Assignment 1 - Random Forests

Liam Duncan - 301476562

February 16, 2024

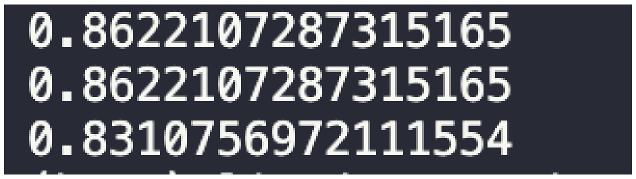
Decision tree

```
generate_tree(dataset D, Attributes A, Class C)
// used to initialize the first node in the decision tree
    root = Node(D);
    split(root,A,C)
    return root;


split(node n, dataset D, Attributes A, Class C)
// determines the best split of numeric and categorical attributes based on gini
// Recursuvely calls until data is empty or all belong to same class
    if all samples in D belong to the same class:
        n.label = majority class in D;
        return n
    else if A is empty or depth limit reached:
        n.label = majority class in D;
        return n
    else:
        best_split = none;
        best_gini = inf;
        best_num_split = null;
        for all a in A:
            tot_size = length(d);
            if a.isnumerical:
            // will find the best split of a for all of D
                best_gini_of_attribute = inf;
                for all d in D:
                    value = d;
                    l_data = D[a] < value;
                    ge_data = D[a] >= value;
                    l_gini = calc_gini(l_data, C);
                    ge_gini = calc_gini(ge_data, C);
                    l_size = length(l_data);
                    ge_size = length(ge_data);
                    weighted_gini = ((l_size/tot_size)*l_gini + (ge_size/tot_size)*ge_gini)
                    if (weighted_gini < best_gini_of_attribute):
                        best_gini_of_attribute = weighted_gini;
                        best_num_split = value;
                if best_gini_of_attribute < best_gini:
                    best_gini = best_gini_of_attribute;
                    best_split = a;
            else:
            //calculates the weighted gini for all the possible values of attribute
                wieghted_gini = 0.0;
                for possible_value in a:
                    temp_data = D[a] == value; all data that has value possible_value;
                    subset_gini = calc_gini(temp_data, c);
                    size = length(temp_data);
                    weighted_gini += ((size/tot_size) * subset_gini);
                if weighted_gini < best_gini:
                    best_gini = weighted_gini;
                    best_split = a;

        if best_split is not None:
        //create new nodes and recursively call split on new nodes
            if best_split.isnumerical:
            //if numeric only have 2 node, Less or equal/greater
                    l_data = D[best_split] < best_num_split;
                    ge_data = D[best_split] >= best_num_split;
                    l_node = node(l_data)
                    ge_node = node(ge_data)
                    n.setchildren(l_node, ge_node);
                    split(l_node, l_data, A-best_split, C)
                    split(ge_node, ge_data, A-best_split, C)
            else:
            //for categoric create node for each value
                for values in best_split:
                    data = D[best_split] == values;
                    new_node = node(data)
                    n.children += new_node;
                    split tree(new_node, data, A-best_split, C)


calc_gini(dataset D, class C)
// Function used to calculate the gini
    gini_index = 0;
    size = length(D);
    for values in C:
        prob = (length(D [where C = values]))/ size;
        gini_index = gini_index - (prob)
    gini_index = gini_index + 1;
    return gini_index;
```

For the main.py I changed the default path of data because it wasn't entering the folder containing the data. Not sure if this is an issue, but hopefully its ok if its not I'd be happy to fix it to make it work for you.

1

For the sake of time, I chose to use the mean for each attribute. The results are about as expected. The test data is less than the training data because the test data had not been seen before. When this is run multiple times the accuracy changes very little and stays around 86% for training and 83% for test.

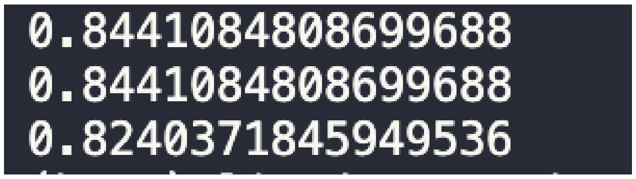


```
0.8622107287315165
0.8622107287315165
0.8310756972111554
```

Figure 1: results from part 1 using Gini index. First 2 are training data, Bottom is test data.

2

Using the same procedure as part 1 except with entropy as the criterion instead of gini, the results changed. The accuracy across the board, for training and test data, dropped. For training data it dropped roughly 1.8% and for the test data it dropped roughly 0.7%. Due to the nature of the data set being relatively unbalanced with 75.22% being of one class and 24.78% of the other gini is more favourable. Typically when a dataset is balanced entropy will be more accurate; however, because this is not true in this case, gini gave a higher accuracy. This does not mean that this will hold true for all datasets but it ddeos for this one.



```
0.8441084808699688
0.8441084808699688
0.8240371845949536
```

Figure 2: Part 2 results using entropy. First 2 are training data, bottom is test data.

3

The training data is not 100% accurate for a variety of reasons. One being that not all the training data may have been seen when creating the trees in the forest. This is because each time a tree is created it takes a random subset of data from the dataset. This means that some data may not have been used when training the trees. Another reason is because not all the features were used when creating the trees. Once again a random subset of attributes was taken to create the trees. This means that the combination of attributes that may have been the best to train the data may not have been used. Third, we set a max depth meaning the trees weren't grown until all data in all leaf nodes belonged to the same class. This means that some of the leaf nodes still had a mix of the 2 classes in it. Therefore when data is sent down the decision tree and it reaches a leaf node, it does not guarantee that it will belong to the class of the leaf node, instead it just gives the class it most likely is based off of the majority class of the node.

a

It is possible to train a tree to 100% accuracy. Although this may seem ideal, in reality it is not. This is because the tree has then become overfit. When the tree is overfit, it has learned the specific training data too well and tailored itself to it instead of the actual trends and patterns of the data. As a result of overfitting, the accuracy of training data would be less due to the tree being tailored to the training data and not the overall patterns of the data.

b

There would be multiple parameters that would need to be change to make achieving 100% accuracy achievable. The first would need to be max depth. I would set it to be unlimited that way the tree is able to grow deep enough to allow for all the data to fit in a leaf node where the class of that node is pure, all data belongs to the same class. The minimum sample split would need to be set as low as possible. This would allow for nodes with very few samples, to be split further to ensure that all leaf nodes only contain data of the same class. In our case by having it set to 20, it allows leaf nodes to contain data of both classes even if a split would yield deeper nodes with higher purity. Training the data this accurate would lead to high variance and low bias. It has high variance because it is trained so exactly to the training data, this means that it may lack when trying generalize new data. For this reason it has high variance. The model would have low bias because it makes no assumptions in the data. It is able to approximate the relationships within the data.

4

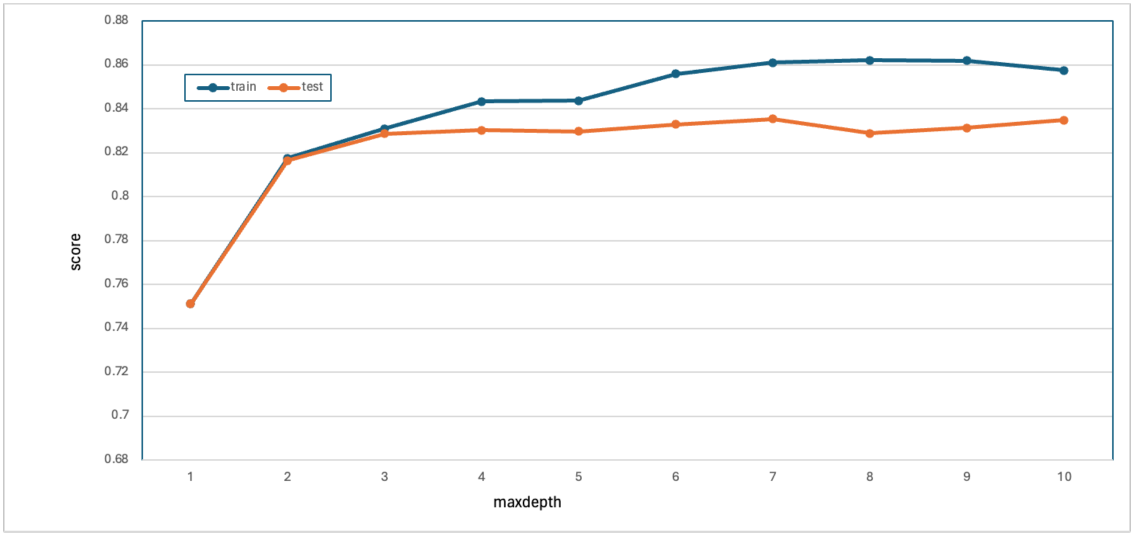


Figure 3: test score vs training for different maxdepth

The graph shows that the deeper max depth is does not mean that it will improve the test score. This shows that the accuracy of the model is largely based off of 2 features. The other features may slightly increase the testing data over time. However once max depth reached 8, we'd likely slightly overfitted as the test score slightly decreases. This means that the model was trained to heavily on the training data and lacks some generalization needed to classify the test data. It is surprising and almost worrying that the test accuracy doesn't increase much more after the depth reaches 2. I ran this experiment again to see if the data would change very much and it was nearly identical. By setting max features to 10 it means it is very likely that the 2 most important features are chosen every time and this is why the model does not change much after. After some analysis, I noticed that the test data is very similar to training data at the start. This is because by only splitting by one feature it will likely have an almost exact representation of what the test data would look like being split by 1 feature. The deeper the tree grows the difference in test and training grows. This is likely because by making the tree more complex based off the training data increases the training score but starts to lack representation of all possible data that could be included in the test data.

Overall I am surprised at the accuracy increase from 1 - 2 and the lack of increase from there on out. The slight dip in training accuracy at depth 9 and 10 is also slightly surprising and concerning.