Machine Learning For Computer Vision: Random Decision Forest

Sagar Patel, CID: 00842688, and Pranav Malhotra, CID: 00823617

14th March 2017

1 Train Decision Forest

1.1 Bootstrap Aggregating

Using Bootstrap Aggregating (Bagging), multiple data subsets are generated by uniformly selecting data points from the training set with replacement. This introduces some randomness into the training data used to grow each tree in the random decision forest. The in-built matlab function `randsample` is used to generate each dataset.

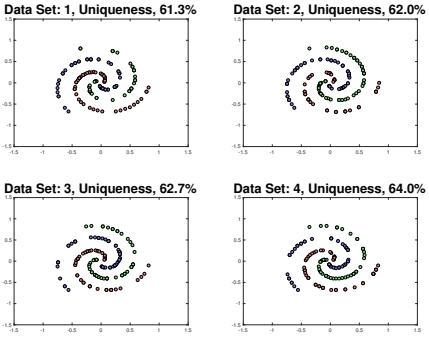

Figure 1: Random dataset generation using Bagging

The resulting data subsets have been visualized in Figure 1. Each data subset has 150 points; however the spirals are not complete. This is due to the fact that samples are drawn from the training dataset with replacement. The amount of uniqueness within each subset is presented in Figure 1 as well. Ideally, as the size of our training dataset increases, the uniqueness should approach 63.3%. The finite size of our training dataset results in uniqueness values spread around the ideal. The table below shows the probability distributions of each class within each data subset. The roughly equal distributions corroborate the fact that sampling was performed uniformly.

Dataset Index	Class Probabilities
1	$[0.3733, 0.2733, 0.3533]$
2	$[0.2800, 0.3667, 0.3533]$
3	$[0.3333, 0.3200, 0.3467]$
4	$[0.3467, 0.3133, 0.3400]$

1.2 Split Function

The first step in the training of the decision forest is to develop the weak-learner functions that will be used to split the data at each node of the tree. Multiple weak-learner functions were implemented, namely axis-aligned, two-pixel, linear, quadratic and cubic.

Note that the axis-aligned, two-pixel and linear learners have the general form presented in the lecture notes whereas the quadratic and cubic split functions have the form expressed in (1) and (2) respectively. The quadratic and cubic learners are non-linear in 2-dimensional space but are implemented by transforming the data and using linear separators.

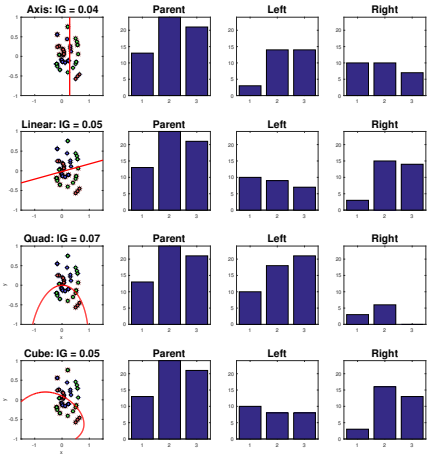

Figure 2: Weak-Learners Trained on Large Number of Points, small value of `numSplit`

Example split functions for all learners, except the two-pixel test, are presented in Figure 2. The two-pixel test does not perform well with 2-dimensional data. The results are indicative of the fact that a stronger class of learner functions does not necessarily translate into a greater information gain; this is especially true if the number of split functions over which optimization is performed is small.

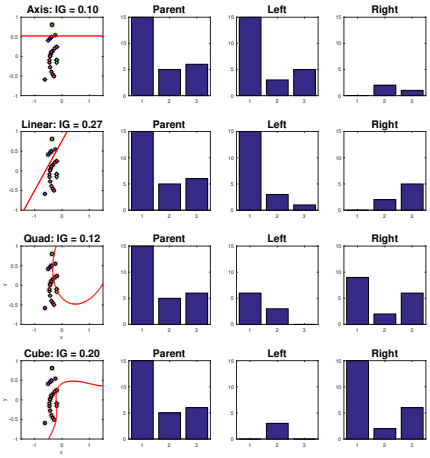

Figure 3: Weak-Learners Trained on Small Number of Points

The graph in Figure 3 shows what happens when a strong learning class is trained on a small number of data points. Observe the split function obtained for the quadratic and cubic learners; **there is clear over-fitting of the data.** This is completely consistent with the results obtained from Vapnik–Chervonenkis (VC) theory. Quadratic and cubic functions have a larger VC dimension and if trained on a small number

of points, these classes overfit training data and do not generalize well. The random forest algorithm provides a mechanism, the committee machine, to combat over-fitting in each individual tree. As such, in the next section, it should be noted that if strong non-linear learners are used, the number of trees should also be increased to achieve the same performance as a weak-learner.

By simply analyzing Figure 2 and 3, the advantages of using a stronger learning class are not clear. For this, we have to increase `param.splitNum`. Figure 4 makes it clear that stronger learning classes are able to split the data more efficiently and obtain a higher information gain. The discriminating power of the learning classes is not obvious when the number of split functions tested is small; however increasing `param.splitNum` clearly highlights the intrinsic discriminating power of each weak-learner.

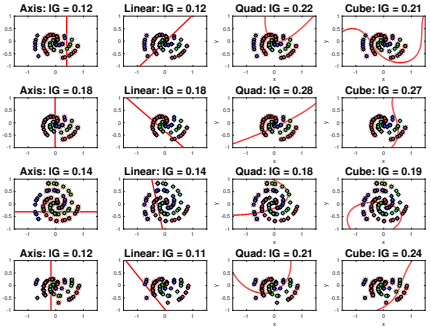

Figure 4: Weak-Learners with large value of `numSplit`

Using a stronger learning class such as the quadratic or cubic non-linear functions has it benefits. They can increase the information gain at each node. However, when increasing the strength of the learning class one should also increase the number of split functions tested.

1.3 Growing Tree

The leafs visualized in the figure below were generated by growing a tree that utilizes axis-aligned weak-learners. Most leaves contain only one class of data points; the recursive splitting of the data points down the tree has a distilling effect. There are some leaves with 2 or more classes of data. This can be attributed to the difficulty in isolating classes near the origin where the training data is tightly clustered.

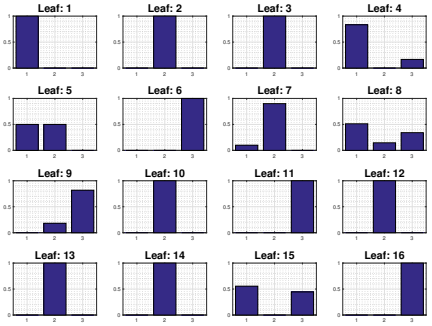

Figure 5: Visualization of Leaf Nodes

The tree was built using two stopping criteria, used in conjunction. Firstly, a node is considered a leaf if less than 5 training points reach it. If only 5 points arrive at a node, they will be concentrated within a small region. Training based on these points will lead to over-fitting. The second stopping criteria is the maximum tree depth. This is simple to implement and gives us direct control on the maximum number of leaves. Using the tree-depth to control the number of leaves will come in handy when growing an Random Forest codebook.

2 Evaluating Decision Forest

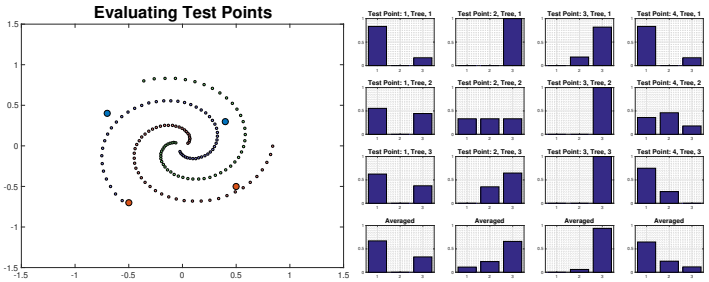

Figure 6: Evaluation of Test Points

The 4 test points are evaluated on a forest containing 3 trees; the results obtained are visualized above. The histograms show the probability distribution functions at each leaf for each point. The averaged values have also been plotted. Note that test point 1 (-0.5, -0.7), has been misclassified; ideally, the point should be blue. The reasons for this misclassification are multi-fold. One of the main factors is that the random forest only has 3 trees and thus the averaging effect of the committee machine is limited.

2.1 Varying Number of Trees

To directly combat the misclassification observed above, the first parameter that is varied is the number of trees in the forest. The graph below shows that increasing the number of trees has a considerable effect on the classification accuracy. This is especially true when the absolute number of trees is small. Increasing the number of trees from 100 to 200 has negligible effect. Notice the straight axis-aligned class boundaries obtained in the region of the grid for which extrapolation is required. Training data is contained within the unit square whereas the region for which the classifier is tested extends beyond the unit square. The classifier is expected to extrapolate the class boundaries. Since the axis-aligned weak-learner is used, straight axis-aligned lines are observed in the extrapolated region.

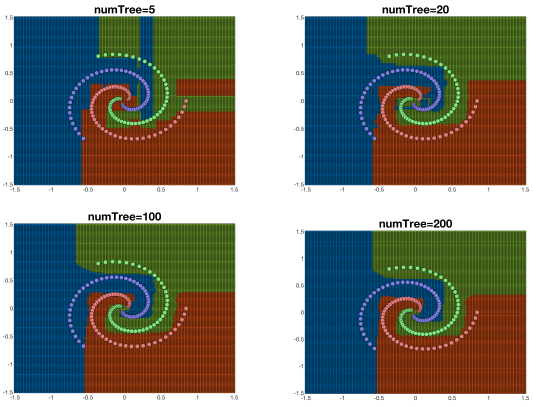

Figure 7: Effect of Changing Number of Trees

Increasing the number of trees is not without a cost. The random forest algorithm scales linearly with the number of trees in the forest. The algorithm has the added benefit that it can be parallelised. Each tree can be grown and tested in parallel.

2.2 Varying Number of Split Functions

Next, we vary the number of split functions while size of the forest is limited to 10. By increasing the number of split functions that we try at each node, we are decreasing the inherent randomness within each tree. If all trees are assumed to be completely uncorrelated, then $E_{com} = {}^{E_{av}}/_{T}$; where E_{com} is the error incurred by the random forest and E_{av} is the average error of each tree. If the trees are correlated, then $E_{com} \leq {}^{E_{av}}/_{T}$. Even with correlated trees, it is possible that the committee machine produces good results if E_{av} is low. An optimal balance point has to be found; in order to achieve a low E_{av} some correlation between trees is tolerated.

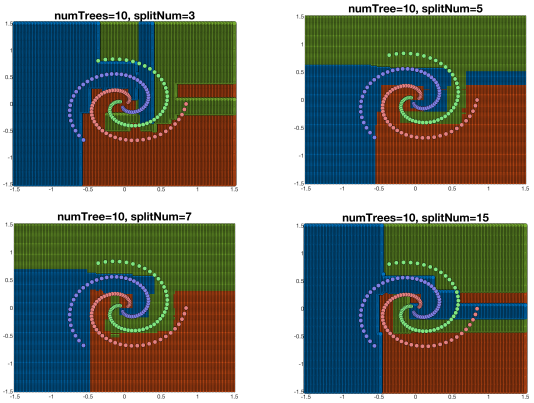

Figure 8: Effect of Changing `numSplit`

Observe above that with a small number of split numbers, artifacts are observed. Each tree is a very bad learner and thus, even though $E_{com} \approx {}^{E_{av}}/_{T}$, E_{com} is still rather high. Upon increasing the number of split functions to 7, the random forest clearly tries its best to map the spiral nature of the training data using axis-aligned weak-learners. Upon setting `param.splitNum` to 15, it can be seen that not only is each tree highly over-fitted, each tree is also highly correlated and thus with only 10 trees, the committee machine cannot sufficiently average away the over-fitting artifacts.

2.3 Varying Tree Depth

The third parameter that can be varied is the depth of the tree. More accurate results are produced when tree depth is increased. With depth set at 3 or 5, there are very obvious artifacts that appear; a thin red line when depth is set at 3, and a massive blue region when the depth is set at 5. These are clearly anomalies that have come about because the tree does not sufficiently refine the training data. Upon increasing the depth to 7, the obvious anomalies disappear but small artifacts still remain at the center of the graph. Results are further improved if the tree depth is increased to 9.

Tree depth should not be increased arbitrarily for two reasons. As we descend down the tree, training points become refined and get clustered together. If we arbitrarily keep increasing the depth, we will be attempting to separate points which lie very close to one another. Each node's weak-learner has an effect over the entire range of values for which the algorithm is used. Observe what happens when we train on very few number of points in Figure 3.

Secondly, computational complexity scales exponentially with tree depth. It is much wiser to average out anomalies by increasing the number of trees; while trees can be trained and tested in parallel, each individual tree must be grown sequentially.

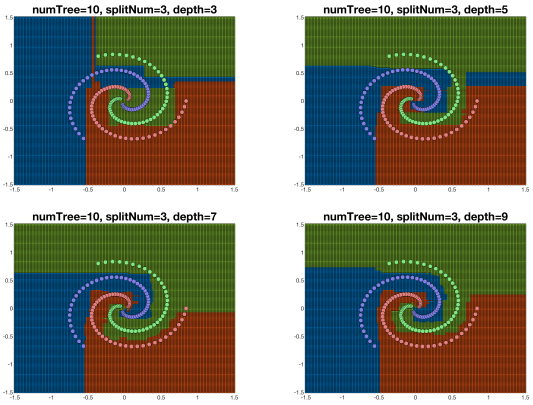

Figure 9: Effect of Varying Tree Depth

2.4 Testing with Different Weak-Learners

The entirety of the discussion above has been made with respect to the axis-aligned weak-learner. The ideal values of random forest parameters very much depend on the class of weak-learner used. The graphs below show the effects of using different weak-learners. The class boundaries in the extrapolated region follow the general shape of the learner class used. For reference, an random forest trained using the axis-aligned weak-learner with all 3 parameters turned has also been graphed.

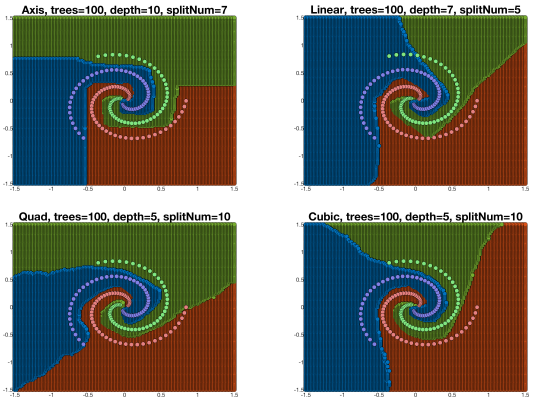

Figure 10: Classification Boundaries obtained Using Different Weak-Learners

3 K-means Codebook

3.1 Vocabulary Size

Using the K-means algorithm to form a Bag-of-Words (BoW) representation, we have direct control over vocabulary size. The vocabulary is going to consist of exactly K codewords.

The BoW can be interpreted as a dimensionality reduction technique that shrinks an image into a histogram which indicates the frequency of occurrence of each codewords in our vocabulary. Note that classification is performed using only the histogram and the physical meaning of each codeword is not relevant to the random forest classifier. In fact, more often than not, codewodes do not match heurestic features such as eyes or wheels. With that in mind, it should be noted that the performance of the classifier is heavily dependent on richness of the vocabulary. To form the K-means codebook, matlab's inbuilt function `kmeans` is used; matlab implements the K-means++ algorithm[1], which exhibits a complexity of $\mathcal{O}(\log K)$.

3.2 Vector Quantisation Process

The first step in the vector quantisation process involves extracting Scale Invariant Feature Transform (SIFT) descriptors for each image in our training dataset. Extraction is performed using the function `vl_phow`, which comes with the reference code provided for this coursework. The function outputs 128-dimensional descriptors; note that the number of descriptors returned depends on the image. After obtaining descriptors for each image in the training dataset, 100,000 descriptors are randomly selected to form a intermediate dataset in a 128-dimensional vector space. This intermediate dataset is fed into the K-means algorithm; only 100,000 descriptors are used to place reasonable bounds the computational complexity both in terms of time and memory. The K-means algorithm will return K cluster centers, where K represents the size of our codebook.

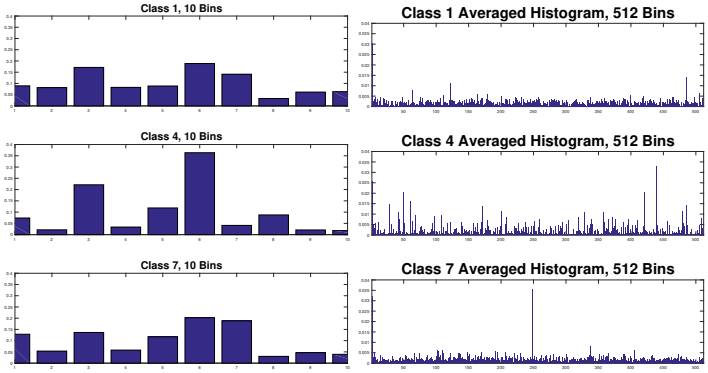

Figure 11: Averaged Class Histograms for Classes 1, 4 and 7

The final step of the vector quantisation process is to build a BoW histogram for each image in the training dataset; for this, all of the descriptors returned by `vl_phow` for a single image are compared to the cluster centers using matlab's inbuilt function `knnsearch`. The euclidean distance between each cluster center and each descriptor is found. The center which is closest to the descriptor in the 128-dimensional descriptor space is returned by `knnsearch`. Occurrences of each cluster center are accumulated and normalized to form a histogram. The figure below shows the averaged histograms for classes 1, 4 and 7 with the two different vocabulary sizes.

4 RF Classifier

The RF classifier is trained using a histograms obtained from a codebook, consisting of 256 codewords; performance is then tested. Example successes and failures are visualized below. Heuristic explanations can be made for some of the misclassifications that are observed. For example, the left-most watch has the characteristic 'spoty' phase transition and circular edges that are observed in wild cats. All misclassifications however cannot be explained heuristically. Two water lilies, which to the human eye are almost identical, were classified very differently.

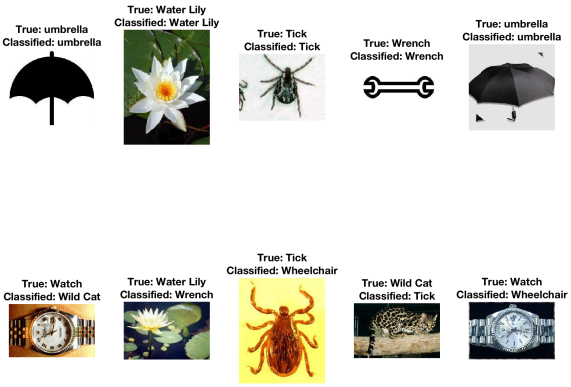

Figure 12: Example Successes and Failures Visualization

In general, instead of looking at individual successes and failure, the results can be summarized into a confusion matrix. The confusion matrix below shows that trilobites and windsor chair were always correctly classified, whereas the forest is the worst at identifying watches.

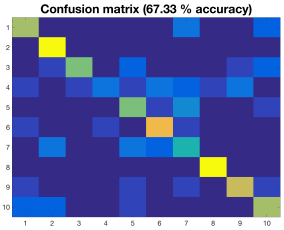

Figure 13: Confusion Matrix Visualization

Vocabulary Size

Using a small vocabulary, the dimension of the space in which the RF classifier acts is small. The classes will be harder to separate in a low dimensional space. Both the numerical results in Figure 14 and the averaged histograms in Figure 11 concretely prove that increasing the number of bins results in improved classification accuracy.

Notice however that the performance increase obtained by increasing the vocabulary size from 512 to 1024 is marginal. In fact, performance drops when an extremely large number of bins are used in conjunction with a small number of trees in the RF classifier. Although 1024 codewords gives us greater discriminating power, if the RF classifier parameters are not tuned to take into account the increase in the vocabulary size, E_{av} will be high. The high error that each tree incurs can either by averaged away using the committee machine, or by increasing depth and the number of split functions tried at

each node, we can lower E_{av}. Again, the increase in discriminating power that is associated with an increase in the number of codewords does not carry on infinitely. If the ratio of cluster centers to training points is too high, each codewords will start to lose meaning.

As mentioned above, the K-means++ algorithm that matlab implements scales with $\mathcal{O}(\log \mathrm{K})$.

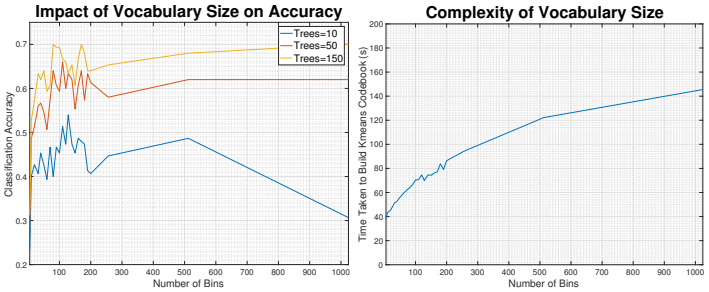

Figure 14: Effect of Increasing Vocabulary Size on Accuracy and Computational Time

Number of Trees

The graphs below corroborate the discussion above about the effects that `numTrees` has. Regardless of the vocabulary size, increasing the number of trees yields an increase in performance. However, setting `numTrees` to be greater that 180 does not lead to any significant improvements in the recognition accuracy. Also, we observe that increasing the number of trees causes computational complexity to grow linearly. Note that, no parallel processing has been done as of yet; the algorithm is susceptible to parallelism.

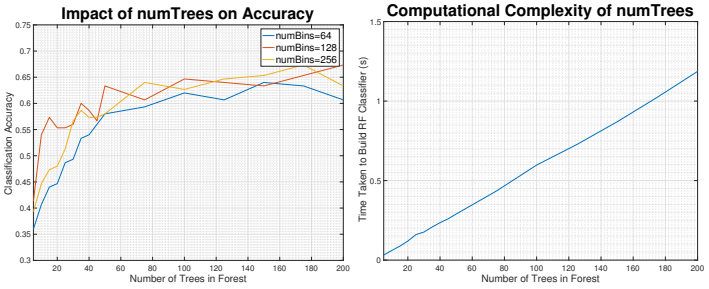

Figure 15: Effect of Changing Number of Trees

Number of Split Functions

Again, the graphs below corroborate the discussion above about the effects of increasing the number of split functions. We observed visually that setting `numSplits` to be small lead to artifacts appearing the Toy Spiral data. These artifacts are analogous to the poor recognition accuracy obtained in the figure below. We also observed that by setting `numSplits` to be extremely large, overfitting and tree interdependence would set in. Again, this is analagous to a reduction in performance after a certain point. Although the general patterns are similar, notice how different the scales are. For performance to drop, we need to set the number of split functions tested at each node to be 150. The reason for this is that instead of separating points in a 2-dimensional space as in the case of the Toy Spiral data, we are now separating points in much higher dimensions. In fact, the graphs below are obtained using a vocabulary consisting of 256 codewords. As such, to overfit the data and correlate the trees in such a large dimension, we need to try a lot of split functions.

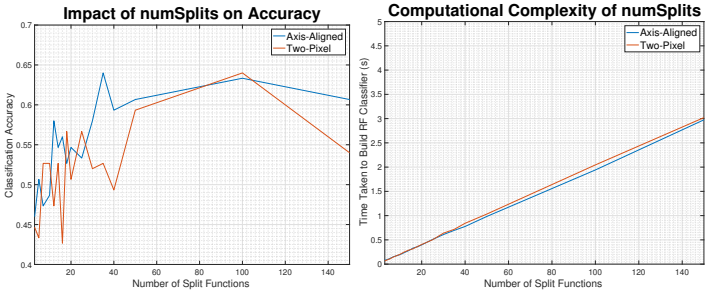

Figure 16: Effect of Varying `numSplit`

Depth of Tree

Lastly, the impact of tree depth is studied. The most important thing to notice from the graphs is how poorly the algorithm scales in terms of complexity. Note that the graph shows the average time taken to grow each tree in the forest. This is expected since the number of nodes in our tree grows exponentially with tree depth. For relative good performance, we would need at least 150 trees in our forest. If each tree took 10 seconds to grow, simply training the forest would take approximately 25 minutes. As mentioned above, increasing tree depth is not that wise and other parameters should be tuned to increase performance. Anyway, performance does not always increase with an increase in tree depth. Notice that for the axis-aligned split function, the peak performance occurs at a depth of 8.

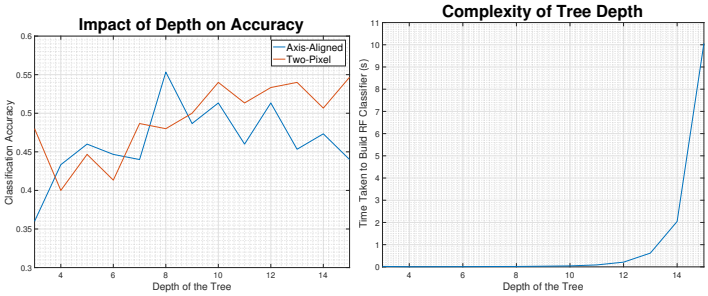

Figure 17: Effect of Increasing Tree Depth

5 RF Codebook

5.1 Overview of Codebook

The previous section explored codebook generation using the K-means algorithm. The K-means algorithm has the advantage that it considers all 128-dimensions of the descriptor space when forming clusters. This is extremely important since the features returned by `vl_phow` are Gaussian derivatives computed at 8 orientation planes over a 4×4 grid. Using a single decision tree to generate a codebook will never return codewords with as much discriminating power as those returned by the K-means algorithm. Decision trees use weak-learners that consider a limited number of dimensions at each node. The component-wise manner in which data is filtered can never expect to produce the same results as the K-means algorithm which considers all dimensions simultaneously. However, by using an ensemble of decision trees, one

can still achieve a rich and diverse codebook. RF codebooks will be analyzed in this section.

5.2 Performance of RF Codebook

The performance of the RF codebook is extremely difficult to test. The K-means algorithm only takes as input, the vocabulary size. As such, since there is only one parameter, the performance can be easily studied. We noted that increasing the vocabulary size for the K-means algorithm, generally improved the performance; oberve Figure 14, there is generally an upward trend regardless especially if the number of trees in the classifer is high. This is not the case for the K-means algorithm. The performance is highly sporadic.

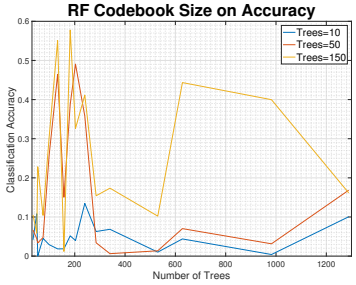

Figure 18: Performance of RF Codebook

The main reason for this is there are 2 main ways to increase the size of the RF codebook. One can increase the size of the codebook by increasing the number of trees or by increasing the tree depth. In the results presented above, codebook size was increased by simply increasing the number of trees. One possible method of selecting the multiple tuning parameters of the RF codebook is to perform cross-validation. Note that using the correct tuning parameters, it is possible to obtain performance that is similar to that of the K-means codebook.

5.3 Computational Complexity

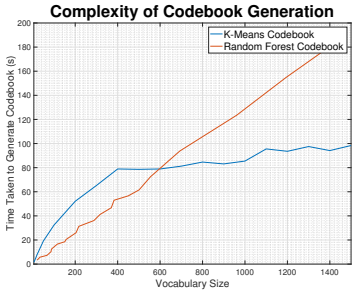

Figure 19: Comparison of Computational Complexity

In the lecture notes, the computational complexity of the K-means algorithm is $\mathcal{O}(\mathrm{DN'K})$ whereas the complexity of the RF codebook is $\mathcal{O}(\sqrt{\mathrm{D}}\mathrm{N'log\ K})$. In the equations, D represents the dimension of the descriptor vectors, $\mathrm{N'}$ represents the number of descriptor vectors and K represents the size of the codebook. The log K complexity with respect to the size of the codebook is the main advantage of using a RF codebook. With this in mind, the computation complexity of both codebook generation algorithms is tested. The results obtained are visualized in the figure above.

Note that the graph obtained for the K-means algorithm is completely expected. As discussed above, matlab implements the K-means++ algorithm which scales with logarithmically with K. The graph obtained for the RF codebook is not scaling according to the bound presented in the lecture notes. There could be multiple reasons for this. One of the many factors causing the random forest to slow down is the number of split functions that are tried at each node. Regardless, it is important to note that the RF codebook admits a parallel implementation whereas the K-Means algorithm cannot be parallelised.

6 Conclusion

All in all, this report explores the Random Decision Forests and the explores the effect that the multiple tuning parameters have on the algorithm's accuracy. To start,multiple weak-learners were explored in section 1. The discriminating power of different learning classes as well as the relation to the number of split functions over which optimization is performed was studied. With the 2-dimensional Toy Spiral dataset, artifacts due to underfitting and overfitting were visualized in section 2. Next, codebook generation using the K-means algorithm was studied. The process of generating an intermediate descriptor vector space using `vl_phow` was briefly discussed. Next, the effect of RF tuning parameters was analyzed analytically using the CalTech 101 dataset. Conclusions such as increasing the number of trees in the forest eventually leads to a plateau and the depth of the tree causes a exponential increase in complexity were drawn and substantiated with empirical evidence. Lastly, the RF codebook was analyzed. The difficulty in generating a rich and diverse codebook due to the influence of many tune parameters was learnt. Lastly, computation complexity of the two codebook generation algorithms were studied.

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Appendix A: Equations for Non-Linear Weak-Leaners

$$h(\mathbf{v}, \theta) = [a_1 x_1^2 + a_2 x_2^2 + a_3 x_1 x_2 + a_4 x_1 + a_5 x_2 > \tau] \tag{1}$$

$$h(\mathbf{v}, \theta) = [a_1 x_1^3 + a_2 x_2^3 + a_3 x_1^2 x_2 + a_4 x_1 x_2^2 + a_5 x_1^2 + a_6 x_2^2 + a_7 x_1 x_2 + a_8 x_1 + a_9 x_2 > \tau] \tag{2}$$