Machine Learning for Computer Vision: Coursework 1 - Randomised Decision Forests

Tormento, Marion
Imperial College London
marion.tormento17@imperial.ac.uk

McLaughlin, Edward
Imperial College London
edward.mclaughlin17@imperial.ac.uk

Abstract

In this paper, the properties and power of Random Forests (RFs) is explored. In question one and two a RF classifier is implemented on a three-class spiral data set, and the resultant forest is tested on sample points and a blanket array of points. In question three, RFs are used to classify the CalTech101 images provided. Firstly, a k-means codebook was built, subsequently its performance was compared with an RF codebook, in each case using an RF classifier.

For this report, the entirety of the code to construct the RF algorithm (except for the getData.m function) was created by the authors. That is to say that the vfleat package was still utilised to process the images. The functions created are available at https://github.com/MarionTormento/MLCV for your information.

Question 1

Figure 1 shows the result from bagging the data into four bags *with* replacement. The size of the bags is determined by equation 1, which results in the bags being roughly 63.2% of the original data set i.e. the bootstrap sampling fraction (BSF).

$$BSF = 1 - \frac{1}{e} \tag{1}$$

It can be seen that the bags display similar, but not identical, histograms. Thus a source of randomness is introduced into the algorithm in this initial step.

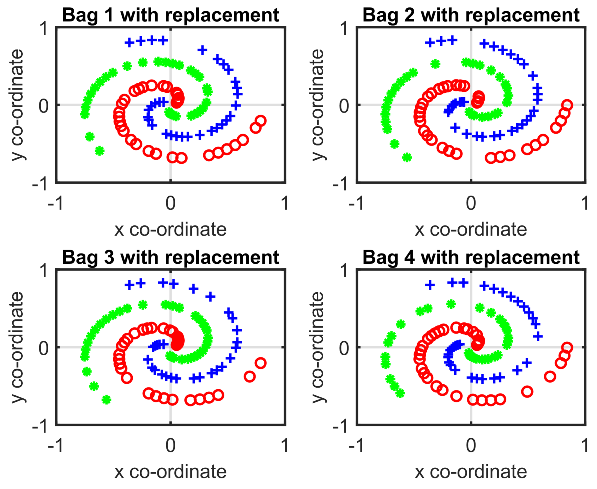
Each bag is used as the root node for each tree in the Randomised Forest (RF). The root node is split in an optimal manner, explained later in the section, to maximise the Information Gain (IG) produced by the split function. This split of the root node is shown graphically in Figure 2. The IG calculation is given in equation 2.

$$I(S,\Theta) = H(S) - \sum_{i \subset L,R} \frac{|S^i|}{|S|} H(S^i) \tag{2}$$

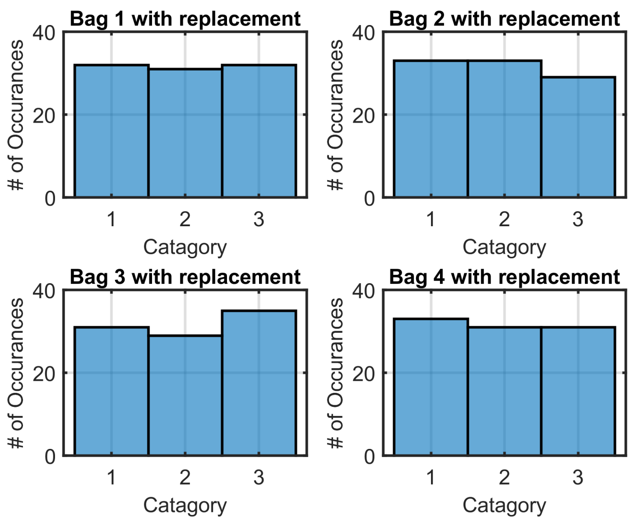
where, S is the total set of data points, S^L and S^R are the two children sets of data defined by the split function in question and H is the Shannon entropy defined by

$$H(S) = -\sum_{c \subset C} p(c) log(p(c)) \tag{3}$$

where S is the set of data points, c denotes the class in question and C denotes the full set of classes considered (here C=$\{1,2,3\}$).



(a) Data in each bag plotted.



(b) Histogram of the data classes in each bag.

Figure 1. Bootstrap Aggregating (Bagging) *with* replacement.

From this split, two children are produced. Each child, if not deemed to be a leaf node, will also be optimally split. This process continues until all branches end with a leaf node. A leaf node is created if: a) there are fewer than 10 points in the child, b) there is only one class left in the child, c) the child node is part of the final level of the tree.

The process of finding an optimal split function is performed as follows: 1) parameter ρ (i.e level of randomness) is defined. 2) each split node function type (either x- or y-axis aligned or linear) is tested with random parametrisations (gradients, y-intercepts etc.) a set number of times, dependent on the chosen value of ρ, 3) the split function calculated to have the highest Information Gain (IG) is selected as the optimal split function. This manner of selecting optimal split functions based on random parametrisation introduces a second source of randomness into the system. In Figure 2 the split which renders the highest IG is plotted in red for each split case (e.g. x-axis, y-axis, or linear). In both cases, this creates one child with only one class in it, and a second child with a mix of classes. In general, high IG will come from split functions which maximise the separation of classes in the children nodes, and as such, is desirable. Here the best split is the red axis-aligned (left), and Figure 3 presents the histograms of the children nodes created by this split.

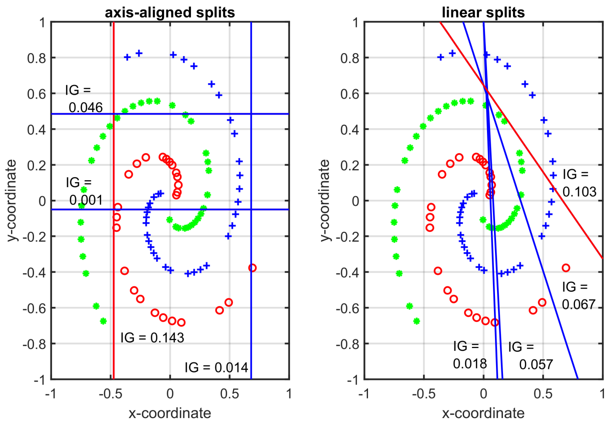


Figure 2. Example split functions shown ($\rho = 2$): on the left the, two functions in both x- and y-axis aligned, on the right, four functions in linear aligned. The Information Gain (IG) of each split function is displayed adjacent to the line, the function with the highest IG in each case is highlighted in red. The best split functions for each type (axis-aligned or linear) are then compared and the optimal one is finally selected for the split function.

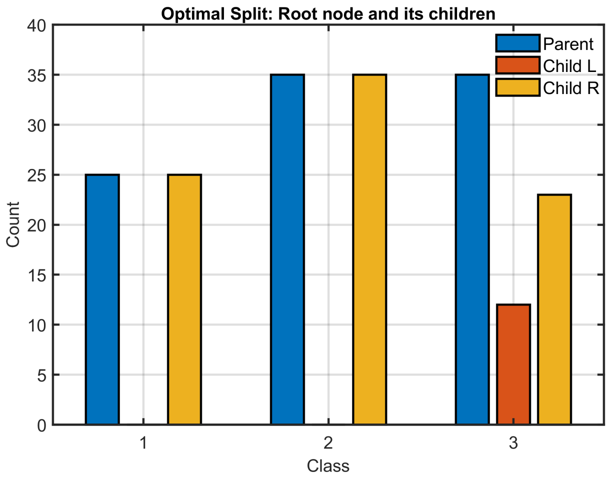


Figure 3. Histogram showing the class distributions of the root node (parent) and two children nodes having applied the axis-aligned split in red shown in Figure 2.

As mentioned previously, three stopping criteria are used to decide whether a node is a split node or a leaf node. Stopping criterion a) was decided, by means of a parameter sweep, because when splitting leaves with fewer than 10 data points the accuracy did not increase noticeably while the training time increased unfavourably. Stopping criteria b) was chosen since a leaf with one class label is the desired outcome. Finally, criterion c) were chosen to ensure all branches end with a leaf node. Figure 4 presents the repartition of leaves nodes for a random forest of eight trees with eight layers each, and $\rho = 10$ (only three leaves out of three trees taken randomly are represented here). In seven out of the nine leaves, there is only one class apparent, from this it can be deduced that under these conditions, stopping criterion b) - the most desirable - is used most frequently, and that the random forest will give accurate prediction when testing. In total for this configuration, 88 out of the 105 leaves return a single label, roughly 84%.

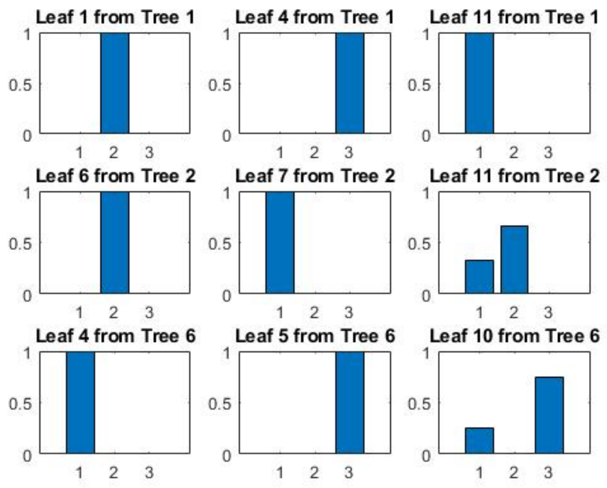


Figure 4. Repartition of nine leaves nodes for a random forest of eight trees, eight layers.

Question 2

The final parameters used for the various plots presented in this question are eight trees, with eight layers each, $\rho = 10$. For these parameters the computation time is 2.66s and accuracy is 98.67% for training. This question will also present how these parameters were chosen.

To evaluate the performance of the RF for the chosen parameters a dense 2D grid of points is used to visualise a colour data space. To this end, Figure 5 shows a map of the classification at each point, whereas Figure 6 visualises the probability distribution of each point by weighting its RBG values by the probability that they belong to the red, green or blue class respectively. The 2D grid captures the shape of the spiral towards the centre of the grid and splits the outside of the grid (where the training data is more sparse) appropriately. From the high contrast and hard boundaries of the colour coded spirals in Figure 6, it can be concluded that these parameters are able to classify data points with good confidence levels.

Subsequently, several points are taken and their class probabilities are visualised using histograms (Figures 17(a) - 17(d) in the Appendix). These histograms show the contribution of the leaf node from each tree which the test point ended the training in. Also shown is the normalised probability that the given test point belongs to each class. Comparing these predicted labels to the location of the test point in the training grid shows that the random forest gives accurate classification.

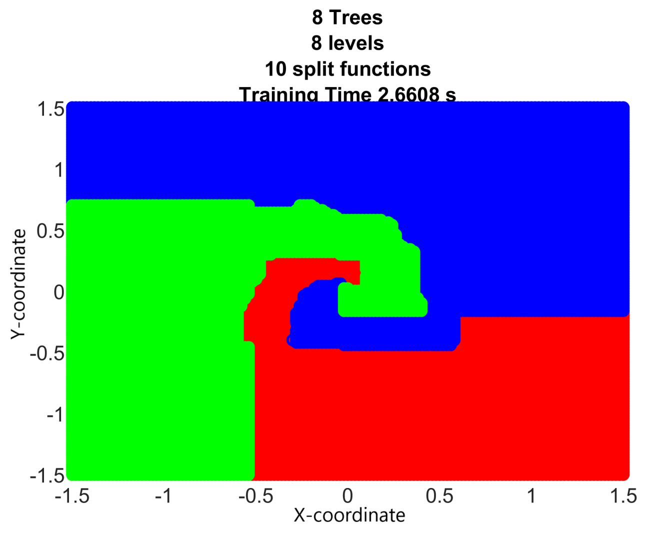


Figure 5. 2d grid of points to test their classification graphically. Red = Class 1, Blue = Class 2 and Green = Class 3.

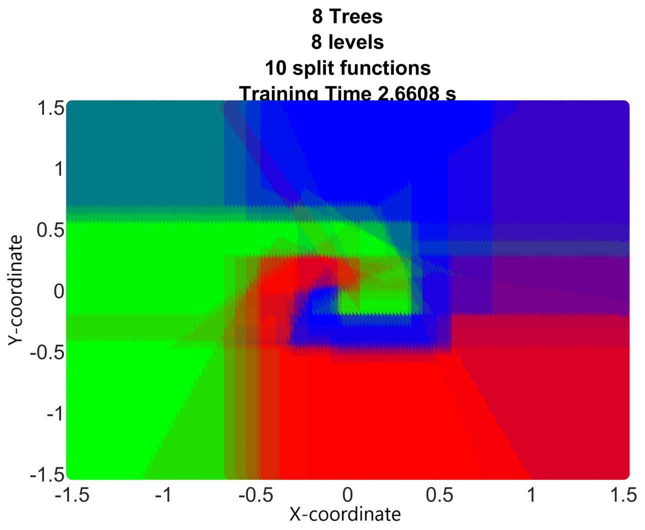


Figure 6. 2D grid of points where the RGB value of the data point corresponds to its probability distribution of being classified in each class.

Due to the simplicity of the problem, various triplets of parameters give good results in a fast training time (more than 95% accuracy on training data, for a training time of approximately 2s). Nevertheless, some particular behaviours correlate with the number of trees, number of layers and ρ. In particular increasing the number of layers, for a fixed number of trees and ρ increases the accuracy (see Figure 7). For seven layers, the training will typically result in an accuracy of 90% or more for any number of trees ($\geqslant 4$) and ρ ($\geqslant 3$). Nevertheless, this growth stops at a point: indeed if the tree has too many layers, the children will all turn to leaves before reaching the last layer resulting in empty, and hence useless, layers.

Question 3

Question 3 is concerned with image classification. In particular, the Caltech101 image set is taken to be classified. This classification is made by firstly producing a codebook from 128 dimensional image descriptors. This codebook is then used to create a bag-of-words for each training and testing image. These bag-of-words are used to train and test on a randomised forest. In Part 1, the codebook is created using a k-means approach. Part 2 is concerned with classification using a randomised forest. In Part 3, a new codebook is created using a randomised forest and the classification results are compared with those found in Part 2 for a k-means codebook.

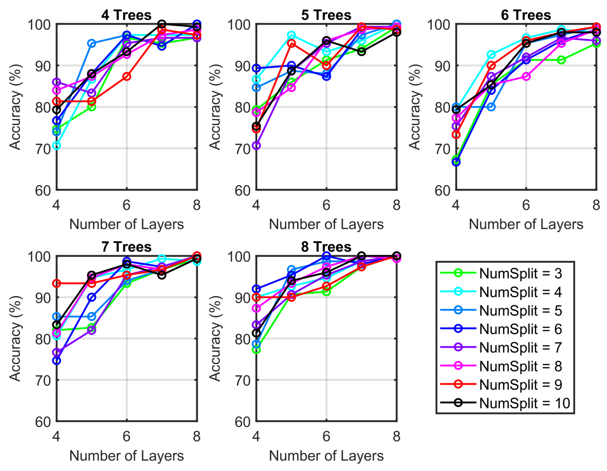


Figure 7. Example accuracy plot depending on the number of layers, for various number of trees, and ρ (= NumSplit).

Part 1

Using 100k randomly selected descriptors from the training images, a k-means algorithm is used to find the centroids of the data in 128 dimensional space. This set of N centroids is used as the 'vocabulary', each centroid being a 'word'. Utilising N as an input parameter gives control over the size of vocabulary used.

Using a k-nearest neighbours algorithm from matlab, the full set (150 images - 15 images for all 10 categories) of training image descriptors are then associated with the closest 'word' in the 128D space. For each image, each descriptor is linked to one of these words. This is done using the knnsearch matlab function which returns a vector $(1 \times size(image, 2))$ indicating the nearest centroid to each descriptor. This $1 \times size(image, 2)$ vector is turned to a $1 \times N$ vector of occurrences of each words thanks to the hist function (see 1.2.2, Line 40). This vector represents for each images its bag-of-words. The label of each image is concatenated to the end of the corresponding bag-of-words vector that is then stored in the 'data˙train' matrix (see 1.2.2, Line 41). These bags of words are used as the training data for the Randomised Forest classification in Part 2 of this question. This process takes on average 11 seconds to run which is acceptable considering that each of the 150 images is composed of around 3000 descriptors.

The previous step is repeated using the test images (again 150 images split into 10 categories - see 1.2.2, from Line 58 to 69). The bags-of-words for each image is used for the testing phase of the Random Forest classification.

In the end, the getData function returns two 150 (15 images x 10 categories) by N+1 (N words + one label) matrices that represents testing and training (see code in Appendix 1.2.2).

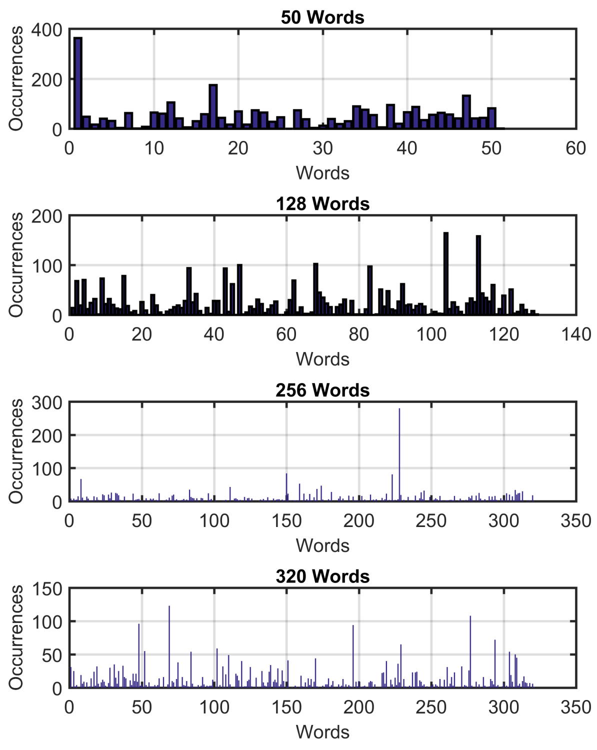


Figure 8. Histogram comparison for a 'Tick' image (category 1) varying the size of the vocabulary (N) in the bag-of-words.

Figure 8 shows the bags-of-words histograms of differing vocabulary sizes using the k-means algorithm to create the codebook. When increasing the number of words, the histograms become more sparse. An initial guess is that wider vocabulary will give more accurate predictions with the random forest but will also increase the computational time. **An optimum is found for 320 words** : fewer words decreases the accuracy, whereas more increase the computational time for a practically unchanged accuracy (see Figure 11).

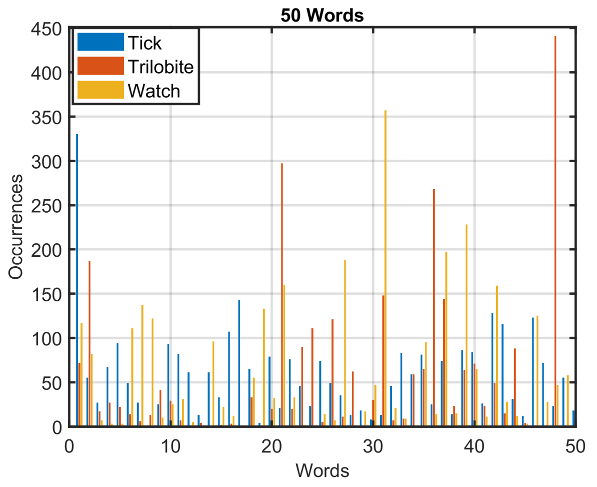


Figure 9. Comparison of the histograms for various images categories ('Tick', 'Trilobite' and 'Watch'). For clarity, a vocabulary size of 50b words was chosen to demonstrate the differences between images.

Figures 9 and 10 show the bag-of-word histograms for three images in different and the same category respectively. It can be concluded visually that the histograms for images in different categories have a large amount of variability compared with the histograms for images in the same category. This intuition is quantitatively confirmed thanks to a two way ANOVA test performed on the data shown in Figures 9 and 10. The P values for the tests are calculated using an ANOVA2 test. It is found that for the histograms of images from different categories the P value is 0.9746, nearly twice as high as the P value for for images from the same category (=0.5967) proving that the variability is way less important in the second case. The data is handled such that they are a 3x50 matrix, where three images contain a bag-of-words 50 words long. The tabular results are given in the Appendix in Figure 18.

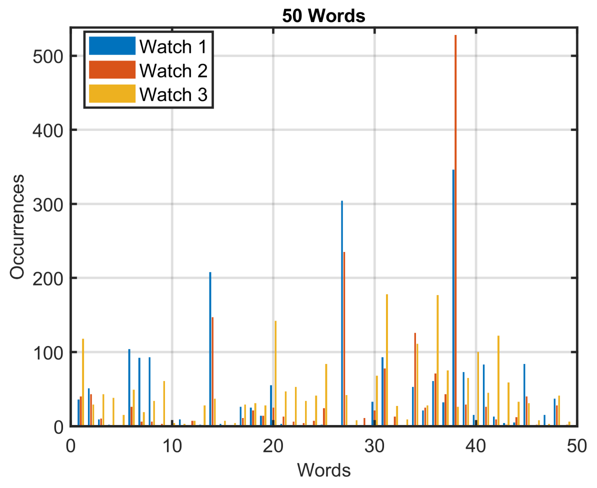


Figure 10. Comparison of the histograms for various images from the 'watch' category. For clarity, a vocabulary size of 50 words was chosen to demonstrate the similarity between images.

Part 2

Using the training data from the k-means codebook, each images bag-of-words is recursively split until one of the stopping criteria to create leaf nodes is met. Once the tree has finished training, each leaf has a set of images in it. The categories of these images creates the leafs probabilistic category distribution. That is to say, in the testing phase, an image which ends up in a leaf will have this probability distribution of belonging to each category. These probabilities are then averaged over all trees in the forest and a best guess of the images category is made.

A similar method to the one presented for question 2 is used in this question to grow and test the random forest (e.g. optimal split between various type of weak learners). The only difference being that each split is performed on a random plane with dimensions sampled from the N-D vocabulary.

Figure 11 shows the impact of the vocabulary size on classification accuracy and on computational time. **Each trial has been computed for the same random forest configuration: 25 trees, eight layers and ρ = 10.** A maximum of 65.3% accuracy is reached for $N = 320$ words in the vocabulary for a reasonable computational time. This value will be used for the rest of the question.

Figure 12 shows the impact of the number of trees, layers and ρ on the classification accuracy. After various trials, the best accuracy is obtained for 25 trees, eight layers and $\rho = 10$. This

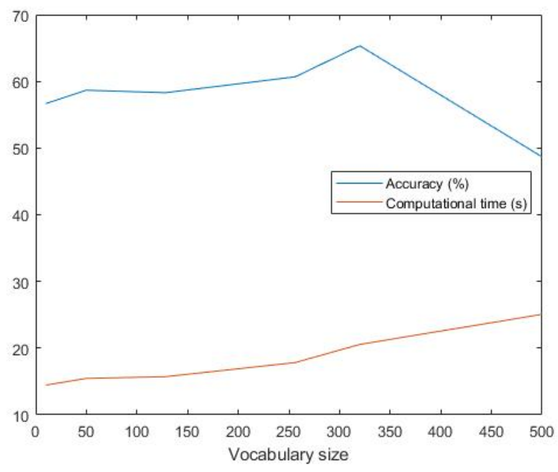


Figure 11. Computational time and accuracy for different vocabulary size with the same forest configuration (25 trees, eight layers, $\rho = 10$).

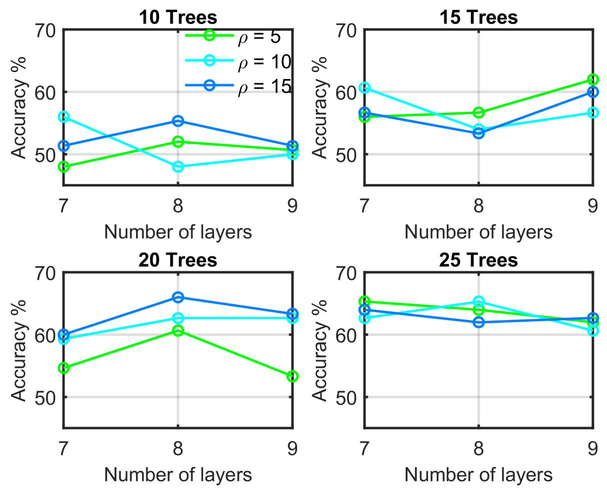


Figure 12. Accuracy for various forest configurations.

value is also optimized in terms of computational time: better accuracy could be reached with more trees or higher ρ, but the simulation would run for a significantly larger amount of time, e.g. for 50 trees instead of 25, the classification accuracy is 68.6%, only 3% more, for twice the computational time (40.7s compared to 20.01s). With these parameters the training and testing runs on average for 20s and 0.3s respectively. This configuration will be used for the rest of the question.

Figure 13 and Table 2 (in Appendix for reasons of clarity) give the confusion matrix obtained for such settings. Out of the 10 categories, 7 are predicted with more than 50% accuracy, and 4 with more than 70% (these corresponds to the trilobite, water lilly, wheelchair, and windsor chair). In general, it can be seen that the on-diagonal terms have the highest values in their rows and columns proving that the actual category is the one predicted most of the time. Some errors are understandable: for example, the second most guessed for the windsor chair, is a wheelchair. Other error seem to be less logical: a wild cat is often confused with a wheelchair.

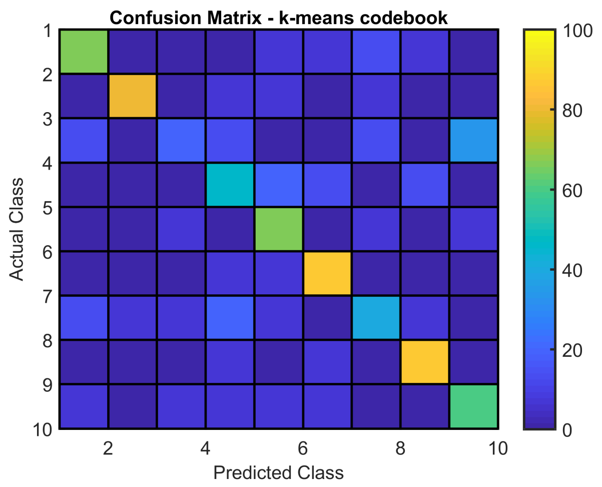


Figure 13. Confusion matrix for the RF classification using a k-means codebook. RF configuration: 25 trees, eight layers, $\rho = 10$, vocabulary size 320.

Part 3

In this part, the codebook is constructed using a random forest. A random selection of 100k descriptors is recursively split along a dimension (axis-aligned split) or a pair of dimensions (linear split) which are randomly selected from the 128 dimensions of the descriptors.

Once the conditions to create leaf nodes are met, the bag-of-words vocabulary is known - where each leaf is a word. In this case, the leaves are analogous to the centroids found in Part 1 using the k-means algorithm. It is important to note that the number of leaves is not predefined when using a RF to construct the codebook whereas the number of centroids is predetermined in the k-means algorithm. However, it is also noted that RFs give control of the algorithm through parameters such as number of trees, number of levels, value of ρ. These are alternative considerations to performance and efficiency which are important when choosing by which means the codebook is created.

Once more, the training and test image descriptors are converted into a bag-of-words for each image by the means of vector quantisation.

These bags-of-words for each training images are used to train the random forest classifier as described in Part 2. The test images are subsequently classified using the Random Forest. The results are analysed and compared with those found using the k-means codebook.

Various sets of parameters shown in Table 1 were tested to build the RF codebook. Increasing the number of trees or ρ drastically increases the training and testing time, making it temporally expensive to collect and test as many codebooks as it was done in Part 1. As in Part 1, increasing the number of layers above a certain threshold does not drastically increase the accuracy. Indeed, when there are few layers (in the fourth case for example with 8 layers - see Table 1) the vocabulary size is close to the maximum number of leaves possible. From this it can be concluded that most of the leaves are created from criterion ’c)’ which is suboptimal. Conversely, when there are many layers (in the third and fifth case for example - respectively 14 and 15 layers - see Table 1) the vocabulary size is considerably less than the maximum number of leaves, meaning that the final layers of the trees are possibly empty. The optimal number of layers seems to be around 10 layers. The two

RF codebooks used for the rest of the question are the first (five trees, 10 layers, $\rho = 5$) and the second (10 trees, 10 layers, $\rho = 10$). They will be referred to as 5/10/5 and 10/10/10 respectively.

Table 1. Vocabulary size, Training and testing time for the various configurations of the RF codebook

Trees (T)	Layers (L)	ρ	Vocab Size	max Leaf $(2^{L-1} \times T)$	Train time (s)	Test time (s)
5	10	5	2331	2560	313	144
10	10	10	3171	5120	1456	321
3	14	5	8814	24576	301	126
25	8	10	3171	3200	2384	153
25	15	5	94824	409600	1840	1343

Figures 14 and 15 show the impact of the number of trees, layers and ρ on the classification accuracy for the RF codebook 5/10/5 and 10/10/10 respectively. **After various trials, the best accuracy is obtained for a RF classification with parameters: 30 trees, nine layers and $\rho = 10$ in both cases.** However, the 5/10/5 RF trains in $91.0s$ compared to 175.4 for the 10/10/10 RF. **Hence, the 5/10/5 configuration is chosen, giving an accuracy of 55% in 80s (test and training time for RF classification).** The corresponding confusion matrix is plotted in Figure 16. Even though, the global accuracy is relatively low compared to the k-means codebook, some categories are best predicted with this classifier (trilobite has 100% accuracy, and wrench has 82%).

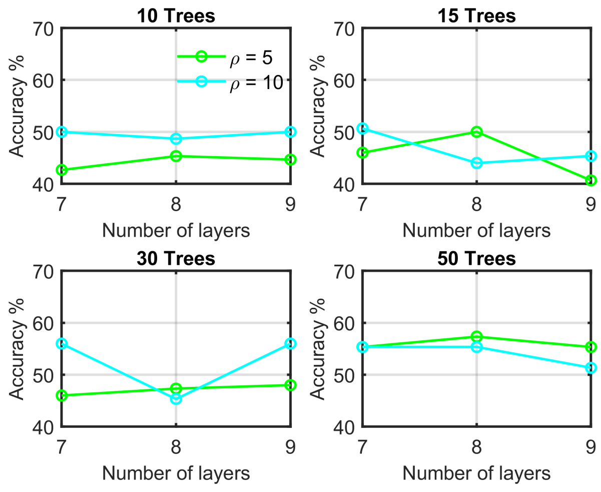


Figure 14. RF classification configurations for the RF codebook configuration: five trees, 10 layers, $\rho = 5$.

Finding an optimal value for the RF codebook and RF Classifier turned out to be a more complicated task than when using the k-means codebook. The long computational time (to the order of 15 minutes for the codebook and sometimes up to 5 minutes for each tree configuration) made it hard to collect various configurations for each. From the theory presented in the lecture course (see MLCV_rf_codebook_latest slide 15), this method should be faster and more accurate than the one using a k-means codebook due to its reduced complexity. However, in the present case, the results computed in question 3 - part 2 are more accurate and faster to obtain. These results highlight an optimization issue in the code developed and implemented for the coursework. This is likely to be due to the fact that a "fast training" script was not implemented in which all the test data is sent through the trees at once for classification.

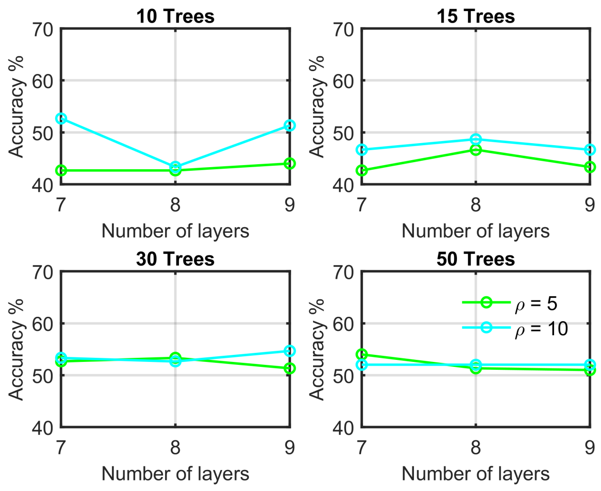


Figure 15. RF classification configurations for the RF codebook configuration: 10 trees, 10 layers, $\rho = 10$.

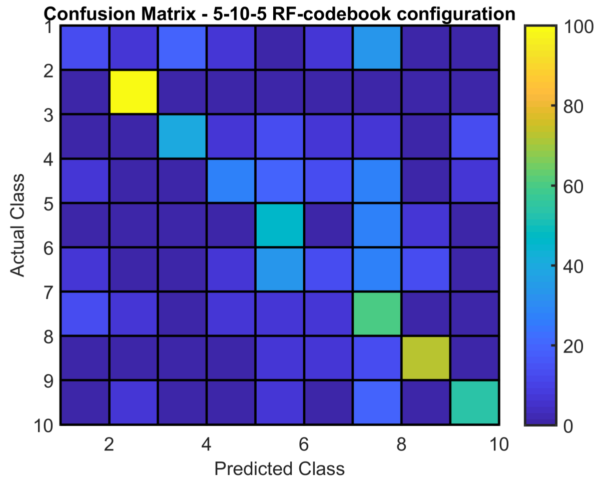


Figure 16. Confusion matrix for the RF classification using a k-means codebook. RF configuration: 30 trees, 9 layers, $\rho = 10$, vocabulary size determined by a RF codebook with configuration: five trees, 10 layers and $\rho = 5$.

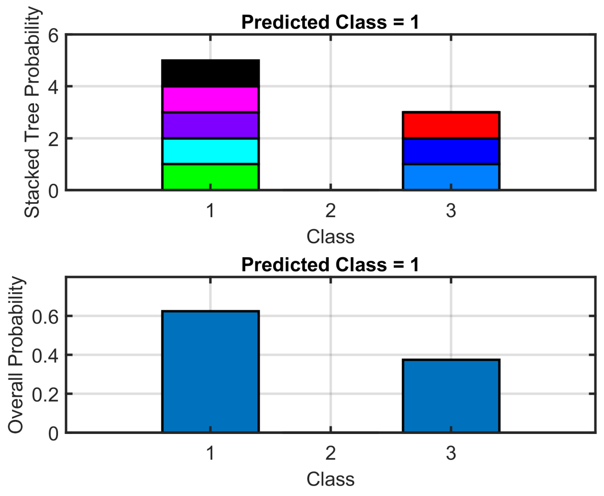
Conclusions

In conclusion, in questions one and two the RF algorithm builds a RF which successfully classifies the toy data. In question three, the k-means algorithm for creating a codebook which performs both faster and more accurately compared with the RF codebook. These results contradict the expectation of the authors and the lecture notes since the RF codebook should have a lower complexity and as such should run faster, if not more accurately. It is likely that this slow running time is due to a lack of a fast testing function. Nonetheless, while the RF codebook has a lower overall accuracy, individual classifications are at times much more accurate as can be seen by comparing the confusion matrices resulting form the two approaches. Finally, it is noted that the performance of the two codebooks is not directly comparable since the RF parameters used for classification differ. However, their performance with their respective optimal RF classification tree parameters is comparable from the findings in this report.

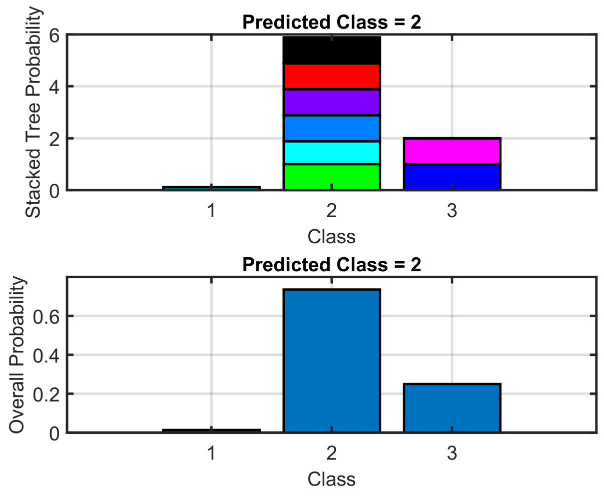
1 Appendix

1.1 Figures

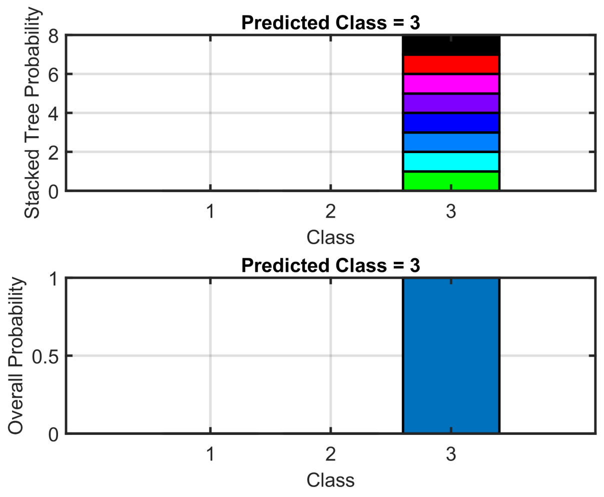
1.1.1 Q2: Class distribution plots



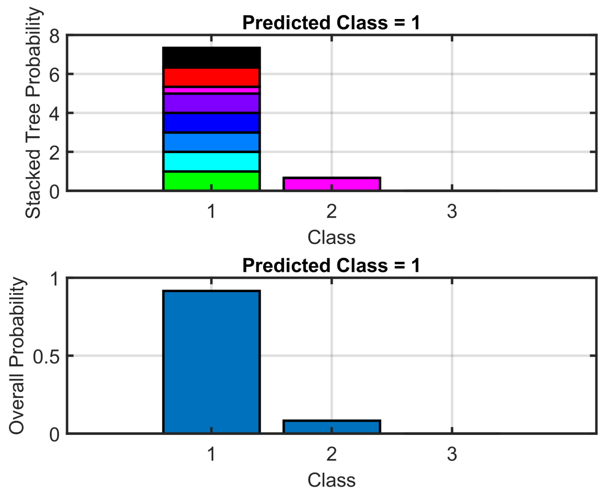
(a) Point A: (-0.7, -0.5)



(b) Point B: (0.4, 0.3)



(c) Point C: (-0.7, 0.4)



(d) Point D: (0.5, -0.5)

Figure 17. Visualisation of the class probability of four sample points. The probability contribution from the leaf node in each randomised tree (top) the normalised probability that the data point belongs to each class (bottom).

1.1.2 ANOVA Test Results

```
Source          SS       df      MS         F     Prob>F
-------------------------------------------------------
Columns   285397.6     49    5824.44    1.18    0.2427
Rows         253.9      2     126.96    0.03    0.9746
Error     483981.4     98    4938.59
Total     769632.9    149
```

(a) ANOVA Test results for the histograms of images from different categories.

```
Source          SS       df      MS         F     Prob>F
-------------------------------------------------------
Columns   423424.3     49    8641.31    3.31    0
Rows        2707.9      2    1353.93    0.52    0.5967
Error     255614.8     98    2608.31
Total     681747      149
```

(b) ANOVA Test results for the histograms of images from the same category.

Figure 18. ANOVA test tabular results.

1.1.3 Q3 - Part 2 : Confusion Matrix

Table 2. Confusion Matrix obtained for a random forest of 25 trees,eight layers and $\rho = 10$

Predicted labels

	Tick	Trilobite	Umbrella	Watch	Water Lilly	Wheelchair	Wild Cat	Chair	Wrench	Yin Yang
Tick	46.7	0	0	6.7	0	20	6.6	20	0	0
Trilobite	0	86.7	6.7	0	0	0	0	0	0	6.6
Umbrella	0	0	26.7	13.3	26.7	0	13.3	13.3	0	6.7
Watch	13.3	0	0	40	26.7	6.7	0	0	13.3	0
Water Lilly	0	0	0	0	86.7	6.7	0	0	13.3	0
Wheelchair		0	0	0	20	73.3	0	6.7	0	0
Wild cat	0	6.7	6.7	6.7	6.7	20	33.3	6.7	13.3	0
Chair	6.7	0	0	6.7	0	6.7	0	80	0	0
Wrench	0	0	6.7	6.7	0	0	6.7	13.3	53.3	13.3
Yin yang	13.3	0	6.7	0	0	0	6.7	6.7	0	66.6

1.2 Code

1.2.1 Q2: MainScript

```
clear all
close all

% Get training and test data
[data_train, data_test] = getData('Toy_Spiral');

%% Setting the parameter of the tree
param.s = size(data_train,1)*(1 - 1/exp(1)); %size of bags s
param.replacement = 1; % 0 for no replacement and 1 for replacement

%% Training Tree

AccTot = [];

% Optional loop for a parameter sweep of the number of bags (i.e. number of
% trees)
for n = 8
    param.n = n;
    % perform bagging with replacement using the training data
    [bags] = bagging(param, data_train);
    % Optional loop for a parameter sweep of the number of levels in each tree
    for numlevels = 8
        param.numlevels = numlevels;
        % Optional loop for a parameter sweep of rho
        for numfunct = 10
            param.numfunct = numfunct;

            tic
            %Train forest using bags and defined parameters
            [leaves, nodes] = trainForest(bags, param);
            param.trainingtime = toc;

            % calculate the accuracy of the
            Acc = [param.n, param.numlevels, param.numfunct, accuracy(param, data_train,
                leaves, nodes)];
            AccTot = [AccTot; Acc];
            clear Acc

            % Test Tree for the 2D grid or for the sample points.
            [classPred] = testForest(param, data_test, leaves, nodes, 0, 0);
            clear leaves
            clear nodes
        end
    end
    clear bags
end
```

1.2.2 Q3 - Part 1 : getData.m function

```
function [data_train, data_query] = getData( MODE )
% Generate training and testing data

% Data Options:
%    1. Toy_Gaussian
%    2. Toy_Spiral
%    3. Toy_Circle
%    4. Caltech 101

showImg = 0; % Show training & testing images and their image feature vector (histogram
    representation)

PHOW_Sizes = [4 8 10]; % Multi-resolution, these values determine the scale of each layer.
PHOW_Step = 8; % The lower the denser. Select from {2,4,8,16}

switch MODE
    case 'Toy_Gaussian' % Gaussian distributed 2D points
        ...

    case 'Toy_Spiral' % Spiral (from Karpathy's matlab toolbox)
        ...

    case 'Toy_Circle' % Circle
        ...


    case 'Caltech' % Caltech dataset
        ...

        % K-means clustering
        numBins = 256; % for instance,
        Centroids = vl_kmeans(desc_sel,numBins); % Generate 256 codewords

        disp('Encoding Images...')
        % Vector Quantisation
        Centroids = Centroids';

        for i=1:length(classList)
            for j=1:imgSel(1)
                image = desc_tr{i,j}';
                idx = knnsearch(Centroids, image);
                data_train((i-1)*15+j,:) = [hist(idx, numBins), i];
            end
        end

        fname = ['data_train_' num2str(numBins)];
        save(fname, 'data_train');

        % Clear unused varibles to save memory
        clearvars desc_tr desc_sel
```

```
end

switch MODE
    case 'Caltech'
        ...

        % Vector Quantisation

        for i=1:length(classList)
            for j=1:imgSel(1)
                image = desc_te{i,j}';
                idx = knnsearch(Centroids, image);
                data_query((i-1)*15+j,:) = [hist(idx, numBins), i];
            end
        end

        fname = ['data_test_' num2str(numBins)];
        save(fname, 'data_query');
    otherwise % Dense point for 2D toy data
        ...
end
end
```

1.2.3 Q3 - MainScript

```
clear all
close all
clc

% Load training and testing data saved from the getData_RF.m function
load data_train_256.mat
load data_test_256.mat

%% Setting the parameter of the tree
param.s = size(data_train,1)*(1 - 1/exp(1)); %size of bags s
param.replacement = 1; % 0 for no replacement and 1 for replacement

%% Training Tree
AccTot = [];

%Find the size of the training data (i.e. size of vocabulary)
param.dimensions = size(data_train,2)-1;

% Optional parameter sweep loops
for n = 25
    param.n = n;
    [bags] = bagging(param, data_train);
    for numlevels = 8
        param.numlevels = numlevels;
        for numfunct = 15
            param.numfunct= numfunct;

            %Train the forest using the bags and parameters defined
            tic
            [leaves, nodes] = trainForest3(bags, param);
            param.trainingtime = toc;

            % Test the trees and evaluate the accuracy of the classification of
            % the test images.
            [classPred] = testForest3(param, data_test, leaves, nodes, 0, 0);
            Acc = [param.n, param.numlevels, param.numfunct,accuracy(param, data_test,
                classPred)];
            AccTot = [AccTot; Acc];
            clear Acc

           % Calculate the confussion matrix for the image categories
            [Conf, order] = confusionmat(data_test(:,param.dimensions+1),classPred(:,1));
            Conf = 100/15.*Conf;

            clear leaves
            clear nodes
        end
    end
    clear bags
end
```

1.2.4 Q3 - Part 3 : getDataRF.m function

```
function [data_train, data_query ] = getData( MODE )
% Generate training and testing data

% Data Options:
%   1. Toy_Gaussian
%   2. Toy_Spiral
%   3. Toy_Circle
%   4. Caltech 101

showImg = 0; % Show training & testing images and their image feature vector (histogram
    representation)

PHOW_Sizes = [4 8 10]; % Multi-resolution, these values determine the scale of each layer.
PHOW_Step = 8; % The lower the denser. Select from {2,4,8,16}

switch MODE
    case 'Toy_Gaussian' % Gaussian distributed 2D points
        ...

    case 'Toy_Spiral' % Spiral (from Karpathy's matlab toolbox)
        ...

    case 'Toy_Circle' % Circle
        ...


    case 'Caltech' % Caltech dataset
        ...

       % Build visual vocabulary (codebook) for 'Bag-of-Words method'
        for i = 1:length(classList)
            for j = 1:imgSel(1)
                labels = i.*ones(1,size(desc_tr{i,j},2));
                tree_tr{i,j} = [desc_tr{i,j}; labels];
            end
        end
        desc_sel = single(vl_colsubset(cat(2,tree_tr{:}), 10e4)); % Randomly select 100k
            SIFT descriptors for clustering
        desc_sel = desc_sel';

        % build RF Codebook
        tic
        param.n = 5;
        param.numlevels = 10;
        param.numfunct = 5;

        [leaves nodes] = buildCodebook(desc_sel, param);

        for i=1:length(classList)
            for j=1:imgSel(1)
                image = single(desc_tr{i,j})';
                predictedLeaf = testCodebook(param, image, leaves, nodes);
                data_train((i-1)*15+j,:) = [predictedLeaf, i];
                clear predictedLeaf
                clear image
            end
        end

        fname = ['data_train_' num2str(param.n) '_' num2str(param.numlevels) '_' num2str(
            param.numfunct)];
        save(fname, 'data_train');

        % Clear unused variables to save memory
        clearvars desc_tr desc_sel
end

switch MODE
    case 'Caltech'
        ...

        % Quantisation
        for i=1:length(classList)
            for j=1:imgSel(1)
                image = single(desc_te{i,j})';
                predictedLeaf = testCodebook(param, image, leaves, nodes);
                data_query((i-1)*15+j,:) = [predictedLeaf, i];
                clear image
                clear predictedLeaf
            end
        end

        fname = ['data_test_' num2str(param.n) '_' num2str(param.numlevels) '_' num2str(
            param.numfunct)];
        save(fname, 'data_query');

    otherwise % Dense point for 2D toy data
        ...
end
end
```