Mickaël FAINET
Nicolas MARTIN
Julien DUBOIS

UNIVERSITE DE ROUEN

M1 GEII

Recherche d'espaces couleurs hybrides

Tuteurs :

Sébastien ADAM

Romain RAVEAUX

Alpage Project

GEii - ROUEN

Introduction

Le projet ALPAGE (AnaLyse diachronique de l'espace urbain Parisien : approche GEomatique) a pour objectif de retrouver des informations au sein d'anciens plans cadastraux urbains. Il est donc nécessaire de trouver un outil permettant de faciliter l'extraction de ces informations.

Dans le cadre de notre projet semestriel, il nous a été demandé de mettre en œuvre et d'évaluer un outil basé sur la méthode d'Analyse en Composantes Principales (ACP), à savoir : l'Espace Hybride Décorellé (EHD).

L'EHD offre une nouvelle représentation de l'espace couleur RGB qui constitue le modèle naturel de nos images. En effet, il est issu des 3 composantes prépondérantes des espaces couleurs de base. En construisant cet EHD, nous espérons ainsi, faciliter l'extraction d'objets graphiques sur les cadastres afin d'appuyer les historiens qui collaborent à l'ensemble du projet ALPAGE.

1. Les espaces couleurs

a. Présentation

Il existe beaucoup d'espaces de représentation de la couleur et on ne peut pas dire qu'un système de couleurs soit meilleur qu'un autre. En effet, cela dépend du contexte et de l'objectif de l'analyse des couleurs.

Tous les espaces couleurs sont constitués de trois composantes principales comme par exemple le système RGB (R :rouge, G :vert et B :bleu). N'importe quelle couleur du spectre peut être retrouvée par un mélange pondéré de trois couleurs primaires. On qualifie une couleur de primaire lorsqu'elle ne peut pas être obtenue par un mélange d'autres couleurs primaires. Suivant le système de synthèse dans lequel on se trouve, on peut représenter une couleur par un point dans un espace tridimensionnel, c'est-à-dire que chaque couleur aura trois composantes, une par couleur primaire.

Le système RGB, qui possède donc trois composantes de couleur primaire, s'avère souvent suffisant pour une analyse des images couleurs comme c'est le cas dans le projet ALPAGE. Mais voyons plus en détail l'espace RGB car il sera le point de départ pour la détermination de notre EHD puisqu'à partir de lui on peut réaliser une conversion pour se projeter dans les autres espaces couleurs comme le CMJ (C :cyan, M :magenta, J :jaune) par exemple.

b. L'espace RGB

L'espace RGB est un espace qui découle de la synthèse additive. La synthèse additive est basée sur le principe d'émission de la lumière (ou plutôt des couleurs) c'est-à-dire que la perception des couleurs provient directement d'une source de lumière et non d'une réflexion de la lumière sur les objets que l'on voit. En synthèse additive, les trois couleurs primaires sont donc le rouge, le vert et le bleu. Cela conduit naturellement à une représentation tridimensionnelle de la couleur selon ces trois primaires : l'espace RGB.

Le mélange égal des primaires donne le blanc : l'ensemble des couleurs est représenté dans un cube, le cube des couleurs. Ainsi dans cet espace à trois dimension, le noir est en (0, 0, 0) et le blanc est à k.(1, 1, 1) avec k la valeur maximale que peut prendre chaque primaire.

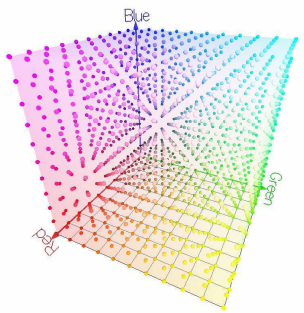


Figure 1 : Cube des couleurs RGB

La figure 1 nous montre l'ensemble des couleurs représentées dans le cube des couleurs de l'espace RGB évoqué précédemment. Dans le cas du projet ALPAGE, les images sont naturellement issus de l'espace RGB :

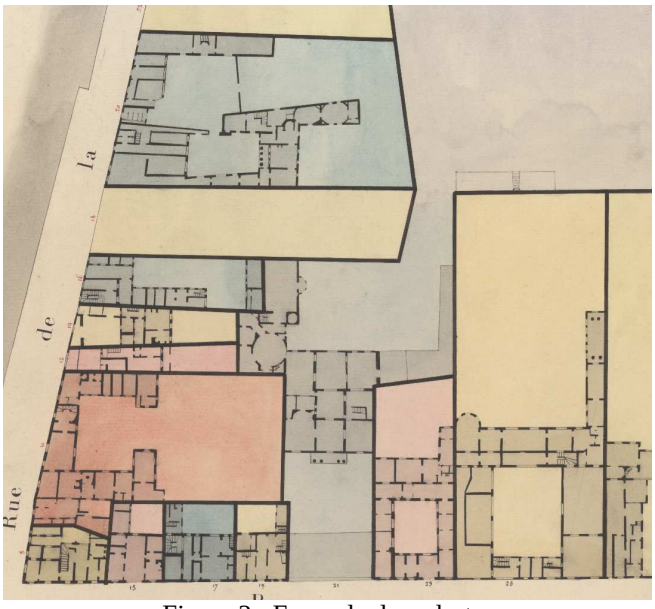


Figure 2 : Exemple de cadastre

Il s'agit donc, tout d'abord, de déterminer la teneur en rouge, en vert et en bleu de cette image comme représenté dans le cube des couleurs.

De plus, le but de notre projet est d'étudier le comportement de chaque pixel dans d'autres espaces couleurs (LAB, CIE…) et ainsi, de trouver l'espace qui maximise la séparabilité des données. Pour trouver cet espace, il faut donc calculer toutes les composantes de chaque espace et déterminer les plus importantes.

Pour cela, nous devons calculer les composantes de l'espace RGB et projeter les résultats obtenus dans les autres espaces couleurs en « convertissant » par le biais de formules connues, les valeurs obtenues pour chaque composante.

c. Conversion de l'espace RGB vers les autres espaces

Toute l'étude en amont de la classification a été réalisée en langage JAVA y compris les conversions d'espaces couleur. Les lignes de code associées à ces conversions sont contenues dans la classe ChangeColorSpace et permettent de calculer les 26 composantes en question. Nous avons résumé les systèmes de conversions de l'espace RGB vers les autres espaces dans le tableau 1 et par souci de concision, on ne montre que le code pour la conversion de l'espace RGB vers les systèmes YIQ et YUV en annexe 1.

	Conversion à partir de l'espace RGB
Y of YIQ	`0.299*r + 0.587*g + 0.114*b`
I of YIQ	`0.596*r - 0.274*g - 0.322*b`
Q of YIQ	`0.212*r - 0.523*g + 0.311*b`
Y of YUV	`0.299*r + 0.587*g + 0.114*b`
U of YUV	`-0.148*r - 0.289*g + 0.437*b`
V of YUV	`0.615*r - 0.515*g - 0.100*b`

Tableau 1 : Systèmes de conversion linéaire entre espaces couleurs

2. Validation de la méthode par le cas d'école

a. Base d'apprentissage et de test

- *Pour l'apprentissage :*

- Image traitée : couleurs pures
- 65 Classes
- 1 pixel par classe

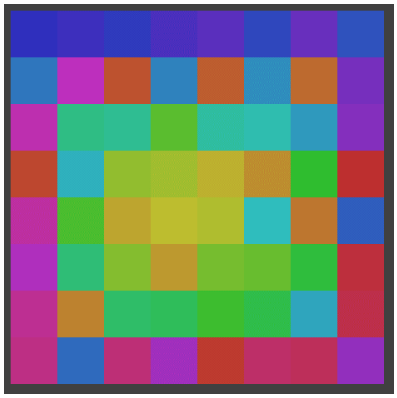


Figure 3 : base d'apprentissage

- *Pour le test :*

- Image bruitée : Bruit uniforme 3%
- 65 classes
- 219 000 pixels dans la base de test
- On connaît la vérité terrain grâce à l'image pure

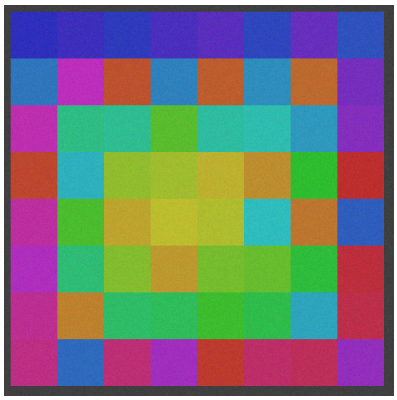


Figure 4 : base de test

b. Résultats de la classification

Nous utilisons le classifieur par fôrets aléatoires (random forest) pour ce cas d'école. L'image étudiée a été construite pour favoriser les espaces perceptuellement uniformes car les couleurs choisies sont très proches dans RGB.

Color Space	Number of attibutes	Classification Rate
RGB	3	94,27
L* a*b*	3	98,56
All Components	26	100
Genetic search	1	97,13
CFS	11	100
HCS (EHD)	3	100

Tableau 2 : Taux de reconnaissance (classification par random forest)

Le cas d'école montre que l'EHD peut s'avérer très efficace pour la reconnaissance des couleurs et donc faciliter l'extraction d'objets graphiques sur une image. Par exemple dans le cas d'école, on peut imaginer extraire chaque carré « rouge ».

Nous parvenons à obtenir une reconnaissance à 100% avec l'EHD qui n'utilise que trois composantes couleurs. On peut également obtenir un taux de reconnaissance de 100% en choisissant un espace couleurs formé des 26 composantes de base ou par un espace obtenu par CFS et qui ramène le nombre de composantes à 11. Nous constatons en effet que les systèmes RGB ou L*a*b*, dans le cas d'école, obtiennent de moins bons résultats.

c. Application au projet ALPAGE

Dans le cadre du projet ALPAGE, les objets graphiques que nous souhaitons extraire sont des informations topographiques à savoir :

- Les parcelles en reprenant l'unité de couleur,
- la présence d'un numéro de rue (et éventuellement reconnaissance du numéro),
- le filaire des rues du 14ème,
- le filaire des rues du 19ème.

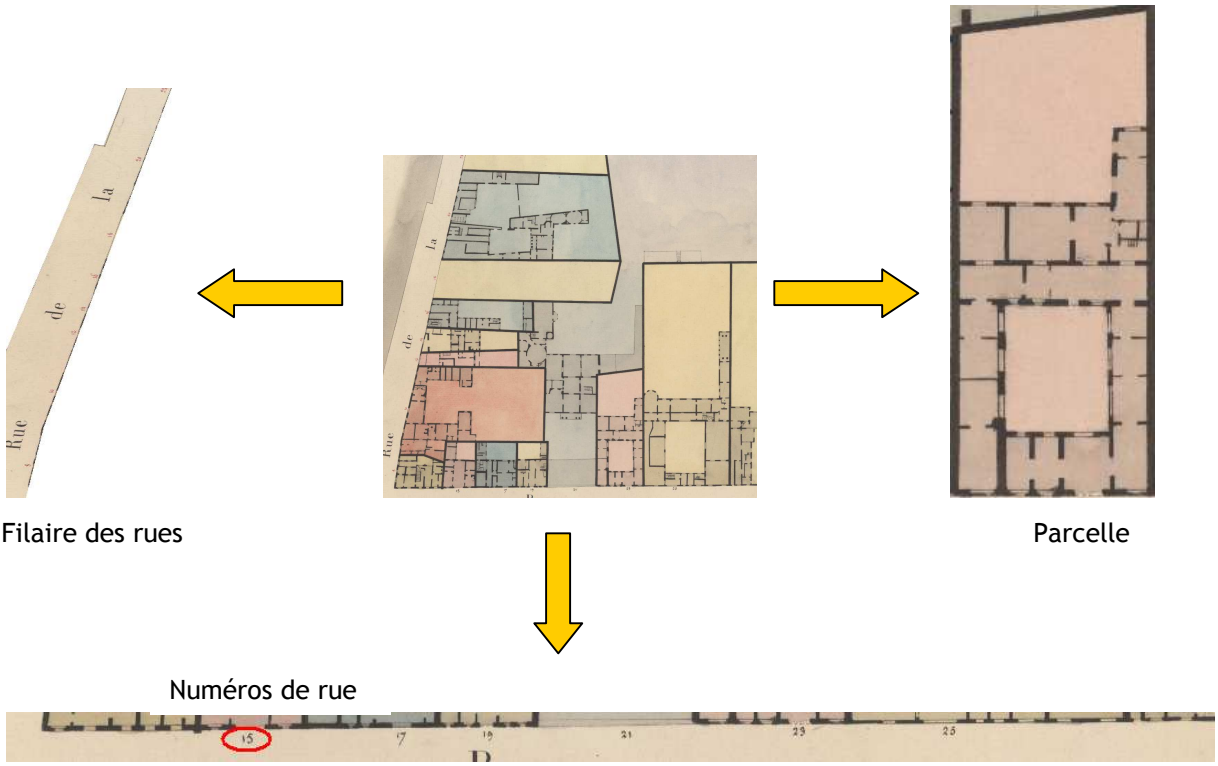


Figure 5 : Extraction d'objets graphiques

3. Détermination de l'EHD par l'ACP

a. Présentation de l'Espace Hybride Décorellé (EHD)

L'idée de base d'un espace hybride décorellé est de combiner 3 composantes couleurs qui proviennent de différents modèles couleurs de bases (RGB, CMY, Lab…) afin de former un nouvel espace à 3 composantes, garantissant une linéarité avec les espaces de base.

L'espace hybride décorellé est un espace couleur réduisant la corrélation entre les composantes et par conséquent la redondance de l'information.

Le choix des trois composantes les plus caractéristiques se fait à l'aide de l'analyse en composantes principales (ACP) et de ses différentes étapes. Passons donc à cette ACP et à la mise en œuvre de ses différentes étapes.

b. Analyse en Composantes Principales (ACP)

- ***Première étape : Définition d'un ensemble de composantes***

La première étape de l'ACP consiste à définir un ensemble de composantes couleurs. Soit les 26 composantes :

```
Components[0]="ROfRGB";
Components[1]="GOfRGB";
```

```
Components[2]="BOfRGB";
Components[3]="AOfAC1C2";
Components[4]="C1OfAC1C2";
Components[5]="C2OfAC1C2";
Components[6]="LOfLAB";
Components[7]="AOfLAB";
Components[8]="BOfLAB";
Components[9]="I1OfI1I2I3";
Components[10]="I2OfI1I2I3";
Components[11]="I3OfI1I2I3";
Components[12]="YOfYUV";
Components[13]="UOfYUV";
Components[14]="VOfYUV";
Components[15]="YOfYIQ";
Components[16]="IOfYIQ";
Components[17]="QOfYIQ";
Components[18]="LOfLUV";
Components[19]="UOfLUV";
Components[20]="VOfLUV";
Components[21]="XOfXYZ";
Components[22]="YOfXYZ";
Components[23]="ZOfXYZ";
Components[24]="Saturation";
Components[25]="Teinte";
```

Par conséquent, soit C un ensemble de composantes couleurs :

$$C=\{R,G,B,...,Saturation,Teinte\}=\{S_i\}_{i=1}^{26}$$

Soit I une image composée de N pixels, où un pixel est représenté comme suit :

$$\vec{P_i}=[S_1,S_2,S_3]$$

Les pixels sur les images des plans cadastraux sont les données sur lesquelles nous allons appliquer l'ACP.

- ***Deuxième étape : Calcul de la matrice de covariance***

Cette phase de l'ACP consiste à calculer une matrice de covariance entre les 26 composantes. Cette matrice que nous noterons M est définie comme suit :

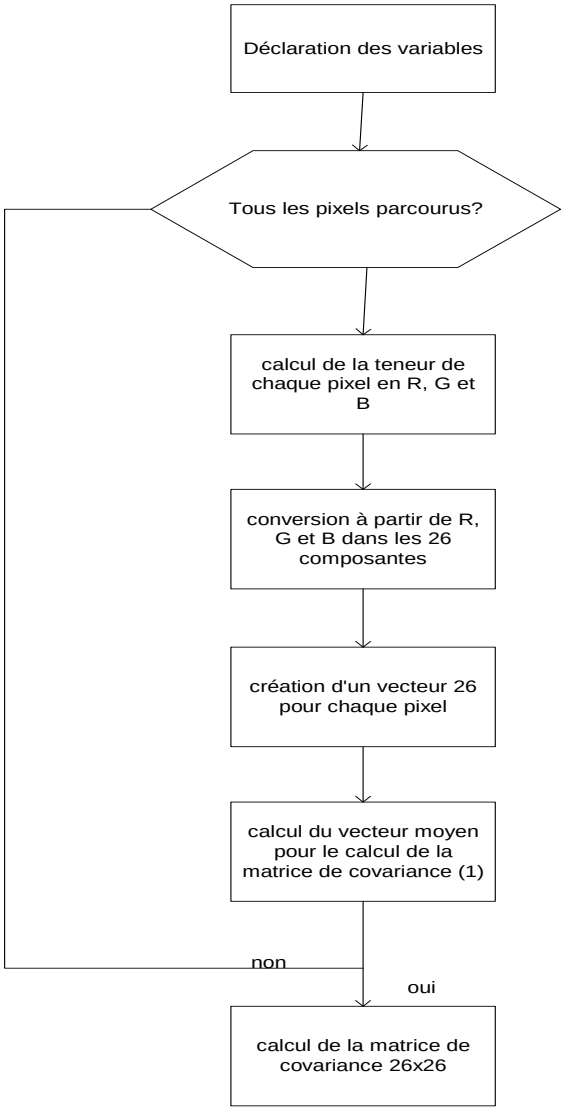
$$M_{ij}=\sum_{k=1}^{N}(S_i(k)-\overline{S}_i)\times(S_j(k)-\overline{S}_j)$$

Où $\overline{S}_i$ est la moyenne du canal S_i.

Dans le cas d'une image RGB, à trois canaux, cette matrice est une matrice 3x3.

$$M = \begin{bmatrix} \text{var}(R) & \text{cov}(R,G) & \text{cov}(R,B) \\ \text{cov}(G,R) & \text{var}(G) & \text{cov}(G,B) \\ \text{cov}(B,R) & \text{cov}(G,B) & \text{var}(B) \end{bmatrix}$$

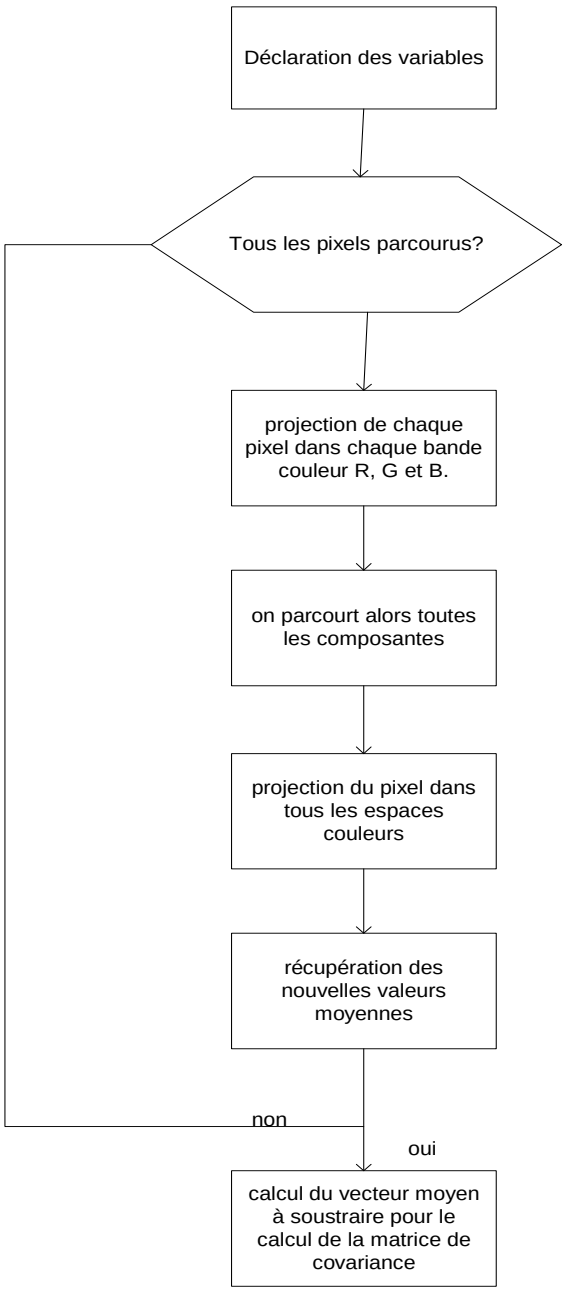
Le travail étant déjà réalisé pour cette matrice 3x3, il nous a fallu l'étendre pour les 26 composantes que nous avons définies dans la classe PCAOnImageMultiSpaces et compléter la méthode getCovarianceMatrix(). Nous proposons ci-après une version algorithmique qui permet de créer la matrice de covariance 26x26. Les lignes de code sont fournies en annexe 2.



(1) : l'étape du calcul du vecteur moyen sera vue en détails ci-après.

Il est à noter que nous utilisons une autre méthode en interaction avec celle-ci pour calculer cette matrice de covariance. En effet, la méthode GetMeanVector()

calcule le S_i défini dans la formule qui permet d'obtenir la matrice de covariance. Voici la version algorithmique de GetMeanVector(), le code est disponible à l'annexe 3 :



Une fois cette matrice de covariance déterminée, nous pouvons passer à la troisième étape de l'ACP qui consiste à déterminer les valeurs propres de cette matrice de covariance. Le package Jama fournit les outils nécessaires à la manipulation de matrices, il nous sera utile lors des troisièmes et quatrièmes étapes.

- ***Troisième étape : Décomposition en valeurs propres***

$$\boxed{D = P^{-1} \bullet M \bullet P}$$

P : est la matrice de passage composée des vecteurs propres de la matrice de covariance.

M : est la matrice de variance covariance.

$P^{-1} = \left[P_1 P_2 P_3 \ldots P_{24} P_{25} P_{26}\right]$: où P_i est le vecteur colonne *i* de la base ACP. P_i est aussi appelé axe factoriel.

D : est la matrice diagonale des *valeurs propres*.

$$D = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & . \\ 0 & 0 & \lambda_3 & 0 & 0 & 0 & . & . & . & . \\ 0 & 0 & . & \lambda_{....} & . & . & . & . & . & . \\ . & . & . & . & \lambda_{....} & . & . & . & . & . \\ . & . & . & . & . & \lambda_{....} & . & . & . & . \\ . & . & . & . & . & . & \lambda_{....} & . & 0 & 0 \\ . & . & . & . & 0 & 0 & 0 & \lambda_{24} & 0 & 0 \\ . & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{25} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{26} \end{bmatrix}$$

Chaque valeur propre est associée à un axe factoriel ou vecteur propre et représente la quantité d'information portée par cet axe.

La classe SingularValueDecomposition et plus particulièrement la méthode getSingularValues() du package Jama retourne les valeurs propres λ_1, λ_2, λ_3, …, λ_{25}, λ_{26} ordonnées dans l'ordre décroissant de la matrice passée en argument. Le code associé à cette méthode est le suivant :

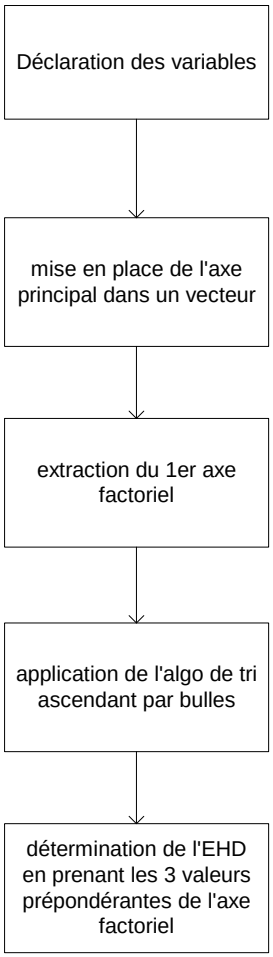
```
public void getVerticalSingularVectors(double[][] mat){

    Matrix matrix = new Jama.Matrix(mat);
    SingularValueDecomposition SVD = new
SingularValueDecomposition(matrix);

    //on récupère la matrice des vecteurs propres dans SingularVector
    SingularVector = SVD.getV();
    //on récupère le tableau des valeurs propres dans SingularValue
    SingularValue = SVD.getSingularValues();
   }
```

Les valeurs propres sont contenues dans la variable SingularValue tandis que les vecteurs propres sont contenus dans SingularVector.

- ***Dernière étape : Détermination de l'EHD***

L'étape précédente nous a permis de déterminer les 26 vecteurs propres et par conséquent les 26 axes factoriels. L'EHD se construit en prenant le premier axe factoriel (première colonne de la matrice des vecteurs propres, SingularVector dans le code) et en choisissant les trois coefficients les plus signifiants.

Pour cela, on utilise la fonction getEHD() dont voici l'algorithme (le code est disponible à l'annexe 4) :



c. Résultats obtenus

Les résultats issus de l'analyse en composantes principales sur l'image tmp3.tif mènent aux résultats suivants :

Les 26 valeurs propres renvoyées par la méthode getSingularValue() λ_1, λ_2, λ_3, …, λ_{25}, λ_{26} sont classées par ordre décroissant.

```
0.29005861886858114
2.0313436432189782E-17
5.512864934192267E-19
2.5105877652884573E-19
1.9326031599774255E-19
1.5409961320719856E-19
1.0353616575284151E-19
6.838261311766143E-20
5.902088240864467E-20
4.3263944502248317E-20
2.5999527255233897E-20
7.585808084831395E-21
3.0840302238229655E-21
1.6225227387086744E-21
3.751503558721205E-22
2.8869341934443046E-22
1.7663976036611656E-22
1.3659954212443167E-22
7.659224994293666E-23
4.876130539396405E-23
4.1298099056158 53E-23
8.633884690491587E-24
3.1141907967036306E-24
6.924979397775288E-36
2.4497680383022857E-37
0.0
```

On remarque que quasiment 100% ($\frac{\lambda_1}{\sum_{i=1}^{25}\lambda_i}$) de l'information est portée par la première valeur propre ci-dessus. Un espace issu de l'analyse en composantes principales seule serait alors formé des 3 axes factoriels principaux associés aux trois valeurs propres principales ci-dessus. Pour construire un espace hybride décorellé, on ne prend seulement que le premier axe factoriel afin d'en extraire les 3 composantes prépondérantes.

Nous disposons d'une interface homme-machine développée en JAVA pour faire l'acquisition d'une image et lancer une analyse en composantes principales sur cette image. La classe FrameMain du package GuiPkg fournit un point d'entrée sur le programme. Ainsi pour l'image tmp3.tif, il en résulte l'EHD suivant :

```
[AOfAC1C2    ZOfXYZ    GOfRGB]
```

Nous avons également réalisé une ACP sur un autre plan cadastral et qui constituera la source de données sur laquelle nous travaillerons lors de la classification.

Nous avons rencontré quelques problèmes d'exécution pour réaliser l'analyse en composantes principales. En effet, l'image fait plus de 50Mo (disponible à l'adresse Internet http://alpageproject.free.fr/TER_M1_GEII/Im_DB1.png) et ne tient pas en mémoire dans la machine virtuelle.

La première initiative a été de redimensionner l'image afin de réduire sa taille et par la même occasion son poids, mais cela induit une interpolation pixellaire et donc une modification de la source de données. L'autre possibilité est de changer les options de compilation (il faut rajouter « -Xss3m -Xms200m -Xmx900m »). Dans le cas contraire, une exception de type « `java.lang.OutOfMemoryError: Java heap space` » surviendra.

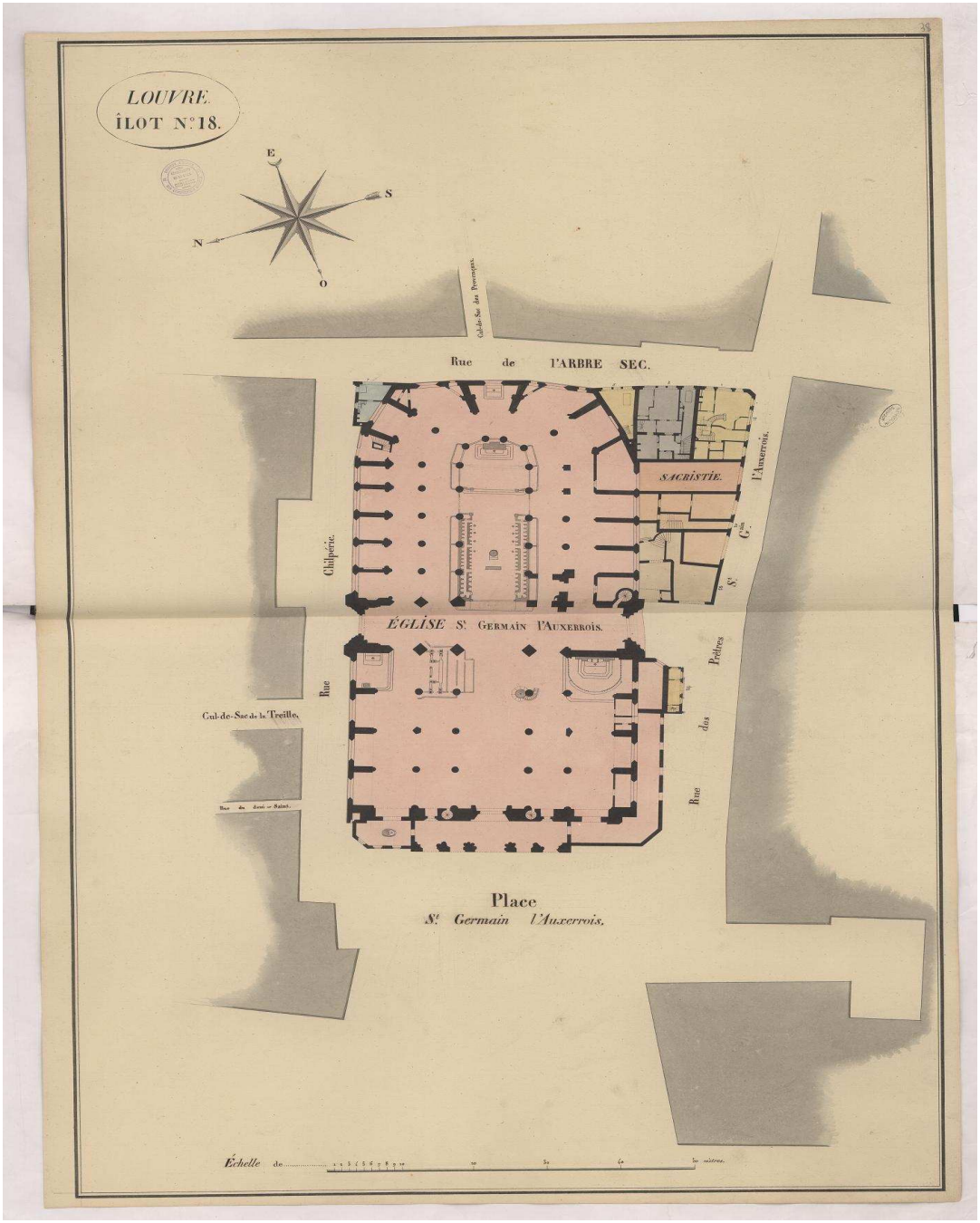


Figure 6 : Source de données DB1

Nous trouvons l'EHD suivant : `[A0fAC1C2 Teinte C20fAC1C2]`

Cette première phase du projet nous a permis de déterminer un EHD par la méthode de l'analyse en composantes principales. Il est sensé maximiser la séparabilité des données et donc rendre plus aisé l'extraction des objets graphiques des plans cadastraux entrant dans le cadre du projet ALPAGE.

A ce stade, vient la question du calcul du taux de reconnaissance avec différents classifieurs qui devra infirmer ou confirmer la pertinence de notre espace hybride décorellé. Pour cela, nous utiliserons les outils de classification proposés sous le logiciel Weka.

4. Classification pixellaire

a. Exportation des sources de données au format XML

Le logiciel Weka prend en entrée des fichiers qui portent l'extension ARFF. Pour obtenir les fichiers ARFF, il est nécessaire d'utiliser un fichier XML contenant les valeurs RGB de chaque pixel ainsi qu'une information sur la classe d'appartenance de ce pixel. Chaque pixel est alors considéré comme une instance dont on connait la vérité terrain, c'est-à-dire sa classe d'appartenance parmi les 7 classes suivantes :

DB1		
	Samples	Name
Class #1		Black
Class #2		Blue
Class #3		Gray
Class #4		Green
Class #5		Orange
Class #6		Pink
Class #7		Yellow

Tableau 3 : Echantillons représentatifs des classes sur l'image DB1

Cette base XML est transformée en fichier ARFF (avec plusieurs espaces couleurs). La lecture et l'écriture du fichier XML se fait par l'API JDOM une surcouche de DOM (parseur XML).

Pour utiliser au mieux la source de données disponibles afin d'évaluer les performances des classifieurs sur l'EHD trouvé, on choisit de diviser la source d'information en deux sous-ensembles ; l'un servira de base d'apprentissage et l'autre de base de test. Cette méthode aussi appelée « hold-out » présente cependant un certain dilemme dans la mesure où il nous faut le plus de données possibles en apprentissage comme en test, tout en gardant une variance considérable dans les données. La voici illustrée dans l'image suivante :

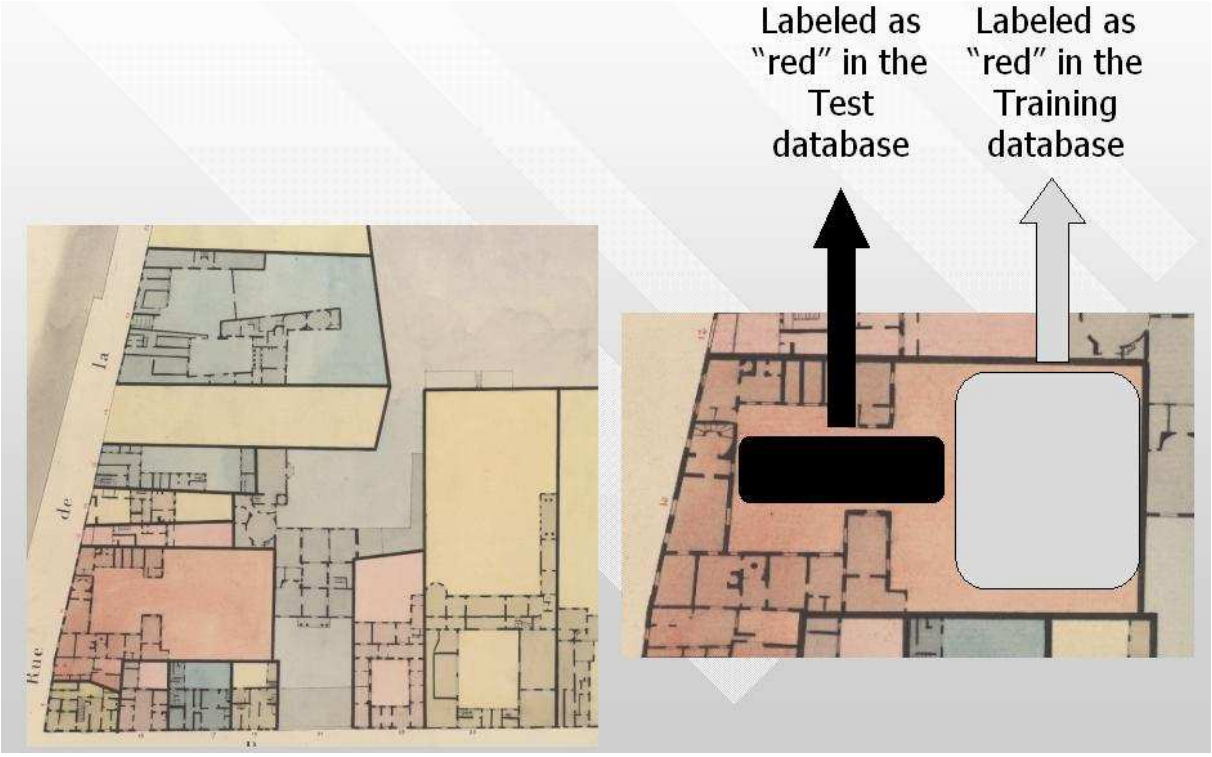


Figure 7 : Exemple d'extraction des bases d'apprentissage et de test sur un cadastre

Ainsi, nous avons a disposition deux fichiers ARFF qui constituent une base d'apprentissage et une base de test pour notre source de donnée DB1 et ceci, avec les 26 composantes couleurs.

Voici les étapes détaillées pour créer la base d'apprentissage sous le logiciel Weka. Nous allons modifier les deux fichiers ARFF de bases afin de traiter uniquement les composantes liées à l'EHD trouvé pour la source DB1 :

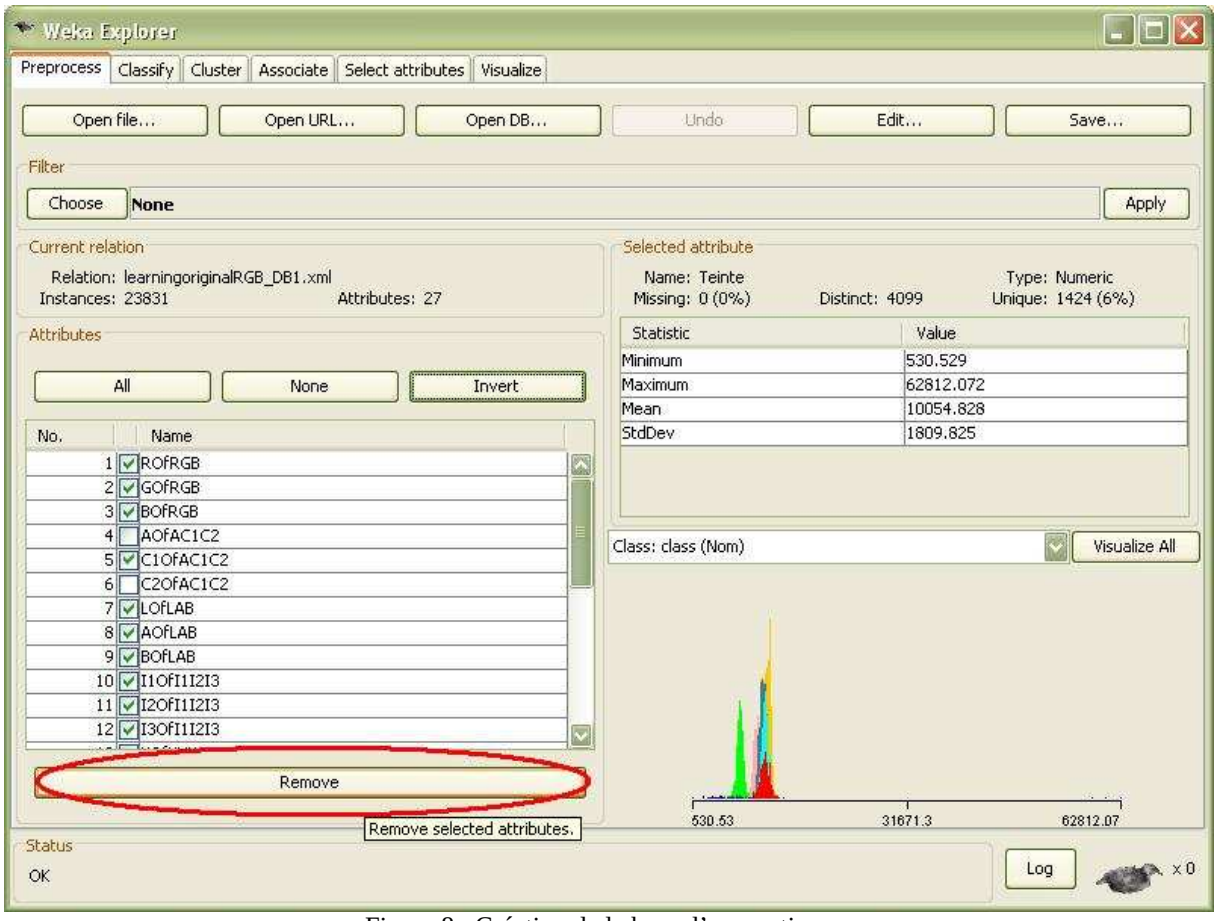


Figure 8 : Création de la base d'apprentissage

On supprime les composantes autres que A de AC1C2, C2 de AC1C2 et Teinte. Puis, on sauvegarde notre nouveau modèle ARFF pour la classification pixellaire concernant l'EHD. On suit le même cheminement pour réaliser le fichier ARFF relatif à la base de test.

On peut commencer à faire la classification sous Weka.

b. Explication de la méthode d'évaluation

Les principaux classifieurs mis en en concurrence lors de l'évaluation de l'EHD sont le 1 plus proche voisin, les forêts aléatoires de décision et le SVM. Cependant, notre étude a plus porté sur la classification au plus proche voisin en considérant les distances euclidiennes.

Le concept général de la classification au plus proche voisin avec les distances euclidiennes est de juger la classe d'appartenance d'une instance (en l'occurrence d'un pixel) en fonction de la classe d'appartenance du représentant de la classe la plus proche, lui-même déterminé par les distances euclidiennes.

Dans le cas des couleurs, on se situe le plus souvent dans des espaces de représentation sur 3 axes. Prenons l'exemple du RGB :

Soient deux pixels p1 et p2, et d(p1, p2) la distance qui les séparent :

$$d(p1,p2)=\sqrt{\left((R1-R2)^2+(G1-G2)^2+(B1-B2)^2\right)}$$

On calcule cette distance entre le pixel qu'on cherche à classer et chacun des autres pixels (soit NbPixels * NbClasses opérations).

On recherche parmi ces distances, le plus proche voisin.

En outre, on connait la vérité terrain du plus proche voisin, on choisit alors d'attribuer au pixel à classer, la classe d'appartenance de son plus proche voisin.

Prenons pour cela un exemple simple. Voici un schéma illustrant un exemple de la méthode du plus proche voisin :

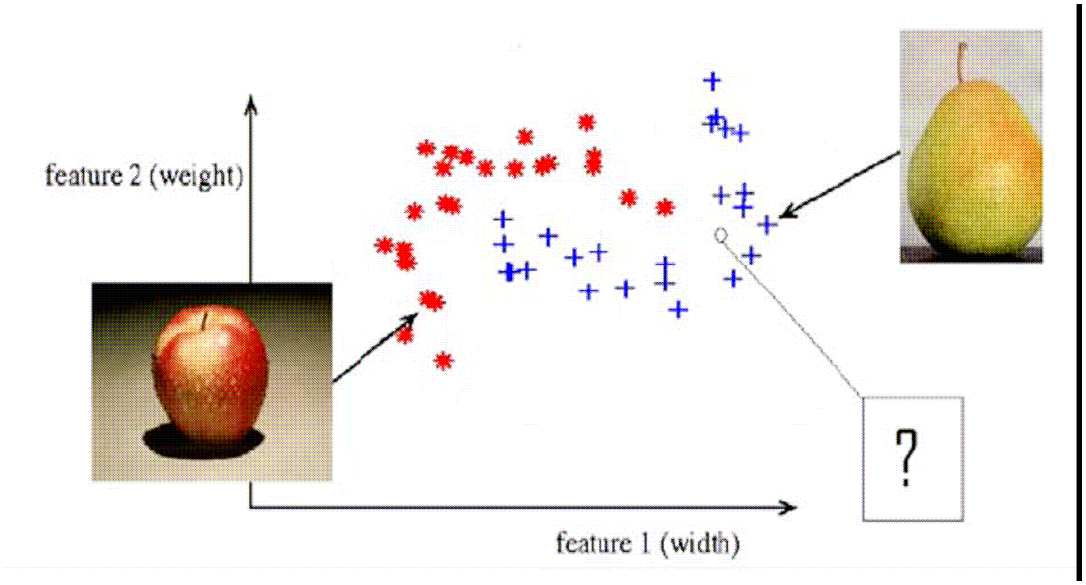


Figure 9 : illustration de la méthode du plus proche voisin

L'instance que nous cherchons à classifier, ici représentée par un cercle, sera attribuée à la classe « poire » car son plus proche voisin appartient à la classe « poire ».

c. Utilisation de Weka pour la classification

A partir du fichier ARFF pour l'apprentissage on va choisir un classifieur (IB1, Random Forest…) puis, on va solliciter une base de test :

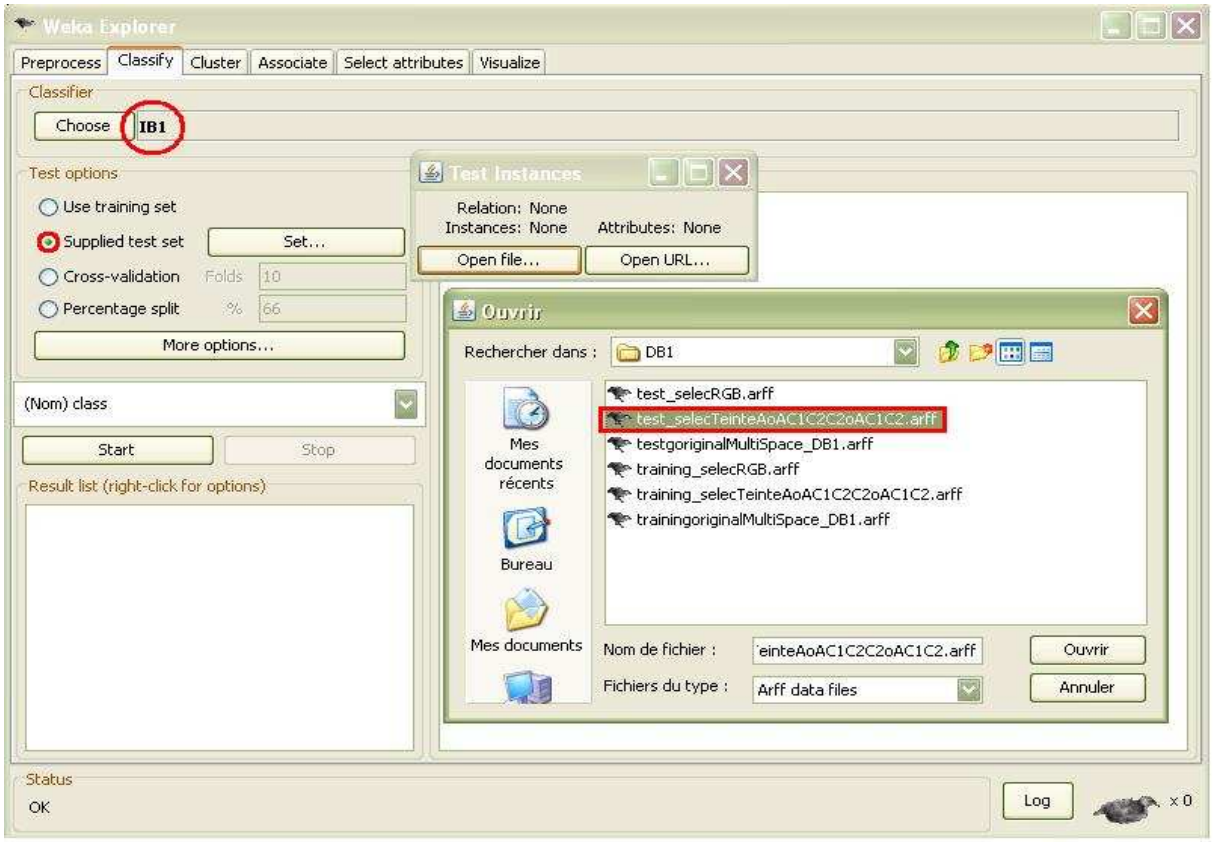


Figure 10 : Utilisation du classifieur Weka

On choisit le classifieur IB1 (ou 1-NN pour 1 plus proche voisin). Puis, on fournit la base de test « test_selecTeinteAoAC1C2C2oAC1C2.arff ». Le calcul par Weka peut alors commencer. Les résultats sont présentés comme suit :

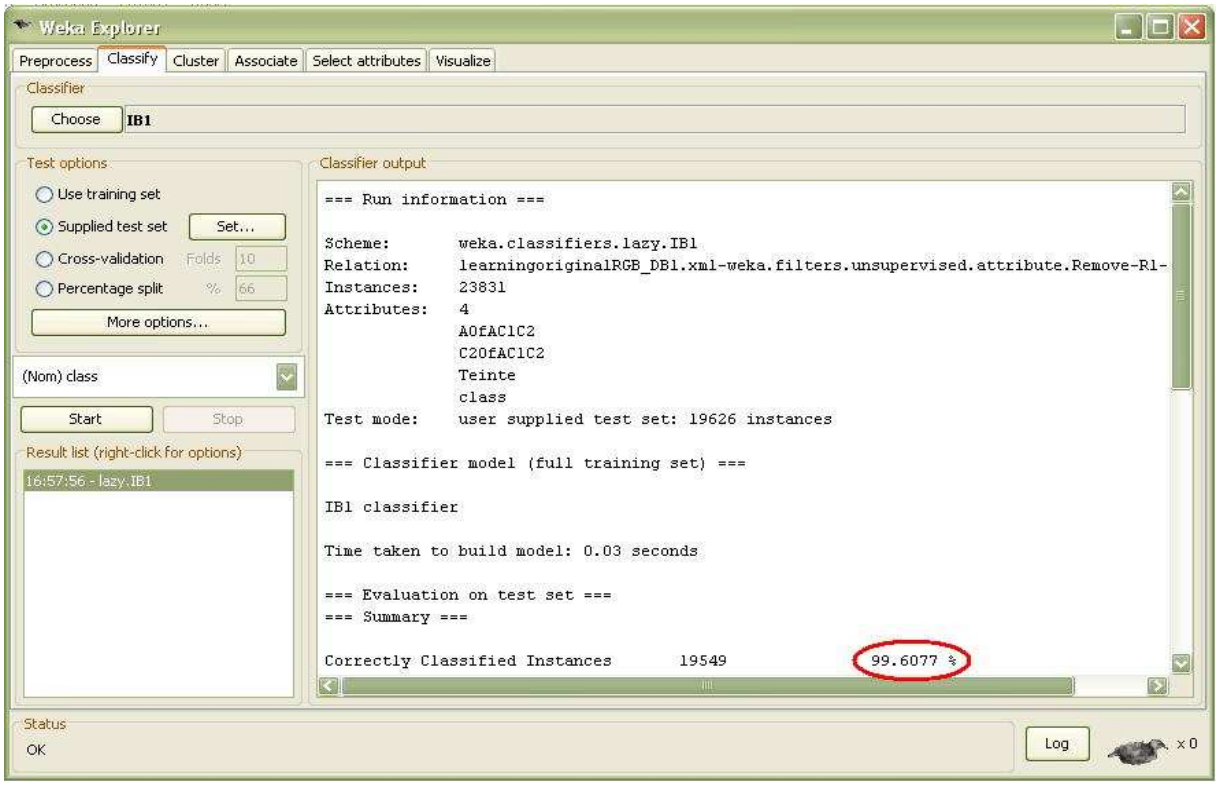


Figure 11 : Présentation des résultats sous Weka

5. Résultats issus de Weka

a. Les résultats menés sur l'EHD

L'ensemble des résultats issus de la classification sous Weka pour l'EHD :

```
=== Run information ===

Scheme:       weka.classifiers.lazy.IB1
Relation:     learningoriginalRGB_DB1.xml-
weka.filters.unsupervised.attribute.Remove-R1-3,5,7-25
Instances:    23831
Attributes:   4
              AOfAC1C2
              C2OfAC1C2
              Teinte
              class
Test mode:    user supplied test set: 19626 instances

=== Classifier model (full training set) ===

IB1 classifier

Time taken to build model: 0.02 seconds

=== Evaluation on test set ===
=== Summary ===

Correctly Classified Instances       19549               99.6077 %
Incorrectly Classified Instances        77                0.3923 %
```

```
Kappa statistic                          0.9952
Mean absolute error                      0.0011
Root mean squared error                  0.0335
Relative absolute error                  0.4644 %
Root relative squared error              9.5424 %
Total Number of Instances            19626

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0          1         1         1        black
  0.998     0          1         0.998     0.999    blue
  1         0          1         1         1        gray
  0.987     0.001      0.993     0.987     0.99     green
  0.986     0.002      0.985     0.986     0.986    orange
  1         0.001      0.993     1         0.996    pink
  0.997     0          0.998     0.997     0.998    yellow

=== Confusion Matrix ===

    a    b    c    d    e    f    g   <-- classified as
  621    0    0    0    0    0    0 |    a = black
    0 2921    0    7    0    0    0 |    b = blue
    0    0 6230    0    0    0    1 |    c = gray
    0    0    1 2483   28    3    1 |    d = green
    0    0    0    8 2120   20    2 |    e = orange
    0    0    0    1    0 3343    0 |    f = pink
    0    0    0    1    4    0 1831 |    g = yellow
```

Nous possédons les informations relatives aux performances du classifieur : notamment le nombre d'instances total et le nombre d'instances correctement classées. Nous en déduisons le taux de reconnaissance :

$$Taux\_\mathrm{Re}co = \frac{Nombre\_d'elements\_bien\_classés}{Nombre\_d'elements\_à\_reconnaître}$$

Soit : 19549 / 19626 = 99.6077 %

b. Comparaison des résultats

Si l'on généralise la méthode décrite précédemment, voici l'ensemble des résultats que l'on peut obtenir sur la base de données DB1 en fonction du classifieur utilisé et de l'espace couleur choisi :

DB1				
	# of attributes	1-NN (L2 Distance)	Random-Forest (L2 distance)	SVM (Polynomial kernel)
RGB	3	99.6994	98.9198	98.9962
All Components	26	99.5975	99.6637	99.9185
AC1C2	3	99.6382	98.8077	99.8013
La*b*	3	99.6382	99.2816	99.8471
I1I2I3	3	99.6994	99.4497	99.9083
Yuv	3	99.679	99.5414	99.893
YIQ	3	99.6586	99.3988	99.893
Luv	3	99.8217	99.4497	99.9083
XYZ	3	99.3223	97.5848	91.6437
HIS	3	98.5937	99.3682	98.3389
Hybrid Color Space[1]	**3**	**99.6077**	**99.358**	**97.1721**
PCA Space[2]	3	99.7045	98.8892	99.5363
OneR Evaluation[3]	3	77.1222	93.8959	94.7671
Cfs[4]	XX	98.5988	99.6688	99.9134
Genetic Search[5]	XX	98.3033	99.4039	99.9236

Tableau 4 : Classification rate performs in 15 color spaces

[1] Hybrid Color Space `Based on Colantony method`
[2] PCA analysis from the 26 primary components, the 3 most significant axis are kept.
[3] Evaluates the worth of an attribute by using the OneR classifier.
[4] Evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them.
[5] Performs a search using the simple genetic algorithm described in Goldberg (1989).

Egalement, on dispose des taux de précision et du recall pour les 7 classes :

DB1 using a 1-NN classifier																
	Name	RGB	All Components	AC1 C2	La*b*	I1I2I3	Yuv	YIQ	Luv	XYZ	HSI	Hybrid Color Space [6]	PCA Space [7]	OneR Evaluation [8]	Cfs [9]	Genetic Search [10]
Class #1	Black	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 0.982	**1** **1**	1 1	1 0.998	1 0.984	1 1
Class #2	Blue	1 1	1 1	1 1	1 0.999	1 1	1 1	1 1	1 0.999	1 0.999	0.996 0.994	**1** **0.998**	1 1	1 0.998	0.997 0.995	1 0.999
Class #3	Gray	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	0.972 0.997	**1** **1**	1 1	0.923 0.312	0.974 0.997	1 1
Class #4	Green	0.997 0.986	0.999 0.981	0.996 0.984	0.996 0.988	0.997 0.987	0.998 0.987	0.998 0.987	0.999 0.996	0.986 0.972	0.984 0.926	**0.993** **0.987**	0.999 0.988	0.354 0.934	0.984 0.924	0.999 0.878
Class #5	Orange	0.984 0.992	0.979 0.989	0.981 0.989	0.986 0.986	0.985 0.99	0.986 0.088	0.986 0.986	0.996 0.991	0.98 0.981	0.987 0.988	**0.985** **0.986**	0.987 0.989	0.994 0.988	0.982 0.988	0.984 0.991
Class #6	Pink	0.996 1	0.993 1	0.995 0.999	0.991 1	0.995 1	0.993 1	0.992 1	0.994 1	0.985 0.997	0.993 1	**0.993** **1**	0.993 1	0.993 1	0.993 1	0.994 0.999
Class #7	Yellow	0.999 0.999	0.998 0.999	0.998 0.999	0.999 0.999	0.999 0.999	0.998 0.999	0.998 0.999	0.999 0.999	0.998 0.998	0.999 0.992	**0.998** **0.997**	0.998 0.999	0.997 0.998	0.999 0.991	0.869 0.999

Tableau 5 : Precision and recall rates. First line : Precision = TP/(TP+FP) // Second line Recall : TP/(TP+FN)

[6] Hybrid Color Space based on Colantony method
[7] PCA analysis from the 26 primary components, the 3 most significant axis are kept.
[8] Evaluates the worth of an attribute by using the OneR classifier
[9] Evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them.
[10] Performs a search using the simple genetic algorithm described in Goldberg (1989).

A titre informatif, voici la matrice de confusion obtenue sous Weka dans le cas de l'espace RGB :

```
=== Confusion Matrix ===

   a    b    c    d    e    f    g   <-- classified as
 621    0    0    0    0    0    0 |    a = black
   0 2927    0    1    0    0    0 |    b = blue
   0    0 6229    1    0    0    1 |    c = gray
   0    0    0 2481   34    1    0 |    d = green
   0    0    0    3 2132   14    1 |    e = orange
   0    0    0    1    0 3343    0 |    f = pink
   0    0    0    1    1    0 1834 |    g = yellow
```

Nous pouvons rapporter cette matrice de confusion au tableau suivant :

DB1 : RGB Space and 1-NN classifier				
	Samples	Name	Confusion 1 (Order 1)	Confusion 2 (Order 2)
Class #1		Black		
Class #2		Blue	Green	
Class #3		Gray	Green	Yellow
Class #4		Green	Orange	Pink
Class #5		Orange	Pink	Green
Class #6		Pink	Green	
Class #7		Yellow	Green	Orange

Tableau 6 : Confusion summary

c. Interprétation des résultats

Comme le démontre les résultats des tableaux 5 et 6, les résultats pour un espace hybride décorellé quelque soit le classifieur ne se démarquent pas des autres résultats. Le RGB offre des résultats plus qu'honorables et rivalise sans complexes avec l'EHD.

Egalement, les matrices de confusion s'approchent de la matrice identité pour l'espace RGB comme pour l'EHD et nous montrons que ce sont les classes Green et Orange qui représentent les plus grosses sources d'erreurs.

La nature des couleurs peut certainement expliquer le fait que l'espace RGB apporte déjà d'excellent résultats sur les plans cadastraux. Effectivement, les images que nous traitons datent de plusieurs dizaines d'années et ont subis les aléas du temps. C'est pourquoi, une phase de prétraitements des images en amont de notre projet a été nécessaire pour rehausser les couleurs, lutter contre l'affadissement des pixels et diminuer le bruit. Les prétraitements subis par les images sont donc :

- Théorie du monde blanc
- Extension dynamique
- Anti-affadissement
- Filtre Médian

Conclusion

Ce projet nous a permis de découvrir certains aspects du monde de la recherche. En effet, il a fallu s'adapter à des moyens non maitrisés : le langage JAVA ainsi que les outils de classification proposés par Weka étaient des concepts nouveaux pour nous.

La problématique était de trouver un espace de représentation de la couleur qui maximise la séparabilité des données. Les résultats menés sur l'EHD montrent que ceux-ci ne se démarquent pas de ceux obtenus dans l'espace naturel des images (RGB). Les algorithmes génétiques s'avèrent être les plus efficaces en terme de séparation chromatique sur les images de cadastres.

La combinaison de classifieurs peut former une nouvelle approche pour résoudre au mieux la problématique.

Bibliographie

- Projet ALPAGE :

[1]Romain Raveaux, Jean-Christophe Burie, Jean-Marc Ogier : "A colour document interpretation: Application to ancient cadastral maps". The 9th International Conference On Document Analysis(ICDAR 2007)

[2]Romain Raveaux, Jean-Christophe Burie, Jean-Marc Ogier : "A contribution to Ancient Cadastral Maps interpretation through color analysis ". The 7th International Workshop on Pattern Recognition in Information Systems (PRIS 2007)

- Les espaces couleurs :

[3] « Image numérique couleur ». Alain Trémeau, Christine Fernandez-Maloigne, Pierre Bonton. Ed. Dunod

[4] P. Colantoni and A. Trémeau, “3d visualization of color data to analyze color images," in *The PICS Conference*, (Rochester, USA), May 2003.

- Les EHD :

[5]J. D. Rugna, P. Colantoni, and N. Boukala, “Hybrid color spaces applied to image database," vol. 5304, pp. 254,264, Electronic Imaging, SPIE, 2004

[6] N. Vandenbrouke, L. Macaire, and J. G. Postaire, “Color pixels classification in a hybrid color space," in *Proceedings of IEEE*, pp. 176-180,1998.

Annexes

Annexe 1 : Systèmes de conversions linéaires entre espaces couleur

```
/* ---------- SYSTEMES YIQ et YUV ---------- */

public double getYOfYIQ(RomPixel p){
      double r = p.GetRed();
      double g = p.GetGreen();
      double b = p.GetBlue();
      double res = 0.299*r + 0.587*g + 0.114*b;
      return res;
}


public double getYOfYUV(RomPixel p){
      double r = p.GetRed();
      double g = p.GetGreen();
      double b = p.GetBlue();
      double res = 0.299*r + 0.587*g + 0.114*b;
      return (res);
}


public double getUOfYUV(RomPixel p){
      double r = p.GetRed();
      double g = p.GetGreen();
      double b = p.GetBlue();
      double res =  -0.148*r - 0.289*g + 0.437*b;
      return (res);
}


public double getVOfYUV(RomPixel p){
      double r = p.GetRed();
      double g = p.GetGreen();
      double b = p.GetBlue();
      double res =  0.615*r - 0.515*g - 0.100*b;
      return res;
}


public double getIOfYIQ(RomPixel p){
      double r = p.GetRed();
      double g = p.GetGreen();
      double b = p.GetBlue();
      double res =  0.596*r - 0.274*g - 0.322*b;
      return res;
}


public double getQOfYIQ(RomPixel p){
      double r = p.GetRed();
      double g = p.GetGreen();
      double b = p.GetBlue();
      double res =  0.212*r - 0.523*g + 0.311*b;
      return res;
}
```

Annexe 2 : méthode permettant d'obtenir la matrice de covariance

```
/**
* Returns the covariance matrix of the multispectral image.
* @return covariance The covariance matrix (dimension numBands*numBands)
* @throws RomException
*/
public double[][] getCovarianceMatrix() throws RomException {

      double[][] covariance = new double[this.ChangeColorSpace.nbcomponent]
      [this.ChangeColorSpace.nbcomponent];
      double[] mean = getMeanVector();

      int numbPixels = width*height; //the number of pixels in the bands

      //on parcourt tous les pixels
      for (int i = 0; i <= width-1; i++) {
            for (int j = 0; j <= height-1; j++) {
                  /*MODIF JMN*/
                  //on instancie un tableau de 3 entiers
                  int tabrg[] = new int[3];

//on parcourt les 3 composantes R,G et B
for (int k = 0; k < numBands; k++) {
      //on fait l'acquisition du pixel de coordonnées (i,j)
      //dans la bande (composante) courante    (R,G ou B)
      tabrg[k] = rasterBands[k].getSample(i, j, 0);
}

      //on instancie un nouvel espace couleur RGB
      RomColor color = new RomColor(tabrg[0],tabrg[1],tabrg[2]);
//on instancie un nouveau pixel aux coordonnées (i,j) avec les valeurs R,G
//et B obtenues précédemment
      RomPixel pcur = new  RomPixel(i,j,color);

      double[] vcur = new double[this.ChangeColorSpace.nbcomponent];
      //on parcourt toutes les composantes
for(int r = 0; r <= this.ChangeColorSpace.nbcomponent-1; r++){
//le vecteur courant contient la valeur du pixel courant dans toutes les
//composantes couleurs
      vcur[r]+= this.ChangeColorSpace.getComponentSpace(pcur,(double)r);
}

      double[] Vector = new double[this.ChangeColorSpace.nbcomponent];

for (int k = 0; k < this.ChangeColorSpace.nbcomponent; k++){
      Vector[k] = vcur[k]-mean[k];
}

for(int n=0; n <=this.ChangeColorSpace.nbcomponent-1;n++){
      for(int p = 0; p <= this.ChangeColorSpace.nbcomponent-1; p++){
            covariance[n][p] = Vector[n] * Vector[p];
      }
    }
  }
}

for (int i = 0; i <= this.ChangeColorSpace.nbcomponent-1; i++){
      for(int j = 0; j <= this.ChangeColorSpace.nbcomponent-1; j++) {
```

```
            covariance[i][j] /= (double)(numbPixels-1);
      }
}

return covariance;
}
/*FIN MODIF JMN*/
```

Annexe 3 : méthode permettant de calculer la moyenne de chaque canal

```
/**
* Return the vector of mean intensity values for each band.
* @return mean The vector of mean intensity values for all the bands.
* @throws RomException
*/
public double[] getMeanVector() throws RomException {
      //numbPixels contient le nombre de pixel de l'image
      int numbPixels = width*height;
      double[] mean = new
double[rom.ColorSpacePkg.ChangeColorSpace.nbcomponent]; //on crée un
tableau de nbcomponent doubles

      //on parcourt tous les pixels de l'image
      for (int i = 0; i <= width-1; i++) {
            for(int j = 0; j <= height-1; j++) {
                  //on crée un tableau de 3 entiers
                  int  tabrg[] = new int[numBands];
                  //on parcourt les 3 composantes R,G et B
                  for (int k = 0; k <= numBands-1; k++) {
                  //on fait l'acquisition du pixel de coordonnées (i,j)
                  //dans la bande (composante) courante    (R,G ou B)
                        tabrg[k] = (int)rasterBands[k].getSample(i, j, 0);
                  }
      //on instancie un nouvel espace couleur RGB
      RomColor color = new RomColor(tabrg[0],tabrg[1],tabrg[2]);
//on instancie un nouveau pixel aux coordonnées (i,j) avec les valeurs R,G
//et B obtenues précédemment
      RomPixel pcur = new RomPixel(i,j,color);

            //on parcourt toutes les composantes
            for(int r =0;r<=this.ChangeColorSpace.nbcomponent-1;r++){
      //On passe le pixel dans tous les espaces couleurs,
      //on recupère les valeurs de ces nouveaux espaces dans mean
      mean[r]+= this.ChangeColorSpace.getComponentSpace(pcur,(double)r);
            }
      }
}

      //on parcourt toutes les composantes
            for (int k = 0; k <= this.ChangeColorSpace.nbcomponent-1; k++){
                  //on calcule la moyenne de chaque canal
                  mean[k] /= (double)numbPixels;
            }
         return mean;
}
```

Annexe 4 : méthode permettant d'extraire l'EHD de l'axe factoriel

```
private int[] getEHD() {
            // TODO Auto-generated method stub

      double PC1[] = new double[this.SingularVector.getRowDimension()];
      int indexcomponent[] = new
int[this.SingularVector.getRowDimension()];

      //on met l'axe principal dans un vecteur
      for(int i=0;i<PC1.length;i++){
            //extraction du premier axe factoriel.
            PC1[i] = Math.abs(this.SingularVector.get(i,0));
            //indexcomponent va servir a indexer les composantes
            indexcomponent[i] = i;
      }

      //on applique l'algo de tri ascendant par bulles au tableau PC1
      //indexcomponent contiendra les indices des composantes triés
      new Tools().sort_vec_modif_param(PC1,indexcomponent);

      int EHD[] = new int[3];

      for(int i=0;i<EHD.length;i++){
//on détermine notre EHD en prenant les 3 dernières valeurs de
//indexcomponent, celles-ci contiennent les index des 3 //valeurs
prépondérantes de l'axe factoriel
            EHD[i] = indexcomponent[PC1.length-1-i];
      }

return EHD;
   }
```

Annexe 5 : Autres exemples de cadastres parisiens