Projet K-means

RAYAN

December 7, 2022

Contents

1 Introduction

K-means ou encore K-moyennes est un algorithme de clustering. Il permet d'analyser un jeu de données puis rassemble ces données sous un même descripteur qui est similaire, ce qui composera ces clusters.

La similarité entre ces données peut être décrite par la "distance" entre ces données et le centroïde, plus une donnée est proche d'un centroïde, plus ce dernier sera similaire au autre qui sont aussi proche de ce centroïde.

Dans une image, cela reviendrait à prendre des couleurs similaires et leurs données une couleur propre au centroïde à qui ils sont le plus proche ce qui composera un cluster sous une certaine couleur.

L'algorithme peut se voir en quatre-temps dans l'espace couleur auquel nous travaillons :

- Initialisé K cluster dans l'espace de couleur (leur donnée une couleur par exemple aléatoirement).
- Regrouper les pixels de l'image aux centroïdes qui sont le plus proche par rapport à leur couleur.
- Faire la moyenne des couleurs associé au centroïde le plus proche pour définir la nouvelle couleur des pixels.
- Appliquer la nouvelle "couleur moyenne" au centroïde.
- Recommencer si les centroïdes ne sont pas stabilisés.

2 Le code

2.1 Création de la Clut de couleur.

```
typedef struct CLUT {
  int frequence;
  int r,g,b;
} Clut;
```

On parcourt initialement une première fois l'image, où on récupère toutes les couleurs de l'image dans le tableau de Clut. Par la suite, le tableau de Clut est trié grâce au H code et au quicksort.

```
int ColorToInt(int r, int g, int b){
  int bits = 8;
  int rgb = 0;

  int nb = 0;
  for(int i = bits ; i > 0; i --){
    rgb = rgb  | ((1&g)<<nb);
    g = g >> 1;
    nb++;

    rgb = rgb |((1&r)<<nb) ;
    r = r >> 1;
    nb++;
```

```
    rgb = rgb | ((1&b)<<nb) ;
    b = b >> 1;
    nb++;
  }
  return rgb;
}
```

Le H code permet de transformer un triplet RGB en un nombre, on applique grâce à cela cette fonction sur tous les éléments du tableau pour pouvoir le trier avec quicksort.

```
int compare (const void * first, const void * second ){
  Clut* tmp = (Clut*)first;
  Clut* tmp2 = (Clut*)second;

  int a =ColorToInt(tmp->r,tmp->g,tmp->b);
  int b = ColorToInt(tmp2->r,tmp2->g,tmp2->b);
  return (int) (a -b);
}

qsort(clut, nb, sizeof(Clut*), compare);
```

Ici, on donne le tableau en argument avec le nombre d'éléments à l'intérieur, sa taille et la fonction qui va comparer les éléments.

On crée un nouveau tableau qui contiendra que la couleur et où ces occurrences sont juste pris en compte pour augmenter le nombre de fréquence que cette couleur a dans l'image.

Exemple d'image ou les chiffres représentes une couleur :

2 3 1
1 4 3
2 1 2

-> | 2 | 3 | 1 | 1 | 4 | 3 | 2 | 1 | 2 | (On prend TOUTES les couleurs de l'image)

-> | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | (On trie avec le H code le tableau)

-> | 1 | 2 | 3 | 4 | (On enlève les occurrences et on augmente la fréquence d'apparition de la couleur dans l'image)

-> | 1 | 2 | 4 | (Imaginons que la couleur 4 est trop proche des autre dans la palette)

2.2 Initialiaser les K centroïdes.

Les centroïdes sont une structure composée : du RGB, une variable tmp (pour vérifier combien de couleur on été associé au centroïde au dernier cycle), NbcA (nombre de couleur associé), MoyCr (moyenne couleur rouge), etc.

```
typedef struct Centre {

  int r,g,b;
  int tmp;
  int NbcA;
  int MoyCr;
  int MoyCg;
  int MoyCb;

} Centroid;
```

Dans le code, les k centroïdes sont demandé à l'utilisateur lorsqu'il veut utiliser la fonction.

Après avoir choisi, un tableau de centroïde sera crée avec K éléments au maximum, chaque cluster prendra : 1 la couleur du tableau de Clut avec la plus grande fréquence en priorité, si toutes les couleurs ont été prises et que le nombre de centroïde a initialiser n'est toujours pas terminé alors le nombre de centroïde s'en retrouvera réduite, car la palette de couleur n'est pas assez grande.

2.3 Associer les couleurs des pixels au centroïde pour former des clusters.

On explore l'image pixel par pixel, pour chaque RGB du pixel, on regarde quelle centroïde il est le plus proche, on lui associe sa couleur dans la moyenne de couleur du centroïde pour sa moyenne.

```
int bestcentroide(int r, int g, int b, Centroid** tabcentr, int nbcentr){
  float dist = 999999.0;
  int posClusterwin = 0;
  for(int i =0; i< nbcentr; i++){
    float tmp = ( sqrt(carre(r-tabcentr[i]->r)+carre(g-tabcentr[i]->g)+carre(b-tabcent
    if( tmp < dist){
      dist = tmp;
```

```
      posClusterwin = i;
    }

  }

  return posClusterwin;
}
```

Étant donnée une couleur RGB, on cherche quelle centroïde est le plus proche et on renvoie sa position.

```
for(int i =0; i < nb_pixel; i++){
   val1 = *crt;
   val2 = *crt+1;
   val3 = *crt+2;

   int bestcentroid = bestcentroide(val1,val2,val3,clust,nbCentr);

   clust[bestcentroid]->NbcA++;
   clust[bestcentroid]->MoyCr += val1;
   clust[bestcentroid]->MoyCg += val2;
   clust[bestcentroid]->MoyCb += val3;
 }
```

On explore donc toute l'image en entier, on recherche donc bien quelle est le centroïde le plus proche, on associe le RGB du pixel a la moyenne des couleurs du centroïde .

2.4 Faire la "moyenne" des couleur associées aux centroïdes.

Suite à l'association des couleurs de l'image et des centroïdes, on effectue la moyenne des couleurs associées aux centroïdes, on explore à nouveau l'image pour donner la nouvelle couleur moyenne aux pixels et par la suite les nouvelles couleurs sont attribuées au centroïdes.

```
void MoyCent(Image* img,Centroid ** clust, int nbCentr){
  GLubyte *crt = img->data;
  int nb_pixel;
  int val1,val2,val3;
  nb_pixel =  img->sizeX * img->sizeY;
  for(int i =0; i < nb_pixel; i++){
```

```
    val1 = *crt;
    val2 = *crt+1;
    val3 = *crt+2;
    int bestcentroid =  bestcentroide(val1,val2,val3,clust,nbCentr);
    *crt++ = clust[bestcentroid]->MoyCr / clust[bestcentroid]->NbcA;
    *crt++ = clust[bestcentroid]->MoyCg / clust[bestcentroid]->NbcA;
    *crt++ = clust[bestcentroid]->MoyCb / clust[bestcentroid]->NbcA;
  }
  for(int i =0; i< nbCentr; i++){
    if( clust[i]->NbcA != 0){
      clust[i]->r =  clust[i]->MoyCr /  clust[i]->NbcA;
      clust[i]->g =  clust[i]->MoyCg /  clust[i]->NbcA;
      clust[i]->b =   clust[i]->MoyCb /  clust[i]->NbcA;
    }
  }
}
```

2.5 Nouveau cycle ?

On effectue après la moyenne, une comparaison sur les centroïdes de combien de couleurs on été associé au dernier cycle à celui de maintenant, si le résultat n'est pas satisfaisant (trop de changement), on refait un nouveau cycle jusqu'à stabiliser les centroïdes.

```
int finished(Image*img,Image*svg,Centroid ** clust, int nbCentr){
  int changement =0;
  for(int i =0; i < nbCentr; i++){
    changement += abs(clust[i]->tmp - clust[i]->NbcA);
  }
  if(changement  <= 500) return 1;
  int size = svg->sizeX * svg->sizeY;
  for(int i = 0,t = 0; i < size; i++,t+=3){
    img->data[t] = svg->data[t];
    img->data[t+1] =  svg->data[t+1];
    img->data[t+2] = svg->data[t+2];
  }
  return 0;
}
```

On compare chaque centroïde a sa version précédente si le nombre de

changement est inférieur ou égale à 500 alors on a fini. Sinon, repart sur un nouveau cycle en remettant l'image de basse à son état d'origine.

3 Les problématiques rencontrées

Mon principal problème, que j'ai pu rencontrer, fut de comprendre le thème et surtout de me détacher de l'espace pixel. Mes premiers testes m'on mené à faire un semi voronoid, et je n'arrivai pas à me détacher de l'espace pixel, mais grâce à mes professeures, j'ai pu comprendre et essayer d'appliquer leur raisonnement. Dans un deuxième temps, j'ai eu du mal a stabiliser les centroïdes à la fin de chaque cycle.

4 Ce que j'ai réussi à faire

Dans un premier temps, j'ai réussi à faire la Clut de couleur, triée et réduite pour optimiser les couleurs à choisir initilalement. De plus, j'ai aussi réussi à associer une couleur à un centroïde pour ensuite faire sa "moyenne de couleur" et lui associer sa nouvelle couleur. Finalement, j'ai réussi à stabilser les clusters pour que le programe se finisse et donne un résultat plutôt convenable.

5 Conclusion

Ce projet m'a permis de faire un premier pas sur: les algorithmes de clusterisation et finalement sur les images. Ce fut assez plaisant de travailler sur ce projet, car on peut voir un vrai retour graphique et on peut voir ce qu'on fait même si cela ne signifie pas qu'on comprend toujours ce que l'on fait.