Eric Aschari, Chenle Chen, Chelsey Tao, Minghui Zhu
David Demeter
CS349 Machine Learning
Thursday, October 17th

CS349 Homework #1: Decision trees

1. **Did you alter the Node data structure? If so, how and why?**

Yes, we altered the Node data structure. We added two fields: attribute and is_leaf and introduced a helper method: add_child.

The attribute field stores the feature or decision attribute associated with the node, which helps in decision-making during the tree traversal. The is_leaf field is a boolean that indicates whether the node is a leaf node, which helps when checking when to stop traversing the tree. The add_child function is a helper method that simplifies the process of adding children nodes to a parent when building the decision tree.

2. **How did you handle missing attributes, and why did you choose this strategy?**

To handle missing attributes in the dataset, we used an imputation strategy where missing values (denoted by ?) are replaced by the most common value for the corresponding attribute. We implemented this in our code using the preprocess function, which can operate in different modes: 'remove', 'impute', and 'keep'.

We decided to adopt the imputation strategy. For each attribute with missing values, we calculate the most common value across all examples. We then replace these values with the most common value. We chose this strategy because removing examples with missing attributes can lead to loss of data, which may result in fewer training examples. This method maintains the size of the dataset. By imputing the most common value in the corresponding attribute column, we preserve the distribution and make reasonable assumptions about the missing values while adding minimum bias to the dataset.

3. **How did you perform pruning, and why did you choose this strategy?**

We implemented error complexity pruning, a post-pruning technique that balances model complexity with performance. The method begins by fully constructing the decision tree to its full depth, then pruning the tree by removing branches and evaluating the pruned tree's accuracy on a validation set (independent from training).

After the tree is built, we begin pruning from the leaf nodes (backtracking). For each internal node, we calculate the error rate for the subtree starting from that node and compare it to the error rate if the node were replaced by a leaf node (the most common class label of the subtree). If pruning the node improves or maintains accuracy on the validation set, we replace the subtree with the single leaf node.

The algorithm generates several pruned versions of the original tree, with varying degrees of simplification. The version of the tree that performs best on the validation set is selected as the final model to ensure the generalization ability of the tree.

We chose this strategy because error complexity pruning helps prevent overfitting by reducing the complexity of the tree while maintaining good performance on unseen data. The method evaluates the pruned versions of the tree on a validation set, which ensures that the chosen tree generalizes well to independent data from the training set. This is crucial for real-world scenarios where we aim to minimize errors on data outside of the training set.

Compared to other strategies, like critical value pruning, error complexity pruning is also more flexible. It allows for the exploration of various tree depths and sizes, finding the version that balances accuracy and simplicity.

4. **Now you will try your learner on the house_votes_84.data, and plot learning curves. Specifically, you should experiment under two settings: with pruning, and without pruning. Use training set sizes ranging between 10 and 300 examples. For each training size you choose, perform 100 random runs, for each run testing on all examples not used for training (see testPruningOnHouseData from unit_tests.py for one example of this). Plot the average accuracy of the 100 runs as one point on a learning curve (x-axis = number of training examples, y-axis = accuracy on test data). Connect the points to show one line representing accuracy with pruning, the other without.**

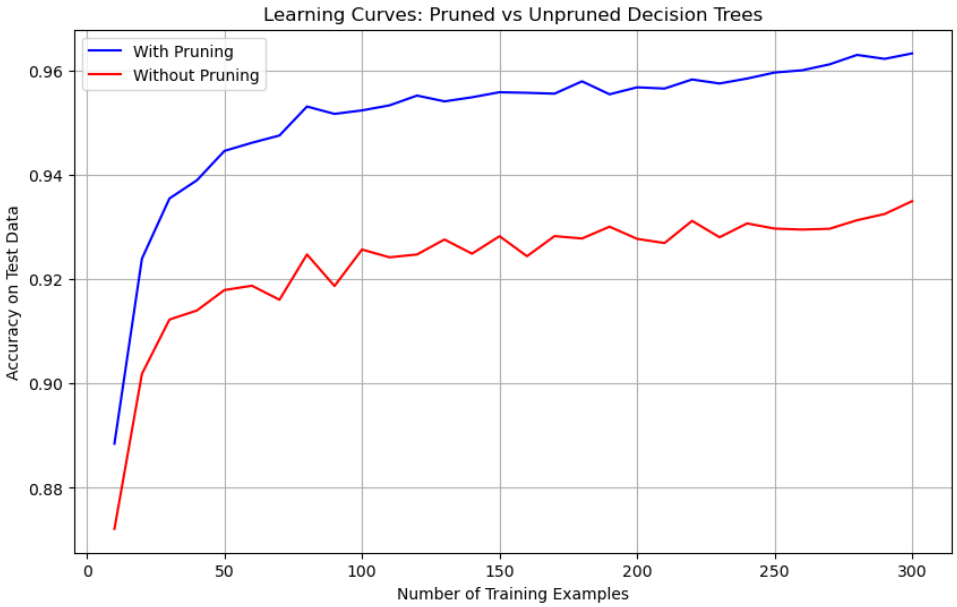

Figure 1: Learning Curves Plot

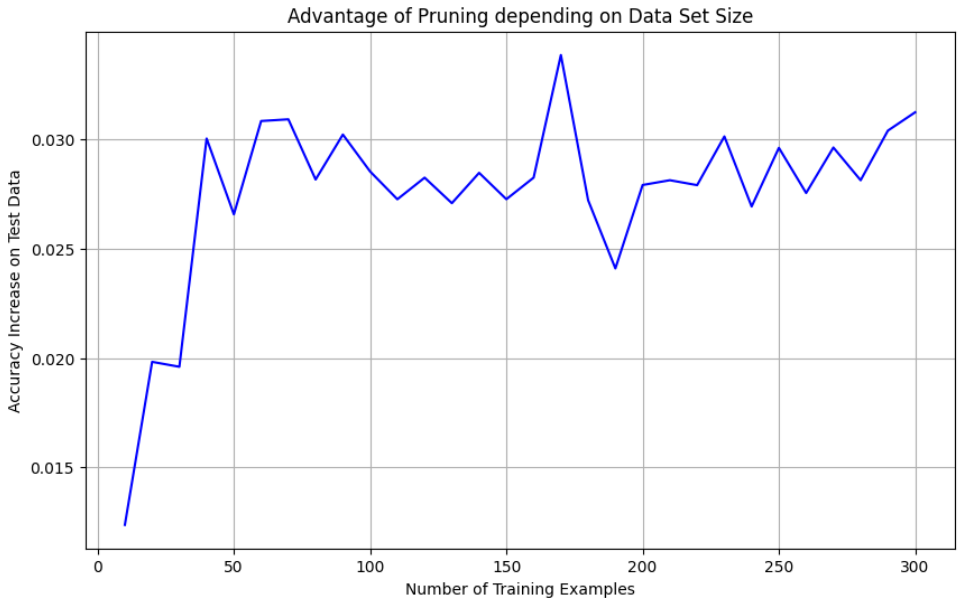

Figure 2: Gap between pruned and unpruned tree

a. What is the general trend of both lines as training set size increases, and why does this make sense?

Looking at figure 1, both lines show an increasing trend as the number of training examples increases. The accuracy for both pruned and unpruned decision trees improves with more training data, which is expected since larger training sets provide more information, allowing the decision tree to make better predictions. This makes sense because as the number of training examples increases, the model has more data to learn from, which leads to more accurate generalizations and fewer errors. In addition, the pruned tree outperforms the unpruned tree in terms of accuracy, which makes sense because pruning the tree helps to simplify the model, reduce overfitting, and also reduce errors.

b. How does the advantage of pruning change as the dataset size increases? Does this make sense, and why or why not?

The advantage is more prominent when the size of the dataset is small. This is shown in figure 2, as the accuracy with pruning for smaller datasets increases more rapidly compared to the unpruned tree. From small dataset sizes, the gap between pruning and not pruning increases until a size of around 50. The increase in accuracy slowly decreases and plateaus for pruning and unpruned as the number of training examples increases, the gap between the trees also stays similar. This makes sense because pruning helps to prevent overfitting which is a bigger risk for smaller datasets, as it is harder to generalize. Yet, as the dataset size increases, both the pruned and unpruned trees have enough data to improve accuracy, so the advantage of pruning diminishes.

5. **Use your ID3 code to learn a decision tree on `cars_train.data`. Report accuracy on the `cars_train.data`, `cars_valid.data` and `cars_test.data` datasets. If your accuracy on `cars_train.data` is less than 100%, explain how this can happen. Prune the decision tree learned on `cars_train.data` using `cars_valid.data`. Run your pruned decision tree on `cars_test.data`, and explain the resulting accuracy on all three datasets.**

Accuracy on training set before pruning: 1.0
Accuracy on the training set after pruning: 0.845

Accuracy on validation set before pruning: 0.8
Accuracy on validation set after pruning: 0.8571428571428571

Test accuracy before pruning: 0.6285714285714286
Test accuracy after pruning: 0.8

The accuracy of the training set is less than 100 percent, which shows that the decision tree does not perfectly fit the training data. This happens because the available features are not sufficient to fully capture the patterns in the training data, or there may be inconsistencies or noise in the data.

The pruning of the tree using the validation set (cars_valid.data) reduced the complexity and overfitting of the decision tree. Pruning allows the decision tree to be more generalized when classifying unseen data and unknown datasets. Therefore, the classification accuracy for the training set decreased, but the classification accuracy for the validation and testing set improved. This is a trade-off between model complexity and generalization. The pruned tree performs better on validation and test data, showing that it is no longer overfitting to the training data.

6. **Use your ID3 code to construct a Random Forest classifier using the `candy.data` dataset. You can construct any number of random trees using methods of your choosing. Justify your design choices and compare results to a single decision tree constructed using the ID3 algorithm.**

Accuracy on training candy.data with single tree with no pruning: 0.8928571428571429
Accuracy on training candy.data with random forest: 0.875

Accuracy on testing candy.data with a single tree with no pruning: 0.6842105263157895
Accuracy on testing candy.data with random forest: 0.7192982456140351

When constructing our Random Forest classifier, we created multiple (100) decision trees by running the ID3 algorithm on different subsets of the training data. We chose 100 trees because after observing a random forest with 50 decision trees, the random forest does not always outperform the single tree, so by increasing the number of trees, we increase the probability that the forest performs better. By randomly sampling both examples and attributes from the original dataset, we create each subset. The random sampling of subsets helps introduce diversity among

the trees, reducing bias and overfitting. A minimum size of 30 examples and 5 attributes for each subset is also chosen to ensure that there is sufficient data to train each tree.

When using the training data, we observed that a single decision tree achieved higher accuracy. However, when evaluating the test data, the Random Forest outperformed the single tree. This is because the performance of a single decision tree is better on the training data as it may be overfitting, so it can capture specific patterns that do not generalize as well to unseen data. On the other hand, the Random Forest reduces overfitting by averaging out the individual trees' predictions among the diverse collection of trees. This allows the forest model to be more generalized and perform better on the test set which was not used for training.

Nonetheless, there may be a limited improvement in accuracy due to the possible lack of diversity and small dataset sizes. The improvement with the Random Forest depends largely on the diversity among the trees, so with limitations in the dataset size, the forest may show similar performance to a single tree.