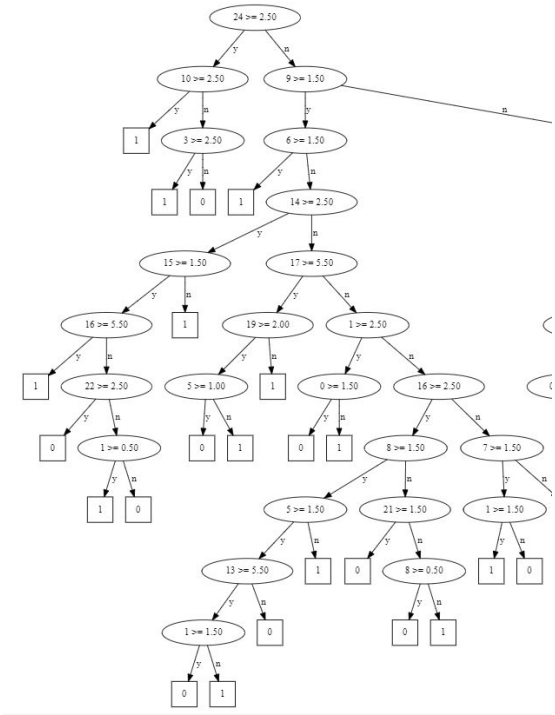
Autism Classification with Decision Trees and Random Forests



Computational Medicine

Addison Boyer, Spring 2020

1. Data

The data was already separated into a training and validation dataset, which was really nice. The first step in the process of building the decision tree classifier was to read in the data. In order to remain consistent, patient observations were removed in the training dataset if they were blank (i.e. nan), and also if a single patient was reviewed less than 3 times. For those who were reviewed as many as 3 times, I randomly selected a subset of 3 observations for the final training dataset. Lastly, because “non-asd” children were highly mis-represented in the training dataset, I performed upsampling by sampling with replacement from the minority population until the number of “non-asd” and “asd” patients were equally represented in the training dataset.

Observations from the validation dataset were removed if they were invalid (i.e. nan). However, observations were not upsampled in the validation dataset. In total the sizes of the training and testing set after upsampling and the methods outlined above were **(654, 31)** and **(201, 31)** respectively. In total, the validation dataset contained 67 unique children.

2. Feature Selection

Two different cost functions were employed in order to select features (questions) and values (responses) at each decision node in the tree. These two functions were overall entropy, and a weighted gini score. I wanted to build a model utilizing entropy, and one utilizing the gini score to compare the results, and to see if one metric outperformed the other on this problem.

In order to select features, I first get all the possible partitions of the current data at a decision node starting from the root. For every feature, I get all unique responses to that question (feature) and sort them from smallest to largest. Then, I take the average of two successive values, and that is deemed the split. A hash table with features as keys, and lists of splits as values is returned for the next step.

During this step, all split questions-value pairs gotten from the previous step are used to partition the data and calculate the entropy, or gini score of that split. This is done for all question-value pairs, and the question-value pair with the minimum for either entropy, or gini score is returned. This feature is then added to a set of features. The left and right children (no/yes) are enqueued onto a queue, and the process is repeated for these children, their children, grandchildren, etc until the set of features reaches size k (where k is the predetermined number of features).

3. The Decision Tree Model

Now that features have been selected, the next step is to build a decision tree model with these features. My decision tree algorithm is recursive and works like so. The base case is simple, and simply checks if the data passed to the function is a single class ("asd" or "non-asd"), and if so, a classification node is returned (either 1 or 0). If not, then all possible partitions are calculated, now considering only the k features that were chosen in the previous feature selection step. Then, the best question-value split is calculated using either the entropy, or gini score. If the data is unable to be partitioned further, then a classification is returned, which will be the majority classification of the data. If able, the data will be split on the question and value returned above, producing two datasets a and b. The algorithm is then recursively called on the left (b) and right (a) subtrees. This will recursively build the tree, until finally the root of the tree is returned.

Visualizing the trees: After the trees are built, a breadth first traversal of the tree is performed, and a digraph symbolizing the decision tree is built. This digraph can be easily visualized with the graphviz package for Python.

4. Validation

After the decision tree models are trained for both entropy, and the gini score they are validated on the validation dataset. For validation, I employ a majority vote for classification similar to that of the authors of the original paper. For each patient, there are 3 independent feature vectors that will each receive a single classification through traversal of the decision tree to a leaf node. The classification with the highest number of votes will become the final classification.

After classifying each child, a confusion matrix is generated by comparing the actual labels, to the labels produced by the decision tree model. From this confusion matrix, sensitivity, specificity, unweighted average recall (average of sensitivity, and specificity), and accuracy are calculated. The training and validation procedure is repeated for 500 trials with 3 to 10 features on trees trained using entropy, and also those using gini scores. From these values 95 percent confidence intervals are calculated, as well as mean values of the metrics outlined above. These plots for 3-10 features with both entropy and gini scores respectively can be found in the jupyter notebook for this project (linked at the end of this document). Using 3-10 features I was able to achieve approximately 84% test set accuracy, and around 92% training set accuracy which suggests that the model is not being overfit.

5. Feature Analysis

After training decision tree models using 3-10 features I wanted to get a more global idea of the most important features being selected at each step during the training process. To do so, I trained the model using all features, not just a subset of them, and kept a hash table of features to their count (weighted proportional to the depth at which they were selected in the tree). For example if feature 1 is selected at depth 1 then it's total count gets incremented by $1 / 2^1 = ½$. In general the feature count is incremented by $1/2^d$, where d is the depth at which the feature is selected to partition the data. After getting the counts, I normalize them into a probability distribution. The results of running all feature analysis on both trees trained with entropy, and gini scores are found in the jupyter notebook. It appears that features 1, 9, and 10 appear most frequently in both models. It'd be interesting to know what features these correspond to, in order to verify the clinical significance.

6. Random Forests

To create a random forest classifier, the decision tree classifier outlined above must be modified slightly. This slight modification, in the step in which a question and split value are chosen to partition the data. I decided to try a few different approaches to this "Random" partition step. First, randomly choosing both the question and value uniformly. In other words each question and split value had an equal probability of being chosen at each step. From here the method to create the random forest is quite simple. Create **t** single decision trees using a random partition step, instead of lowest entropy or gini index, and then get a classification of the child from each tree. The final classification is the consensus agreement among all trees in the random forest.

First, I ran trials of the random forest classifier with 3-10 features using a uniform probability distribution to select splits at each decision node. A plot of the mean sensitivity, specificity, uar, and accuracy as well as 95% confidence intervals can be found in the jupyter notebook. Next, I performed this same experiment using all features instead. In order to see if there was any difference in using a non-uniform probability distribution to select which features, and values to split the data, I ran these same experiments with the probability distributions gotten from step 5. I did this for the Entropy, and Gini probability distributions using all features and the same number of trees and trials as before. The average specificity, sensitivity, uar, and accuracy for these trials as well as 95 percent confidence intervals (given as +- a value) are displayed in the jupyter notebook.

7. Conclusions

It appears that the decision tree models do pretty well classifying new observations in the validation dataset, and are not being overfit. However, the specificity is low for both models suggesting that they might be better suited as a screening tool as opposed to performing strict diagnoses. Moreover, it shows that there is likely a better classification method for this problem / dataset. The random forests appear to perform better than the single decision trees, but are more computationally expensive. I suspect that increasing the number of trees from 21 in each random forest will increase the accuracy of the model. It appears that using a uniform probability distribution produces similar results to those trained with probability distributions gotten from the feature analysis of both decision tree models. This suggests that this knowledge does little to help a random forest make better classifications. Overall the models produce similar results to those found in the paper on this topic.

Jupyter notebook link*:
https://github.com/addiboyer24/AutismClassifier/blob/master/Autism%20Classification%20-%20Final.ipynb

*** Be sure to check out DecisionTree.py (code), and the Part 1, Part 2, and Part 3 notebooks, which have some visualizations of the decision trees, training set metrics, etc..**