Repte 8 - Visió per Computador

Martí Caixal (1563587)

11 Març 2022

1 Introducció

Al repte 8 es demana realitzar una classificació de terrenys a vista d'ocell. L'objectiu és obtenir una imatge resultant on només es mostren les zones amb arbres.

El treball es pot dividir en els següents apartats:

- Definir mostres manualment.
- Crear descriptors i dataset.
- Entrenar models classificadors.
- Aplicar models a la imatge sencera.

2 Imatge utilitzada

La figura 1 mostra la imatge utilitzada durant el desenvolupament del treball. La mida és, aproximadament, de 400x300 píxels.

Figura 1: Imatge original

Com a punt a destacar, cal fixar-se que la proporció d'arbres a la imatge sigui propera al 50%. Més endavant s'explica detalladament el per què.

3 Definir mostres

Per assolir l'objectiu final és necessari l'ús de classificadors. Aquest cas permet fàcilment definir unes mostres prèvies ben classificades, podent així utilitzar models supervisats.

La figura 2 mostra els píxels utilitzats per extreure'n les dades.

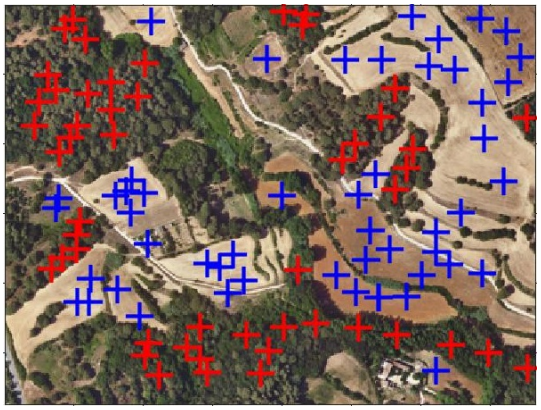

Figura 2: Punts utilitzats

Diferenciant per colors, ambdues classes tenen 50 mostres amb l'objectiu que les dades estiguin balancejades durant l'entrenament i proves.

A l'hora de tria les mostres, és important agafar per una mateixa classe mostres que siguin el més diferent entre elles possible. Per exemple, la classe que no pertany a arbre ha de tenir mostres de prats secs, prats verds, camins, cases, ...

4 Descriptors

Un descriptor no és altre cosa que trobar números que permetin definir el que es veu de manera visual a certa posició de la imatge. D'aquesta manera els models classificadors són capaços de buscar similituds i diferències entre les classes.

Per poder entrenar els classificadors, cal trobar el descriptor de cadascun dels píxels seleccionats anteriorment. Una manera de descriure les característiques és mitjançant els colors. El programa ho fa de la següent manera per cada píxel:

1. Seleccionar un cub MxM al voltant del píxel.
2. Separar les 3 capes de colors.
3. Per cada capa, sumar el valor de tots els elements de la matriu.

Tenir aquesta separació per colors permet als classificadors poder distingir entre un color i un altre. Si únicament es fes un sumatori de dels valors de les 3 capes, un classificador només podria diferenciar entre la intensitat.

```
function desc = get_color_descriptor(img,
    points, radius)
    for i=1:size(points, 1)
        A = get_block(img, points, radius
            , i);
        r = sum(A(:,:,1), 'all');
        g = sum(A(:,:,2), 'all');
        b = sum(A(:,:,3), 'all');
        desc(i,1) = r;
        desc(i, 2) = g;
        desc(i, 3) = b;
    end
end
```

El dataset, conjunt de dades utilitzat per entrenar un model, està compost per les característiques que permeten identificar una classe i l'anomenat *Ground Truth*. Aquest segon terme és la classificació correcte de cada mostra.

Addicionalment, cal tenir present que, si bé el conjunt de descriptors prové d'un espai de 2 dimensions, ara s'ha de passar la matriu a un vector. Quan es rebin més endavant les prediccions, serà necessari passar el vector a la seva forma i dimensió original.

Un cop el dataset està creat, es pot passar a entrenar i testejar els models.

5 Models classificadors

Els models classificadors es basen en separar un seguit de mostres en un espai discret. És a dir, assignar a cada mostra una classe que, normalment, ja és coneguda. Es diferencien dels regressors perquè aquests últims busquen trobar un valor dins d'un interval continu.

Els models utilitzats són:

- Naive Bayes
- K-Nearest Neighbours
- SVM

Com a punts a destacar, pel SVM s'ha realitzat proves amb kernels lineal, polinomial i RBF, obtenint sempre un resultat pràcticament idèntic. Sembla ser que les mostres estan representades sobre l'espai de manera que linealment es poden separar, així que no fa falta kernels més potents i que requereixen més recursos com els polinomials. Per altra banda, el KNN ha funcionat millor quan el nombre de veïns és de 5. Al següent apartat s'explicarà què passa quan el nombre de veïns és massa gran o petit.

Les proves d'entrenament han donat unes mètriques satisfactòries pels tres models.

	NB	KNN	SVM
Accuracy	96%	98%	100%
Precision	93%	96%	100%
Recall	100%	100%	100%
F1-Score	96%	98%	100%

Taula 1: Mètriques de cada model

Si bé els quatre tipus de mètriques tenen un bon valor, és interessant saber què vol dir cada una d'elles. Resulten especialment útils quan les dades amb les que es treballa no estan balancejades.

- Accuracy: percentatge de prediccions ben fetes.
- Precision: dels classificats com a arbres, quants realment ho són.
- Recall: de tots els arbres que existeixen, quants ha trobat.
- F1-Score: Mètrica que només és alta quan tant la precission i el recall són alts. S'utilitza quan ambdues classes són igual d'importants.

Un clar exemple de mètriques enganyoses és un hospital amb 100 pacients, 10 dels quals tenen càncer. Si un classificador diu que els 100 pacients estan sans, l'accuracy és del 90%. Però clarament, l'objectiu que és identificar les persones malaltes no s'ha complert.

Addicionalment, també hi ha la matriu de confusió que permet veure d'una forma més visual el resultat de les prediccions.

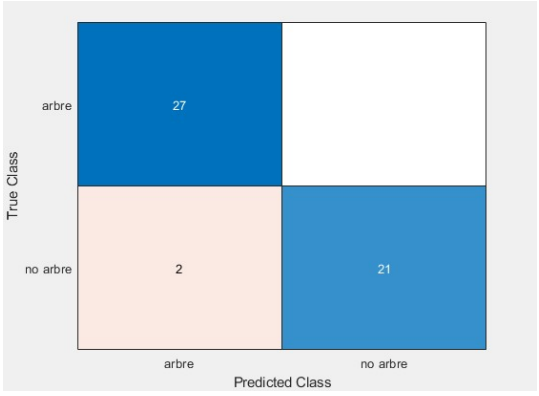

Figura 3: Matriu de confussió per Naive Bayes

Dels models superiors, la màquina de vectors de suport és la que ha aconseguit el millor resultat. Si més no, només s'estaven utilitzant 50 mostres per l'etapa de *training* i 50 per l'etapa de *test*. Amb la poca diferència dels tres models i la baixa quantitat de mostres, no es pot dir a ciència certa quin és el millor.

6 Aplicar models entrenats

Una vegada els classificadors estan entrenats i tenen bons resultats, es procedeix a aplicar-los a la imatge sencera.

Ara és el moment de transformar el resultat del classificador a la seva forma original. És a dir, passar el vector amb les classificacions a una matriu de 2 dimensions que actua com a màscara. Les figures 4, 5 i 6 mostren la màscara resultant per cada model.

Les tres són molt semblants, però es pot veure com el Naive Bayes té alguns punts que no detecta com a arbres.

Per obtenir la imatge final, cal multiplicar punt a punt la imatge original amb la màscara desitjada. De la mateixa manera, es pot calcular el complement de la màscara per acabar obtenint una imatge amb totes les zones que no estan formades per arbres.

Les figures 7, 8 i 9 mostren el resultat final per cada classificador.

Figura 4: Mask NB

Figura 5: Mask KNN

Figura 6: Mask SVM

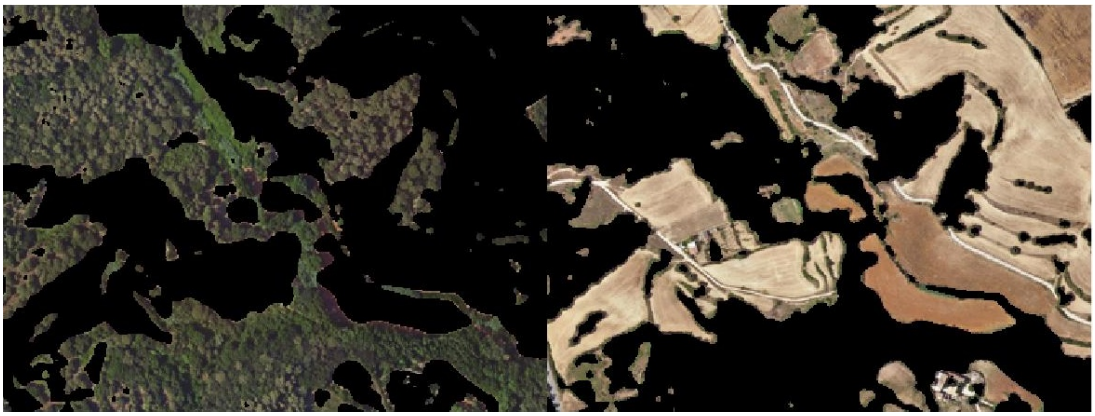

Figura 7: Resultat Naive Bayes

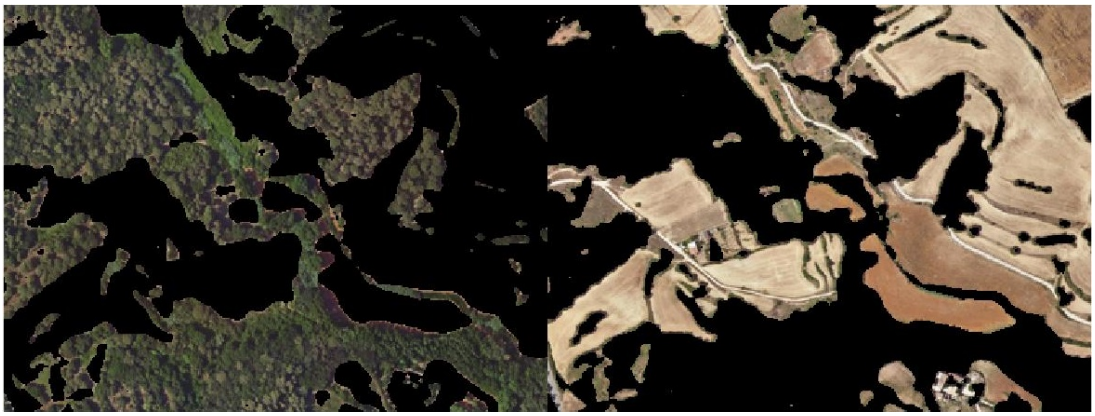

Figura 8: Resultat KNN

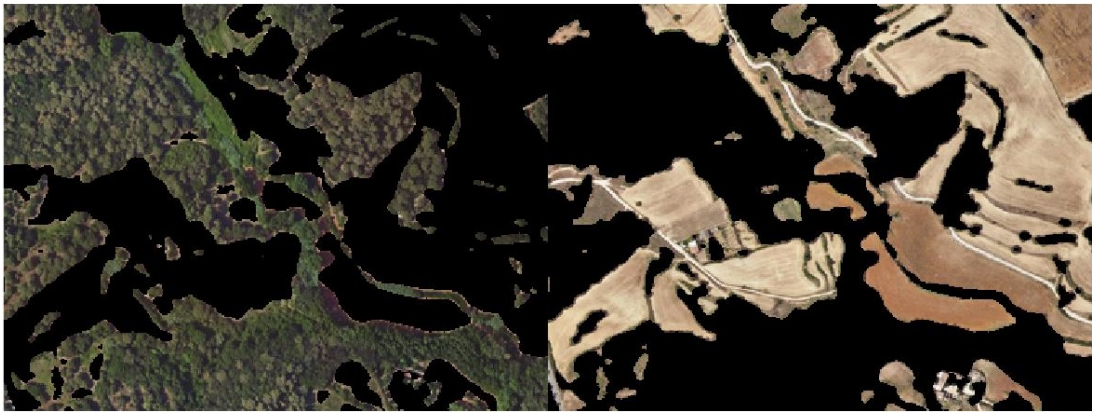

Figura 9: Resultat SVM

7 Males classificacions

Tot i tenir un resultat general bastant bo, els tres classificadors semblen haver fallat a la mateixa zona de la imatge. La figura 10 permet veure un prat verd que ha estat detectat com a zona amb arbres.

Aquest és un clar exemple de la importància de les mostres utilitzades durant l'entrenament. Els classificadors, realment, en cap moment han fallat, sempre s'han comportat com s'espera d'ells. El problema, doncs, està a les dades que han utilitzat durant la fase de *training*.

Mirant de nou la figura 2, els punts que actuen com a mostres no contemplen tots els tipus de zones diferents. La zona mal classificada no forma part de cap mostra de zones sense arbres. Els classificadors, per tant, intenten classificar els descriptors del prat verd a la classe més similar i acaben formant part de la classe "Arbre".

Figura 10: Zona mal classificada

Annex: Codi del programa

```
%% init vars
radius = 3;
k_neighbours = 5;
desc_type = 'color';

%% read data
img = imread('../input/img.jpg');
img = imresize(img, 0.3);
load('../data/no_trees.mat')
load('../data/trees.mat')

%% print points
imshow(img)
axis on
hold on;
for r=1:size(tree, 1)
    plot(tree(r, 1), tree(r, 2), 'r+', 'MarkerSize', 15, 'LineWidth', 2)
    plot(no_tree(r, 1), no_tree(r, 2), 'b+', 'MarkerSize', 15, 'LineWidth', 2)
end

%% creating descriptors
desc_tree = get_descriptor(img, tree, radius, desc_type);
desc_tree(:,end+1) = 1;
desc_no_tree = get_descriptor(img, no_tree, radius, desc_type);
desc_no_tree(:,end+1) = 0;

%% creating dataset
```

```
df = cat(1, desc_tree, desc_no_tree);
df = df(randperm(size(df, 1)), :);
cv = cvpartition(size(df,1),'HoldOut',0.5);
idx = cv.test;
train = df(~idx,:);
test = df(idx,:);
X_train = train(:,1:end-1);
y_train = train(:,end);
X_test = test(:, 1:end-1);
y_test = test(:, end);

%% KNN Classifier
% mdl = fitcknn(X_train, y_train, 'NumNeighbors',k_neighbours, 'Standardize',1);

%% SVM Binary Classifier
mdl = fitcsvm(X_train, y_train);

%% Naive Bayes Classifier
% mdl = fitcnb(X_train, y_train);

%% prediction
[y_pred, score, cost] = predict(mdl, X_test);
display_metrics(y_pred, y_test);

%% full image data
dim1_points = radius+1:size(img, 1)-radius-1;
dim2_points = radius+1:size(img, 2)-radius-1;
[n, m] = ndgrid(dim1_points, dim2_points);
all_points = [m(:), n(:)];
desc = get_descriptor(img, all_points, radius, desc_type);

%% predicion full image
[y_pred, score, cost] = predict(mdl, desc);
show_results(y_pred, dim1_points, dim2_points, img, radius, "Naive Bayes");

%% functions
function show_results(y_pred, dim1_points, dim2_points, img, radius, model)
    y_pred_mask = reshape(y_pred, [size(dim1_points, 2), size(dim2_points, 2)]);
    y_pred_mask_complement = imcomplement(y_pred_mask);
    figure, imshow([y_pred_mask, y_pred_mask_complement]);

    img_cropped = img(radius+1:size(img, 1)-radius-1, radius+1:size(img, 2)-radius-1, :);
    img_tree = img_cropped .* cast(y_pred_mask, 'uint8');
    img_no_tree = img_cropped .* cast(y_pred_mask_complement, 'uint8');
    figure, imshow([img_tree, img_no_tree]);
    title(model);
end

function display_metrics(y_pred, y_test)
    c = confusionmat(y_pred,y_test);
    labels = categorical({'no arbre', 'arbre'});
    figure, confusionchart(c, labels);
    true_positives = c(2,2);
    false_positives = c(1,2);
    false_negatives = c(2,1);
    true_negatives = c(1,1);
    accuracy = (true_positives + true_negatives) / (true_positives + true_negatives +
        false_positives + false_negatives);
    precision = true_positives / (true_positives+false_positives);
    recall = true_positives / (true_positives+false_negatives);
    f1_score= 2* precision * recall / (precision + recall);
    fprintf("\n")
    disp("Accuracy (confusion Mat): ");
    disp(accuracy)
    disp("Precicion (confusion Mat): ");
    disp(precision)
```

```
    disp("Recall (confusion Mat): ");
    disp(recall)
    disp("F1-score (confusion Mat): ");
    disp(f1_score)
end

function desc = get_descriptor(img, points, radius, type)
    if (strcmp(type, 'color'))
        desc = get_color_descriptor(img, points, radius);
        return;
    end
    if (strcmp(type, 'lbp'))
        radius = 1;
        desc = get_lbp_descriptor(img, points, radius);
    end
end

function desc = get_lbp_descriptor(img, points, radius)
    if size(img, 3) > 1
        img = rgb2gray(img);
    end
    [m, n] = size(img);
    % mirror-reflecting
    I = img;
    I = cat(1, I(1, :), I, I(end, :));
    I = cat(2, I(:, 1), I, I(:, end));
    w = uint8(2 .^ (7:-1:0));

    LBP = zeros(m, n);
    for i = 2:m+1
        for j = 2:n+1
            neighbor = I(i-1:i+1, j-1:j+1);
            neighbor(neighbor< neighbor(2, 2)) = 0;
            neighbor(neighbor>=neighbor(2, 2)) = 1;
            neighbor = [neighbor(1), neighbor(4), neighbor(7), neighbor(8), neighbor(9),
                neighbor(6), neighbor(3) ,neighbor(2)];
            n0 = sum(w .* neighbor);
            n1 = sum(w .* (circshift(neighbor,-1, 2)));
            n2 = sum(w .* (circshift(neighbor,-2, 2)));
            n3 = sum(w .* (circshift(neighbor,-3, 2)));
            n4 = sum(w .* (circshift(neighbor,-4, 2)));
            n5 = sum(w .* (circshift(neighbor,-5, 2)));
            n6 = sum(w .* (circshift(neighbor,-6, 2)));
            n7 = sum(w .* (circshift(neighbor,-7, 2)));

            LBP(i-1, j-1) = max([n0 n1 n2 n3 n4 n5 n6 n7]);
        end
    end

    for i=1:size(points, 1)
        desc(i, 1) = LBP(uint8(points(i, 2)), uint8(points(i, 1)));
    end
end

function desc = get_color_descriptor(img, points, radius)
    for i=1:size(points, 1)
        A = img(points(i, 2)-radius:points(i, 2)+radius, points(i, 1)-radius:points(i, 1)
            +radius, :);
        r = sum(A(:,:,1), 'all');
        g = sum(A(:,:,2), 'all');
        b = sum(A(:,:,3), 'all');
        desc(i,1) = r;
        desc(i, 2) = g;
        desc(i, 3) = b;
    end
end
```