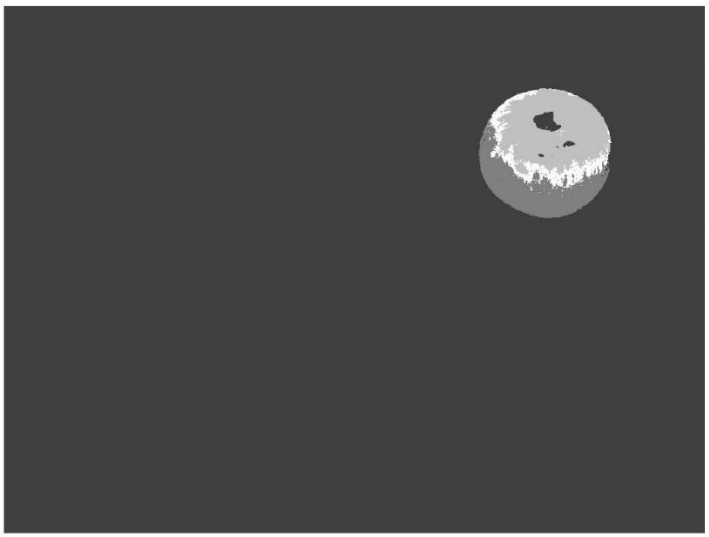
1. The Naive MIL classifier

a) …

b) …

c) Next implement a function extractinstances. What value of the width parameter did you find?

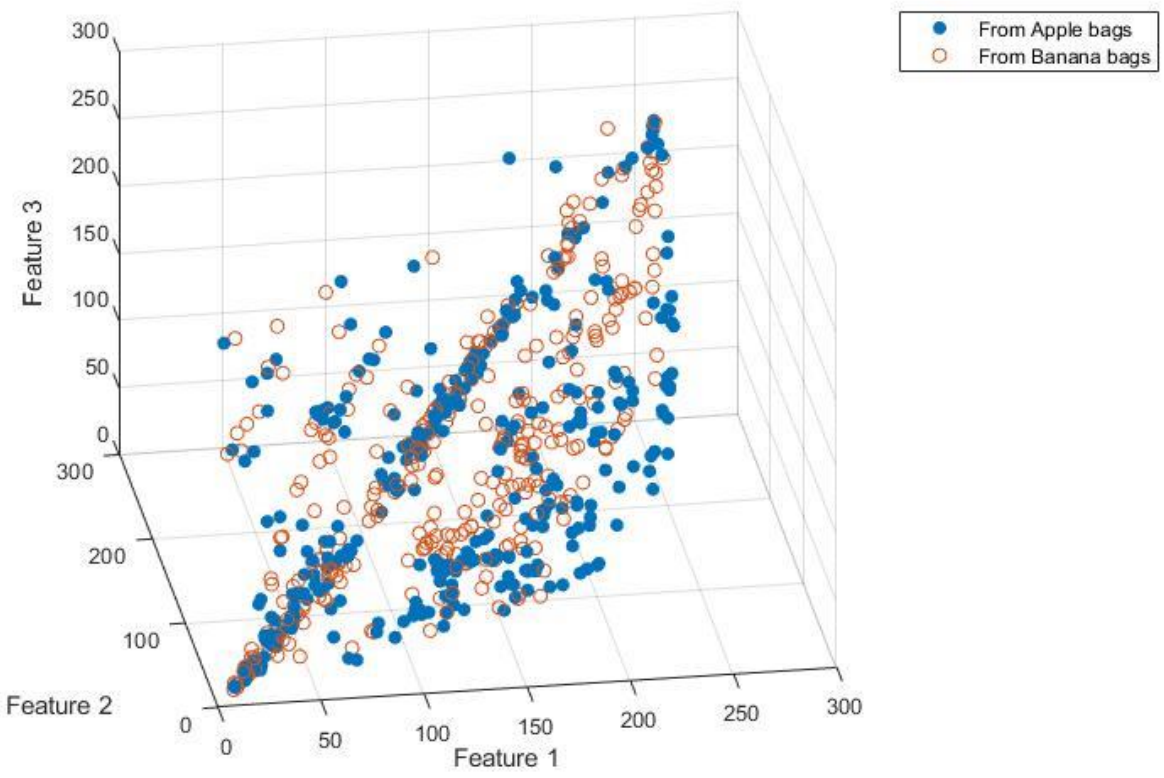
I determined the value of width parameter by trial-and-error. I found that when width=0.1, the background and foreground can be segmented well in the first few images, as shown in the following Figure where the apple and background can be easily segmented.



d) Create a function gendatmilsival that creates a MIL dataset. How many bags did you obtain? How many features do the instances have? How many instances are there per bag? Make a scatterplot to see if the instances from the two classes are a bit separable.

Number of bags: 120 (60 for apples, 60 for bananas)
Number of features: 3(average red, average blue, average green)
Number of instances: 2-10 (different bags have different number of instances)

The scatterplot can be found below. A large number of the instances from two classes are not quite separable as some instances have very similar features but belongs to different bags(usually backgrounds). Also, some instances are a bit separable since their colors are quite different, such as bananas and apples.



e) …

f) Now we are almost ready to classify images... How many apple images are misclassified to be banana? And vice versa? Why is this error estimate not trustworthy? Estimate the classification error in a trustworthy way!

10 apple images(17%) are misclassified to bananas
24 banana images(40%) are misclassified to apples

This error estimate is not trustworthy since there are irrelevant instances at each bag, such as the background. If the irrelevant instances are larger than the positive ones in the bag, then the misclassification would occur due to the highly overlapped instances.

Now we adjust the error estimate method of a bag by using the max combining of the log posterior probability of all instances within a bag:

p = sum(log(q),1); where q is the outputs (Posterior probability) for all instances, and p is q is the outputs (Posterior probability) for a bag

The label with higher probability will be taken as bag output. The results can be found as:

13 apple images(21.667%) are misclassified to bananas
17 banana images(28.3333%) are misclassified to apples

g) Invent at least two ways in which you may improve the performance of this classifier (think of how you obtained your MIL dataset).

Methods to improve the performance:

- Extract deeper features when constructing the dataset, such as hue, saturation, value features.
- Perform k-means clustering for a better segmentation, especially in the separation of instances in positive bags.

2. MILES

a) How large will this feature vector m(Bi) become for our apple-banana problem?

The size of this feature becomes to 120 x 662.

b) Make a Prtools dataset with the vectors m(Bi) and their corresponding labels yi . Choose a sensible value for σ such that not all numbers become 0 or 1. Train on this large dataset a L1-support vector classifier (or, more correctly called, LIKNON): liknonc

The scaling factor σ is chosen as 0.5 in this case.

c) Test the LIKNON classifier on this dataset: how many errors is this classifier making? Is this classifier better than the naive MIL classifier trained in the previous section? What can you do to make this MILES classifier perform better?

By using the gendat function, 70% of datasets are randomly selected as training set, while the remaining as test set. The experiment is repeated 1 0 times and the average error rate is taken as the final value.

Errors on the training set: 5%
Errors on the test set: 6.11%

In the MILES, a bag is embedded in a feature space defined by the instances in all the training bags where the coordinates of a bag represent its similarities to various instances. A feature selection technique using 1-norm SVM is applied to identify discriminative features (or instances) and construct bag classifiers at the same time, which improves the accuracy. This classifier generally performs better than Naïve MIL classifier in terms of the error rates. To improve the performance of MILES, other classifiers like a classical Support

Vector Machine might be more adapted. Moreover, performing dimensionality reduction might help get better performances since the extracted feature space remains at a high level of dimensions.

3. Other MIL classifier

a) Citation-KNN algorithms[1]

This method adapted KNN to solve the multiple-instance problem based on the minimal Hausdorff distance between bags and the notion of citation (Citation-KNN). The modified Hausdorff distance here provides a metric function between subsets of a metric space, which is given by taking the k-th ranked distance rather than the maximum, or the largest ranked one; The notion of citation is to take not only into account the neighbors of a bag b (according to the Hausdorff distance) but also the bags that count b as a neighbor. In this way, we could use either references or citers of an unseen example to predict the class of the example rather than only use the references. The distribution of positive and negative bags in R-nearest references and C-nearest citers of an unseen bag can be found in the following Table,

	positive bags	negative bags	
R-nearest references	R_p	R_n	R
C-nearest citers	C_p	C_n	C
	$p=R_p+C_p$	$n=R_n+C_n$	

if p>n, then the class of the bag b is predicted as positive, otherwise negative.

Following the algorithm mentioned in the paper, I implemented in our dataset as shown as the following code, including the construction of dataset, training of classifier and testing. The code was iterated 10 times and the mean average error is taken as the final result. The mean training error is 14.76%, while the mean testing error is 28.89%. Clearly the results are better than the Naïve MIL classifier, since the similarities are computed based on the features extracted from the whole image instead of one single instance. But this approach demonstrates inferior performance than the MILES in terms of the error rates.

Code:

```
%% Create training and test set from prtools
[train_data_bag,test_data_bag,train_idx_bag,test_idx_bag] =
gendat(dataset_bag,0.7);
list_train_idx = [];
list_test_idx = [];
```

```
da = train_data_bag.data;
for i=1:size(train_data_bag.data)
    list_train_idx = cat(1,list_train_idx, find(getident(dataset,'milbag') ==
da(i)));
end
train_data = dataset(list_train_idx,:);
da = test_data_bag.data;
for i=1:size(test_data_bag.data)
    list_test_idx = cat(1,list_test_idx, find(getident(dataset,'milbag') ==
da(i)));
end
test_data = dataset(list_test_idx,:);
%% Classifier training, these optimal parameters are determined by tuning.
k_neighbor = 2;
nb_ref = 11;
nb_cit = 11;
% Distance_bag
[dist_matrix , list_idx_bag] = prepare_matrix_dist(train_data,k_neighbor);
%% Testing
% Test error
bagid_test = getident(test_data,'milbag');
id_bags = unique(getident(test_data,'milbag'));
error = 0;
for i=1:size(id_bags,1)
    bag_test = test_data(find(bagid_test == id_bags(i)),:);
    output = nnclassifier(train_data, dist_matrix, list_idx_bag,
bag_test,k_neighbor,nb_ref,nb_cit);
    true_lab = getlabels(bag_test);
    if output ~= true_lab(1)
        error = error +1;
    end
end
error = error/size(id_bags,1);
% Train error
bagid_train = getident(train_data,'milbag');
id_bags = unique(getident(train_data,'milbag'));
error_train = 0;
for i=1:size(id_bags,1)
    bag_train = train_data(find(bagid_train == id_bags(i)),:);
    output = nnclassifier(train_data, dist_matrix, list_idx_bag,
bag_train,k_neighbor,nb_ref,nb_cit);
```

```
    true_lab = getlabels(bag_train);
    if output ~= true_lab(1)
        error_train = error_train +1;
    end
end
```

Reference

[1] Wang, Jun, and Jean-Daniel Zucker. "Solving multiple-instance problem: A lazy learning approach." (2000): 1119-1125.