IN4320: Machine Learning Assignment 6

Zheng Liu (4798406)

June 7, 2019

1 The Naive MIL classifier

1.1 (a)

I implemented the image reading function by applying `dir,imread` function and stored the result of each objects class, apple, banana, medal and box, in 60×1 `cell` respectively.

1.2 (b)

By reading Mean-Shift material [4], I generally understand that the idea of width is the size of region of interests (or known as Window), so I manually and approximately evaluated the size of apple and banana of the images, then I did experiment around my manually guessing value of width, taking 10 as the step size and increasing width size from 20 to 100, and determine a range to decide which width value to use. 4 examples shown from Figure 1, 2, 3 and 4 (2 from banana and 2 from apple) illustrate the influences of width value, I set **width=30** for both banana and apple.

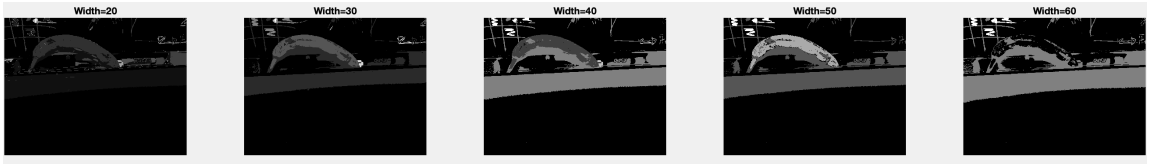


Figure 1: Mean-Shift result by applying `imshow()` with different width value from 20 to 60, sival_apple_banana/banana/fluor5_Banana_094.jpg

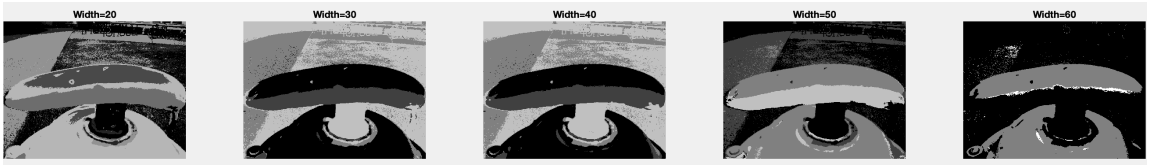


Figure 2: Mean-Shift result by applying `imshow()` with different width value from 20 to 60, sival_apple_banana/banana/SunD3_Banana_074.jpg

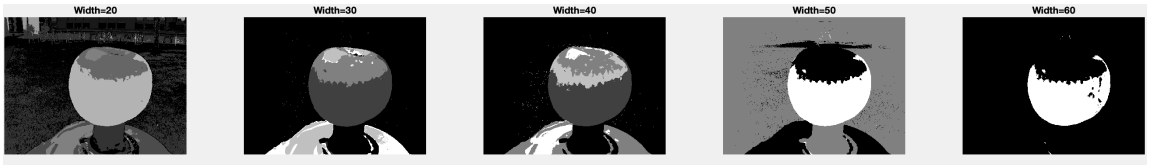


Figure 3: Mean-Shift result by applying `imshow()` with different width value from 20 to 60, sival_apple_banana/apple/SunD3_Banana_080.jpg

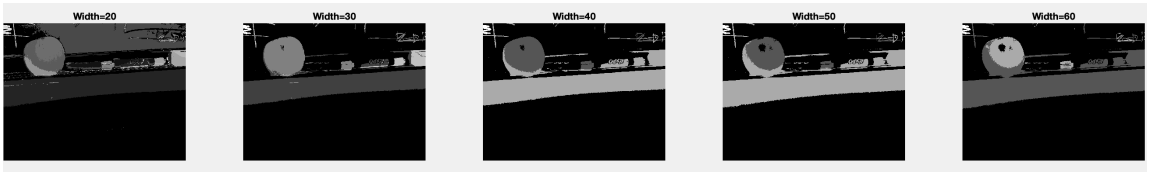


Figure 4: Mean-Shift result by applying `imshow()` with different width value from 20 to 60, sival_apple_banana/apple/flour5_Apple_100.jpg

It would be good to segment based on the approximately "good" width value, thus neither over-segment nor under-segment is suitable, because no matter the width value becomes large or small, the vector of object could not be represented properly, meaning too much foreground would be excluded (over-segment) or too much background would be included (under-segment). When I experiment width with a small value, the computation is rather slow comparing with when I did it with large value.

After the segment, I obtained the feature vectors of each image from 3 channels. Here (image processing/classification), generally, each image is a bag and each cluster (feature vector) is regarded as an instance in certain bag [1, 2].

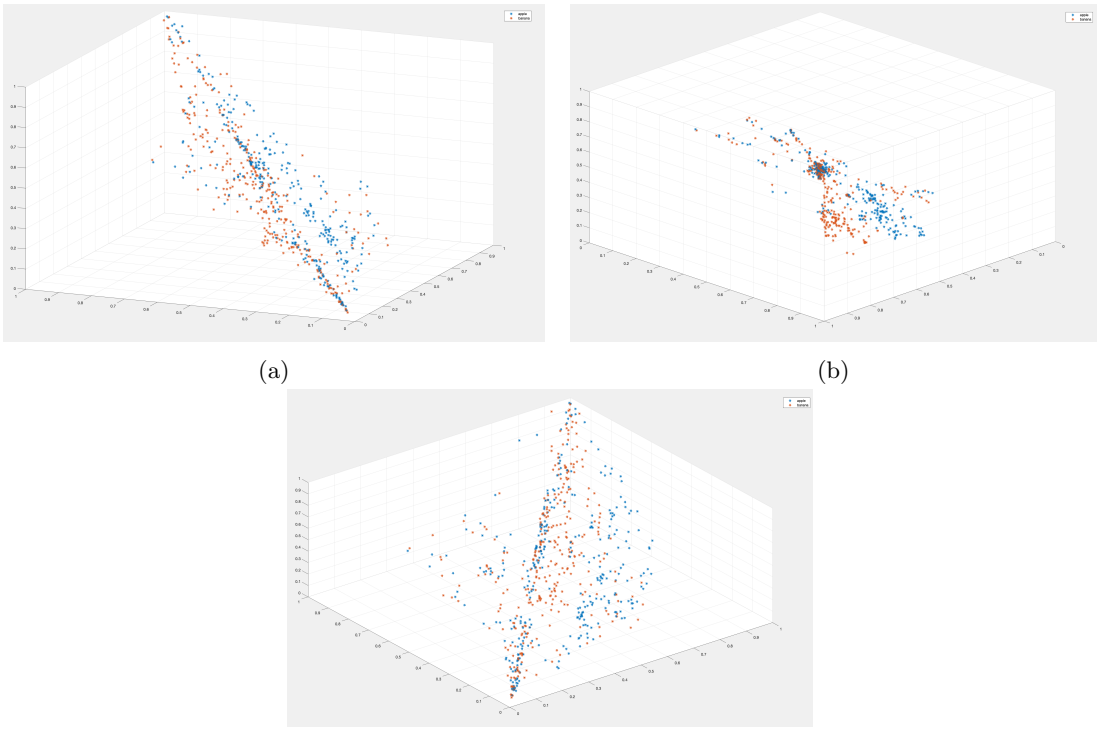
1.3 (c)

As I described in detail in part 1(b), I reached **120 bags** as there are 60 apple images and 60 banana images in 'sival_apple_banana' folder. Each instance has **3 features from red, green and blue channel** respectively. Different bags (images) have different instances (feature vectors/matrix) size, vary from **3-9** (with setting width=30 for apple and banana), which was decided by the width value of Mean-Shift algorithm. I claim that because different images would be segmented into different number of clusters, each cluster has its own feature vector. Totally I obtained 659 instances in these 120 bags, the dataset is 659×3 `prdataset`.

The 3D-scatterplot of the instances from the two classes is shown in Figure 5, the blue ones are from apple images and the red ones are from banana images. From Figure 5 we can see that in somewhere two classes show clear clusters; many of the scatters aggregate in the diagonal, I claim because they are the vectors of background segments from different images.

1.4 (d)

I created a function `combineinstlabels` to accept a list of labels classified by trained classifier and return a single label obtained by majority voting rule for training period. It is easy to do as there



(a) (b)

(c)

Figure 5: 3D Scatterplot of the instances from the apple and banana class, examples from 3 angles

are only labels of 0 and 1 and the task is to find which label has the largest number, so I applied a single sorting algorithm to search the label with largest number.

1.5 (e)

I implemented the Fisher classifier. In the beginning I set size of training set is instances from random 90 bags (45 from apple set and 45 from banana set) and size of testing data are from 120 bags (all data are used). I reached 1.18 misclassified apple images, error rate 15.7%, recognised as banana, 2.08 wrongly recognised banana images (error rate 27.7%), overall error rate is 21.7%.

The reason that the error estimate is not trustworthy is because in the exercise it requires "apply the trained classifier to each instance in a bag" which means there must be some instances from some bags are used for both training and testing, that is why not trustworthy. The method to solve this is to exactly separate the dataset, meaning training set and testing set should not have intersection.

I reset instances from 90 randomly selected bags (45 from apple and 45 from banana) for training and the rest 30 bags' instances are for testing, repeating experiment 1000 times and taking average error rate. After modification, the error rates become 18.63% for apple and 34.68% for banana, overall error rate is 26.65%, which is higher than the untrustworthy one's.

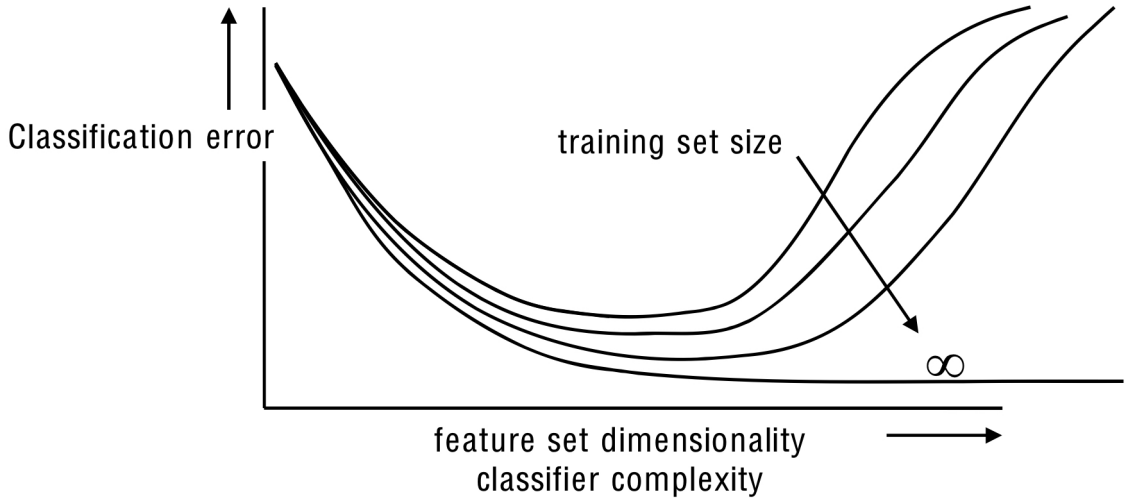


Figure 6: Performance improvement based on sample size and feature size (together with curse of dimensionality), from Pattern Recognition course slide

1.6 (f)

Based what I learned in Pattern Recognition course and in this MIL scene, I consider some possible improvement (sample size, feature size, feature, classifier) as follows:

- Sample size: In total we have (only) 120 images, 60 for each class which may not be sufficient for MIL. It would be good if we increase the sample size which may improve the performance of classification result. Shown in Figure 6.
- Feature size: In total we have only 3 features, which might not be enough, we may increase the size of features to improve the performance. However, if we increase the size of features we need to consider the curse of dimensionality. Shown in Figure 6.
- Features: Different from the previous one, feature size, I would like to talk about what feature we apply. RGB colour channels might not be good features at all. We may apply some other features like gray-value of images or SIFT and HOG (common in computer vision field) or edge detection (apple and banana have special edges differ from each other and the background) or we could extract original RGB features based on K-means algorithm rather than Mean-shift. Also, reading the lecture slide page 33 and 34, I found out that not only mean RGB could be the features from bags of instances, but also the maximum value and minimum value could be the feature.
- Classifier: Original Fisher classifier might be too simple for this scene, we could apply SVM with the kernel to improve the performance. We can also adjust the width value and control other variables to see which width is optimal.

2 MILES

2.1 (a)

I implemented the function `bagembed` by applying equation 1 [3].

$$\mathbf{m}(\mathbf{B}_i) = [s(x^1, \mathbf{B}_i), s(x^2, \mathbf{B}_i), \cdots, s(x^n, \mathbf{B}_i)]^T \tag{1}$$

With the result I obtained in section 1, there are 659 instances, thus the size of this feature vector $\mathbf{m}(\mathbf{B}_i)$ for our apple-banana problem will become 120×659. Differing form the previous situation where there are 659 3D (RGB) vectors, here there are 120 sample vectors, each vector's dimension is represented by the whole instances space, known as bag's similarity to the instances.

2.2 (b)

This part is to implement the MILES with L-1 support vector. The experiment setting is as follow: firstly I randomly separate the dataset into two part, 60×659 as training set and 60×659 as test set (with same number of apple and banana) without intersection, then applied `prdataset` function to generate the dataset respectively, next, I applied the bagembed I did before and obtained two **m(B)** dataset, one for training and one for testing. Then it was training time, applying given *liknonc* function with C value set as 10 (here I have no idea what does input C mean, I tried some value from 1 to 100 and did not see obvious differences) and obtain the classifier W, which is a 659×2 mapping.

For suitable σ value, I tested it in the training period by applying a loop, setting σ from 1 to 300 and repeating 50 times (randomness), the result (error rate) is stored in a 1 matrix, I found out that pixel value range is $[0, 255]$ when $\sigma \in (230, 280)$ (center is around 255), the error rate is significantly lower than with other values, satisfying that not all numbers become 0 or 1 in **m(B)**, meaning when σ takes the maximum of pixel value range the result would be optimal, I also tested this converge pixel value range as $[0, 1]$, the result is the same. The result is shown in Figure 7.

2.3 (c)

I have already "touched" a bit of testing in section 2.(b) when I talk about choosing σ value. With the suitable σ, the error is 21.17%, compare with the result form section 1 the Naive method, this classifier is better than the previous version.

Similarly as in section 1, the way to improve the performance could be similar. Firstly, it would be good to increase the size of dataset; secondly, as there are 659 dimension of the data, when I realised it I indeed worried a lot about the curse of dimensionality would ruin the performance, so I would like to say to use feature selection or feature extraction technique to reduce the feature size. Before feature reduction, we could consider to introduce some possibly better feature into the system.

3 Another MIL classifier

In the last part of section 1 and 2, I discussed about the possible way to improve the performance of classifier, then I decided to apply some of them together and introduce a novel method to this problem. The ideas are as follows:

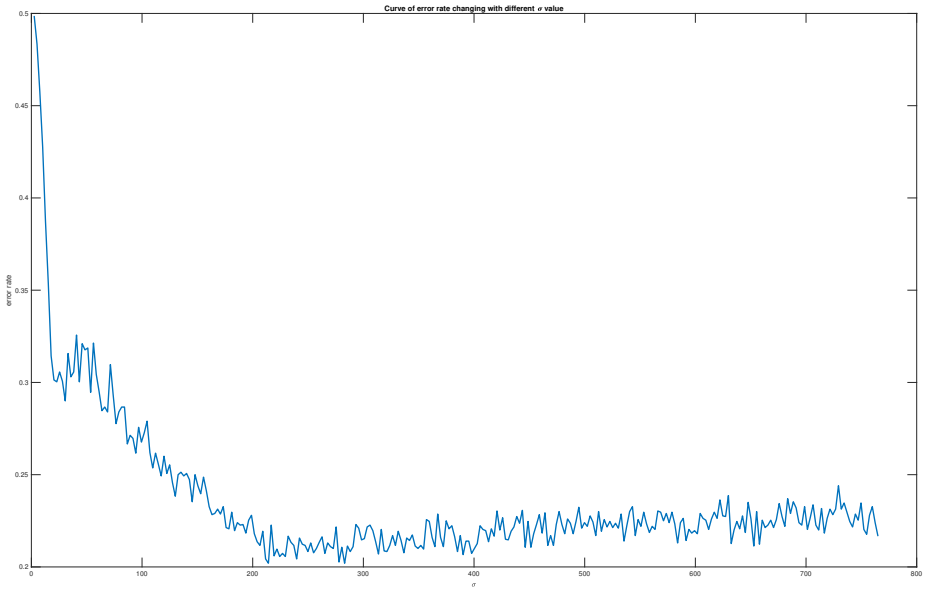


Figure 7: Curve of error rate with changing of σ value

- New features: Not only using RGB channels, having learned from lecture slide page 33, I noticed I could also apply maximum and minimum value of RGB channels. I added the maximum and minimum of RGB of each instance, those are 6 new features, the mean of RGB value is also introduced into the system.
- Feature reduction: Before training and testing, I applied PCA algorithm to do the feature extraction. `PRTool` was utilised to do so. After this step is my training and testing part.
- Bags (dis)similarity: From lecture slide, I found out that not only can I use MILES, but applying bags (dis)similarity is a possible method, shown in 2, then I calculated the (dis)similarity matrix and obtained 120×120 matrix, setting 50% bags for training and the rest for testing.

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

The new method is designed as **`new_method`** by applying all new things talked above with using L-1 support vector `liknonc`. the code is in the following part in the report. I achieved error rate 11.65%, which is much lower than both Naive MIL and MILES. The comparison is shown in Table 1.

```
1 function [error_final] = new_method()
2
3 prwarning off;
```

	Naive approach	MILES	My method
error rate	26.65%	21.17%	11.65%

Table 1: Error rate of 3 methods applied in this assignment

```
[apple, banana]=fileread();

% feature result initialisation
AppleFeat = {};
BananaFeat = {};
% mean-shift result initialisaiton
MS_apple = {};
MS_banana = {};

width = 30;

for i = 1:size(apple,1)
    i
    [result_apple, im_apple] = extract_modification(apple{i,1}, width);
    [result_banana, im_banana] = extract_modification(banana{i,1}, width);
    % features
    AppleFeat{i,1} = result_apple;
    BananaFeat{i,1} = result_banana;
    % mean-shit label
    MS_apple{i,1} = im_apple;
    MS_banana{i,1} = im_banana;
end
% combine apple and banana for dissimilarity
in = [AppleFeat;BananaFeat];
% calculate (dis)similarity
[dissimilarities] = dissimilarity(in);
%
% split dataset, taking 80% as training set and rest fo testing
percent = 0.5;
repeat = 1000;
error_rate_all = 0;
prwarning off;
for rep = 1:repeat
[train_in, test_in] = split(dissimilarities,percent);
% make prdataset
n_train = 60*percent;
n_test = uint8((1-percent)*60);
training_data = prdataset(train_in,[zeros(n_train,1);ones(n_train,1)]);
testing_data = prdataset(test_in,[zeros(n_test,1);ones(n_test,1)]);

% feature extraction and get extraction mapping %
Pca_Coeff = pcam(training_data, 0.90);
% extract training data %
training_data = training_data * Pca_Coeff;
% extract testing data
testing_data = testing_data * Pca_Coeff;

% train & test
[w, C] = liknonc(training_data,10);
label = labeld(testing_data, w);
error_a = 0;
```

```
error_b = 0;
for i = 1:n_test
    if label(i) == 1
        error_a = error_a + 1;
    end

    if label(i+n_test) == 0
        error_b = error_b + 1;
    end
end

error_rate = (error_a+error_b)/double(n_test*2);
error_rate_all = error_rate_all+error_rate;
end
error_final = error_rate_all/repeat;

end
```

References

[1] Multiple instance learning. `https://en.wikipedia.org/wiki/Multiple_instance_learning`. Accessed April 12, 2019.

[2] Oded Maron and Aparna Lakshmi Ratan. Multiple-instance learning for natural scene classification. In *ICML*, volume 98, pages 341–349, 1998.

[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.

[4] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, Aug 1995.