Introduction to Machine Leaning Project 3: Decision Trees

Charlie Hammond CHAMMO24@JHU.EDU

1. Introduction

Decision Trees are a type of Inductive Logic Programming where rules are arranged in a tree. They attempt to model data with a tree containing basic logic rules at each node. Values are predicted for instances by traversing the tree to a leaf node, following child nodes where the rules evaluate to true.

Decision Trees are useful because they are powerful and simple to understand. The nodes have basic rules that determine splits. This makes the trees easily understandable and traceable to people who may not have deep understanding of Machine Learning.

I hypothesize Decision Trees will work well to model the six data sets. It makes intuitive sense that a tree trained on similar data will produce a tree and rules that closely model other data. I additionally hypothesize pruned or early stopped decision trees will perform better than a tree that is fully grown. I believe this because Decision trees are susceptible to overfitting, and this can reduce their predictive power. In many cases, trees are grown until leaves approximate just a single data point in the training set, in effect memorizing the training data. Early stopping helps this by only having branches and nodes that help with the predictive power of the tree.

2. Experimental Approach

I used the ID3 algorithm to build Decision Trees for the three classification data sets. ID3 only works for classification data. The algorithm starts with the full data set at the root of a tree. It then attempts to split the data into child nodes, where each split is on a particular attribute. The attribute is chosen to minimize the entropy after the node has been split into child nodes. The algorithm pseudocode is below (Alpaydin 223).

```
GenerateTree(X)
    If NodeEntropy(X) < threshold
        Create leaf
        Return
    I = SplitAttribute(X)
    For each branch in xi
        Find dataset Xi
        GenerateTree(Xi)
SplitAttribute(X)
    maxGain = 0, bestSplit = 0
    For all attributes
        If x is discrete
            Split x into each attribute
            Gain = information gain
```

```
                If Gain > maxGain -> maxGain = Gain, bestSplit = i
        Else // numeric
                For all possible splits
                        Split X into X1, X2 at split
                Gain = information gain
                If Gain > maxGain -> maxGain = Gain, bestSplit = i
Return bestSplit
```

Features with discrete values are split with each unique value getting a child node. Then, when evaluating a new point against the tree, a feature with a particular value will follow that child in the tree. Numeric trees are a little more complicated because we cannot create a child node for each unique value. I therefore used basic binary splits for numeric attributes with the mid-point of the data used as the evaluation criteria. Points with a feature value less than the middle point go to the left child, while the rest go to the right child.

Entropy is used to show how much uncertainty there is in the data set. The function output ranges from 0 to 1, where 0 indicates the class values are all the same, while 1 indicates the class values are equally split among the set of possible values. We attempt to minimize entropy with splits, so leaf nodes accurately predict class labels. The entropy formula for a single node is shown below:

$$I(c_1, \dots, c_k) = -\sum_{i=1}^{k} \frac{c_i}{c_1, \dots, c_k} \log \frac{c_i}{c_1, \dots, c_k}$$

This entropy value is used to determine the information gain with splits at each attribute in the data. Each split attempts to maximize the information gain, thus generating a tree that better models the data. The formula for information gain for a split on attribute π is shown below. $c_{\pi,1}^{j}$ is the number of examples in the jth partition from the feature we are currently splitting on.

$$gain(f_i) = I(c_1, \dots, c_k) - \sum_{j=1}^{m_i} \frac{c_{\pi,1}^{j} + \cdots + c_{\pi,k}^{j}}{c_{\pi,1} + \cdots + c_{\pi,k}} I\left(c_{\pi,1}^{j}, \dots, c_{\pi,k}^{j}\right)$$

In general, Decision Trees are biased towards splits on discrete attributes with many different values. It is more common for splits with many different child nodes to reduce entropy more than basic binary splits. To reduce the bias, I used an adjusted entropy calculation, Gain Ratio to normalize discrete value splits with many unique values. I used the equation below to determine which split, f_i, has the highest gain ratio and split the tree based on that feature.

$$gainRatio = \frac{gain(f_i)}{-\sum_{i=1}^{k} \frac{c_i}{c_1, \dots, c_k} \log \frac{c_i}{c_1, \dots, c_k}}$$

I created Decision Trees for regression data sets using the CART algorithm. The CART algorithm is very similar to the ID3 algorithm, with the main difference in how splits are determined. Entropy is not a valid measure because the class values are continuous. Therefore, CART uses mean squared error to determine the best splits. With each split in the tree, we chose the attribute that minimizes total mean squared error.

Once the trees were built, I used test datasets to evaluate their performance. The algorithm for traversing a tree to leaves is relatively simple. It takes a data point, x, and at each node in the tree, finds the child node that corresponds to the rule output. The full algorithm is below.

```
TraverseTree(x)
    While (!currentNode.leaf)
        currentNode = currentNode.findChild(x) // finds the node that matches x
    prectedValue = currentNode.leafValue
```

I used two methods to combat overfitting of the data, one for classification sets and another for regression sets. I used early stopping to reduce overfitting on regression sets. Early stopping uses an error threshold to determine if the current node is good enough and should be a leaf node. This prevents the tree from growing too large and overfitting data.

I used pruning to reduce overfitting in classification trees. Pruning is done after the tree is grown. In pruning, the tree is recursively traversed, and each node is flagged for pruning. The node flagged for pruning is turned into a leaf with the value at the leaf the most common value in the data at that node. Then the performance of the pruned tree is evaluated using a pruning data set. If the performance of the pruned tree is no worse than the performance of the current best tree, the node is pruned.

Both pruning and early stopping require extra data. I removed 20% of the original data for both sets. This reduces the training power of the remaining data because a large chunk of the data has been removed. I used the remaining 80% to build and test trees using cross-validation.

I used 5-fold cross-validation when tuning and testing the data. In 5-fold cross validation the data set is split into 5 different folds. When building decision trees, a tree is grown with data from four of the folds. Then the data from the remaining fold is evaluated against the newly built tree and a prediction is made. This process is repeated until every fold has had values predicted. In this way, every point in the data set is tested against a large set of remaining values. It was important to stratify label values across each of fold to ensure a roughly equal distribution of labels in each fold. This ensures values from a single bin have several similar points in the other 4 bins.

Tuning

As mentioned above, I used early stopping to reduce overfitting regression datasets. Early stopping requires an error threshold and this hyperparameter value needs tuning. Because there is only one hyperparameter, it is relatively straightforward to find good values for the error threshold. I simply tried many different error values and used the one with the best performance. I made sure to scale the error thresholds to make sure I covered a large range of possible values. The performance of different error threshold values for all three data sets is shown below.

Error Threshold	Abalone Performance	Computers Performance	Forest Fires Performance
0	122.26	39137.95	22195.5
5	74.92	39009.38	22194

10	23.78	38880.81	22194.14
20	10.48	38648.65	22189.53
50	10.48	38488.13	22142
100	10.48	37846.74	22124.46
1000	10.48	38402.93	21839.46
10000	10.48	29210.12	5807.25

Table 1. Tuned Error Threshold

Datasets

I used six different data sets in my experiments. These data sets can be broken down into two categories: classification and regression. Classification data sets classify input points into discrete buckets or categories. Regression data sets, on the other hand, have output labels that are continuous numeric values.

Breast Cancer Dataset

The breast cancer data uses categorical values for several fields, including clump thickness and tumor diagnosis. All the category fields are ordinal, so it was relatively straightforward to load the categories into my data structure. Each attribute is ordinal from 1-10 and there are two classes 2 and 4, mapping to benign and malignant. There were a handful of missing values in the dataset. I replace the missing values with the average value of the attribute across the dataset. I chose to remove the attribute for ID number. The ID number was randomly assigned to entries. It therefore could only improve predictions by overfitting the data.

Car Evaluation Dataset

The car evaluation dataset is a classification set and estimates the overall car acceptability given several different attributes of the used car. These attributes include fields such as price paid for maintenance and the number of doors. Like the breast cancer data, all the attributes are ordinal. I mapped each to integers ranging from 0 to the number of different values – 1. For example, one attribute for luggage boot contains values of small, medium, and large. These values are mapped to 0, 1, and 2. The class field is also ordinal and car acceptability ranges from unacceptable (0) to very good (3). There are no missing values in the car dataset.

Congressional Voting Dataset

The congressional vote data is data relating to 16 different votes in congress. The class value is the party affiliation of the congress person (Republican or Democrat). All attributes are yes, no binary values representing yay or nay. I chose to encode these attributes as a simple 1/0 for yes and no. I did the same for the class where I assigned 1 and 0 to the different parties. Again, there are a handful of missing values in the dataset. These are not true missing values, but rather votes that were not taken by a particular member of congress. I chose to encode them with the average of the attribute to help smooth the predicted value.

Abalone Dataset

The first regression dataset is data related to Abalone size. The class of the dataset is ring count, which directly maps to the age of the Abalone. There are several continuous attributes including length, diameter, and weight. The most interesting attribute is the sex. The values include M, F, and I for infant. Since these values are nominal, I chose to use one-hot encoding. I created a total of three fields from the one sex input. Each value is mapped to a field and is assigned 1 when the value in the input field equals the column. For example, if an input had a value of M, the three columns (M, F, and I) would be 1, 0, 0. There are no missing values in the Abalone dataset.

Computer Hardware Performance Dataset
The second regression dataset is computer hardware speed. The class variable for the data is the estimated speed as calculated by a performance study. I chose to drop the vendor and model number from the dataset because they should have no impact on the computer performance. The model number is unique to each field and therefore will have no power to fit the data. There are also 30 different vendors representing 209 instances. I worry one-hot encoding would create too many attributes that overfit the data. The remaining attributes are continuous, and I was able to load directly into my data structure. There are no missing values in the computer dataset.

Forest Fire Dataset
The final dataset is forest fire data. This dataset represents the area of forest fires based on 12 different attributes. The most interesting attributes are month and day. As these are ordinal data, I chose to map the strings to 0-11 for months and 0-6 for days. The remaining attributes are continuous, and I added them directly to my data structure. There are no missing values in the forest fire dataset.

3. Results

I calculated the performance of the algorithm with different methods for the regression and classification data sets. The performance of the three classification datasets were calculated with a simple majority algorithm. In these three cases, the prediction score is simply the number of correct predictions divided by the total number of instances. The performance of the regression data sets was calculated with a mean squared formula. The total score is the mean squared difference between the actual value and predicted value of each point. Based on these two methods, large scores in the classification data indicates better performance and smaller scores in the regression sets are better. The table below presents all results for the three different algorithms.

	Decision Tree	**Pruned Decision Tree**
Breast Cancer	0.937	0.912
Cars	0.727	0.818
House Votes	0.951	0.963
Abalone	216.99	10.36 (threshold = 20)
Computer Performance	390,999.90	389,353.73 (threshold = 10,000)
Forest Fires	9738.21	3635.26 (threshold = 10,000)

Table 2. Experiment Results

4. Discussion

The ID3 algorithm performed well and accurately predicted values for the three classification data sets. Comparing ID3 with kNN from project 2, we can see a Decision Trees performed better in two of the three data sets. It is interesting to compare two very different algorithms together to see that both predict the data well. ID3 prioritizes attributes that have the largest impact in predicting class labels. This lends itself to datasets where there are a few powerful attributes while other attributes are used for fine-tuning. On the other hand, kNN treats all attributes identically, which lends itself to datasets with attributes that predict the class label relatively equally. My guess is the Breast Cancer dataset has attributes that are relatively equal in their power to predict the class label, while cars and house votes have more powerful attributes where decision trees prioritized splits.

Additionally, the CART algorithm predicted values for the three regression data sets. However, I did not find the predictive power to be very good. When tuning the error threshold, I found that larger error thresholds always improved the accuracy of the algorithm. This shows to me that the mean value of the data set is a better predictor than the trained decision trees. I am wondering if the datasets are difficult to predict and the class value is not strongly correlated with each attribute.

	kNN	**Decision Tree with Pruning**
Breast Cancer	0.9588	0.912
Cars	0.7539	0.818
House Votes	0.9454	0.963

Table 3. Classification Results in kNN and Decision Trees

Pruning and early stopping improved the performance on five of the six datasets. This proves trees fully grown tend to overfit the data and this reduces their prediction power with new data. Pruning and early stopping help with overfitting by removing nodes that don’t help improve accuracy of the tree. In my opinion, pruning should always be evaluated when using decision trees with ID3 or CART.

5. Conclusion

In this project, I implemented the ID3 and CART algorithms to produce Decision Trees. I learned how trees grow, along with algorithms for entropy, information gain, gain ratio, and mean squared error. Additionally, I learned how pruning can help improve the predictive power of the algorithms by reducing overfitting. It was great to see how the performance improved as the trees were pruned!

In the future, I would like to evaluate more regression data sets. I did not feel like CART did a great job predicting values for the three data sets used. I would like to evaluate with more data to determine why the algorithm fell short and what improvements could be made to improve it.

References

Alpaydin, Ethem. *Introduction to Machine Learning, Fourth Edition (Adaptive Computation and Machine Learning Series)*. Fourth edition, The MIT Press, 2020.