Clustered Decision Trees

Team Members: Justin Lee, Abhi Palikala

1/15/2025

Table of Contents

Part 1 – Abstract

Classification models like decision trees utilize a heuristic in order to determine what features to split by at each step. Common heuristics include information gain, gain ratio, and Gini impurity. While these are commonly used due to their simplicity and quick computation, they fail to consider an important aspect when determining where to split: the purity of future splits. To alleviate this issue, we create an intuitive classification by running K-means clustering and calculate the weighted entropy of the various clusters. In this report, we explore the advantages and compromises of this approach.

Part 2 – Introduction

Decision trees are a popular, simple method of creating a set of classification rules for a dataset by splitting it into subsets based on its feature values. They are an example of inductive inference, using previous data to extrapolate to future instances. Decision trees are favored for their ease of implementation and solid predictive performance. They're used to make predictions in fields ranging from business to healthcare.

The main algorithm behind the success of the decision tree is the heuristic it uses to select the features for splitting. Some popular heuristics include information gain, gain ratio, and Gini impurity, each of which attempt to quantify and determine how good a potential 'splitting' is. However, a common issue that these heuristics run into is that they act greedily, maximizing the immediate next step rather than considering how it may help in the future.

This is the issue we hope to address. We compare the performance of our novel algorithm with several established heuristics to analyze the benefits and costs over several different datasets. The model input takes in various continuous variables and uses a decision tree to output a (potentially) nominal classification.

Part 3 – Related Work

The most common approach has been to adjust the purity function used in calculating information gain. For instance, some works have explored replacing the weighted logarithm in the standard entropy calculation with alternative mathematical forms, such as power means or the square root, to better capture class distributions [1, 2]. While these methods are computationally efficient and easy to implement, they often exhibit limitations when applied to datasets with significant class imbalance or noise. This highlights a trade-off between simplicity and robustness in practical scenarios.

A second category of approaches generalizes the information gain formula by introducing tunable parameters or alternative definitions of entropy. For instance, methods based on Tsallis entropy use non-logarithmic metrics, such as powermeans or square roots [3]. These generalizations often require optimization of additional parameters, which can make implementation more complex. Nonetheless, they offer a theoretical framework to balance between overfitting and underfitting, making them appealing for certain applications.

Finally, there have been attempts to move beyond single-feature splits by evaluating higher-order splitting criteria. These methods, including those based on joint entropy or conditional mutual information [4], aim to capture interactions between attributes that are often missed in traditional heuristics. Although computationally expensive, these techniques align closely with our work and show promise in improving decision tree accuracy for high-dimensional data.

Among these approaches, modifications to the purity function are the simplest to implement and are often computationally efficient. However, they may not significantly improve tree performance in all cases. Generalized entropy measures offer theoretical flexibility but introduce challenges in parameter tuning and interpretation. Higher-order splitting criteria are arguably the most innovative but can be computationally difficult for large datasets.

The current state-of-the-art often involves combining these approaches to achieve a balance between computational efficiency and predictive accuracy. For example, methods that adaptively select between different entropy measures or incorporate feature interactions without significantly increasing complexity have gained attention. Despite these advancements, many practitioners still rely on traditional heuristics, such as information gain or gain ratio, due to their simplicity and widespread implementation in machine learning libraries.

Part 4 – Dataset and Features

The dataset we used for testing was taken from cases of credit card fraud. Among more balanced datasets, most heuristics would perform similarly. So it was purposely chosen to be extremely unbalanced, where the differences between our ClusteredInfo heuristic and the traditional InfoGain heuristic could be amplified and observed. The dataset was also chosen for its complexity, as it has thirty continuous attributes, which amplifies the need to consider future best splittings in addition to just the current best split. The original dataset consisted of over 200,000 instances, from which we chose a representative subset of 10000 instances split among two classes, 38 instances of class value 1 (credit card fraud) and 9962 instances of class value 0 (no fraud). This data was split into training and testing using an 80-20 split, resulting in 8000 training instances and 2000 testing instances. We verified afterwards that the unbalanced-ness of the main dataset was also observed in the train and test datasets. The original dataset, the train and the test dataset are all attached alongside this report.

As part of preprocessing, it was necessary to include discrete and continuous data for our algorithm, as decision trees require discrete data and K-means clustering requires continuous data. Therefore, when preprocessing our data, we discretized each feature, and kept those discrete attributes along with the old continuous ones. For discretizing our data we used equal 3 width bins for each attribute. The width of each bin was calculated by finding the min and max values within the attribute and creating equal width bins along that range. Thus, our modified dataset contains 60 attributes, one copy of the original continuous attribute and one that is discretized. The direct meaning of each of these attributes is not directly apparent (ie, they are not attributes like “swip-time” or “number of swipes”) as Kaggle describes the values as results of a PCA analysis of several other attributes. Nonetheless, that does not change the applicability of our project.

Part 5 – Methods

The clustered decision tree algorithm works similarly to a normal decision tree. Attributes are selected to split based on minimizing a heuristic we define. In this case, instead of Info, we used ClusteredInfo.

Info can be expressed in the following equation:

$$Info \;=\; -\sum_{D_i}^{D}\frac{|D_i|}{|D|}\sum_{C_{i,j}}^{C_i}\frac{|C_{i,j}|}{|C_i|}log_2(\frac{|C_{i,j}|}{|C_i|})$$

Where D is all the instances before being split by an attribute and D_i is the instances associated with the i’th node after branching off of an attribute split. In a similar fashion, $C_{i,j}$ is the j’th class label associated with the i’th node. In the case of most of the datasets we worked with this year, it would just be a simple binary classification task, so j would only take a value of 0 or 1 representing two class labels.

ClusteredInfo can be defined as

$$ClusteredInfo \;=\; -\sum_{D_i}^{D}\frac{|D_i|}{|D|}\sum_{K_{i,j}}^{K_i}\frac{|K_{i,j}|}{|K_i|}\sum_{C_{i,j,k}}^{C_{i,j}}\frac{|C_{i,j,k}|}{|C_{i,j}|}log_2(\frac{|C_{i,j,k}|}{|C_{i,j}|})$$

Where D and C carry the same meaning as the Info formula. We introduce a new variable K to represent a K-means cluster after splitting the node’s data using the K-means algorithm. In this new formula, D_i is the instances associated with the i’th node after splitting using an attribute, $K_{i,j}$ is the j’th cluster in the i’th node, and $C_{i,j,k}$ is the k’th class label within the j’th cluster of the i’th node.

Essentially, before calculating the class label based entropy on each node of our decision tree, we first cluster the tree with the k-means algorithms. This acts as our heuristic for the future predictive power of our data. Normal decision trees act on a completely greedy approach, maximizing the instantaneous predictive power of the data while failing to consider how predictive the rest of our data will be. By utilizing K-means, we add an additional ability for our model to consider the future predictive power. Even seemingly impure data can be considered purer if data within each individual cluster is pure (represented by a low entropy), as it showcases that the data has high potential that future splitting will be successful. For example, consider the two simple datasets with 1 feature remaining after splitting on a previous attribute. [(1, 1), (1, 1), (0, 0) (0, 0)] and [(1, 0), (0, 1), (1, 1), (0, 0)]. Where the first element of each tuple represents the feature value and the second element of each tuple the class label. When utilizing normal InfoGain, both of these dataset have the same entropy of 1 for a completely impure state. However, these datasets are clearly not equal. In the first dataset the first attribute correlates absolutely with the feature, while the second dataset has zero correlation. If we decided to cluster the dataset first, we would see that each cluster of the first dataset is pure, while the clusters of the second dataset will still remain impure. Thus, we can see where our algorithm can improve datasets: where a greedy approach is not sophisticated enough.

Part 6 – Results

Because our model is not completely greedy like a normal decision tree, it is more effective than a normal decision tree on more complicated datasets, where there is more correlation between features and the splitting heuristic matters more. The dataset we decided to test it on was a case where typical decision trees perform poorly: extremely unbalanced datasets. The dataset we picked was one on credit card fraud, which has substantially more negative instances compared to positive instances. Fraud transactions make up only 0.172% of all transactions. For the K-means clustering, we decided to choose a value of K=2, as that seemed to have the best results on both the credit card dataset and the diabetes dataset. Other values of K tended to select less optimal features and result in poorer performing decision trees.

Going into analyzing our model performance we used the sensitivity metric, as it better captured the nuance of our dataset, since in the context of credit fraud, the worst possible prediction is a false negative, since it would be better to be overly cautious and wrongly investigate a false positive than to entirely miss a case of credit card fraud. Sensitivity can be defined as the following:

$$Sensitivity = \frac{TP}{TP + FN}$$

Below are the confusion matrices after training for our clustered decision tree as well as a benchmark with scikitlearn’s decision tree implementation.

```
##### Clustered Decision Tree Train #####
Accuracy: 99.99%
Confusion Matrix:
        Actual (1) Actual (0)
Pred (1) | 024        000
Pred (0) | 001        7975

##### Clustered Decision Tree Test #####
Accuracy: 99.85%
Confusion Matrix:
        Actual (1) Actual (0)
Pred (1) | 010        000
Pred (0) | 003        1987

##### Regular Decision Tree Train #####
Accuracy: 99.99%
Confusion Matrix:
        Actual (1) Actual (0)
Pred (1) | 024        000
Pred (0) | 001        7975

##### Regular Decision Tree Test #####
Accuracy: 99.25%
Confusion Matrix:
        Actual (1) Actual (0)
Pred (1) | 005        007
Pred (0) | 008        1980
```

The sensitivity of our clustered decision tree on the test dataset is 10 / (10 + 3) = 0.77, while the sensitivity of scikitlearn's decision tree is 5 / (5 + 8) = 0.39. In this way, we can see how taking into account the future predictive power of our dataset can help for building models that perform better on complicated, unbalanced datasets.

Part 7 – Conclusion

In this paper, we presented a novel method for decision tree splitting, ClusteredInfo, which we have designed to alleviate the nearsightedness of other algorithms. By combining the K-means clustering algorithm within the more traditional entropy formula, we were able to consider the future predictive ability of potential splits. Testing this heuristic on the highly unbalanced credit card fraud dataset revealed that our algorithm achieved a sensitivity of 0.77 on the testing data, performing almost twice as well as the Infogain heuristic, which had a sensitivity of 0.39. The significant improvement in performance by the clustered decision tree (CDT) can be attributed to the importance of accounting for non-greedy splitting techniques, one's that take future splitting value and correlation between attributes into account.

In the future, further research could explore the performance of CDTs on various other datasets and distributions, particularly those that perform poorly under traditional heuristics such as InfoGain. Additionally, different methods for optimizing the K-value within the K-means task of a CDT could also yield improvements. Optimizing speed would also be a priority, as running K-means at every step is expensive. It is likely possible to cache optimal clusters between runs to speed this up. Finally, utilizing a hybrid method that combines ClusteredInfo with additional heuristics, such as Gini impurity, may help us further improve computational efficiency and our predictive accuracy.

Part 8 – Contributions

Justin Lee focused primarily on writing and testing the Clustered Decision Tree algorithm and dataset preprocessing. Abhi Palikala focused primarily on delving into previous research and interpreting the results of the algorithm in order to write the analysis. Both authors have come together to cross-check their results and writing and contributed equally to this paper.

Part 9 – References

Bahnsen, A. C., Aouada, D., & Ottersten, B. (2015). Example-dependent cost-sensitive decision trees. arXiv preprint arXiv:1509.07266.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (2012). Classification and regression trees. Microsoft Research. Retrieved from
https://www.microsoft.com/en-us/research/wp-content/uploads/2012/06/1206.4620.pdf

Dal Pozzolo, A., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. arXiv preprint arXiv:1511.08136.
Dugas-Phocion, G., & Bengio, S. (2020). Learning optimal decision trees using reinforcement learning heuristics. arXiv preprint arXiv:2010.08633.

Kaggle. (n.d.). Credit card fraud detection dataset. Retrieved from
https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Keylabs AI. (n.d.). Decision trees: How they work and practical examples. Retrieved from https://keylabs.ai/blog/decision-trees-how-they-work-and-practical-examples/

Lomax, S., & Vadera, S. (2013). A survey of cost-sensitive decision tree induction algorithms. Knowledge Engineering Review, 28(2), 159–188. Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC2701298/

Vation Ventures. (n.d.). Decision trees: Definition, explanation, and use cases. Retrieved from https://www.vationventures.com/glossary/decision-trees-definition-explanation-and-use-cases

Zhang, Y., & Wu, J. (2018). A survey on decision tree learning for heterogeneous data. arXiv preprint arXiv:1801.08310.