1 Gradient Boosting Machines

In this problem we'll derive two special cases of the general gradient boosting framework: L2-Boosting and BinomialBoost.

1. Consider the regression framework, where $\mathcal{Y} = \mathbf{R}$. Suppose our loss function is given by

$$\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2,$$

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

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

In other words, at each stage we find the weak prediction function $h_m \in F$ 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.]

The expression for $(g_m)_i$ which is the ith component of (g_m) can be given as:

$(g_m)_i = \frac{\partial}{\partial f_{m-1}(x_i)} [\frac{1}{2}(y_i - f_{m-1}(x_i)^2]$

$= f_{m-1}(x_i) - y_i$

So (g_m) is $= \sum_{i=1}^{n} [f_{m-1}(x_i) - y_i]$

Now h_m can be written as:

$h_m = argmin_{h \in F} \sum_{i=1}^{n} [(-g_m)_i - h(x_i)]^2$

Substituting the value of (g_m),

$= argmin_{h \in F} \sum_{i=1}^{n} [(y_i - f_{m-1}(x_i)) - h(x_i)]^2$

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 not very robust to outliers (i.e. outliers can have a large effect on the final prediction function). Instead, let's consider instead the logistic loss

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

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 $\mathcal{F}$.

Using the same expression for $(g_m)_i$ as in the previous part with $m = y_i f_{m-1}(x_i)$,

$(g_m)_i = \frac{\partial}{\partial f_{m-1}(x_i)}[\ell(y_i, f_{m-1}(x_i))]$

$= \frac{\partial}{\partial f_{m-1}(x_i)}[ln(1 + e^{-y_i f_{m-1}(x_i)})]$

$= \frac{-y_i}{1+e^{y_i f_{m-1}(x_i)}}$

Using the expression for h_m from the previous part,

$h_m = argmin_{h \in F} \sum_{i=1}^{n} [(-g_m)_i - h(x_i)]^2$

Substituting g_m,

$= argmin_{h \in F} \sum_{i=1}^{n} [\frac{-y_i}{1+e^{y_i f_{m-1}(x_i)}} - h(x_i)]^2$

2 From Margins to Conditional Probabilities

1. Write $E_y[\ell(yf(x))|x]$ in terms of $\pi(x)$ and $\ell(f(x))$. [Hint: Use the fact that $y \in -1, 1$.]

Since $y \in -1, 1$

$E_y[\ell(yf(x))|x] = \pi(x)E[\ell f(x)|x] + (1 - \pi(x))E[\ell(-f(x))|x]$

$= \pi(x)\ell(f(x)) + (1 - \pi(x)\ell(-f(x)))$

2. Show that the Bayes prediction function $f^*(x)$ for the exponential loss function $\ell(y, f(x)) = e^{-yf(x)}$ is given by

$$f^*(x) = \frac{1}{2}ln(\frac{\pi(x)}{1-\pi(x)})$$

and, given the Bayes prediction function f^*, we can recover the conditional probabilities by

$$\pi(x) = \frac{1}{1+e^{-2f^*(x)}}.$$

[Hint: Differentiate the expression in the previous problem with respect to $f(x)$. If this is confusing, you may find it more comforting to change variables a bit: Fix an $x \in X$. Then write $p = \pi(x)$ and $\hat{y} = f(x)$. After substituting these into the expression you had for the previous problem, you'll want to find $\hat{y}$ that minimizes the expression. Use differential calculus. Once you've done it for a single x, it's easy to write the solution as a function of x.]

Differentiate $E_y[\ell y f(x)|x]$ with respect to $\hat{y}$,

$p\ell'(\hat{y}) + (1-p)\ell'(-\hat{y}) = 0$

$p[-e^{-\hat{y}}] + (1-p)[e^{\hat{y}}] = 0$

$\frac{p}{1-p} = e^{2\hat{y}}$

Solving for $\hat{y}$, we get

$\hat{y} = \frac{1}{2}ln(\frac{p}{1-p})$

Therefore,

$f^*(x) = \frac{1}{2}ln(\frac{\pi(x)}{1-\pi(x)})$

Solving for p, we get

$p = \frac{e^{2\hat{y}}}{1+e^{2\hat{y}}}$

Which can be written as,

$\pi(x) = \frac{e^{2f^*(x)}}{1+e^{2f^*(x)}}$

Simplifying,

$\pi(x) = \frac{1}{1+e^{-2f^*(x)}}$

3. Show that the Bayes prediction function $f^*(x)$ for the logistic loss function $\ell(y, f(x)) = ln(1 + e^{-yf(x)})$ is given by

$$f^*(x) = ln(\frac{\pi(x)}{1 - \pi(x)})$$

and the conditional probabilities are given by

$$\pi(x) = \frac{1}{1 + e^{-f^*(x)}}.$$

Again, we may assume that $\pi(x) \in (0, 1)$.

Similar to the last part, we differentiate with respect to f(x),

$\pi(x)\ell^{'}(f(x)) + (1 - \pi(x))\ell^{'}(-f(x)) = 0$

Simplifying the equation and substituting the loss function, we get,

$\frac{(1-\pi(x))e^{f(x)}}{1+e^{f(x)}} = \frac{\pi(x)}{1+e^{f(x)}}$

$\implies (1 - \pi(x))e^{f(x)} - \pi(x) = 0$

Solving for f(x), we get that,

$f^*(x) = ln\frac{\pi(x)}{1-\pi(x)}$

Similarly, solving for $\pi(x)$, we get

$\pi(x) = \frac{1}{1+e^{-f^*(x)}}$

4. Show that the Bayes prediction function $f^*(x)$ for the hinge loss function $\ell(y, f(x)) = max(0, 1 - yf(x))$ is given by

$$f^*(x) = sign(\pi(x) - \frac{1}{2}).$$

Note that it is impossible to recover $\pi(x)$ from $f^*(x)$ in this scenario. However, in practice we work with an empirical risk minimizer, from which we may still be able to recover a reasonable estimate for $\pi(x)$. An early approach to this problem is known as "Platt scaling": https://en.wikipedia.org/wiki/Platt scaling.

Using the solution from the first part,

$E_y[\ell(yf(x))|x]$

$= \pi(x)\ell(f(x)) + (1 - \pi(x)\ell(-f(x)))$

Substituting the hinge loss function, we get

$= \pi(x)max(0, 1 - f(x)) + (1 - \pi(x))max(0, 1 + f(x))$

We can see that the function is piece-wise linear and would be minimum at 2 non-differentiable points, f(x) = 1 and f(x) = -1, depending upon the value of $\pi(x)$

$$f^*(x) = \begin{cases} 1 & if \quad \pi(x) > 1 - \pi(x) \\ -1 & if \quad \pi(x) < 1 - \pi(x) \end{cases}$$

Therefore, $f^*(x) = sign(\pi(x) - \frac{1}{2})$

3 AdaBoost Actually Works

1. For any function g in $\{-1, 1\}$, show that $1(g(x) \neq y) < exp(-yg(x))$.

   We can consider both the possible cases here, if $g(x) = y$, then $yg(x) = 1$, this implies that lhs=0 and rhs=e^{-1},

   $\therefore$ lhs<rhs;

   On the other hand, If $g(x) \neq y$, then $yg(x) = -1$, this implies that lhs=1 and rhs=e,

   $\therefore$ lhs<rhs.

2. Use this to show $L(G, D) < Z_T$

   Using the result from the last part, we get

$$Z_T = \frac{1}{n}\sum_{i=1}^{n} exp(-y_i f_t(x_i)) > \frac{1}{n}\sum_{i=1}^{n} 1(f_t(x_i) \neq y_i))$$

   Taking into consideration the RHS of the expression above, we can write

$$1(f_t(x_i) \neq y_i)) \geq 1[sign(f_t(x_i)) \neq y_i](2)$$

   We can prove the above statement, by taking both the cases,

   if $sign(f_t(x_i)) \neq y_i$, then $f_t(x_i) \neq y_i$ and LHS = RHS,

   In the other case, when $sign(f_t(x_i)) = y_i$, then RHS=0, and LHS will be either 0 or 1, which give lhs$\geq$rhs.

   We substitute this result in the first inequality get,

$$Z_T = \frac{1}{n}\sum_{i=1}^{n} exp(-y_i f_t(x_i)) > \frac{1}{n}\sum_{i=1}^{n} 1[sign(f_t(x_i)) \neq y_i] = L(G, D)$$

3. Show that $w_i^{t+1} = exp(-y_i f_t(x_i))$

   The expression for w_i^{t+1}, can be written as,

   $w_i^{t+1} = w_i^t * exp(-\alpha_t y_i G_t(x_i))$

   Substituting the value of w_i^t,

   $= exp(\sum_{j=1}^{t} -\alpha_j y_i G_j(x_i))w_1$

   This can be written as,

   $= exp(-y_i f_t(x_i))$

Not attempted

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 should use the regression tree from the previous problem, if you completed it. Otherwise, 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 pseudo_residual_L2(train_target, train_predict):

    return train_target - train_predict

    # yfx = np.multiply(train_target, train_predict)
    # exp_yfx = np.exp(train_predict)
    # exp_yfx_rec = np.reciprocal(exp_yfx)

    # res = np.zeros(train_predict.shape[0])

    # for i in range(train_predict.shape[0]):

    #     res[i] = ((train_target[i] * (1.0 + exp_yfx_rec[i])) / (1.0 + exp_yfx_rec

    # return res


class gradient_boosting():

    def __init__(self, n_estimator, pseudo_residual_func, learning_rate=0.1, min_sa

        self.n_estimator = n_estimator
        self.pseudo_residual_func = pseudo_residual_func
        self.learning_rate = learning_rate
        self.min_sample = min_sample
        self.max_depth = max_depth
        self.estimators = list()

    def fit(self, train_data, train_target):

        preds = np.zeros(train_target.shape[0])
        for i in range(self.n_estimator):

            pseudo_res = pseudo_residual_L2(train_target.reshape(-1), preds.reshape

            new_est = DecisionTreeRegressor(max_depth = self.max_depth, min_samples
            new_est.fit(train_data, pseudo_res)

            self.estimators.append(new_est)

            preds += self.learning_rate * new_est.predict(train_data)


    def predict(self, test_data):

        prediction_final = np.zeros(test_data.shape[0])

        for i in range(self.n_estimator):

            prediction_final += self.learning_rate * self.estimators[i].predict(tes

        return prediction_final
```

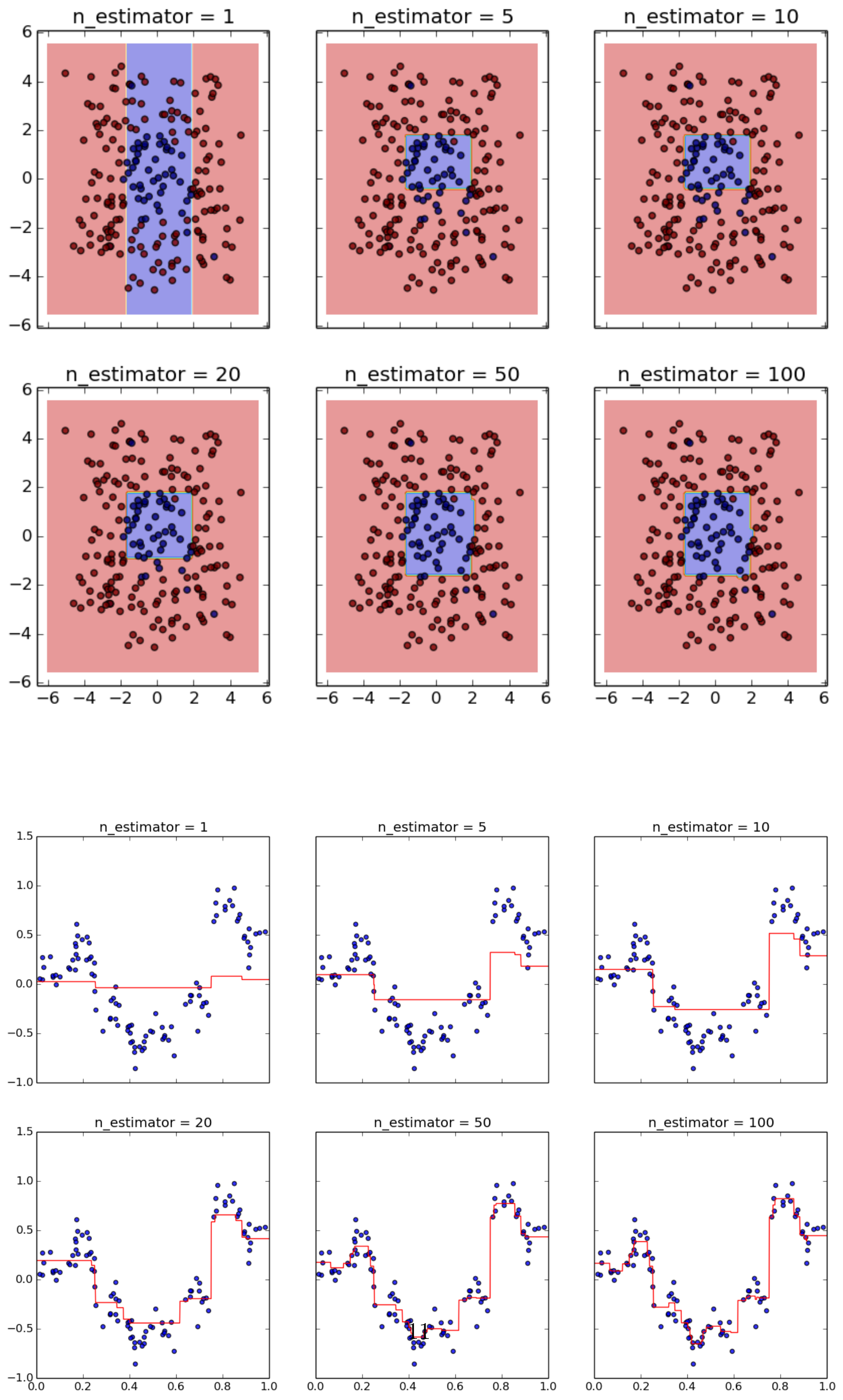

Figure 1: Gradient Boosting using square loss

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_L2(train_target, train_predict):

     # return train_target - train_predict

    yfx = np.multiply(train_target, train_predict)
    exp_yfx = np.exp(train_predict)
    exp_yfx_rec = np.reciprocal(exp_yfx)

    res = np.zeros(train_predict.shape[0])

    for i in range(train_predict.shape[0]):

        res[i] = ((train_target[i] * (1.0 + exp_yfx_rec[i])) / (1.0 + exp_yfx_rec[i

    return res


class gradient_boosting():

    def __init__(self, n_estimator, pseudo_residual_func, learning_rate=0.1, min_sa

        self.n_estimator = n_estimator
        self.pseudo_residual_func = pseudo_residual_func
        self.learning_rate = learning_rate
        self.min_sample = min_sample
        self.max_depth = max_depth
        self.estimators = list()

    def fit(self, train_data, train_target):

        preds = np.zeros(train_target.shape[0])

        for i in range(self.n_estimator):

            pseudo_res = pseudo_residual_L2(train_target.reshape(-1), preds.reshape

            new_est = DecisionTreeRegressor(max_depth = self.max_depth, min_samples
            new_est.fit(train_data, pseudo_res)

            self.estimators.append(new_est)
            preds += self.learning_rate * new_est.predict(train_data)

    def predict(self, test_data):

        prediction_final = np.zeros(test_data.shape[0])

        for i in range(self.n_estimator):

            prediction_final += self.learning_rate * self.estimators[i].predict(tes

        return prediction_final
```

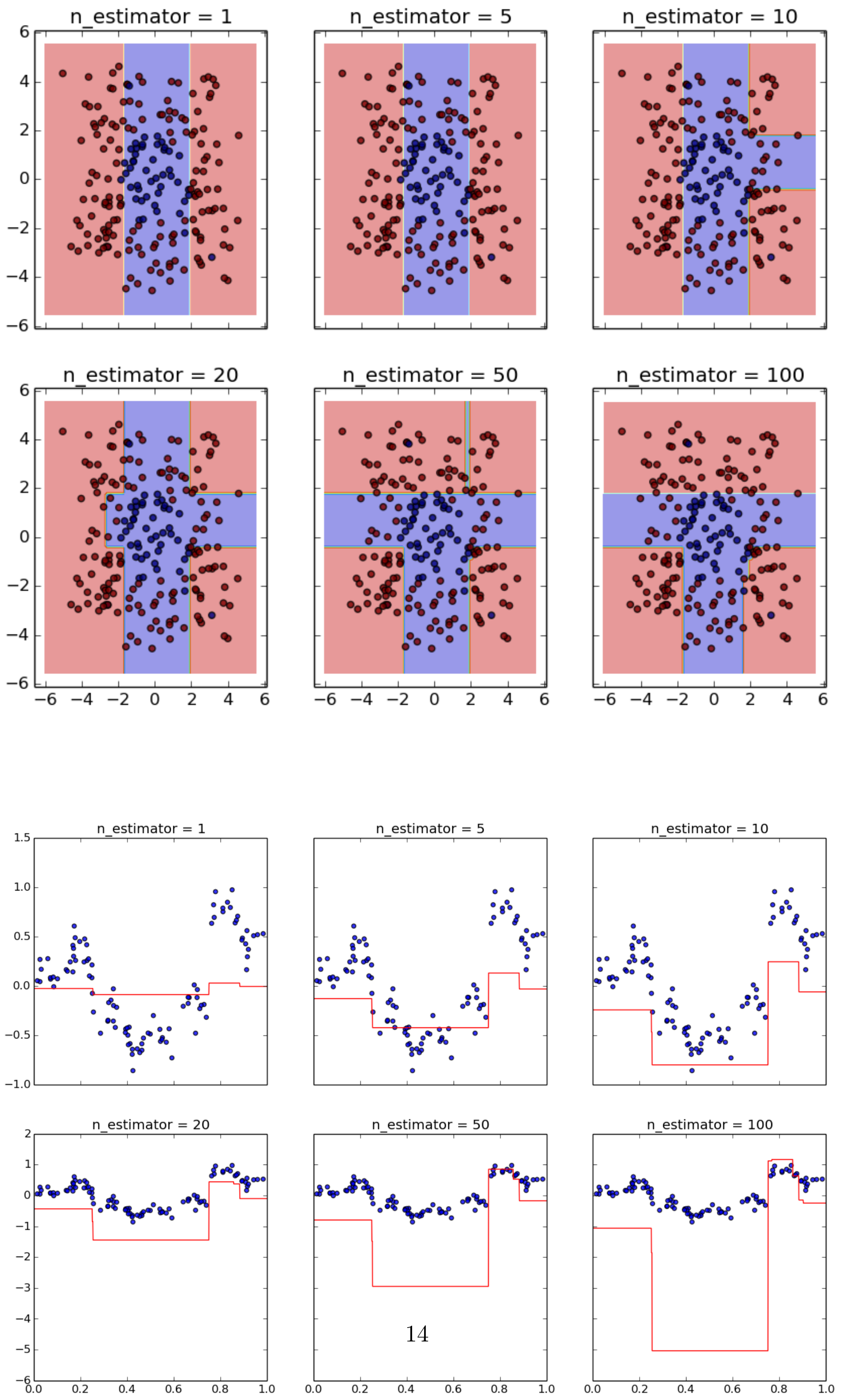

Figure 2: Gradient Boosting using logistic loss