CS419M - Assignment 1

Varun Patil - 160010005

August 15, 2018

1 Introduction & Tools

The problem statement is to create a regression tree to fit the given data. No python libraries may or have been used including but not limited to machine learning libraries such as `sklearn`, `TensorFlow` and other general computational libraries such as `numpy`. All code has been written and tested in Python 3.6 on Microsoft Windows 10. The only library used is `matplotlib` for plotting loss graph.

2 Training Algorithm

The regression tree was trained by creating splits for each feature at every step, minimizing the loss by choosing the split with lowest loss of all. In case multiple splits have the same minimum loss, one of these is chosen randomly. In case one feature has multiple points of splits with the same loss, the median of these is taken to make the split. Internally, splits are taken by sorting the data to split for each feature and then iterating it in the reverse sense calculating losses for each possible split. All trees were constrained to a maximum depth of 15 levels.

3 Dataset Split

25% (150) and 5% (200) of the samples were used for validation and pruning for the first and second (larger) dataset respectively. These values were obtained by experimentation for lowest validation loss.

4 Notes on Implementation

All training, pruning and inference code is kept in a single file `helpers.py` which is non-specific to the data provided. Functions from this file are called everywhere else with the hyperparameters as arguments. All `main.py` files take the following arguments:

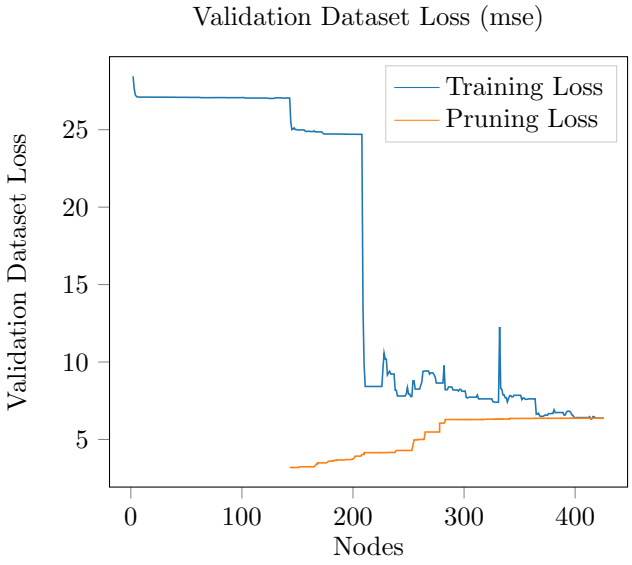
- `-d` or `--train_data` - CSV file to be used as train input. Defaults to `train.csv`.
- `-d` or `--test_data` - CSV file to be used as test input. Defaults to `test.csv`.
- `-m` or `--mean_squared` - use the mean squared error for loss. True by default.
- `-a` or `--absolute` - use the absolute error for loss.
- `-s` or `--std` - minimize split standard deviation instead of a loss function.
- `-v` or `--verbose` - turn on verbose mode. Significantly increases training time.
- `-f` or `--forest` - train a random forest-like model instead of a single tree.
- `-l` or `--min_leaf_size` - specify the minimum number of examples for each leaf node.

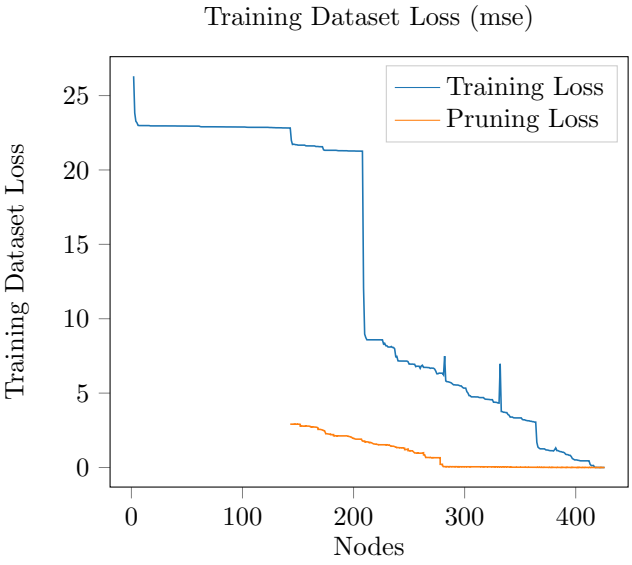
All experiments were carried out on a Core i7-6500U mobile processor with 8GB memory, single threaded.

5 Results

5.1 Dataset 1

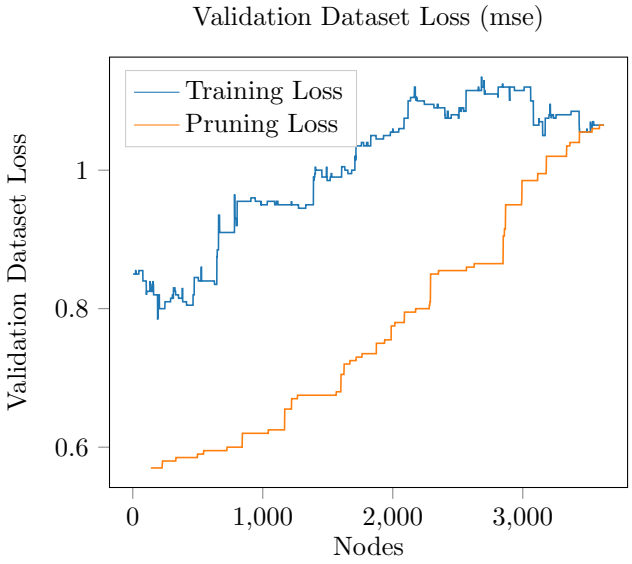
A minimum mean squared validation loss of approx. **3.20** and absolute loss of **3.70** were observed after training. Similar values were obtained on the test dataset with loss around 4.5 to 4.8. Due to the random nature of selection of conflicting best features, some variation in losses was observed for different trained trees on the same hyperparameters. Each leaf was constrained to have a minimum of 2 samples, changing this to 10 increased final validation loss to approx. 5.44. The model showed exactly zero loss on training data before pruning. The tree had approx. **130-140 nodes** after pruning.

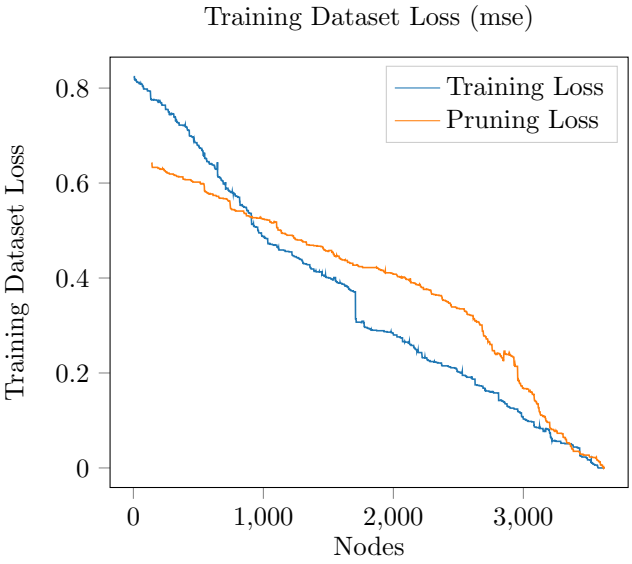




5.2 Dataset 2

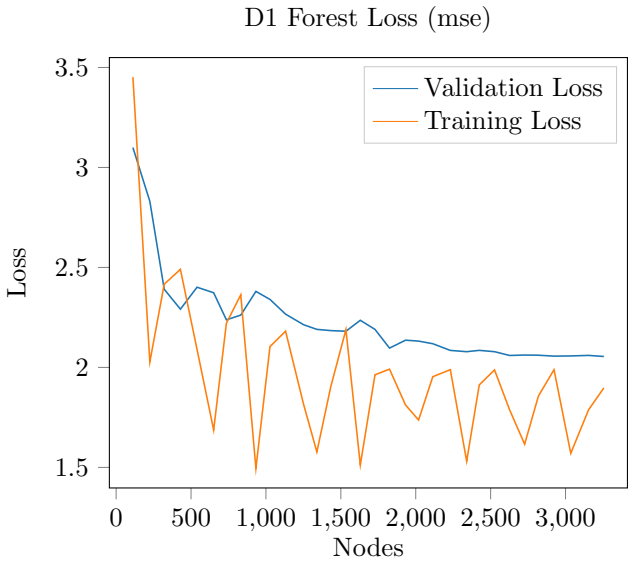
A minimum mean squared validation loss of approx. **0.57** and absolute loss of **0.39** were observed after training. Very minor loss on training data was observed before pruning due to multiple samples having same features but different output values. The final tree had approx. **140-180 nodes** after pruning. As with dataset 1, each node leaf was constrained to 2 samples. For this dataset, a loss function with the probabilities of each split multiplying the loss yielded better results and much better timings.

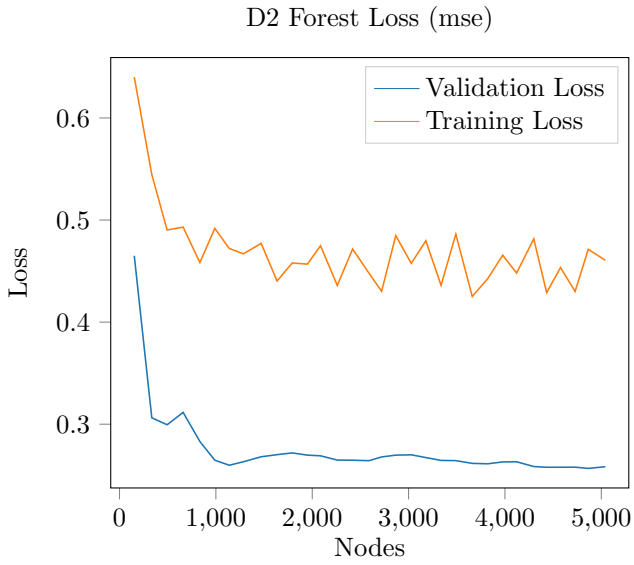




6 Forest-Like Model

The existed model could be easily extended to a random forest. Further reduction in loss can be obtained by dropping features at each split randomly instead of doing it for every tree, and using standard deviation minimization. Training a forest with 32 trees on Dataset 1 with a feature drop probability of 0.2 yielded a validation loss of approx. **2.2** for MSE and **2.3** for absolute error. For Dataset 2, a similar forest with 16 trees and dropout 0.2 yielded MSE on validation data as approx. **0.25**.





7 Timings

The following approximate timings were observed for each operation, taking into account errors due to CPU and memory caching by averaging:

- Training - Dataset 1 - MSE - **0.45s**
- Training - Dataset 1 - MAE - **0.43s**
- Training - Dataset 2 - MSE - **19.95s**
- Training - Dataset 2 - MAE - **18.82s**
- Inference - Dataset 1 - **0.001s** (8μs per sample)
- Inference - Dataset 2 - **0.004s** (10μs per sample)

Notes:

1. All timings were calculated using CPU nano-timing
2. Training includes pruning
3. All timings were calculated with the non-verbose mode

8 Conclusion

We may conclude that using a decision tree may produce significant and acceptable results on certain types of datasets, with improvements on using random forests. The loss function does not have significant bearing on the results, since it gives contradicting results for two datasets.
The only references used for this Assignment were class notes, StackOverflow and python documentation.