Classification: model selection and evaluation
Lab exercices

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

23 de noviembre de 2023

During the master course FMLCV (Fundamentals of machine learning for computer vision), students will do different exercises in order to practice the practical contents of unsupervised and supervised classification models.

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, like `Python`. 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 (see the main program in file `haralick1nn.m`).

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 in subject webpage `images.zip`. 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. `haralick1nn.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).

```
% use the haralick texture features of a grey level image and
    the 1NN
% classifier
```

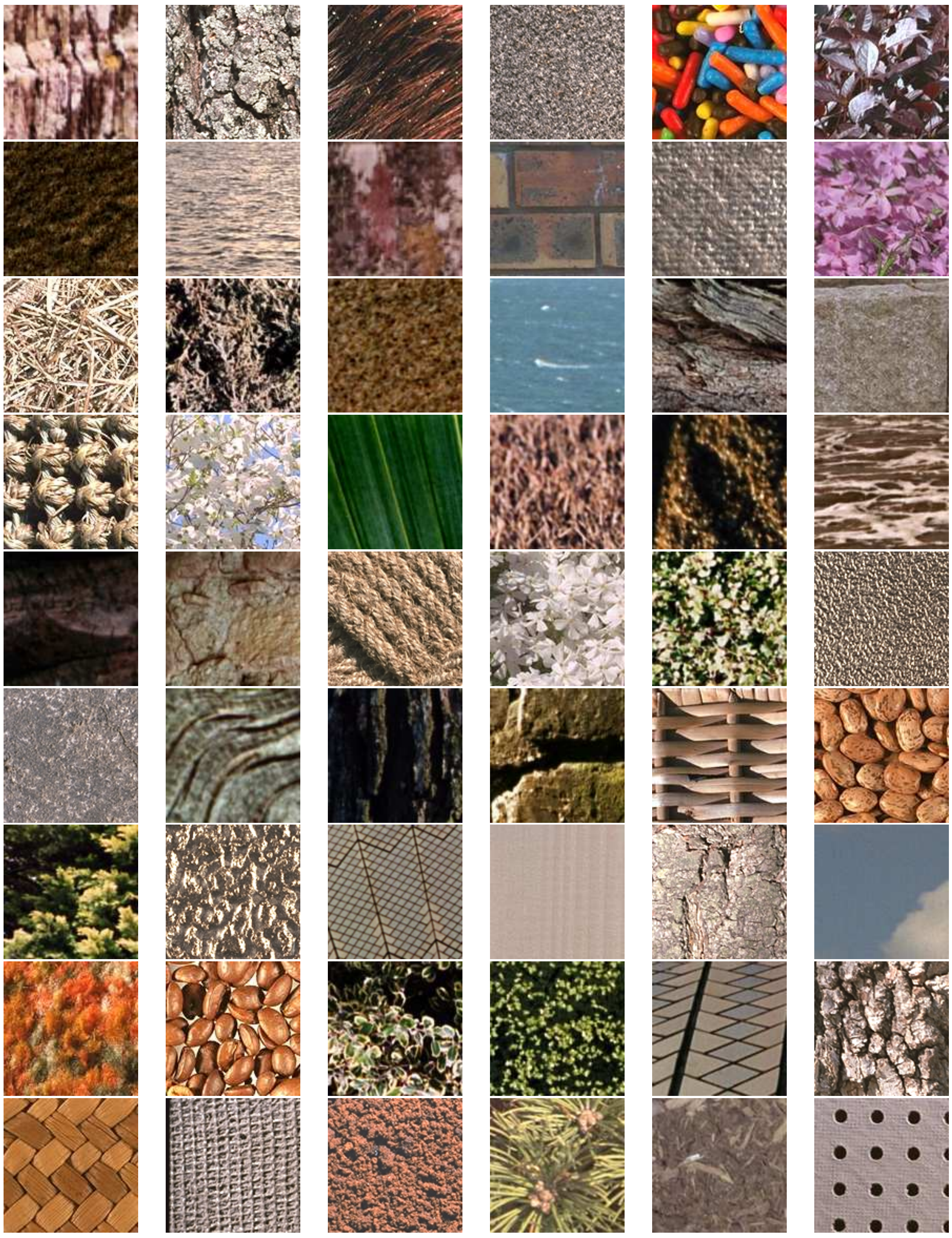

Figura 1: Images belonging to 54 classes of image dataset.

```
clear all;
% image folder
pathImages='/home/cernadas/docencia/mvc/lab/matlab/images/
```

```
    Contrib_TC_00006';
numImages=864; % number of images

% neighborhoods for different scales
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
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:numImages;
    f=[];
    filename = sprintf('%s/images/%06d.bmp', pathImages, 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 distances d=1 and 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=1, d=2 and 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
trainTxt = sprintf('%s/000/train.txt', pathImages)
```

```
testTxt = sprintf('%s/000/test.txt', pathImages)
% read class ID of training and test samples
[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] = NNClassifier_L1(trains',tests',
   trainClassIDs,testClassIDs);
accu_list3 = final_accu;
close all;
```

2. LBP1nn.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).

```
% use the Local Binary Patterns (LBP) texture features of a
   grey level image and the 1NN
% classifier
clear all;
% image folder
pathImages='/home/cernadas/docencia/mvc/lab/matlab/images/
   Contrib_TC_00006';
numImages=864; % number of images
mapping=getmapping(8, 'riu2'); % mapping type: radio,
   neighbours and LBP type

% compute texture features
for i=1:numImages;
    filename = sprintf('%s/images/%06d.bmp', pathImages, 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
trainTxt = sprintf('%s/000/train.txt', pathImages)
testTxt = sprintf('%s/000/test.txt', pathImages)
```

```
% read class ID of training and test samples
[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] = NNClassifier_L1(trains',tests',
    trainClassIDs,testClassIDs);
accu_list3 = final_accu;
close all;
```

3. <u>`mLBP1nn.m`</u>: idem that `LBP1nn.m` but this allows to concatenate LBP features for different radious.

```
% use the multiscalar Local Binary Patterns (LBP) texture
    features of a grey level image
% and the 1NN classifier
clear all;
% image folder
pathImages='/home/cernadas/docencia/mvc/lab/matlab/images/
    Contrib_TC_00006';
numImages=864; % number of images
mapping8=getmapping(8, 'riu2'); % mapping type: radio,
    neighbours and LBP type
mapping12=getmapping(12, 'riu2');
mapping16=getmapping(16, 'riu2');

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

% read picture ID of training and test samples
trainTxt = sprintf('%s/000/train.txt', pathImages)
testTxt = sprintf('%s/000/test.txt', pathImages)
% read class ID of training and test samples
[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] = NNClassifier_L1(trains',tests',
    trainClassIDs,testClassIDs);
accu_list3 = final_accu;
close all;
```

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

5. NNClassifier_L1.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 NNClassifier_L1(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] = NNClassifier_L1(
   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
```

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

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


function [filenames, classIDs] = ReadOutexTxt(txtfile)
% Version 1.0
% Authors: Zhenhua Guo, Lei Zhang and David Zhang
% Copyright @ Biometrics Research Centre, the Hong Kong
   Polytechnic University

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. For students who want to develop the exercises in `Python`, it is provided text files with the texture features (first column is the class label and the remaining columns are the texture features): 1) haralickTrain.txt and haralickTest.txt for the texture features of Haralick's coeficients; 2) lbpTrain.txt and lbpTest.txt for the texture features of LBP using $d = 1$; and 3) mlbpTrain.txt and mlbpTest.txt for the texture features of multidimensional LBP using $d =$

$\{1, 2, 3\}$. You can use the `Python` package `scikit-learn` (the documentation is here, and the code to use with LBP features is:

```
from sklearn.neighbors import *
from sklearn.metrics import *
from numpy import *
tx=loadtxt('lbpTrain.txt')
ty=tx[:,0] # class label, first column
tx=delete(tx,0, 1) # delete first column
sx=loadtxt('lbpTest.txt')
sy=sx[:,0]
sx=delete(sx,0, 1)
model=KNeighborsClassifier(n_neighbors=1).fit(tx, ty)
z=model.predict(sx) # apply classifier
acc=accuracy_score(sy, z)
kappa = cohen_kappa_score(sy, z)
print('Accuracy= %.2f kappa= %.2f\n'%(acc*100, kappa*100))
```

2. Test the above code using the Euclidean distance and report the results.

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

4. Joint the training and test sets and calculate the performance using cross-validation with four folds (K=4) and the classifier 1NN. Calculate the accuraccy and Cohen kappa.

5. Joint the training and test sets and 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.

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.