons utilisé l'idée suivante :
Pour simuler une variable aléatoire X à valeurs dans $\{x_1, x_2, ..., x_n\}$ tel que :

$$\forall i \in [1; n], \mathbf{P}(X = x_i) = p_i$$

Il suffit de disposer d'une variable **U** qui suit une uniforme sur $[0, 1]$ (Matlab simule cette loi grâce à rand) et de construire une variable aléatoire **X** de la manière suivante :

$$X(\omega) = x_i \; si \; \mathbf{U}(\omega) \in (pour \; i > 0) \left[\sum_{j=1}^{i-1} p_j, \sum_{j=1}^{i} p_j\right) et \; (pour \; i = 0) \, [0, p_0) \, .$$

Notons que nous faisons exactement la même chose (également dans la fonction "beest" de matlab) pour estimer

$$\int \mathbf{1}_{f_0(x)<f_1(x)} d\mu_0(x)$$

Grâce à cette nouvelle approche, c'est à dire en utilisant :

$$\mathbf{P}(d(X) \neq C)$$

comme critère de pertinence d'une caractéristique, nous nous trompons environ 7 fois sur 100 en moyenne avec 50 images pour l'apprentissage de base et 50 images pour le test.

Nous allons maintenant chercher des informations sur la précision des estimateurs qu'on a utilisés pour approcher les deux quantités :
$\mathbf{E}(\mathbf{1}_{f_0(X)<f_1(X)}|C = 0)$ *et* $\mathbf{E}(\mathbf{1}_{f_0(X)\geq f_1(X)}|C = 1)$

9.2.5 Intervalle de confiance pour l'estimateur

Le cadre est le suivant :
Nous disposons d'un échantillon $(X_1, ..., X_N)$ (dans notre cas $N = 1000$) de variables aléatoires **i.i.d** suivant une **Bernoulli** de paramètre p.
nous voulons alors estimer l'espérance de cette Bernoulli qui n'est autre que p.

D'après notre fonction matlab "beest" évoquée dans la section précédente, notre estimateur est : $\overline{X_N} = \frac{1}{N}\sum_{i=1}^{N} X_i$

Notons que : $\forall i \in [1; N]$ la variable aléatoire $Y_i = \frac{X_i - p}{N}$ admet les propriétés suivantes :

1. $\mathbf{E}(Y_i) = 0$
2. $Y_i \in \left[\frac{-p}{N}, \frac{1-p}{N}\right]$

En utilisant l'inégalité de Hoeffding, pour un $\epsilon > 0$, nous avons alors :

$$\mathbf{P}(|\sum_{i=1}^{N} Y_i| > \epsilon) \leq 2e^{-2N\epsilon^2}$$

c'est à dire :

$$\mathbf{P}(|\overline{X_N} - p| > \epsilon) \leq 2e^{-2N\epsilon^2}$$

i.e

$$\mathbf{P}(|\overline{X_N} - p| \leq \epsilon) \geq 1 - 2e^{-2N\epsilon^2}$$

Ce qui nous fournit un encadrement de p :

$$p \in \left[\overline{X_N} - \epsilon, \overline{X_N} + \epsilon\right]$$

avec une probabilité supérieure à $1 - 2e^{-2N\epsilon^2}$.
Par exemple pour $\epsilon = 0.05$ la probabilité pour p d'être dans l'intervalle de confiance est supérieure à 99/100. Remarquons que nous pouvons également construire un intervalle de confiance **asymptotique** avec notre estimateur puisque, d'après le T.C.L :

$$\sqrt{n}(\overline{X_n} - p) \overset{\mathcal{L}oi}{\to} \mathcal{N}(0, p(1-p))$$

En utilisant alors la δ-méthode avec :

$$f(x) = 2Arcsin(\sqrt{(x)})$$

nous obtenons une nouvelle convergence en loi :

$$2\sqrt{n}(Arcsin(\overline{X_n}) - Arcsin(p)) \overset{\mathcal{L}oi}{\to} \mathcal{N}(0, 1)$$

En utilisant alors la caractérisation de la convergence en loi avec la convergence des fonctions de répartitions en les points de continuité de la fonction de répartition limite, nous obtenons :

$$\mathbf{P}(|2\sqrt{n}(Arcsin(\sqrt{\overline{X_n}}) - Arcsin(\sqrt{p}))| \leq \alpha) \overset{n\to\infty}{\to} \mathbf{P}(|\mathcal{N}(0,1)| \leq \alpha)$$

Ce qui nous fournit bien un intervalle de confiance asymptotique :

$$\left[sin^2\left(Arcsin(\sqrt{\overline{X_n}} - \frac{\alpha}{2\sqrt{n}}\right), sin^2\left(Arcsin(\sqrt{\overline{X_n}} + \frac{\alpha}{2\sqrt{n}}\right)\right]$$

Nous avons donc un intervalle de confiance asymptotique qui a pratiquement la même longueur que le premier intervalle de confiance (construit grâce à l'inégalité de Hoeffding).

Chapitre 10

Problèmes d'ordres pratiques et techniques

10.1 Programmation des caractéristiques

Nous avons voulu dans une large mesure suivre ces idées de choix de caractéristiques qui, à notre avis, sont naturelles dans le sens où c'est pour nous la manière dont nous procéderions pour classer les images, et permettent de décrire efficacement la géométrie et l'organisation des formes d'une image.
En fait, généralement le calcul des caractéristiques au niveau 0 permet d'être performant sur beaucoup de problèmes avec les 4 relations de similitudes, d'aires, de translation et de contact. Néanmoins les niveaux 1 et 2 ont été programmés pour les trois premières relations d'équivalences.
Le programme qui donne les résultats finaux donnés au chapitre 1 ne prend pas en compte les invariances par rotation et homothéties du fait de leur inutilités sur les 13 problèmes du SVRT, et qui de plus augmentent la vitesse d'execution du programme et donc ne pas prendre en compte ces invariances a présenté pour nous un avantage, bien qu'elles aient été programmées. Ainsi nous avons pour le moment un peu adaptées les caractéristiques aux 13 problèmes du SVRT, et certain compléments devraient certainement être programmées si l'on souhaite être aussi performant sur les derniers problèmes dévoilés le 31 août.

10.2 Choix des formes sur lesquelles on calcule les caractéristiques

Rappelons que nous n'étudions que certaines figures séléctionnés par la fonction "seuilaires" qui élimine toutes les composantes connexes d'une image dont la taille est 30 fois plus petite que la plus grande composante connexe de l'image. Cette fonction n'est pas optimale et pourrait être améliorée. En plus de cela, nous choisissons de ne travailler que sur les formes de matrice de covariance de déterminant assez grand pour pouvoir effectuer les calculs nécéssaires (notamment la distance par similitudes où l'on a besoin d'extraire l'inverse d'une racine carrés matricielle).

10.3 Choix de seuils

Généralement dans le calcul des caractéristiques nous disposons d'une distance entre deux figures. En théorie deux figures sont dans le même cluster si la distance est nulle. En pratique plusieurs causes sont à l'origine du fait que cela n'est pas tout à fait vrai.
Tout d'abord nous ne prenons pas en argument l'image mais sa matrice de covariance et cela induit forcément des erreurs, si petites soient-elles. Ensuite si l'on omet ce détail rien que la résolution de l'image fait que deux formes qui à l'oeil sont semblables ne le sont pas forcément aux yeux de la machine qui effectue les calculs. De plus si l'on regarde par exemple le problème 21, on remarque que les formes ont l'air semblables mais si l'on y regarde de plus près, on remarque que ce n'est pas tout à fait le cas (voir figure 10.1). Enfin, les tests effectués avec un seuil de distance par similitude trop petit ne donnait

pas de bons résultats sur les problèmes 20 et 21 sur lesquels nous avons décidés de baser les seuils de similitude, qui est le seul a avoir vraiment posé des difficultés.

Nous avons alors représentés sur des histogrammes les distances par similitudes entre formes comme le montrent les figures 10.2 et 10.3. En rouge sont les distances des formes qui sont semblables et en bleu les distances entre formes qui ne le sont pas. Nous avons alors choisi le seuil à 0.07. Pour ces deux problèmes nous avons obtenus des résultats corrects.

Pour les translations et aires nous n'avons pas eu de problèmes, un seuil assez haut pour la translation (0.2) et pour les aires (100 pixels) donnent de bons résultats.

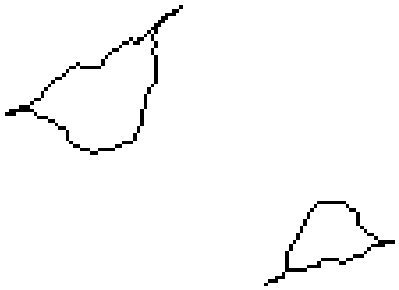

FIGURE 10.1 – Exemple du problème 21

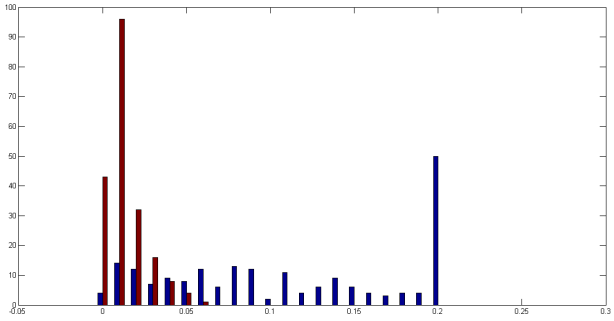

FIGURE 10.2 – histogramme problème 20

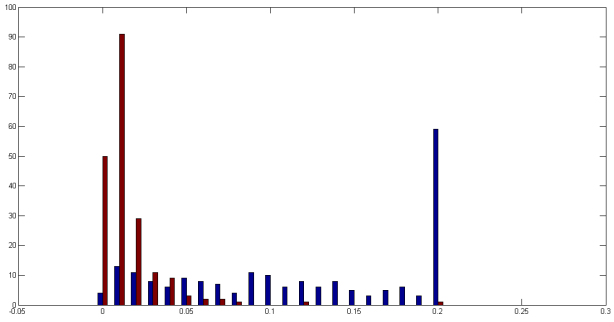

FIGURE 10.3 – histogramme problème 21

10.4 Petits détails concernant la matrice de covariance

Premièrement, pour que le niveau 1 soit pris en compte pour les clusters composés de deux formes, nous avons juste rajouté un point en plus à côté d'un des deux barycentre.
Deuxièmement, en regardant le problème 18 nous nous sommes aperçus que quelques images avaient

systématiquement des vecteurs où toutes les caractéristiques valaient 0 (ce que l'on peut qualifier d'"image non lue"), et cela est dû au fait que le programme qui extrait les composante connexes et nous donne accès à la matrice de covariance d'une composante connexe prend en fait la matrice de covariance de l'intérieur de la composante. Comme on peut le voir sur la figure 10.4, les composantes connexes n'ont que deux pixels et donc la matrice de covariance est non inversible. Une amélioration à apporter serait de calculer la matrice de covariance de l'intérieur avec son contour. Du fait de la faible quantité de telles images et des résultats obtenus, nous ne nous sommes pas focalisés sur ce problème et les images non lues ont été dans un premier temps rejetés de la base d'apprentissage et de test. Dans un deuxième temps le vecteurs des caractéristiques a été rempli de symboles "*".

FIGURE 10.4 – Image du problème 18

Chapitre 11

Résultats finaux

11.1 Aspect programmation

11.1.1 Importance de la patie programmation dans le stage

L'aspect programmation est la partie qui nous a demandé le plus de temps durant ce stage, elle nous a en effet occupé pendant toute la deuxième partie de ce stage.
C'est la partie qui nous a permis de confronter nos choix dans l'approche du problème aux résultats pratiques (c'est à dire aux scores) et donc de changer la manière d'aborder le problème quand cela était nécessaire.

11.1.2 Présentation des données

Tout d'abord, tous les codes ont été écrits sur Matlab, qui présentait l'avantage d'être maîtrisé par tous les membres du groupe. En utilisant la fonction Imread de Matlab, les images sont données sous forme de matrices 128×128 avec un zéro si on a du blanc sur l'image et 1 sinon (en réalité nous faisons l'hypothèse que même les problèmes cachés ne prennent pas en compte les couleurs comme critère de classification).

11.1.3 Structuration du code

Tout d'abord, notons que les détails concernant chaque fonction sont fournis dans les commentaires présents dans les fichiers Matlab (en annexe). Nous allons donc évoquer la structuration globale du code.

La première étape est d'écrire une fonction qui nous fournit, pour chaque image, les informations dont nous avons besoin et que nous utiliserons dans la construction des caractéristiques (par exemple : le nombre de composantes connexes qui n'ont pas une taille "négligeable", les coordonnées correspondant à ces composantes connexes, les matrices d'inertie des composantes, leurs barycentres, etc...).
Cette tache est faite par la fonction Matlab que nous avons appelée ccomp_2 (et ses sous programmes).

La seconde étape est d'extraire les caractéristiques. Cette tâche est faite par study1, classesI2_3Inf et ses sous-programmes caractsI2_Inf, caracttInf, caractaInf.

Une fois que nous avons construit un tableau où chaque ligne correspond à une image et chaque colonne à une caractéristique (la première étant la classe 0 ou 1), vient alors la phase d'apprentissage et de test grâce à la fonction naivebayes62 (et ses sous programmes). Cette fonction prend en argument le tableau d'apprentissage et de test.

Enfin, le traitemant final des images et l'affichage des scores est fait par la fonction Matlab Studyall2 (et ses sous programmes) qui prend en argument le nombre d'image de la base d'apprentissage et le nombre d'image sur lesquels on fait le test.
Alternativement, si l'on souhaite effectuer les tests sur les mêmes images que celles de l'apprentisssage, on utilise studyall et naivebayes6, qui fournissent les résultats finaux "bruts" présentés ci-après.

11.2 Résultats "bruts"

On redonne sur la figure 11.1 les résultats finaux donnés en introduction, ils ont été calculés sur une base d'apprentissage de 400 images et une base de test de 400 images qui s'avèrent être les même images. Les résultats d'AdaBoost sont eux sur une base d'apprentissage de 1000 images et la même base de test.

```
Probleme 1: adaboost score 0.467 our score 0
Probleme 2: adaboost score 0.054 our score 0.0275
Probleme 3: adaboost score 0.058 our score 0.1125
Probleme 5: adaboost score 0.427 our score 0
Probleme 6: adaboost score 0.269 our score 0.0075
Probleme 8: adaboost score 0.083 our score 0.08
Probleme 11: adaboost score 0.104 our score 0.005
Probleme 12: adaboost score 0.198 our score 0
Probleme 13: adaboost score 0.399 our score 0.035
Probleme 17: adaboost score 0.388 our score 0.1
Probleme 18: adaboost score 0.067 our score 0.065
Probleme 20: adaboost score 0.482 our score 0.0825
Probleme 21: adaboost score 0.501 our score 0.1375

ans =

    0.0502
```

FIGURE 11.1 – Notre score et celui d'AdaBoost

Le dernier résultat donné est notre erreur moyenne. Le score moyen d'AdaBoost est 0.310 sur une base d'apprentissage de 1000, et environ 0.2 pour une base d'apprentissage de 10 000.

Ces résultats sont satisfaisant, néanmoins suscitent quelques remarques. En effet on s'attendait à ce que tous les problèmes soient résolus parfaitement ou presque (c'est-à-dire pour nous en dessous de 0.1) puisque parmis les caractéristiques codées, au moins une doit être classante. Mais dans le test de Bayes on prend en compte les 10 caractéristiques les plus classantes, ce qui peut induire des erreurs qui s'ajoutent aux erreurs de la méthode (les caractéristiques ne sont pas indépendantes ce qui est le cadre du test de Bayes). De plus le fait de mettre des "*" à la place d'un manque de données dans le vecteur des caractéristiques qui peut être dû à une erreur de notre programme (nous choisissons les figures que nous prenons en considération pour le calcul des caractéristiques et dans certains cas il se peut qu'il nous en manque où que nous prenons des figures que nous ne devrions pas considérer) peut introduire des erreurs de décision. Enfin, les problèmes 20 et 21 sont légèrement décevant pour nous puisque nous avions développés des outils mathématiques adaptées à la reconnaissance de formes invariantes par similitude. Cela étant dit, les résultats sont plutôt bon et prouve une certaine robustesse du programme face aux erreurs de toutes sortes (sur les seuils, les formes à prendre en compte, les erreurs introduites par le test de Bayes, etc...).

11.3 Apprentissage

On présente ici les résultats obtenus en modifiant la base d'apprentissage : sur les graphiques des figures 11.2 et 11.3 sont donnés les résultats de notre programme sur les différents problèmes en fonction de nombre d'exemples d'apprentissage par classes en abscisses. Les tests sont effectués sur une base de 100 images par classes différente de la base d'apprentissage.

On constate un apprentissage assez rapide, et des résultats assez bon assez rapidement.

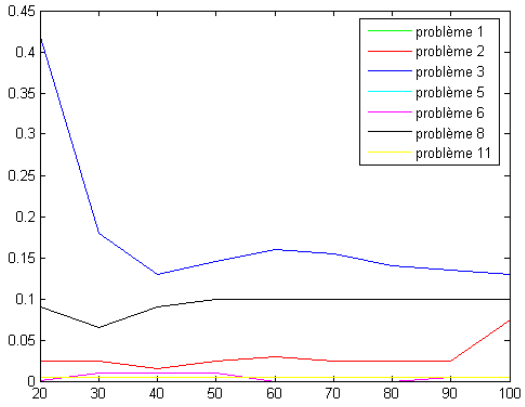

FIGURE 11.2 – Résultats en fonction de la base d'apprentissage sur les problèmes 1,2,3,5,6,8,11

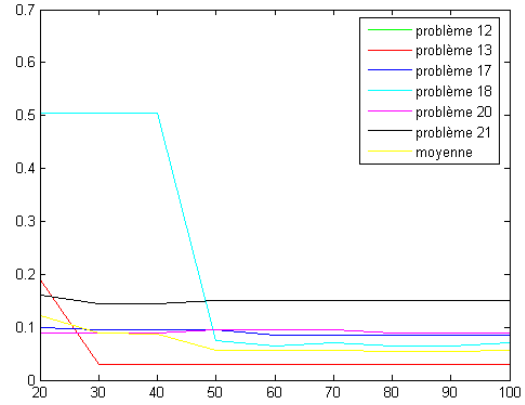

FIGURE 11.3 – Résultats en fonction de la base d'apprentissage sur les problèmes 12,13,17,18,20,21 et l'évolution de la moyenne

11.4 Comparaison avec Adaboost, avec des Humains

Sur le graphique de la figure 11.4 on compare les résultats d'AdaBoost avec les nôtres. On remarque que notre programme a de bien meilleurs résultats, même si sur un nombre très grand nombre d'exemples AdaBoost finira par nous battre. Sur le graphique de la figure 11.5 on compare nos résultats avec le nombre d'essais moyen qu'il a fallu à des sujets humains pour comprendre la règle permettant de classer les images. On voit que si pour certains problèmes là où les Humains comprennent vite notre programme est efficace, l'inverse se produit aussi. De même si notre programme est efficace pour certains problèmes l'Humain l'est moins (comparé à lui-même), l'inverse se produit aussi.

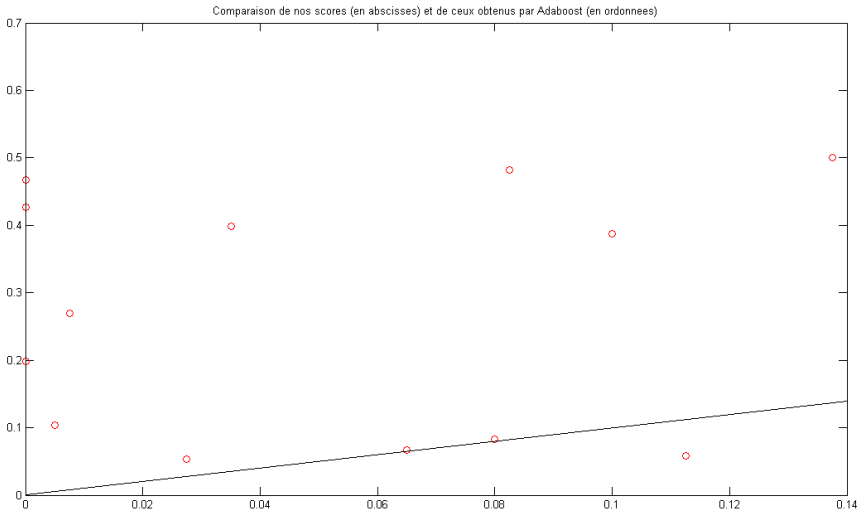

FIGURE 11.4 – Comparaison avec AdaBoost

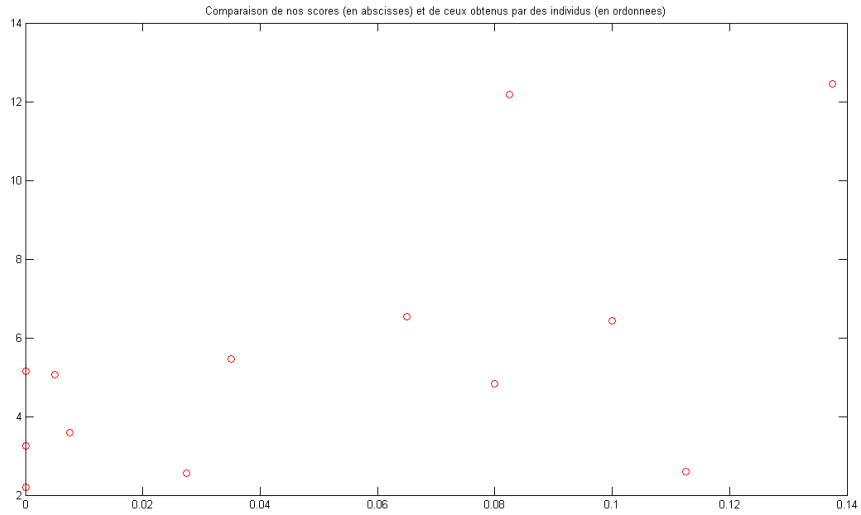

FIGURE 11.5 – Comparaison avec des Humains

11.5 Conclusion

Au cours de ce stage, nous avons donc dû travailler dans des livres des notions de géométrie différentielle de niveau M1 pour se donner les outils nécessaires pour traiter l'invariance. Le calcul des distances invariantes a d'ailleurs été un bel exercice. Kamel a pu utiliser ses connaissances acquises en M1 de statistiques pour améliorer le classifieur de Bayes. Enfin ce stage a été pour nous une première expérience de travail de recherche en groupe qui a pour nous été bénéfique, qui s'est conclue sur des résultats dont nous sommes assez satisfaits. Nous serions curieux de voir ce que donnerait notre programme sur les 10 derniers problèmes.

Nous remercions ici M. Trouvé pour nous avoir guidé tout au long de ce stage et nous avoir aidé à mettre en place des connaissances mathématiques, et le programme de classification. Nous remercions aussi M. Morel pour l'organisation des stages. Enfin nous remercions le CMLA pour nous avoir acceuilli pendant ce stage et permis le travail en groupe.

Quatrième partie

Annexes

On présente ici le code Matlab utilisé pour donner les résultats de la partie précédentes

```
function score=studyall()
%
% STUDYALL
%
% Le programme teste l'ensemble des problÃÍmes publics de svrt pour 100
% exemples par classe pour l'apprentissage et donne pour chacun d'eux
% le taux d'erreur de classification.

list=[1 2 3 5 6 8 11 12 13 17 18 20 21];
adascore=[0.467 0.054 0.058 0.427 0.269 0.083 0.104 0.198 0.399 0.388 ...
    0.067 0.482 0.501];

for i=1:length(list)
    X=study1(list(i),199);
    X;
    size(X,1);
    score(i)=1-naivebayes6(X(:,1:end))/size(X,1);
    disp(['Probleme ' num2str(list(i)) ': adaboost score ' ...
        num2str(adascore(i)) ' our score ' num2str(score(i))]);
end
mean(score)

function score=studyall2(k,l)
%
% STUDYALL2
%on prend k exemples d'apprentissage pour chaque classe et on teste sur les
%l suivants de chaque classe.
% k+l<200
% Le programme teste l'ensemble des problÃÍmes publics de svrt pour 200
% exemples par classe pour l'apprentissage et donne pour chacun d'eux
% le taux d'erreur de classification.

list=[1 2 3 5 6 8 11 12 13 17 18 20 21];
adascore=[0.467 0.054 0.058 0.427 0.269 0.083 0.104 0.198 0.399 0.388 ...
    0.067 0.482 0.501];

for i=1:length(list)
    X=study1(list(i),k);
    Y=studydata(list(i),k,l);
    score(i)=1-naivebayes62(X(:,1:end),Y(:,1:end))/size(Y,1);
    disp(['Probleme ' num2str(list(i)) ': adaboost score ' ...
        num2str(adascore(i)) ' our score ' num2str(score(i))]);
end
mean(score)

function X=study1(casenb,samplesize)
%
% STUDY1 (période d'apprentissage)
% Etude d'un problÃÍme de vision
% X=study1(casenb,samplesize)
%
% casenb est le numero du problÃÍme
% samplesize est le nombre d'images par classe pour l'apprentissage
% En sortie X est la matrice des caractÃĭristiques pour nourrir la
% classification avec 1 images par ligne (la classe constitue la
% premiÃÍre colonne)
```

```
pth=['C:\Users\David\Desktop\outils\ImagesSVRT\results_problem_' num2str(casenb) '/'];

%X=zeros(2*samplesize,23);
X=[];
cc=1;
l=0;
%for h=0:0
%    for i=1:1
%        A=sum(imread([pth 'sample_' num2str(h) '_' fnum2str(i) '.png']),3)/765;
%        [L,shapes]=ccomp_2(A);
%        x=classesI2_3Inf(shapes);
%        l=max(l,length(x));
%    end
%end
%l;
%X=zeros(2*samplesize,l+4);
for h=0:1
    for i=0:samplesize
        A=sum(imread([pth 'sample_' num2str(h) '_' fnum2str(i) '.png']),3)/765;
        % La ligne suivante permet de prendre en compte le contact : a
        % améliorer pour être plus générale : ici seule la première partie
        % du niveau 0 est faite.
        [u,v]=contact_final(imread([pth 'sample_' num2str(h) '_' fnum2str(i) '.png']));
        [L,shapes]=ccomp_2(A);
        COVNNUL=[];
        v;
        u;
        %% Les images qui ne sont pas lues sont remplies d'Inf. %%
        for j=1:shapes.ml
            if det(shapes.shapes(j).cov)>10^(-8)
            COVNNUL=[COVNNUL,j];
            end
        end
        longueur=length(COVNNUL);
        if longueur>0
            if numel(v)<=1
            X(cc,:)=[h classesI2_3Inf(shapes),u,v(1),Inf];
            cc=cc+1;
            else
                X(cc,:)=[h classesI2_3Inf(shapes),u,v(1),v(2)];
            cc=cc+1;
            end
        else X(cc,:)=(1+zeros(1,length(X(cc-1,:))))*Inf;cc=cc+1;
        end
    end
end

function score=studyall()
%
% STUDYALL
%
% Le programme teste l'ensemble des problÃÍmes publics de svrt pour 100
% exemples par classe pour l'apprentissage et donne pour chacun d'eux
% le taux d'erreur de classification.

list=[1 2 3 5 6 8 11 12 13 17 18 20 21];
```

```
adascore=[0.467 0.054 0.058 0.427 0.269 0.083 0.104 0.198 0.399 0.388 ...
    0.067 0.482 0.501];

for i=1:length(list)
    X=study1(list(i),199);
    X;
    size(X,1);
    score(i)=1-naivebayes6(X(:,1:end))/size(X,1);
    disp(['Probleme ' num2str(list(i)) ': adaboost score ' ...
        num2str(adascore(i)) ' our score ' num2str(score(i))]);
end
mean(score)

function x=classesI2_3Inf(pop,longueur,COVNNUL)

% Le programme fait appel aux fonctions qui extraits les caractéristiques
% voulues utilisant la similitude, translation, aire.

%Modif à faire : introduire longueur comme le cardinal d'un cluster CAD
%faire le même type de fonction que caracts_I2 mais pour la relation
%triviale.
%Introduire les histoires d'intérieur dans les programmes qui font du
%clustering.

shapes=pop.shapes;
Scale=shapes(1).aires;

x=[];
cc=1;
COVNNUL=[];

for i=1:pop.ml
    %determinantforme=det(pop.shapes(i).cov)
    if det(pop.shapes(i).cov)>10^(-8)
        COVNNUL=[COVNNUL,i];
    end
end
longueur=length(COVNNUL);

x=[x, longueur];
x=[x, caracts_I2Inf(pop,longueur,COVNNUL)];
%x=[x, caractr(pop,longueur,COVNNUL)];
%x=[x, caracth(pop,longueur,COVNNUL)];
x=[x, caracttInf(pop,longueur,COVNNUL)];
x=[x, caractaInf(pop,longueur,COVNNUL)];
%x=[x, Scale];
pop.ml;
for i=1:2
    if pop.ml>=i
    lbarI(i)=shapes(i).barI;
    lbarJ(i)=shapes(i).barJ;
    else
        lbarI(i)=0;
        lbarJ(i)=0;
    end
end
x=[x, mean(lbarI), mean(lbarJ)];
```

```
if pop.ml>=2
x=[x, indice(pop.shapes(1).cov,pop.shapes(2).barI+i*pop.shapes(2).barJ)];
else x=[x,Inf];
end

function feat=caracts_I2Inf(pop,longueur,COVNNUL)

% Extrait les caractéristiques voulues d'une image, en considérant la
% relation de similitude.
% Le manque d'infirmation est remplacé par Inf
% Prend en argument ce qui est rendu par Ccomp_2, le nombre de formes à
% prendre en compte, et les indices (dans pop) de ces formes
% On ne travaille que sur les formes de matrice de cov inversible

%
seuils=0.15;
x=[];
cc=1;
I2=[1,0;0,1];
M=[];
longueur;
       for k=1:longueur
           M(1,k)=dists(I2,pop.shapes(COVNNUL(k)).cov);
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
baryI=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
COV1=[];
ELEM=[];
LONG=[];
while numel(M)>0
    X=M;
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j)-X(1))<seuils
                i=i+1;
                L=[L,j];
                absc=[absc,pop.shapes(j).barI];
                ord=[ord,pop.shapes(j).barJ];
```

```
            end
        end
        L;
        if length(L)>0
        LONG=[LONG,length(L)];
        end
        P=L+n;
        ELEM=[ELEM,P];
        for j=1:length(P)
            for k=1:length(P)
            MT=[MT,sqrt((pop.shapes(P(k)).barI-pop.shapes(P(j)).barI)^2+(pop.shapes(P(k)).barJ-pop.s
            end
        end
        MT;
        L;
        [absc,ord]=grossissement(absc,ord);
        Z=[absc;ord];
        Z=Z';
        %determinant=det(cov(Z))
        COV1=[COV1,[cov(Z)]];
        baryI=[baryI,mean(absc)];
        baryJ=[baryJ,mean(ord)];
        Y=[Y,i];
        Y;
        size(MT,1);
        for i=1:length(MT)
            p=0;
            for k=1:length(AT)
                if abs(MT(i)-AT(k))<5
                    p=1;
                end
            end
            if MT(i)>0 & p==0 AT=[AT,MT(i)];
            end
        end

        for j=1:length(L)
            X(L(j))=0;
        end
        Z=[];
        for i=1:length(X)
            if X(i)>0
                Z=[Z,X(i)];
            end
        end
        X=Z;
    M=X;
end
AT;
Y;
% Niveau 0
if numel(baryI)>0
    baryI=sortt(Y,baryI);
    for i=1:3
        if numel(baryI)>=i
            x(cc)=(baryI(i));
        else
```

```
            x(cc)=Inf;
        end
        cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(baryJ)>0
    baryJ=sortt(Y,baryJ);
    for i=1:3
        if numel(baryJ)>=i
            x(cc)=(baryJ(i));
        else
            x(cc)=Inf;
        end
           cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(AT)>0
    AT=sortt(Y,AT);
end
Y;
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
        if numel(Y)>=i
            x(cc)=Y(i);cc=cc+1;
        else
            x(cc)=Inf;cc=cc+1;
        end
    end
else
    for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
n=0;
x(cc)=length(AT);cc=cc+1;
for i=1:3
    if numel(AT)>=i
        x(cc)=AT(i);
        cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
end

% Niveau 2
if numel(baryI)>0 && numel(baryJ)>0
```

```
    [baryI,baryJ]=grossissement(baryI,baryJ);
    Z=[baryI;baryJ];
    Z=Z';
    M_cov=cov(Z);
    det(M_cov);
    if det(M_cov)>10^(-(8))
        x(cc)=dists(I2,M_cov);cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
else
    x(cc)=Inf;cc=cc+1;
end

% Niveau 0.5 barycentre intercluster
M=[];
       for k=1:3
           for l=1:3
               if numel(baryI)>=k & numel(baryJ)>=l
                    M(l,k)=sqrt((baryI(k)-baryI(l)).^2+(baryJ(k)-baryJ(l)).^2);
               else M(l,k)=0;
               end
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
X=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<5
                i=i+1;
                L=[L,j];
            end
        end
        L;
        Y=[Y,i];
        Y;
        for j=1:length(L)
```

```
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>5
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end

% Niveau 0.5 intercluster. On ne considère que les interdistances entre le
% plus gros cluster et la plus grosse forme du deuxième cluster (pb 17) -> c'est à
% compléter.
indice=1;
max_courant=LONG(1);
for i=1:length(LONG)
    if LONG(i)>max_courant
        indice=i;
        max_courant=LONG(i);
    end
end
CLUSTER1=[];
LONG1=LONG(indice);
ELIMINATION=[];
if indice<length(ELEM)
    if indice>1
    debut=sum(LONG(1:indice-1))+1;
    else
        debut=1;
    end
    for i=debut:debut+LONG(indice)-1
        CLUSTER1=[CLUSTER1,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
else
    for i=LONG(indice):length(ELEM)
        CLUSTER1=[CLUSTER1,ELEM(i)];
        ELEM(i)=0;
```

```
        ELIMINATION=[ELIMINATION,i];
    end
end
LONG(indice)=0;
LONGNEW=[];
for i=1:length(LONG)
    if LONG(i)>0
        LONGNEW=[LONGNEW,LONG(i)];
    end
end
ELEM2=[];

for i=1:length(ELEM)
    if ELEM(i)>0
        ELEM2=[ELEM2,ELEM(i)];
    end
end
LONG=LONGNEW;
ELEM=ELEM2;
CLUSTER1;
ELEM;
LONG;

if numel(LONG)>0
indice=1;
max_courant=LONG(1);
for i=1:length(LONG)
    if LONG(i)>max_courant
        indice=i;
        max_courant=LONG(i);
    end
end
CLUSTER2=[];
LONG2=LONG(indice);
ELIMINATION=[];
if indice<length(ELEM)
    if indice>1
    debut=sum(LONG(1:indice-1))+1;
    else
        debut=1;
    end
    for i=debut:debut+LONG(indice)-1
        CLUSTER2=[CLUSTER2,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
else
    for i=LONG(indice):length(ELEM)
        CLUSTER2=[CLUSTER2,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
end
LONG(indice)=0;
LONGNEW=[];
for i=1:length(LONG)
    if LONG(i)>0
```

```
        LONGNEW=[LONGNEW,LONG(i)];
    end
end
ELEMNEW=[];
for i=1:length(ELEM)
    if ELEM(i)>0
        ELEMNEW=[ELEMNEW,ELEM];
    end
end
LONG=LONGNEW;
ELEM=ELEMNEW;
CLUSTER2;
else
    CLUSTER2=[];
    LONG2=[];
end
% On a CLUSTER1 avec LONG1 sa longueur et pareil pour 2
% On étudie les distances des formes du cluster 1 à la première forme
% du cluster 2
CLUSTER1;
LONG1;
CLUSTER2;
LONG2;
M=[];
% Les lignes suivantes considères les distances interclusters sur le quotient par les
% similitudes : relativement non significatif
%if LONG2>=1
%for i=1:3
%    if LONG1>=i
%        x(cc)=dists(pop.shapes(CLUSTER1(i)).cov,pop.shapes(CLUSTER2(1)).cov);cc=cc+1;
%    else
%        x(cc)=0;cc=cc+1;
%    end
%end
%else
%    for i=1:3
%        x(cc)=0;cc=cc+1;
 %   end
%end
if LONG2>=1
for i=1:3
    if LONG1>=i
        M(1,i)=sqrt((pop.shapes(CLUSTER1(i)).barI-pop.shapes(CLUSTER2(1)).barI).^2+(pop.shapes(CLUST
        x(cc)=M(1,i);cc=cc+1;
    else
        M(1,i)=0;
        x(cc)=Inf;cc=cc+1;
    end
end
else M=[0,0,0];
    for i=1:3 x(cc)=Inf;cc=cc+1;end
end

Y=[];
X=[];
M;
n=0;
```

```
L=[];
AT=[];
nombre_de_tour=0;
while numel(M)>0
    nombre_de_tour=nombre_de_tour+1;
    X=M;
    MT=[];
    n=length(L);
        i=0;
        for j=1:length(X)
            if abs(X(j)-X(1))<5
                i=i+1;
                L=[L,j];
            end
        end
        Y=[Y,i];
        for j=1:length(L)
            X(L(j))=0;
        end
        Z=[];
        m=1;
        for j=1:length(X)
            if X(j)>0 Z(m)=X(j);
                m=m+1;
            end
        end
        X=Z;
    M=X;
end
Y;
if numel(AT)>0
AT=sortt(Y,AT);
end
AT;
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end


% Niveau 1 -> Niveau 0 on recommence les distances intra et intercluster
% avec une nouvelle image : les matrices de covariances des clusters.
% Attention : les distances intercluster ne sont pas programmées. A
% rajouter.
% Attention : erreur de programmation, on ne doit pas prendre les formes
```

```
% comme c'est fait pour le calcul des distances intracluster mais les
% barycentres ...
COVNNUL1=[];
if numel(COV1)>0
    for i=1:size(COV1(1,:))/2
        if det(COV1([1,2],[i,i+1]))>10^(-8)
            COVNNUL1=[COVNNUL1,i];
        end
    end
end
I2=[1,0;0,1];
longueur=length(COVNNUL1);
M=[];
COV2=[];
for i=1:longueur
    COV2=[COV2,COV1([1,2],[COVNNUL(i),COVNNUL(i)+1])];
end
COV1=COV2;

longueur;

M=[];
       for k=1:longueur
           M(1,k)=dists(I2,COV1([1,2],[l,l+1]));
       end
       for i=1:3
           if length(M)>=i
               x(cc)=M(i);cc=cc+1; % Relativement inutile.
           else
               x(cc)=Inf;
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
baryI=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
COV1=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
```

```
            if abs(X(j))<seuils
                i=i+1;
                L=[L,j];
                absc=[absc,pop.shapes(j).barI];
                % Même erreur, on doit prendre les barycentres des
                %barycentres des matrices de covariances étant semblables
                % Les matrices de covariance sont celles des clusters du
                % niveau 1.
                ord=[ord,pop.shapes(j).barJ];
            end
        end
        L;
        P=L+n;
        for j=1:length(P)
            for k=1:length(P) % C'est ici l'erreur
            MT=[MT,sqrt((pop.shapes(P(k)).barI-pop.shapes(P(j)).barI)^2+(pop.shapes(P(k)).barJ-pop.s
            end
        end
        MT;
        L;
        baryI=[baryI,mean(absc)];
        baryJ=[baryJ,mean(ord)];
        [absc,ord]=grossissement(absc,ord);
        Z=[absc;ord];
        Z=Z';
        %determinant=det(cov(Z))
        COV1=[COV1,[cov(Z)]];
        Y=[Y,i];
        Y;
        size(MT,1);
        for i=1:length(MT)
            p=0;
                for k=1:length(AT)
                    if abs(MT(i)-AT(k))<5
                        p=1;
                    end
                end
                if MT(i)>0 & p==0 AT=[AT,MT(i)];
                end
        end
        for j=1:length(L)
            X(L(j))=0;
        end
        Z=[];
        for i=1:length(X)
            if X(i)>0
             Z=[Z,X(i)];
            end
        end
        X=Z;
    M=X
end
if numel(baryI)>0
    baryI=sortt(Y,baryI);
    for i=1:3
        if numel(baryI)>=i
            x(cc)=(baryI(i));
```

```
        else
            x(cc)=Inf;
        end
            cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(baryJ)>0
    baryJ=sortt(Y,baryJ);
        for i=1:3
            if numel(baryJ)>=i
                x(cc)=(baryJ(i));
            else
                x(cc)=Inf;
            end
            cc=cc+1;
        end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(AT)>0
    AT=sortt(Y,AT);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
        if numel(Y)>=i
            x(cc)=Y(i);cc=cc+1;
        else
            x(cc)=Inf;cc=cc+1;
        end
    end
else
    for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
n=0;
x(cc)=length(AT);cc=cc+1;
for i=1:3
    if numel(AT)>=i
        x(cc)=AT(i);
        cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
end

% On retourne au niveau 2 après avoir revisité les niveaux 0 et 1
if numel(baryI)>0 && numel(baryJ)>0
```

```
    [baryI,baryJ]=grossissement(baryI,baryJ);
    Z=[baryI;baryJ];
    Z=Z';
    M_cov=cov(Z);
    det(M_cov);
    if det(M_cov)>10^(-(8))
        x(cc)=dists(I2,M_cov);cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
else
    x(cc)=Inf;cc=cc+1;
end

%Niveau 0.5 distances intercluster entre barycentres après avoir revisité
%le niveau 0. Attention erreur de programmation voir au dessus.
M=[];
       for k=1:3
           for l=1:3
               if numel(baryI)>=k & numel(baryJ)>=l
                    M(l,k)=sqrt((baryI(k)-baryI(l)).^2+(baryJ(k)-baryJ(l)).^2);
               else M(l,k)=0;
               end
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
X=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<5
                i=i+1;
                L=[L,j];
            end
        end
        L;
        Y=[Y,i];
        Y;
```

```
        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>5
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end



feat=x;

function feat=caracttInf(pop,longueur,COVNNUL)

%Cf les commentaires pour la similitude caracts_I2Inf : le programme est le même.
%Seule différence : on considère des interdistances par translation et non
%pas des distances à I2.

seuilt=0.2;
x=[];
cc=1;
I2=[1,0;0,1];
Scale=pop.shapes(1).aires;
M=[];
       for k=1:longueur
           for l=1:longueur
           M(l,k)=distt(pop.shapes(COVNNUL(l)).cov,pop.shapes(COVNNUL(k)).cov);
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
```

```
P=[];
baryI=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
COV1=[];
ELEM=[];
LONG=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<seuilt
                i=i+1;
                L=[L,j];
                absc=[absc,pop.shapes(j).barI];
                ord=[ord,pop.shapes(j).barJ];
            end
        end
        L;
        if length(L)>0
        LONG=[LONG,length(L)];
        end
        P=L+n;
        ELEM=[ELEM,P];
        for j=1:length(P)
            for k=1:length(P)
            MT=[MT,sqrt((pop.shapes(P(k)).barI-pop.shapes(P(j)).barI)^2+(pop.shapes(P(k)).barJ-pop.s
            end
        end
        MT;
        L;
        [absc,ord]=grossissement(absc,ord);
        Z=[absc;ord];
        Z=Z';
        %determinant=det(cov(Z))
        COV1=[COV1,[cov(Z)]];
        baryI=[baryI,mean(absc)];
        baryJ=[baryJ,mean(ord)];
        Y=[Y,i];
        Y;
        size(MT,1);
        for i=1:length(MT)
            p=0;
            for k=1:length(AT)
                if abs(MT(i)-AT(k))<5
                    p=1;
                end
            end
```

```
            if MT(i)>0 & p==0 AT=[AT,MT(i)];
            end
        end

        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>seuilt
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(baryI)>0
    baryI=sortt(Y,baryI);
    for i=1:3
        if numel(baryI)>=i
            x(cc)=(baryI(i));
        else
            x(cc)=Inf;
        end
        cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(baryJ)>0
    baryJ=sortt(Y,baryJ);
    for i=1:3
        if numel(baryJ)>=i
            x(cc)=(baryJ(i));
        else
            x(cc)=Inf;
        end
           cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(AT)>0
    AT=sortt(Y,AT);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
        if numel(Y)>=i
            x(cc)=Y(i);cc=cc+1;
        else
```

```
            x(cc)=Inf;cc=cc+1;
        end
    end
else
    for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
n=0;
x(cc)=length(AT);cc=cc+1;
for i=1:3
    if numel(AT)>=i
        x(cc)=AT(i);
        cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
end

if numel(baryI)>0 && numel(baryJ)>0
    [baryI,baryJ]=grossissement(baryI,baryJ);
    Z=[baryI;baryJ];
    Z=Z';
    M_cov=cov(Z);
    det(M_cov);
    if det(M_cov)>10^(-(8))
        x(cc)=dists(I2,M_cov);cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
else
    x(cc)=Inf;cc=cc+1;
end

M=[];
       for k=1:3
           for l=1:3
               if numel(baryI)>=k & numel(baryJ)>=l
                    M(l,k)=sqrt((baryI(k)-baryI(l)).^2+(baryJ(k)-baryJ(l)).^2);
               else M(l,k)=0;
               end
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
X=[];
```

```
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<5
                i=i+1;
                L=[L,j];
            end
        end
        L;
        Y=[Y,i];
        Y;
        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>5
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
indice=1;
max_courant=LONG(1);
for i=1:length(LONG)
    if LONG(i)>max_courant
        indice=i;
        max_courant=LONG(i);
    end
end
CLUSTER1=[];
LONG1=LONG(indice);
ELIMINATION=[];
```

```
if indice<length(ELEM)
    if indice>1
    debut=sum(LONG(1:indice-1))+1;
    else
        debut=1;
    end
    for i=debut:debut+LONG(indice)-1
        CLUSTER1=[CLUSTER1,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
else
    for i=LONG(indice):length(ELEM)
        CLUSTER1=[CLUSTER1,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
end
LONG(indice)=0;
LONGNEW=[];
for i=1:length(LONG)
    if LONG(i)>0
        LONGNEW=[LONGNEW,LONG(i)];
    end
end
ELEM2=[];

for i=1:length(ELEM)
    if ELEM(i)>0
        ELEM2=[ELEM2,ELEM(i)];
    end
end
LONG=LONGNEW;
ELEM=ELEM2;
CLUSTER1;
ELEM;
LONG;

if numel(LONG)>0
indice=1;
max_courant=LONG(1);
for i=1:length(LONG)
    if LONG(i)>max_courant
        indice=i;
        max_courant=LONG(i);
    end
end
CLUSTER2=[];
LONG2=LONG(indice);
ELIMINATION=[];
if indice<length(ELEM)
    if indice>1
    debut=sum(LONG(1:indice-1))+1;
    else
        debut=1;
    end
    for i=debut:debut+LONG(indice)-1
```

```
        CLUSTER2=[CLUSTER2,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
else
    for i=LONG(indice):length(ELEM)
        CLUSTER2=[CLUSTER2,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
end
LONG(indice)=0;
LONGNEW=[];
for i=1:length(LONG)
    if LONG(i)>0
        LONGNEW=[LONGNEW,LONG(i)];
    end
end
ELEMNEW=[];
for i=1:length(ELEM)
    if ELEM(i)>0
        ELEMNEW=[ELEMNEW,ELEM];
    end
end
LONG=LONGNEW;
ELEM=ELEMNEW;
CLUSTER2;
else
    CLUSTER2=[];
    LONG2=[];
end
% On a CLUSTER1 avec LONG1 sa longueur et pareil pour 2
% On étudie les distances des formes du cluster 1 à la première forme
% du cluster 2
CLUSTER1;
LONG1;
CLUSTER2;
LONG2;
M=[];
if LONG2>=1
for i=1:3
    if LONG1>=i
        M(1,i)=sqrt((pop.shapes(CLUSTER1(i)).barI-pop.shapes(CLUSTER2(1)).barI).^2+(pop.shapes(CLUST
        x(cc)=M(1,i);cc=cc+1;
    else
        M(1,i)=0;
        x(cc)=Inf;cc=cc+1;
    end
end
else M=[0,0,0];
    for i=1:3 x(cc)=Inf;cc=cc+1; end
end

Y=[];
X=[];
M;
n=0;
```

```
L=[];
AT=[];
nombre_de_tour=0;
while numel(M)>0
    nombre_de_tour=nombre_de_tour+1;
    X=M;
    MT=[];
    n=length(L);
        i=0;
        for j=1:length(X)
            if abs(X(j)-X(1))<5
                i=i+1;
                L=[L,j];
            end
        end
        Y=[Y,i];
        for j=1:length(L)
            X(L(j))=0;
        end
        Z=[];
        m=1;
        for j=1:length(X)
            if X(j)>0 Z(m)=X(j);
                m=m+1;
            end
        end
        X=Z;
    M=X;
end
Y;
if numel(AT)>0
AT=sortt(Y,AT);
end
AT;
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
```

```
COVNNUL1=[];
```

```
if numel(COV1)>0
    for i=1:size(COV1(1,:))/2
        if det(COV1([1,2],[i,i+1]))>10^(-8)
            COVNNUL1=[COVNNUL1,i];
        end
    end
end
I2=[1,0;0,1];
longueur=length(COVNNUL1);
M=[];
COV2=[];
for i=1:longueur
    COV2=[COV2,COV1([1,2],[COVNNUL(i),COVNNUL(i)+1])];
end
COV1=COV2;

longueur;

M=[];
Ms=[];
       for k=1:longueur
           for l=1:longueur
           M(l,k)=distt(COV1([1,2],[k,k+1]),COV1([1,2],[l,l+1]));
           end
       end
       for k=1:longueur
           Ms(1,k)=dists(I2,COV1([1,2],[l,l+1]));
       end
       for i=1:3
           if length(Ms)>=i
               x(cc)=Ms(i);cc=cc+1;
           else
               x(cc)=Inf;cc=cc+1;
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
baryI=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
COV1=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
```

```
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<seuilt
                i=i+1;
                L=[L,j];
                absc=[absc,pop.shapes(j).barI];
                ord=[ord,pop.shapes(j).barJ];
            end
        end
        L;
        P=L+n;
        for j=1:length(P)
            for k=1:length(P)
            MT=[MT,sqrt((pop.shapes(P(k)).barI-pop.shapes(P(j)).barI)^2+(pop.shapes(P(k)).barJ-pop.s
            end
        end
        MT;
        L;
        [absc,ord]=grossissement(absc,ord);
        Z=[absc;ord];
        Z=Z';
        %determinant=det(cov(Z))
        COV1=[COV1,[cov(Z)]];
        baryI=[baryI,mean(absc)];
        baryJ=[baryJ,mean(ord)];

        Y=[Y,i];
        Y;
        size(MT,1);
        for i=1:length(MT)
            p=0;
                for k=1:length(AT)
                    if abs(MT(i)-AT(k))<5
                        p=1;
                    end
                end
                if MT(i)>0 & p==0 AT=[AT,MT(i)];
                end
        end
        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>seuilt
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(baryI)>0
    baryI=sortt(Y,baryI);
    for i=1:3
        if numel(baryI)>=i
            x(cc)=(baryI(i));
```

```
        else
            x(cc)=Inf;
        end
            cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(baryJ)>0
    baryJ=sortt(Y,baryJ);
        for i=1:3
            if numel(baryJ)>=i
                x(cc)=(baryJ(i));
            else
                x(cc)=Inf;
            end
            cc=cc+1;
        end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(AT)>0
    AT=sortt(Y,AT);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
        if numel(Y)>=i
            x(cc)=Y(i);cc=cc+1;
        else
            x(cc)=Inf;cc=cc+1;
        end
    end
else
    for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
n=0;
x(cc)=length(AT);cc=cc+1;
for i=1:3
    if numel(AT)>=i
        x(cc)=AT(i);
        cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
end

if numel(baryI)>0 && numel(baryJ)>0
    [baryI,baryJ]=grossissement(baryI,baryJ);
```

```
    Z=[baryI;baryJ];
    Z=Z';
    M_cov=cov(Z);
    det(M_cov);
    if det(M_cov)>10^(-(8))
        x(cc)=dists(I2,M_cov);cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
else
    x(cc)=Inf;cc=cc+1;
end

M=[];
       for k=1:3
           for l=1:3
               if numel(baryI)>=k & numel(baryJ)>=l
                    M(l,k)=sqrt((baryI(k)-baryI(l)).^2+(baryJ(k)-baryJ(l)).^2);
               else M(l,k)=0;
               end
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
X=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<5
                i=i+1;
                L=[L,j];
            end
        end
        L;
        Y=[Y,i];
        Y;
        for j=1:length(L)
            X(L(j))=0;
        end
```

```
        P=[];
        for j=1:length(X)
            if abs(X(j))>5
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end



feat=x;

function feat=caractaInf(pop,longueur,COVNNUL)

%Cf les commentaires pour la similitude caracts_I2Inf : le programme est le même.
%Seule différence : on considère des interdistances par aires et non
%pas des distances à I2.

seuila=100;
seuils=0.07;

x=[];
cc=1;
I2=[1,0;0,1];
Scale=pop.shapes(1).aires;
M=[];
       for k=1:longueur
           for l=1:longueur
           M(l,k)=abs(pop.shapes(COVNNUL(l)).aires-pop.shapes(COVNNUL(k)).aires);
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
```

```
baryI=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
COV1=[];
ELEM=[];
LONG=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<seuila
                i=i+1;
                L=[L,j];
                absc=[absc,pop.shapes(j).barI];
                ord=[ord,pop.shapes(j).barJ];
            end
        end
        L;
        if length(L)>0
        LONG=[LONG,length(L)];
        end
        P=L+n;
        ELEM=[ELEM,P];
        for j=1:length(P)
            for k=1:length(P)
            MT=[MT,sqrt((pop.shapes(P(k)).barI-pop.shapes(P(j)).barI)^2+(pop.shapes(P(k)).barJ-pop.s
            end
        end
        MT;
        L;
        [absc,ord]=grossissement(absc,ord);
        Z=[absc;ord];
        Z=Z';
        %determinant=det(cov(Z))
        COV1=[COV1,[cov(Z)]];
        baryI=[baryI,mean(absc)];
        baryJ=[baryJ,mean(ord)];
        Y=[Y,i];
        Y;
        size(MT,1);
        for i=1:length(MT)
            p=0;
            for k=1:length(AT)
                if abs(MT(i)-AT(k))<5
                    p=1;
                end
            end
            if MT(i)>0 & p==0 AT=[AT,MT(i)];
```

```
            end
        end

        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>seuila
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
AT;
if numel(baryI)>0
    baryI=sortt(Y,baryI);
    for i=1:3
        if numel(baryI)>=i
            x(cc)=(baryI(i));
        else
            x(cc)=Inf;
        end
        cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(baryJ)>0
    baryJ=sortt(Y,baryJ);
    for i=1:3
        if numel(baryJ)>=i
            x(cc)=(baryJ(i));
        else
            x(cc)=Inf;
        end
           cc=cc+1;
    end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(AT)>0
    AT=sortt(Y,AT);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
        if numel(Y)>=i
            x(cc)=Y(i);cc=cc+1;
        else
```

```
            x(cc)=Inf;cc=cc+1;
        end
    end
else
    for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
n=0;
x(cc)=length(AT);cc=cc+1;
for i=1:3
    if numel(AT)>=i
        x(cc)=AT(i);
        cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
end

if numel(baryI)>0 && numel(baryJ)>0
    [baryI,baryJ]=grossissement(baryI,baryJ);
    Z=[baryI;baryJ];
    Z=Z';
    M_cov=cov(Z);
    det(M_cov);
    if det(M_cov)>10^(-(8))
        x(cc)=dists(I2,M_cov);cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
else
    x(cc)=Inf;cc=cc+1;
end

M=[];
       for k=1:3
           for l=1:3
               if numel(baryI)>=k & numel(baryJ)>=l
                    M(l,k)=sqrt((baryI(k)-baryI(l)).^2+(baryJ(k)-baryJ(l)).^2);
               else M(l,k)=0;
               end
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
X=[];
```

```
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<5
                i=i+1;
                L=[L,j];
            end
        end
        L;
        Y=[Y,i];
        Y;
        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>5
                P=[P,j];
            end
        end
        P;
    M=M(P,P);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
indice=1;
max_courant=LONG(1);
for i=1:length(LONG)
    if LONG(i)>max_courant
        indice=i;
        max_courant=LONG(i);
    end
end
CLUSTER1=[];
LONG1=LONG(indice);
ELIMINATION=[];
```

```
if indice<length(ELEM)
    if indice>1
    debut=sum(LONG(1:indice-1))+1;
    else
        debut=1;
    end
    for i=debut:debut+LONG(indice)-1
        CLUSTER1=[CLUSTER1,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
else
    for i=LONG(indice):length(ELEM)
        CLUSTER1=[CLUSTER1,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
end
LONG(indice)=0;
LONGNEW=[];
for i=1:length(LONG)
    if LONG(i)>0
        LONGNEW=[LONGNEW,LONG(i)];
    end
end
ELEM2=[];

for i=1:length(ELEM)
    if ELEM(i)>0
        ELEM2=[ELEM2,ELEM(i)];
    end
end
LONG=LONGNEW;
ELEM=ELEM2;
CLUSTER1;
ELEM;
LONG;

if numel(LONG)>0
indice=1;
max_courant=LONG(1);
for i=1:length(LONG)
    if LONG(i)>max_courant
        indice=i;
        max_courant=LONG(i);
    end
end
CLUSTER2=[];
LONG2=LONG(indice);
ELIMINATION=[];
if indice<length(ELEM)
    if indice>1
    debut=sum(LONG(1:indice-1))+1;
    else
        debut=1;
    end
    for i=debut:debut+LONG(indice)-1
```

```
        CLUSTER2=[CLUSTER2,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
else
    for i=LONG(indice):length(ELEM)
        CLUSTER2=[CLUSTER2,ELEM(i)];
        ELEM(i)=0;
        ELIMINATION=[ELIMINATION,i];
    end
end
LONG(indice)=0;
LONGNEW=[];
for i=1:length(LONG)
    if LONG(i)>0
        LONGNEW=[LONGNEW,LONG(i)];
    end
end
ELEMNEW=[];
for i=1:length(ELEM)
    if ELEM(i)>0
        ELEMNEW=[ELEMNEW,ELEM];
    end
end
LONG=LONGNEW;
ELEM=ELEMNEW;
CLUSTER2;
else
    CLUSTER2=[];
    LONG2=[];
end
% On a CLUSTER1 avec LONG1 sa longueur et pareil pour 2
% On étudie les distances des formes du cluster 1 à la première forme
% du cluster 2
CLUSTER1;
LONG1;
CLUSTER2;
LONG2;
M=[];
if LONG2>=1
for i=1:3
    if LONG1>=i
        M(1,i)=sqrt((pop.shapes(CLUSTER1(i)).barI-pop.shapes(CLUSTER2(1)).barI).^2+(pop.shapes(CLUST
        x(cc)=M(1,i);cc=cc+1;
    else
        M(1,i)=0;
        x(cc)=Inf;cc=cc+1;
    end
end
else M=[0,0,0];
    for i=1:3 x(cc)=Inf;cc=cc+1; end
end

Y=[];
X=[];
M;
n=0;
```

```
L=[];
AT=[];
nombre_de_tour=0;
while numel(M)>0
    nombre_de_tour=nombre_de_tour+1;
    X=M;
    MT=[];
    n=length(L);
        i=0;
        for j=1:length(X)
            if abs(X(j)-X(1))<5
                i=i+1;
                L=[L,j];
            end
        end
        Y=[Y,i];
        for j=1:length(L)
            X(L(j))=0;
        end
        Z=[];
        m=1;
        for j=1:length(X)
            if X(j)>0 Z(m)=X(j);
                m=m+1;
            end
        end
        X=Z;
    M=X;
end
Y;
if numel(AT)>0
AT=sortt(Y,AT);
end
AT;
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
```

```
COVNNUL1=[];
```

```
if numel(COV1)>0
    for i=1:size(COV1(1,:))/2
        if det(COV1([1,2],[i,i+1]))>10^(-8)
            COVNNUL1=[COVNNUL1,i];
        end
    end
end
I2=[1,0;0,1];
longueur=length(COVNNUL1);
M=[];
COV2=[];
for i=1:longueur
    COV2=[COV2,COV1([1,2],[COVNNUL(i),COVNNUL(i)+1])];
end
COV1=COV2;

longueur;

M=[];
       for k=1:longueur
           M(1,k)=dists(I2,COV1([1,2],[l,l+1]));
       end
       for i=1:3
           if length(M)>=i
               x(cc)=M(i);cc=cc+1;
           else
               x(cc)=Inf;cc=cc+1;
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
baryI=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
COV1=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<seuils
                i=i+1;
                L=[L,j];
```

```
                absc=[absc,pop.shapes(j).barI];
                ord=[ord,pop.shapes(j).barJ];
            end
        end
        L;
        P=L+n;
        for j=1:length(P)
            for k=1:length(P)
            MT=[MT,sqrt((pop.shapes(P(k)).barI-pop.shapes(P(j)).barI)^2+(pop.shapes(P(k)).barJ-pop.s
            end
        end
        MT;
        L;
        [absc,ord]=grossissement(absc,ord);
        Z=[absc;ord];
        Z=Z';
        %determinant=det(cov(Z))
        COV1=[COV1,[cov(Z)]];
        baryI=[baryI,mean(absc)];
        baryJ=[baryJ,mean(ord)];

        Y=[Y,i];
        Y;
        size(MT,1);
        for i=1:length(MT)
            p=0;
                for k=1:length(AT)
                    if abs(MT(i)-AT(k))<5
                        p=1;
                    end
                end
                if MT(i)>0 & p==0 AT=[AT,MT(i)];
                end
        end
        for j=1:length(L)
            X(L(j))=0;
        end
        Z=[];
        for i=1:length(X)
            if X(i)>0
             Z=[Z,X(i)];
            end
        end
        X=Z;
    M=X
end
if numel(baryI)>0
    baryI=sortt(Y,baryI);
    for i=1:3
        if numel(baryI)>=i
            x(cc)=(baryI(i));
        else
            x(cc)=Inf;
        end
            cc=cc+1;
    end
else
```

```
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(baryJ)>0
    baryJ=sortt(Y,baryJ);
        for i=1:3
            if numel(baryJ)>=i
                x(cc)=(baryJ(i));
            else
                x(cc)=Inf;
            end
            cc=cc+1;
        end
else
    for i=1:3
        x(cc)=Inf;cc=cc+1;
    end
end
if numel(AT)>0
    AT=sortt(Y,AT);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
        if numel(Y)>=i
            x(cc)=Y(i);cc=cc+1;
        else
            x(cc)=Inf;cc=cc+1;
        end
    end
else
    for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end
n=0;
x(cc)=length(AT);cc=cc+1;
for i=1:3
    if numel(AT)>=i
        x(cc)=AT(i);
        cc=cc+1;
    else
        x(cc)=Inf;cc=cc+1;
    end
end

if numel(baryI)>0 && numel(baryJ)>0
    [baryI,baryJ]=grossissement(baryI,baryJ);
    Z=[baryI;baryJ];
    Z=Z';
    M_cov=cov(Z);
    det(M_cov);
    if det(M_cov)>10^(-(8))
        x(cc)=dists(I2,M_cov);cc=cc+1;
```

```
    else
        x(cc)=Inf;cc=cc+1;
    end
else
    x(cc)=Inf;cc=cc+1;
end

M=[];
       for k=1:3
           for l=1:3
               if numel(baryI)>=k & numel(baryJ)>=l
                    M(l,k)=sqrt((baryI(k)-baryI(l)).^2+(baryJ(k)-baryJ(l)).^2);
               else M(l,k)=0;
               end
           end
       end
M;
Y=[];
k=1;
p=1;
n=0;
AT=[];
L=[];
P=[];
absc=[];
ord=[];
baryJ=[];
PT=[];
LT=[];
X=[];
while numel(M)>0
    X=M(1,:);
    MT=[];
        i=0;
        n=length(L);
        L=[];
        absc=[];
        ord=[];
        X;
        for j=1:length(X)
            if abs(X(j))<5
                i=i+1;
                L=[L,j];
            end
        end
        L;
        Y=[Y,i];
        Y;
        for j=1:length(L)
            X(L(j))=0;
        end
        P=[];
        for j=1:length(X)
            if abs(X(j))>5
                P=[P,j];
            end
        end
```

```
        P;
    M=M(P,P);
end
if numel(Y)>0
    x(cc)=length(Y);
    cc=cc+1;
    Y=sort(Y,'descend');
    for i=1:3
    if numel(Y)>=i
       x(cc)=Y(i);
    else
       x(cc)=Inf;
    end
       cc=cc+1;
    end
else for i=1:4
        x(cc)=Inf;cc=cc+1;
    end
end




feat=x;

function score=naivebayes6(X)%,Y)
% NAIVEBAYES
% Naive bayes classifier.
% X est la matrice pour apprentissage
% Y est la matrice sur laquelle on effectue le test



tag=X(:,1);
data=X(:,2:end);
p1=size(data,1);
p2=size(data,2);

sig0=10^(-5); % à revoir
for h=0:1
    cc=h+1;
    for i=1:p2
        nreal=sum((tag==h)&(data(:,i)~=Inf));
        if nreal==0
            mc(cc,i)=0;
            sig1(cc,i)=sig0;
        else
            mc(cc,i)=mean(data((tag==h)&(data(:,i)~=Inf),i));
            sig(cc,i)=std(data((tag==h)&(data(:,i)~=Inf),i));
            sig1(cc,i)=sqrt(((nreal-1)*sig(cc,i)^2+sig0)/(nreal+1));
        end
        A=data(:,i);
        dirac(cc,i)=(occurence(A(tag==h),Inf)+1)/(sum(tag==h)+2);
    end
end

%Ici on met un ordre de preference pour les caracteristiques en prenant en
```

```
%compte comme critère de pertinence l'erreur de Bayes pour chaque
%caractéristique
distpertinence=beest(mc,sig1,dirac);
mc=[sortt(distpertinence,mc(1,:));sortt(distpertinence,mc(2,:))];
sig1=[sortt(distpertinence,sig1(1,:));sortt(distpertinence,sig1(2,:))];
dirac=[sortt(distpertinence,dirac(1,:));sortt(distpertinence,dirac(2,:))];
mc;
sig1;
dirac;

tag=X(:,1);
data=X(:,2:end);
n=size(data,1);
X;
%Y;



for i=1:n
    data(i,:)=sortt(distpertinence,data(i,:));
end;
cc=1;
%Ici on a pris les 10 caractéristiques les plus "pertinentes" mais on peut
%changer selon le score.
for h=0:1
for i=1:n
    lv(i,cc)=vrais(mc(cc,[1:10]),sig1(cc,[1:10]),dirac(cc,[1:10]),data(i,[1:10]));
end;

cc=cc+1;
end;

decision=((lv(:,2)-lv(:,1))>0);
score=sum(decision==tag);


function score=naivebayes62(X,Y)
% NAIVEBAYES
% Naive bayes classifier.
% X est la matrice pour apprentissage
% Y est la matrice sur laquelle on effectue le test



tag=X(:,1);
data=X(:,2:end);
p1=size(data,1);
p2=size(data,2);

sig0=10^(-5); % à revoir
for h=0:1
    cc=h+1;
    for i=1:p2
        nreal=sum((tag==h)&(data(:,i)~=Inf));
        if nreal==0
            mc(cc,i)=0;
            sig1(cc,i)=sig0;
```

```
        else
            mc(cc,i)=mean(data((tag==h)&(data(:,i)~=Inf),i));
            sig(cc,i)=std(data((tag==h)&(data(:,i)~=Inf),i));
            sig1(cc,i)=sqrt(((nreal-1)*sig(cc,i)^2+sig0)/(nreal+1));
        end
        A=data(:,i);
        dirac(cc,i)=(occurence(A(tag==h),Inf)+1)/(sum(tag==h)+2);
    end
end

%Ici on met un ordre de preference pour les caracteristiques
distpertinence=beest(mc,sig1,dirac);
mc=[sortt(distpertinence,mc(1,:));sortt(distpertinence,mc(2,:))];
sig1=[sortt(distpertinence,sig1(1,:));sortt(distpertinence,sig1(2,:))];
dirac=[sortt(distpertinence,dirac(1,:));sortt(distpertinence,dirac(2,:))];
mc;
sig1;
dirac;

tag=Y(:,1);
data=Y(:,2:end);
n=size(data,1);
X;
Y;



for i=1:n
    data(i,:)=sortt(distpertinence,data(i,:));
end;
cc=1;
%Ici on a pris les 10 caractéristiques les plus "pertinentes" mais on peut
%changer selon le score
for h=0:1
for i=1:n
    lv(i,cc)=vrais(mc(cc,[1:10]),sig1(cc,[1:10]),dirac(cc,[1:10]),data(i,[1:10]));
end;

cc=cc+1;
end;

decision=((lv(:,2)-lv(:,1))>0);
score=sum(decision==tag);

function [L,pop]=ccomp(A)
%
% [L,pop]=ccomp(A) calcule les composante connexe de A pour
% le systeme des quatre plus proches voisins.
%
% A est une entrÃľe binaire (bord -> 0, fond -> 1)
%
% L est la matrice des labels: L=-1 pour le fond, L=0 pour les bords
% L=1,2,... pour les autres composantes
% La structure pop code pour une population de shapes
% pop.ml nombre total de shapes
% pop.shapes listes des formes
% La structure shapes code les differentes composantes
```

```
% avec les champs suivants
%
%  aires: nb de pixels dans la composante
%  I coordoonÃľes i des pixels
%  J coordonnÃľes j des pixels
%  barI, barJ positions des barycentres
%  cov matrice de covariance
%  scale; sqrt(trace(cov))
%
% Exemple: shape(2).barI donne la coordoonÃľe i du barycentre
% de la composante (shape) 2

L=bwlabel(A,4);

L=L-1;
INTER=L;
ml=max(max(L));
if ml>2
L=seuil_aires(L);
%On élimine les petites formes
ml=max(max(L));
end;
if ml<2
    L=INTER;
    ml=max(max(L));
end;

% Calcul des aires par tailles dÃľcroissantes
tmp=zeros(1,ml);
for l=1:ml
    tmp(l)=sum(L(:)==l);
end
[taires,lab]=sort(tmp,'descend');
aires=cell(ml,1);
I=cell(ml,1);
J=cell(ml,1);

for i=1:length(lab)
    aires{i}=taires(i);
    [tI,tJ]=find(L==lab(i));
    I{i}=tI;
    J{i}=tJ;
end

shapes=struct('aires',aires,'I',I,'J',J);

for l=1:ml
    shapes(l).barI=mean(shapes(l).I(:));
    shapes(l).barJ=mean(shapes(l).J(:));
    shapes(l).cov=cov(shapes(l).I(:),shapes(l).J(:));
    shapes(l).scale=sqrt(det(shapes(l).cov));

end

pop.ml=ml;
pop.shapes=shapes;
```

```
function L=seuil_aires(M);
[a,b]=size(M);
n=max(max(M));
aire=[];
inter=0;
for i=1:n
   inter=sum(M(:)==i);
   aire=[aire,inter];
end;
aire;
maxair=max(aire);
v=[];
for k=1:n
    if ((maxair)/aire(k))>30
        v=[v,k];
    end;
end;
v;
p=length(v);
if p==0
    L=M;
else
for k=1:length(v)
    for i=1:a
        for j=1:b
            if M(i,j)==v(k)
                M(i,j)=0;
            end;
        end;
    end;
end;
end;
U=[];
for i=1:a
    for j=1:b
        if (M(i,j)>=1) &(not(exsiste(U,M(i,j))) | length(U)==0)
            U=[U,M(i,j)];
        end;

    end;
end;
A=zeros(a,b);
for k=1:length(U)
for i=1:a
    for j=1:b
        if (M(i,j)==U(k))&(A(i,j)==0)
            M(i,j)=k;
            A(i,j)=1;
        end;
    end;
end;
end;
L=M;


function y=fnum2str(x)
%
```

```
% fnum2str
%  Permet d'Ãl'crire les numÃl'ros de fichiers sous le format svrt

if x<10
    y=['000' num2str(x)];
elseif (x<100)
    y=['00' num2str(x)];
   elseif (x<1000)
       y=['0' num2str(x)];
else
    y=num2str(x);
end

%Important:le programme prend Imread de l'image comme argument
%u est le nombre de classes pour la relation d'equivalence"contact"
%v est le tableau des nombres de composantes connexes par classe trié par
%ordre croissant
%les nouv ss programmes utilisés
%sont:resum_contact,classcontact,transf,testbizarre,virezeros et enfin mx


function [u,v]=contact_final(Im)
A=resum_contact(Im);
B=transf(A);
n=size(B,1);
T=[];
k=0;
for i=1:n
    l=sum(B(i,:));
    T=[T,l];
    if testbizarre(B(i,:),zeros(1,n))==0
        k=k+1;
    end;
end;
T=sort(T);
T=virezeros(T);
v=T;
u=k;


function boul=exsiste(A,x)
n=length(A);
boul=0;
i=1;
while (boul==0) & (i<=n)
    if A(i)==x
        boul=1;
    end;
    i=i+1;
end;

function x=beest(mc,sig1,W)
% Estimation du nombre d'images bien classées en utilisant la loi de Bayes.
[p q]=size(mc);
N=1000;
Nberror=zeros(2,q);
for j=1:q
```

```
    for h=1:2
        B=(rand(1,N)<W(h,j));
        G=mc(h,j)+sig1(h,j)*randn(1,N);
        f1=B.*W(1,j)+...
            (1-B).*(exp(-(G-mc(1,j)).^2/(2*sig1(1,j)^2))*(1-W(1,j))/sqrt(2*pi*sig1(1,j)^2));
        f2=B.*W(2,j)+...
            (1-B).*(exp(-(G-mc(2,j)).^2/(2*sig1(2,j)^2))*(1-W(2,j))/sqrt(2*pi*sig1(2,j)^2));
        if h==1
            Nberror(h,j)=sum(f2>=f1);
        else
            Nberror(h,j)=sum(f1>f2);
        end
    end
end
x=2*N-sum(Nberror,1);

function M=classcontact(A);
n=size(A,1);
for i=1:n
    for j=1:n
        if A(i,j)>0
            A(i,:)=mx(A(i,:),A(j,:));
             A(j,:)=mx(A(i,:),A(j,:));
        end;
    end;
end;
M=A;

function dh=disth(s1,s2)

s=sqrtm(s1);
s=s^(-1);
s=s*s2*s;
s=s/sqrt(det(s));
E1=eig(s);

dh=sqrt(sum(log(E1).^2));

function dh=disth(s1,s2)

s=sqrtm(s1);
s=s^(-1);
s=s*s2*s;
s=s/sqrt(det(s));
E1=eig(s);

dh=sqrt(sum(log(E1).^2));

function dh=disth(s1,s2)

s=sqrtm(s1);
s=s^(-1);
s=s*s2*s;
s=s/sqrt(det(s));
E1=eig(s);

dh=sqrt(sum(log(E1).^2));
```

```
function dt=distt(s1,s2)


dt=sqrt(trace((s1-s2)^2));

function [bigA,bigO]=grossissement(A,O)

%Prend en argument des abscisses et des ordonnées de barycentre et y rajoute
%une épaisseur

testA=[];
testO=[];
for i=1:length(A)
    for j=1:1
    testA=[testA,A(i),A(i)+j];
    testO=[testO,O(i),O(i)+j];
    end
    %for j=2:
    %testA=[testA,A(i)-j,A(i)+j];
    %testO=[testO,O(i)-j,O(i)+j];
    %end
end

bigA=testA;
bigO=testO;

function y=indice(N,z)

%calcul l'indice de l'ellipse associÃľe Ãă la matrice M au point z


[V,D]=eig(N);
f=@(t) (-D(1,1).*sin(t)+i.*D(2,2).*cos(t))./(D(1,1).*cos(t)+i.*D(2,2).*sin(t)-z);

y=real(1/(2*i*pi)*quadl(f,0,2*pi,10^(-10)));


end

%prend en argument deux vecteur et renvoie le vecteur des maximums des
%composantes (une par une)
function u=mx(A,B)
n=length(A);
for i=1:n
    u(i)=max(A(i),B(i));
end;


%occurence (A,y) renvoie le nombre de fois où y apparait dans A
(%le vecteur A)

function x=occurence(A,y)
k=0;
n=length(A);
for i=1:n
    if A(i)==y
```

```
        k=k+1;
    end;
end;
x=k;

%resum_contact prend en argument l'image et donne le résumé de tous le
%contact qui y figure,on numérote les composantes connexes de 1 jusqu'à n
%on construit la matrice Tab de la manière suivante:
%si il y a contact entre la comosante "i" et la composante "j" alors
%Tab(i,j) et Tab(j,i) reçoit 1,sinon la case reste à zero(c'est à dire il
%n'y a pas de contact entre les deux formes

function Tab=resum_contact(Im)
[l,pop]=ccomp_2(Im(:,:,1));
n=max(max(l));
M=eye(n);
n1=size(Im,1);
for i=5:n1-4
    for j=5:n1-4
        K=[l(i-3,j),l(i-2,j),l(i-1,j),l(i,j),l(i+1,j),l(i+2,j),l(i+3,j),l(i,j-3),l(i,j-2),l(i,j-1),l
        for p1=1:length(K)
            for p2=1:length(K)
                if (K(p1)>0) & (K(p2)>0)
                    M(K(p1),K(p2))=1;
                     M(K(p2),K(p1))=1;
                end;
            end;
        end;
    end;
end;
Tab=classcontact(M);

%sortt(X,Y) renvoie Y trié selon l'ordre descendant pour X exemple
%sortt([1 2 3],[0 5 9]) renvoie [9 5 0]



function v=sortt(X,Y)
n=length(X);
interx=0;
intery=Y(1);
if length(X)~=length(Y)
    v=Y;
else
for i=1:n
    for j=i:n
        if X(i)<X(j)
            interx=X(j);
            X(j)=X(i);
            X(i)=interx;
            intery=Y(j);
            Y(j)=Y(i);
            Y(i)=intery;
        end
    end
end
v=Y;
```

```
end


%vérifie si deux vecteurs sont égaux
function x=testbizarre(A,B)
n=length(A);
x=1;
k=1;
while (x==1)&(k<=n)
    if A(k)~=B(k)
        x=0;
    end;
     k=k+1;
end;
x;

%cette fonction transforme la matrice M de la manière suivante dès qu'elle
%trouve un ligne de la matrice identique à une ligne qui a été rencontrée
%avant elle met le vecteur nul à cette lligne

function u=transf(M)
n=size(M,1);
for i=1:n
    for j=1:n
        if testbizarre(M(i,:),M(j,:))==1 & (j>i)
           M(j,:)=zeros(1,n);
        end;
    end;
end;
u=M;

%enlève les zeros du début du vecteur A
%ce qui revient à enlever les zéros tout court puisque notre vecteur est
%supposé trié
function u=virezeros(A);
k=A(1);
while (k==0)|(length(A)>1)
    A=A(2:length(A));
    k=A(1);
end;
u=A;

%Somme_log renvoie lv(i,cc) avec A=moyenne des gaussiennes de la classe h,B
%variances de la même classe et C les probas d'apparition de "Inf" dans la
%classe h,enfin x est le vecteur de caractéristiques correspondant à
%l'image i

function somme_log=vrais(A,B,C,x)
n=length(A);
u=zeros(1,n);
for i=1:n
    if (x(i)==Inf) |(A(i)==Inf) |(B(i)==Inf)
        u(i)=log(C(i));
    else

        u(i)=log(1-C(i))-log(sqrt(2*pi*B(i)))-(x(i)-A(i))^2/(2*(B(i)^2));
        end;
```

```
end;

somme_log=sum(u);
```

Bibliographie

[1] Sylvestre Gallot, Dominique Hulin, and Jacques Lafontaine. *Riemannian geometry.* Springer-Verlag, 2004.

[2] Jacques Lafontaine. *Introduction aux variétés différentielles.* Presses universitaires de Grenoble, 1997.

[3] Yadolah Dodge and Giuseppe Melfi. *Premiers pas en simulation.* Springer-Verlag, 2008.

[4] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *International Journal of Computer Vision*, 2005.

[5] Frédéric Helein. Calcul différentiel et géométrie différentielle. 2002.

[6] *Synthetic visual reasoning test challenge.* www.idiap.ch/ fleuret/svrt/challenge.html.

[7] Donald Geman. Statistical model for images. Talk in Luminy, France, 2010.