Machine Learning Exercise 5

Multiple Instance Learning: image classification

Lu Liu (4621832)

May 31, 2017

1. The Naive MIL classifier

There are three functions implemented for the naive MIL classifier, which are `extractinstances`, `gendatmilsival`, and `combineinstlabels`. The function `extractinstances` is to segment an image using `im_meanshift`, to compute the average RGB value of every segment, and return it as features. Here the both `width`-parameters for apple and banana are set as 40 for `im_meanshift`. The function `gendatmilsival` is to generate a MIL dataset containing all apple and banana images. There are 120 bags representing 120 images of apple and banana. For every instances in bags, there are three features, which is the average RGB value. The number of instances per bag varies in different bags with range between 3 and 7. The scatterplot of the instances from the two classes is show in Figure 1. The scatterplot is plot using `scatterd(set,'gridded')` in Prtools.

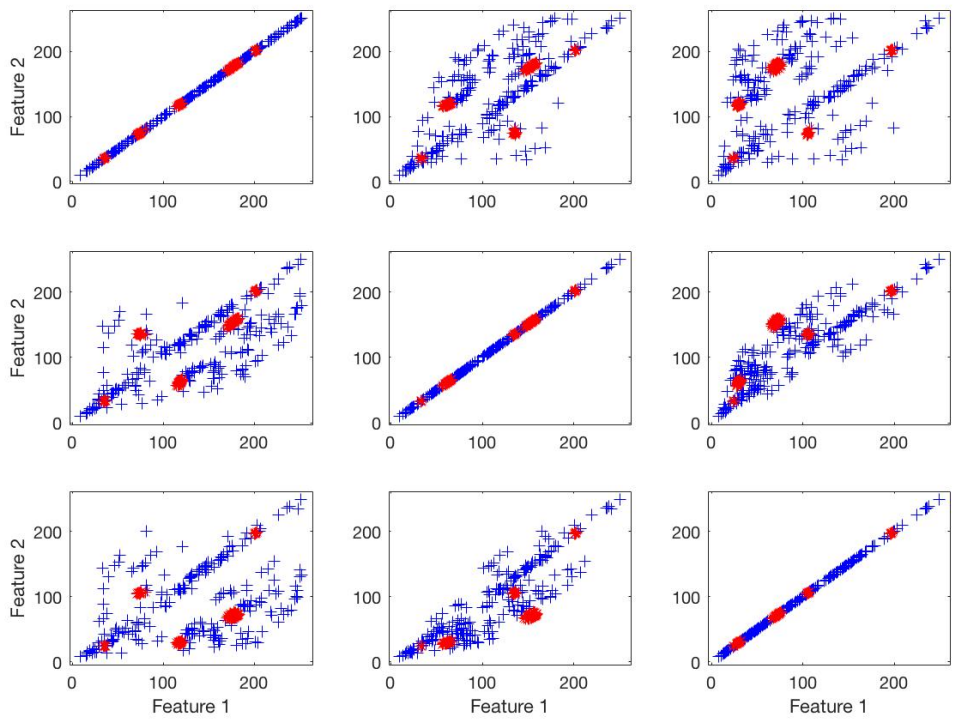

Figure 1: Scatterplot of the instances

In Figure 1, every pair of features in each instance are plotted as a sub scatterplot. Blue crosses represent the instances of apple while red crosses represent that of banana. It can be seen from the Figure 1 that the the instances of banana always aggregate at some small area.

After training the classifier using a Fisher classifier and applying the trained classifier to each instance in a bag, there are 47 apple images that are misclassified to be banana, while all the banana images are classified correctly. Thus the error rate of this classification is 0.3852. Since the training and testing are done with the same dataset, the error rate is not trustworthy.

In this classifier, the average RGB value is used as features in instances. To improve the performance of this classifier, different values could be used. For example, the gradient of the RGB value, the gray scale value of the image, covariance and so on. Different features can be tried to find the most proper features for the classifier. Another way to improve the accuracy is to adjust the `width`-parameter in `extractinstances` to obtain more accurate instances in each bag.

2. MILES

In this classifier, 30 images are chosen randomly from apple images and banana images, totally 60 images as training data, while the other images are testing data. Since after extract of instances, there are 539 instances in the dataset. The feature vector obtained by `bagembed` $m(B_i)$ is with length of 539. For training data, the size of $m(B_i)_{training}$ is 60×539, while for testing data, the size of $m(B_i)_{testing}$ is 60×539.

After training on the prdataset with vector $m(B_i)$ the classifier `liknonc`, the testing dataset is used to estimate the error rate. In this classifier, 2 apple images are misclassified as banana images, while 1 banana images is misclassified as apple image. Thus the error rate is 0.05, which is much less than the error rate of the Naive MIL classifier.

To improve the performance of MILES, different features of instances can tried such as minimum and maximum of RGB or increase the number of features in each instance.

3. Another MIL classifier

The implemented code of this classifier is in Appendix.

There are three parts in the implementation of this classifier.

- First, read all 120 images of apple and banana and store them in two cells separately.
- Second, extract the instances of all the apple and banana objects using function `extractinstancesminmax` with `width`-parameter as 40. Here, the features of instance are replaced by minimum and maximum of RGB value. Thus there are 6 feature values for each instance. With all the extracted instances, apple bags and banana bags are formed with labels. To get a trustworthy error rate, all the data is divided into two small datasets randomly with 60 bags in each dataset (30 apple bags and 30 banana bags) for training and testing.
- Third, calculate bag-distance matrix for all the bags in training data and testing data with distance between bags is defined as

$$D(B_i, B_j) = \min_{k,l} ||x_{ik} - x_{jl}||^2$$

  Here two functions implemented by myself are used, which are `bagdistance` and `bagdissimi`. `bagdistance` is to calculate the distance between two bags, while

`bagdissimi` obtains the distance of each pair bags in given set of bags. And then make Prtools datasets for training data and testing data with matrix D and corresponding labels. Train on this dataset a L_1-support vector classifier: `liknonc`.

With the new trained classifier, testing is done with the testing dataset of 60 bags. The classifier performs well that all the images are classified correctly. Compare with Naive MIL classifier and MILES, this classifier performs the best. The error rate of every classifier is shown in Figure 2.

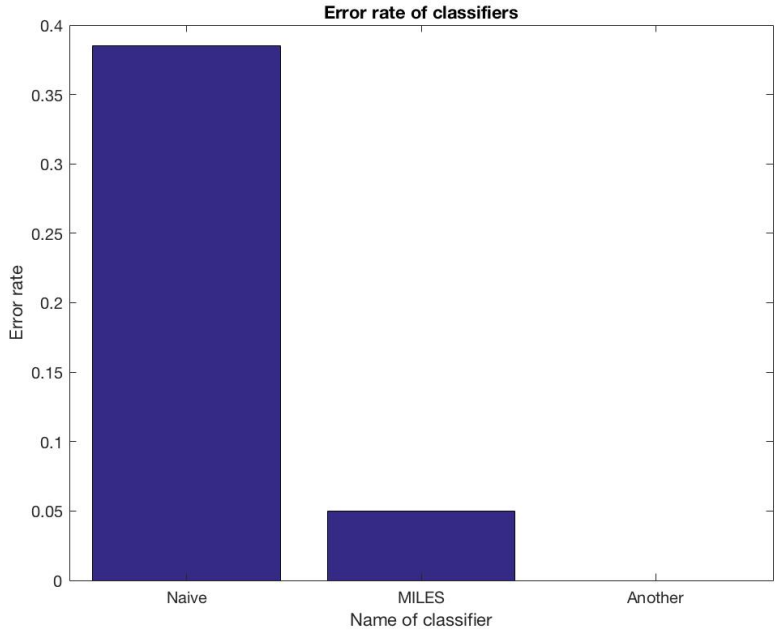

Figure 2: Error rate of classifiers

A. Another MIL classifer

```
%% Another MIL using bag representation and dis-similarities
%% read all the apple and banana images
imDir1 = 'sival_apple_banana/apple';
imDir2 = 'banana';
cd('..');
cd(imDir1);
apple = dir;
apple_object = {};
banana_object = {};
% read apple images
for i = 1:length(apple)
    if apple(i).isdir == 0
        apple_object = [apple_object, imread(apple(i).name)];
    end
end

cd('..');
cd(imDir2);
banana = dir;
% read banana images
for j = 1: length(banana)
```

```
    if banana(i).isdir == 0
        banana_object = [banana_object, imread(banana(i).name)];
    end
end
banana_object(62) = [];
banana_object(61) = [];
cd('..');

%% Bag representation
% extract instances using extractinstancesminmax
width = 40;
[apple_label, apple_bags] = extractinstancesminmax(apple_object,width);
[banana_label, banana_bags] = extractinstancesminmax(banana_object,width);
bags = [apple_bags, banana_bags];

% seperate the whole dataset as training data and testing data
apple_rand = randperm(60);
banana_rand = randperm(60)+ 60*ones(1,60);

train_data = {};
test_data = {};
% 60 training data
for i = 1:30
    train_data{i} = bags{1, apple_rand(i)};
    train_data{i+30} = bags{1, banana_rand(i)};
end
% 60 testing data
for i = 1:30
    test_data{i} = bags{1, apple_rand(i+30)};
    test_data{i+30} = bags{1, banana_rand(i+30)};
end
%% Main implementation of bag dis-similarities
% calculate bag dissimilarities
bagdis_train = bagdissimi(train_data);
bagdis_test = bagdissimi(test_data);

% make prdataset
bagdis_train_dataset = prdataset(bagdis_train, [ones(30,1); 2*ones(30,1)]);
bagdis_test_dataset = prdataset(bagdis_test, [ones(30,1); 2*ones(30,1)]);

% train Liknon classifier
C = liknonc(bagdis_train_dataset, 30);

%% test and estimate error rate
% test
labels = labeld(bagdis_test_dataset, C);

% error rate
error_a = 0;
error_b = 0;

for i = 1:30
    if labels(i) == 2
        error_a = error_a + 1; % apple misclassification
    end

    if labels(i+30) == 1
        error_b = error_b + 1; % banana misclassification
    end
end

error_rate = (error_a+error_b)/60
accuracy = 1-error_rate
```

B. extractinstancesminmax

```
function [label, bag_minmax] = extractinstancesminmax(images,width)
% images            a cell of multi-images
% width             width-parameter for im_meanshift
% label             a cell for label of each image
% bag_minmax        a cell with N*num_ins*6 matrix as extracted instances

N = length(images); % number of images
label = {};

for i = 1:N
   label = [label, im_meanshift(images{1,i},width)];
end

%% min and max of RGB

bag_minmax = {};

for i = 1:N
    image = images{1,i};
    num_ins = max(max(label{i})); % number of instance
    instance = zeros(num_ins,6); % matrix for instance for each image

    for n = 1:num_ins
        ins_rmin = min(min(image(:,:,1).*uint8(label{i}==n)));
        ins_rmax = max(max(image(:,:,1).*uint8(label{i}==n)));
        ins_gmin = min(min(image(:,:,2).*uint8(label{i}==n)));
        ins_gmax = max(max(image(:,:,2).*uint8(label{i}==n)));
        ins_bmin = min(min(image(:,:,3).*uint8(label{i}==n)));
        ins_bmax = max(max(image(:,:,3).*uint8(label{i}==n)));

        instance(n,:) = [ins_rmin, ins_rmax, ins_gmin, ins_gmax, ...
            ins_bmin, ins_bmax];
    end

    bag_minmax = [bag_minmax, instance];
end
end
```

C. bagdistance

```
function [distance] = bagdistance(bag1, bag2)
% bag1, bag2   two image bag with multi-instance
% distance     the distance between two bags

[num_ins1, ¬] = size(bag1);
[num_ins2, ¬] = size(bag2);

distance_multi = zeros(num_ins1, num_ins2);

% calculate distance between each pair of instances
for i = 1:num_ins1
    for j = 1:num_ins2
        distance_multi(i,j) = norm(bag1(i,:)-bag2(j,:));
    end
end
% min distance
```

```
17 distance = min(min(distance_multi));
18 end
```

D. bagdissimi

```
1  function [bagdissimi] = bagdissimi(bags)
2  % bags                  a cell with multi-bags
3  % bagdissimi            bag dis-similarities
4
5  N = length(bags);
6
7  bagdissimi = zeros(N, N);
8
9  for i = 1:N
10     for j = 1:N
11         bagdissimi(i,j) = bagdistance(bags{i}, bags{j});
12     end
13 end
14
15 end
```