Random Forest Ensemble Visualization

Ken Lau*
University of British Columbia

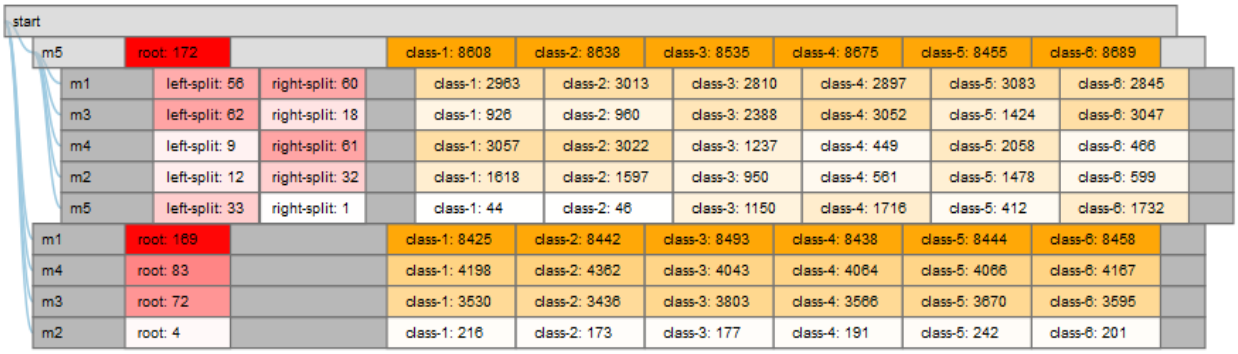

Fig. 1. Indented Tree visualization of aggregated ensemble of classification trees. The indented tree shows both the number of feature variable (red) and class prediction count distributions (orange).

Abstract—The Random forest model for machine learning has become a very popular data mining algorithm due to its high predictive accuracy as well as simiplicity in execution. The downside is that the model is difficult to interpret. The model consists of a collection of classification trees. Our proposed visualization aggregates the collection of trees based on the number of feature appearances at node positions. This derived attribute provides a means of analyzing feature interactions. By using traditional methods such as variable importance, it is not possible to determine feature interactions. In addition, we propose a method of quantifying the ensemble of trees based on correlation of class predictions.

1 Introduction

Table 1. Weather data example. The feature variables include amount of rain, humidity level, and percentage of sunshine today. The class prediction variable is the weather for tomorrow consisting of rainy, cloudy, or sunny.

Obs	Rain	Humidity	Sunshine	Weather Tomorrow
1	1mm	81%	10%	Rainy
2	2mm	85%	40%	Cloudy
3	0mm	80%	80%	Sunny
4	1mm	81%	20%	Cloudy
..	..	..	..	..

1.1 Machine Learning

A machine learning task involves finding a map between a set of feature variables and class prediction. Furthermore, the map should be able to automatically predict any new feature variable data. A simple example would be predicting the weather for tomorrow given various atmospheric features. These features include amount of precipitation, humidity level, and percentage of sunshine. The weather data example is presented in Table 1.

*e-mail: ken.lau@stat.ubc.ca

Manuscript received 31 Mar. 2014; accepted 1 Aug. 2014; date of publication xx xxx 2014; date of current version xx xxx 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

A classification tree algorithm is one example of a mapping algorithm between feature variables and class predictions. This algorithm takes the feature variables as inputs. It recursively partitions the features space to a set of regions that maximizes homogeneity of class predictions. The algorithm outputs decision thresholds as well as ordering of feature variables used in the tree. Figure 2 provides an illustration of the classification tree model to solve a machine learning task. To obtain class predictions for a new feature variable data, simply traverse down the nodes of the tree until a leaf node is reached.

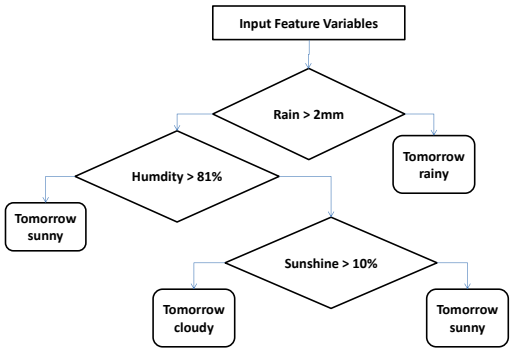

Fig. 2. Classification tree example for weather data. The interior nodes display the feature variable applied as well as threshold chosen for the split. The leaf nodes represent the class predictions

The random forest model is another example of a map between feature variables and class predictions. The random forest model is part

of a group of classifiers called ensemble methods. In fact, the model consists of a collection of classification trees. The model algorithm bootstraps on the original data set. For each random sample of the original data, a classification tree model is fit, and a prediction is made for each tree by traversing down to its leaf node. The final class prediction is the class prediction that appeared the most often [9]. The algorithm is depicted in Figure 3.

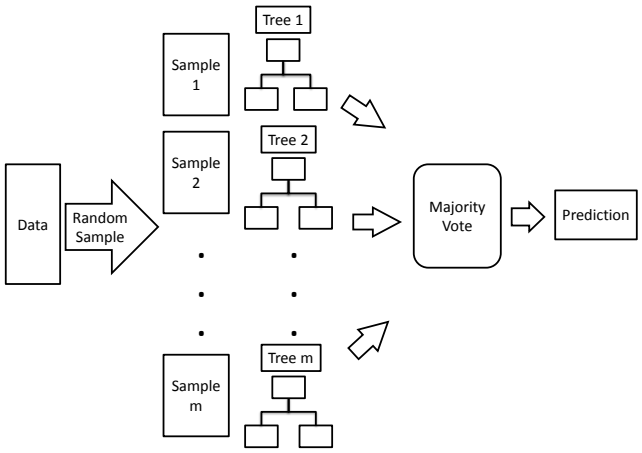

Fig. 3. The random forest model fitting process. The original data is bootstrapped, and classification trees are fit on each random sample. Given a new feature input data, prediction is made based on the majority vote of all class predictions obtained by the classification trees.

1.2 Motivation

The random forest model obtains a considerably higher accuracy prediction over the classification tree. However, interpretation of the model becomes a difficult task. It is inefficient to analyze every tree in the model. Obtaining information on feature variables applied at each node position of the trees provide useful intuition behind feature importance and feature interaction. In addition, class prediction count distribution up to nodes traversed may provide useful interpretation about the final prediction made by a random forest model. Finally, a method to quantify the diversity across the trees could allow us to compare between unrelated and related members. The inclusion of unrelated trees is one reason why the random forest work so well.

2 Related Work

One method of visualizing the entire collection of classification trees is to use 3D techniques [5]. Relevant information is first extracted and mapped to an intermediate structure. The information includes feature variable and threshold at each node position, as well as class predictions at leaf nodes. The positions of trees are spread across a 2D surface, while the depth of the tree is encoded by spatial position along the vertical axis.

Our initial idea was to use a consensus tree approach. One paper uses a partition table which summarizes the frequencies of class predictions occurring in bootstrap trees. The paper presents several methods of majority-rule consensus methods [14]. The consensus method, however, does not incorporate feature variable information at the interior nodes.

An aggregation approach called PivotGraph is relevant to our proposed visualization [10, 6]. The method derives a new network by adding up counts from two categorical attribute values. This idea is incorporated into our proposed visualization by using one categorical attribute instead of two. We treat the node positions as the categorical attribute.

A variable importance plot is a very common method of visualizing feature variable information from a random forest model. The

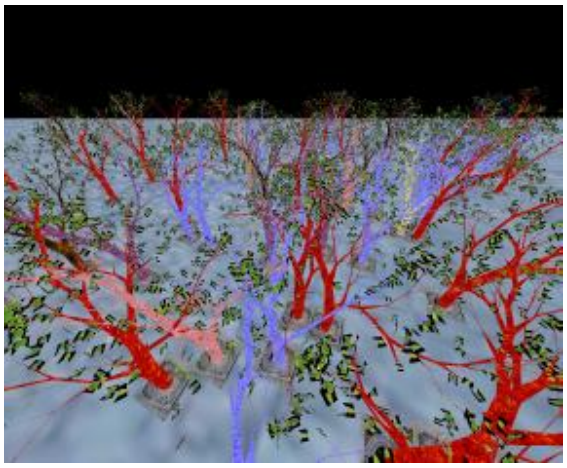

Fig. 4. Visualization of a 3D collection of trees generated by a random forest model.

importance measure for each feature in a classification tree is the information gain contributed towards maximizing homogeneity of class predictions at leaf nodes. The number of appearances of a feature variable is actually proportional to the importance measure of a feature variable. This is because the algorithm selects the feature at any specific node position based on maximal information gain. Figure 5 presents a variable importance plot based on 5 feature variables labeled from m_1 to m_5.

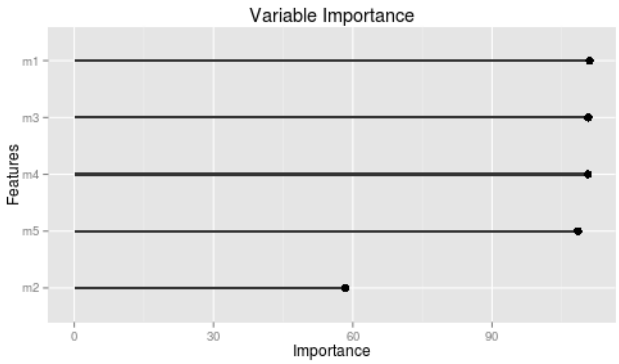

Fig. 5. Variable importance measures of 5 features using a dot plot visualization. The features are ordered by importance.

A common method of visualizing the class predictions is through a proximity measure [9] (page 595). The measure gives an indication of which observations are close together based on the class predictions made across the collection of trees. Each random sample has a set of out-of-bag observations which are excluded due to random sampling. For each tree, predictions are made based on the out-of-bag observations. The proximity measure of a pair of predictions is increased whenever the two predictions made share the same class. An *NxN* proximity matrix is accumulated. The matrix is then represented in two dimensions by multidimensional-scaling. The goal is to find a lower-dimensional representation of the data while preserving the pairwise distances in the higher-dimension as well as possible [9, 6]. Figure 6 presents a multi-dimensional scaling plot based on proximity measures on out-of-bag observation predictions.

The Curve Boxplot visualization presents a method that quantifies an ensemble of 2D and 3D curves [4]. The method uses functional band depth to derive a new attribute. The curve with the highest band depth is the most central within the ensemble, while curves with lowest

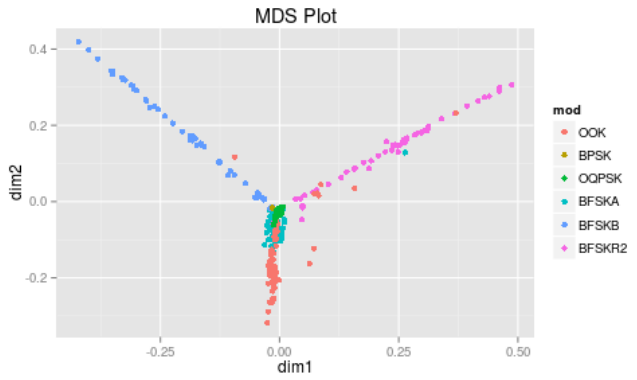

Fig. 6. Multi-dimensional scaling plot based on proximity measures across observations.

band depth values are identified as outliers. The derived attribute is displayed as a boxplot. Figure 7 presents an example of the curve boxplot visualization based on an ensemble of 50 simulated hurricane tracks. The most central curve is coloured in yellow. The 50% most central curves are coloured in dark purple.

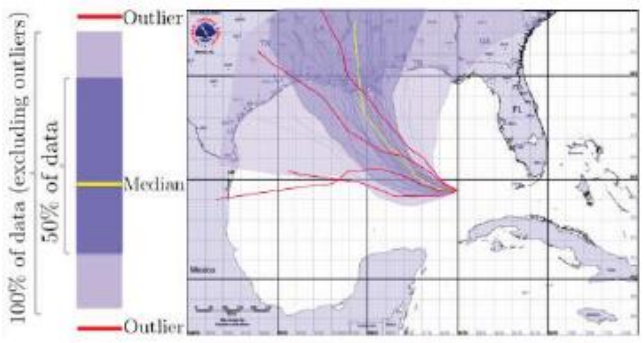

Fig. 7. Curve Boxplot visualization of an ensemble of 50 simulated hurricane tracks.

3 Data and Task Abstraction

3.1 Data

Data is initially processed in the form of a table similar to the weather data example presented in Table 1. The rows correspond to the observations, and the columns correspond to the feature variables and class prediction variable. The feature variables are all quantitative, and the class prediction variable is categorical.

The table of data is fed into the random forest model, and the model outputs an ensemble of classification trees. These classification trees are binary tree structured and usually range from a depth of 2 to a depth of 8. Interior nodes of the trees contain the feature variable used at that node position. Each leaf node contains a specific class prediction. The examples we use in this paper will consist of 500 trees.

In the remaining sections, we use data based on features engineered from modulation signals. There are 6 class predictions in total. The classes correspond to on-off shift keying, binary phase shift keying, offset quadrature phase shift keying, and binary frequency shift keying at different levels of carrier frequency. For simplicity, we label these 6 classes with integers from 1 to 6. The feature variables are extracted from Discrete Fourier Transforms of the digitized form of the signals. There are 5 features in total. We simply denote these $m_1, m_2, \ldots, m_5$.

3.2 Task

The first goal is to discover and compare features of high importance from the fitted model. In this paper, we regard feature importance as the number of times the feature appears at node positions. This definition of feature importance is proportional to the usual definition which uses information gain. In addition to individual feature importance, we're also interested in multiple feature level interactions. The user task includes both locating and browsing for important features or multiple feature interaction combinations.

The second goal is to obtain more intuition about the final predictions made by the random forest model. The final prediction is made by the majority vote prediction based on the collection of classification trees. The task is to locate and compare tree structures that reveal different final class predictions made by taking the majority vote of all classification trees.

The third goal is to summarize the random forest model based on feature variable importance and class prediction. Common methods to execute both these tasks are variable importance and multi-dimension scaling plots as described in Section 2. An additional task is to derive an attribute that would quantify the ensembles of classification trees. Subsequently, the ensemble of trees can be further filtered to allow more flexibility in summarization. Hence, another task would be to compare classification trees of unrelated and related members.

4 Solution

4.1 Aggregating Feature Appearances

The solution to locating and browsing for feature importance and interactions require a method to visualize the entire ensemble of trees. For this reason, a new derived attribute is computed. The computation involves aggregating feature variable appearances at node positions.

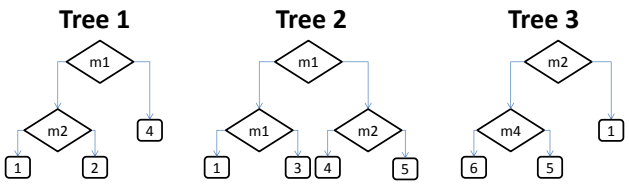

Fig. 8. A simple example of 3 classification trees. Note that the classification tree algorithm in random forests are usually grown deeper than the ones in this example.

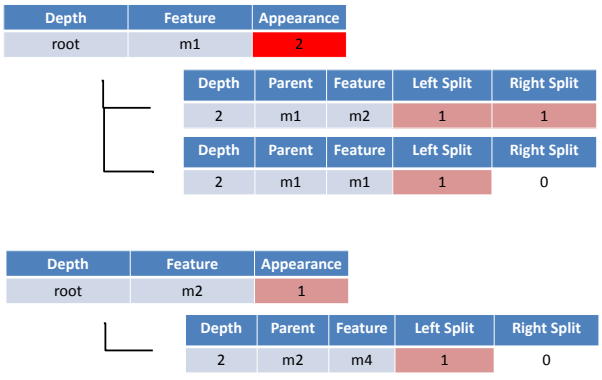

Fig. 9. Derived feature variable appearances at each depth of tree. The derived attribute is encoded with red colour saturation. Feature variables at each depth are ordered based on number of appearances.

For each tree, we determine the features used at each depth, and also whether the split was on the left or right. The first depth corresponds to the root node, so the left or right split is irrelevant. We then compute a cumulative sum total of the number of feature appearances across all

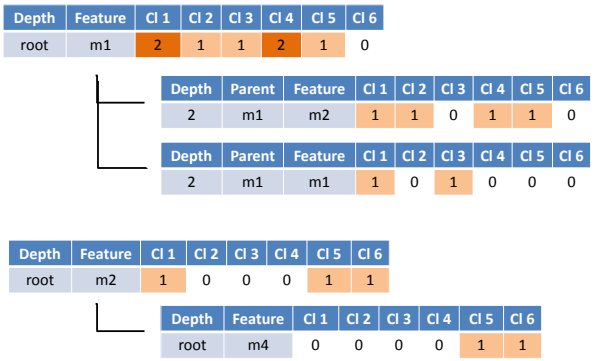

Fig. 10. Derived class count distribution for nodes traversed down so far. The derived attribute is encoded with orange colour saturation. Feature variables at each depth are ordered based on number of appearances.

the trees at corresponding depths. Figure 9 presents the derived feature appearance attribute based on the 3 classification trees example. The derived attribute is encoded with red colour saturation. The features are ordered based on the sum of the left and right split appearances at the specified depth.

From the classification trees example, the m_1 feature occurs twice at the root depth. Therefore, the number of appearances is 2 as shown on the first bar of Figure 9. Similarly, m_2 occurs once at the root depth, therefore it gets a value of 1 as shown on the fourth bar down. Moreover, feature m_2 occurs twice at a depth of 2. Once in tree 1 and once in tree 2. In tree 1, m_2 was split on the left. In tree 2, m_2 was split on the right. These values are shown on the second bar down. The other feature variables are computed similarly.

The reason for choosing red colour saturation is to provide enough contrast with the text. Arguably, other colours would suffice such as yellow, green, and teal. There were no other reasons to choosing red. Originally, we wanted to order the features by name. However, ordering by the derived attribute would be more useful for analysis.

4.2 Aggregating Class Predictions

Another derived attribute is the class count distribution for nodes traversed down the tree so far. At the root depth, all class predictions for a feature variable are included. Let us use the 3 classification trees example of Figure 8. To determine the derived value for m_1 at the root depth, we count up all class predictions obtained given the root node is m_1. Classes 1 and 4 appear twice. While classes 2, 3, 5, and 6 appear once. To compute the class count distribution of m_2 given parent m_1, we count up the class predictions up to node traversed. In the same example, classes 1, 2, 4, and 5 are predicted once given the split m_2 and m_1. The class count distribution for node traversals for m_1 and m_1 has class 1 and 3 occurring once.

4.3 Indented Aggregate Tree

The proposed visualization is based on the indented tree implementation by Mike Bostock. In addition, the nodes of the indented tree contains the encoding for both the feature appearance attribute and class count distribution. The indented aggregate tree is shown in Figure 14. The feature appearance attribute is displayed in red, and the class count distribution is displayed in orange.

4.4 Tree Diversity

One method of quantifying the ensemble of classification trees is to use the hamann similary measure [1, 2]. The measure is computed for every combination of two trees in the ensemble. This hamann similarity measure is modified for multiple classes. The hamann measure in Gatner's paper uses binary classes [1]. Using Figure 11, the hamann

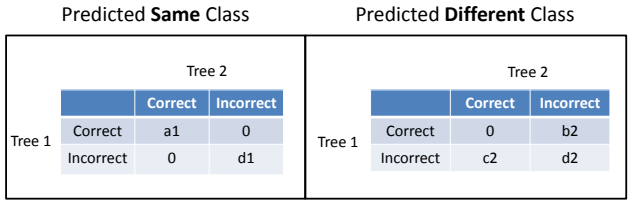

Fig. 11. Contingency tables for two classification trees separated by same and different prediction.

similarity measure for two trees is computed as follows,

$$H(tree_1, tree_2) = \frac{(a_1 + d_1) - (b_2 + c_2 + d_2)}{a_1 + d_1 + b_2 + c_2 + d_2} \quad (1)$$

The hamann similarity measure for a single tree is the average pairwise measure with the remaining trees. The average hamann measure for tree 1 is expressed below where B is the total number of trees.

$$H_1 = \frac{1}{B-1} \sum_{j:j \neq 1} H(tree_1, tree_j) \quad (2)$$

The set of derived measures for the ensemble of trees are presented as a histogram in Figure 12. The unrelated trees are identified as the trees with the smallest values of hamann similarity. Furthermore, the trees can be filtered to allow comparison of unrelated or related members within the ensemble of trees. Figure 13 presents an example of filtering by hamann similarity using a slider widget. The proportion of trees filtered is reflected by the black region filled in the histogram.

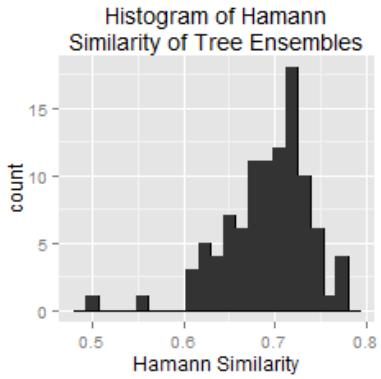

Fig. 12. Histogram of derived similarity measure for each tree in the ensemble.

From the filtered trees, variable importance and multidimensional scaling plots can be plotted. The prediction error is also computed.

5 Implementation

5.1 Aggregated Tree

5.1.1 Model Fitting

Random forests were fit using Scikit-learn from a library in Python [7]. The function RandomForestClassifier takes a table of data consisting of feature variables and one class prediciton variable as input. The output is a collection of classification tree objects. Another parameter is the number of trees which usually range from 500 to 1000.

5.1.2 Extract Fitted Models to JSON

A classification tree object is expressed as a nested dictionary. A dictionary in Python is synonymous to a hash table. The top level of the nested dictionary corresponds to the root node. At each level,

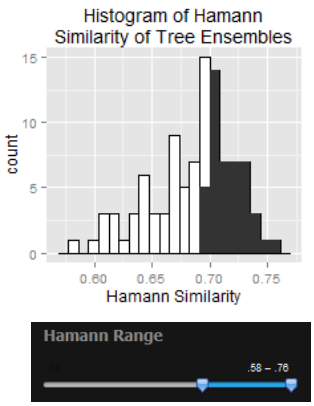

Fig. 13. The ensemble of trees can be filtered based on the derived hamann measures, and the proportion filtered is reflected in the histogram by the area filled out in black.

there is a key corresponding to the name of the feature variable used. There is also a key that maps to its children nodes. If the node is a leaf node, then the class count distribution is given instead of the feature variable. The goal is to recurse down each of the classification trees, and extract just the relevant information described above. An example of the clean classification tree nested dictionary structure is shown below. We output this format into a JSON file. We repeat the process for each classification tree. We used code from a blog post as a base to extract information from a single classification tree https://gist.github.com/pprett/3813537. We modified the code to clean and extract relevant information for aggregation. An example of the nested dictionary structure is shown below.

```
{
feature:m1,
type:split,
children:[{
    type:leaf,
    {
      class1:5, class2:10, class3:0, class4:0,
      class5:5, class6:30
    }
  },
  {
    feature:m2,
    type:split,
    children:[...]
  }
]}
```

5.1.3 Permutate All Feature Node Positions

We created an empty tree by permuting all combinations of feature variables at each depth. Let us call this empty tree as the aggregated tree. The nodes of the aggregated tree will store the number of feature variable appearances and the class count distributions. If we have 5 features, we obtain a 5-ary tree. The aggregated tree uses a nested dictionary in Python. The root node of this nested dictionary only indicates where to begin aggregation. That is, the aggregation of the classification trees begin at depth 2. The example below only uses 1 feature variable and a class prediction variable of 3 classes. The purpose of this example is to show that all the values are set 0 initially. This aggregated tree with all 0 values is exported to a JSON file.

```
{
name:start,
children:[{
    feature:m1,
    appearance:0,
    classCounts:
    {
      class1:0, class2:0, class3:0
    },
    children:[
      feature:m1,
      left-split:0,
      right-split:0,
      classCounts:
      {
        class1:0, class2:0, class3:0
      }
    ]}
]}
```

5.1.4 Aggregating Feature Appearances

The aggregate tree JSON file is loaded back into Python as a nested dictionary. We then iterate through the list of classification tree JSON files. We recurse down each classification tree JSON data while updating the aggregate tree with feature appearances at node positions. For example, if the aggregate tree contains m_1 at its root node, then we add 1 to the aggregate tree at the first feature variable of m_1. The updated structure from the example above looks like the following,

```
{
name:start,
children:[{
    feature:m1,
    appearance:1,
    classCounts:
    {
      class1:0, class2:0, class3:0
    },
    children:[
      feature:m1,
      left-split:0,
      right-split:0,
      classCounts:
      {
        class1:0, class2:0, class3:0
      }
    ]}
]}
```

Notice that the appearance key at feature variable m_1 has a value of 1 now.

5.1.5 Aggregating Class Count Distributions

There is an intermediate step that computes the class count distribution for nodes traversed so far. To compute the class count distribution at a specific node traversed so far, we need to percolate up the class predictions from the leaf nodes. The class predictions are percolated up to the node traversed so far. This process is repeated for all node positions.

The class count distributions are updated similarly to the feature appearances. Instead of adding a value of 1, the values from the class count distributions are cumulatively added onto the class count distributions of the aggregated tree. For example, if the class count distribution of m_1 for class 1, class 2, and class 3 were 20, 50, and 10 respectively, then the nested dictionary would be updated as follows,

```
{
name:start,
children:[{
    feature:m1,
    appearance:1,
    classCounts:
    {
```

```
      class1:20, class2:50, class3:10
    },
    children:[
      feature:m1,
      left-split:0,
      right-split:0,
      classCounts:
      {
        class1:0, class2:0, class3:0
      }
    ]}
]}
```

5.1.6 Ordering the Aggregated Tree

After the aggregated tree is updated from all the classification trees. We recurse down the aggregated tree and re-order the nodes at each depth by the number of feature appearances based on the sum of the left and right split components. If there are no left or right splits, the number of appearances are used.

5.1.7 Filtering the Aggregated Tree

After the aggregated tree is re-ordered, the tree is filtered on any nodes with 0 for both the derived attributes. The aggregated tree is then exported to a JSON file.

5.1.8 Domain Scales for D3

The maximum value of the derived feature appearance attribute is computed by traversing down the aggregated tree. The resulting domain scale is $[0, max(featureAppearance)]$. The domain scale for class count distribution attribute is computed similarly.

5.1.9 Other Libraries Used in Python

The numpy library in Python was used for manipulation of arrays. The pandas library was used for manipulation of data frames. A data frame is synonymous to a table. The json library was used for handling JSON files.

5.2 D3

5.2.1 Indented Tree

The visualization is based on Mike Bostock's indented tree. The implementation is found at http://bl.ocks.org/mbostock/1093025.

5.2.2 Feature Appearance Attribute

The derived feature appearance attribute is encoded with red colour saturation. A linear scale colour map is created by mapping the domain scale to the range. The range varies from white to red. I obtained the code to build the colour map from this post, http://synthesis.sbecker.net/articles/2012/07/16/learning-d3-part-6-scales-colors. Rectangular boxes are used to display the derived attributes. The colour is filled in according to the linear scale colour map. The attribute value is displayed in the middle of the box. Labels are displayed within the boxes. If the node corresponds to a root node from the classification tree, then it is labeled as 'root'. Otherwise, the labels correspond to 'left-split' or 'right-split'.

5.2.3 Class Count Distribution Attribute

The derived class count distribution attribute is encoded with orange colour saturation. We used the domain scale stored in the JSON file containing the aggregated tree. A linear scale colour map is created similar to the one for feature appearance. The range varies from white to orange. Rectangular boxes are used to display the derived attributes. There are 6 boxes in total. One for each class. The boxes are filled in with colour corresponding to the linear scale colour map. The derived attribute value is displayed in the middle of the box. Labels are displayed within the boxes. The labels correspond to each of the 6 classes.

5.3 R and Shiny

An app was built for quantifying the ensemble of trees using R shiny [8]. The app allows the user to fit a random forest model which uses the randomForest library [3]. We used Matthew Leonawicz's implementation of variable importance and multidimensional scaling plots. Leonawicz's Random Forest app can be found at http://blog.snap.uaf.edu/2014/03/25/r-shiny-randomforest-with-base-graphics-and-ggplot2/. The hamann similarity derived attribute is computed with base R functions mostly. The plyr library was used to allow split-apply-combine algorithms to be applied on data frames [13]. The background theme uses one of the default themes from the Twitter Bootstrap library.

5.3.1 Other Libraries Used

The reshape2 library was used to re-shape data frames to easily allow the correct format to be used in ggplot2 [11]. The plotting functions of variable importance and multi-dimensional scaling plots used the ggplot2 library [12].

6 Results

6.1 Indented Aggregate Tree

Initially, the indented aggregate tree visualization displays the number of feature appearance at root nodes across the ensemble of classification trees. The initial visualization is shown in Figure 14. The figure reveals that feature m_5 appeared at the root nodes of 172 classification trees. Whereas, feature m_2 only appeared at root nodes of 4 trees. This suggests that m_5 has greater individual feature importance than m_2. We could click on the nodes at the current depth to display the next depth. For example, clicking on m_5 of Figure 14 will result in the image shown in Figure 15. The second image reveals two way interactions. More importantly, there are more than 40 classification trees where m_3 feature were on the left split of feature m_5. This suggests that smaller values of m_5 favours splits of m_3 on the left much more than on the right. Although, the indented aggregate tree visualization easily reveals information at the first and second level of the tree, it requires more effort to search through deeper levels of the tree. Moreover, the traditional feature variable importance plot has the advantage of summarizing feature importance based on all levels of the trees for each feature. Figure 16 shows multiple-level feature interactions.

From Figure 15, the class count distribution appears to be fairly uniform for all features at the root node split. However, splitting by m_3 given m_5 reveals higher frequency predictions for classes 3, 4, and 6. This suggests that classification trees with splits m_3 given m_5 contribute towards class predictions of classes 3, 4, and 6 in the majority vote final prediction.

6.1.1 Scale

The algorithm currently does not scale well. With 5 feature variables, the maximum depth allowed for the indented aggregate tree is 8. This limitation is due to an out-of-memory issue. The problem occurs in the step of the algorithm that builds the initial aggregate tree by permuting all feature node positions. This step should be removed. The nodes should be appended on the aggregate tree as we iterate through the classification trees. Nevertheless, the algorithm runs quickly for 1000 trees with depths less than 7. A simple computational benchmark for 4 different settings of number of trees and tree depth is shown in Figure 17.

6.2 Tree Diversity App

In the R Shiny app, we can enter the number of trees to fit the random forest model. The minimum number of trees is 100. The 'Fit Random Forest' button fits a random forest model, and computes the average pairwise hamann similarity measure for each tree in the ensemble. The slider filters the trees. The filtering is highlighted in a histogram as shown in Figure 13. Furthermore, we could plot between 2 common visualizations of random forest models. These 2 visualizations are the variable importance and multidimensional scaling plots as shown in Figures 5 and 6.

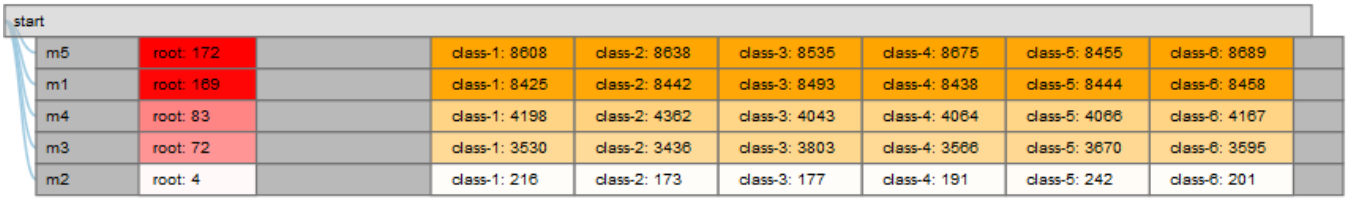

Fig. 14. Indented Tree visualization of aggregated ensemble of classification trees. This screen shot shows feature appearances at root nodes of classification trees. In addition class count distributions are shown after splitting by corresponding root node.

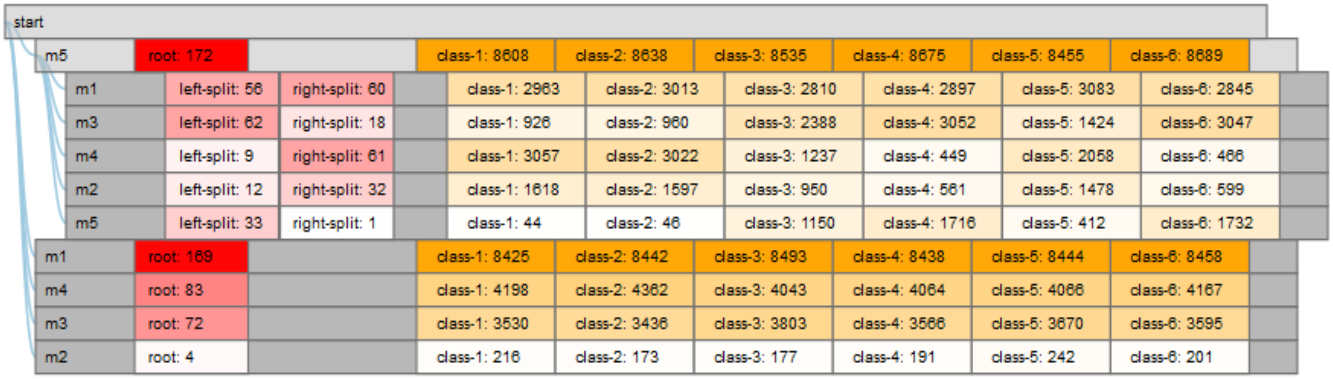

Fig. 15. Indented Tree visualization of aggregated ensemble of classification trees. This screen shot shows two-way interaction between m_5 and the other 4 features.

Number of Trees	Depth	Time
200	7	59 sec
800	3	10 sec
1500	3	15 sec
1500	6	22 sec

Fig. 17. Computational benchmark for the aggregation process based on number of depths and trees to generate the aggregate tree.

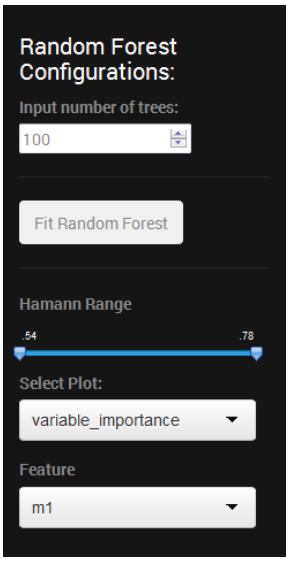

Fig. 18. Shiny app configurations.

7 Discussion and Future Work

7.1 Strengths

The indented aggregate tree provides a way to visualize the most important information from an ensemble of classification trees in a feasible and efficient manner. The visualization provides both feature importance and two-way feature interactions.

The strength of the R shiny app is the ability to allow the user to filter trees to enable comparison between the most unrelated or related members in the ensemble of trees. The commonly used variable importance and multidimensional scaling plots are static, and only show a summary of feature importances and class predictions.

7.2 Weaknesses

Browsing multiple-feature interactions is still difficult and time-consuming. It is a difficult task to find interesting four-way feature interactions. Not to mention, the task becomes more difficult as we increase the number of feature variables. The visualization should provide a legend that labels the feature and class prediction variables. Otherwise, a button should be provided that allows the user to switch between the labels to be on or off. The visualization should allow the user to link highlight a particular path down the indented aggregate tree. The aggregation process also does not scale very well. There is an out-of-memory issue for an aggregate tree with depth greater than 8.

The problem with the R shiny app is also in scale. However, the computational complexity is affected based on the number of trees instead of the number of feature variables. The current approach computes the hamann similarity measure B^2 times, where B is the number of trees. Whereas, it should only compute $\binom{B}{2}$ times, because that is the number of feature combination pairs. Fitting more than 400 trees is not recommended.

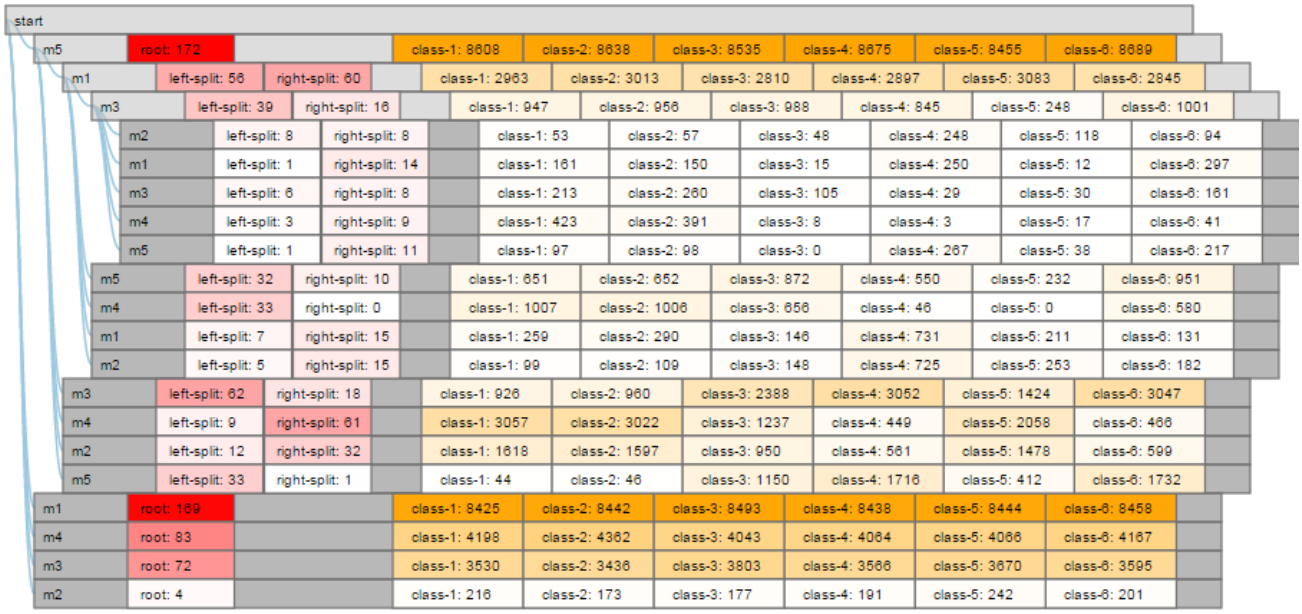

Fig. 16. Indented Tree visualization of aggregated ensemble of classification trees. This screen shot shows multiple-level interactions.

7.3 Lessons Learned

We learned about the process of taking a complex data such as an ensemble of trees, and apply aggregation methods to build a simple visualization that preserves relevant information from the trees. We also learned about how different combinations of visual encodings could impact the visualization, such as colour mixed with spatial position. We learned about methods of quantifying and filtering data based on a derived attribute.

7.4 Future Work

The aggregation algorithm could definitely be improved. An interface should be built for the indented aggregate tree to provide more intractability. The interface should allow the user to load in data in the form of a table with multiple feature variables and one class prediction variable. There should be a button that fits the random forest model which automatically produces the indented aggregate tree. The visualization should allow link highlighting of paths along the tree.

8 Conclusions

We introduced a visualization that aggregates over an ensemble of classification trees. The visualization is based on the indented tree built in D3. The nodes of the indented tree contain total number of feature appearances at node positions across all classification trees. This information is relevant in determining feature importance and two-way interactions. The visualization also provides class prediction count distributions. In the R shiny app, we derived a diversity measure score for each tree in the ensemble. We could then filter the trees by diversity measure, and produce variable importance and multidimensional scaling plots.

Acknowledgments

The authors wish to thank Dr. Tamara Munzner for providing helpful comments and guidance.

References

[1] E. Gatnar. A diversity measure for tree-based classifier ensembles. In *Data Analysis and Decision Support*, pages 30–38. Springer, 2005.

[2] L. Kuncheva and C. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

[3] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[4] R. W. M. Mirzargar and R. Kirby. Curve boxplot: Generalization of boxplot for ensembles of curves. 2014.

[5] D. Z. M. Yang, H. Xu and H. Chen. Visualizing the random forest by 3d techniques. *Internet of Things*, pages 639–645, 2012.

[6] T. Munzner. *Visualization Analysis and Design*. A K Peters Visualization Series. CRC Press., 2014.

[7] F. Pedregosa et al. Scikit-learn: Machine learning in python. *The Jour. of Mach. Learn. Res.*, 12:2825–2830, 2011.

[8] RStudio and Inc. *shiny: Web Application Framework for R*, 2014. R package version 0.10.2.1.

[9] R. T. T. Hastie and J. Friedman. *The elements of statistical learning*, volume 2. Springer, 2009.

[10] M. Wattenberg. Visual exploration of multivariate graphs. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 811–819. ACM, 2006.

[11] H. Wickham. Reshaping data with the reshape package. *Jour. of Stat. Soft.*, 21(12):1–20, 2007.

[12] H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009.

[13] H. Wickham. The split-apply-combine strategy for data analysis. *Jour. of Stat. Soft.*, 40(1):1–29, 2011.

[14] M. Wilkinson. Majority-rule reduced consensus trees and their use in bootstrapping. *Molec. Bio. and Evo.*, 13(3):437–444, 1996.