UADecisionTree: An implementation of Decision Trees

Noah Buchanan
Computer Science Department
University of Arkansas - Fort Smith

February 21, 2021

Introduction

UADecisionTree makes use of entropy, conditional entropy and information gain to classify categorical values based on training data. Minimum impurity can be set to specify conditions on which to classify when the entropy is $\leq$ minimum impurity. As well as maximum depth, when we decide to stop splitting on features and classify the response where it is.

Background

This approach of classification requires knowledge in entropy, the impurity of data or how evenly split the individual values of a feature are laid out, information gain, a metric for determining which feature if split upon has the greatest impurity reduction, and decision trees which classify a value based on training done beforehand by creating a tree with branches that correspond to specific values for features that if followed will lead to a predicted response. This algorithm is a supervised learning algorithm, meaning it infers a function from training data that is labeled and attempts to reproduce those results with the inferred function.

Decision Tree classification can be an extremely valuable asset in situations of categorical data for predicting certain things. A perfect example of this is heart disease diagnoses based on patient data. There is a huge amount of data on this to extract useful knowledge as it is world-wide and increasing in mortality rate each year [1]. In the paper they mention multiple discretization methods, I wont go into these in depth, but one of the methods they use is entropy, same as my model [1]. Something interesting I thought to take note of which wasn't prevalent in the training data I used so it was not included but something that should definitely be noted is the gain ratio. The gain ratio was used to reduce the effect of the bias resulting from the use of Information Gain, the Information Gain measure is biased toward tests with many outcomes [1] so

they used the gain ratio to counteract that, Gain Ratio = Information Gain/ Split Information, where split information is a value based on the columns sum of the frequency table [1].

Specification

For this to work we must first acquire a training model from data that we wish to use, in the process of acquiring this training model we remove non-categorical values, specified as having over 20 unique values for this specification. Then based on this training model, specified as a 2 dimensional array for this algorithm, we use the following recursive algorithm to build the tree from a preconceived node that we pass into it as "head". Note that min_impurity and max_depth must both be set before building the tree.

```
train(matrix[][], currentdepth, parent){

  if(entropy(matrix) <= min_impurity or currentdepth
                              >= max_depth or homogeneous(matrix)){
    parent.leaf = true
    parent.response = classifyResponse(matrix)

  } else {

    featureToSplit = findHighestIGIndex(matrix)
    parent.feature = featureToSplit
    iterations = 0
    for(x in distinctValuesOf(featureToSplit)){
      node = new Node
      parent.Children.put(x, node)
      train(filter(matrix, featureToSplit, x),
                  ++currentdepth-iterations, node)
      iterations++
    }
  }
}
```

For the entropy function and findHighestIGIndex function, instead of pseudocode I will provide the mathematical formulas instead so that I can detail the extra utility functions in my implementation section.

$$H(X) = -\sum_{i=1}^{n} Pr(X_i) log_b(Pr(X_i))$$

$$H(Y|X) = \sum_{x} Pr(X = x) * H(Y|X = x)$$

$$= -\sum_{\forall x \in X} Pr(X = x) \sum_{\forall y \in Y} Pr(Y = y|X = x) log_b[Pr(Y = y|X = x)]$$

$$IG(X, a_k) = H(X) - H(X|a_k)$$

Where a_k is an attribute of the matrix.

Implementation

Things specific to my implementation were various utility methods that made things simpler. In specific the distincts() method returned a hashmap of distinct values for a feature given a matrix and the column of the feature you want to find distincts for. It returns the distincts in the keyset however the keys map to integers which I use for when I need to count the distincts. For example I create an array of integers of the same size and use the map.get() to use as an index and add 1 to that value of that index and it makes counting distinct values much easier which I thought was quite clever. homogeneous() simply returns whether the response is all one value or not. filter() takes in a matrix, a column index, and a specific value of that column and filters all the data specifically where that column had that specific value and removes the column then leaving a filtered matrix. train() calls training(), training() is really the recursive algorithm and train() calls training(matrix[][],currentdepth,Head) just for ease of access when calling train so that you dont have to specifically provide the same static variables every time as it will always be the same. The same is true for classifying() and classifyValue(), classifyValue() calls classifying() with some added parameters that I do for you so that you only need to call classifyValue(). classifyResponse() determines the highest probability response based on the matrix passed in, either it returns the response with the highest count or just returns any of them if it is homogeneous. Lastly removeNonCategorical() simply does exactly what it says, it removes any non-categorical features.

Evaluation

The data set I used for the majority of my testing was a titanic data set of 6 binary features. It also works with non-binomial data sets as well, however if the data set is too small and I split the data 80% for training 20% for testing the accuracy can be quite skewed as there are not enough values for the tree to build an accurate model. On the other hand on the small data sets where I used 100% of the data for training it accurately predicts 100% of the time. The difference

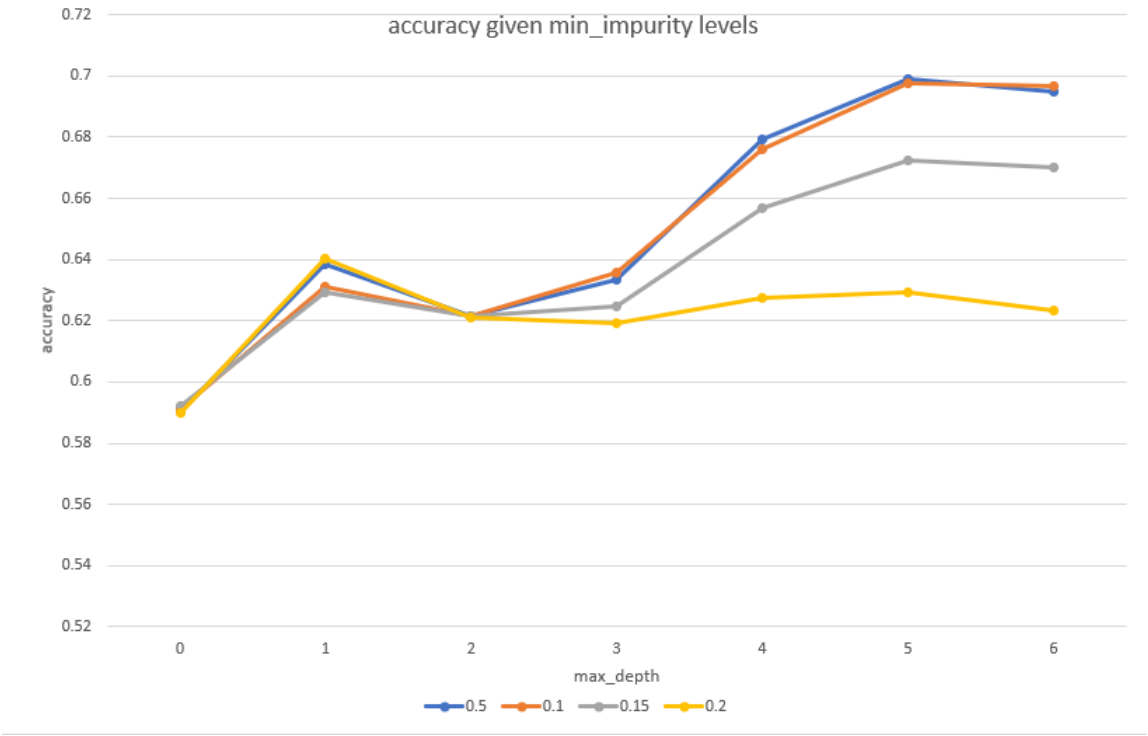
is negligible on larger data sets however as when I did the same for a larger data set (used 100% for training) it predicted with 72% accuracy roughly on average and that is about where the average is when calling getRandomyAccuracy() which does in fact do the 80 20 split. Below is an included image of a test run in PS2Runtime, I demonstrate all capabilities of the program, classifying an input using UADecistionTree and then after using UADecisionTreeTest I call getBestDepth() and getRandomAccuracy(), here are the results:

```
Classifying value: "1,1,1,1,0,1"
input classified as: 1

finding best depth
----------------------------------------
depth 0, average accuracy and variance: 0.5832867 0.001696997
depth 1, average accuracy and variance: 0.61230767 0.005022565
depth 2, average accuracy and variance: 0.615944 0.0026286659
depth 3, average accuracy and variance: 0.63307697 0.002638228
depth 4, average accuracy and variance: 0.6732868 0.0016138491
depth 5, average accuracy and variance: 0.7058042 0.0011504275
depth 6, average accuracy and variance: 0.6907692 0.0013698271
best depth: 5

data split randomly 100 times
-----------------------------------
average accuracy and variance: 0.69748247 0.0011811791
```

As you can see there are heavy diminishing returns past depth 4, it is usually a coin toss for depth 5 or depth 6 being the best on 80 20 splits of the titanic data set specified above. It would make sense that more splits always equals more accuracy as long as they are reasonable, the reason depth 5 sometimes is better is probably just a case of getting better splits on average than depth 6 and since the diminishing returns are so large and their values are so similar it could make the change easily. The titanic data set has 6 binomial covariants and 1 binomial response with 714 records. However as I said before it does not need to be binomial for it to work. I will also provide the results of getRandomAccuracy given multiple variations in min_impurity and max_depth.



Conclusions

UADecisionTree works as a supervised learning classification algorithm using Entropy and Information Gain to build a decision tree to classify responses. From testing with these on the chart above we can see that in general the accuracy tends to improve as splits increase and we are more strict on the minimum impurity, there seems to be more scaling the lower the minimum impurity we allow. We can assume based on this data that we can tend to accurately predict responses using this approach specific to my implementation and this algorithm quite well with variations depending on the data of course and the parameter values you provde.

References

[1] M. Shouman, T. Turner, and R. Stocker, "Using decision tree for diagnosing heart disease patients," in *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, 2011, pp. 23–30.