Homework 6: Decision Trees and Boosting

Due: Wednesday, April 19th, 2023 at 11:59PM EST

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g.LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better. **The optional problems should not take you too much time and help you navigate the material, consider taking a shot at them.**

---

1 Decision Tree Implementation

In this problem we'll implement decision trees for both classification and regression. The strategy will be to implement a generic class, called `Decision_Tree`, which we'll supply with the loss function we want to use to make node splitting decisions, as well as the estimator we'll use to come up with the prediction associated with each leaf node. For classification, this prediction could be a vector of probabilities, but for simplicity we'll just consider hard classifications here. We'll work with the classification and regression data sets from previous assignments.

1. Complete the `compute_entropy` and `compute_gini` functions.

2. Complete the class `Decision_Tree`, given in the skeleton code. The intended implementation is as follows: Each object of type `Decision_Tree` represents a single node of the tree. The depth of that node is represented by the variable self.depth, with the root node having depth 0. The main job of the fit function is to decide, given the data provided, how to split the node or whether it should remain a leaf node. If the node will split, then the splitting feature and splitting value are recorded, and the left and right subtrees are fit on the relevant portions of the data. Thus tree-building is a recursive procedure. We should have as many `Decision_Tree` objects as there are nodes in the tree. We will not implement pruning here. Some additional details are given in the skeleton code.

3. Run the code provided that builds trees for the two-dimensional classification data. Include the results. For debugging, you may want to compare results with sklearn's decision tree (code provided in the skeleton code). For visualization, you'll need to install `graphviz`.

4. Complete the function `mean_absolute_deviation_around_median` (MAE). Use the code provided to fit the `Regression_Tree` to the krr dataset using both the MAE loss and median predictions. Include the plots for the 6 fits.

2 Ensembling

Recall the general gradient boosting algorithm , for a given loss function ℓ and a hypothesis space $\mathcal{F}$ of regression functions (i.e. functions mapping from the input space to $\mathbb{R}$):

0: Initialize $f_0(x) = 0$.

1: For $m = 1$ to M:

(a) Compute:

$$\mathbf{g}_m = \left(\frac{\partial}{\partial f_{m-1}(x_j)} \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i)\right) \right)_{j=1}^{n}$$

(b) Fit regression model to $-\mathbf{g}_m$:

$$h_m = \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left(\left(-\mathbf{g}_m\right)_i - h(x_i) \right)^2 .$$

(c) Choose fixed step size $\nu_m = \nu \in (0, 1]$, or take

$$\nu_m = \arg\min_{\nu > 0} \sum_{i=1}^{n} \ell\left(y_i, f_{m-1}(x_i) + \nu h_m(x_i)\right) .$$

(d) Take the step:

$$f_m(x) = f_{m-1}(x) + \nu_m h_m(x)$$

3: Return f_M.

This method goes by many names, including gradient boosting machines (GBM), generalized boosting models (GBM), AnyBoost, and gradient boosted regression trees (GBRT), among others. One of the nice aspects of gradient boosting is that it can be applied to any problem with a subdifferentiable loss function.

Gradient Boosting Regression Implementation

First we'll keep things simple and consider the standard regression setting with square loss. In this case the we have $\mathcal{Y} = \mathbb{R}$, our loss function is given by $\ell(\hat{y}, y) = 1/2\,(\hat{y} - y)^2$, and at the m'th round of gradient boosting, we have

$$h_m = \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left[(y_i - f_{m-1}(x_i)) - h(x_i) \right]^2 .$$

5. Complete the `gradient_boosting` class. As the base regression algorithm to compute the argmin, you should use sklearn's regression tree. You should use the square loss for the tree splitting rule (`criterion` keyword argument) and use the default sklearn leaf prediction rule from the `predict` method [1]. We will also use a constant step size ν.

6. Run the code provided to build gradient boosting models on the regression data sets `krr-train.txt`, and include the plots generated. For debugging you can use the sklearn implementation of `GradientBoostingRegressor`[2].

[1] Examples of usage are given in the skeleton code to debug previous problems, and you can check the docs `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html`

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html`

Classification of images with Gradient Boosting

In this problem we will consider the classification of MNIST, the dataset of handwritten digits images, with ensembles of trees. For simplicity, we only retain the '0' and '1' examples and perform binary classification.

First we'll derive a special case of the general gradient boosting framework: BinomialBoost. Let's consider the classification framework, where $\mathcal{Y} = \{-1, 1\}$. In lecture, we noted that AdaBoost corresponds to forward stagewise additive modeling with the exponential loss, and that the exponential loss is not very robust to outliers (i.e. outliers can have a large effect on the final prediction function). Instead, let's consider the logistic loss

$$\ell(m) = \ln\left(1 + e^{-m}\right),$$

where $m = yf(x)$ is the margin.

7. Give the expression of the negative gradient step direction, or pseudo residual, $-\mathbf{g}_m$ for the logistic loss as a function of the prediction function f_{m-1} at the previous iteration and the dataset points $\{(x_i, y_i)\}_{i=1}^n$. What is the dimension of g_m?

8. Write an expression for h_m as an argmin over functions h in $\mathcal{F}$.

9. Load the MNIST dataset using the helper preprocessing function in the skeleton code.Using the scikit learn implementation of `GradientBoostingClassifier`, with the logistic loss (`loss='deviance'`) and trees of maximum depth 3, fit the data with 2, 5, 10, 100 and 200 iterations (estimators). Plot the train and test accurary as a function of the number of estimators.

Classification of images with Random Forests (Optional)

10. Another type of ensembling method we discussed in class are random forests. Explain in your own words the construction principle of random forests.

11. Using the scikit learn implementation of `RandomForestClassifier`[3], with the entropy loss (`criterion='entropy'`) and trees of maximum depth 3, fit the preprocessed binary MNIST dataset with 2, 5, 10, 50, 100 and 200 estimators.

12. What general remark can you make on overfitting for Random Forests and Gradient Boosted Trees? Which method achieves the best train accuracy overall? Is this result expected? Can you think of a practical disadvantage of the best performing method? How do the algorithms compare in term of test accuracy?

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier`