Model selection and evaluation

Eva Cernadas
FMLCV Course
CITIUS: Centro de Tecnoloxías Intelixentes da USC
Universidade de Santiago de Compostela

24 de noviembre de 2021

We will develop the practical contents of classification starting from an exercise that classifies images based on their textures. The programming language of the example provides is `matlab`, but the exercises may be developped in the other language. The specific objectives are:

1. Compute some texture features for grey-level images.
2. Classify the images using differents classification models and validation measures.

Specifically, we use two texture features: Harakick's features of the co-ocurrence matrix and Local Binary Patterns (LBP) (see section 1 for a brief introduction and the reference [1] for a wider description). Section 2 briefly describes the material provided to the lab exercises and section 3 enumerates the exercises to do the students.

1. Texture features

Let $I(\mathbf{z}) \in G$ be the grey-level of the image in the point $\mathbf{z} = (x, y)$, $x\,, y \in N)$ and $G = 0, 1, \ldots, 255$ the posible number of grey levels.

1.0.1. Coocurrence matrix and Haralick's features

The Grey Level Coocurrence Matrix (GLCM) describes the probability of finding two given pixel values in a predefined relative position in the image. The spatial displacement describes the scale and orientation between two points in the image lattice. A matrix is obtained for each scale and orientation. The main problem of GLCM is to choose the appropiate set of scale and orientation parameters that effectively capture the structural information of texture. We average the matrices for each scale and all orientations. From the GLCM matrices, we compute the following features for each scale: contrast, homogeneity, correlation and energy. In `matlab`, the option *offset* in function `graycomatrix` (in the `image processing toolbox`), you can configure the neighborhoods.

1.0.2. Local Binary Patterns (LBP)

The LBP operator describes each image pixel by comparing each pixel with its neighbors. Precisely, for each neighboring pixel, the result will be set to one if its value is higher than the value of central pixel, otherwise the result will be set to zero. The LBP code of the central pixel is then obtained by multiplying the results with weights given by powers of two, and summing then up together. The histogram of the binary patterns computed over all pixels of the image is generally used for texture description. The final LBP feature vector is very fast to compute and is invariant to monotonic illumination changes. The main drawback of LBP features lies in the high dimensionality of histograms produced by LBP codes (if P is the number of neighboring pixels, then the LBP feature will have 2^P distinct values, resulting in a 2^P-dimensional histogram). Many classifiers can not operate with high dimensional patterns. The LBP with *uniform patterns* have been proposed to the dimensionality of original LBP. The uniform patterns are binary patterns with only two transitions (from 0 to 1 and vice versa). It was found that most of the micro-structures such as bright/dark spots and flat regions can be successfully represented by uniform patterns. In a circularly symmetric neighbor set of P pixels can occur $P+1$ *uniform* binary patterns. The number of "1's" in the binary pattern is the label of the pattern, while the nonuniform patterns are labelled by $P+1$. The histogram of the pattern labels accumulated over the intensity image is employed as texture feature vector.

2. Lab exercises programs for classification

In the lab, we will use the image dataset (*suite*) **Contrib_TC_00006**(it can be downloaded from: *http://www.cse.oulu.fi/CMV/ImageData* or the virtual campus). It contains 864 color images of 128×128 pixels belonging to 54 classes (16 images per class). Figura 1 can see an example of each class.

Matlab programs provided to solve the exercises:

1. `clasificadorCoocur.m`: compute the correct classification percentage of images using the texture features *Haralick's features* and the 1NN classifier using the distance L1. Use the half of patterns to train and the other half to test. Return the accuraccy for the test set. It allows to use a distance d or various (multiresolution).

```matlab
%compute Haralick features of the images and apply the 1-NN classifier
clear all;
%images path
pathImaxes='images';
numImaxes=864; %number of images

%neighbours for different distances
offset1 = [0 1; -1 1; -1 0; -1 -1]; %d=1
offset2 = [0 2; -1 2; -2 1; -2 0; -2 -1; -1 -2]; %d=2
```

Figura 1: Images belonging to 54 classes of image dataset.

```
offset3 = [0 3; −1 3; −2 2; −3 1; −3 0; −3 −1;−2 −2; −1 −3];
    % d=3
offset4 = [0 4; −1 4; −2 4; −2 3; −3 3; −3 2; −4 2: −4 1; −4
```

```
0; -4 -1; -4 -2; -3 -2; -3 -3; -2 -3; -2 -4; -1 -4]; %d=4

for i=1:numImaxes;
    f=[];
    filename = sprintf('%s/images/%06d.bmp', pathImaxes, i-1);
    rgb = imread(filename);
    grey = rgb2gray(rgb);
    %Coocurrence matrix for distance d=1
    glcm=graycomatrix(grey, 'offset', offset1, 'Symmetric', true);
    fs=graycoprops(glcm,{'contrast','homogeneity', 'correlation', 'Energy'});
    f=[f mean(fs.Contrast) mean(fs.Correlation) mean(fs.Energy) mean(fs.Homogeneity)];
    %Coocurrence matrix for distance d=2
    glcm=graycomatrix(grey, 'offset', offset2, 'Symmetric', true);
    fs=graycoprops(glcm,{'contrast','homogeneity', 'correlation', 'Energy'});
    f=[f mean(fs.Contrast) mean(fs.Correlation) mean(fs.Energy) mean(fs.Homogeneity)];
    %Coocurrence matrix for distance d=3
    glcm=graycomatrix(grey, 'offset', offset3, 'Symmetric', true);
    fs=graycoprops(glcm,{'contrast','homogeneity', 'correlation', 'Energy'});
    f=[f mean(fs.Contrast) mean(fs.Correlation) mean(fs.Energy) mean(fs.Homogeneity)];
    features(i,:)=f;
end

%read picture ID of training and test samples, and read class ID of
%training and test samples
trainTxt = sprintf('%s/train.txt', pathImaxes)
testTxt = sprintf('%s/test.txt', pathImaxes)
[trainIDs, trainClassIDs] = ReadOutexTxt(trainTxt);
[testIDs, testClassIDs] = ReadOutexTxt(testTxt);

 %classification test
    trains=features(trainIDs', :);
    tests=features(testIDs', :);
```

```
    trainNum = size(trains,1);
    testNum = size(tests,1);

% use L1 distance as metric measure
    [final_accu,PreLabel] = NNClassifierL1(trains',tests',trainClassIDs,testClassIDs);
    accu_list3 = final_accu;
    close all;
```

2. `clasificadorLBP.m`: idem the the previous but using LBP texture features. You can change the radius, number of neighbours and the method (*ri* invariant rotation, *riu2* uniform and invariant to rotations LBP, *u2* uniform LBP and contrast histogram (`lbpvar` function).

```
% apply the 1NN classifier using the LBP texture features
clear all;
% images path
pathImaxes='images';
numImaxes=864; % number of images
mapping=getmapping(8, 'riu2'); % mapping type: radius, number of neighbours and LBP type

% LBP feature computation
for i=1:numImaxes;
    filename = sprintf('%s/images/%06d.bmp', pathImaxes, i-1);
    rgb = imread(filename);
    grey = double(rgb2gray(rgb));
    features(i,:)=lbp(grey,1,8,mapping,'h');
end

% read picture ID of training and test samples, and read class ID of
% training and test samples
trainTxt = sprintf('%s/train.txt', pathImaxes);
testTxt = sprintf('%s/test.txt', pathImaxes);
[trainIDs, trainClassIDs] = ReadOutexTxt(trainTxt);
[testIDs, testClassIDs] = ReadOutexTxt(testTxt);


 % classification test
    trains=features(trainIDs', :);
    tests=features(testIDs', :);
```

```
    trainNum = size(trains,1);
    testNum = size(tests,1);

% use L1 distance as metric measure
    [final_accu,PreLabel] = NNClassifierL1(trains',tests',
       trainClassIDs,testClassIDs);
    accu_list3 = final_accu;
    close all;
```

3. `clasificadorLBPMultiresolucion.m`: idem that `clasificadorLBP.m` but this allows to concatenate LBP features for different radious.

```
% calcula una descritor multiresolucion LBP e aplicar o
    clasificador
clear all;
% images path
pathImaxes='images';
numImaxes=864; % number of images
mapping8=getmapping(8, 'riu2'); % mapping type: radius,
    number of neighbours and LBP type
mapping12=getmapping(12, 'riu2');
mapping16=getmapping(16, 'riu2');

% compute multiresolution features
for i=1:numImaxes;
    mlbp=[];
    filename = sprintf('%s/images/%06d.bmp', pathImaxes, i-1)
       ;
    rgb = imread(filename);
    grey = double(rgb2gray(rgb));
    f=lbp(grey,1,8,mapping8,'h'); %LBP for R=1 and P=8
    mlbp=[mlbp f];
    f=lbp(grey,2,12,mapping12,'h'); %LBP for R=2 and P=12
    mlbp=[mlbp f];
    f=lbp(grey,3,16,mapping16,'h'); %LBP for R=3 and P=16
    mlbp=[mlbp f];
    features(i,:)=mlbp;
end

% read picture ID of training and test samples, and read
    class ID of
% training and test samples
```

```
trainTxt = sprintf('%s/train.txt', pathImaxes)
testTxt = sprintf('%s/test.txt', pathImaxes)
[trainIDs, trainClassIDs] = ReadOutexTxt(trainTxt);
[testIDs, testClassIDs] = ReadOutexTxt(testTxt);


 % classification test
    trains=features(trainIDs', :);
    tests=features(testIDs', :);
    trainNum = size(trains,1);
    testNum = size(tests,1);

% use L1 distance as metric measure
    [final_accu,PreLabel] = NNClassifierL1(trains',tests',
       trainClassIDs,testClassIDs);
    accu_list3 = final_accu;
    close all;
```

4. `getmapping.m`: return the mapping to compute the LBP codes.

```
%GETMAPPING returns a structure containing a mapping table
   for LBP codes.
% MAPPING = GETMAPPING(SAMPLES,MAPPINGTYPE) returns a
% structure containing a mapping table for
% LBP codes in a neighbourhood of SAMPLES sampling
% points. Possible values for MAPPINGTYPE are
%       'u2'   for uniform LBP
%       'ri'   for rotation-invariant LBP
%       'riu2' for uniform rotation-invariant LBP.
%
% Example:
%       I=imread('rice.tif');
%       MAPPING=getmapping(16,'riu2');
%       LBPHIST=lbp(I,2,16,MAPPING,'hist');
% Now LBPHIST contains a rotation-invariant uniform LBP
% histogram in a (16,2) neighbourhood.
%

function mapping = getmapping(samples,mappingtype)
% Version 0.1.1
% Authors: Marko Heikkilï¿½ and Timo Ahonen

table = 0:2^samples-1;
```

```
newMax  = 0; %number of patterns in the resulting LBP code
index   = 0;

if strcmp(mappingtype,'u2') %Uniform 2
  newMax = samples*(samples-1) + 3;
  for i = 0:2^samples-1
    j = bitset(bitshift(i,1,samples),1,bitget(i,samples));
    numt = sum(bitget(bitxor(i,j),1:samples));
    if numt <= 2
      table(i+1) = index;
      index = index + 1;
    else
      table(i+1) = newMax - 1;
    end
  end
end

if strcmp(mappingtype,'ri') %Rotation invariant
  tmpMap = zeros(2^samples,1) - 1;
  for i = 0:2^samples-1
    rm = i;
    r  = i;
    for j = 1:samples-1
      r = bitset(bitshift(r,1,samples),1,bitget(r,samples));
      if r < rm
        rm = r;
      end
    end
    if tmpMap(rm+1) < 0
      tmpMap(rm+1) = newMax;
      newMax = newMax + 1;
    end
    table(i+1) = tmpMap(rm+1);
  end
end

if strcmp(mappingtype,'riu2') %Uniform & Rotation invariant
  newMax = samples + 2;
  for i = 0:2^samples - 1
%     j = bitset(bitshift(i,1,samples),1,bitget(i,samples));
   %original
    j = bitset(bitand(bitshift(i,1),2^samples-1),1,bitget(i,
       samples)); %corrected
```

```
      numt = sum(bitget(bitxor(i,j),1:samples));
      if numt <= 2
        table(i+1) = sum(bitget(i,1:samples));
      else
        table(i+1) = samples+1;
      end
    end
end

mapping.table=table;
mapping.samples=samples;
mapping.num=newMax;
```

5. `lbp.m`: compute the histogram of codes using a certain mapping.

6. `lbpvar.m`: compute the histogram of LBP codes to measure the image contrast.

7. `NNClasifierL1.m`: calculate the accuraccy on a test set using the 1NN classifier. The input arguments are two matrix with the training and testing patterns respectively.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    NN Classifier with L1 distance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Function NNClassifierL1(Samples_Train,Samples_Test,
    Labels_Train,Labels_Test)
%TO calculate the accuracy of the given otesting round and
    obtain the
% predicted labels using the nearest neighbor classifer
%%INPUT Arguments:
% Samples_Train: d x no of training samples matrix
% Samples_Test:  d x no of testing samples matrix
% Labels_Train:  1 x no of training samples vector including
    all the labels of the training samples
% Labels_Test:   1 x no of testing samples vector including
    all the labels of the testing samples
%%OUTPUT Arguments:
% final_accu: the accuracy of this testing round
% PreLabel:       1 x no of testing samples vector including
    all the predicted labels of the testing samples
%
function [final_accu,PreLabel] = NNClassifierL1(Samples_Train
    ,Samples_Test,Labels_Train,Labels_Test)

```

```
Train_Model = Samples_Train;
Test_Model = Samples_Test;
numTest = size(Test_Model,2);
numTrain = size(Train_Model,2);

PreLabel = [];

for test_sample_no = 1:numTest

    testMat = repmat(Test_Model(:,test_sample_no), 1,
        numTrain);
    scores_vec = cal_matrix_distance(testMat, Train_Model);

    [min_val min_idx] = min(scores_vec);
    best_label = Labels_Train(1,min_idx);
    PreLabel = [PreLabel, best_label];
end

Comp_Label = PreLabel - Labels_Test;
final_accu = (sum((Comp_Label==0))/numel(Comp_Label))*100

end

function disVec=cal_matrix_distance(mat1,mat2)
% using L1 as the distance metric
disVec = sum(abs(mat1 - mat2), 1);
% you may add other distance matric here:
end
```

8. `ReadOutexTxt.m`: read the images names and output for the images in the dataset.

```
% [filenames, classIDs] = ReadOutexTxt(txtfile)
% gets picture IDs and class
% IDs from TXT file for Outex Database

function [filenames, classIDs] = ReadOutexTxt(txtfile)

fid = fopen(txtfile,'r');
tline = fgetl(fid); % get the number of image samples
i = 0;
while 1
    tline = fgetl(fid);
    if ~ischar(tline)
```

```
            break;
        end
        index = findstr(tline,'.');
        i = i+1;
        filenames(i) = str2num(tline(1:index-1))+1; %the picture ID starts from 0, but the index of Matlab array starts from 1
        classIDs(i) = str2num(tline(index+5:end));
    end
fclose(fid);
```

3. Exercises to do by the students

The lab work for the students is:

1. Run the provided code and report the accuraccy for each texture feature. Comments about the difficulties founded.

2. Implement and report other measures to estimate the quality of the classifier: the Cohen kappa and the confusion matrix.

3. Include in the above code cross-validation using four folds (k=4) and calculate the accuraccy and Cohen kappa.

4. Implement the cross-validation using four folds and three sets (training, validation and test set). Use a kNN classifier instead of 1NN classifier, tuning the k parameter (number of neighbours) using the validation set. Calculate the Cohen kappa and the value of k.

Submit before 22 December by TEAMS the results and dificulties founded. It can be done individually or by groups.

Referencias

[1] E. Cernadas, M. Fernández-Delgado, E. González-Rufino, P. Carrión, Influence of normalization and color space to color texture classification, Pattern Recogn. 61 (2017) 120 – 138.