IN4320 : Machine Learning
Assignment 6 - Multi-Instance Learning

Navin Raj Prabhu - 4764722
N.LaxminarayananRajPrabhu@student.tudelft.nl

June 6, 2019

1 Naive MIL classifier

1.1 Function - extractinstances()

```
function [data_matrix] = extractinstances(image, width_param)

    img_seg = im_meanshift(image,width_param); % Segment Image
    seg_num = length(unique(img_seg));

    data_matrix = zeros(seg_num,3); %Return data - Init

    r_data = image(:,:,1);
    b_data = image(:,:,2);
    g_data = image(:,:,3);

    for i=1:seg_num
        target = (img_seg==i);

        % Average R per segment
        red_pix = r_data(target==1);
        red_avg  = sum(red_pix)/length(red_pix);

        % Average R per segment
        blue_pix = b_data(target==1);
        blue_avg  = sum(blue_pix)/length(blue_pix);

        % Average G per segment
        green_pix = g_data(target==1);
        green_avg  = sum(green_pix)/length(green_pix);

        % return features for image - data_matrix
        data_matrix(i,1) = red_avg;
        data_matrix(i,2) = blue_avg;
        data_matrix(i,3) = green_avg;
    end

end
```

1.2 Mean Shift - Window Size

A key problem in segmentation is that of splitting up into too few (*Under Segmentation*) or too many regions (*Over Segmentation*). For the problem at hand (Apple Banana dataset), I believe Over Segmentation is better. Because, as the images have random backgrounds, segmenting into very few regions might tend to miss the target regions itself (The Apples and Bananas in the image), Hence as Over Segmentation splits image to many regions and both the classes have similar backgrounds, I expect the classifier and the MIL algorithm to detect the differences in the classes and have a better accuracy over usage of Under Segmentation. A window size parameter of '*30*' was used for the mean shift algorithm.

1.3 Function - gendatmilsival()

```
function mil_dataset = gendatmilsival()

    data1_apple_path  = dir(fullfile('dataset','sival_apple_banana','apple','*.jpg'));
    data1_banana_path = dir(fullfile('dataset','sival_apple_banana','banana','*.jpg'));
    dataset_pointers = [data1_apple_path, data1_banana_path];
    apple_raw   = cell(length(data1_apple_path),1);
    banana_raw  = cell(length(data1_banana_path),1);

    width_param = 30; % Width Parameter for Mean Shift

    for i=1:size(dataset_pointers,2)
        dataset = dataset_pointers(:,i);
        for j=1:length(dataset)
            % Read Data Images from dataset
            img     = imread(fullfile(dataset(j).folder, dataset(j).name));
            % Extract Instances - Using Mean shift and color average
            img_seg = extractinstances(img, width_param);
            if i== 1
                apple_raw{j,1} = img_seg;
            elseif i == 2
                banana_raw{j,1} = img_seg;
            end
        end
    end

    apple_label  = ones(length(data1_apple_path),1);
    banana_label = (-1)*ones(length(data1_banana_path),1);
    bags_appl_bana   =   [apple_raw; banana_raw];
    baglab_appl_bana =   [apple_label; banana_label];

    % MIL - Dataset Creation
    mil_dataset = bags2dataset(bags_appl_bana, baglab_appl_bana);

end
```

1.4 MIL Dataset

- **Number of Bags**: 120 bags consisting of 60 apples bag and 60 bananas
- **Number of Features per instance**: 3 features (mean red, mean green, mean blue).

- **Number of instances per bag**: 2 to 8 instances per image (bag). Each image (bag) can produce different number of instances (2 to 8) from the mean shift algorithm.

The MIL dataset created is plotted in Figure - 1 for the 3 features. The two classes Apple and Banana and not completely separable. But at the same time we do see a dense yellow cluster and a red cluster amongst a highly overlapping feature space.

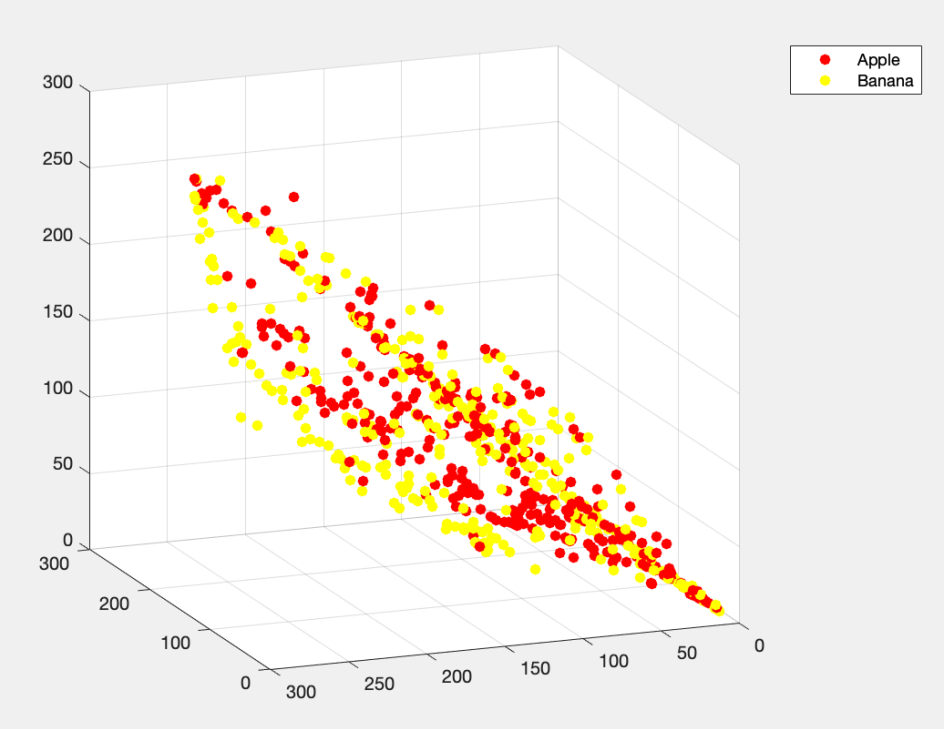

Figure 1: MIL Dataset - 3D Scatter plot for 3 features (Avg-RGB)

1.5 Function - combineinstlabels()

```
function max_label = combineinstlabels(labels)

    % Majority voting for label return
    possible_label = unique(labels);
    majority_vote = -10000;
    for i = 1:length(possible_label)
        label_len = sum(labels==possible_label(i));
        if label_len > majority_vote
            max_label = possible_label(i);
            majority_vote = label_len;
        end
    end
end
```

1.6 Classification Results

```
% Fit Fisher Classifier on instances dataset
fishclf = fisherc(app_ban_dataset);
instance_labels = labeld(app_ban_dataset, fishclf);
```

```
bag_start = 1;
bag_prediction = [];
for i=1:length(bags)
    %Combine for Each Bag - (Collection of instances)
    bag_size     = size(bags(i),1);
    bag_set      = instance_labels(bag_start:bag_start+bag_size-1);
    bag_start    = bag_start + bag_size;

    % Finally - Label for bag obtained
    bag_label = combineinstlabels(bag_set);

    bag_prediction = [bag_prediction; bag_label];
end

% Total error for dataset
error = sum((bag_prediction ~= bags_true_label));
% Mis-Classified Apple
apple_error = sum((bag_prediction(1:60) ~= bags_true_label(1:60)));
% Mis-Classified Banana
banana_error = sum((bag_prediction(61:120) ~= bags_true_label(61:120)));
```

- **Total misclassified bags**: 46 (38% error)
- **Number of Apple images misclassified**: 19 (32% apple error)
- **Number of Banana images misclassified**: 27 (45% banana error)
- **Error Estimate not trustworthy**: The Dataset in hand, is such that both the classes have similar backgrounds and only a small portion of the image is the target object. Because of this, if a bag has more non-target object defining instances (like background) a misclassification is inevitable. In simple terms, the background instances can disrupt the result.

1.7 Improvements in Naive MIL

1. Better Features - Each instance in the bag contains only 3 features (Mean R,G,B). For a better classification, I believe better features like even as simple as pixel data (but makes the problem more complex) will perform better as it can overcome the adverse effects of average R,G,B.
2. Alternate Segmentation - The mean shift algorithm has a drawback, in the sense that while Under Sampled can miss the target object completely, and while Over Segmented can results in to many background segments which are available in both classes (which might result in misclassification). To solve this a target based or color similarity based image segmentation can solve this issue, One such example of this type of segmentation method can be seen in the work of T.Gevers [1].

2 MILES classifier

2.1 Function - bagembed()

```
function m_B = bagembed(bags, instance)
    sigma = 70; % Tuned Sigma value
    distance_f = 0;
    m_i = [];
    m_B = cell(length(bags),1);
    for i=1:length(bags)
```

```
        bag_ins = bags{i};
        for j=1:size(instance,1)
            for z=1:size(bag_ins,1)
                % Measure of Similarity
                distance_z = exp(-sum((instance(j,:)-bag_ins(z,:)).^2)/sigma^2);
                if(distance_z>distance_f) % Track best similar
                    distance_f = distance_z;
                end
            end
            m_i = [m_i distance_f];
            distance_f = 0;
        end
        m_B{i,1} = m_i;
        m_i = [];
    end
end
```

- **Size of Feature vector** $m(B_i)$: 660 features per image (660 is the total number of instances present in all the bags combined)
- **Sigma (σ) value chose**: 70

2.2 LIKNON Classifier train/test on MILES dataset

```
% MILES - Bags Embed
miles_feats   = bagembed(bags,instances);
miles_label   = [ones(60,1); (-1)*ones(60,1)];
dataset_miles = bags2dataset(miles_feats, miles_label);

acc_array = [];
for i = 1:100 % Average Error of 10 runs

    [train_set, test_set, train_ind, test_ind] = gendat(dataset_miles,0.7);

    svm_1_classif   = liknonc(train_set);
    predict         = labeld(test_set, svm_1_classif);

    acc_i = sum(predict ~= test_set.nlab)/length(predict);
    acc_array = [acc_array acc_i];

end
disp("MEAN ACCURACY - " + mean(acc_array))
```

2.3 Classification Results - LIKNON & MILES

The dataset was split in to train and test dataset with a 70% split. This experiment was repeated 100 times to draw the following conclusions. The reported numbers are with respect to the test dataset of 36 (18 - Class Apple and 18 - Class Banana).

- **Number of Errors**: $\approx$2 out of 36 test objects (6% - On Average of 100 runs)
- **Naive MIL & MILES comparison**: The performance of MILES (6%) is clearly better than the performance of Naive MIL (38%). It is also important to note that the FP and FN are imbalanced in Naive MIL (19, 27 respectively) has been balanced and reduced in MILES (1, 1 respectively).

- **Possible improvements on MILES**: The size of M(Bi) derived from the bag embedding of MILES is determined by the total number of instances in all the bags. Therefore, the LP requires the storage of a data matrix of size $(l^+ + l^-) * n$,where $(l^+ + l^-)$ is the number of training bags, and n is the number of instances in all the training bags. For some applications, n can be several orders of magnitude greater than $(l^+ + l^-)$ Therefore, the storage requirement can be prohibitive. The needed storage space could be significantly reduced by an instance similarity measure that generates sparse features [3].

3 Another MIL classifier

3.1 Citation KNN - MIL Classifier

For this section, to implement a new MIL algorithm form literature, I selected the Citation KNN MIL Algorithm - a widely used MIL technique. The algorithm implemented is KNN based Citation approach, an MIL technique inspired from the research work by Jun Wang and Jean-Daniel Zucker [2].

Distance Metric: The k-nearest neighbors algorithm (k-NN)is a type of instance-based learning, or lazy learning, which is a non-parametric method used for classification. This method of classification can be extended into a MIL setup by using distance measures such as the Hausdorff distance - it is the greatest of all the distances from a point in one set to the closest point in the other set, in our case from one bag to another.

References and Citers: Once the distance between bags are found, it can be used on a lazy classification algorithm like the KNN. The paper [2] explains one such method as the Citation method - based on *references* and *citers. How do we find references (R_n, R_p) and citers (C_n, C_p)*: It is easy to define the R-nearest references of an example bag - b as the R-nearest neighbors of b. To find the C-nearest citers of b, we use the logic of - Citers(b, C)=$\{b_i \| \text{ Hausdorff}(b_i, b) \leqslant \text{C}$, where $b_i \in$ BS=set of bags$\}$. Now as the references (R_n, R_p) and citers (C_n, C_p) are found, this can be used to predict labels of the bags. With this, the Citation KNN has two parameters to tune, k and c which explains the number of references/neighbors and citers respectively. For this report, the values of c an k were 3 and 40 respectively.

Classification in Citation KNN: Let p=$R_p + C_p$, and n=$R_n + C_n$. Citation-KNN the KNN algorithm in which p and n are computed by using the Hausdorff distance and classification is - if p>n, then the class of the bag b is predicted as positive, otherwise negative.

The code implemented to test the Citation KNN on our Apple-Banana dataset is as follows.

Train Citation KNN:

```
function classifier = train_citation_KNN(bags, bags_labels, train_bag, train_lab,
                                         test_bag, test_index, c_param, k_param)

    % Get Citers and Neighors for Dataset
    citations = knn_citations(bags, bags_labels, test_index, c_param);
    neighbors = knn_references(train_bag, train_lab, test_bag, k_param);

    % Assign Neighbors and Citers to Classifer Obj
    classifier.citations    = citations;
    classifier.neighbors    = neighbors;

end
```

Test Citation KNN:

```
function test_label = test_citation_KNN(obj)
    if obj.citations ~= 0
```

```
            for j = 1:length(obj.neighbors)
                test_label(j) = combineinstlabels([obj.neighbors{j};obj.citations]);
            end
        else
            for j = 1:length(obj.neighbors)
                test_label(j) = combineinstlabels([obj.neighbors{j}]);
            end
        end
end
```

Get Neighbors and Citations:

```
function references = knn_references(train_data, train_label, test_data, k)

    for i = 1:length(test_data)
        bag_test = test_data{i};
        distance = [];
        for j = 1:length(train_data)
            bag_train = train_data{j};
            distance(1,j) = get_bag_distance(bag_train, bag_test);
        end
        [k_min_values, k_ind] = mink(distance,k);
        references{i} = train_label(k_ind);
    end
```

```
function citers = knn_citations(dataset, dataset_lbl, test_index, k)

    for i = 1:length(dataset)
        distance = [];
        for j = 1:length(dataset)
            distance(1,j) = get_bag_distance(dataset{i}, dataset{j});
        end
        [values, indexes] = mink(distance,k);
        neighbor_ind(i,:) = indexes;
    end

    counter = 0;
    for j = 1:length(dataset)
        if ismember(test_index,neighbor_ind(j,:))
            counter = counter + 1;
            citers(counter,1) = dataset_lbl(j);
        end
    end
```

3.2 Comparison - (Naive MIL, MILES, Citation KNN MIL)

The above Citation KNN was testion on a 0.7 train, test data split, which gives 36 test images and 84 train images. This experiment was repeated 100 times and the error rates are as below,

- **Number of Errors**: 8 out of 36 test objects (21% - On Average of 100 runs
- Balanced FP and FN - ≈ 4, ≈ 4 respectively.

The Citation KNN MIL (21% error) performs better than the Naives MIL (38% error) but performs worse than the MILES (6% error) on the given Apple-Banana dataset. The Citation KNN used the features (Average RGB) which from Figure - 1 can be seen that the classes are not separable completely, this could be a reason Citation KNN (bag distance dependent) cannot perform better than MILES.

References

[1] T. Gevers. Image segmentation and similarity of color-texture objects. *IEEE Transactions on Multimedia*, 4(4):509–516, Dec 2002.

[2] Jun Wang and Jean-Daniel Zucker. Solving multiple-instance problem: A lazy learning approach, 2000.

[3] Yixin Chen, Jinbo Bi, and J. Z. Wang. Miles: Multiple-instance learning via embedded instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):1931–1947, Dec 2006.