Homework 6: Multiclass, Trees, and Gradient Boosting

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.

1 Reformulations of Multiclass Hinge Loss

1.1 Multiclass setting review

Consider the multiclass output space $\mathcal{Y} = \{1, \ldots, k\}$. Suppose we have a base hypothesis space $\mathcal{H} = \{h : \mathcal{X} \times \mathcal{Y} \to \mathbf{R}\}$ from which we select a compatibility score function. Then our final multiclass hypothesis space is $\mathcal{F} = \left\{f(x) = \arg\max_{y \in \mathcal{Y}} h(x, y) \mid h \in \mathcal{H}\right\}$. Since functions in $\mathcal{F}$ map into $\mathcal{Y}$, our action space $\mathcal{A}$ and output space $\mathcal{Y}$ are the same. Nevertheless, we will write our class-sensitive loss function as $\Delta : \mathcal{Y} \times \mathcal{A} \to \mathbf{R}$, even though $\mathcal{Y} = \mathcal{A}$. We do this to indicate that the true class goes in the first slot of the function, while the prediction (i.e. the action) goes in the second slot. This is important because we do not assume that $\Delta(y, y') = \Delta(y', y)$. It would not be unusual to have this asymmetry in practice. For example, false alarms may be much less costly than no alarm when indeed something is going wrong.

In the spirit of empirical risk minimization, we would like to find $f \in \mathcal{F}$ minimizing the empirical cost-sensitive loss:

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \Delta\left(y_i, f(x_i)\right),$$

possibly with some regularization. But this is clearly intractable, since we already know binary classification is intractable and that's a special case of this formulation. In lecture we proposed an alternative, tractable objective function: the multiclass SVM based on the convex multiclass hinge loss.

1.2 Two versions of multiclass hinge loss (or generalized hinge loss)

In lecture, we defined the **margin** of the compatibility score function h on the ith example (x_i, y_i) for class y as

$$m_{i,y}(h) = h(x_i, y_i) - h(x_i, y).$$

We also gave a formulation of a multiclass SVM objective function, where the loss on an individual example (x_i, y_i) was

$$\ell_1(h,(x_i,y_i)) = \max_{y\in\mathcal{Y}-\{y_i\}} \left(\max\left[0, \Delta(y_i,y) - m_{i,y}(h)\right]\right).$$

There's an alternative formulation, called the **generalized hinge loss** in SSBD Section 17.2. There they define

$$\ell_2(h,(x_i,y_i)) = \max_{y\in\mathcal{Y}} \left[\Delta\left(y_i,y\right) + h(x_i,y) - h(x_i,y_i)\right].$$

1. Show that if $\Delta(y,y) = 0$ for all $y \in \mathcal{Y}$, then $\ell_2\left(h,(x_i,y_i)\right) = \ell_1(h,(x_i,y_i))$. [Hint: Note that $\max_{y\in\mathcal{Y}} \phi(y) = \max\left(\phi(y_i), \max_{y\in\mathcal{Y}-\{y_i\}} \phi(y_i)\right)$.]

   If $\Delta(y_i,y_i) = 0$:

   $\ell_1(h,(x_i,y_i)) = \ell_1(h,(x_i,y_i)) = \max_{y\in\mathcal{Y}-\{y_i\}} \left(\max\left[0, \Delta(y_i,y) - m_{i,y}(h)\right]\right)$

   $\ell_2(h,(x_i,y_i)) = \max_{y\in\mathcal{Y}} \left[\Delta\left(y_i,y\right) + h(x_i,y) - h(x_i,y_i)\right]$

   $= \max\left(\Delta(y_i,y_i) + h(x_i,y_i) - h(x_i,y_i), \max_{y\in\mathcal{Y}-\{y_i\}} \Delta(y_i,y) + h(x_i,y) - h(x_i,y_i)\right)$

   $= \max(0, \max_{y\in\mathcal{Y}-\{y_i\}} \Delta(y_i,y) + h(x_i,y) - h(x_i,y_i))$

   $= \ell_1$

   If $\Delta(y_i,y_j) = 0$ and $i \neq j$:

   $\ell_1(h,(x_i,y_i)) = \ell_1(h,(x_i,y_i)) = \max_{y\in\mathcal{Y}-\{y_i\}} \left(\max\left[0, -m_{i,y}(h)\right]\right)$

   $\ell_2(h,(x_i,y_i)) = \max_{y\in\mathcal{Y}} \left[-m_{i,y}(h)\right]$

   $= \max\left(-m_{i,y}(h)), \max_{y\in\mathcal{Y}-\{y_i\}} -m_{i,y}(h))\right)$

   $= \max(0, \max_{y\in\mathcal{Y}-\{y_i\}} [-m_{i,y}(h)])$

   $= \ell_1$

2. In the context of the generalized hinge loss, we've said that $\Delta(y_i,y)$ is like the "target margin" between the score for true class y_i and the score for class y. Suppose that for our compatibility function h, all target margins are reached or exceeded on x_i. That is

   $$m_{i,y}(h) = h(x_i,y_i) - h(x_i,y) \geq \Delta(y_i,y),$$

   for all $y \in \mathcal{Y} - \{y_i\}$. Assume that $\Delta(y_i,y) > 0\ \forall y \neq y_i$ and $\Delta(y_i,y) = 0$ for $y = y_i$.]

(a) Show that under the conditions above, $\ell_1(h,(x_i,y_i)) = \ell_2(h,(x_i,y_i)) = 0$.

$$\begin{aligned}
\ell_1(h,(x_i,y_i)) &= \max_{y\in\mathcal{Y}-\{y_i\}} (\max[0, \Delta(y_i,y) - m_{i,y}(h)]) \\
&= \max_{y\in\mathcal{Y}-\{y_i\}} (0) = 0 \\
\ell_2(h,(x_i,y_i)) &= \max_{y\in Y}(\Delta(y_i,y) - m_{i,y}(h)) \\
&= \max(\Delta(y_i,y_i) - m_{i,y_i}(h), \max_{y\in Y-y_i}(\Delta(y_i,y) - m_{i,y}(h)) \\
&= \max(0, \max_{y\in Y-y_i}(\Delta(y_i,y) - m_{i,y}(h)) \\
&= 0
\end{aligned}$$

(b) Show that under the conditions above, we make the correct prediction on x_i. That is, $f(x_i) = \arg\max_{y\in\mathcal{Y}} h(x_i,y) = y_i$.
Based on the conclusion above, for any $y_j, j \neq i$, we have $h(x_i,y_j) < h(x_i,y_i)$ since $\Delta(y_i,y_j) > 0$, thus $l_{1,2}(h,(x_i,y_j)) > 0$, by minimizing l loss function, we can get $f(x_i) = argmax_{y\in Y} h(x_i,y) = y_i$.

2 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. [Optional] 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.]
According to the notes, $\max_{y\in\mathcal{Y}}[\Delta(y_i,y) + \langle w, \Psi(x_i,y) - \Psi(x_i,y_i)\rangle]$ is a convex function. So the point-wise sum $\sum_{i=1}^n \max_{y\in\mathcal{Y}}[\Delta(y_i,y) + \langle w, \Psi(x_i,y) - \Psi(x_i,y_i)\rangle]$ is also convex. And $\|w\|$ is convex function $\implies$ $\|w\|^2$ is convex(composition.) Thus $\lambda\|w\|^2$ is convex. And by summation of convex funtions is still convex, $J(w) = \lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^n \max_{y\in\mathcal{Y}}[\Delta(y_i,y) + \langle w, \Psi(x_i,y) - \Psi(x_i,y_i)\rangle]$ is a convex function.

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 $\hat{y}_i = \arg\max_{y \in \mathcal{Y}} [\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle]$.)

$$\hat{y}_i = \arg\max_{y \in \mathcal{Y}} [\Delta(y_i, y) + \langle w, \Psi(x_i, y) - \Psi(x_i, y_i) \rangle]$$

$$\implies J(w) = \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} [\Delta(y_i, \hat{y}_i) + \langle w, \Psi(x_i, \hat{y}_i) - \Psi(x_i, y_i) \rangle]$$

$$\text{The subgradient of } J(w) = 2\lambda w + \frac{1}{n} \sum_{i=1}^{n} (\Psi(x_i, \hat{y}_i) - \Psi(x_i, y_i))$$

3. Give an expression for the stochastic subgradient based on the point (x_i, y_i).

$$2\lambda w + (\Psi(x_i, \hat{y}_i) - \Psi(x_i, y_i))$$

4. Give an expression for a minibatch subgradient, based on the points $(x_i, y_i), \ldots, (x_{i+m-1}, y_{i+m-1})$.

$$2\lambda w + \frac{1}{m} \sum_{i=1}^{i+m-1} (\Psi(x_i, \hat{y}_i) - \Psi(x_i, y_i))$$

3 [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:

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

$$\begin{aligned} \text{When } y = 1 \text{ and } y \neq y', \ell(h, (x, y)) &= \max_{y' \in \mathcal{Y}} [1 + h(x, -1) - h(x, 1)] \\ &= \max_{y' \in \mathcal{Y}} (1 - \frac{g(x)}{2} - \frac{g(x)}{2}) \\ &= \max\{0, 1 - yg(x)\} \\ \text{When } y = -1 \text{ and } y \neq y', \ell(h, (x, y)) &= \max_{y' \in \mathcal{Y}} [1 + h(x, 1) - h(x, -1)] \\ &= \max_{y' \in \mathcal{Y}} (1 + \frac{g(x)}{2} + \frac{g(x)}{2}) \\ &= \max\{0, 1 - yg(x)\} \end{aligned}$$

$$\text{When } y = y', \ell = 0$$

4 Multiclass Classification - Implementation

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

4.1 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.

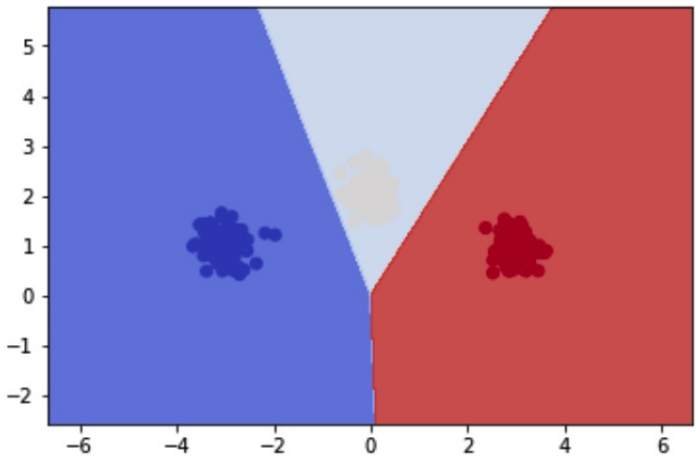
```
def fit(self, X, y=None):
    """
    This should fit one classifier for each class.
    self.estimators[i] should be fit on class i vs rest
    @param X: array-like, shape = [n_samples,n_features], input data
    @param y: array-like, shape = [n_samples,] class labels
    @return returns self
    """
    #Your code goes here
    y_fit={}
    for i in range(self.n_classes):
        y_fit[i]=(np.where(y==i,1,0))
    for i in range(self.n_classes):
        self.estimators[i].fit(X,y_fit[i])
    self.fitted = True
    return self
```

```
def decision_function(self, X):
    """
    Returns the score of each input for each class. Assumes
    that the given estimator also implements the decision_function method (which
    and that fit has been called.
    @param X : array-like, shape = [n_samples, n_features] input data
    @return array-like, shape = [n_samples, n_classes]
    """
    if not self.fitted:
        raise RuntimeError("You must train classifer before predicting data.")

    if not hasattr(self.estimators[0], "decision_function"):
        raise AttributeError(
            "Base estimator doesn't have a decision_function attribute.")

    #Replace the following return statement with your code
    if len(X.shape)==1:
        score=np.zeros([self.n_classes])
        score=self.estimators[i].decision_function(X)
        score=score[np.newaxis, :]
        return score

    else:
        score=np.zeros([self.n_classes,X.shape[0]])
        for i in range(self.n_classes):
            score[i]=self.estimators[i].decision_function(X)
        score=score.T
        return score
```

```
def predict(self, X):
    """
    Predict the class with the highest score.
    @param X: array-like, shape = [n_samples,n_features] input data
    @returns array-like, shape = [n_samples,] the predicted classes for each inp
    """
    #Replace the following return statement with your code
    score=self.decision_function(X)
    predicted_y=np.zeros([score.shape[0]])
    for i in range(len(predicted_y)):
        predicted_y[i]=np.where(score[i]==max(score[i]))[0][0]
    return predicted_y
```

```
Coeffs 0
[[-1.05852747 -0.90296521]]
Coeffs 1
[[ 0.22117096 -0.38900908]]
Coeffs 2
[[ 0.89162796 -0.82467394]]
```

```
array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```



4.2 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 and in SSBD Section 17.2.1.

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.

```
def featureMap(X,y,num_classes) :
    '''
    Computes the class-sensitive features.
    @param X: array-like, shape = [n_samples,n_inFeatures] or [n_inFeatures,], inp
    @param y: a target class (in range 0,..,num_classes-1)
    @return array-like, shape = [n_samples,n_outFeatures], the class sensitive fea
    '''
    #The following line handles X being a 1d-array or a 2d-array
    num_samples, num_inFeatures = (1,X.shape[0]) if len(X.shape) == 1 else (X.shap
    if len(X.shape) == 1:
        X=X[np.newaxis,:]
    y_n=np.array([y]) if type(y)!=np.ndarray else y
    n_outFeatures=num_classes*num_inFeatures
    X_out=np.zeros([num_samples,n_outFeatures])
    for i in range(num_samples):
        X_out[i][y_n[i]*num_inFeatures:(y_n[i]+1)*num_inFeatures]=X[i]
    return X_out
```

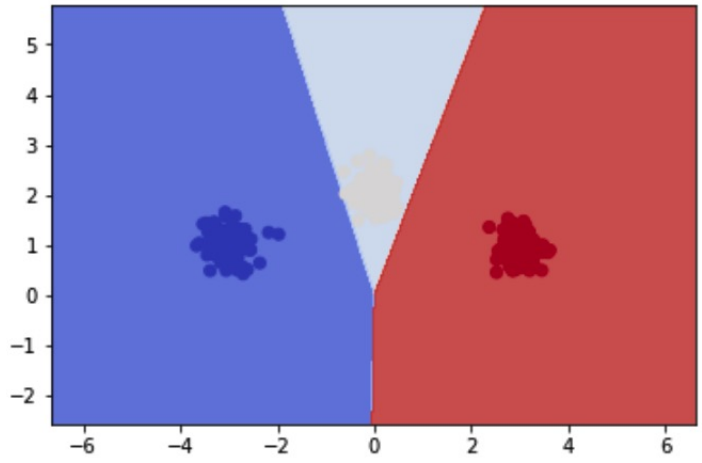
```
def sgd(X, y, num_outFeatures, subgd, eta = 0.001, T = 10000):
    '''
    Runs subgradient descent, and outputs resulting parameter vector.
    @param X: array-like, shape = [n_samples,n_features], input training data
    @param y: array-like, shape = [n_samples,], class labels
    @param num_outFeatures: number of class-sensitive features
    @param subgd: function taking x,y and giving subgradient of objective
    @param eta: learning rate for SGD
    @param T: maximum number of iterations
    @return: vector of weights
    '''
    num_samples = X.shape[0]
    w=np.zeros(num_outFeatures)
    for t in range(T):
        idx=np.arange(num_samples)
        shuffle(idx)
        X=X[idx]
        y=y[idx]
        local_sg = subgd(X[i],y[i],w)
        w=w-eta*local_sg
    return w
```

```
 def subgradient(self,x,y,w):
     '''
     Computes the subgradient at a given data point x,y
     @param x: sample input
     @param y: sample class
     @param w: parameter vector
     @return returns subgradient vector at given x,y,w
     '''
     #Your code goes here and replaces the following return statement
     y_max=0
     local_max=self.Delta(y,y_max)+np.dot(w,(self.Psi(x,y_max)-self.Psi(x,y)).T)
     for y_i in range(self.num_classes):
         max_=self.Delta(y,y_i)+np.dot(w,(self.Psi(x,y_i)-self.Psi(x,y)).T)
         if max_ > local_max:
             local_max = max_
             y_max=y_i
     local_sgd=2*self.lam*w+self.Psi(x,y_max)-self.Psi(x,y)
     return(local_sgd)
```

```
def decision_function(self, X):
    '''
    Returns the score on each input for each class. Assumes
    that fit has been called.
    @param X : array-like, shape = [n_samples, n_inFeatures]
    @return array-like, shape = [n_samples, n_classes] giving scores for each samp
    '''
    if not self.fitted:
        raise RuntimeError("You must train classifer before predicting data.")

    #Your code goes here and replaces following return statement
    if not self.fitted:
        raise RuntimeError("You must train classifer before predicting data.")

    num_samples=X.shape[0]
    score=np.zeros([num_samples,self.num_classes])
    for i in range(num_samples):
        for j in range(self.num_classes):
            X_out=self.Psi(X[i],j)
            score[i][j]=np.dot(self.coef_,X_out.T)
    return(score)
```

```python
def predict(self, X):
    '''
    Predict the class with the highest score.
    @param X: array-like, shape = [n_samples, n_inFeatures], input data to predic
    @return array-like, shape = [n_samples,], class labels predicted for each dat
    '''

    #Your code goes here and replaces following return statement
    score_=self.decision_function(X)
    score=np.zeros(score_.shape[0])
    for i in range(score_.shape[0]):
        score[i]=np.where(score_[i]==max(score_[i]))[0][0]
    return(score)
```

```
w:
[[-0.35320673 -0.03521964  0.01063275  0.08368118  0.34257398 -0.04846154]]

array([[100,   0,   0],
       [  0, 100,   0],
       [  0,   0, 100]])
```



5 [Optional] Audio Classification

In this problem, we will work on the urban sound dataset URBANSOUND8K from the Center for Urban Science and Progress (CUSP) at NYU. (You should download the data from that link.) We will first extract features from raw audio data using the LibROSA package, and then we will train multiclass classifiers to classify the sounds into 10 sound classes. URBANSOUND8K dataset contains 8732 labeled sound excerpts broken into 10 folds. Use folds 1 and 2 for training, and folds 3 and 4 for validation.

1. In LibROSA, there are many functions for visualizing audio waves and spectra, such as display.waveplot() and display.specshow(). Load a random audio file from each class as a floating point time series with librosa.load(), and plot their waves and linear-frequency power spectrogram. If you are interested, you can also play the audio in the notebook with functions display() and Audio() in IPython.display.

2. Mel-frequency cepstral coefficients (MFCC) are a commonly used feature for sound processing. We will use MFCC and its first and second differences (like discrete derivatives) as our features for classfication. First, use function feature.mfcc() from LibROSA to extract MFCC features from each audio sample. (The first MFCC coefficient is typically discarded in sound analysis, but you do not need to. You can test whether this helps in the optional problem below.) Next, use function feature.delta() to calculate the first and second differences of MFCC. Finally, combine these features and normalize each feature to zero mean and unit variance.

3. Train a linear multiclass SVM on your training set. Evaluate your results on the validation set in terms of 0/1 error and generate a confusion table. Compare the results to a one-vs-all classifier using a binary linear SVM as the base classifier. For each model, may use your code from the previous problem, or you may use another implementation (e.g. from sklearn).

4. [More Optional] Compare results to any other multiclass classification methods of your choice.

5. [More Optional] Try different feature sets and see how they affect performance.

6 [Optional] 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. [Optional] 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. [Optional] 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.

7 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:
   $$\mathbf{g}_m = \left(\frac{\partial}{\partial f(x_j)} \sum_{i=1}^{n} \ell\left(y_i, f(x_i)\right) \Big|_{f(x_i)=f_{m-1}(x_i),\, i=1,\ldots,n} \right)_{j=1}^{n}$$
   (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 two special cases of the general gradient boosting framework: ℓ_2-Boosting and BinomialBoost.

1. Consider the regression framework, where $\mathcal{Y} = \mathbf{R}$. Suppose our loss function is given by
   $$\ell(\hat{y}, y) = \frac{1}{2} \left(\hat{y} - y\right)^2 ,$$
   and at the beginning of the m'th round of gradient boosting, we have the function $f_{m-1}(x)$. Show that the h_m chosen as the next basis function is given by
   $$h_m = \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left[\left(y_i - f_{m-1}(x_i)\right) - h(x_i) \right]^2 .$$

[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.

In other words, at each stage we find the base prediction function $h_m \in \mathcal{F}$ that is the best fit to the residuals from the previous stage. [Hint: Once you understand what's going on, this is a pretty easy problem.]

$$\begin{aligned}
\ell(y_i, f(x_i)) &= \frac{1}{2}(y_i - f(x_i))^2 \\
(\mathbf{g}_m)_i &= f_{m-1}(x_i) - y_i \\
h_m &= \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left((-\mathbf{g}_m)_i - h(x_i)\right)^2 \\
&= \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left[(y_i - f_{m-1}(x_i)) - h(x_i)\right]^2
\end{aligned}$$

2. Now 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 $\mathcal{F}$.

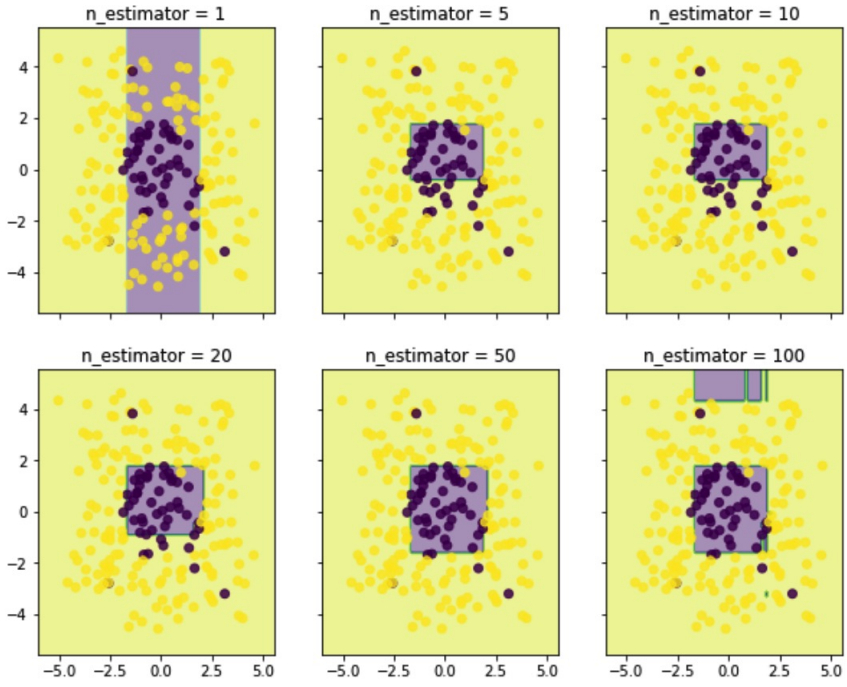
$$\begin{aligned}
\ell(m) &= \ln\left(1 + e^{-m}\right) \\
(\mathbf{g}_m)_i &= \frac{\partial ln(1 + e^{-y_i f_{m-1}(x_i)})}{\partial f_{m-1}(x_i)} \\
&= \frac{-y_i e^{-y_i f_{m-1}(x_i)}}{1 + e^{-y_i f_{m-1}(x_i)}} \\
h_m &= \arg\min_{h \in \mathcal{F}} \sum_{i=1}^{n} \left[\frac{-y_i e^{-y_i f_{m-1}(x_i)}}{1 + e^{-y_i f_{m-1}(x_i)}} - h(x_i)\right]^2
\end{aligned}$$

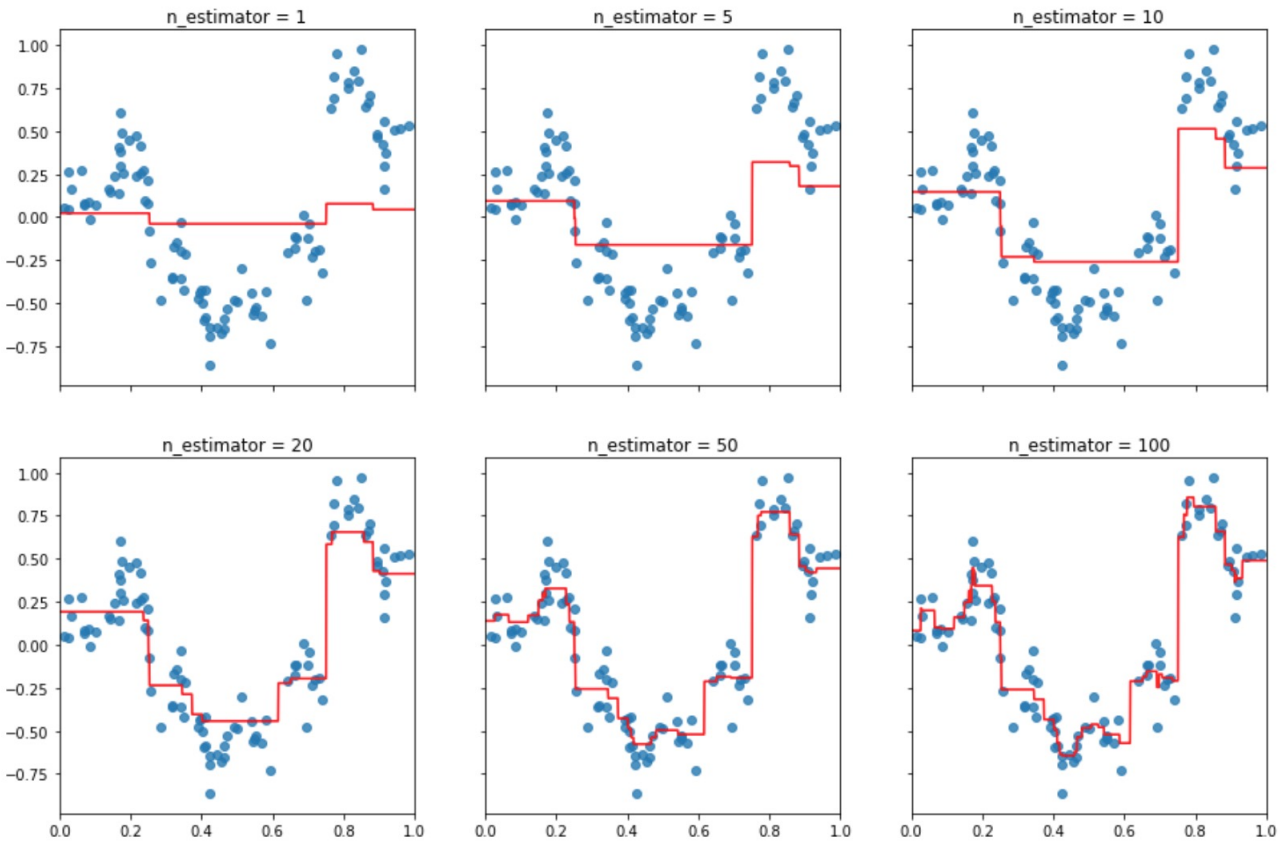
8 Gradient Boosting Implementation

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.

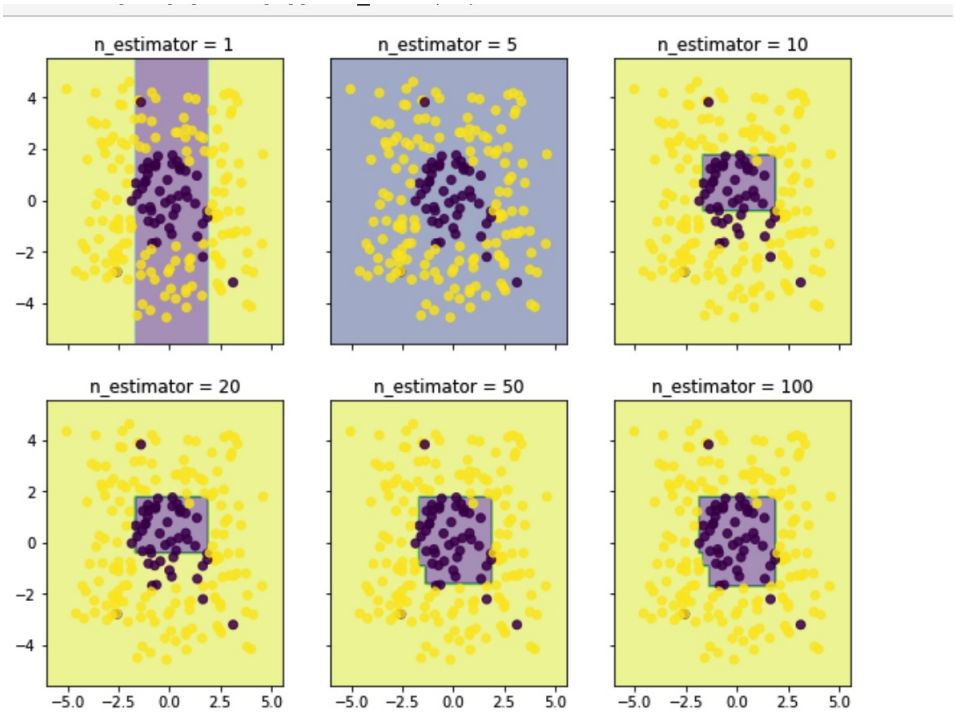
```
def current_estimator(self,x,back=0):
    predicted_y=np.zeros([len(x)])
    for i in range(len(self.estimators)-back):
        predicted_y+=self.learning_rate*self.estimators[i].predict(x)
    predicted_y=predicted_y[:,np.newaxis]
    return predicted_y

def fit(self, train_data, train_target):
    '''
    Fit gradient boosting model
    '''
    # Your code goes here
    self.estimators=[]
    for i in range(self.n_estimator):
        self.estimators.append(DecisionTreeRegressor(min_samples_split=self.min_sample,max_depth=self.max_de
        self.estimators[i].fit(train_data,self.pseudo_residual_func(train_target,self.current_estimator(trai
    return self

def predict(self, test_data):
    '''
    Predict value
    '''
    # Your code goes here
    predicted_y=self.current_estimator(test_data,0)
    return predicted_y
```

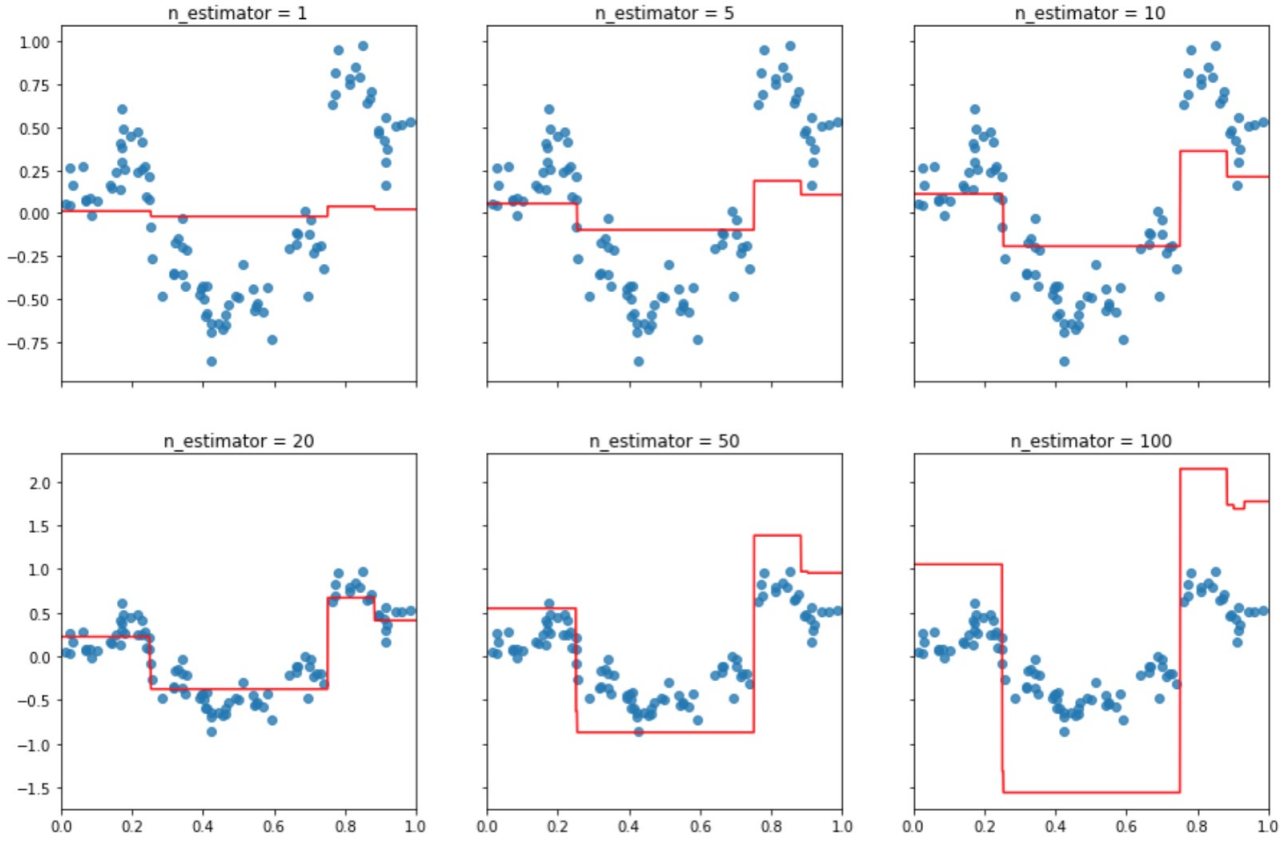




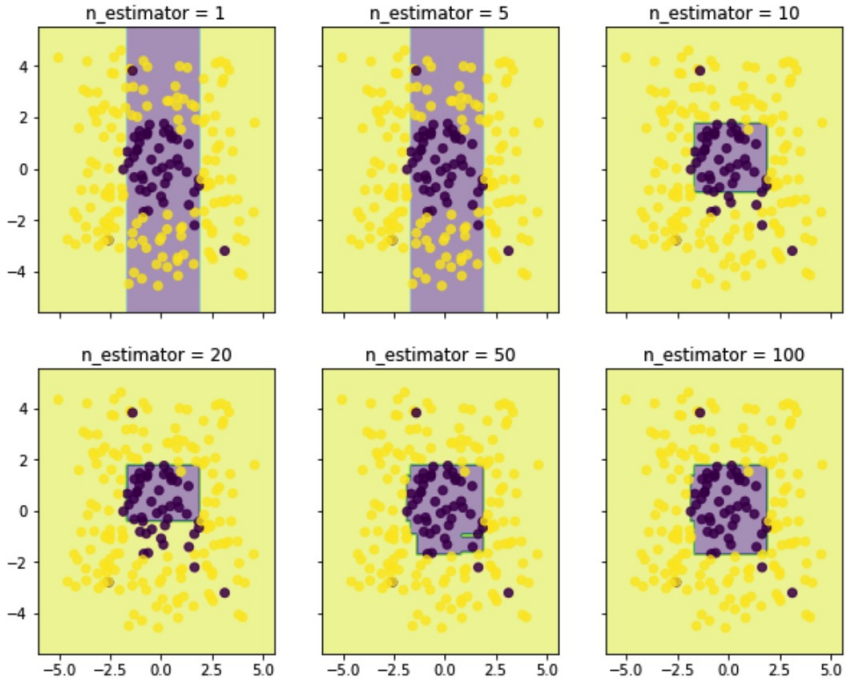
2. [Optional] Repeat the previous runs on the classification data set, but use a different classification loss, such as logistic loss or hinge loss. Include the new code and plots of your results. Note that you should still use the same regression tree settings for the base regression algorithm.

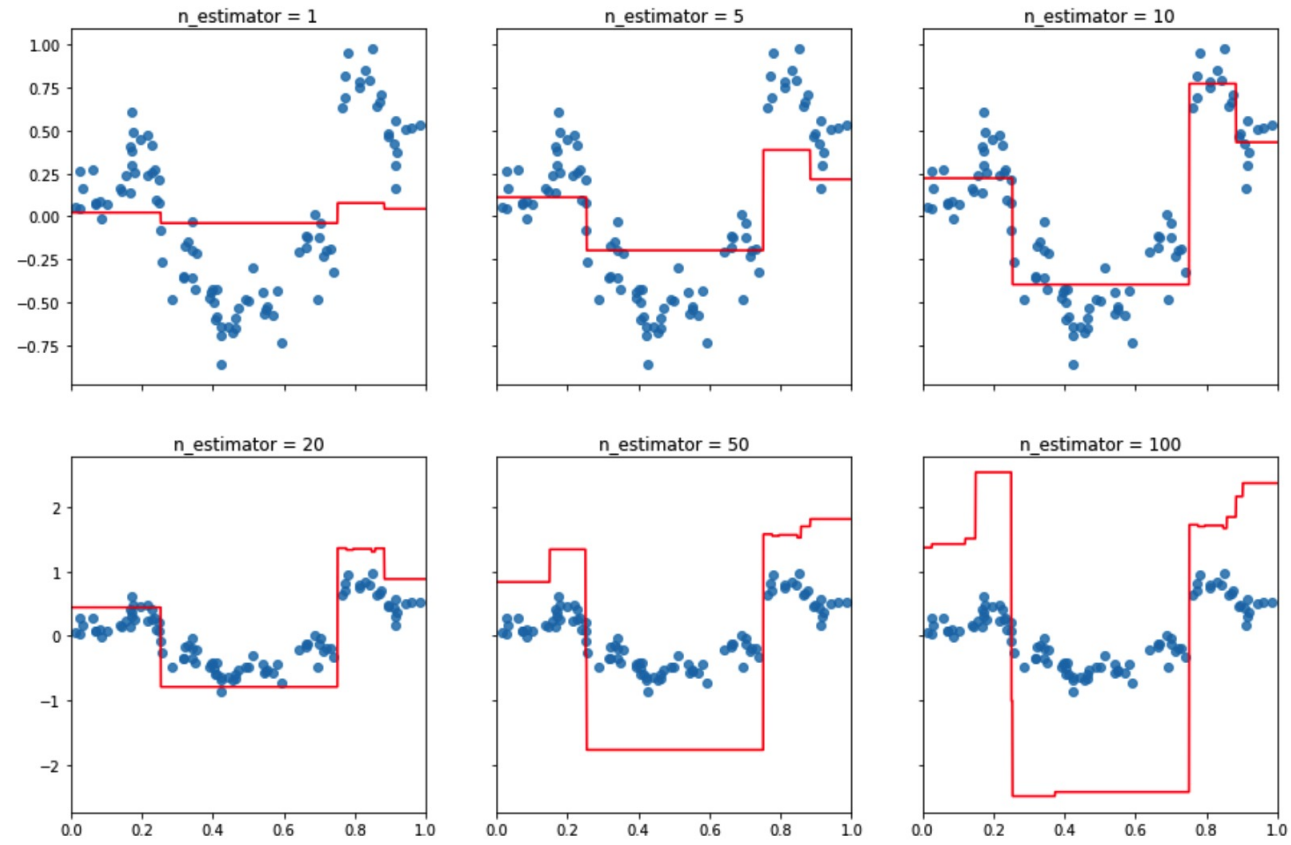
```
def pseudo_residual_Log(train_target, train_predict):
    num_samp=len(train_target)
    out_res=np.zeros(num_samp)
    for i in range(num_samp):
        out_res[i]=train_target[i]*np.exp(-(train_target[i]*train_predict[i]))/(1+np.exp(-(train_target[i]*tr

    return out_res
```


n_estimator = 1
n_estimator = 5
n_estimator = 10
n_estimator = 20
n_estimator = 50
n_estimator = 100
4
2
0
−2
−4
−5.0
−2.5
0.0
2.5
5.0




```
def pseudo_residual_hinge(train_target, train_predict):

    num_samp=len(train_target)
    out_res=np.zeros(num_samp)
    for i in range(num_samp):
        if max(0,1-train_target[i]*train_predict[i])==0:
            out_res[i]=0
        else:
            out_res[i]=train_target[i]

    return out_res
```


n_estimator = 1
n_estimator = 5
n_estimator = 10
n_estimator = 20
n_estimator = 50
n_estimator = 100
4
2
0
−2
−4
−5.0
−2.5
0.0
2.5
5.0


n_estimator = 1
n_estimator = 5
n_estimator = 10
n_estimator = 20
n_estimator = 50
n_estimator = 100