Applied Machine Learning Exercise #3

Gal Lavee
June 3, 2018

In this exercise we will implement the Gradient Boosted Regression Trees model and apply our implementation to several public datasets. Code can be written in the programming language of your choice, but no external libraries are allowed, with the following exceptions:

1. libraries that perform vector and matrix calculation (e.g. numpy in python or Math.Net Numerics for C#)
2. randomization functions (such as *random.multivariate_normal* , *random.choice* in python)
3. plotting libraries or other tools for generating the required plots
4. dataframe libraries (e.g. *pandas* in python)

Part 1. The Data

From [1] download the file *train.csv*. You may also read further about the dataset and the various features on this page.

In this part you are required to :

1. implement a function that reads in the data (path to the csv file is a parameter), removes the *Id* field and removes entries for which the *Saleprice* attribute is not known.
2. implement a function that splits the data into training and test, the split should be 80% to the Train set and 20% to the Test set.
3. implement a class that encapsulates the training data. The constructor to this class should be given a subset of the data
4. The class above should code categorical features using the following scheme: For each possible value (including the 'missing' value) of feature i compute the average SalePrice (or label value in general). Next rank the values according to the average computed value (ascending) and code the features using the rank.

   Example: Feature *Alley* has possible values: (missing), *Grvl* , and *Pave*

Value	Avg *SalePrice*	Rank
Grvl	122219	1.0
Pave	168000	2.0
(missing)	183452	3.0

   The coding map should be saved for use on the test set.
5. The class above should deal with missing numerical features using mean imputation. That is, for each numerical feature compute the average value over the training data excluding missing values. Then use this calculated value to fill in the missing values.

   The imputation map should be saved for use on the test set.

[1] `https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data`

6. implement a class encapsulating the test set. The constructor to this class should take as input a subset of the data as well as an imputation map and a categorical coding map. Categorical features should be coded using the coding map, and missing numeric features should be encoded using the imputation map.

Part 2. Data structures for boosted trees

In this part we will implement the data-structures we need in order to implement the gradient boosted regression trees.

You are required to implement the following data structures:

1. *RegressionTreeNode* - This represents a single node in a regression tree. It should have the following member variables:
   (a) j - the index of the feature on which we split at this node
   (b) s - the threshold on which we split at this node
   (c) $leftDescendent$ - the *RegressionTreeNode* which is the immediate left descendent of the current node
   (d) *rightDescendent* - the *RegressionTreeNode* which is the immediate right descendent of the current node
   (e) *const* - the constant scalar associated with this node

   Note that some fields may take on null values (e.g. *const* in non-leaf nodes)

   The class should also implement several member methods:

   (a) MakeTerminal(c) - make the node into a terminal (leaf node) and set its constant value to c
   (b) Split(j,s) - make the node into an internal node which splits on feature index j at threshold s and instantiate its left and right descendents.
   (c) printSubTree() - print the sub- regression tree rooted at the current node in a readable "if-then" format, for example:

```
if x['LotArea']<=7.000000 then:
       return -23307.173126
if x['LotArea']>7.000000 then:
       return 122362.658913
```

2. *RegressionTree* - this represents an entire regression tree It should have the following member variables:
   (a) root - this is *RegressionTreeNode* that represents the root of the tree

   The class should also implement several member methods:

   (a) GetRoot() - Get the root of the tree
   (b) Evaluate(x) - for a vector valued x compute the value of the function represented by the tree

3. *RegressionTreeEnsemble* - this represents an entire regression tree It should have the following member variables:

   (a) trees - An ordered collection of type *RegressionTree*

   (b) weights - the weight associated with each regression tree

   (c) M - the number of regression trees

   (d) c - the initial constant value returned before any trees are added

   The class should also implement several member methods:

   (a) Addtree(tree, weight) - add a *RegressionTree* to the ensemble with corresponding weight

   (b) SetInitialConstant(c) - Set the intital constant c

   (c) Evaluate(x,m)= for a vector valued x compute the value of the function represented by the ensemble of tree, if we consider only the first m trees

   This class encapsulates the model:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbb{I}\left[\mathbf{x} \in \mathcal{R}_m\right]$$

Part 3. The Algorithm

In this section we will implement the gradient boosted regression tree algorithm. In this part you are required to implement the following:

1. A function named *CART* that takes the hyper parameters *MaxDepth* and *MinNodeSize* as arguments as well as a dataset object. The function should implement the CART algorithm that we saw in class for building regression trees. It should return an object of type *RegressionTree*

2. A function named *GBRT* which takes as input the hyperparameters *NumTrees*,*MaxDepth* and *MinNodeSize*. The function should implement the Gradient Boosted Regression Tree Algorithm we saw in class and return an object of type *RegressionTreeEnsemble*

Part 4. Machine Learning Diagnostics

In regression trees we are maximizing some loss function, in particular we are interested in squared loss:

$$L(\theta) = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \left(y_i - f\left(\mathbf{x}_i\right)\right)^2$$

In order to verify our machine learning code, we should check that with every additional tree the quantity listed above is decreasing. Further, we also would like to evaluate our generalization performance by evaluating the loss on test data.

In this part you are required to:

1. Extend the function GBRT above to take a second dataset (a test set). The function should now output the (average) test and train loss after each added tree.

Part 5. **Bells and Whistles**

Industrial strength Gradient boosted machines are extended in several ways we discussed in class

1. Shrinkage - A hyperparameter $\nu \in [0, 1]$ dampens the impact of every additional tree. There exists a tradeoff between the number of trees and the shrinkage factor ν.

2. Subsampling - Another hyperparameter $\eta \in [0, 1]$ controls the portion of the data considered for each tree. This adds some stochasticity which has a regularizing effect.

In this section you are required to modify the function *GBRT* above to consider the two additional hyperparameters ν and η and implement the functionality of shrinkage and subsampling.

Part 6. **Feature importance**

One great advantage of tree models is their ability to quantify the importance of the various features under consideration. We say that a feature m is important, with respect to an ensemble of trees and a training dataset, if there are many splits according to feature m in the various trees in the ensemble and the reduction in error achieved by making the split is large compared to using the average of the parent node.

Letting $\mathcal{R}_L$ and $\mathcal{R}_R$ be the partions of the data created by splitting internal node t on feature j, the improvement is given by the quantity:

$$\hat{i}_t^2 = \left(\sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_L} (y_i - c_L)^2 + \sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_R} (y_i - c_R)^2 \right) - \left(\sum_{(\mathbf{x}_i, y_i) \in \mathcal{R}_{bef}} (y_i - c_{bef})^2 \right)$$

where c_R and c_L are the average y values of the datapoints in partitions $\mathcal{R}_L$ and $\mathcal{R}_R$, respectively. for a particular tree the importance of a feature k is given by:

$$\mathcal{I}_k^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbb{I}\left[v\left(t\right) == k\right]$$

where $v\left(t\right)$ denotes the index of the feautre used to split at internal node t.

For an ensemble of trees the quantity above is averaged over all trees in the ensemble.

$$\frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_k^2(T_m)$$

In this part :

1. implement a function that takes as input a single tree and a dataset and computes the relative feature importance according to the above formula.

2. implement a function that takes as input an ensemble of trees and a dataset and computes the relative importance of all features over the entire dataset. The output of the function should normalize the importance so that the most important feature has an importance of 1.0

3. implement a function that takes as input a dataset, a tree ensemble, and a feature index and plots the partial dependence plot for the corresponding feature.

Part 7. Runtime optimization

If you try to profile your runtime you may find that the computation bottleneck is the search for the partition threshold. One way to improve this is to accept a suboptimal split by not trying all possible split thresholds, but rather a subset. One way to select the subset is to compute evenly spaced percentiles of the feature values (e.g. 25-th 50-th and 75-th percentile). This ensures that each threshold we try represents a change in an equal amount of examples.

In this part:

1. Modify the CART and GBRT procedure to take a *numThresholds* hyperparameter, which determines how many threshold candidates to try. Modify the logic to consider evenly spaced percentiles such that the total number of candidates is equal to *numThresholds*. Note that it does not make sense to consider the 0-th or 100-th percentile as these would , by definition, lead to a degenerate split. Thus only internal percentiles should be considered as possible split points.

Part 8. The Main Flow

In this part we will put all the pieces together. Implement the main flow of your program to do the following:

1. Read in data and split into train and test
2. Code categorical variables and impute missing values in the training data using the methods implemented in part 1
3. Use the coding and imputation maps to code and impute variables in the test data
4. Set hyperparameters based on input arguments/configuration file
5. Learn tree ensemble model from the training data using GBRT algorithm with hyperparameters specified by input arguments/configuration file (should output train and test performance to files)
6. Output the hyperparameters and final train and test error to a file , the file should also include the amount of time needed for training.

Part 9. Final Presentation

This project will be presented to the class as a twenty minute presentation. During the presentation you will be asked to present results on the housing dataset from kaggle described above and an additional dataset (which describes a regression). You may choose the additional dataset/regression problem from the following list of problems where gradient boosting is known to work well(or elsewhere):

1. `http://2017.recsyschallenge.com/`
2. `https://www.kaggle.com/c/caterpillar-tube-pricing/data`
3. `https://www.kaggle.com/c/liberty-mutual-group-property-inspection-prediction/`
4. `https://www.kaggle.com/c/crowdflower-search-relevance`
5. `https://www.kaggle.com/c/grasp-and-lift-eeg-detection`

6. `https://www.kaggle.com/c/coupon-purchase-prediction`

7. `https://www.kaggle.com/c/avito-context-ad-clicks`

8. `https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings`

9. `https://www.kaggle.com/c/homesite-quote-conversion`

Note, that many of these are datasets from challenges where the main problem is not necessarily regression. However, you are not limited to solving the main task of the challenge. You may use the dataset to formulate and solve your own regression problem so be creative in this regard.

For each of the two datasets your presentation should include the following deliverables (marked by green Deliverable tag). In addition to the presentation you should also turn in your code and the plots below online via moodle.

Deliverable 1. A description of the dataset and how you formalized the regression problem. The description should include a list of features (possibly feature categories), number of examples, number of categorical vs numerical features, number of features with missing values, etc...

Deliverable 2. Hyperparams impact on error analysis. Plot train and test error as a function of the number of trees in the ensemble. Compare plots for different values of the various parameters. You may choose to analyze the impact of 3 out of the 4 following hyperparams $\{MaxDepth, \eta, \nu, numThresholds\}$, M the number of tree in the ensemble should be set to at least 100. You may choose the settings of the parameters.

Deliverable 3. Hyperparams impact on runtime analysis. Plot the time it takes to train a boosted tree ensemble of at least $M = 50$ trees as function of hyperparameters. Choose 2 of the 3 hyperparameters: $\{MaxDepth, \eta, numThresholds\}$ and create a separate analysis plot for each. Each plot should evaluate at least 5 unique values for the hyperparameter chosen.

Deliverable 4. Feature Importance. For a particular learned model (make sure to note the hyperparameters used in the model in your presentation) we will analyze the feature importance. First, print the first 5 trees of the tree ensemble in a readable "if-then" format, then create a bar graph of feature importance for the $5 - 10$ most important features according to the feature importance metric described above.

Deliverable 5. Score On Leaderboard. The housing dataset has a *test.csv* that has all the features of the training, excluding the sales price. Run your model on this file and submit the result to kaggle (you may have to create an account). In your presentation, show where your submission ended up on the leaderboard. If your second dataset is also from a kaggle competition you can optionally make a submission to that competition and report your position on the leaderboard.

Deliverable 6. Your Code. Be prepared to explain where and how you implemented each of parts 1-8 above.