Codebook representation: in feature generation, we start by applying a scale-invariant interest point detector to obtain a set of informative regions for each image, and then represent the extracted image region by a local descriptor. Next, we group visually similar features to create a codebook of prototypical local appearances.

For training data (e.g. boxing), there are 2090 features in total and each feature is described by a 162-dimensional vector. In the example of boxing training data, there are 41 sets of boxing action (image sequence), and each action is represented by a set of frames. The idea can be denoted by the annotation below.

$$[boxing\ 1, boxing\ 2, \dots, boxing41]$$
$$boxing: [frame1, frame2, \dots]$$

The training data stores the frame identity that the feature appears in the action (frame_num), the spatial location that the feature appears in the frame (spa_center, spa_offset, hei_wid_bb, location), and the temporal record of each action (st_end_frame, frame_center, st_end_offset, tem_offset). In addition, due to the fact that similar objects in different images might have different scale and multiple records of the same action have different temporal scale or image sequence, thus the spatial and temporal scale for each feature are stored as well.

2. Creating voting map for each of the codewords

2.2 voting map

For each codeword, it is possible that multiple features/patches match it, thus in voting map their corresponding patches' information are recorded. It is notable that these patches are given by a new name – occurrence in the paper [1].

3. Localizing actions in image sequences

3.1 I wrote a function to compute the Euclidean distance between the codewords and the descriptors, and just called it in ism_test_voing.m.

```
%-------------------------------------------------------------------
% Write a code to compute eucl_mat, the Eucleadian distance between th
% The output matrix eucl_mat should have dimensions Dsize x NoSTIPs wh
% and NoSTIPs is the number of STIPs in the test sequence
eucl_mat = compute_dist(patches',Dictionary');
%-------------------------------------------------------------------
```

ab4.m | ism.m | ism_test_voting.m | compute_dist.m | Votemap.m | +

```
function D = compute_dist(X,B)
%compute_dist Compute the Euclidean distance between the dictionary
%elements and the descriptors extracted in an image sequence
%   X: the patches/features in the image sequence (eg. 34-by-162)
%   B: the codebook (eg. 550-by-162)
%   D: the Euclidean distance matrix between codewords and features
%      (eg. 550-by-34)
nframe = size(X, 1);
nbase=size(B,1);
XX = sum(X.*X, 2);
BB = sum(B.*B, 2);
D  = repmat(XX, 1, nbase)-2*X*B'+repmat(BB', nframe, 1);
D  = D';
end
```

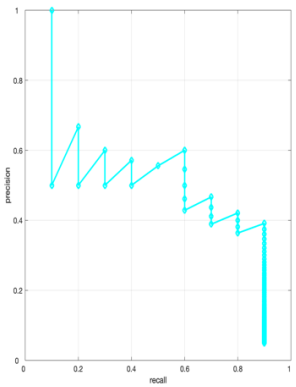
3.2 Write a function that implements the voting scheme.

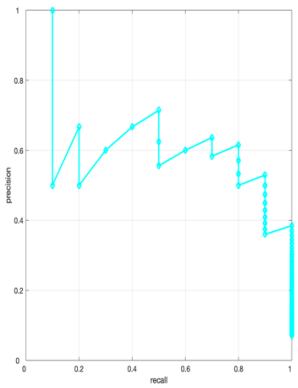
```
1. hough_array = [];
2.
```

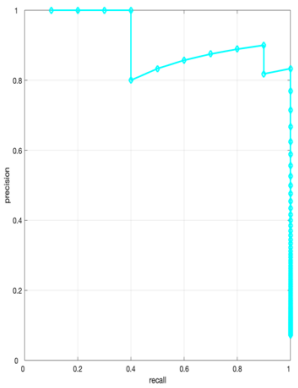
```
% cast the votes
for i = 1:size(patches,2)
    match_codeword_list = find(flag_mat(:,i)==1);
    for j = 1:size(match_codeword_list,1)

        match_codeword = match_codeword_list(j);
        for k = 1:struct_cb.offset(match_codeword).tot_cnt
            % set the vote location
            x = position(1,i) - struct_cb.offset(match_codeword).spa_offset(1,k) * spa_scale(i);
            y = position(2,i) - struct_cb.offset(match_codeword).spa_offset(2,k) * spa_scale(i);

            % set the start and end frame
            s = frame_num(i) - struct_cb.offset(match_codeword).st_end_offset(1,k) * tem_scale(
i);
            e = frame_num(i) - struct_cb.offset(match_codeword).st_end_offset(2,k) * tem_scale(
i);
            s = round(s);
            e = round(e);

            % set the vote weight
            match_weight = 1/size(match_codeword_list,1);
            occurrence_weight = 1/struct_cb.offset(match_codeword).tot_cnt;
            v = match_weight * occurrence_weight;

            % set the bounding box
            b1 = struct_cb.offset(match_codeword).hei_wid_bb(1,k)*spa_scale(i);
            b2 = struct_cb.offset(match_codeword).hei_wid_bb(2,k)*spa_scale(i);
            b1 = round(b1);
            b2 = round(b2);

            hough_one_vote = [x y s e v b1 b2]';
            hough_array = [hough_array hough_one_vote];

        end

    end
end
```

4. Evaluation

4.1 plot the recall precision curve.





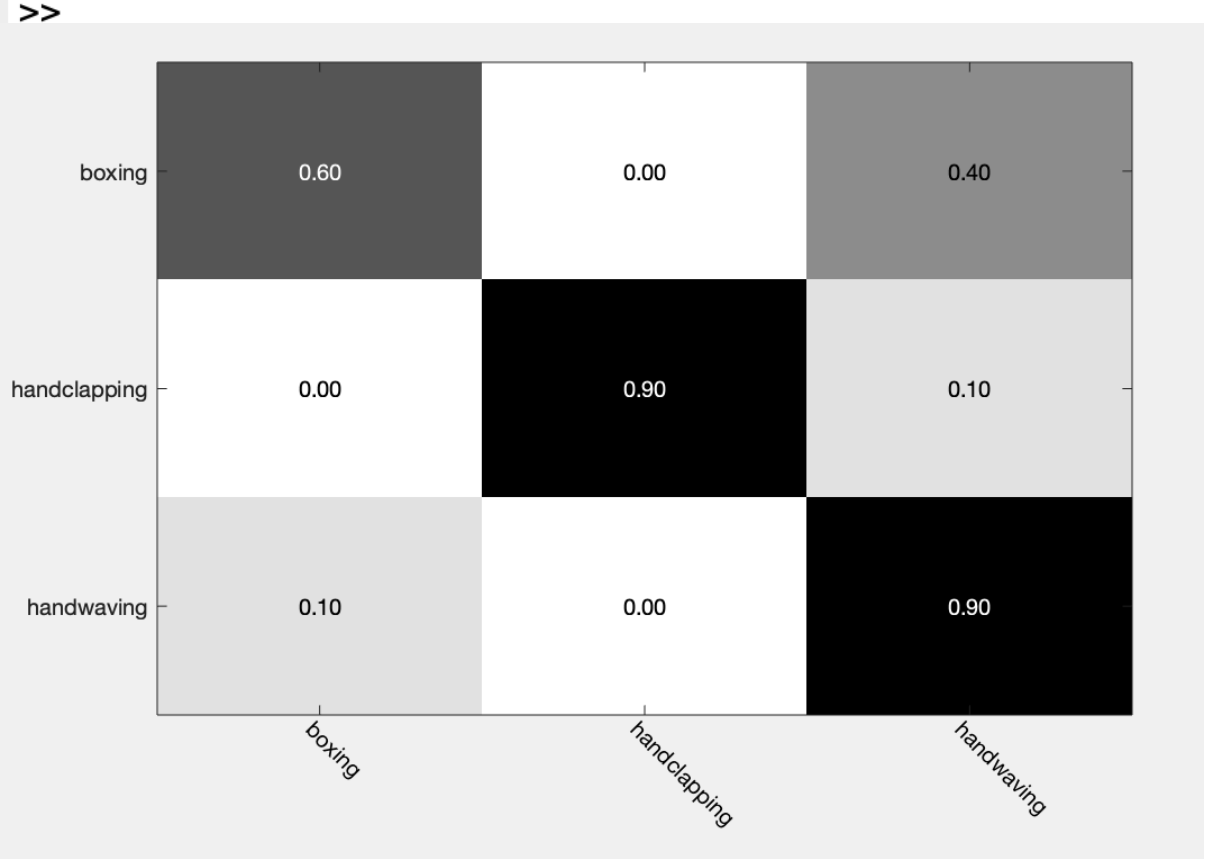


As we know, precision is calculated as the number of true positives divided by the total number of true positives and false positives, and recall is calculated as the number of true positives divided by the total number of true positives and false negatives (i.e. it is the true positive rate). From the precision-recall area under curve (AUC), the predicted results preform good. The AUC for class 3 (handwaving) is the largest, so the predicted results for class 3 is the best, by contrast, the predicted results for class 1 (boxing) is relatively bad.

4.2 Assign each sequence to a class according to which hypothesis received the higher number of votes. Report the misclassification error and build the confusion matrix.

```
function Evaluation()
%Evaluation predict these image sequences into a class,
%   calculate the error rate and draw the confusion matrix

load('struct_TP_FP');

% predict each image sequence into a class and the predicted
% results are stored into predict_label
predict_label = ones(3,10);
for i = 1:3
    cl_sz = size(struct_TP_FP.class(i).seq,2);
    for j = 1:cl_sz
        predict_label(i,j) = struct_TP_FP.class(i).seq(j).array(3,1);
    end
end

% calculate the error rate
num_c = 3; % the number of class
num_pc = 10; % the number of predicted sequences for each class
error_n = zeros(num_c,1); % the error number for each class
for k = 1:num_c
    error_n(k) = num_pc-length(find(predict_label(k,:)==k));
end
err = error_n/num_pc; % the misclassification rate for each class
err_all = sum(error_n)/(num_pc*num_c); % the misclassification for all the test dat
a
disp(['error rate for class', num2str(1),': ',num2str(err(1))]);
disp(['error rate for class', num2str(2),': ',num2str(err(2))]);
disp(['error rate for class', num2str(3),': ',num2str(err(3))]);
disp(['the total error rate: ', num2str(err_all)]);

% draw the confusion matrix
num_class = 3;
confusion_matrix = ones(num_class);
class_names={...
    'boxing'
    'handclapping'
    'handwaving'
    };
for ci = 1:num_class
    for cj = 1:num_class
        confusion_matrix(ci,cj)=length(find(predict_label((ci-
1)*1+1:ci,:)==cj))/num_pc;
    end
end

figure;
draw_cm(confusion_matrix,class_names,num_class);

end
```

```
error rate for class1: 0.4
error rate for class2: 0.1
error rate for class3: 0.1
the total error rate: 0.2
>>
```

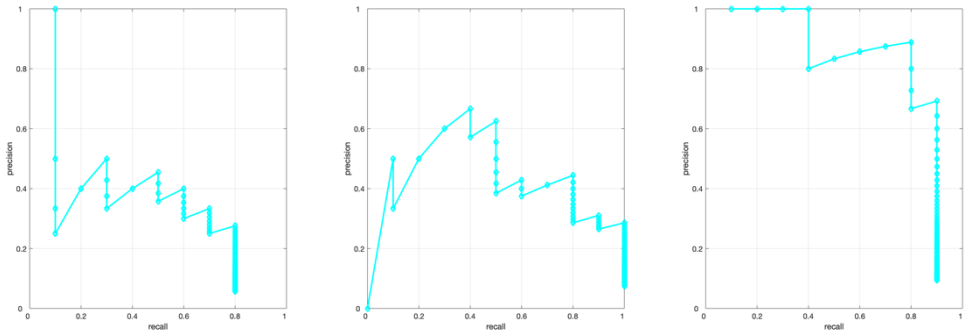


From the plots above, the total error rate for test data is just 20%. For handwaving and handclapping action predictions, the accuracies of both achieve 90%, whereas for boxing the accuracy is relatively low (60%). Meanwhile, we can find that 40% boxing action is misclassified into handwaving class, thus we can refer to that the two actions might have a lot of similar features. In addition, the results are different when I run the code multiple times, which probably is because the codebooks generated by K-means clustering are different every time.

5. Dictionary size

5.1 Perform the localization experiment using a very small dictionary and report the precision – recall curves.

I change all the sizes of the three codebooks as 20 in train_ism.m file.



It is apparent that the second plot is abnormal. Besides, from the first and second plot, the area under the curve is intuitively small, which means that there is an obvious drop in the performance of the model compared with the big dictionary size.

5.2 Explain the drop in the performance.
When we only use small number of clusters, many features are clustered into a group even though they are extracted from different patterns (i.e., they might be visually irrelevant and contain different kinds of structure). That is probably why the performance drops when a small dictionary size is used.

When using clustering, a necessary condition for this is that the cluster center is a meaningful representative for the whole cluster. In that respect, it becomes evident that the goal of the grouping stage must not be to obtain the smallest possible number of clusters, but to ensure that the resulting clusters are visually compact and contain the same kind of structure. This is an important consideration to bear in mind when choosing the clustering method. [1]

Reference:
[1] Leibe, Bastian, Aleš Leonardis, and Bernt Schiele. "Robust object detection with interleaved categorization and segmentation." *International journal of computer vision* 77, no. 1-3 (2008): 259-289.