Question 3

Please submit Question3.pdf on Moodle using the Final Exam - Question 3 object. You must submit a single PDF. You may submit multiple .py files if you wish. The parts are worth 4 + 3 + 8 + 1 + 3 + 2 + 3 + 1 = 25.

In the 9417 group project, many of you applied gradient boosting, which is sometimes regarded as the best out-of-the-box learning algorithm. In this question, we will derive and implement the gradient boosting algorithm from scratch, and apply it to a toy data set. The gradient boosting algorithm can be described as follows: Let $\mathcal{F}$ be a set of base learning algorithms[6], and let $\ell(y, \hat{y})$ be a loss function, where y is the target, and $\hat{y}$ is the predicted value. Note that this set-up is general enough to include both regression and classification algorithms. Suppose we have data (x_i, y_i) for $i = 1, \ldots, n$, which we collect into a single data set D_0. We then set the number of desired base learners to T, and proceed as follows:

(I) Initialize $f_0(x) = 0$ (i.e. f_0 is the zero function.)

(II) For $t = 1, 2, \ldots, T$:

(GB1) Compute:

$$r_{t,i} = -\frac{\partial}{\partial f(x_i)} \sum_{j=1}^{n} \ell(y_j, f(x_j)) \Big|_{f(x_j)=f_{t-1}(x_j),\ j=1,\ldots,n}$$

for $i = 1, \ldots, n$. We refer to $r_{t,i}$ as the i-th pseudo-residual at iteration t.

(GB2) Construct a new pseudo data set, D_t, consisting of pairs: (x_i, r_{ti}) for $i = 1, \ldots, n$.

(GB3) Fit a model to D_t using our base class $\mathcal{F}$. That is, we solve

$$h_t = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \ell(r_{t,i}, f(x_i))$$

(GB4) Choose a step-size. This can be done by **either** of the following methods:

(SS1) Pick a fixed step-size $\alpha_t = \alpha$

(SS2) Pick a step-size adaptively according to

$$\alpha_t = \arg\min_{\alpha} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i) + \alpha h_t(x_i)).$$

(GB5) Take the step

$$f_t(x) = f_{t-1}(x) + \alpha_t h_t(x).$$

(III) return f_T.

We can view this algorithm as either a generalization of AdaBoost covered in lectures/labs, or as performing (functional) gradient descent on the base class $\mathcal{F}$. Note that in (GB1), the notation means that after taking the derivative with respect to $f(x_i)$, set all occurences of $f(x_j)$ int he resulting expression with the prediction of the current model $f_{t-1}(x_j)$, for all j. For example,

$$\frac{\partial}{\partial x} \log(x+1) \Big|_{x=23} = \frac{1}{x+1} \Big|_{x=23} = \frac{1}{24}.$$

[6] For example, you could take $\mathcal{F}$ to be the set of all depth-1 decision trees, often referred to as decision stumps. Alternatively, $\mathcal{F}$ could be the set of all depth-2 trees, or the set of all 1-layer neural networks, etc.

(a) Consider the regression setting where we allow the y-values in our data set to be real numbers. Suppose that we use squared error loss $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$. For round t of the algorithm, show that $r_{t,i} = y_i - f_{t-1}(x_i)$. Then, write down an expression for the optimization problem in step (GB3) that is specific to this setting (you don't need to actually solve it).

What to submit: your working out, either typed or handwritten.

(b) Using the same setting as in the previous part, show that the adaptive approach (SS2) leads to a step-size:

$$\alpha_t = \frac{\sum_{i=1}^{n} h_t(x_i)(y_i - f_{t-1}(x_i))}{\sum_{i=1}^{n} (h_t(x_i))^2}.$$

What to submit: your working out, either typed or handwritten.

(c) We will now implement gradient boosting on a toy dataset from scratch, and we will use the class of decision stumps (depth 1 decision trees) as our base class ($\mathcal{F}$), and squared error loss as in the previous parts. In your implementation, you may make use of `sklearn.tree.DecisionTreeRegressor`, but all other code must be your own[7]. The following code generates the data and demonstrates plotting the predictions of a fitted decision tree (more details in `q3.py`):

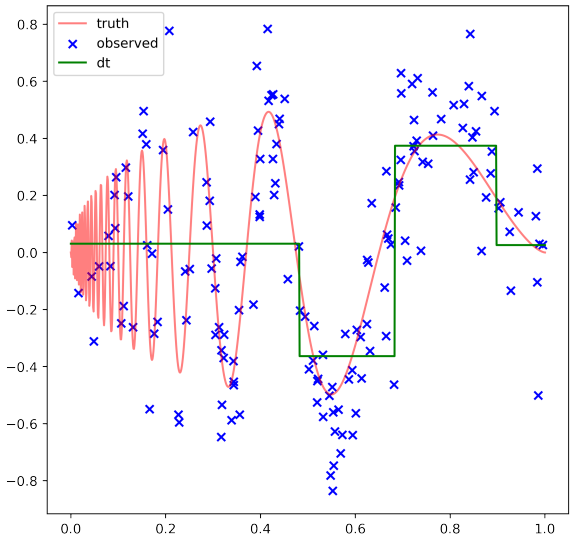
```
np.random.seed(123)
X, y = f_sampler(f, 160, sigma=0.2)
X = X.reshape(-1,1)

fig = plt.figure(figsize=(7,7))
dt = DecisionTreeRegressor(max_depth=1).fit(X,y)  # learn a model
xx = np.linspace(0,1,1000)
plt.plot(xx, f(xx), alpha=0.5, color='red', label='truth')
plt.scatter(X,y, marker='x', color='blue', label='observed')
plt.plot(xx, dt.predict(xx.reshape(-1,1)), color='green', label='dt')  # plotting
example model
plt.legend()
plt.show()

```

The figure generated is

[7] you can use `NumPy` and `matplotlib`, but do not use an existing implementation of gradient boosting.



Your task is to generate a 5 x 2 figure of subplots showing the predictions of your fitted gradient boosted model. There are 10 subplots in total, the first should show the model with 5 base learners, the second subplot should show it with 10 base learners, etc. The last subplot should be the gradient boosted model with 50 base learners. Each subplot should include the scatter of data, as well as a plot of the true model (basically, the same as the plot provided above but with your implemented gradient boosted model in place of `dt`). Comment on your results, what happens as the number of base learners is increased? You should do this two times (two 5x2 plots), once with the adaptive step size, and the other with the step-size taken to be $\alpha = 0.1$ fixed throughout. There is no need to split into train and test data here. Comment on the differences between your fixed and adaptive step-size implementations. How does your model perform on different parts of the data?

What to submit: two 5 x 2 plots, one for adaptive and one for fixed step size, some commentary, and a screen shot of your code and a copy of your code in your .py file.

(d) Repeat the analysis in the previous question but with depth 2 decision trees as base learners instead. Provide the same plots. What do you notice for the adaptive case? What about the non-adaptive case? *What to submit: two 5 x 2 plots, one for adaptive and one for fixed step size, some commentary, and a copy of your code in your .py file.*

(e) Now, consider the classification setting where y is taken to be an element of $\{-1, 1\}$. We consider the following classification loss: $\ell(y, \hat{y}) = \log(1 + e^{-y\hat{y}})$. For round t of the algorithm, what is $r_{t,i}$? Write down an expression for the optimization problem in step (GB3) that is specific to this setting (you don't need to actually solve it).

What to submit: your working out, either typed or handwritten.

(f) Using the same setting as in the previous part, write down an expression for α_t using the adaptive approach in (SS2). Can you solve for α_t in closed form? Explain.

What to submit: your working out, either typed or handwritten, and some commentary.

(g) Consider a different classification loss: $\ell(y, \hat{y}) = e^{-y\hat{y}}$. For round t of the algorithm, what is $r_{t,i}$? Write down an expression for the optimization problem in step (GB3) that is specific to this setting (you don't need to actually solve it). Further, can you find an expression for α_t in (SS2) for this setting? Explain.

What to submit: your working out, either typed or handwritten.

(h) In practice, if you cannot solve for α_t exactly, explain how you might implement gradient boosting. Assume that using a constant step-size is not a valid alternative. Be as specific as possible in your answer. What, if any, are the additional computational costs of your approach relative to using a constant step size ?

What to submit: some commentary.