Merlin Carson
CS446, Spring 2018

Decision Trees experimental results

1.

I started the process of creating my regression based decision tree algorithms by splitting my datasets into k-folds. For this, k is a hyperparameter, it determines how many equal sized sets I split the data into. This allows for cross-validation of multiple trees each using a different subset of the data as the validation set. Averaging the RMSE between each tree gives less biased accuracy results since each tree is trained and validated with different data from the same dataset. A fold represents one of the equal sized sets the data is split into. k trees are built, each one using a different fold for the validation set, while using the rest of the data as the training set.

2.

Following the inclass slides, I implemented the decision tree algorithm. I used python for coding and dictionaries for my tree data structure. I loaded the csv data, Carseats.csv from file, and saved them into a multidimensional array. I wrote a recursive algorithm for determining optimal splits. At each node, for each feature and for each row value of the feature in the training set. Once a split feature and value is made the training data must be iterated over and split in to 2 sets, one where the featured value is less and one where it is greater than or equal. Once the splits are selected, the data is once again iterated over to calculate the RSS of both sides and added together to determine the total RSS of the split. The feature and value that gives the lowest RSS is selected as the optimal split and new nodes in the tree are created with the subsets of data. This is a highly nested algorithm, however since the data is decreasing at each level of the tree, it's complexity is O(plog(n)^3), where p is the number of features and n is the number of training examples.

To determine the complexity of the tree and thus the number of predictor leafs, I implemented three stopping conditions. These stopping conditions consisted of two hyperparameters, max height of the tree and max number of data in a node and the base case, when all the items are split to a single side. Once one of these stopping conditions is reached the median of the data, which is more resilient to outliers than mean, is chosen as the prediction value for an element in the validation set that traverses that path within the tree.

Once the tree was built I tested two methods of data preprocessing, Standardization to achieve zero mean and unit variance and min/max Normalization to achieve a unit value. Cross-validation of the two showed that min/max Normalization

achieved higher accuracies. I also implemented tuning functions to grid search my hyperparameters to determine the optimal values. Through tuning I found a max tree height of 5 and max number of data in a leaf to also be 5. I only show results of 2 folds to minimize the output. Below are my results.

```
max tree height:  5
max data:  5

Decision Tree (fold 1):
(X6 = Good)
 (X5 < 0.385)
  (X7 < 0.364)
   [15.63]
   (X1 < 0.605)
         (X8 < 0.750)
         [11.82]
         [10.96]
         [12.855]
  (X3 < 0.154)
   (X7 < 0.855)
         (X7 < 0.273)
         [8.64]
         [7.58]
         [5.609999999999999]
   (X7 < 0.564)
         (X7 < 0.218)
         [12.530000000000001]
         [10.51]
         [9.16]
 (X5 < 0.516)
  (X5 < 0.221)
   (X2 < 0.525)
         (X1 < 0.237)
         [7.71]
         [9.48]
         (X7 < 0.691)
         [11.67]
         [9.35]
   (X7 < 0.473)
```

(X1 < 0.421)
[6.44]
[9.01]
(X6 = Bad)
[5.25]
[7.57]
(X7 < 0.764)
(X1 < 0.750)
(X5 < 0.590)
[6.03]
[4.68]
(X5 < 0.713)
[8.225000000000001]
[6.5]
(X2 < 0.444)
(X4 < 0.696)
[2.52]
[0.0]
(X1 < 0.632)
[2.34]
[5.08]

Decision Tree (fold 2):
(X6 = Good)
(X5 < 0.455)
(X7 < 0.582)
[13.91]
(X1 < 0.724)
(X1 < 0.276)
[9.46]
[11.22]
[12.66]
(X2 < 0.424)
(X4 < 0.797)
(X2 < 0.121)
[6.39]
[8.21]
[9.99]
(X3 < 0.034)

```
        (X8 < 0.625)
        [8.085]
        [10.61]
        (X1 < 0.551)
        [10.26]
        [12.29]
(X5 < 0.491)
 (X1 < 0.551)
  (X5 < 0.317)
        [9.885]
        (X2 < 0.808)
        [6.68]
        [8.595]
  (X6 = Medium)
        (X2 < 0.414)
        [10.01]
        [11.96]
        [7.62]
 (X6 = Bad)
  (X4 < 0.205)
        (X5 < 0.581)
        [4.609999999999999]
        [0.53]
        (X1 < 0.694)
        [4.56]
        [7.68]
  (X3 < 0.448)
        (X7 < 0.418)
        [7.32]
        [5.47]
        (X4 < 0.988)
        [8.73]
        [4.21]

Fold RMSE:  [3.8102809443661756, 3.2861921733215778]
Mean RMSE: 3.548
```

3.

After evaluating my results for the naive decision tree, I moved on to regularization by implementing the pruning algorithm. This algorithm works by penalizing complexity and determining if by removing any of the decision nodes the tree could achieve a higher accuracy. I wrote a recursive algorithm that temporarily removes a decision node and cross validates the error. If the cross validated error is smaller than the node is permanently removed, else it's placed back into the tree. This is done over the whole tree for as many times is necessary until removing leaves no longer reduces the error. This algorithm tends to prune quite a bit of nodes and creates a significant decrease in error rate, even with the hyperparameter α, the scaler for how much we discourage complexity, set to a fairly low 0.1. See results below.

alpha: 0.1

Decision Tree (fold 1):

```
(X5 < 0.442)
 (X6 = Good)
  (X5 < 0.292)
   [14.14]
   [10.98]
  (X6 = Medium)
   (X5 < 0.300)
        (X4 < 0.976)
        [11.695]
        [8.41]
        (X7 < 0.564)
        [9.850000000000001]
        [7.63]
   (X1 < 0.427)
        (X7 < 0.564)
        [7.63]
        [4.745]
        [9.14]
 (X6 = Good)
  (X5 < 0.867)
   (X7 < 0.564)
        (X1 < 0.404)
        [9.03]
        [11.62]
```

```
        (X8 < 0.125)
        [10.49]
        [7.77]
  [6.67]
 (X5 < 0.608)
  (X7 < 0.473)
        (X2 < 0.404)
        [5.52]
        [8.71]
        (X1 < 0.629)
        [5.58]
        [7.96]
  (X1 < 0.449)
        (X8 < 0.875)
        [4.109999999999999]
        [1.2999999999999998]
        (X3 < 0.517)
        [5.3]
        [8.25]

Pruned Tree(17 leaves pruned):
(X5 < 0.442)
 (X6 = Medium)
  (X7 < 0.564)
   [9.850000000000001]
   [7.63]
  (X1 < 0.427)
   [6.88]
   [9.14]
 [6.39]

Decision Tree (fold 2):
(X6 = Good)
 (X5 < 0.671)
  (X8 < 0.375)
   (X1 < 0.424)
        [10.74]
        (X3 < 0.731)
        [12.635]
```

[16.27]
(X3 < 0.538)
(X8 < 0.500)
[7.5]
[9.71]
[12.025]
[7.119999999999999]
(X6 = Medium)
(X7 < 0.655)
(X5 < 0.497)
(X1 < 0.694)
[9.33]
[11.855]
(X1 < 0.576)
[6.52]
[8.44]
(X5 < 0.964)
(X1 < 0.518)
[5.74]
[6.89]
[0.0]
(X5 < 0.395)
[8.075]
(X2 < 0.990)
(X1 < 0.306)
[1.42]
[5.140000000000001]
[10.14]

Pruned Tree(10 leaves pruned):
(X6 = Good)
(X5 < 0.671)
(X8 < 0.375)
(X1 < 0.424)
[10.74]
[12.66]
[9.48]
[7.119999999999999]
(X6 = Medium)

```
 (X5 < 0.497)
  [9.370000000000001]
  [7.654999999999999]
 (X5 < 0.395)
  [8.075]
  [4.94]
```

Fold RMSE: [2.7334029157809856, 3.526856567823535]
Mean RMSE: 3.130

4.

Having determined optimal decision trees, it was time to move onto building and testing ensembles of decision trees. The first algorithm I implemented was Bagging, bootstrap aggregation. This process involves build several different trees by randomly selecting a portion of the training set with replacement. This ratio of the training set I also setup up as a hyperparameter, however I went with the typically used ⅔ of the size of the original training set as the number of examples that make up the training set created for each new tree that is built. I also created a hyperparameter for the number of trees that are built and included that in my Bagging tuning function to determine the optimal number of trees. While this does decorrelate the trees, it does not affect or change which features are used for determining splits and thus is not relevant for feature selection. Through the tuning process, cross-validation showed the optimal number of trees to be 30. See results below.

number of trees : 30

fold 1
Building tree 0
Building tree 1
Building tree 2
Building tree 3
Building tree 4
Building tree 5
Building tree 6
Building tree 7
Building tree 8
Building tree 9
Building tree 10
Building tree 11

Building tree 12
Building tree 13
Building tree 14
Building tree 15
Building tree 16
Building tree 17
Building tree 18
Building tree 19
Building tree 20
Building tree 21
Building tree 22
Building tree 23
Building tree 24
Building tree 25
Building tree 26
Building tree 27
Building tree 28
Building tree 29

fold 2
Building tree 0
Building tree 1
Building tree 2
Building tree 3
Building tree 4
Building tree 5
Building tree 6
Building tree 7
Building tree 8
Building tree 9
Building tree 10
Building tree 11
Building tree 12
Building tree 13
Building tree 14
Building tree 15
Building tree 16
Building tree 17
Building tree 18

Building tree 19
Building tree 20
Building tree 21
Building tree 22
Building tree 23
Building tree 24
Building tree 25
Building tree 26
Building tree 27
Building tree 28
Building tree 29

Fold RMSE: [2.724736030792708, 3.3517972371177223]
Mean RMSE: 3.038

5.

For my final experiment I implemented the Random Forest algorithm. Random Forest is an extension of the Bagging algorithm. However, instead of training on the whole feature set, a randomly selected subset of the features are used to train each tree. This decorrelates the features among each of the trees and like bootstrap aggregation should reduce the high variance found in single trees. For the size I used the well documented, $\sqrt{num\,features}$, as the number of randomly selected features each tree is trained on. While it would be possible to determine the importance of feature and tune feature selection by viewing the set of features each tree is trained with and comparing it to each trees accuracy, this was ultimately out of scope for my experiment, since the output of the Random Forest is the average of all the predictions and thus an error rate is not calculated on a per-tree basis. This algorithm did achieve the highest results of all my experiments after tuning to determine the optimal number of trees to be 500 and the trees to be stumps with a max height of 3. See results below.

number of trees : 500
max tree height: 3

fold 1
Building tree 0
Building tree 1
...
{removed for brevity}

...
Building tree 498
Building tree 499

fold 2
Building tree 0
Building tree 1
...
{removed for brevity}
...
Building tree 498
Building tree 499

Fold RMSE: [2.9741322263813355, 2.564871950994825]
Mean RMSE: 2.770