

Scene Recognition with bag of words Documentation

PROJECT 3

Alaa Hesham | ID:201500638 | CIE 552 | Date :20/4/2018

INTRODUCTION

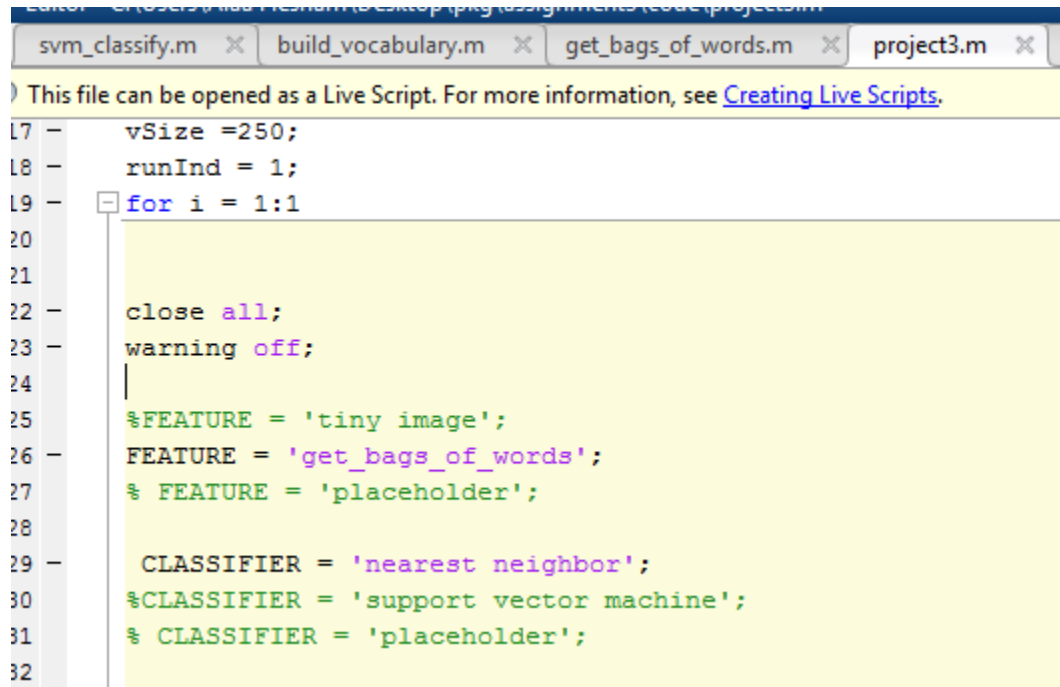
We will perform scene recognition with three different methods. We will classify scenes into one of 15 categories by training and testing on the 15 scene database

Task: Implement three scene recognition schemes:

- Tiny images representation (`get_tiny_images.m`) and nearest neighbor classifier (`nearest_neighbor_classify.m`).
- Bag of words representation (`build_vocabulary.m`, `get_bags_of_words.m`) and nearest neighbor classifier.
- Bag of words representation and linear SVM classifier (`svm_classify.m`).

MANUAL

Run project 3 script to see output, change feature, and classifier to choose either tiny image or bag of words, and change classifier to choose nearest neighbor or SVM .

A screenshot of the MATLAB script editor. The top toolbar shows several tabs: 'svm_classify.m', 'build_vocabulary.m', 'get_bags_of_words.m', and 'project3.m'. The 'project3.m' tab is active. Below the tabs, a yellow banner reads: 'This file can be opened as a Live Script. For more information, see [Creating Live Scripts](#).' The script content is as follows:

```
17 - vSize = 250;  
18 - runInd = 1;  
19 - for i = 1:1  
20 -  
21 -  
22 - close all;  
23 - warning off;  
24 -  
25 - %FEATURE = 'tiny image';  
26 - FEATURE = 'get_bags_of_words';  
27 - % FEATURE = 'placeholder';  
28 -  
29 - CLASSIFIER = 'nearest neighbor';  
30 - %CLASSIFIER = 'support vector machine';  
31 - % CLASSIFIER = 'placeholder';  
32 -
```

THEORY AND IMPLEMENTATION DECISIONS

In **tiny images**, we simply resize image to be 16*16 and then consider these 256 pixel values as our features vector.

In **nearest neighbor classify**, we get tiny image of train function besides test function we then create matrix that represent distance between a given test image features and all train image features and then find the train image with minimum distance . The label of test image will be the same as train image.

In **build_vocabulary** , we want to have option to sample images from set of image specified in the `image_path` folder . Then for every image we will pick certain number of features. Then we will collect all features in one matrix, and use function `vl_kmeans` to divide them into number of centroids equivalent to number of vocab size * 128 .

As a reflection of implementation decisions, `n_img_samples` variable is used to give you option of either take all images specified in the image path or you can sample from them. If you choose to sample then `Rand` function is used to generate random indices.

The same applies of `no_of` features or interest points you can pick them I choose 100 points or just take the default of `vdsift` that's why we have variable `no_features` .

In **get bag of words**, at the first glance it's so similar to build vocabulary however it has really different purposes , first it doesnot sample images . Second it already has visual vocabs then we have two loops first one loop over all images , second one loop on every feature on this image (also unlike `build_vocabulary` , we do not sample features) and assign every feature to a class .Yes , every image has set of features , each feature belongs to a class then the class to which image belongs is the class that has maximum number of features described it as their class .

As a reflection of implementation decisions ,
`image_feats = zeros(n_images,n_clusters)`

```

    %get closest cluster center matched
    [minVal, ind] = min(dist);

    image_feats(i,ind) = image_feats(i,ind) + 1;
end

%%

% normalise histogram
maxfeature_value = max(image_feats(i,:));
image_feats(i,:) = image_feats(i,:)/maxfeature_value;

```

SVM classify

I have used `vl_svmtrain` function
 This function trains linear svms based on training examples, binary labels (-1 or 1), and LAMBDA which regularizes the linear classifier by encouraging W to be of small magnitude. LAMBDA is a very important parameter by some trial and error when $\lambda = 0.000008$ it results in better performance (62.8% accuracy)

As a reflection on implementation, the point is that we use linear and binary classifier so we should for every image feature test if it belongs to a certain category vs all other categories. And loop over other categories in the same manner .After that for every image, we see which category get the maximum number of votes so it wins .

Here is the code that illustrate how this could be achieved where M is image features

```

%iterate through each test image feature
for i = 1:M

    current_feat = test_image_feats(i,:);

    current_confid = zeros(1,num_categories);

    for j = 1:num_categories
        tempW = W(j,:)';
        current_confid(j) = dot(tempW, current_feat') + B(j,:);
    end

    [maxVal maxInd] = max(current_confid);

    predicted_categories(i,1) = categories(maxInd);
end

```

RESULTS

```

accuracy =

    0.2100

```

Figure 1 Tiny image with nearest neighbour

```

ans =

     7

accuracy =

    0.5200

Time taken = 4.871172e+03

```

Figure 2 Bag of words with nearest neighbour

```
Accuracy (mean of diagonal of confusion matrix) is 0.619
```

```
ans =  
7
```

```
accuracy =  
0.6187
```

```
Time taken = 4.948750e+03
```

Figure 3 Bag of words with SVM

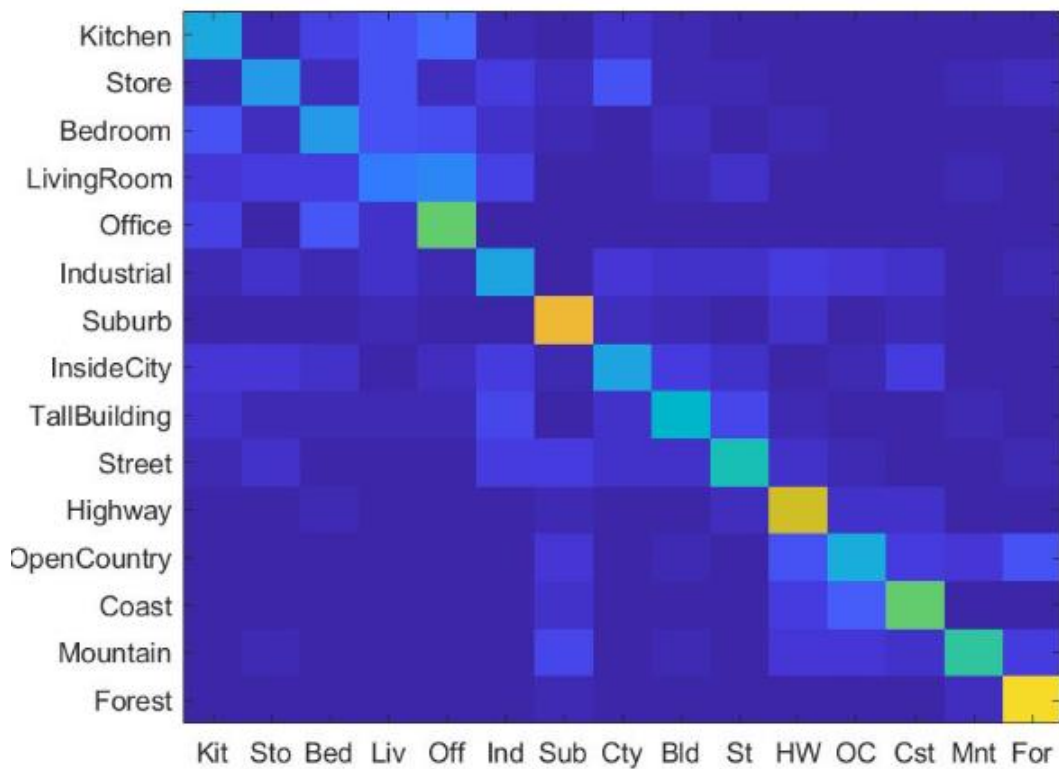


Figure 4 Histogram 1

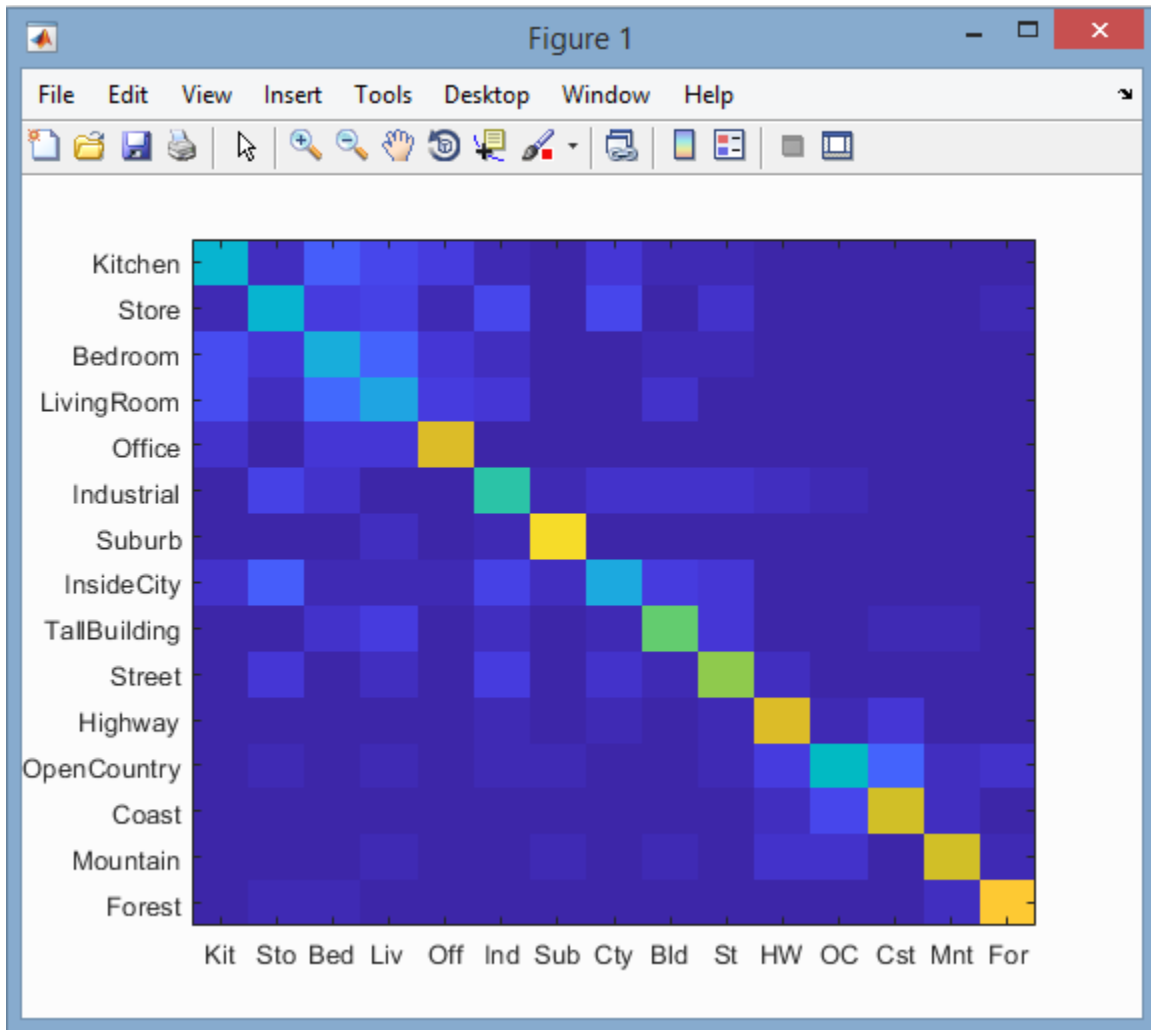


Figure 5 Histogram 2

Algorithm	Accuracy
Tiny image with nearest neighbour	0.21
Bag of words with nearest neighbour	0.52
Bag of words with SVM	0.69

Disclaimer

Code takes about 1 hour to run bag of words as I don't sample images I just sample features to achieve kind of high accuracy .