Rapport RdF - TP10: Arbres de décision et reconnaissance de visages

Benjamin Van Ryseghem - François Lepan

11 avril 2013

Introduction

Dans ce rapport nous allons voir comment faire pour reconnaitre des visages par le biais d'arbres de décision. Nous verrons comment préparer les données, la construction de l'arbre, et sur quels critères se baser pour choisir une branche de l'arbre.

1 Préparation des données

En entrée nous possédons une image faisant 660*800 pixels contenant 10 exemples de 40 visages soit 400 visages. Une première chose à faire est de créer une pile de visage afin de faciliter les calculs par la suite.

Pour ce faire nous allons utiliser la fonction suivante :

```
// Créé un tableau de 40*33*400
// à partir d'un tableau de 660*800
function sf = stackedFaces(allFaces)
    sf = zeros(40,33,400); // numLignes, numColonnes, numFaces

    for i=0:19
        for j=0:19
            sf(:,:, (i*20 + j + 1) ) = allFaces( 1+i*40 : (i+1)*40 , 1+j*33 : (j+1)*33 );
        end
    end
endfunction
```

Cette fonction retourne une matrice de taille 40*33*400. Les dix premières images seront la première classe, les dix suivantes la seconde classe, etc.

Maintenant que nous avons cette pile d'images nous allons la séparer en deux matrices. La première contiendra les 9 premières images de chaque classe servant à la création de l'arbre et la seconde contiendra les dernières images de chaque classe pour effectuer les tests.

Voici la fonction permettant cette séparation :

```
// séparation des visages:
//  on retourne les derniers visages de chaque classes dans une matrice
//  et les 9 premier de chaque classe dans une autre matrice
function [treePool, pickPool] = getPools(stackedFace)
    pickIndexes = [1:40] * 10;

    // On créer un tableau contenant des chiffres de 1 à 10
    // [1, ..., 10, 1, ..., 10, ...]
    treeIndexes = modulo([0:399], 10) + 1;
    // on ne conserve les indices que pour les valeurs différentes de 10
    // c'est à dire les 9 premières images de chaque classe
    treeIndexes = find(treeIndexes ~= 10);

    pickPool = stackedFace(:, :, pickIndexes);
    treePool = stackedFace(:, :, treeIndexes);
endfunction
```

Maintenant que les données sont prêtent à l'emploi nous allons voir comment construire l'arbre de décisions.

2 Méthodologie

Afin de construire l'arbre de décisions nous allons choisir comme critère de sélection un pixel de l'image. Mais pour savoir quel pixel choisir il va falloir calculer la variation d'entropie pour chaque pixel du sous-ensemble choisi (à la première itération on prend toutes les images).

Ensuite on choisit parmi toutes les valeurs d'entropie celle qui est la plus significative (i.e la plus grande).

Et lorsqu'on a choisi le pixel, on sépare l'ensemble courant en deux sous-ensemble (l'un contenant les visages qui possèdent ce pixel et ceux qui ne le possèdent pas).

Enfin on regarde si l'image pour laquelle on cherche la classe possède ce pixel afin de descendre dans l'arbre.

Il faudra recommencer ces étapes (choix pixel, séparation ensemble, choix ensemble) tant que l'ensemble courant possèdent plus d'une classe.

Voici les fonctions permettant d'effectuer ces calculs :

```
// Calcul de l'entropie pour les images contenu dans stackedFace
function m = entropie(stackedFace)
    // on fait la moyenne des valeurs des pixels
    // pour toutes les images
    m = mean(stackedFace,3);

    // calcul de l'entropie
    m = - log2(m.^m) - log2((1 - m).^(1 - m));
endfunction
```

```
// Fonction de partage:
// permet de séparer un ensemble en deux sous ensemble
// en fonction du pixel choisi
function [A,B,i] = partage(I,stackedFace)

    // récupère les visages significatifs
    significatif = stackedFace(:,:,I);

    // calcul l'entropie de ces images
    entrop = entropie(significatif);

    // récupère l'indice du pixels le plus significatif
    [m,i] = max(entrop);

    x = i(1);
    y = i(2);

    // on met la 3ème dimension dans la première
    stckedFace(1,:) = stackedFace(x,y,:);

    // partage les visages significatifs en deux sous-ensembles
    // ceux qui possède ce pixel à 1
    A = I & (stckedFace == 1);

    // ceux qui ne le possède pas
    B = I & (stckedFace == 0);
endfunction

// Cette fonction verifie si il ne reste plus qu'une seule classe
// parmis celles proposer dans tabClasses grace a la fonction tabul
function index = onlyOneClass(tabClasses)
    // stock les elements de cette facon
    //      classe ocurrence
    //ex:     1     3
    //        3     1
    //        4     1
    //        5     2
    t = tabul(tabClasses)

    // Donc si il ne reste plus qu'une ligne
    // il ne reste plus qu'une classe et on la retourne
    if (size(t,1) == 1) then
         index = t(1);
    // sinon on retourne -1
    else
        index = -1;
    end
endfunction
```

```
// fonction principale
//      - trouve a quel classe corespond aFace
//      - utilise stackedFace comme base d'apprantissage
//      - tabClasses sert a savoir si dans le noeud courant
//        il reste plus d'une classe
function ind = findClass(stackedFace, aFace, tabClasses)

    // initialisation du vecteur de booleen pour le partage
    n = size(stackedFace,3);
    I = ([1:n]>0)

    index = -1;

    // Tant qu'il reste plus d'une classe dans un noeud
    // voir fonction onlyOneClass
    while (index == -1) do

        // On partage en deux sous-ensembles
        [A,B,i]=partage(I)

        x = i(1);
        y = i(2);

        // Si l'image contient le pixel en x y
        // on va en A
        if aFace(x,y) == 1 then
            // Est-ce qu'il reste une seule classe ?
            index = onlyOneClass(tabClasses(A));
            I=A;

        // Sinon on va en B
        else
            index = onlyOneClass(tabClasses(B));
            I=B;
        end
    end

    ind = index;
endfunction
```

Conclusion

Après avoir exécuter l'algorithme on n'obtient pas de très bon résultat. En effet pour connaître la classe de l'image à rajouter on ne se base que sur un critère (ici un pixel de l'image). Si on voulait de meilleur résultat il faudrait rajouter d'autres critères comme la couleur moyenne de l'image, le voisinage du pixel, etc. Mais malgré cela l'algorithme est simple à mettre en oeuvre.