CS 475 Machine Learning: Homework 3
Non-linear Methods

Due: Wednesday October 12, 2016, 11:59pm

100 Points Total Version 1.0

JinYong Shin
Jshin44

1 Analytical (50 points)

1) Decision Trees (12 points) Consider a binary classification task with the following set of training examples:

x_1	x_2	x_3	x_4	x_5	classification
0	1	1	-1	-1	+
0	1	1	-1	-1	+
0	1	1	1	1	-
0	-1	1	1	1	+
0	-1	1	-1	-1	-
0	-1	1	-1	-1	-

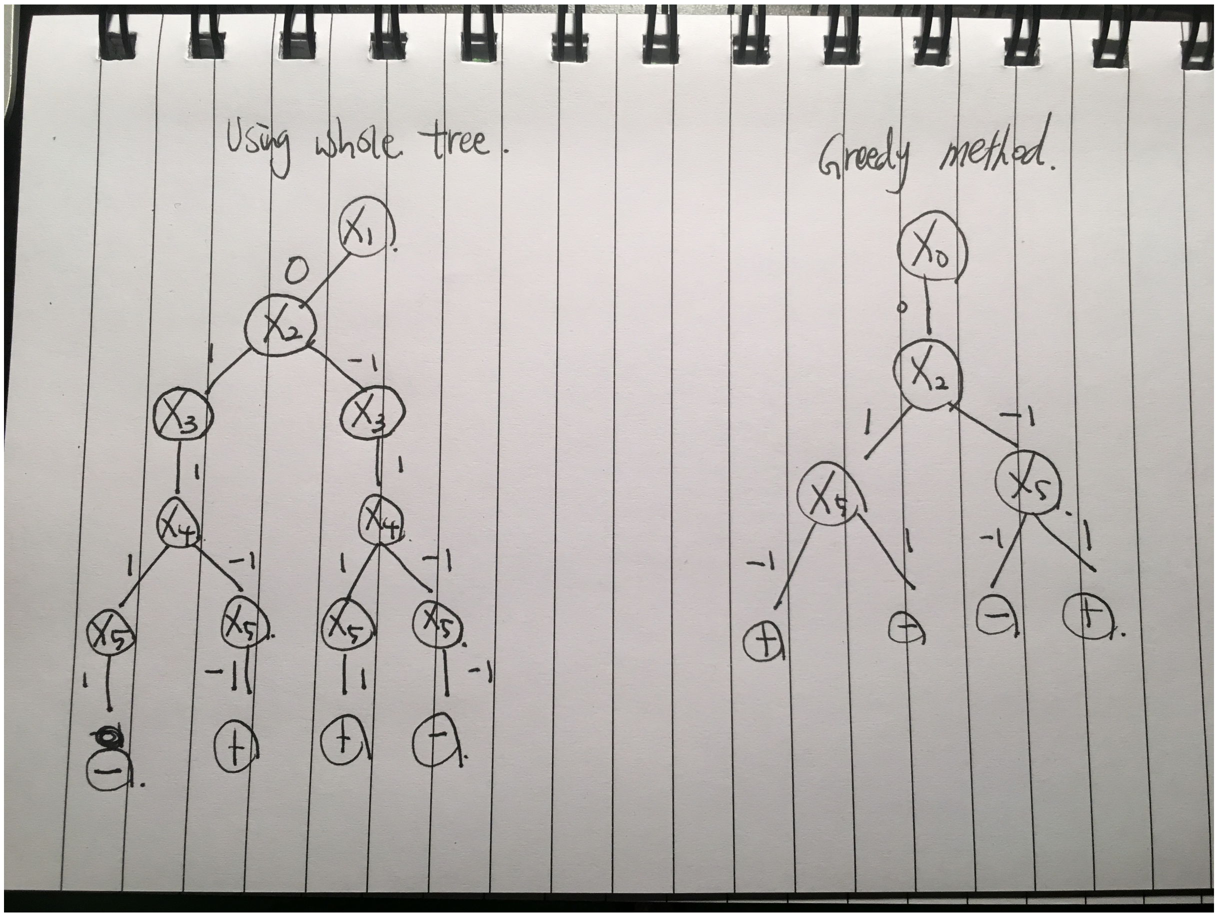
(a) Can this function be learned using a decision tree? If so, provide such a tree (describe each node in the tree). If not, prove it.

(b) Can this function be learned using a logistic regression classifier? If yes, give some example parameter weights. If not, why not.

(c) What is the entropy of the labels in the training data?

(d) What is the information gain of x_2 relative to labels?

Answer:

(a) 

We can use both method. Just follow the root as left method. But as we learned from the lecture, we could also use greedy method to follow short node.

(b) The decision tree that we have described in this problem is classification tree that the response only be done into binary split. The variable that the each node return is in the form of binary classification. Additionally, our targeting variables (+ or -) is also in a binary classification sense. If we try to learn logistic regression classification in this decision tree, it is not going to be possible. The reason is that to learn regression classifier (i) tree must contain variables that is either numeric or continuous. (ii) tree variables cannot be trained with not enough data for logistic regression classifier. (Target variable is also not continuous). Therefore, the decision tree described above cannot be learned by regression classifier.

(c) Entropy of the following problem is that $H(labels) = -p(label = -) * \log_2(p(label = -)) - p(label = +) * \log_2(p(label = +))$. Through the calculation, I got $0.5 + 0.5 = 1$

(d) Information gain from x_2 is going to be $H(labels|x_2) = H(labels) - H(labels|x_2)$. We have calculated $H(labels)$ from part c and

$$H(labels|x_2) = -(\frac{1}{3} * \log_2(\frac{1}{3}) + \frac{2}{3} * \log_2(\frac{2}{3})) = 0.9183$$

Therefore, the calculation will have 1 - 0.9183 = 0.0817

2) Decision Tree (12 points) Let's investigate the accuracy of decision tree learning. We start by constructing a unit square ($[0; 1] \times [0; 1] \times [0; 1]$). We select n samples from the cube, each with a binary label (+1 or −1), such that no two samples share either x, y, or z coordinates. Each feature can be used multiple times in a decision tree. At each node we can only conduct a binary threshold split using one single feature.

(a) Prove that we can find a decision tree of depth at most $\log_2 n$, which perfectly labels all n samples.

(b) If the samples can share either two coordinates but not all three, can we still learn a decision tree which perfectly labels all n samples with the same depth as (a)? Why or why not?

Answer:

(a) As the problem has described, we will only conduct a binary threshold split by using on single feature. Since no two samples are sharing same x,y, or z coordinates, we know the the tree with binary splitting will be balanced. If we recursively split the node with the binary threshold, it is clear that the whole tree will be balanced. More intuitively, we will select one value (let's say feature) from a single feature and conduct binary split with a value within a single feature. Then we have divided into two groups. Now for another group (or for each children node from the first split), recursively conduct the binary split using the features we have from the samples. This is just same as a balanced binary tree which will be reduced by $\frac{1}{2}$. For example, since the examples are distinct, let's use x from set X to split. Then use y from Y to split again. Then this will automatically make 4 sets ...etc. By doing this recursively to the newly created sets, we will continue to conduct binary split until we go to leaf. Therefore, we can conclude that the tree is balanced and complete.
For the given examples size of n, the tree will be recursively go down by $\frac{1}{2}$. Therefore, the height of the tree will be $\log_2(n)$ because we have n numbers of node and $n \geqslant 2^h$. Therefore, the height of this case is $\log_2(n)$ at most.

(b) No. The reason is simple as before. Note that the balanced tree will have the height of the tree at most $\log_2(n)$. Also balanced tree has at most 1 height difference between right and left subtrees. From this, we will start.
Suppose we selected sample of size n and try to train them based on decision tree like before. But different than the previous work, now samples can share either two coordinates. This mean that whenever we conduct binary threshold split using one single feature, now we have few examples with sharing two same coordinates. Then it takes at least two more binary split operations to perfectly labels feature sharing two coordinates. Additionally, when we sample, they will be more than two examples sharing two coordinates. This situation will end up making the tree more skewed to either right or left subtrees which will lost the property of balanced subtree. Therefore, perfectly labeling samples with sharing two coordinates will take more than $\log_2(n)$. At the worst case, it might take n^2 time if it is fully skewed.

3) Adaboost (14 points) There is one good example at $x = 0$ and two negative examples at $x = \pm 1$. There are three weak classifiers are

$$\begin{aligned} h_1(x) &= 1 \cdot \mathbf{1}(x > 1/2) - 1 \cdot \mathbf{1}(x \le 1/2), \\ h_2(x) &= 1 \cdot \mathbf{1}(x > -1/2) - 1 \cdot \mathbf{1}(x \le -1/2) \\ h_3(x) &= 1. \end{aligned}$$

Show that this data can be classified correctly by a strong classifier which uses only three weak classifiers. Calculate the first two iterations of AdaBoost for this problem. Are they sufficient to classify the data correctly?

Answer:

1. According to the lecture note, we can find the strong classifier by using given weak classifier and modify them with some weights. For now, let's find what kind of the pure values we can get from the weak classifier $h_1(x), h_2(x), h_3(x)$. We have $x = -1, 1, 0$. If we plug them in respectively, then we can get the following numbers.
   $h_1(-1) = -1, h_2(-1) = -1, h_3(-1) = 1$
   $h_1(0) = -1, h_2(0) = 1, h_3(-1) = 1$
   $h_1(1) = 1, h_2(1) = 1, h_3(1) = 1$
   As explained, we can make weighted majority with w_k that we select such that prediction $\hat{y}$
   $$\hat{y} = sign(\sum_k w_k * h_k)$$
   where k $\in \{1, 2, 3\}$. As problem has mentioned at the beginning, we have negatives at x = -1, 1 and positive at x = 0. So.
   $$\hat{y} = sign(\sum_k w_k * h_k(-1)) < 0$$
   $$\hat{y} = sign(\sum_k w_k * h_k(0)) > 0$$
   $$\hat{y} = sign(\sum_k w_k * h_k(1)) < 0$$
   If we try arbitrary to choose w_1, w_2, w_3, we can now choose $w_1 = -1, w_2 = 2, w_3 = -1.1$. Then for the prediction values we will get
   $$\hat{y} = sign(\sum_k w_k * h_k(-1)) = 1 - 2 - 1.1 < 0$$
   $$\hat{y} = sign(\sum_k w_k * h_k(0)) = 1 + 2 - 1.1 > 0$$
   $$\hat{y} = sign(\sum_k w_k * h_k(1)) = -1 + 2 - 1.1 < 0$$
   Therefore, our strong classifier is $f(x) = -h_1(x) + 2h_2(x) - 1.1h_3(x)$

2. For adaboost, without choosing weight like above strong classifier, we can also get a correct classifier by learning.
   First of all let's follow the given algorithm of adaboost.
   (i) initialize $D_t(i) = \frac{1}{n} = \frac{1}{3}$ This is because we have three samples x = 1, -1, 0.
   (ii) For each iteration, we first need to calculate error.
   When t = 1, only correct prediction with h_1 is at x = -1. Therefore, we have two that are not correct. so $\varepsilon_t(h_1) = 2 * \frac{1}{3}$
   then we need to calculate $\alpha_t = \frac{1}{2} * \log \frac{1-\varepsilon_t(h_1)}{\varepsilon_t(h_1)}$, which is just -0.3467
   For easy notation, let $x_1 = -1, x_2 = 0, x_3 = 1$.

We need to now update our weight of each sample $D_{t+1}(i)$.
$Z_{t+1} = \sum_{i=1}^{3} D_t i {*} \exp^{-\alpha_t * y_i * h_1(x_i)} = \frac{1}{3}\exp^{-0.3467*-1*-1} + \frac{1}{3}\exp^{-0.3467*1*-1} + \frac{1}{3}\exp^{-0.3467*-1*1}$
Therefore, $Z_2 = 0.94274$. So
$D_2(x_1) = \frac{1}{0.94274} * \frac{1}{3}\exp^{-0.3467*-1*-1} = 0.5$
$D_2(x_2) = \frac{1}{0.94274} * \frac{1}{3}\exp^{-0.3467*1*-1} = 0.25$
$D_2(x_3) = \frac{1}{0.94274} * \frac{1}{3}\exp^{-0.3467*1*1} = 0.25$

Since we got all the values ready, now t = 2.
As before, we need to find error. $\varepsilon_t(h_2) = 0.25$ This is because only x_3 got the wrong prediction.

$$\alpha_t = \frac{1}{2} * \log\frac{1-\varepsilon_t(h_2)}{\varepsilon_t(h_2)} = 0.5 * \log\frac{1-0.25}{0.25} = .5493$$

We will repeat the process again.

$$Z_{t+1} = \sum_{i=1}^{3} D_t i {*} \exp^{-\alpha_t * y_i * h_1(x_i)} = 0.5\exp^{-.5493*-1*-1} + 0.25\exp^{-.5493*1*1} + 0.25\exp^{-.5493*-1*1}$$

$$Z_3 = 0.1443 + 0.289 + 0..443 = 0.876$$

$D_3(x_1) = \frac{0.289}{0.876} = 0.33$
$D_3(x_2) = \frac{0.1443}{0.876} = 0.16$
$D_3(x_3) = \frac{0.443}{0.876} = 0.51$
Now we can make the prediction using Adaboost.

$$\hat{y} = f(x) = argmax_{\hat{y}} \sum_{t=1}^{2} \alpha_t * [h_t(x) = \hat{y}]$$

$$\hat{y} = argmax_{\hat{y}}(-0.3467[h_1(x_1) = \hat{y} + 0.5493[h_2(x_1) = \hat{y}) = -1$$

$$\hat{y} = argmax_{\hat{y}}(-0.3467[h_1(x_2) = \hat{y} + 0.5493[h_2(x_2) = \hat{y}) = 1$$

$$\hat{y} = argmax_{\hat{y}}(-0.3467[h_1(x_3) = \hat{y} + 0.5493[h_2(x_3) = \hat{y}) = 1$$

The prediction is followed by above. At = 2, we have classified correctly because only -1 will be penalized (or getting weight) from the h_2 function while x = 0, 1 will lie on positive side. Therefore, adaboost is sufficient to classify the data correctly.

4) Ensemble Methods (12 points) Consider the following binary classification Boosting algorithm.

1. Given $\{\mathbf{x}_i, y_i\}_{i=1}^N$, number of iterations T, weak learner f.
2. Initialize $\mathcal{D}_0$ to be a uniform distribution over examples.
3. For each iteration $t = 1 \ldots T$:
   (a) Train a weak learner f on the data given $\mathcal{D}_t$ to produce hypothesis h_t.
   (b) Compute the error of h_t as $\epsilon_t = P_{\mathcal{D}_t}[h_t(\mathbf{x}_i) \neq y_i]$
   (c) Compute $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$

(d) Update $\mathcal{D}$ as:

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t + (T-t)/T) & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp(\alpha_t + (T-t)/T) & \text{otherwise} \end{cases}$$

4. Output final hypothesis $H(\mathbf{x}) = \text{sign}\left\{\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right\}$

Z_t is a normalization constant so that $\mathcal{D}$ is a valid probability distribution.

Describe the difference between this algorithm and the AdaBoost algorithm we learned about in class. What problem of AdaBoost is this change designed to fix? How does changing the algorithm's user provided parameter affect this behavior?

Answer: The difference of this algorithm from the algorithm that we have learned during the lecture is that they have added $\frac{T-t}{T}$ to the exponential of updating $D_{t+1}(i)$. As the result, the updating D will look like $D_{t+1}(i) = \frac{D_t(i)}{Z_t} * exp^{((-\alpha_t + \frac{T-t}{T})y_i h_t(x_i))}$.
As we have high iteration, we will attribute less to α and when we have low iteration, we will have higher attribution to α.
The problem that this algorithm is trying to make error quickly than the other model. There is particular behavior of adaboost that as iteration goes up, the error reduces and then the alpha value increases. This will push the error to zero. However, by added another term $\frac{T-t}{T}$ to the new equation does not affect the behavior as it is attempted. The same constant $\frac{T-t}{T}$ multiplied to the every term and this will be canceled out due to the normalization value Z_t. As we think of the equation of normalization distribution, then it is clear that the multiplying $\frac{T-t}{T}$ will be reduced by the normalization Z_t. Hence there is no impact on this algorithm with the addition of $\frac{T-t}{T}$.