EE462, EE9SO25, EE9CS728

Machine Learning for Computer Vision

Coursework 3 (25% mark)

Release on 17 Nov 2014, the report due in 2 weeks (on 30 Nov 2014, 11:59pm)

About lectures 7-8 and lectures 9-10

(Randomised Decision Forests, Bag of Words, Support Vector Machine)

This course work requires Matlab programming. Use the provided Matlab codes and data.

Submission instructions: Submit the ***Matlab code and data*** that you wrote, as well as, the report, through the Blackboard system, electronically. And drop a hardcopy of the report in the homework dropbox at EEE 1017. Write your full name and CID number on the top of the first page.

Note: Some questions require writing your lines of code in the report also.

In this coursework, we learn the machine learning algorithm called randomised decision forest (RF) by Matlab. Following the lecture on RF, we implement the algorithm, see into the core parts of the algorithm step-by-step, while training and testing it on the toy data sets and Caltech101 image categorisation data set. Try also different parameter values of RF and see their effects. Although the coursework is limited due to the given time and efficiency of the Matlab codes provided, for a real-scale problem, ponder about why RF is so important in computer vision and machine learning and in comparison to other alternative methods, in terms of scalability, efficiency, multi-classes, and generalisation. For its various up-to-date application works, please have a look at http://www.iis.ee.ic.ac.uk/icvl/publication.htm.

1. Getting Started

Start Matlab, go to the directory where the provided files are. Open the main_guideline.m then copy and paste lines of code to the command window of Matlab and press Enter to execute them, step by step, as guided below.

Do initialisation.

```
>> init;
```

Set the random forest parameters.

```
>> param.num = 10; % Number of trees
>> param.depth = 5; % trees depth
>> param.splitNum = 3; % Number of random feature/threshold pairs in split
function
>> param.split = 'IG'; % Currently support 'information gain' only
```

2. Loading datasets

We provide three toy data sets, each of which is a set of 2D points with their class labels, and the Caltech101 image categorisation data set to experience a real vision problem. Start with the Toy Spiral data set.

Select a dataset.

```
>> [data_train, data_test] = getData('Toy_Spiral'); % {'Toy_Gaussian',
'Toy_Spiral', 'Toy_Circle', 'Caltech'}
```

Check the training and testing data loaded. The data formats are

```
data_train(:,1:2) : [num_data x dim] Training 2D vectors,
data_train(:,3) : [num_data x 1] Label of training data, {1,2,3}.
```

Plot the training data.

```
>> plot_toydata(data_train);
```

The test data formats are

```
data_test(:,1:2) : [num_data x dim] Testing 2D vectors, 2D points
in the uniform dense grid within the range of [-1.5, 1.5],
data_test(:,3) : N/A.
```

Plot the testing data.

```
>> scatter(data_test(:,1),data_test(:,2),'.b');
```

3. Training Random Decision Forest

We learn the random forest using the training data above. Given the single set of data points, we first generate multiple subsets by Bagging (Boostrap Aggregating). Each of the bagged data sets is formed by randomly subsampling the original data set. Then, we grow a tree per each subset, by recursively splitting the tree nodes using the information gain. The tree growth stops when it reaches a predefined depth or meets stopping criterions.

3.1. Generating data subsets randomly

We do bagging to create subsets of training data.

```
>> [N,D] = size(data_train);
>> frac = 1 - 1/exp(1); % Bootstrap sampling fraction: 1 - 1/e (63.2%)
>> [labels,~] = unique(data_train(:,end));
```

Q1 [10]. We generate e.g. 4 data subsets. Write your own codes to generate the 4 subsets (you may use randsample). Explain the meaning of prior in the provided code and show their values in the report. Show the data subsets using plot_toydata. Discuss the results and the meanings of Bagging in this case.

```
% idx: index of a data subset, for which a tree is grown.
```

3.2. Splitting nodes

Now we take a data subset generated above, and grow a tree by splitting the data. Here we grow the first tree, out of (param.num) trees.

```
>> T=1; % the trees number 1
```

Q2 [5]. In order to split nodes, we measure the entropy of the class distributions. Write your own codes to measure the entropy, and provide the lines of code in the report.

```
function H = getE(X) % Entropy

…

End
```

Split the first (i.e. root) node. We try a few of split features and thresholds randomly chosen, decide the best ones in terms of information gain, and save them in the node.

Q3 [5]. Complete the provided code below, and show your own lines of code in the report.

```
>> ig_best = -inf;
>> for n = 1:3
>>    dim = randi(D-1); % Pick one random dimension as a split feature
>>    d_min = single(min(data_train(idx,dim))); % Find the data range of this
dimension
>>    d_max = single(max(data_train(idx,dim)));
>>    t = d_min + rand*((d_max-d_min)); % Pick a random value within the
range as a threshold
>>

% write your own codes here…

% idx_: index of data going to the left-child node by chosen dimension and
threhold

>>    if ig_best < ig
>>        ig_best = ig; % maximu information gain saved
>>        t_best = t; % the best threhold to save
>>        dim_best = dim; % the best split function (dimension) to save
>>        idx_best = idx_;
>>    end
>>
>>    % Visualise the split function and its information gain
>>    visualise_splitfunc(idx_,data_train(idx,:),dim,t,ig,0);
>>    drawnow;
>>    pause; % please hit any key to continue
>> end
```

We visualise the best split function saved.

```
>> visualise_splitfunc(idx_best,data_train(idx,:),dim_best,t_best,ig_best,0);
```

Q4 [5]. Show the class distributions and information gains of different split functions in the above. Discuss the results with the underlying ideas of maximising the information gain.

Initialise the base node.

```
>> trees(T).node(1) = struct('idx',idx,'t',nan,'dim',-1,'prob',[]);
```

We now split the nodes recursively in the way above to further grow a tree. Split the nodes recursively. Complete the code splitNode using the above.

```
>> for n = 1:2^(param.depth-1)-1
>>     [trees(T).node(n),trees(T).node(n*2),trees(T).node(n*2+1)] =
>>splitNode(data_train,trees(T).node(n),param);
>> end
```

The tree growth is done, we save the class distributions in the leaf nodes.

```
>> makeLeaf;
```

Q5 [5]. (related to Q4) Visualise the class distributions of the first 9 leaf nodes for example. Discuss the results.

```
>> visualise_leaf;
```

Similarly, we grow multiple trees, a tree per a subset of data. Complete the code using the above. Use the settings in 1. Getting Started (e.g. param.num = 10; % Number of trees)

```
>> trees = growTrees(data_train,param);
```

4. Evaluating Test Data by the Random Decision Forest

Using the random forest we trained above, we evaluate novel data points in the test dataset. Let us grab the few data points and evaluate them one by one by the leant RF.

Q6 [5]. Show the results and explain them, in the report.

```
>> test_point = [-.5 -.7; .4 .3; -.7 .4; .5 -.5];
>>
>> figure(1)
>> plot_toydata(data_train);
>> plot(test_point(:,1), test_point(:,2), 's', 'MarkerSize',20,
'MarkerFaceColor', [.9 .9 .9], 'MarkerEdgeColor','k');
>>
>> for n=1:4
>>    figure(2)
>>    subplot(1,2,1)
>>    plot_toydata(data_train);
>>    subplot(1,2,2)
>>    leaves = testTrees([test_point(n,:) 0],trees);
>>
```

```
>>    % average the class distributions of leaf nodes of all trees
>>    p_rf = trees(1).prob(leaves,:);
>>    p_rf_sum = sum(p_rf)/length(trees);
>>
>>    % visualise the class distributions of the leaf noes which the data
>>    % point arrives at
>>    for L = 1:size(leaves,2)
>>        subplot(3,5,L); bar(p_rf(L,:)); axis([0.5 3.5 0 1]);
>>    end
>>    subplot(3,5,L+3); bar(p_rf_sum); axis([0.5 3.5 0 1]);
>>
>>    figure(1);
>>    hold on;
>>    plot(test_point(n,1), test_point(n,2), 's', 'MarkerSize',20,
>>'MarkerFaceColor', p_rf_sum, 'MarkerEdgeColor','k');
>>    pause;
>>end
>>hold off;
>>close all;
```

Q7 [5]. We evaluate all the data points in the dense 2D grid, and visualise their classification results, color encoded. Show and discuss in brief the result.

```
>> leaves = testTrees_fast(data_test,trees);
>>
>> for T = 1:length(trees)
>>    p_rf_all(:,:,T) = trees(1).prob(leaves(:,T),:);
>> end
>>
>> p_rf_all = squeeze(sum(p_rf_all,3))/length(trees);
```

Visualise the classification results of RF.

```
>> visualise(data_train,p_rf_all,[],0);
```

5. Comparison between SVM and RF

Q8 [15]. We evaluate all the data points in the dense 2D grid (above), and visualise the classification results, color encoded, by Support Vector Machine (SVM), a standard nonlinear classification method.
You may use the provided code (libsvm-3.18), or the statistical pattern recognition toolbox which we used for the coursework2.

Using the results, explain the way of setting the SVM parameters (discuss underfitting/overfitting in this case), the way of extending the binary SVM to the multi-class problem (did we use one-versus-the-rest or one-versus-one?). Check how many support vectors did we obtain and show the support vectors (by varying the SVM parameter values).

Compare and discuss the result of RF with that of Support Vector Machine (SVM).

Do SVM for comparison.

```
>> disp('Training SVM...');
>> model_svm = svmtrain(data_train(:,end),data_train(:,1:end-1),'-t 2 -b 1 -c
>> 10');
>>
>> disp('Testing SVM...');
>> [p_svm, accuracy_svm, p_svm_prob] = svmpredict(data_test(:,end),
>>data_test(:,1:end-1), model_svm, '-b 1');
```

Visualise the classification results of both RF (left) and SVM (right).

```
>> visualise(data_train,p_rf_all,p_svm_prob,1);
```

6. Trying different parameter values of RF and see the effects

Q9 [15]. RF has a few of important parameters, which need to be set to achieve good results. Here we try changing the number of trees and the depth of trees. Try the different parameter values and show/discuss the results comprehensively.

Set the random forest parameters. We change the number of trees.

```
>> for N = [1,3,5,10,20]; % Number of trees, try {1,3,5,10, or 20}
>>    init;
>>    param.num = N;
>>    param.depth = 5; % trees depth
>>    param.splitNum = 3; % Number of trials in split function
>>    param.split = 'IG'; % Currently support 'information gain' only
>>
>>    % Select dataset
>>    [data_train, data_test] = getData('Toy_Spiral'); % {'Toy_Gaussian',
>>'Toy_Spiral', 'Toy_Circle', 'Caltech'}
>>
>>    % Train Random Forest
>>    trees = growTrees(data_train,param);
>>
>>    % Test Random Forest
>>    testTrees_script;
>>
>>    % Visualise
>>    visualise(data_train,p_rf,[],0);
>>
>>    pause;
>> end
```

Change the tree depth.

```
>> for N = [2,3,5,11]; % Tree depth, try {2,3,5, or 11}
>>    init;
>>    param.num = 20; % number of trees
>>    param.depth = N; % trees depth
>>    param.splitNum = 3; % Number of trials in split function
>>    param.split = 'IG'; % Currently support 'information gain' only
>>
>>    % Select dataset
>>    [data_train, data_test] = getData('Toy_Spiral'); % {'Toy_Gaussian',
>>'Toy_Spiral', 'Toy_Circle', 'Caltech'}
>>
>>    % Train Random Forest
>>    trees = growTrees(data_train,param);
>>
>>    % Test Random Forest
>>    testTrees_script;
>>
>>    % Visualise
>>    visualise(data_train,p_rf,[],0);
>>
>>    pause;
>> end
```

7. Experiment with Caltech101 dataset for image categorisation

We have seen the RF on the toy data above. In this part, we see how we apply RF to a real vision problem. We use a small subset of Caltech101 dataset for image categorisation. The dataset contains a number of different object categories (or classes), and a large number of images per category. In principle, we convert images to high dimensional vectors using the popular technique called bag-of-words, and then apply RF to the vectors in the same way above for the toy data experiments. Note that the toy data has 2 dimensional vectors, while the bag-of-words vectors of images have a higher-dimension.

Initialise and set the RF parameters.

```
>>  init;
>> param.num = 20; % number of trees
>> param.depth = 10; % trees depth
>> param.splitNum = 3; % Number of trials in split function
>> param.split = 'IG'; % Currently support 'information gain' only
```

Q10 [20]. Write your own codes for the bag-of-words by k-means. Use and complete the provided code *(the correctness and efficiency of the code will be marked)*. Show example images and their bag-of-word histograms. Discuss the results, the quantisation process, and the possible way(s) to accelerate the quantisation process.

Select dataset. We do bag-of-words technique to convert images to vectors (histogram of codewords).
Set 'showImg = 0' in getData.m if you want to stop displaying training and testing images and their feature vectors.

```
>> [data_train, data_test] = getData('Caltech'); % Select dataset
```

Train Random Forest.

```
>> trees = growTrees(data_train,param);
```

Test Random Forest.

```
>> testTrees_script;
```

Q11 [10]. Show recognition accuracy and confusion matrix in the report. Discuss the results using the confusion matrix, and example images and their success/failure cases.

```
>> confus_script;
```