Decision Tree Coursework

Report

Patrick Chen and Hantang Sun

Introduction to Machine Learning

COMP 70050
Department of Computing
Imperial College London

1 Implementation Details

1.1 Decision Tree Representation

We start by defining a decision tree class with the following attributes:

```
class Decision_tree:
    def __init__(self, feature_index, threshold, left_sub_tree, right_sub
                 label = -1, majority_label = -1):
        self.__feature_index = feature_index
        self.__threshold = threshold
        self.__left_sub_tree = left_sub_tree
        self.__right_sub_tree = right_sub_tree
        self.__label = label
        self.__majority_label = majority_label
```

The feature index is the feature that we are using to split the tree into sub-trees. The values of the selected features are compared to the threshold value. If they are smaller than the threshold, the search will proceed into the left sub-tree, otherwise the right.
The label is used to determine whether or not we have reached a leaf node. If the label is -1, we are at an internal node. Otherwise, the node is a leaf. The field majority_label is used for pruning and will be explained later in this report.

1.2 Building the Decision Tree

To build the tree, we start by choosing the feature that we would like to split on, represented by the feature index attribute. We want to choose the feature that gives the most information gain, i.e. that minimises the weighted average entropy of the sub-trees.
To do this, we iterate through all the features. For each one, we split the dataset in half and calculate the weighted average entropy of the two sub-sets. We find the feature that gives us the smallest average entropy value, and return its feature index, threshold and sub-trees. As this feature gives the smallest average entropy, it also gives the smallest remainder, resulting in the largest information gain.
After we split the dataset into two subsets, we use each one to build a subtree. From the subsets, we then recursively build further subtrees until every label in the subsets is identical.

1.3 Evaluation

We use 10-fold cross validation to evaluate our model. We split our dataset into ten partitions, nine for training, and one for testing. Once the tree is built, we use our test dataset to build the confusion matrix. From the matrix we calculate the overall accuracy, as well as the precision, recall and f1 for each class label. Afterwards, we repeat this process with each partition as the testing set, and average the values throughout all ten iterations.

2 Evaluation Results

Experiment on clean dataset

tree depth : 12

confusion matrix: $\begin{bmatrix} 49.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 47.7 & 2.3 & 0.0 \\ 0.3 & 3.1 & 46.4 & 0.2 \\ 0.2 & 0.0 & 0.7 & 49.1 \end{bmatrix}$

precision for the four classes : [0.996, 0.990, 0.938, 0.930]
recall for the four classes : [0.983, 0.990, 0.954, 0.929]
f1 for the four classes : [0.989, 0.990, 0.946, 0.929]
accuracy : 0.964

Experiment on clean dataset after pruning

tree depth : 10

confusion matrix: $\begin{bmatrix} 49.1 & 0.0 & 0.4 & 0.5 \\ 0.0 & 46.3 & 3.7 & 0.0 \\ 0.3 & 1.3 & 47.9 & 0.5 \\ 0.3 & 0.0 & 0.9 & 48.8 \end{bmatrix}$

precision for the four classes : [0.979, 0.989, 0.973, 0.906]
recall for the four classes : [0.977, 0.982, 0.928, 0.958]
f1 for the four classes : [0.978, 0.985, 0.950, 0.930]
accuracy : 0.961

Experiment on noisy dataset

tree depth : 12

confusion matrix: $\begin{bmatrix} 38.5 & 2.5 & 3.3 & 4.7 \\ 2.6 & 40.8 & 4.7 & 1.6 \\ 2.4 & 3.7 & 42.4 & 3.0 \\ 3.9 & 2.0 & 3.6 & 40.3 \end{bmatrix}$

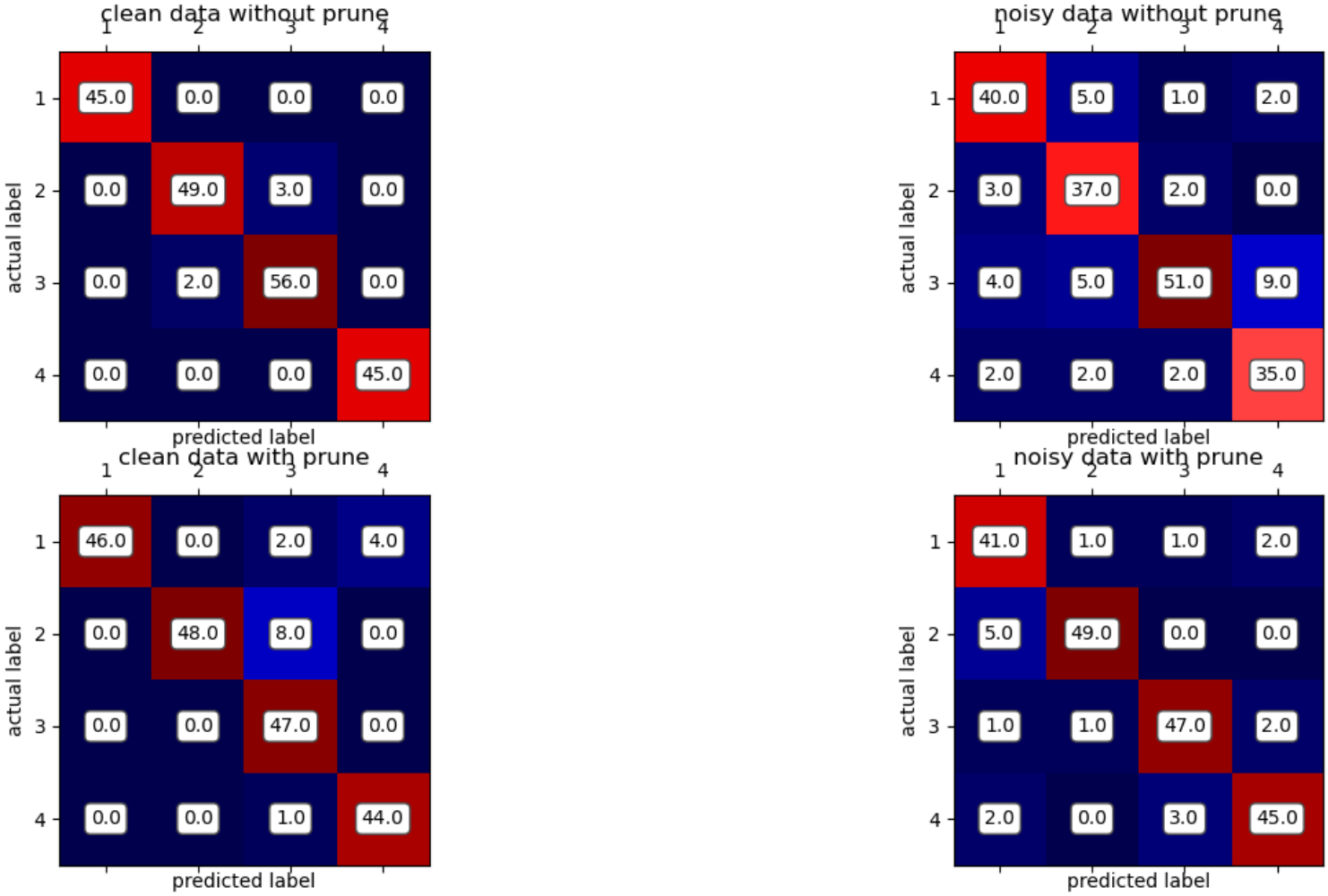
precision for the four classes : [0.817, 0.810, 0.833, 0.785]
recall for the four classes : [0.810, 0.779, 0.817, 0.827]
f1 for the four classes : [0.811, 0.793, 0.823, 0.802]
accuracy : 0.810

Experiment on noisy dataset after pruning

tree depth : 10

confusion matrix: $\begin{bmatrix} 44.0 & 1.0 & 1.5 & 2.5 \\ 1.8 & 43.6 & 3.2 & 1.1 \\ 2.0 & 3.3 & 43.9 & 2.3 \\ 2.1 & 1.4 & 1.9 & 44.4 \end{bmatrix}$

precision for the four classes : [0.887, 0.883, 0.886, 0.868]
recall for the four classes : [0.893, 0.897, 0.877, 0.853]
f1 for the four classes : [0.889, 0.890, 0.881, 0.859]
accuracy : 0.879


Confusion matrices for the experiments
clean data without prune
actual label
predicted label
1 2 3 4
45.0 0.0 0.0 0.0
0.0 49.0 3.0 0.0
0.0 2.0 56.0 0.0
0.0 0.0 0.0 45.0
noisy data without prune
40.0 5.0 1.0 2.0
3.0 37.0 2.0 0.0
4.0 5.0 51.0 9.0
2.0 2.0 2.0 35.0
clean data with prune
46.0 0.0 2.0 4.0
0.0 48.0 8.0 0.0
0.0 0.0 47.0 0.0
0.0 0.0 1.0 44.0
noisy data with prune
41.0 1.0 1.0 2.0
5.0 49.0 0.0 0.0
1.0 1.0 47.0 2.0
2.0 0.0 3.0 45.0

3 Questions

3.1 Noisy-clean datasets

The accuracy of the model in the clean dataset is approximately 96 percent, whereas the accuracy for the noisy dataset is only 81 percent. This shows that the model's error rate using the noisy dataset is approximately 5 times higher than when we use the clean dataset, from which we can conclude the overall performance is far greater when using the clean dataset.
The model performs worse when predicting data in class 4 in the clean dataset, compared to the other classes. In the noisy dataset, the precison, recall and f1 decrease for every class. However, the performance is more consistent between each class than in the clean dataset.
This change is caused by noisy data containing corruption and many anomalies. Therefore, when we use the noisy data to train the tree, the tree created will overfit the data as it is modelling randomness in the training set. An overfitting model will then lead to many false conclusions, decreasing the accuracy. Also, noise exists in the test set, meaning that even a robust model will under-perform in the test.

3.2 Pruning

During the construction of the tree, we store the majority label in each non-leaf node. Now when we carry out 10-fold cross validation, we divide our dataset into a testing set, a validation set and we use the remaining eight partitions for the training set.
During the pruning process, we start at the root node and divide the validation set into two subsets, based on their features and the threshold stored in the tree node. We continue splitting the set until we have reached a node with only leaves as its children. Then, we use the validation set to calculate the accuracy of the unpruned tree. Afterwards, we replace the internal node with a leaf with the majority label as the label, and calculate the accuracy using the validation set again. We compare the unpruned and pruned accuracy to determine whether or not we should prune the node. We repeat the process for its parents until we get back to the root node, ensuring that after we prune the child, we can prune the parent.
Pruning the model trained by the clean dataset had little to no effect on its accuracy, as it decreased from 96.4 percent to just 96.1 percent. However, for the noisy dataset, the accuracy significantly increased from 81.0 percent to 87.9 percent.
Pruning is used to reduce overfitting in the model. When we trained the model using the clean dataset, it resulted in little overfitting so pruning had no significant effect on the model's performance.
However, when we trained the model using the noisy dataset, it resulted in a tree that heavily overfit due to excessive noise in the dataset. Pruning reduces the overfitting and thus increases the performance of the model.

3.3 Depth of the decision tree

Pruning both trees reduced the maximal tree depth from 12 before pruning to 10 after pruning. For the model trained by the clean dataset, there is no correlation between the tree depth and accuracy, because when we decreased the tree depth, the accuracy had no significant change. However, for the model trained by the noisy dataset, when we decreased the depth, the accuracy significantly increased.
This is because reducing the tree depth reduces the model's susceptibility to overfitting as it allows the tree to better generalise data with fewer decisions. Overfitting is a much more significant problem when we train a model using a noisy dataset, as oppose to a clean dataset. Therefore, decreasing the maximal depth can improve accuracy if the model is overfitting.
A tree with low depth can easily lead to underfitting, whereas a tree with high depth can easily lead to overfitting. Therefore, to optimise performance, the decision tree should have a depth that is a good compromise. The suitable depth will depend on the problem and the noisiness of the dataset.