Université de lorraine

Ecole nationale supérieure d'électricité et mécanique

Apprentissage automatique

Compte rendu du mini-projet

Akli Yahya

Sommaire :

I- Introduction

Le sujet des machines pensantes préoccupe les esprits depuis des années, ce concept a conduit au développement de plusieurs théories qui ont permets d'introduire la notion de l'intelligence artificielle ainsi que ses branches. L'apprentissage automatique est l'un de ses branches. En général, l'objectif de l'apprentissage automatique est de comprendre la structure des données et de les intégrer dans des modèles qui peuvent être comprises et utilisés par des machines. Dans ce projet, nous allons se contenter d'élaborer des algorithmes d'apprentissage automatique qui permettrons à la machine d'analyser et apprendre les caractéristiques d'un certain nombre de feuilles d'arbres. Dans ce cadre, nous allons étudier trois types d'algorithmes. Premièrement, on a des algorithmes d'apprentissage supervisé, pour lesquelles l'apprentissage de l'ensemble des caractéristiques se base sur un ensemble de connaissances sur les données d'apprentissage. En effet, cette catégorie d'algorithme consiste à apprendre une fonction de prédiction à partir d'exemples annotés. On appelle les problèmes de prédictions des problèmes de classifications. Ensuite, on a des algorithmes non-supervisé. Ces algorithmes concernent l'étude d'un ensemble de données non étiqueté, il s'agit donc de détecter les différentes structures et groupes ou classes de données que nous pourrons avoir dans l'ensemble d'entrée sans avoir des connaissances sur les données d'apprentissage. Finalement, nous allons passer aux réseaux de neurones qui sont des systèmes dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques. Il s'agit en général d'une succession de couches dont chacune prend ses entrées sur les sorties de la précédente, à chaque couche est associe un ensemble de poids que nous optimisons à chaque itération afin d'aboutir finalement à une succession de couches qui permettra par la suite d'identifier des données.

II- Apprentissage supervisé

Pour cette première approche, on dispose d'un ensemble d'entrainement D = $\{X_i, L_i\}$ tel que X_i est les caractéristiques de chaque élément i de label L_i. Nous allons considérer 4 classes de feuilles d'arbre : papaya, pimento, chrysanthemum, chocolate_tree. Au début, nous allons extraire les caractéristiques de ces feuilles, et les stocker dans une matrice X. on obtient la répartition de données affiché dans la figure 1. Ensuite, nous allons réduire la dimension de l'espace vectorielle des données d'entrée en faisant une analyse de la composante principale. Cette étape a pour but d'établir un résumé descriptif de ces données qui soit appréhendable et exploitable par l'analyse tout en entrainant une perte d'information minimale. On va donc chercher à ajuster le nuage des n points de la figure 1 par un sous-espace vectoriel de dimension $p \leq n$. On prend comme nouvelle origine le centre de gravité du nuage de points $X = X - mean(X)$, vu que l'espace affine passant par celui-ci rend mieux compte des proximités entre les points que l'espace vectoriel passant par l'origine du repère absolue.

Ensuite on calcul la nouvelle matrice de données Xp qui est la projection de X sur le vecteur propre des valeurs propre de la matrice de covariance et on affiche le nouveau nuage de points (figure 2)

```
covV = X'*X;
[U,D,V] = eig(covV);
[d,Isort] = sort(diag(D),'desc');
V = V(:,Isort);
Xp = X*V;
```

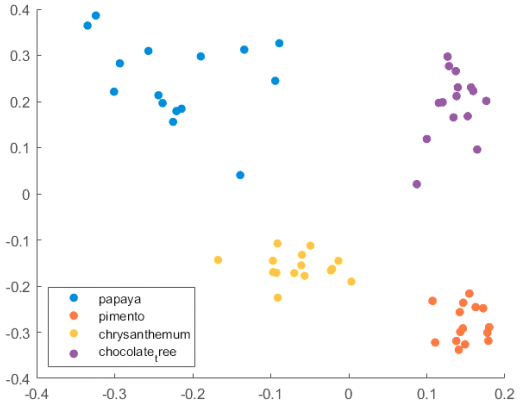


Figure 1 – Nuage de points avant ACP

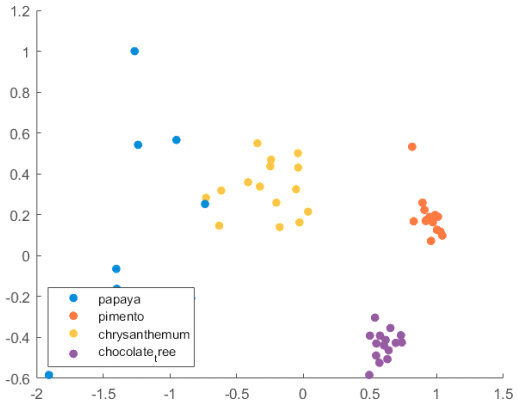


Figure 2 – Nuage de points après ACP

Passons maintenant aux algorithmes d'apprentissage supervisé.

a. Méthode non paramétrique (noyau de Parzen)

Pour cette première méthode on ne fait pas d'hypothèses sur le modèle que suivent les données, l'objectif est de trouver des propriétés de convergence et de consistance quand le nombre de données tend vers l'infini. Il s'agit d'une méthode d'estimation de la densité de probabilité des variables aléatoire X_p, en utilisant l'estimateur non-paramétrique qui est la somme de contributions de noyaux centrés autour de chaque échantillon observé : $f(x) = \frac{1}{Nv}\sum_{i=1}^{N} K\left(\frac{x-x_i}{v}\right)$. On adopte un noyau gaussien $K = \frac{1}{(2\pi)^{\frac{d}{2}}} \exp\left(-\frac{||x||^2}{2}\right)$. Ceci nous permet d'obtenir les probabilités de vraisemblance de chaque classe sur l'ensemble d'apprentissage.

On définit la fonction *gaussParzen*$(X, X_k, sigma)$ sur Matlab qui permet de calculer cette densité de probabilité, cette méthode est utilisé par la méthode isoContourParzen qui visualise les isocontours des classes d'apprentissage en les superposant sur l'ensemble des données.

On fixe l'écart-type du noyau gaussien $\sigma = 0.5$ et on considère que le nombre de composantes retenu est 3. Sachant que le nombre de classes est 4, on alors 4 matrices d'apprentissage X_{pi} de taille 15x3.

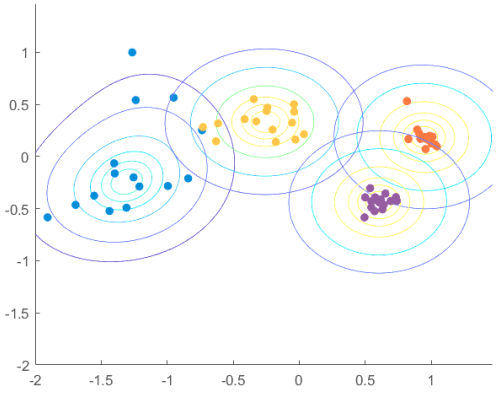


Figure 3 – les iso contours des 4 classes de feuilles

Dans la figure précédente, on remarque que l'algorithme de classification par maximum de vraisemblance classifie parfaitement les 4 classes. Cette première phase est la phase d'apprentissage. L'algorithme apprend les caractéristiques des différentes classes afin de pouvoir identifier d'autre données d'entrée qui seront dans ce cas des données de tests, permettant de vérifier la validité de ce classifieur. On appel la matrice des variables de test X_{test} et la matrice projetée sur le vecteur des valeurs propre X_{Ptest}. On utilise la fonction gaussParzen pour calculer la probabilité de la vraisemblance de chaque vecteur des caractéristiques de la matrice X_{Ptest}. Finalement, on trace les points corresponds aux éléments de tests de la figure précédente et on obtient :

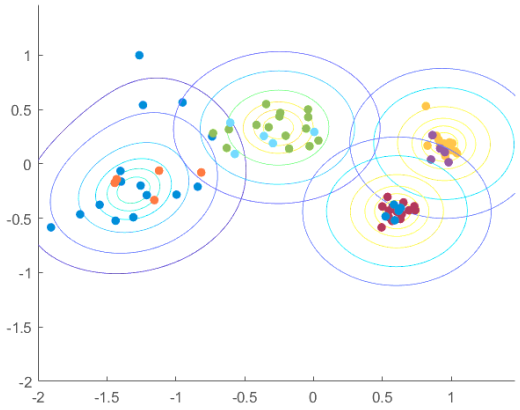


Figure 4 – Projections des points de la matrice X_{Ptest}.

On stocke les probabilités de vraisemblance dans une matrice Vsmb de taille 4x20 pour pouvoir calculer les probabilités P_{ei} d'erreur de classification qui est La probabilité d'erreur d'avoir classé x dans ωi alors que l'état de nature est $\omega_{i\neq j}$: $P_{ei} = \sum_{j=1,j\neq i}^{K} \int p(x|w_i)p(w_i)dx$. On suppose que les classes sont équiprobable. Alors $p(w_i) = \frac{1}{4}$

On obtient :

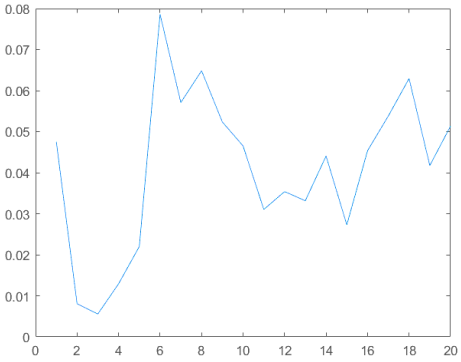


Figure 5 – L'évolution de l'erreur de classification (approche non-paramétrique)

En moyenne, la probabilité de l'erreur est relativement faible, donc on a un bon classifieur.

b. Approche paramétrique

Passons maintenant à l'approche paramétrique. Dans cette approche, on suppose que l'on connaît la forme du modèle qui a généré les données. On estime alors les paramètres à partir de l'ensemble d'apprentissage. Autrement dit, les vraisemblances des classes ont des formes connues et se résument à un nombre fini de paramètres θ. On dispose de 15 observation issue de chaque classe ω_k et on veut estimer les paramètres θ_k de chaque classe. On suppose que les variables suivent des lois gaussiennes de paramètres μ_k, Σ_k, tel que :

$$\theta_k = \{\mu_k, \Sigma_k\}, \qquad \mu_k = \frac{1}{N_k}\sum_{n=1}^{N_k} x_n, \quad \Sigma_k = \frac{1}{N_k}\sum_{n=1}^{N_k}(x_n - \mu_k)'(x_n - \mu_k)$$

On utilise les fonctions mean() et cov() de Matlab pour calculer respectivement les moyennes et covariance des variables. Finalement on utilise la fonction isoContourGauss() pour afficher les isocontours des classes. (Figure 6).

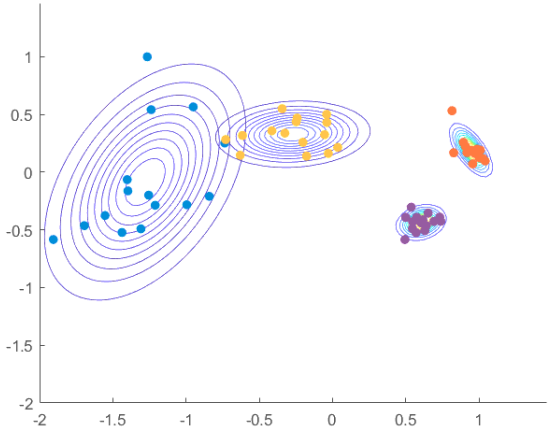


Figure 6 – Classification des feuilles par l'approche paramétrique

Finalement, on calcule les probabilités de vraisemblance des éléments de l'ensemble du test par rapport aux 4 classes et les probabilités d'erreur de classification. On obtient :

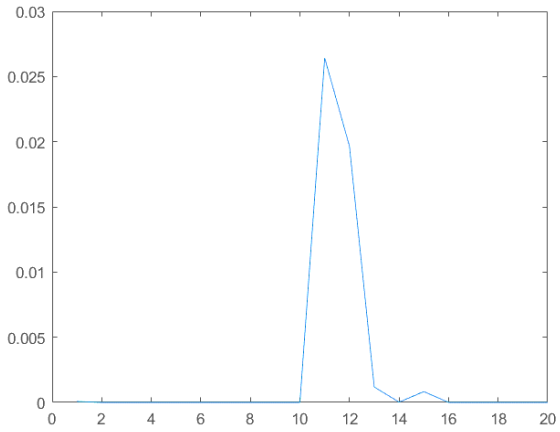


Figure 7 – L'évolution de l'erreur de classification (approche paramétrique)

L'erreur globale étant égale à 0.0482, l'algorithme classifie bien les éléments de tests.

III- Apprentissage non supervisé

A la différence des algorithmes d'apprentissage supervisé, l'apprentissage non supervisé se fait de façon totalement autonome. Les algorithmes de ce type opèrent sur un ensemble de données non annotés. L'objectif est de trouver des formes intéressantes dans les données. Il s'agit d'extraire des classes ou groupes d'individus présentant des caractéristiques communes. La qualité d'une méthode de classification est mesurée par sa capacité à découvrir certains ou tous les motifs cachés. Dans ce cadre, nous allons étudier deux types d'algorithmes : K-means et EM.

On considère que l'ensemble d'apprentissage D=$\{X_i\}$ avec X_i les vecteurs de la matrice X utilisé dans la partie précédente. On rappelle que la matrice de dimension réduite est notée X_p.

a. Algorithme k-moyennes

L'algorithme K-moyennes se base sur l'optimisation de la somme des carrés des distances entre les points de nuage pour former les regroupements de données appelé clusters. On initialise l'algorithme en précisant le nombre de cluster et les représentants de chaque cluster de façon aléatoire. Dans un premier temps, l'algorithme associe chaque point au représentant le plus proche. Ensuite il déplace les représentants aux centres de chaque cluster et on reprend les mêmes étapes jusqu'à la convergence de l'algorithme.

On fixe le nombre de cluster en 4, on initialise les représentants par les valeurs suivante :

```
Cinit=[-1.5 0 1;-1.5 1 -1;0 -0.4 0;1 0.6 1];
```

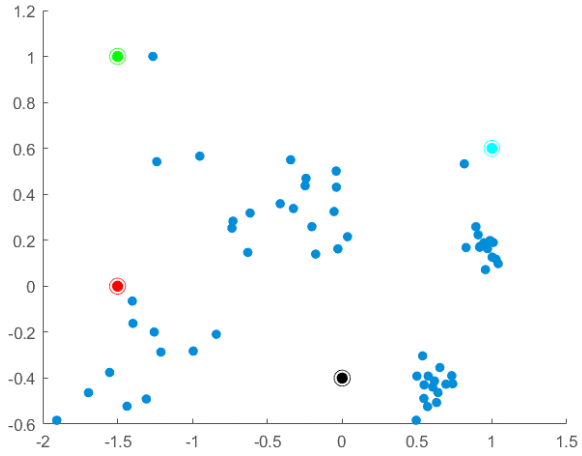
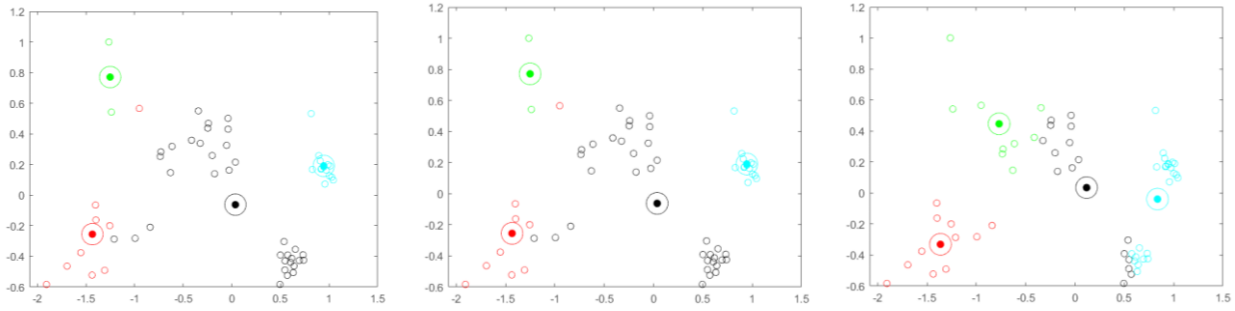
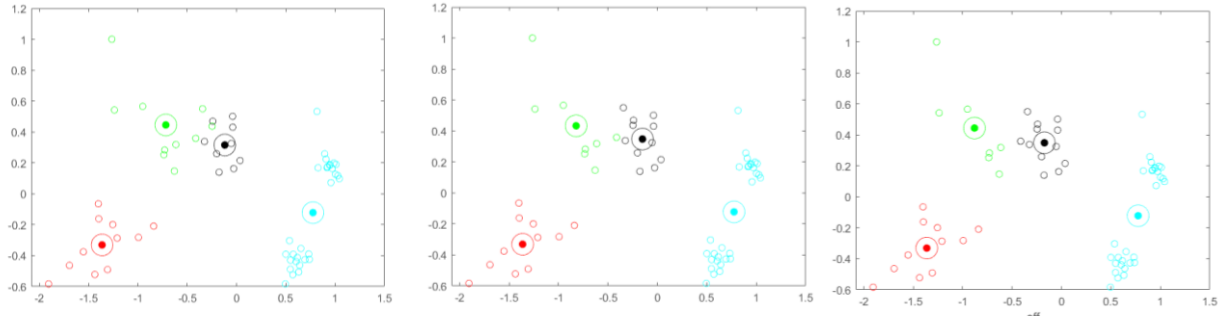


Figure 8 – Initialisation de l'algorithme K-moyennes



Itération 1 Itération 2 Itération 3



Itération 4 Itération 5 Itération 6

Figure 9 – Les itérations de l'algorithme K-moyennes

Si on initialise l'algorithme par un autre vecteur de représentants :

```
Cinit = [0.5 -0.5 0;1 0.5 0;0 0.4 0;-1 1 0];
```

On obtient des clusters différents:

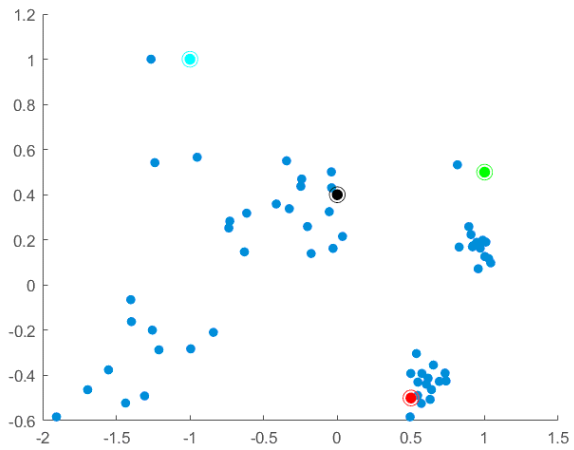


Figure 10 – Initialisation de l'algorithme K-moyennes

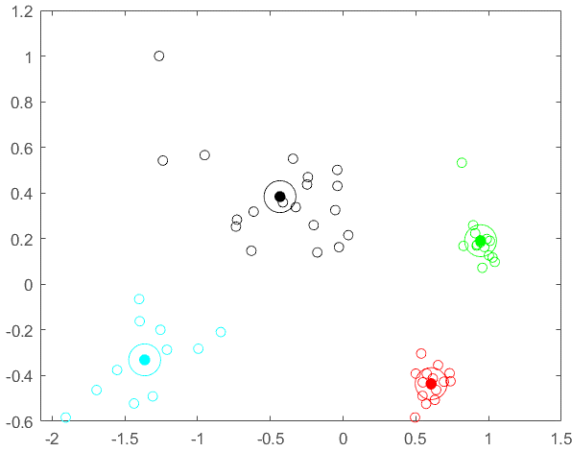


Figure 11 – Résultat de l'algorithme K-moyennes

On remarque que la répartition des données dépend de l'initialisation de l'algorithme, il n'est donc pas déterministe, et on peut obtenir des résultats différents en le faisant tourner plusieurs fois. Et certains de ces résultats peuvent être très mauvais, c'est-à-dire très éloignés de la solution exacte.

b. Gaussian mixture algorithm - Expectation Maximization

Le modèle de mélange gaussien est un modèle statistique qui permet d'estimer paramétriquement la distribution de variables aléatoires en les modélisant comme une somme de plusieurs gaussiennes, ces paramètres sont optimisés selon un critère de maximum de vraisemblance pour approcher le plus possible la distribution recherchée. Cette procédure se fait par l'algorithme espérance-maximisation (EM). On exprime un modèle de mélange de K composants :

$$p(x|\theta) = \sum_{k=1}^{K} p(z_k) p_k(x|z_k, \theta)$$

Pour $x \in \mathbb{R}^d$, on définit un modèle de mélange de gaussienne en choisissant une densité gaussienne pour chaque composant de paramètres $\mu_k et\ \Sigma_k$ tel que :

$$p_k(x|z_k, \theta) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_k)^t \Sigma_k^{-1}(x-\mu_k)\right)$$

On initialise les paramètres des gaussiennes de façon aléatoire, et on essaie de trouver les meilleurs paramètres descriptifs du modèle.

On obtient les figures suivantes :

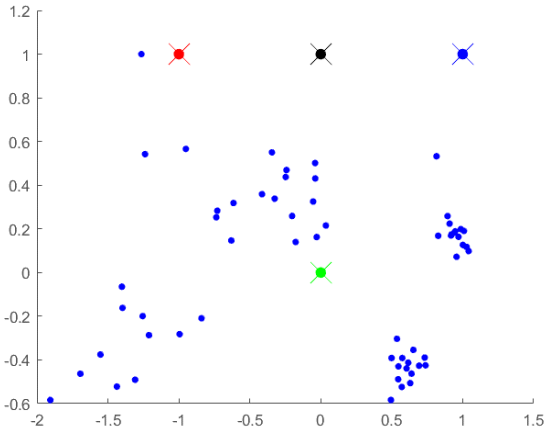


Figure 12 – Initialisation de l'algorithme EMM-GM

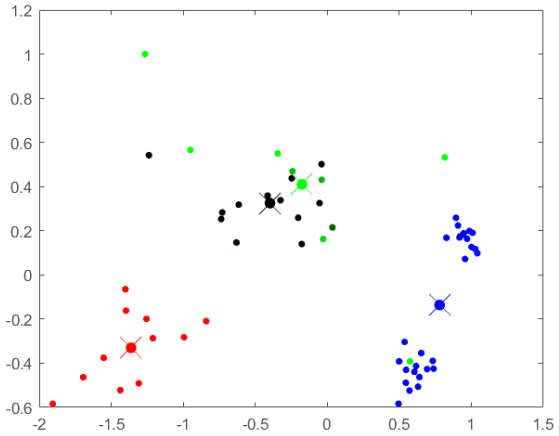


Figure 13 – Résultat de l'algorithme EMM-GM

IV- Réseau de neurones

Un réseau de neurones artificiels permet à l'ordinateur d'apprendre à partir de nouvelles données. Ce type d'algorithme est similaire aux méthodes d'apprentissage supervisé. L'ordinateur doté du réseau de neurones apprend à effectuer une tâche en analysant des exemples pour s'entraîner. Ces exemples ont préalablement été étiquetés afin que le réseau puisse savoir ce dont il s'agit. Autrement dit, les réseaux de neurones ne sont rien d'autre qu'une façon de construire des modèles paramétriques, c'est-à-dire pour lesquels la fonction de décision est explicite. Contrairement à d'autres algorithmes paramétriques comme la régression linéaire, ils permettent de construire facilement des modèles très complexes et non linéaires.

Un réseau de neurones est en général composé d'une suite de couche l'une après l'autre. A chaque couche on associe un ensemble de poids qu'on essaie d'optimiser à l'aide d'un algorithme d'optimisation comme l'algorithme du gradient. L'algorithme converge une fois qu'on trouve les paramètres de couches qui modélise parfaitement l'ensemble des données et permettent d'identifier chaque élément de l'ensemble d'apprentissage.

Un réseau de neurones convolutifs ou réseau de neurones à convolution est un type de réseau de neurones artificiels. Il fonctionne comme un extracteur de features. Pour cela, il effectue du template matching en appliquant des opérations de filtrage par convolution. La première couche filtre l'image avec plusieurs noyaux de convolution, et renvoie des "feature maps", qui sont ensuite normalisées (avec une fonction d'activation) et redimensionnées. Ce procédé est réitéré plusieurs fois : on filtre les features maps obtenues avec de nouveaux noyaux, ce qui nous donne de nouvelles features maps à normaliser et redimensionner.

Dans ce projet, nous allons appliquer ces réseaux de neurones sur deux bases de données de feuilles d'arbre. Ils contiennent respectivement 4 puis 32 feuilles d'arbres. Finalement nous allons analyser les performances de ce type d'algorithme.

a. Classification de 4 feuilles d'arbre avec un RNC

Dans un premier lieu, nous allons commencer par la base qui contient 4 feuilles : sweet potato, bitter orange, vieux garcon, rose.

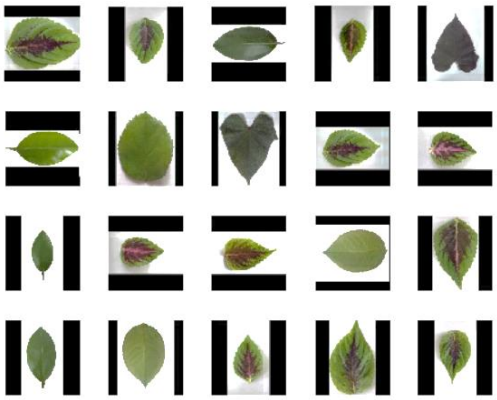


Figure 14 – feuilles d'arbres

On découpe les données en deux parties, des données d'apprentissage et des données de validations :

```
numTrainFiles = 120;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

Ensuite on définit la structure du réseau de neurones. On choisit un réseau fully connected avec 3 couches de convolutions, et des filtres de dimensions 2 tel que :

```
layers = [
    imageInputLayer([175 175 3])

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(4)
    softmaxLayer
    classificationLayer];
```

On lance l'algorithme et on obtient :

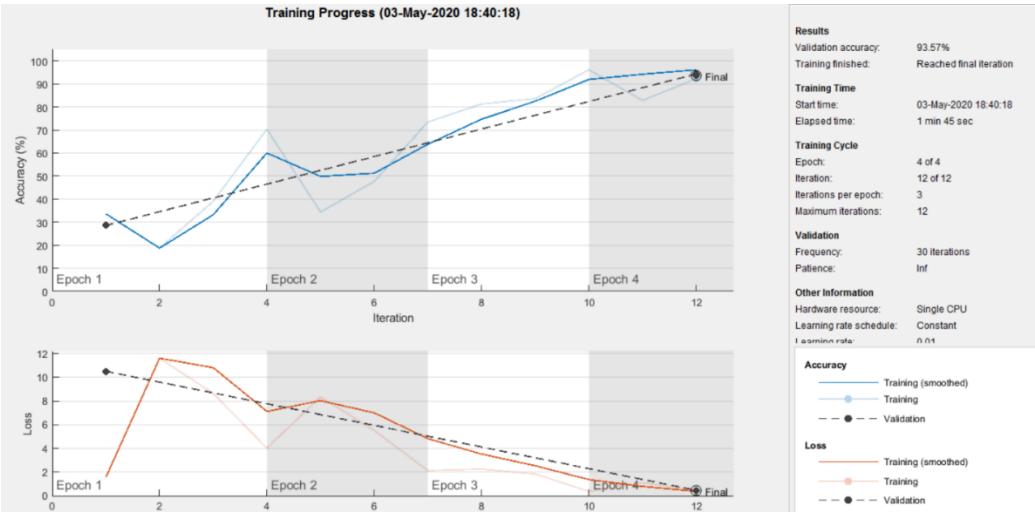


Figure 15 – Training progress

D'après la figure précédente, la précision sur les données d'apprentissage est de 93.57%. On test l'ensemble de validation sur ce réseau de neurones et on calcule sa précision à identifier les éléments de cet ensemble.

```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation);
```

On trouve accuracy = 0.9357, ce qui signifie que l'algorithme est assez performant.

b. Classification de 32 feuilles d'arbre avec un RNC

De la même manière nous allons traiter les éléments de la base contenant les 32 catégories de feuilles d'arbres. Finalement on calcule la précision d'identification des données de validation. Celle-ci est égale à 0.8622 ce qui prouve la qualité du réseau.

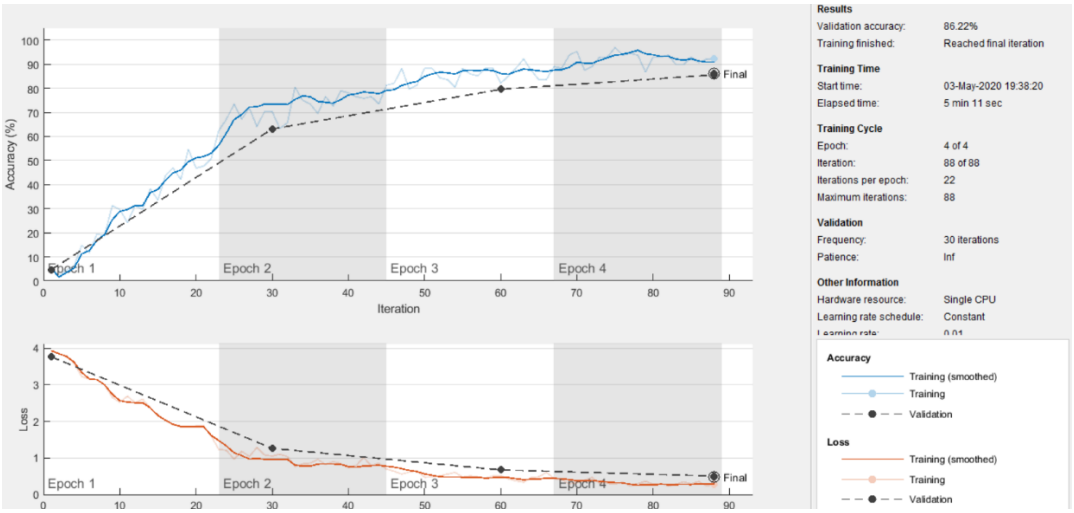


Figure 16 – Training progress

ANNEXE

Algorithmes d'apprentissage supervisé et non supervisé

```
% clear all
close all

LeafType={'papaya','pimento','chrysanthemum','chocolate_tree'};

label=[];
X=[];
for LT=LeafType([1 2 3 4])

    filenames=dir([LT{1},filesep,'Training',filesep,'*.png']);


    for ifile=1:length(filenames)

        img=imread([filenames(ifile).folder,filesep,filenames(ifile).name]);
        X=[X;extractFeatures(img)];
        label=[label,LT];
        close all;

    end
end
labeltest=[];
Xtest=[];
for LT=LeafType([1 2 3 4])

    filenames=dir([LT{1},filesep,'Test',filesep,'*.png']);


    for ifile=1:length(filenames)

        img=imread([filenames(ifile).folder,filesep,filenames(ifile).name]);
        Xtest=[Xtest;extractFeatures(img)];
        labeltest=[labeltest,LT];
        close all;

    end
end

%% manual feature selection

Xs=X(:,1:3);
figure, hold,
for LT=LeafType
    Ilt=find(strcmp(label,LT));
    scatter3(Xs(Ilt,1),Xs(Ilt,2),Xs(Ilt,3),'o','filled');
%     pause
end
legend(LeafType(1:4),'Location','SouthWest');


%% Dimension reduction by PCA
avgX = mean(X);
X = X-avgX;
Xtest = Xtest-avgX;
covV = X'*X;
[U,D,V] = eig(covV);
[d,Isort] = sort(diag(D),'desc');
V = V(:,Isort);
Xp = X*V;
Xptest = Xtest*V;
figure, hold,
for LT = LeafType
```

```
    Ilt = find(strcmp(label,LT));
    scatter3(Xp(Ilt,1),Xp(Ilt,2),Xp(Ilt,3),'o','filled');
end
legend(LeafType(1:4),'Location','SouthWest');

%% Supervised Approaches (nonparamatric - parzen)
sig = 0.5;

figure; hold on;
% Training
for LT=LeafType
    Ilt=find(strcmp(label,LT));
    isoContoursParzen(Xp(Ilt,1:2),sig);
    scatter(Xp(Ilt,1),Xp(Ilt,2),'o','filled');

    Ilttest = find(strcmp(labeltest,LT));
    scatter(Xptest(Ilttest,1),Xptest(Ilttest,2),'o','filled');
end

% Test
K = length(LeafType);
Vsmb = zeros(K,size(Xptest,1));
for i=1:size(Xptest,1)
    for k=1:K
        Vsmb(k,i)=gaussParzen(Xptest(i,1:3)',Xp((k-1)*15+1:k*15,1:3),sig);
    end
end
[erreur,errClass] = calculerErreur(Vsmb);

%% Supervised Approaches (parametric)
% Training
d=3;
K = length(LeafType);
muT = zeros(d,K);
CovT = zeros(d,d,K);
figure; hold on;
i=1;
for LT = LeafType
    Ilt = find(strcmp(label,LT));
    muT(:,i) = mean(Xp(Ilt,1:d))';
    CovT(:,:,i) = cov(Xp(Ilt,1:d));
    dim=[1 2];   %choix de la dimension de projection
    isoContoursGauss(muT(dim,i),CovT(dim,dim,i));
    scatter(Xp(Ilt,dim(1)),Xp(Ilt,dim(2)),'o','filled');
    i=i+1;
end

K = length(LeafType);
Vsmbparametric = zeros(K,size(Xptest,1));
for i=1:size(Xptest,1)
    for k=1:K
        Vsmbparametric(k,i)=mvnpdf(Xptest(i,1:3)',muT(:,k),CovT(:,:,k));
    end

end

[erreurparam,errClassparam] = calculerErreur(Vsmbparametric);


%% Unsupervised Approaches: K-means and GMM-EM



%% K-means
% initialisation des centroides des classes
figure, scatter(Xp(:,1),Xp(:,2),'o','filled');
% Cinit=[-1.5 0 1;-1.5 1 -1;0 -0.4 0;1 0.6 1];
```

```
% Cinit = [0.5 -0.5 0;1 0.5 0;0 0.4 0;-1 1 0];
Cinit = [0.5 -0.5 0;0 0.5 0;0 0.4 0;-1 1 0];
% Cinit = [0.5 -0.5 0;1 0.5 0;0 0.4 0;-1 1 -1];
gr=[1,2,3,4];

hold;
h1=gscatter(Cinit(:,1),Cinit(:,2),gr,'rgkc','oo',20,[],'off');
set(h1,'MarkerSize',10)
h1=scatter(Cinit(1,1),Cinit(1,2),50,'r','O','filled');
h2=scatter(Cinit(2,1),Cinit(2,2),50,'g','O','filled');
h3=scatter(Cinit(3,1),Cinit(3,2),50,'k','O','filled');
h4=scatter(Cinit(4,1),Cinit(4,2),50,'c','O','filled');
axis([-2 1.5 -0.6 1.2]);

% Mise en oeuvre des k-moyennes
% Cette boucle dÃ©tail le rÃ©sultat pour chaque itÃ©ration
% 7 itÃ©rations sont nÃ©cessaires pour converger dans ce cas

for i=1:7
    pause(1)
    opts = statset('MaxIter',i);
    [IDX,C] = kmeans(Xp(:,1:3),4,'start',Cinit,'options',opts);
    hold off;
%     h1=gscatter(C(:,1),C(:,2),gr,'rg','OO',20,[],'off');
%     set(h1,'MarkerSize',10);
    h1=gscatter(C(:,1),C(:,2),gr,'rgkc','oo',20,[],'off');
    hold on;
    h1=scatter(C(1,1),C(1,2),50,'r','O','filled');
    h2=scatter(C(2,1),C(2,2),50,'g','O','filled');
    h3=scatter(C(3,1),C(3,2),50,'k','O','filled');
    h4=scatter(C(4,1),C(4,2),50,'c','O','filled');
    axis([-2 1.5 -0.6 1.2]);
    gscatter(Xp(:,1),Xp(:,2),IDX,'rgkc','oo',[],'filled','off');

end

%% EM
clear Sigma
clear Sigmak

N=size(Xp,1);
D=3;
K=4;

% Initialisation "Ã  la main" des moyennes et covariances des classes
mu=[-1 1 1; 0 0 1;1 1 -1;0 1 -1];
Sigma(1,:,:)=eye(D);
Sigma(2,:,:)=eye(D);
Sigma(3,:,:)=eye(D);
Sigma(4,:,:)=eye(D);
pi=ones(1,K)/K;
apost=zeros(K,N);


figure, scatter(Xp(:,1),Xp(:,2),20,'b','fill');

gr=[1,2,3,4];

hold;
h1=gscatter(mu(:,1),mu(:,2),gr,'rgbk','xx',[],'off');
set(h1,'MarkerSize',20)
h1=scatter(mu(1,1),mu(1,2),50,'r','O','filled');
h2=scatter(mu(2,1),mu(2,2),50,'g','O','filled');
h3=scatter(mu(3,1),mu(3,2),50,'b','O','filled');
h4=scatter(mu(4,1),mu(4,2),50,'k','O','filled');

axis([-2 1.5 -0.6 1.2]);
```

```
% code de mise en oeuvre de l'EM (limitÃ© Ã  50 itÃ©rations)

for i=1:50
    pause(0.2)
    % ï¿½tape E
    for k=1:K
        muk=mu(k,:);
        Sigmak(:,:)=Sigma(k,:,:);
        apost(k,:)=mvnpdf(Xp(:,1:3),muk,Sigmak)*pi(k);
    end
    apost=apost./repmat(sum(apost),K,1);

    hold on;
%     gscatter(data(:,1),data(:,2),IDX,'rg','**',[],'off');

    color=[apost(1:3,:)'];
    scatter(Xp(:,1),Xp(:,2),20,color,'fill');
    axis([-2 1.5 -0.6 1.2]);
    pause(0.2)

    % Etape M
    for k=1:K

mu(k,:)=sum(repmat(apost(k,:)',1,D).*Xp(:,1:3))./repmat(sum(apost(k,:)'),1,D);
        Sigma(k,:,:)=(repmat(apost(k,:)',1,D).*(Xp(:,1:3)-
repmat(mu(k,:),N,1)))'*(Xp(:,1:3)-
repmat(mu(k,:),N,1))./(repmat(sum(apost(k,:)'),D,D));
        pi(:,k)=sum(apost(k,:))/N;
    end

    % DÃ©cision du MAP pour affichage
    [~,IDX]=max(apost);
    hold off;
    h1=gscatter(mu(:,1),mu(:,2),gr,'rgbk','xx',[],'off');
    set(h1,'MarkerSize',20);
    hold on;
    h1=scatter(mu(1,1),mu(1,2),50,'r','O','filled');
    h2=scatter(mu(2,1),mu(2,2),50,'g','O','filled');
    h3=scatter(mu(3,1),mu(3,2),50,'b','O','filled');
    h4=scatter(mu(4,1),mu(4,2),50,'k','O','filled');

%     gscatter(data(:,1),data(:,2),IDX,'rg','**',[],'off');
    color=[apost(1:3,:)'];
    scatter(Xp(:,1),Xp(:,2),20,color,'fill');
    axis([-2 1.5 -0.6 1.2]);
    hold off;
end
```

Fonction GaussParzen

```
function d =  gaussParzen(x,Xk,sigma)
    sig = (sigma.^2)*eye(size(Xk,2));
    d=0;
    for j=1:size(Xk,1)
        d=d+mvnpdf(x',Xk(j,:),sig);
    end
    d=d/size(Xk,1);
end
```

Fonction calculerErreur

```
function [erreur,errClass] = calculerErreur(Vmb)
    errClass = [];
    Label = [ones(1,5),2*ones(1,5),3*ones(1,5),4*ones(1,5),5*ones(1,5)];
    erreur = 0;
    for i=1:size(Vmb,2)
       S = 0;
       for j=1:size(Vmb,1)
           if(j~=Label(i))
                S=S+Vmb(j,i);
           end
       end
       S=S/4;
       errClass = [errClass,S];
       erreur = erreur + S;
    end
end
```

Algorithme RNC appliqué sur la base de données contenant 4 feuilles

```
clear all
close all
clc

digitDatasetPath = fullfile('.\Leaf\');
imds =imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

figure;
perm = randperm(620,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end

%% label count
labelCount = countEachLabel(imds);

%% image size
img = readimage(imds,1);
size(img)

%% devide elements
numTrainFiles = 120;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

%% train network
layers = [
    imageInputLayer([175 175 3])

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)
```

```
    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(4)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,layers,options);

%% compute validation accuracy

YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation);
```

Algorithme RNC appliqué sur la base de données contenant 32 feuilles

```
clear all
close all
clc

digitDatasetPath = fullfile('.\Imagros\');
imds =imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

figure;
perm = randperm(620,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end

%% label count
labelCount = countEachLabel(imds);

%% image size
```

```
img = readimage(imds,1);
size(img)

%% devide elements
numTrainFiles = 90;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

%% train network
layers = [
    imageInputLayer([175 175 3])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(2,64,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(size(labelCount,1))
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,layers,options);

%% compute validation accuracy
```

```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation);
```