CS 475 Machine Learning: Homework 3 Analytical (50 points)

Assigned: Friday, October 04, 2024

Due: Friday, October 18, 2024, 11:59 pm US/Eastern

Trevor Black (tblack20)

Instructions

We have provided this LaTeX document for turning in this homework. We give you one or more boxes to answer each question. The question to answer for each box will be noted in the title of the box. You can change the size of the box if you need more space.

Other than your name, do not type anything outside the boxes. Leave the rest of the document unchanged.

Do not change any formatting in this document, or we may be unable to grade your work. This includes, but is not limited to, the font sizes, and the spacing of text and tables. Additionally, do not add text outside of the answer boxes. Entering your answers are the only changes allowed.

We strongly recommend you review your answers in the generated PDF to ensure they appear correct. We will grade what appears in the answer boxes in the submitted PDF, NOT the original latex file.

1 Adaboost [25 pts]

Assume a one-dimensional dataset with $x = \langle -1, 0, 1\rangle$ and $y = \langle -1, +1, -1\rangle$. Consider three weak classifiers:

$$h_1(x) = \begin{cases} 1 & \text{if } x > \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}, \qquad h_2(x) = \begin{cases} 1 & \text{if } x > -\frac{1}{2} \\ -1 & \text{otherwise} \end{cases}, \qquad h_3(x) = 1.$$

(1) Show your work for the first 2 iterations of AdaBoost. You must use the method from the lecture slides. In the table below, replace the "?" in the table below with the value of the quantity at iteration t. $h_t \in \{1, 2, 3\}$, weak learner's weight in the ensemble, α_t, error rate, ε_t. Additionally, give the distribution over example at iteration t, $D_t(1), D_t(2), D_t(3)$. Use log with base e in your calculations.

For the first iteration, start with predictions from $h_t(x)$.

$h_1(x)$ predicts $\langle -1, -1, 1\rangle$, so the error is 2/3. $h_2(x)$ predicts $\langle -1, 1, 1\rangle$, so the error is 1/3. $h_3(x)$ predicts $\langle 1, 1, 1\rangle$, so its error is 2/3. $h_2(x)$ had the lowest error, so $h_1 = h_2(x)$ with an error rate of $\epsilon_1 = 1/3$ Computing the α value, we get $\alpha_1 = \frac{1}{2}\ln(\frac{1-\epsilon_1}{\epsilon_1}) = .347$

Moving on to the next iteration, we start by updating the values for $D_2(1), D_2(2), D_2(3)$ using the formula $D_2(i) = \frac{D_1(i)}{Z_1} \cdot \exp(-\alpha_1 y_i h_1(x_i))$. First, we find the normalizing factor $Z_1 = \Sigma D_1(i)\exp(-\alpha_1 y_i h_1(x_i)) = .472 + .236 + .472 = 1.18$. Now we can plug in everything to find $D_2(1) = .472/1.18 = .4, D_2(2) = .236/1.18 = .2, D_2(3) = .472/1.18 = .4$.

With these values found, we can proceded as before to find the new values for h_2, α_2, and ϵ_2. The lowest error weak classifier is $h_1(x) = 0.2 + 0.4 = 0.6$ as $h_2(x)$ has already been chosen and $h_3(x)$ has a higher error rate. This lets us fill in our table with values $h_2 = 1$ and $\epsilon_2 = .6$. Computing $\alpha_2 = \frac{1}{2}\ln(\frac{1-\epsilon_2}{\epsilon_2}) = -.202$.

t	h_t	α_t	ε_t	$D_t(1)$	$D_t(2)$	$D_t(3)$
1	2	.347	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
2	1	-.202	.6	.4	.2	.4

(2) After running AdaBoost for 10 iterations, what is the error rate of the ensemble?

The ensemble error rate is defined as the fraction of data points that are classified incorrectly. It is bounded by $\prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}$. We can assume that the classifier does better than random (especially after an iteration of AdaBoost). Basing off of the first value of $\epsilon_1 = .33$, we can approximate an upper bound the error rate to be $(2\sqrt{.33 \cdot .77})^{10} = .541$.

(3) In general, getting near-perfect training accuracy in machine learning leads to overfitting. However, AdaBoost can get perfect training accuracy within overfitting. Give a brief justification for why that is the case.

Typically, high training accuracy leads to overfitting because the model learns exactly the data it is trained on, so any minor deviations from that tend to spit out garbage. AdaBoost overcomes this by tending to maximize the margin. This ensures it doesn't conform too much to the training data and allows for small deviations in the input to not alter the output. It does this by altering the weights of the classifiers so that they only have an effect in smaller regions of the wider model. This feature of the algorithm allows it to maintain accuracy while avoiding overfitting.

(4) The AdaBoost algorithm makes calls to a weighted classification solver which approximately solves the weighted 0/1-loss problem.[1] In other words, the classifier training algorithm is a function from a weighted dataset $\{(w_i, x_i, y_i)\}_{i=1}^n$ to a classifier which approximately minimizes,

$$\operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} w_i \mathbf{1}(h(x_i) \neq y_i) \tag{1}$$

Suppose we have a new implementation of a classifier and want to use it in AdaBoost. However, the classifier's training algorithm only accepts a dataset of x-y pairs, $\{(x_i, y_i)\}_{i=1}^n$ and thus it solves,

$$\operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(h(x_i) \neq y_i) \tag{2}$$

Give a give a principled method to *approximate* the weighted classification solution (1) given access to a training algorithm that only accepts x-y pairs (2). Briefly sketch your idea and why it is a reasonable approximation.

The key concept for the solution is to transform dataset into one that mimics the weighted version by resampling the dataset to include more of the heavily weighted datapoints. In other words, we must create an unweighted dataset where datapoints with higher weights are more likely to be sampled multiple times. This keeps the relative importance of the datapoints while removing the need for another parameter needing to be passed in.

In more detail, the original weighted dataset must be resampled to form a new dataset. To maintain that more heavily weighted datapoints keep their importance, the chance of any datapoint being sampled is dependent on its weight.

This approximation works because it keeps the relative importance of of each datapoint. It transfers that information from a separate 'weight' parameter to a frequency of occurences in the dataset.

[1] The approximation may come from minimizing an upper bound on 0/1 loss such as hinge loss.

2 Linear Classifier and Perceptron [25 pts]

Given a dataset with two binary features, X_1 and X_2, the table below outlines all possible data points along with the results of applying the OR and AND functions to each example.

x_1	x_2	OR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Suppose there is a linear classifier with parameters $\boldsymbol{w} = (w_0, w_1, w_2)$ such that

$$y = \begin{cases} 1 & \text{if } w_1 x_1 + w_2 x_2 + w_0 \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

(1) Give two sets of valid values for w_1, w_2 and w_0 to represent the OR function. Show your work.

To find a set of valid values for w_1, w_2, and w_0 to represent the OR function, we start by setting the inequality $w_1 x_1 + w_2 x_2 + w_0 < 0 \iff x_1 = x_2 = 0$. This is true because based on the above table, we can see that a 0 must be output if and only if both x_1 and x_2 are 0. Plugging in our values, we see that $w_0 < 0$. Additionally, we see that if either value of x_1 or x_2 is 1, the other condition must hold: $w_1 x_1 + w_2 x_2 + w_0 \geq 0$. From these constraints we can choose a negative value for w_0 that can be overcome by just w_1 or w_2 (no need for both). The following values fit that criteria: $w_0 = -1, w_1 = w_2 = 2$.

To find another set of valid values, we can easily modify w_1 and w_2 to be something larger, as any value for them larger than $-w_0$ will work. i.e. $w_0 = -1, w_1 = w_2 = 5$.

(2) Give two sets of valid values for w_1, w_2 and w_0 to represent the AND function. Show your work.

To find a set of valid values for w_1, w_2, and w_0 to represent the AND function, we start by setting the inequality $w_1 x_1 + w_2 x_2 + w_0 \geq 0 \iff x_1 = x_2 = 1$. From this, we can set our values to find $w_1 + w_2 + w_0 \geq 0$. The setup also requires that $w_1 x_1 + w_2 x_2 + w_0 < 0$ if either value of x is 0. From this, we can see that w_0 must be -2, meaning w_1 and w_2 should each be 1.

To find another set of valid values, we can easily multiply each parameter by the same constant value, i.e. 10. So $w_0 = -20, w_1 = w_2 = 10$. In general, w_1 and w_2 must be between -1x and -0.5x the value of w_0 so that when both terms are present they can 'overcome' w_0.

(3) Can the Perceptron algorithm learn the above-defined OR and AND functions? Briefly explain.

There is no perceptron algorithm that can learn the above-defined OR and AND functions because the constraints on them both are mutually exclusive. While they both require a negative value for w_0, the OR function requires w_1 and w_2 be larger than $-w_0$, while the AND function requires them to be less than $-w_0$.

(4) We know that the XOR function is not linearly separable in two dimensions, but it can be made separable in higher dimensions with the addition of a suitable feature. Propose a new feature that can be added to the input space to make the XOR function linearly separable in three dimensions. Your answer should start with why XOR is not linearly separable in two dimensions, then define a new feature, give a new truth table, and finally demonstrate the feature effectively makes XOR linearly separable in three dimensions.

XOR is not linearly separable in two dimensions as can be visually proven in a simple graph. In the below graph for XOR, we can see that the common outputs are diagonal to each other, so any hyperplane correctly grouping two of them together would inevitably incorrectly group another value with them.

XOR	$x_1 = 1$	$x_2 = 1$
$x_1 = 0$	1	0
$x_2 = 0$	0	1

We can make the new feature $z = x_1 \cdot x_2$. This brings the problem space into the third dimension, allowing it to be linearly separable. The new truth table below, while cannot be represented in 3D, provides some insight on how this extra dimension allows the XOR function to be linearly separable.

x_1	x_2	z	XOR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

This is linearly separable because the points $(0, 0, 0)$ and $(1, 1, 1)$ (both of which result in 0) lie on the plane $z = x_1 \cdot x_2$. Additionally, points $(0, 1, 0)$ and $(1, 0, 0)$ lie on a different part of the 3D space. Because of this, we can define a plane to separate the classes, like $x_1 + x_2 - z \geq 0$ to classify a 1 for the XOR function, and 0 otherwise.