

Homework 09

08 November 2019

DATA 1010

Problem 1

In this problem, we'll explore the reason **gradient boosting** has the w

Problem 1

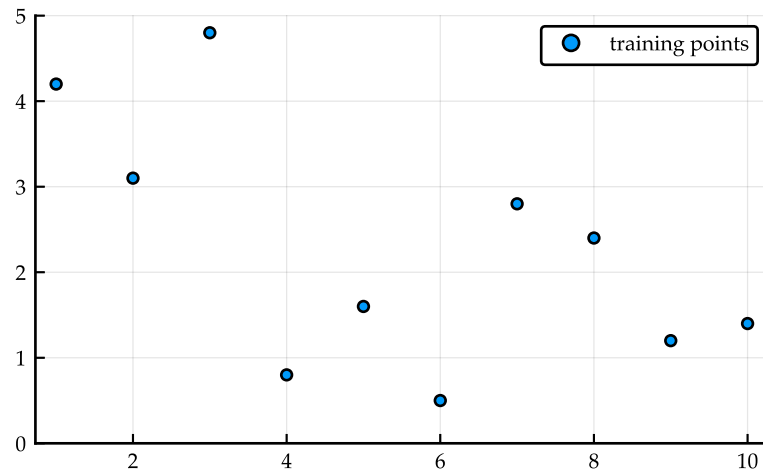
In this problem, we'll explore the reason *gradient boosting* has the word *gradient* in its name.

Step 1: Consider a very simple learning setup where $\mathcal{X} = \{1, 2, \dots, 10\}$ and $\mathcal{Y} = \mathbb{R}$. We'll co
Suppose that we happen to get 10 training observations, one for each value of \mathcal{X} . What is emp

```
[1] using Test, Plots, LaTeXStrings
```

```
[2] pyplot(fontfamily = "Palatino", size = (400, 250), fmt = :svg);
```

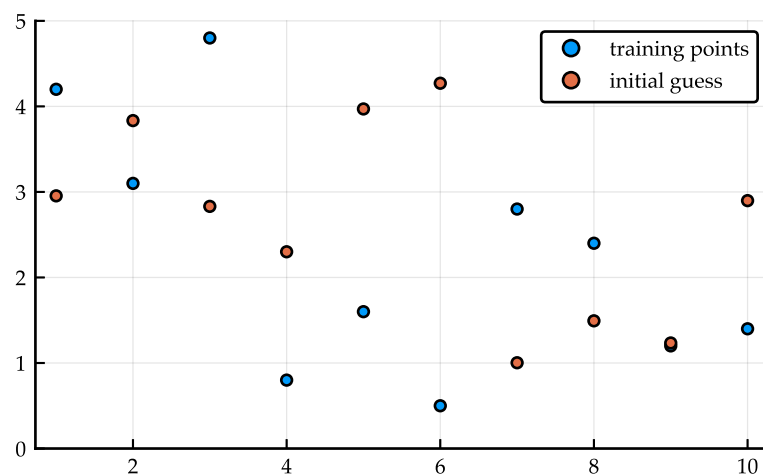
```
[3] x = 1:10  
y = [4.2, 3.1, 4.8, 0.8, 1.6, 0.5, 2.8, 2.4, 1.2, 1.4]  
scatter(x,y, ylims = (0,5), label = "training points")
```




Solution.

Step 2: Now, let's pretend we didn't notice how to cut straight to the empirical risk minimizer,

```
[4] using Random;
function training_plot()
    scatter(x, y, ylims = (0,5), label = "training points")
end
function random_start_plot!()
    Random.seed!(1234)
     $\hat{y}$  = 5rand(10)
    scatter!(x,  $\hat{y}$ , ylims = (0,5), label = "initial guess")
end
training_plot()
random_start_plot!()
```



Make a plot which looks like the one below, which shows convergence of the gradient descent

 Note: if we had used a learning rate of 1.0, we would actually skip directly to the optimizing

```
[0] function grad_descent_plot()
    Random.seed!(1234)
     $\hat{y}$  = 5rand(10)
    training_plot()
    random_start_plot!()
     $\hat{y}_{\text{prev}}$  =  $\hat{y}_{\text{new}}$  =  $\hat{y}$ 
    for i in 1:5
        # TODO: perform grad descent update with learning rate 0.5
        for j in 1:10
            # TODO: draw arrows
            #plot!([(,), (,)], arrow = arrow(), color = :rebeccapurple,
        end
    end
    current()
end
grad_descent_plot()
```

Step 3. There are several problems with using a unconstrained model (for example, it's prone to overfitting). Complete the definitions of the functions in the following cell and run the cell following that one.

```
[6] """
Return the vector obtained by replacing the first
k values with their average and the remaining values
with their average
"""
function decision_stump_fit(y, k)
end

@test decision_tree_fit([2, 2, 5, 5, 5], 3) ≈ [3, 3, 3, 5, 5]

"""
Return the best decision stump fit, by mean squared error
"""
function decision_stump_fit(y)
    mse( $\hat{y}$ ) = sum((y -  $\hat{y}$ ).^2)
    #k_min =
    decision_tree_fit(y, k_min)
end
```

```
@test decision_tree_fit([1, 2, 1, 5, 5]) ≈ [4/3, 4/3, 4/3, 5, 5]
```

Test Passed

In the plot below, we'll make double use of the horizontal axis, so that the arrows indicating it
Infer *from the graph* where each of the six decision trees made its split.

```
[0] function grad_descent_decision_tree_plot(num_iterations)
    training_plot()
    colors = Plots.Colors.distinguishable_colors(num_iterations, colora
                                                lchoices = 15:50, ccho

    ŷ_prev = ŷ_new = decision_tree_fit(y)
    scatter!(1:10, ŷ_new, label = "initial points")
    for i in 1:num_iterations
        ŷ_prev = ŷ_new
        ŷ_new += decision_tree_fit(y - ŷ_new)
        for j in 1:10
            plot!([(j + (i-1))/(3*num_iterations), ŷ_prev[j]),
                  (j + i/(3*num_iterations), ŷ_new[j])], arrow = arrow
                  color = colors[i], lw = 1, msw = 0, label = "")
        end
    end
    scatter!((1:10) .+ 1/3, ŷ_new, label = L"\mathbf{\hat{y}}", size
    current()
end
grad_descent_decision_tree_plot(5)
```

Solution.

Step 4. Steps 1 through 3 above establish the connection between gradient descent and gradi
The answer is actually that gradient boosting *does* involve a learning rate. It's just that there's
(a) Suppose that f is a differentiable function we're trying to minimize, and suppose that \mathbf{x}_k is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [H(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k).$$

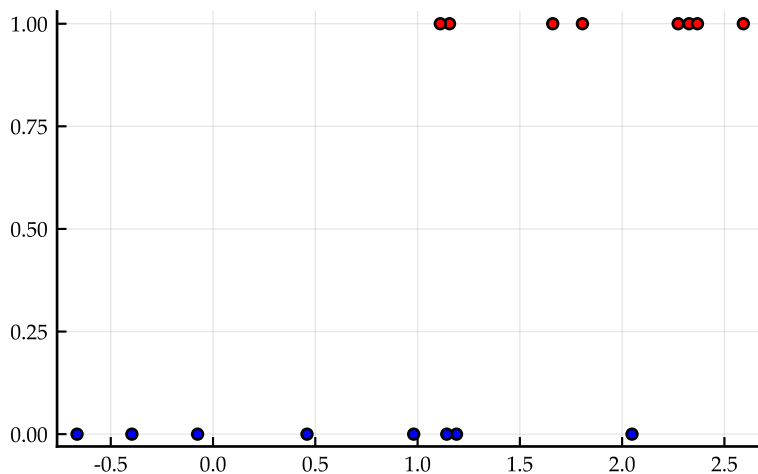
(b) Show further the Hessian H is the identity matrix if f is one-half times a mean squared err

Solution.

Problem 2

Armed with a gradient boosting perspective that generalizes the fit-the-residuals idea we began

```
[8] n = 8
Random.seed!(123)
blues = randn(n)
reds = 2 .+ randn(n)
xs = [blues; reds]
ys = repeat([0,1], inner = n)
function logistic_training_points()
    scatter([(p,0) for p in blues], color = :blue, legend = false)
    scatter!([(p,1) for p in reds], color = :red)
end
logistic_training_points()
```



Find the derivative of the logistic loss with respect to the vector rs of predictions at the training points. This is called the *pseudoresidual*, and it's the vector we'll

```
[9] # fill out the ternary conditional in the array comprehension:
#pseudoresidual(rs, ys) = [ (condition ? result_if_true : result_if_fal
```

```
[10] rs = fill(1/2, 2n); # start with an estimate of 1/2 for every point
```

Now you can execute this cell repeatedly to fit the pseudoresiduals with a decision stump and rs.

```
[0] # we're using a small learning rate:
rs .+= 0.05decision_stump_fit([y for (x,y) in
                             sort(collect(zip(xs, pseudoresidual(rs, ys))), by = first)]
logistic_training_points()
plot!(sort(xs), rs)
```

You should find that the model eventually misbehaves, because the probability estimates go p

Problem 3

Read the first three sections of the [Wikipedia article on Random Forests](#). Comment on the ass

Solution.

Problem 4

It is not useful to apply bagging or boosting to linear regression models. Why not?

Solution.

Problem 5

Suppose that weights and biases have been chosen for the neural network shown, and that a



- What vector is recorded at the second green node (the one between A_1 and A_2)?
- Now suppose that we are in the midst of the backpropagation process, and we have just

Solution.

[0]