NYU Center for Data Science

DS-GA 1003: Machine Learning (Spring 2020)

Homework 6: Multiclass, Trees, Gradient Boosting

Due: Tuesday, May 1, 2020 at 11:59pm

Instructions. You should upload your code and plots to Gradescope. Please map the Gradescope entry on the rubric to your name and NetId. You must follow the policies for submission detailed in Homework 0.

---

1 SGD for Multiclass Linear SVM

Suppose our output space and our action space are given as follows: $\mathcal{Y} = \mathcal{A} = \{1, \ldots, k\}$. Given a non-negative class-sensitive loss function $\Delta : \mathcal{Y} \times \mathcal{A} \to [0, \infty)$ and a class-sensitive feature mapping $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}^d$. Our prediction function $f : \mathcal{X} \to \mathcal{Y}$ is given by

$$f_w(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \Psi(x, y) \rangle$$

For training data $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, let $J(w)$ be the ℓ_2-regularized empirical risk function for the multiclass hinge loss. We can write this as

$$J(w) = \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max_{y \in \mathcal{Y}} \left[\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle \right],$$

for some $\lambda > 0$.

1. Show that $J(w)$ is a convex function of w. You may use any of the rules about convex functions described in our notes on Convex Optimization, in previous assignments, or in the Boyd and Vandenberghe book, though you should cite the general facts you are using. [Hint: If $f_1, \ldots, f_m : \mathbf{R}^n \to \mathbf{R}$ are convex, then their pointwise maximum $f(x) = \max\{f_1(x), \ldots, f_m(x)\}$ is also convex.]
2. Since $J(w)$ is convex, it has a subgradient at every point. Give an expression for a subgradient of $J(w)$. You may use any standard results about subgradients, including the result from an earlier homework about subgradients of the pointwise maxima of functions. (Hint: It may be helpful to refer to $\hat{y}_i = \arg\max_{y \in \mathcal{Y}} [\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle]$.)
3. Give an expression for the stochastic subgradient based on the point (x_i, y_i).
4. Give an expression for a minibatch subgradient, based on the points $(x_i, y_i), \ldots, (x_{i+m-1}, y_{i+m-1})$.

2 [Optional] Hinge Loss is a Special Case of Generalized Hinge Loss

Let $\mathcal{Y} = \{-1, 1\}$. Let $\Delta(y, \hat{y}) = 1(y \neq \hat{y})$. If $g(x)$ is the score function in our binary classification setting, then define our compatibility function as

$$\begin{aligned} h(x, 1) &= g(x)/2 \\ h(x, -1) &= -g(x)/2. \end{aligned}$$

Show that for this choice of h, the multiclass hinge loss reduces to hinge loss:

$$\ell\left(h, (x, y)\right) = \max_{y' \in \mathcal{Y}} \left[\Delta\left(y, y'\right)\right) + h(x, y') - h(x, y)] = \max\left\{0, 1 - yg(x)\right\}$$

3 Multiclass Classification - Implementation

In this problem we will work on a simple three-class classification example. The data is generated and plotted for you in the skeleton code.

3.1 One-vs-All (also known as One-vs-Rest)

In this problem we will implement one-vs-all multiclass classification. Our approach will assume we have a binary base classifier that returns a score, and we will predict the class that has the highest score.

1. Complete the class OneVsAllClassifier in the skeleton code. Following the OneVsAllClassifier code is a cell that extracts the results of the fit and plots the decision region. Include these results in your submission.

3.2 Multiclass SVM

In this question, we will implement stochastic subgradient descent for the linear multiclass SVM, as described in lecture and in this problem set. We will use the class-sensitive feature mapping approach with the "multivector construction", as described in our multiclass classification lecture (slide 24).

1. Complete the skeleton code for multiclass SVM. Following the multiclass SVM implementation, we have included another block of test code. Make sure to include the results from these tests in your assignment, along with your code.

4 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 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.
2. Complete either the `compute_entropy` or `compute_gini` functions. 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. For visualization, you'll need to install `graphviz`.
3. [Optional] 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.

5 Gradient Boosting Machines

Recall the general gradient boosting algorithm[1], for a given loss function ℓ and a hypothesis space $\mathcal{F}$ of regression functions (i.e. functions mapping from the input space to $\mathbf{R}$):

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute:

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

[1] Besides the lecture slides, you can find an accessible discussion of this approach in http://www.saedsayad.com/docs/gbm2.pdf, in one of the original references http://statweb.stanford.edu/~jhf/ftp/trebst.pdf, and in this review paper http://web.stanford.edu/~hastie/Papers/buehlmann.pdf.

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

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

(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.

In this problem we'll derive a special case of the general gradient boosting framework: BinomialBoost.

1. 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. Similar to what we did in the ℓ_2-Boosting question, write an expression for h_m as an argmin over $\mathcal{F}$.

6 Gradient Boosting Implementation

This method goes by many names, including gradient boosting machines (GBM), generalized boosting models (GBM), AnyBoost, and gradient boosted regression trees (GBRT), among others. Although one of the nice aspects of gradient boosting is that it can be applied to any problem with a subdifferentiable loss function, here we'll keep things simple and consider the standard regression setting with square loss.

1. Complete the `gradient_boosting` class. As the base regression algorithm, you may use sklearn's regression tree. You should use the square loss for the tree splitting rule and the mean function for the leaf prediction rule. Run the code provided to build gradient boosting models on the classification and regression data sets, and include the plots generated. Note that we are using square loss to fit the classification data, as well as the regression data.

2. [Optional] Repeat the previous runs on the classification data set, but use a different classification loss, such as logistic loss or hinge loss. Include the new code and plots of your results. Note that you should still use the same regression tree settings for the base regression algorithm.