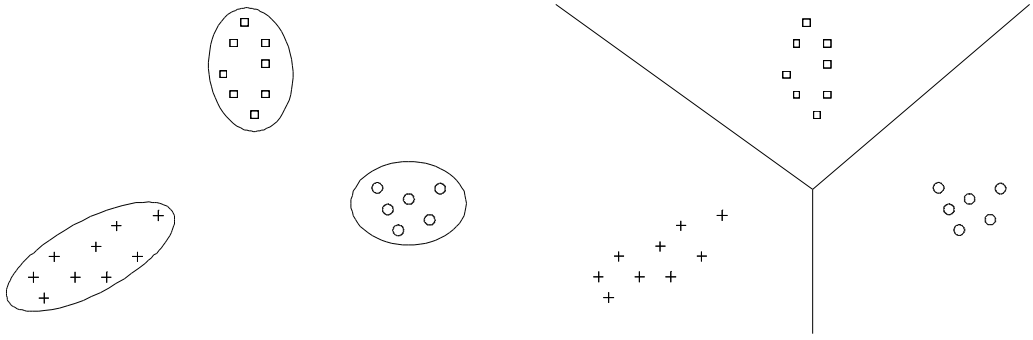
Reconnaissance des formes, Sys800

- *Laboratoire 2 : Algorithmes de classification*-

Lors du premier laboratoire, vous avez dans un premier temps extrait des caractéristiques des images de chiffres en utilisant le codage rétinien. Vous avez ensuite réduit la dimension des vecteurs de caractéristiques. Dans le cadre de ce second laboratoire, vous devrez implémenter, tester et comparer différents algorithmes de classification.

Notons tout d'abord que parmi l'ensemble des techniques de classification, il est possible de distinguer deux catégories d'approches, celles agissant par modélisation et celles agissant par séparation (voir figure 1).



(a) par modélisation (b) par séparation

Figure 1 : Deux catégories d'algorithmes de classification

L'objectif du premier type d'approche (Figure 1-a) est de déterminer un modèle le plus fidèle possible de chacune des classes, alors que le second type (Figure 1-b) cherche à optimiser des frontières de décision de manière à séparer au mieux les classes. La décision est alors prise dans le premier cas en utilisant une mesure de similarité pour comparer la donnée à classifier à chacun des modèles et dans le second cas en se basant sur la position de la donnée par rapport aux frontières.

Dans le cadre de ce laboratoire, trois algorithmes de classification seront abordés, deux approches agissant par modélisation (Bayes Quadratique et *k*-PPV) et une agissant par séparation (SVM).

1 – BAYES QUADRATIQUE

Il est classique en reconnaissance de formes d'utiliser des approches paramétriques qui consistent à faire une hypothèse sur la nature de la distribution des données puis à estimer les paramètres de la distribution en utilisant les données d'apprentissage. Ainsi, le classifieur Bayes Quadratique est basé sur l'hypothèse que les données de chacune des classes suivent une distribution normale multi-variable :

$$p(x \mid \omega_j) = \frac{1}{(2\pi)^{d/2} \mid \Sigma_j \mid^{1/2}} \exp\left(-\frac{1}{2} \left(x - \mu_j\right)^T \Sigma_j^{-1} \left(x - \mu_j\right) \right), \tag{1}$$

où Σ_j et μ_j sont la matrice de covariance et le vecteur moyenne des données de la classe ω_j et d la dimension des vecteurs de caractéristiques.

La phase d'apprentissage consiste alors à utiliser les données d'apprentissage afin d'estimer pour chacune des classes, la moyenne et la matrice de covariance. Ces paramètres seront ensuite sauvegardés de manière à pouvoir être utilisés par la suite pour classifier les exemples de test en utilisant la règle de Bayes afin d'estimer les probabilités *a posteriori* :

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

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

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

où c représente le nombre de classes du problème.

En pratique, il est préférable de calculer les fonctions discriminantes associées à chacune des classes qui sont obtenues en prenant le logarithme naturel de l'équation (2) et en supprimant les termes constants :

$$d_j(x) = \ln \hat{P}(\omega_j) - \frac{1}{2}\ln\left(\left|\hat{\Sigma}_j\right|\right) - \frac{1}{2}\left(x - \hat{\mu}_j\right)^T \hat{\Sigma}_j^{-1}\left(x - \hat{\mu}_j\right). \quad (4)$$

Notons que dans notre cas, les dix classes sont équiprobables ; c'est-à-dire que toutes les classes ont la même probabilité *a priori* : $\hat{P}(\omega_j) = 1/c$. Le premier terme de l'équation (4) n'intervient donc pas dans la décision et peut donc être supprimé.

Par ailleurs, il est intéressant de constater que le dernier terme de l'équation (4) représente la distance de Mahalanobis entre l'exemple à classifier et la distribution de la classe ω_j.

Enfin, la décision est prise en cherchant la valeur maximale parmi les fonctions discriminantes associées à chacune des classes.

2 – K - PLUS PROCHES VOISINS

Afin d'éviter de devoir faire une hypothèse sur la nature de la distribution des données, il est possible d'utiliser une méthode non-paramétrique telle que l'algorithme des k-plus proches voisins (k-PPV).

L'ensemble des données d'apprentissage forme alors un modèle des différentes classes et pour classer une donnée de test, il suffit de calculer la distance entre cette donnée et l'ensemble des données d'apprentissage, puis d'effectuer un vote majoritaire parmi les *k* données les plus proches, chacune des données votant pour la classe à laquelle elle appartient. Le seul paramètre qu'il est nécessaire de fixer est donc le nombre *k* de voisins

à considérer pour prendre la décision. La mesure de distance la plus couramment utilisée est la distance euclidienne :

$$d_i = (x - x_i)^T (x - x_i) \quad (i = 1,2,\ldots,n), \tag{5}$$

où x est la donnée à classifier et x_i l'un des n exemples de la base d'apprentissage.

3 – MACHINES À VECTEURS DE SUPPORT

Une machine à vecteurs de support (SVM de l'anglais Support Vector Machine) est un classifieur binaire dont l'objectif est de déterminer la frontière « optimale » séparant les données d'apprentissage des deux classes. Cette frontière est définie par :

$$\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + b = 0, \tag{6}$$

où les $y_i \in \{1,-1\}$ correspondent aux étiquettes des données d'apprentissage x_i et K est un noyau de Mercer qui permet de projeter les données dans un espace éventuellement plus grand dans lequel la séparation linéaire des classes est possible.

La phase d'apprentissage consiste à déterminer les coefficients α_i et le biais b qui maximisent la marge de séparation ; c'est-à-dire la distance entre les données de la classe positive et les données de la classe négative les plus proches de la frontière de séparation. L'apprentissage résout un problème d'optimisation équivalent à l'équation ci-après :

$$\max_{\alpha}\left[\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i,x_j)\right] \text{ avec } \sum_{i=1}^{n} y_i\alpha_i = 0 \text{ et } 0 \le \alpha_i \le C, \quad i = 1,...,n \qquad (7)$$

Les données d'apprentissage x_i associées à des coefficients α_i non nuls sont nommées **vecteurs de support**. Pour plus d'information sur les SVMs, nous vous conseillons de lire le document intitulé « Classifieurs à noyaux et SVM»[1].

La sortie d'un SVM sera donc définie par :

$$f(x) = \sum_{i=1}^{m} y_i\alpha_i K(x_i,x) + b \qquad (8)$$

et la décision sera prise en considérant le signe de celle-ci.

Si l'on utilise un noyau linéaire $K(x_i,x) = x_i \cdot x$, la frontière de séparation est alors un hyperplan défini par $w \cdot x + b = 0$ avec $w = \sum_{i=1}^{n} y_i\alpha_i x_i$. Un exemple en deux dimensions est présenté figure 2, où les données d'apprentissage marquées d'un cercle correspondent aux vecteurs de support.

---

[1] **Extrait de** : Mathias M. Adankon, *"Optimisation de ressources pour la sélection de modèle des SVM "*, Mémoire de Master, École de Technologie Supérieure, Montréal, Septembre 2005

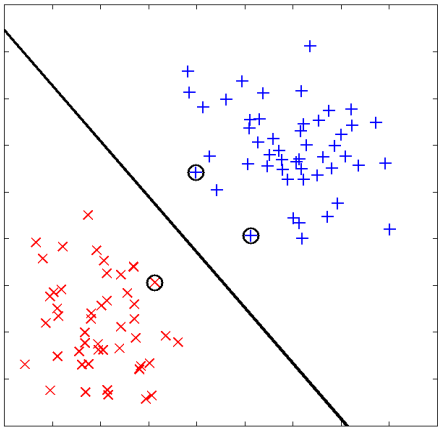
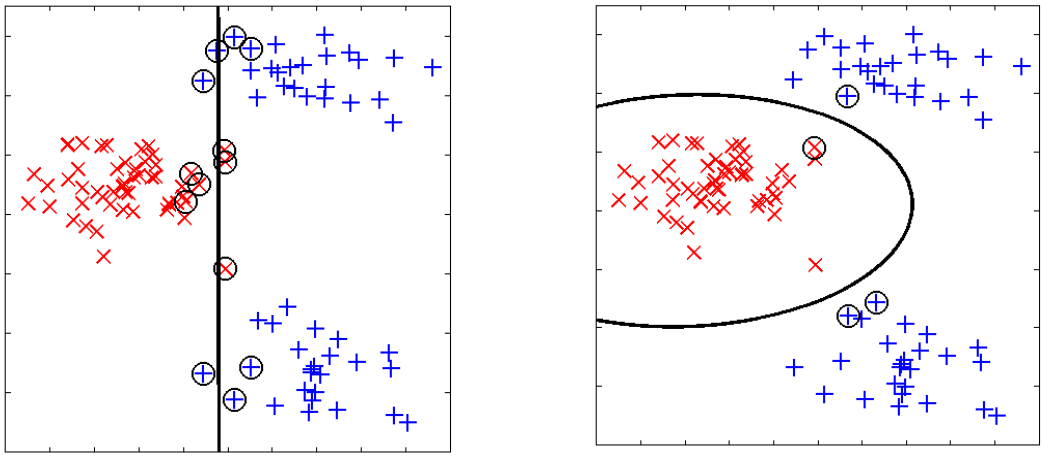


Figure 2 : Exemple de frontière de décision obtenue à l'aide d'une SVM linéaire

Or, dans le cas d'applications réelles, les données ne seront malheureusement pas toujours linéairement séparables. Il sera alors nécessaire d'utiliser d'autres types de noyaux, tels que le noyau polynomial $K(x_i, x_j) = (ax_i.x_j + b)^d$, le noyau gaussien $K(x_i, x_j) = \exp\left(-\gamma \left\| x_i - x_j \right\|^2\right)$. Une illustration de l'utilité du noyau polynomial est présentée en figure 3. Les données ne peuvent alors pas être séparées par une frontière linéaire, mais l'utilisation d'un noyau polynomial d'ordre 2 permet de déterminer une frontière de séparation non-linéaire bien plus adaptée.



(a) (b)

Figure 3 : Frontières de séparation obtenues par une SVM linéaire (a) et non-linéaire (b)

Par ailleurs, dans la majorité des problèmes de reconnaissance de formes, le nombre de classes est supérieur à deux. Il est alors nécessaire de décomposer le problème de classification en plusieurs sous-problèmes binaires qui pourront être résolus par différents SVM. La stratégie la plus simple à mettre en place consiste à construire autant de SVM qu'il y a de classes. Chaque SVM est alors entraîné à distinguer les exemples d'une classe à ceux de toutes les autres classes. On parle de stratégie « un contre tous » et la décision est prise en cherchant la valeur maximale parmi les sorties des différents SVM.
En pratique, la fonction MATLAB permettant l'apprentissage d'un SVM vous est fournie. Le script fourni en annexe 1 vous permettra d'en comprendre l'utilisation. Aussi, vous trouverez en annexe 2 les différentes options à préciser pour utiliser la fonction d'apprentissage. Dans le cadre de ce laboratoire, vous devrez donc utiliser cette fonction et implémenter la fonction de test.

RÔLE DES DIFFÉRENTES BASES DE DONNÉES

Les données d'apprentissage seront utilisées pour construire les classifieurs, alors que les données de test permettront d'évaluer leur capacité à généraliser. En effet, il est fréquent que le taux d'erreur évalué sur la base d'apprentissage soit bien plus faible que celui évalué en utilisant d'autres données. Ce phénomène de surapprentissage des données justifie donc le fait d'utiliser une seconde base de données pour évaluer la capacité à généraliser d'un classifieur. D'autre part, de manière à être rigoureux, aucun paramètre du classifieur ne devra être fixé en tenant compte des résultats obtenus sur la base de test. Ainsi, dans certains cas, il peut être nécessaire de diviser la base d'apprentissage en deux sous-ensembles. Une partie des données sera alors utilisée pour l'entraînement des classifieurs alors que le reste sera utilisé comme base de validation pour optimiser certains paramètres du classifieur, tels que le nombre k de voisins du k-PPV ou les hyperparamètres (paramètres du noyau et C) du SVM. Ce second sous-ensemble est appelé base de validation et est généralement constitué d'un tiers des données de la base d'apprentissage.

CRITÈRES DE COMPARAISON DES CLASSIFIEURS

Le critère principal de comparaison des différents algorithmes de classification sera donc la capacité à généraliser qui sera évaluée en mesurant le taux d'erreur sur la base de test. Néanmoins, d'autres critères devront être pris en considération. En effet, selon l'application visée, la complexité de calcul nécessaire à la prise de décision, la capacité de mémoire nécessaire pour stocker le classifieur ou encore la complexité de mise en place de l'algorithme (apprentissage et optimisation) peuvent être des critères primordiaux. Enfin, il pourra être intéressant de comparer le comportement de chaque classifieur par rapport à des contraintes telles que le nombre d'exemples d'apprentissage.

Travail à faire

Toutes les questions doivent êtres traitées en considérant les caractéristiques :

- La technique de rétine avec ACP (RetineACP)

1. Coder l'algorithme d'apprentissage du Bayes Quadratique ainsi que la fonction de test pour évaluer le taux d'erreur.
    a. Dresser la matrice de confusion et faites **l'analyse appropriée**, d'une manière quantitative et supportez votre analyse des erreurs par des exemples de chiffres.

2. Mettre en œuvre le classifieur K-PPV.
    a. Utiliser 1/3 de la base d'apprentissage comme base de validation pour fixer le nombre *k* de voisins.
    b. Tracer un graphe représentant les erreurs d'entrainement en fonction de *k*.
    c. Utiliser la totalité de la base d'apprentissage avec le nombre *k* trouvé en (a) pour évaluer le taux d'erreur sur la base de test.
    d. Dresser la matrice de confusion et faites **l'analyse appropriée**, d'une manière quantitative et supportez votre analyse des erreurs par des exemples de chiffres.
    e. Trouvez le k optimal pour chacune des classes.

3. Coder le script pour l'apprentissage des SVMs en utilisant la stratégie «un contre tous» puis évaluer la performance en test. Pour ce problème, il est suggéré d'utiliser le noyau gaussien $K(x_i, x_j) = \exp\left(-\gamma \left\| x_i - x_j \right\|^2\right)$.
    a. Utiliser 1/3 de la base d'apprentissage comme base de validation pour sélectionner les hyperparamètres C et γ optimaux.
    b. Utiliser les valeurs de C et de γ déterminées pour refaire l'apprentissage avec la totalité de la base d'apprentissage puis évaluer le taux d'erreur en test.
    c. Tracer un graphe (3D) de l'erreur de test en fonction de C et γ et faites votre analyse.

4. Comparer les trois classifieurs suivant: la capacité de généralisation, le temps d'apprentissage, le temps de la phase de test, la capacité de mémoire de stockage des paramètres du modèle.

5. **Effet du bruit sur la performance de classification**

    Un script `AddNoiseToBWImage.m` qui permet d'ajouter du bruit aux images des chiffres est décrit en Annexe 3.
    Étudier la performance et analyser la robustesse de chaque classifieur (K-PVV, SVM et BayesQuadratique) selon la variance du bruit ajouté.

ANNEXE 1

demo_svm.m

```
%Demo sur l'utilisation de la fonction mexsvmlearn.dll
%conçu par Tom Briggs, interface matlab et SVMlight
%SVMlight a été developpé par  Thorsten Joachims en C.

fprintf('*** Formation de la base de donnée *** \n');
X=[1 2;2 1;3 2;3 3;2 4;4 7;5 2;5 4;7 2;7 4];
Y=[1;1;1;1;1;-1;-1;-1;-1;-1];
fprintf('*** Apprentissage du SVM *** \n');
model = mexsvmlearn(X,Y,'-t 0 -c 0.5');

fprintf('*** Affichage des valeurs de alpha avec signe de Yi *** \n');
Yalpha=model.a

fprintf('*** Affichage de la valeur du biais  *** \n');
biais=-model.b

fprintf('*** Affichage de l indice des vecteurs de support *** \n');
index_vs = find(Yalpha~=0)

fprintf('*** Dessin de la frontière de décision ***\n');
%Récupérer les vecteurs de support
VS=X(index_vs,:);
%Afficher les données d'apprentissage
plot(X(1:5,1),X(1:5,2),'+',X(6:10,1),X(6:10,2),'.');
%Encercler en rouge les vecteurs de support
hold on
plot(VS(:,1),VS(:,2),'or');
%Calculer les paramètres de la droite de sépapration a1.x+a2.y+bias=0
a1=Yalpha(index_vs)'*X(index_vs,1)
a2=Yalpha(index_vs)'*X(index_vs,2)
%Tracer la droite de séparation
hold on
plot([2.5 4.7],[-(biais+2.5*a1)/a2 -(biais+4.7*a1)/a2]);
%Tracer la marge de séparation a1.x+a2.y+bias=-1
hold on
plot([2 4],[-(-1+biais+2*a1)/a2 -(-1+biais+4*a1)/a2]);
%Tracer la marge de séparation a1.x+a2.y+bias=1
hold on
plot([3 5.5],[-(1+biais+3*a1)/a2 -(1+biais+5.5*a1)/a2]);

%Trouver la classe d'un exemple représenté par Xt1=(6,7)
Xt1=[6,7];
Kt1 = X(index_vs,:)*Xt1'  %Noyau lineaire
Yt1 = sign(Yalpha(index_vs)'*Kt1+biais)

%Trouver la classe d'un exemple représenté par Xt2=(3,0)
Xt2=[3,0];
Kt2 = X(index_vs,:)*Xt2'  %Noyau lineaire
```

```
Yt2 = sign(Yalpha(index_vs)'*Kt2+biais)
```

Résulat de demo_svm.m

>> demo_svm

*** Formation de la base de donnée ***

*** Apprentissage du SVM ***

Optimizing.....done. (15 iterations)

Optimization finished (0 misclassified, maxdiff=0.00066).

Runtime in cpu-seconds: 0.00

Number of SV: 4 (including 1 at upper bound)

L1 loss: loss=0.19987

Norm of weight vector: |w|=1.01978

Norm of longest example vector: |x|=8.06226

Estimated VCdim of classifier: VCdim<=68.59656

Computing XiAlpha-estimates...done

Runtime for XiAlpha-estimates in cpu-seconds: 0.00

XiAlpha-estimate of the error: error<=40.00% (rho=1.00,depth=0)

XiAlpha-estimate of the recall: recall=>60.00% (rho=1.00,depth=0)

XiAlpha-estimate of the precision: precision=>60.00% (rho=1.00,depth=0)

Number of kernel evaluations: 206

Skipping lin_weights (array is empty)

Clearing 68 un-freed( ) memory blocks

---------- | memory cleaner statistics | ----------

Blocks allocated: 115

Blocks freed: 115

Block double-frees prevented: 0

Hash bucket collisions: 0

List traversal steps: 0

Collision Rate: 0.000

Average list depth: 0.000

*** Affichage des valeurs de alpha avec signe de Yi ***

Yalpha =

0

0

0.0700

0.5000

0

-0.1400

-0.4300

0

0

0

*** Affichage de la valeur du biais ***

biais =

4.3997

*** Affichage de l indice des vecteurs de support ***

index_vs =

3

4

6

7

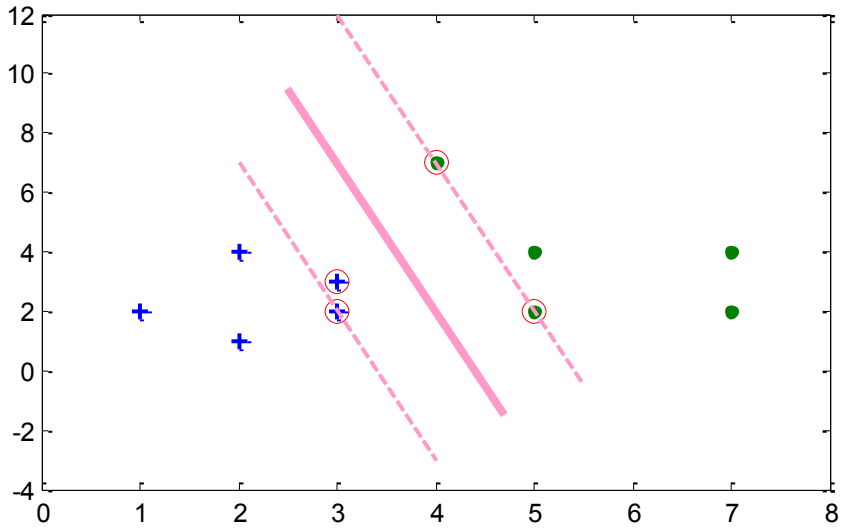
*** Dessin de la frontière de décision ***

a1 =

-1.0000

a2 =

-0.1999



Kt1 =

32

39

73

44

Yt1 =

-1

Kt2 =

9

9

12

15

Yt2 =

1

ANNEXE 2

Available options are:

```
General options:
         -?           - this help
         -v [0..3]    - verbosity level (default 1)
Learning options:
         -z {c,r,p}   - select between classification (c), regression (r),
and
                        preference ranking (p) (see [Joachims, 2002c])
                        (default classification)
         -c float     - C: trade-off between training error
                        and margin (default [avg. x*x]^-1)
         -w [0..]     - epsilon width of tube for regression
                        (default 0.1)
         -j float     - Cost: cost-factor, by which training errors on
                        positive examples outweight errors on negative
                        examples (default 1) (see [Morik et al., 1999])
         -b [0,1]     - use biased hyperplane (i.e. x*w+b0) instead
                        of unbiased hyperplane (i.e. x*w0) (default 1)
         -i [0,1]     - remove inconsistent training examples
                        and retrain (default 0)


Performance estimation options:
         -x [0,1]     - compute leave-one-out estimates (default 0)
                        (see [5])
         -o ]0..2]    - value of rho for XiAlpha-estimator and for pruning
                        leave-one-out computation (default 1.0)
                        (see [Joachims, 2002a])
         -k [0..100] - search depth for extended XiAlpha-estimator
                        (default 0)
Transduction options (see [Joachims, 1999c], [Joachims, 2002a]):
         -p [0..1]    - fraction of unlabeled examples to be classified
                        into the positive class (default is the ratio of
                        positive and negative examples in the training
data)



Kernel options:
         -t int       - type of kernel function:
                         0: linear (default)
                         1: polynomial (s a*b+c)^d
                         2: radial basis function exp(-gamma ||a-b||^2)
                         3: sigmoid tanh(s a*b + c)
                         4: user defined kernel from kernel.h
         -d int       - parameter d in polynomial kernel
         -g float     - parameter gamma in rbf kernel
         -s float     - parameter s in sigmoid/poly kernel
         -r float     - parameter c in sigmoid/poly kernel
         -u string    - parameter of user defined kernel
```

```
Optimization options (see [Joachims, 1999a], [Joachims, 2002a]):
         -q [2..]    - maximum size of QP-subproblems (default 10)
         -n [2..q]   - number of new variables entering the working set
                       in each iteration (default n = q). Set n<q to
prevent
                       zig-zagging.
         -m [5..]    - size of cache for kernel evaluations in MB
(default 40)
                       The larger the faster...
         -e float    - eps: Allow that error for termination criterion
                       [y [w*x+b] - 1] = eps (default 0.001)
         -h [5..]    - number of iterations a variable needs to be
                       optimal before considered for shrinking (default
100)
         -f [0,1]    - do final optimality check for variables removed by
                       shrinking. Although this test is usually positive,
there
                       is no guarantee that the optimum was found if the
test is
                       omitted. (default 1)
         -y string   -> if option is given, reads alphas from file with
given
                        and uses them as starting point. (default
'disabled')
         -# int      -> terminate optimization, if no progress after this
                        number of iterations. (default 100000)
Output options:
         -l char     - file to write predicted labels of unlabeled
examples
                       into after transductive learning
         -a char     - write all alphas to this file after learning (in
the
                       same order as in the training set)
```

ANNEXE 3

% add noise

```
function N = AddNoiseToBWImage(u,v)
% u : input image
% v : variance of the noise
i = u;
x = rand(size(i));
d = find(x < v/2);
i(d) = 0;
d = find(x >= v/2 & x < v);
i(d) = 1;
N = i;
```