Formations en Informatique de Lille – 2006/2007

Fouille de Données

mars 2007

Clustering de couleurs

Dans ce TP, il s'agit de transformer une image en k couleurs en une image en k' couleurs, avec $k' < k$. Partitionner uniformément l'espace des couleurs en k' parties ne donne en général pas de bons résultats visuels. Certaines couleurs apparaissant très souvent vont se voir réaffecter à une seule et même couleur, alors que ce sont elles qui font toute la richesse de l'image.
Ainsi, il est important d'accorder plus d'importance aux couleurs qui apparaissent plus souvent. L'approche couramment utilisée est celle du clustering. Chaque pixel à une couleur codée par trois composantes, R, G, et B. Un pixel peut donc représenté un exemple dans un espace à trois dimensions. Il s'agit alors de partitionner l'ensemble des exemples en k' clusters. Un cluster C est alors un ensemble de couleurs, et chaque couleur de cet ensemble se verra affecter la couleur c, où c est le centre de gravité de C. Il reste alors à remplacer tous les pixels par les nouvelles couleurs dans l'image initiale.
Nous utiliserons l'algorithme des k-moyennes pour faire le clustering. Il est implémenté dans Weka, nous utiliserons donc Weka directement dans nos classes Java. La compilation et l'exécution s'effectuent alors comme suit :

```
javac -classpath /usr/local/weka-3-4-5/weka.jar Exemple.java
java -classpath .:/usr/local/weka-3-4-5/weka.jar Exemple
```

Il existe une classe pour manipuler les images `MyImage` dans le répertoire `/home/vacataire/filiot/clustering` plus des images exemples. Cette classe comporte les méthodes suivantes :

```
public class MyImage {

   public MyImage(String filename)
   // crée un objet image à partir d'un fichier au format PNG

   public int getHeight()
   // retourne la hauteur de l'image

   public int getWidth()
   // retourne la largeur

   public int getR(int x, int y)
   // retourne la composante rouge du pixel (x,y)
```

```
    public int getG(int x, int y)
    // retourne la composante verte du pixel (x,y)

    public int getB(int x, int y)
    // retourne la composante bleue du pixel (x,y)

    public void setRGB(int r, int g, int b, int x, int y)
    // remplace la couleur du pixel (x,y) par la couleur (r,g,b)

    public String getName()
    // retourne le nom de l'image (nom du fichier d'origine)

    public void writeImage(String type, String filename)
    // écrit l'image dans un fichier,
    // au format [type] avec pour nom [filename]
    // exemple: writeImage(''PNG'', ''output.png'')
}
```

1 Définir un ensemble d'exemples avec Weka

Vous aurez besoin d'importer les classes suivantes :

```
import weka.core.Instances;
import weka.core.Instance;
import weka.core.Attribute;
import weka.core.FastVector;
```

Il faudra, dans l'ordre :
- définir quels sont les attributs, et leur type (un attribut est un objet de la classe `Attribute`)
- construire l'ensemble d'exemples à partir de ces attributs (un exemple est un objet de la classe `Instance`)
- les intégrer à l'ensemble d'exemples (l'ensemble des exemples est un objet de la classe `Instances`)

1.1 Définir les attributs

Il y deux types principaux d'attributs dans Weka, les attributs numériques et les attributs nominaux. Dans notre cas, les attributs sont tous numériques.
Pour créer un attribut numérique, il faut appeler la méthode `Attribute (String attname)`.

1.2 Construire un ensemble d'exemples

Il faut d'abord regrouper tous les attributs dans un `FastVector`.

```
FastVector attributes = new FastVector(n) ;
attributes.addElement(attribut1) ;
attributes.addElement(attribut2) ;
attributes.addElement(attribut3) ,
```

```
...
attributes.addElement(attributn) ;
```

Ensuite, il faut créer un ensemble d'instance, en initialisant à la bonne taille :

```
Instances dataset = new Instances(''nom'', attributes, nb_exemples) ;
```

Enfin, il faut ajouter les exemples à l'ensemble d'instances. Pour ajouter un exemple :

```
Instance inst = new Instance(n) ;
// s'il y n valeurs d'attributs

inst.setValue(att1, 2.0) ;
inst.setValue(att2, 3.0) ;
...
inst.setValue(attn, 0.2) ;
```

Et il faut ajouter cet exemple à l'ensemble d'instances :

```
inst.setDataset(dataset) ;
dataset.add(inst) ;
```

Question Définir une classe `ImageInstance` avec un constructeur `ImageInstance(MyImage img)`, qui construit l'ensemble d'instances. Il faut donc balayer l'image, et créer une instance `Instance` par pixel, avec les trois valeurs d'attributs (R,G,B). Vous aurez besoin d'autres méthodes par la suite, mais ce n'est pas le propos pour l'instant.

2 Clustering de couleurs

Il s'agit ici de définir une classe `Quantization` qui réalise la conversion de k couleurs en k' couleurs. Il faudra utiliser l'algorithme des K-moyennes de Weka :

```
import weka.clusterers.* ;
```

Allez voir la javadoc weka, pour la classe `SimpleKMeans`.
Pour effectuer le clustering sur l'ensemble d'instances `dataset`, il faut :
- créer un clusterer (constructeur `SimpleKMeans`)
- définir le nombre de clusters (ici k'), avec `setNumCluster(int nb)`
- initialiser éventuellement le générateur aléatoire `setSeed`
- construire les clusters `buildClusterer(Instances dataset)` (c'est l'étape la plus coûteuse)

Une fois que les clusters sont calculés, il faut maintenant réaffecter chaque couleur de pixel à la couleur représentée par le centre de gravité du cluster à laquelle elle appartient.
Pour cela, il faut :
- balayer l'ensemble des pixels
- récupérer sa couleur
- le classifier avec le clusterer construit, avec la méthode `int clusterInstance(Instance inst)`

- la méthode précédente retourne un entier i qui correspond à l'indice du cluster auquel l'instance appartient. Pour récupérer le centre de ce cluster, il faut faire appel à la méthode `Instances getClusterCentroids()`, et en récupérer la i-ème instance :

Voici le prototype :

```
SimpleKMeans clusterer =  (SimpleKMeans) new SimpleKMeans() ;

clusterer.setNumClusters(k) ;
clusterer.setSeed(new Random().nextInt()) ;
clusterer.buildClusterer(inst.getInstance()).;

Instances centroids = clusterer.getClusterCentroids() ;

for (int x = 0 ; x < largeur ; x++){
for (int y = 0 ; y < hauteur ; y++){
   Instance old_color = ... // récupérer la couleur du
                            //  cluster (x,y) et la
                            //  transformer en instance


   int indexcluster = clusterer.clusterInstance(old_color) ;
// classifie la couleur
   Instance new_color = centroids.instance(indexcluster) ;
// récupère le centre du cluster de old_color, sous forme d'instance

          new_color_R = ... // composante R de l'instance
          new_color_G = ... // composante G de l'instance
          new_color_B = ... // composante B de l'instance

          image.setRGB(new_color_R, new_color_G, new_color_B,x,y)
```

Vous pouvez afficher des informations sur le clusterer avec la méthode `toString()`. Ce n'est qu'un prototype, certains détails de conversion de types ne sont pas mentionnés.

Question Implémenter la classe `Quantization` avec une méthode `void quantization(int k, MyImage img)` qui réalise la conversion d'une image en k couleurs en une image à k' couleurs.

Question Ecrire la méthode `main` pour tester (elle devrait prendre en argument le nom du fichier d'entrée, en format PNG, et le nombre de couleurs k'). Utiliser par exemple l'outil `display` pour afficher les images.

Question Ecrire une méthode qui affiche en noir un cluster d'indice i sur l'image. Il suffit de ne modifier que les pixels dont la couleur est dans le cluster i, et de les remplacer par la couleur noir. De la même manière, au lieu de remplacer par noir, augmenter la composante rouge, ou verte, ou bleu des pixels dont la couleur est dans le cluster i.

3 Améliorations

La méthode précédente est un peu coûteuse car elle crée autant d'exemples qu'il y a de pixels. Beaucoup d'exemples sont alors superposés (autant qu'il y a de pixels d'une même couleur). Il est possible de donner des poids aux instances. Ainsi, au lieu de créer 100 exemples de même coordonnées, ont en crée un seul avec un poids 100. Cela est possible via la méthode `Instance.setWeight(douple w)`.

Question Quel est l'effet de ce changement sur la complexité de la quantization ? (en coût uniforme). Ajouter cette amélioration à votre implémentation, au niveau de la construction de l'ensemble des instances. *Vous devrez d'abord construire l'histogramme des couleurs, i.e. par exemple une hash table qui associe à chaque couleur (R,G,B) le nombre de pixels qui ont cette couleur, profitez-en pour afficher le nombre de couleurs de l'image*