Linxi Fan

Part II (a) Add pruning

The chi-square pruning is implemented. The test on the simple two-node set runs as expected.

The pruning is done in a bottom-up manner: first prune the children subtrees, and then look at the parent node itself.

(b) Add node count

The `count_node()` method counts the nodes recursively. Works as expected.

Part III

The generated code conforms to python syntax and works as expected.

Part IV

The following table is generated by my modified script `testDecisionTree.py`

`maxdev`: max deviation, different hyperparameters used by all the trial algorithms

`F1`: macro F1 score on the validation set after pruning.

`#nodes`: macro (average) number of nodes after pruning.

I tried `maxdev` values from 0.1 to 4.9. The table below shows the corresponding F1 scores on validation set and the number of nodes for each `maxdev`. The best `maxdev` is around `2.00`, which yields an F1 score as high as 0.88.

On the next page, figure 1 and 2 plot `maxdev` versus `F1` and `maxdev` versus `#nodes`, respectively.

maxdev	F1	#nodes
0.10	0.84	33.4
0.20	0.84	33.4
0.30	0.84	33.4
0.40	0.84	33.4
0.50	0.84	33.4
0.60	0.84	33.4
0.70	0.84	33.4
0.80	0.84	31.6
0.90	0.87	30.9
1.00	0.87	30.9
1.10	0.87	30.9
1.20	0.87	30.9
1.30	0.87	30.9
1.40	0.88	30.1
1.50	0.88	30.1
1.60	0.88	29.7
1.70	0.88	28.9
1.80	0.88	28.9

1.90	0.88	28.8
2.00	0.88	28.8
2.10	0.87	25.2
2.20	0.87	25.2
2.30	0.87	24.3
2.40	0.87	24.3
2.50	0.87	24.1
2.60	0.87	24.1
2.70	0.87	23.9
2.80	0.87	23.9
2.90	0.87	23.5
3.00	0.86	21.8
3.10	0.84	19.1
3.20	0.84	19.1
3.30	0.84	19.1
3.40	0.84	19.1
3.50	0.84	19.1
3.60	0.84	19.1
3.70	0.84	19.1
3.80	0.82	18.5
3.90	0.82	18.2
4.00	0.82	18.2
4.10	0.82	16.8
4.20	0.82	16.8
4.30	0.82	16.8
4.40	0.82	16.8
4.50	0.82	16.8
4.60	0.82	16.8
4.70	0.82	16.8
4.80	0.82	16.8
4.90	0.81	16.2

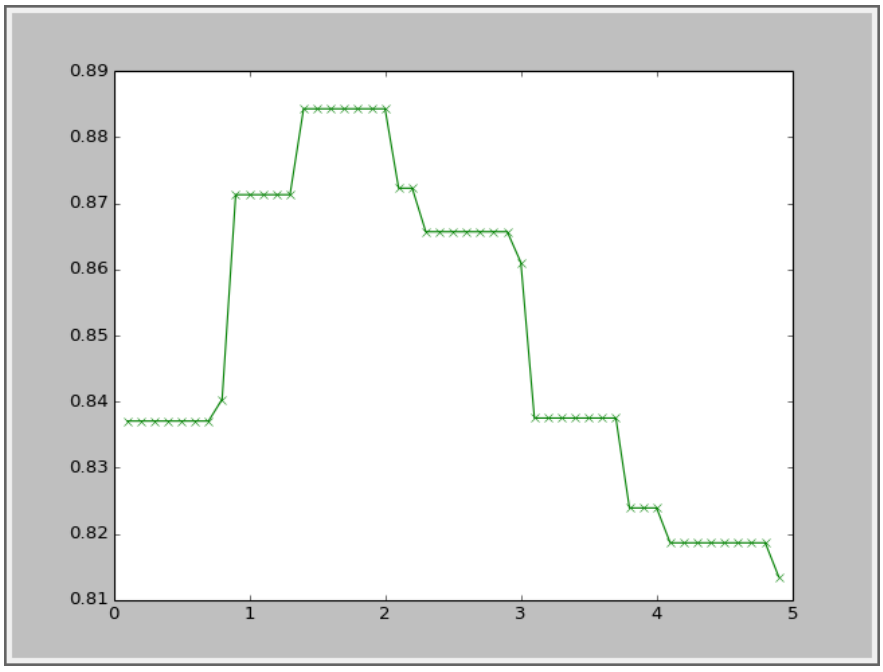

Figure 1: `maxdev` (x) VS F1 score on validation (y)

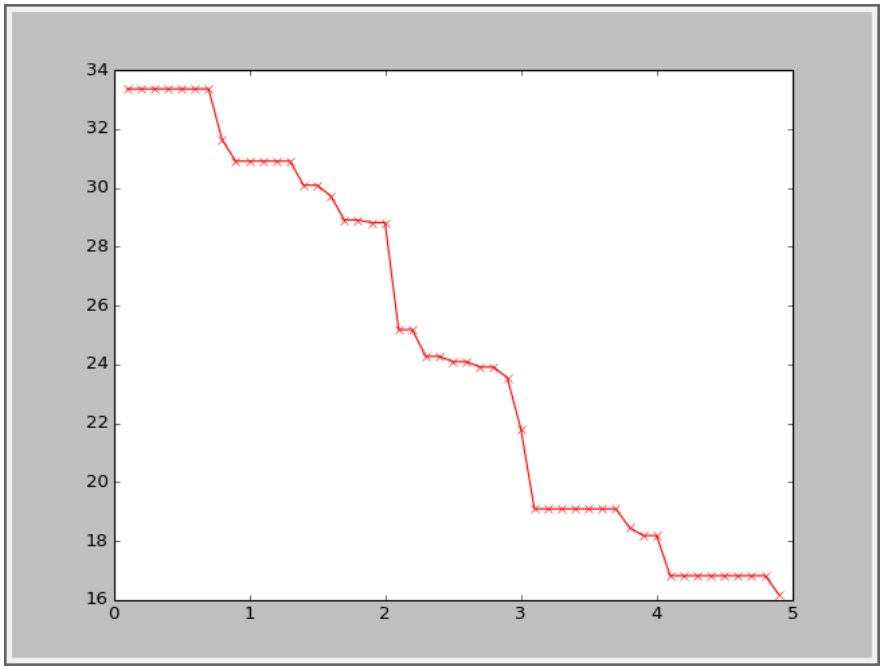

Figure 2: `maxdev` (x) VS `#nodes` after pruning (y)

Figure 2 shows a steady decrease in the number of nodes. This is an expected behavior, because a larger maxdev value implies a more aggressive pruning standard. More and more nodes are pruned as maxdev increases.

Figure 1 shows that F1 score after pruning increases at first, and then drops as maxdev goes too high. This is because the decision tree overfits the data at first, and as we prune, we decrease the level of overfitting. After a certain point, however, more pruning leads to an unreasonably weak decision tree, which turns the overfitting problem into underfitting. That's why we see a decline in performance as more and more nodes are removed.

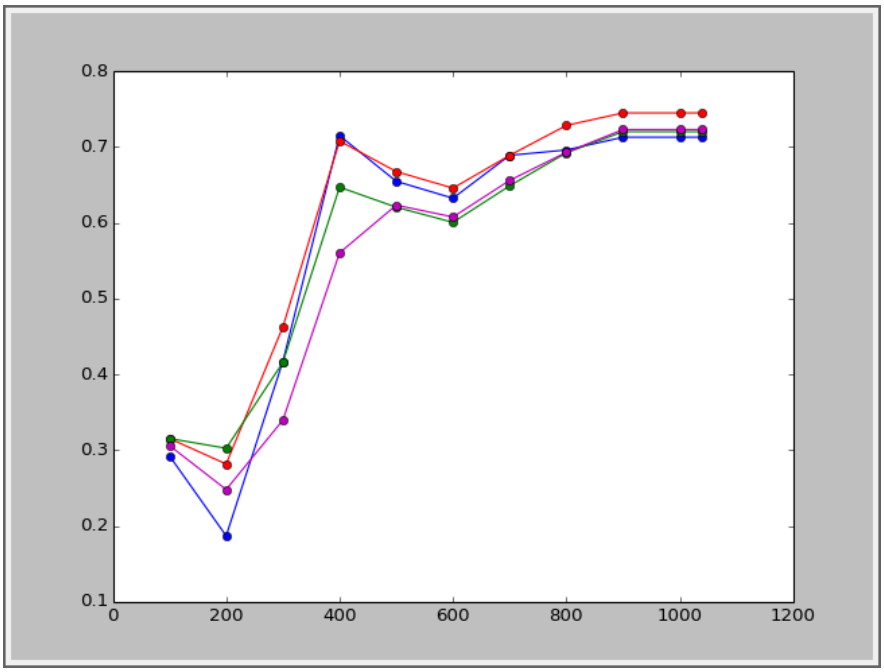

Figure 3 Learning Curve: training size VS F1 score on test set

Figure 3 shows 4 learning curves on the test set, each with a different maxdev value.
Blue line: maxdev = 0 no pruning, macro F1 = 0.72
Red line: maxdev = 2.00 best hyperparameter on the validation set, F1 = 0.75
Green line: maxdev = 3.00, F1 = 0.71
Purple line: maxdev = 4.00, F1 = 0.70

The graphs show that performance normally increases as training set size increases. This is expected because more training leads to less overfitting, thus a better decision tree. The red line is the best model on the validation set. The green and purple lines cannot beat the one without pruning, because underfitting is degrading their performances.

All the figures above are generated by `testDecisionTree.py`, included in my submission. Python `matplotlib` is used to draw the graphs.