

CS 221

Problem Set #2 Solutions: Machine Learning

Due by 9:30am on Tuesday, October 27. Please see the course information page on the class website for late homework submission instructions. SCPD students can also fax their solutions to (650) 725-1449. We will not accept solutions by email or courier.

NOTE: These questions require thought, but do not require long answers. Please try to be as concise as possible.

Written part (65 points)

1. [20 points] LEAST SQUARES

In lecture, we discussed a method for least squares linear regression using gradient descent applied to a cost function $J(\theta)$. It turns out it is actually possible to solve for the linear regression parameters θ analytically. In this problem, you will derive analytic solutions for some important special cases.

- (a) [2 points] First consider the simplest case, where we have no features, and we simply try to approximate the target variable y with a constant function $h_\theta = \theta$. Find the closed form solution for $\theta \in \mathbb{R}$ which minimizes the least-squares cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta)^2$$

Answer: Take the partial derivative with respect to θ :

$$\frac{\partial J}{\partial \theta} = \sum_{i=1}^m (y^{(i)} - \theta) = -m\theta + \sum_{i=1}^m y^{(i)}$$

Setting this equal to zero, we find that θ is the mean, e.g.

$$\theta = \frac{1}{m} \sum_{i=1}^m y^{(i)} = \mu_y$$

Almost everybody got this part perfect.

Grading criteria:

- (-1) Math mistake
- (-1) No / lack of explanation
- (-1) Conceptual mistake
- (-2) Missing or completely wrong

- (b) [7 points] Now consider the case of $n = 1$, so that there is only a single feature and each $x^{(i)} \in \mathbb{R}$ is a real number. (For example, trying to predict housing prices from

a single feature, the size of the house.) Find the closed form solutions for θ_0 and θ_1 which minimize the least-squares cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta_0 - \theta_1 x^{(i)})^2.$$

Express your answer in terms of the empirical means μ_x and μ_y and the empirical moments S_{xx} , S_{xy} , S_{yy} , defined as:

$$\begin{aligned} \mu_x &= \frac{1}{m} \sum_{i=1}^m x^{(i)} \\ \mu_y &= \frac{1}{m} \sum_{i=1}^m y^{(i)} \\ S_{xx} &= \frac{1}{m} \sum_{i=1}^m (x^{(i)})^2 \\ S_{xy} &= \frac{1}{m} \sum_{i=1}^m x^{(i)} y^{(i)} \\ S_{yy} &= \frac{1}{m} \sum_{i=1}^m (y^{(i)})^2 \end{aligned}$$

Hint: Take the partial derivatives of J with respect to θ_0 and θ_1 , and set them to zero.

Answer: Computing the partial derivatives, we get

$$0 = \frac{\partial J}{\partial \theta_0} = -2 \sum_{i=1}^m (y^{(i)} - \theta_0 - \theta_1 x^{(i)}) \quad (1)$$

$$= -2m(\mu_y - \theta_0 - \theta_1 \mu_x) \quad (2)$$

$$0 = \frac{\partial J}{\partial \theta_1} = -2 \sum_{i=1}^m (y^{(i)} - \theta_0 - \theta_1 x^{(i)}) x^{(i)} \quad (3)$$

$$= -2m(S_{xy} - \theta_0 \mu_x - \theta_1 S_{xx}) \quad (4)$$

$$(5)$$

Rewriting these two equations and cancelling m ,

$$\theta_0 = \mu_y - \theta_1 \mu_x$$

$$\theta_1 = \frac{S_{xy} - \theta_0 \mu_x}{S_{xx}}$$

Substituting in θ_0 and solving for θ_1 ,

$$\begin{aligned} \theta_1 &= \frac{S_{xy} - (\mu_y - \theta_1 \mu_x) \mu_x}{S_{xx}} \\ &= \frac{S_{xy} - \mu_x \mu_y + \theta_1 \mu_x^2}{S_{xx}} \end{aligned}$$

hence

$$(S_{xx} - \mu_x^2)\theta_1 = S_{xy} - \mu_x$$

giving

$$\theta_1 = \frac{S_{xy} - \mu_x}{S_{xx} - \mu_x^2}$$

Substituting this back into (2), we get:

$$\begin{aligned}\theta_0 &= \mu_y - \theta_1 \mu_x \\ &= \mu_y - \frac{S_{xy} - \mu_x \mu_y}{S_{xx} - \mu_x^2} \mu_x \\ &= \frac{S_{xx} \mu_y - S_{xy} \mu_x}{S_{xx} - \mu_x^2}\end{aligned}$$

Technically we need to verify that what we found is a minimum and not, say, a maximum. Since it was mentioned in lecture that the least-squares cost function is convex, we didn't take off for this point.

In general, most people got this part correct, minus perhaps a minor math error somewhere. We assigned 4 points for deriving each linear equations for θ_0 and θ_1 , and 3 points for solving this system of linear equations. We accepted any correct final answer as long as at least one of θ_0 or θ_1 was given in closed form.

Grating criteria:

- (-1) Math mistake
- 3 points for first linear equation, as specified in B2.1-B2.4
- (-1) Math mistake
- (-2) No / lack of explanation
- (-2) Conceptual mistake
- (-3) Missing or completely wrong
- 4 points for first linear equation, as specified in B3.1-B3.4
- (-1) Math mistake
- (-2) No / lack of explanation
- (-2) Conceptual mistake
- (-3) Missing or completely wrong
- (-2) No working shown for solving linear equations
- (-3) Did not solve for closed form solution of θ_0 or θ_1

(c) [**11 points**] We've seen in class that attempting to fit functions with many parameters (e.g., a high-order polynomial) using too little data can result in a overfitting. One solution to this problem is to use a simpler hypothesis with fewer parameters. In this problem, we will look a different solution to the problem of overfitting, called *regularization*.

- i. [**3 points**] Suppose we are fitting a hypothesis $h_\theta(x)$ with parameters $\theta \in \mathbb{R}^{n+1}$ to a training set $\{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$ as usual. Consider the cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2} \|\theta\|_2^2$$

where $\|\theta\|_2$ is defined as

$$\|\theta\|_2 = \sqrt{\sum_{j=0}^n \theta_j^2}$$

Note that this is the same cost function used in class, but with the addition of the “regularization” term $\frac{\lambda}{2}\|\theta\|_2^2$ at the end. This term serves to keep the norm of the parameters θ small, and minimizing $J(\theta)$ now involves making a tradeoff between fitting the data well (minimizing the sum of squared errors term), and keeping the parameter values small (minimizing the $\frac{\lambda}{2}\|\theta\|_2^2$ term). The value λ , which we assume is given, controls the relative importance of the two terms in the objective. This cost function is called the “ L_2 regularized” cost (because the regularization term involves the square of the L_2 norm of the vector θ).

Suppose we want to fit a linear function to our data, so that $h_\theta(x^{(i)}) = \theta^\top x^{(i)}$ as in class. Suppose we use batch gradient descent to minimize our regularized cost function $J(\theta)$, defined above. Show that the batch gradient descent learning rule for this cost function is

$$\theta_j := \theta_j - \alpha \left(\sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)}) x_j^{(i)} + \lambda \theta_j \right)$$

where α is the learning rate.

Answer: The gradient descent learning rule was defined in lecture as

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

where α is the learning rate.

Computing $\frac{\partial}{\partial \theta_j} J(\theta)$, we get:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \left(\frac{1}{2} \sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2} \sum_{k=1}^n \theta_k^2 \right) \\ &= \sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)}) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)} - y^{(i)}) + \frac{\partial}{\partial \theta_j} \left(\frac{\lambda}{2} \theta_j^2 \right) \\ &= \sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)}) x_j^{(i)} + \lambda \theta_j \end{aligned}$$

Therefore the gradient descent learning rule is

$$\theta_j := \theta_j - \alpha \left(\sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)}) x_j^{(i)} + \lambda \theta_j \right)$$

Grading criteria:

- (-1) Math mistake
- (-2) No / lack of explanation
- (-2) Conceptual mistake
- (-3) Missing or completely wrong

ii. [3 points] Suppose that instead we use the cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2} \|\theta\|_1$$

where

$$\|\theta\|_1 = \sum_{j=0}^n |\theta_j|$$

Here, the regularization function uses the sum of the absolute values of θ_j . This is called the “ L_1 regularized” cost and is frequently used because it encourages the vector θ to be “sparse” (i.e., θ will have many entries that are exactly 0). Derive the batch gradient descent learning rule for this cost function. You may use $\frac{\partial}{\partial \theta_j} |\theta_j| = 0$ when $\theta_j = 0$.

Answer: The gradient descent learning rule was defined in lecture as

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

where α is the learning rate.

Computing $\frac{\partial}{\partial \theta_j} J(\theta)$, we get:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \left(\frac{1}{2} \sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2} \sum_{k=1}^n |\theta_k| \right) \\ &= \sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)}) \frac{\partial}{\partial \theta_j} (\theta^\top x^{(i)} - y^{(i)}) + \frac{\partial}{\partial \theta_j} \left(\frac{\lambda}{2} |\theta_j| \right) \end{aligned}$$

Consider $\frac{\partial}{\partial \theta_j} |\theta_j|$ for the three possible cases of the value of θ_j :

$$\frac{\partial}{\partial \theta_j} |\theta_j| = \begin{cases} -1 & \text{if } \theta_j < 0, \\ 0 & \text{if } \theta_j = 0 \text{ (given)}, \\ 1 & \text{if } \theta_j > 0. \end{cases}$$

In other words,

$$\frac{\partial}{\partial \theta_j} |\theta_j| = \text{sgn}(\theta_j)$$

where sgn is the sign function, and has value 0 when its input is 0.

Therefore the gradient descent learning rule is

$$\theta_j := \theta_j - \alpha \left(\sum_{i=1}^m (\theta^\top x^{(i)} - y^{(i)}) x_j^{(i)} + \frac{\lambda}{2} \text{sgn}(\theta_j) \right)$$

Grading criteria:

- (-1) Math mistake
- (-2) No / lack of explanation
- (-2) Conceptual mistake
- (-3) Missing or completely wrong

An error code of E1 (-2 points) was also frequently given out (as a one-time deduction for both parts) for giving a gradient ascent rule instead of descent. Remember that we are trying to minimize the cost function $J(\theta)$, so we apply gradient descent:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

instead of gradient ascent, where the sign of the second term would be positive. This confusion may have been caused by the rule shown in class, where the second term was positive; however, if one follows through the derivation, one will find that the negation was simply pushed in, and that we got

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

which is equivalent to the first term of the answer above:

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- iii. **[5 points]** *Important: This is considered part of the “written” assignment, not the programming assignment, and thus you must submit your individual solution. You must implement the solution from scratch and may not look at anyone else’s code.*

In Matlab, implement batch gradient descent for L_2 regularized linear regression with a 5th order polynomial. Thus, the form of your hypothesis should be

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

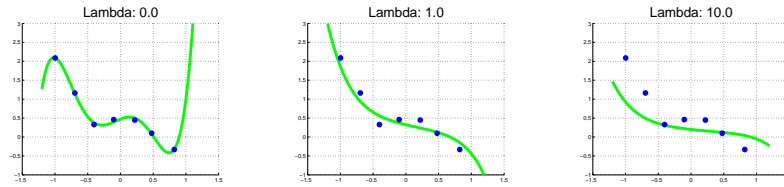
We provide two datasets, `points1.mat` and `points2.mat`, both available next to the problem set handout at

<http://cs221.stanford.edu/handouts.html>

- Run `load points1` to load in the data. You will then have two variables, `x` and `y`, in your Matlab workspace.
- Since you are fitting a 5th order polynomial to your data, you will need to compute 6 features of `x`.¹
- Initialize `theta` to all zeros.
- Use $\alpha = 0.1$ as your learning rate.
- You can use the `norm` function to compute the L_2 norm of `theta`.
- For the convergence criteria of gradient descent, stop when the `norm(theta)` changes by less than $1e - 8$ between two subsequent iterations.
- Run your algorithm with three different values for `lambda`: 0, 1, and 10.
- Compare your solution to ours. For the numerical results, we obtained 8.1690, 0.7910, and 0.3867 as values of `norm(theta)` after running gradient descent

¹In class, we talked briefly about a preprocessing step of scaling each feature to take on about the same range of values, and said that this can speed up gradient descent. For this assignment, *don't* do any such preprocessing step, and just use the appropriate powers of x (i.e., 1, x , x^2 , etc.) as your features.

with each of the three values of `lambda` respectively. Here are the plots of the polynomial functions:



- Once you are convinced your solution is correct, load `points2` and run your code on the new data.

For this problem, you need to turn in a printout of your code, and your results on `points2`. Specifically, for *each of the three lambdas*, please include: (1) the values of `theta` you obtain after convergence, (2) the value of `norm(theta)` after convergence, and (3) a plot of the results, similar to the ones above.

Answer:

Here is the code we used to generate the solution (obviously yours might be very different, especially the plotting):

```

close all;
% Get x,y data
load points2.mat;
% Run gradient descent with multiple lambdas and plot the results
xx = -1.2:0.01:1.2;
XX = bsxfun(@power, xx', 5:-1:0); % computing powers of xx
lambds = [0 1 10];
for i=1:length(lambds)
    figure;
    hold on, grid on;
    axis([-1.5, 1.5, -1, 3]);
    th = gradDescent(x,y,lambds(i))
    norm(th)
    plot(xx,XX*th,'g','LineWidth',5);
    plot(x,y,'b.','markersize',30);
    title(sprintf('Lambda: %0.1f',lambds(i)), 'fontsize', 24);
end

function th = gradDescent(x,y,lambda)
maxPower = 5;
alpha = 0.1;
th = zeros(maxPower + 1,1);
normChange = inf;
% Simple (unoptimized) version
while normChange > 1e-8
    % include regularization
    thnew = th - alpha * lambda * th;
    for i = 1:length(x)
        % create a feature vector of [x^5; x^4; ...; 1]

```

```

X = [];
for j = 0:maxPower;
    X = [x(i)^j; X];
end
% include the contribution of each example
thnew = thnew - alpha * ((X' * th - y(i)) * X);
end
normChange = norm(th - thnew);
th = thnew;
end
% Alternative (optimized) version of gradDescent
% X = bsxfun(@power, x', maxPower:-1:0);
% while normChange > 1e-8
%     thnew = th - alpha*(sum(bsxfun(@times, (X*th - y'), X),1) + ...
%         lambda*th);
%     normChange = norm(th - thnew);
%     th = thnew;
% end

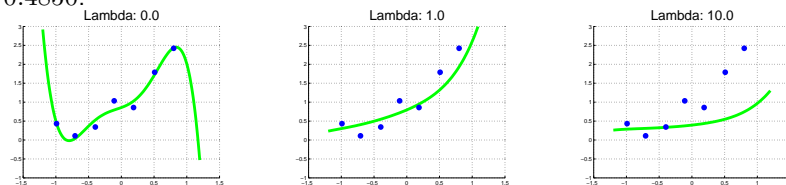
```

Numerical solution:

For Lambda = 0, theta = -3.5513, -0.3911, 3.6740, 0.7983, 0.6366, 0.8582, and norm(theta) = 5.2955.

For Lambda = 1, theta = 0.1084, 0.2762, 0.3034, 0.3786, 0.7380, 0.7967, and norm(theta) = 1.2258.

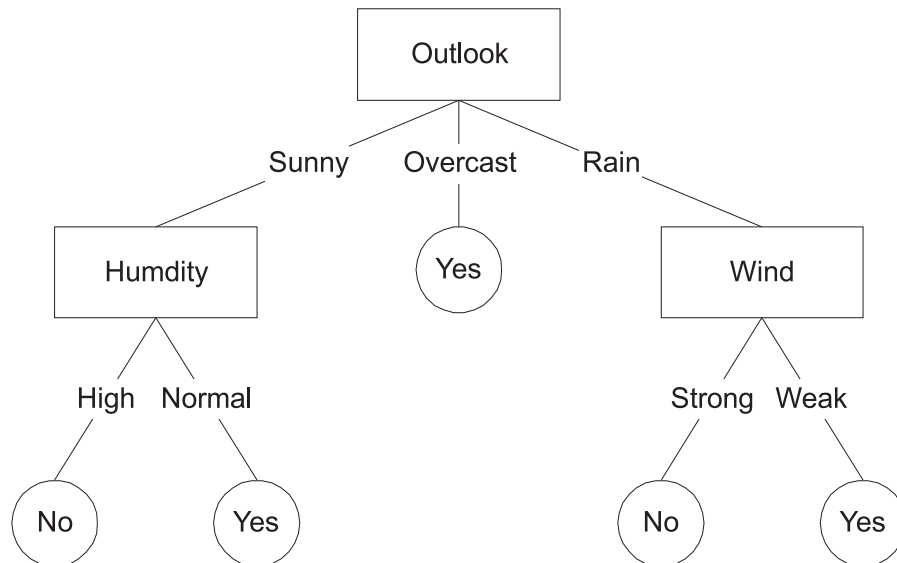
For Lambda = 10, theta = 0.0488, 0.0862, 0.0952, 0.1463, 0.1970, 0.3951, and norm(theta) = 0.4850.



2. [12 points] DECISION TREES

In this question we assume that there is some decision tree generating the data. This question investigates some properties of the decision tree learning algorithm.

Suppose Beatrice uses the following decision tree with features $\{Outlook, Temperature, Humidity, Wind\}$ to decide whether she will practice her archery:



We wish to reconstruct her original decision tree by observing when she does and when she does not go to practice her archery. Suppose we gather the following observations over the course of two weeks:

Day	<i>Outlook</i>	<i>Temperature</i>	<i>Humidity</i>	<i>Wind</i>	<i>Practice Archery</i>
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Notice that ID3 (the decision tree learning algorithm presented in class that uses information gain to split on attributes) actually learns the original tree perfectly when presented with these 14 training examples. In the following questions we will require that all of the training data are consistent with Beatrice's original tree, meaning the PracticeArchery label on each of the training data should be what Beatrice herself would do if she were to observe those settings of the attributes. (Recall that we assumed that Beatrice is using the tree shown above.) Duplicate training examples are allowed; they count as separate individual examples in the information gain computations.

- (a) [5 points] Is it possible for a training set to get ID3 to learn a tree that is identical to Beatrice's tree except that the learned tree further elaborates the tree below the rightmost leaf? Justify your answer.

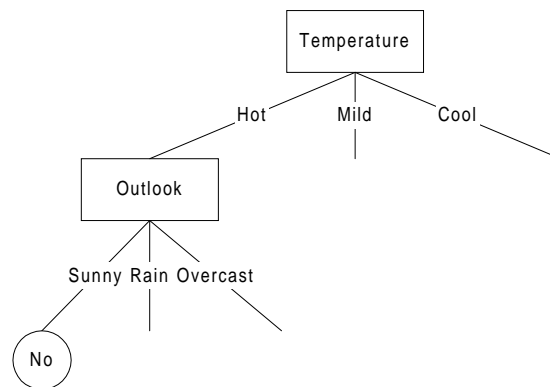
Answer: No it is not possible. All data cases that have *Outlook*=*Rain* and *Wind* = *Weak* will have to be labeled *PracticeArchery* = *Yes* since they must be consistent with Beatrice's original tree. Thus, all data cases at this lower right leaf will have the same label. This is one of the terminating conditions for ID3 and so it will not split this branch any further.

Error Codes:

- a0 (-5) Wrong / skipped
- a1 (-3) Mostly wrong
- a2 (-1) Almost correct

- (b) [7 points] Can there be a decision tree T that is consistent with the Beatrice's original tree on examples $D1$ — $D14$ (i.e., tree T gives the correct *Practice Archery* classification on these examples), and that classifies *Practice Archery* differently from that original tree for at least one other example? Justify why this is not possible, or give such a tree as a counter example.

Answer: Yes it is possible. Consider a tree splits on the *Temperature* attribute first, and then splits on the *Outlook* attribute on the *Temperature* = *Hot* branch. (The rest of the tree can be completed by splitting on all other attributes, and agreeing everywhere with Beatrice's tree.)



Now, note that this tree classifies examples $D1$ through $D14$ like Beatrice's tree. But an example such as *Outlook*=*Sunny*, *Temperature*=*Hot*, *Humidity*=*Normal*, *Wind*=*Strong* will be classified as *No* even though Beatrice's tree will classify it as *Yes*.

Error Codes:

- b0 (-7) Wrong / skipped
- b1 (-2) Right idea, but given tree is not consistent with Beatrice's tree on examples $D1$ - $D14$.
- b2 (-5) Example doesn't work: identical output to Beatrice's tree for all inputs.

- b3 (-1) Forgot to include sample input on which the trees differ
- b4 (-3) Incomplete example, did not complete tree

3. [17 points] BOOSTING

In this question, you will develop some intuition as to why boosting can combine a few classifiers with low accuracy to get a final classifier with improved accuracy. Assume that our learning algorithm is a “weak learner” that is guaranteed to get an accuracy of *exactly* 70% on any input distribution. In other words, if we have a distribution p on instances x , and we give our algorithm a training set of m instances sampled randomly from p , then the classifier h learned by our algorithm will have a 0.7 probability of correctly labeling a new random (test) example sampled from p . I.e., the generalization error of h will be $1 - 0.7 = 0.3$. More formally, in this problem we assume that there is a deterministic relation between x and y , given by $y = c(x)$. Let $Correct = \{x : h(x) = c(x)\}$. Then, the probability mass of $Correct$ according to p — $p(Correct)$ — is 0.7. We assume that this property holds for *any* distribution p .

The first step in our very simple boosting algorithm² is to apply our learning algorithm to the original target distribution; i.e., we get m examples from the the distribution, and learn a hypothesis h_1 .

- (a) [3 points] When we choose a random example from the distribution p , h_1 has a 70% chance of getting it right. Our first goal is to define a new distribution p_2 such that, if we sample a random example from p_2 , then h_1 will have a 50% chance of being correct. How should we define $p_2(x)$ so that it will have this property? You may use the results of running h_1 on each example in p to determine the weight of the example in p_2 . [Hint: p_2 will give a higher weight/probability than p to examples that h_1 classified incorrectly, and a lower weight than p to examples that h_1 classified correctly.]

Answer:

$$\begin{aligned} \text{For } x \in \text{Correct}, p_2(x) \text{ should be } \frac{5}{7}p(x). \\ \text{For } x \notin \text{Correct}, p_2(x) \text{ should be } \frac{5}{3}p(x). \end{aligned}$$

Grading criteria:

- Partial credit was given for various expressions that attempted to achieve the correct result.
- Full points for correct weights. Some people forgot to normalize their distributions; we did not penalize for this.

The next step in boosting is to use this new distribution p_2 to learn a new hypothesis h_2 . For a random example x chosen from p_2 , h_1 is essentially random, in that it classifies x correctly only 50% of the time. However, by our weak learning assumption, h_2 will classify x correctly 70% of the time.

The problem, of course, is that given only the two hypotheses that give different answers, there is no way of telling which is right. To fix this, we construct a third hypothesis, using a third training distribution p_3 , which is identical to p except that we throw out instances where h_1 and h_2 agree. (In other words: p_3 is p , conditioned on the event that h_1 and h_2 disagree.) Using our assumption, whenever h_1 and h_2 disagree, we have a hypothesis h_3 that has a 70% chance of being correct.

²This algorithm is actually the very first boosting algorithm developed.

We can now combine the hypotheses as follows: If h_1 and h_2 agree, use the answer they both give; otherwise, use h_3 . This works out to a majority vote. Let h^* be the combined hypothesis. Our goal now is to analyze h^* and get a lower bound on its accuracy.

(b) [2 points] Divide the domain into four sets:

- R_1 x such that both h_1 and h_2 are correct
- R_2 x such that h_1 is correct but h_2 is incorrect
- R_3 x such that h_2 is correct but h_1 is not
- R_4 x such that both are incorrect

For each of these sets R_i , find $q_i = P(h^*(x) = c(x) \mid x \in R_i)$, i.e., the probability that a random example chosen from R_i is correct.

Answer:

R_1 : Two classifiers agree on the correct answer, so $q_1 = 1.0$.

R_2 : If the third classifier gives the correct answer, two networks will agree. This occurs 0.7 of the time, so $q_2 = 0.7$.

R_3 : Same as R_2 , $q_3 = 0.7$.

R_4 : We already have two classifiers agreeing on the wrong answer, so $q_4 = 0$.

Grading criteria:

b0 -1 point for each incorrect q_i value.

- Full points for all correct probabilities.

(c) [3 points] Let $p(R_i)$ be the probability mass of R_i according to p , i.e., the chance that an example chosen randomly according to p is in the set R_i . Similarly, define $p_2(R_i)$ to be the probability mass of R_i according to p_2 . Find $p(h^*(x) = c(x))$ for distribution p in terms of these expressions and the quantities q_i from part (b). Simplify this expression by substituting your values for q_i . You may not need to use the value $p_2(R_i)$ in your expression.

Answer: The accuracy of h^* relative to p can be defined as:

$$p(h^*(x) = c(x)) = q_1 \cdot p(R_1) + q_2 \cdot p(R_2) + q_3 \cdot p(R_3) + q_4 \cdot p(R_4).$$

This simplifies to

$$p(h^*(x) = c(x)) = p(R_1) + 0.7p(R_2) + 0.7p(R_3).$$

Grading criteria:

- -0.5 point for minor mistake.
- Full points for correct computation. We did not penalize if the only error was due to mistake in previous part.

(d) [3 points] The key step remaining in analyzing the algorithm is to compute the probability masses of the different sets R_i . While we can't compute the exact masses, we do have some information; for example, we know that

$$p(R_1) + p(R_2) = 0.7 \tag{6}$$

as the chance that an example is classified correctly by h_1 is 0.7, so the chance that it is in one of the first two sets is 0.7. Come up with a similar expression for p_2 involving two subsets, and substitute your definitions of p_2 from part (a) to get another expression involving only p .

Answer: The chance that h_2 classifies something correctly is 0.7, so

$$p_2(R_1) + p_2(R_3) = 0.7 \quad (7)$$

Using the definition of p_2 we gave in part a, we get

$$\frac{5}{7}p(R_1) + \frac{5}{3}p(R_3) = 0.7. \quad (8)$$

Common mistakes: Some people had $p_2(R_1) + p_2(R_3) = 0.5$, apparently thinking the total was 0.5 because p_2 was constructed so that h got 0.5 of it right. However, the total is 0.7 because $R_1 \cup R_3$ is the total fraction of correct labels from h_2 under p_2 , and that is assumed to be 0.7 by the assumption about our learning algorithm.

Also: many people computed individual masses for $p(R_1)$, etc. This requires some assumptions that are not necessarily true. Still, we did not penalize since the problem didn't specify the details of the weak learning algorithms and training procedure (which may or may not have justified such assumptions.)

Grading criteria:

d1 Partial credit for messy / vague statements.

- 1.5 points for only specifying constraint in terms of p_2 .
- Full points for constraint + simplifying in terms of p .

(e) [6 points] Using Eq. (6) and the equations you obtained in parts 3d and 3c, find an exact number for $p(h^*(x) = c(x))$.

(Hint: Start by splitting the $p(R_1)$ term in your answer from 3c into: $0.3 \cdot p(R_1) + 0.7 \cdot p(R_1)$, then regrouping terms.)

Answer: We now have:

$$\begin{aligned} p(h^*(x) = c(x)) &= p(R_1) + 0.7 \cdot p(R_2) + 0.7 \cdot p(R_3) \\ &= (0.3 \cdot p(R_1) + 0.7 \cdot p(R_1)) + 0.7 \cdot p(R_2) + 0.7 \cdot p(R_3) \end{aligned}$$

Regrouping:

$$p(h^*(x) = c(x)) = 0.3 \cdot p(R_1) + 0.7 \cdot p(R_3) + 0.7 \cdot (p(R_1) + p(R_2)) \quad (9)$$

Equation (6) says:

$$0.7 \cdot (p(R_1) + p(R_2)) = 0.7 \cdot 0.7 = 0.49.$$

Equation (8) says:

$$\begin{aligned} 0.3 \cdot p(R_1) + 0.7 \cdot p(R_3) &= \frac{21}{50}(5/7 \cdot p(R_1) + 5/3 \cdot p(R_3)) \\ &= \frac{21}{50} \cdot 7/10 = 0.294. \end{aligned}$$

Now (9) becomes $p(h^*(x) = c(x)) = 0.294 + 0.49 = 0.784$, the number we're looking for.

Grading criteria:

- e1** 4 points for a correct manipulation thrown awry by earlier mistakes.
e2 4 points for apparently correct answer that is vague / unclearly justified.
- Full points for complete proof that includes the basic steps above.

4. [16 points] SHORT ANSWER

The following questions require a yes/no answer accompanied by one sentence of explanation.

No credit will be given for an incorrect yes/no, or for answers without a correct explanation.

(a) [6 points] Suppose that we use least squares linear regression to fit a hypothesis to a training set. We then test the learned function against another (different) set of data and discover that the test set error is much larger than the training error. We would like to reduce our test set error.

i. [2 points] It is likely that training on more data will help significantly? (I.e., does collecting more training data seem a promising thing to try to improve the algorithm's performance?)

Answer: Yes. Since testing error is much larger than training error, we have a "high variance" (or "over-fitting") problem (which we know from class can be helped by having more data.)

ii. [2 points] Is it possible that our test error could improve by *removing* some features from our data?

Answer: Yes. Having too many features is one cause of over-fitting; thus, removing some features may improve our generalization error.

iii. [2 points] Suppose that we are using regularized least-squares, as in problem 1c. Would decreasing λ be expected to improve our test set error?

Answer: No. Reducing the penalty on the parameters will allow us to fit our training data even more precisely than before, which will make over-fitting more severe.

(b) [4 points] Suppose that we fit a hypothesis using the least squares algorithm described in class (without any regularization) to a training set, and we observe a large training error. We then test the learned function against another (different) set of data and discover that the test set error is roughly the same as the training error. We would like to reduce our test set error.

i. [2 points] It is likely that training on more data will help significantly? (I.e., does collecting more training data seem a promising thing to try to improve the algorithm's performance?)

Answer: No. The high training error (and similar test error) means we have high bias (as opposed to high variance), which cannot be fixed by training on more data.

ii. [2 points] Suppose that we add some new features to our training data. Is it possible for the training error to get worse?

Answer: No. Our learning algorithm may simply choose to ignore the additional feature when it doesn't help; thus, having more features can only improve our training error.

(c) [3 points] Consider training a logistic regression classifier to classify input vectors $x \in \mathbb{R}^{n+1}$, where each input vector x contains n features $\{x_1, x_2, \dots, x_n\}$ along with the constant term $x_0 = 1$.

Suppose we construct a new feature $x_{n+1} = x_1 + x_2$, so that now there are $n + 1$ features $\{x_1, x_2, \dots, x_n, x_{n+1}\}$ along with the constant term $x_0 = 1$. Is it possible for the classifier trained with these new features to achieve strictly better training error than the classifier trained with the original features?

Answer: No. The logistic regression classifier is just learning a linear function of the features, and hence already considers linear combinations of x_1 and x_2 .

- (d) **[3 points]** Suppose we have a least squares regression problem with n features, and where one of the features x_j takes only two values: -1 and $+1$. Further, whenever $x_j = -1$ in the training set, the output y is positive; and whenever $x_j = +1$ in the training set, the output y is negative.

Is it possible for the least squares coefficient θ_j corresponding to the feature x_j to be positive?

(As in class, assume θ is chosen by minimizing $\sum_{i=1}^m (y^{(i)} - \sum_j \theta_j x_j^{(i)})^2$.)

Answer: Yes. Take a 2 feature example, with $j = 2$ and $y = x_1 + x_2$. Also, say $|x_1| \geq 50$. (For example, consider a training set of $(x_1^{(i)}, x_2^{(i)}, y^{(i)})$ tuples as the following: $S_{train} = \{(100, -1, 99), (-50, 1, -49), (50, -1, 49)\}$.) Then, the conditions in the problem are satisfied, and the optimum will be at $\theta_1 = \theta_2 = 1$.