Problem Set 1 Answers

Jonathan McEntee

9/22/2018

1 For almost every case we have discussed where we are doing supervised learning, we have assumed a deterministic function. Imagine instead a world where we are trying to capture a non-deterministic function. In this case, we might see training pairs where the x value appears several times, but with different y values. For example, we might be mapping attributes of humans to whether they are likely to have had chicken pox. In that case, we might see the same kind of person many times but sometimes they will have had chicken pox, sometimes not. We would like to build a learning algorithm that will compute the probability that a particular kind of person has chicken pox. So, given a set of training data where each x is mapped to 1 for true or 0 for false:

1.1 Derive the proper error function to use for finding the ML hypothesis using Bayes Rule. You should go through a similar process as the one used to derive least squared error in the lessons.

For hypothesis space H, and sample distribution D, the maximum a priori hypothesis is defined as:

$$h_{MAP} = argmax_{h \in H} P(h|D) = argmax_{h \in H} \frac{P(D|h)P(h)}{P(D)} = argmax_{h \in H} P(D|h)P(h)$$

If we assume all $h \in H$ are equally likely, then the maximum likelihood likelihood function is defined as:

$$h_{ML} = argmax_{h \in H} P(D|h)$$

If we have a set of m samples with features and labels as such $< x_1, d_1 >, < x_2, d_2 > ... < x_m, d_m >$, then the hypothesis with the highest likelihood is:

$$h_{ML} = argmax_{h \in H} \prod_{n=1}^{m} f(x_n)$$

where:

$$f(x_n) = \begin{cases} h(x_n), & \text{if } d_n = 1 \\ 1 - h(x_n), & \text{if } d_n = 0 \end{cases}$$

Which is equivalent to:

$$h_{ML} = argmax_{h \in H} \prod_{n=1}^{m} h(x_n)^{d_n} (1 - h(x_n))^{1 - d_n}$$

And can be made simpler to compute by taking the logarithm of the statement inside the $argmax$. By multiplying the statement by -1, we make it into a loss function to be minimized.

$$h_{ML} = argmin_{h \in H} - \sum_{n=1}^{m} d_n ln(h(x_n)) - (1 - d_n) ln(1 - h(x_n))$$

1.2 Compare and contrast your result to the rule we derived for a deterministic function perturbed by zero-mean gaussian noise. What would a normal neural network using sum of squared errors do with these data? What if the data consisted of x,y pairs where y was an estimate of the probability instead of 0s and 1s?

While both seek to maximize the likelihood of the chosen hypothesis, the rules are based on different assumptions. One assumes the hypothesis outputs a probability, the other assumes the hypothesis estimates a some target function $f(x)$ perturbed by gaussian noise. If a neural network seeking to minimize the sum of squared error might output funny results. For example the NN might select a hypothesis that outputs 1.1 rather than a hypothesis that outputs 0.8 for some sample with target value 1, despite the fact that 1.1 is not a valid probability. The same issue persists if we're estimating probabilities rather than 0s and 1s.

2 Design a two-input perceptron that implements the boolean function $A \wedge \neg B$. Design a two-layer network of perceptrons that implements $A \oplus B$ ($\oplus$ is XOR).

Imagine a two input perceptron with inputs A and B, weights w_A and w_B, and a threshold θ. Let $w_A = 1$, $w_B = -2$ and $\theta = \frac{1}{2}$. Then:

- if $A = 1$ and $B = 0$ then $w_A \cdot A + w_B \cdot B = 1 \cdot 1 + -2 \cdot 0 = 1 > \frac{1}{2} = \theta$. The perceptron outputs 1.
- if $A = 1$ and $B = 1$ then $w_A \cdot A + w_B \cdot B = 1 \cdot 1 + -2 \cdot 1 = -1 < \frac{1}{2} = \theta$. The perceptron outputs 0.
- if $A = 0$ and $B = 1$ then $w_A \cdot A + w_B \cdot B = 1 \cdot 0 + -2 \cdot 1 = -2 < \frac{1}{2} = \theta$. The perceptron outputs 0.
- if $A = 0$ and $B = 0$ then $w_A \cdot A + w_B \cdot B = 1 \cdot 0 + -2 \cdot 0 = 0 < \frac{1}{2} = \theta$. The perceptron outputs 0.

Therefore this perceptron implements $A \wedge \neg B$. If $w_A = 1$, $w_B = 1$ and $\theta = \frac{3}{2}$, the perceptron implements $A \wedge B$. If $w_A = 1$, $w_B = 1$, and $\theta = \frac{1}{2}$, the perceptron implements $A \vee B$.

Imagine a two layer perceptron network with inputs A and B. In the first layer consists of two perceptrons. One (C) implements $A \vee B$, and the second (D) implements $A \wedge B$ (henceforth D). Let the second layer's single perceptron (E) implement $C \wedge \neg D$. Then:

- if $A = 1$ and $B = 0$ then $C = 1$ and $D = 0$. Then $E = 1$
- if $A = 1$ and $B = 1$ then $C = 1$ and $D = 1$. Then $E = 0$
- if $A = 0$ and $B = 1$ then $C = 1$ and $D = 0$. Then $E = 1$
- if $A = 0$ and $B = 0$ then $C = 0$ and $D = 0$. Then $E = 0$

Therefore this perceptron network implements $A \oplus B$.

3.1 Derive the perceptron training rule and gradient descent training rule for a single unit with output $o = w_0 + w_1 x_1 + w_2 x_1^2 + ... + w_n x_n^2$.

This is the perceptron rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where normally

$$\Delta w_i = \eta(t - o)x_i$$

But for this perceptron, we have:

$$\Delta w_i = \eta(t - o)x_i + \eta(t - o)x_i^2 = \eta(t - o)(x_i + x_i^2)$$

For gradient descent we need to partial derivative of the error :

$$E(w) = \frac{1}{2}(t - o)^2$$

with respect to each weight $w_1, w_2, ..., w_n$. By the chain rule we have:

$$\frac{\partial E}{\partial w_i} = \frac{dE}{do}\frac{\partial o}{\partial w_i}$$

Where:

$$\frac{dE}{do} = \frac{1}{2}(t - o)^2 \frac{d}{do} = \frac{1}{2}2(t - o)\frac{d(t - o)}{do} = o - t$$

And:

$$\frac{\partial o}{\partial w_i} = \sum_{n=1}^{m} w_n x_n + w_n x_n^2 \frac{\partial}{\partial w_i} = x_i + x_i^2$$

Therefore:

$$\frac{\partial E}{\partial w_i} = (o - t)(x_i + x_i^2)$$

And our weight update rule is:

$$\Delta w_i = -\frac{\partial E}{\partial w_i} = (t - o)(x_i + x_i^2)$$

If we want true gradient descent rather than stochastic gradient descent, we calculate Δw_i by summing the gradient over all samples:

$$\Delta w_i = (t - o) \sum_{n=1}^{m} x_{in} + x_{in}^2$$

where x_{in} is the nth sample of the ith feature.

3.2 What are the advantages of using gradient descent training rule for training neural networks over the perceptron training rule?

The perceptron training rule will always be able to perfectly classify data which is linearly seperable. However, data isn't always linearly seperable in practice, and the perceptron may never converge on non-linearly seperable data. Gradient descent can usually converge to a local optima even for non linearly seperable data, making it more robust than the perceptron. However, gradient descent is not guaranteed to find a perfect classification for the data or even the global optimum.

4.1 Explain how you can use Decision Trees to perform regression?

To perform regression using decision trees you need a new criterion with which to select features. Information gain via entropy or gini measures won't work for a non-classification problem. The scikit learn library for example, uses a measure that finds splits which minimize the average variance of the leaf nodes. Once you've constructed a tree, you can find the output value by, for example, returning the mean of the samples in the leaf, or performing a localized regression on the samples.

4.2 Show that when the error function is squared error, then the expected value at any leaf is the mean.

Suppose we have a set of n target points in a leaf node: $x_1, x_2, ..., x_n$, and our error function is squared error.

4.3 Take the Boston Housing dataset (https://archive.ics.uci.edu/ml/datasets/Housing) and use Decision Trees to perform regression.

6 Imagine you had a learning problem with an instance space of points on the plane and a target function that you knew took the form of a line on the plane where all points on one side of the line are positive and all those on the other are negative. If you were constrained to only use decision tree or nearest-neighbor learning, which would you use? Why?

I would use a decision tree. K nearest neighbors holds a fundamental assumption that values which are closer to each other are more likely to be similar. This does not hold with a divided plane.

For example, imagine 4 points on a 2-D plane divided on the y-axis with the same y value and x values equal to -3, -2, -1, 1, 6, and 7. The points on the negative side of the y-axis are of one class and those on the positive side are on the other. A 3 nearest neighbor algorithm would misclassify -1 as a positive example. A decision tree however, could create branches for $x > 0$ or $x \leq 0$, which would perfectly classify the data.

The target concept will always be within the version space of the decision tree algorithm, but that is not the case with KNN.

7 Give the VC dimension of the following hypothesis spaces. Briefly explain your answers.

1. An origin-centered circle (2D)

Imagine a point $x \in \mathbb{R}^2$, and an origin-centered circle with radius r. I define x as being within the circle (+) if and only if $\|x\| \leq r$. Otherwise it will be outside of the circle (-). Therefore the VC dimension is greater than or equal to 1.

Imagine two points $x_1, x_2 \in \mathbb{R}^2$ such that $\|x_1\| \leq \|x_2\|$. For the VC dimension to be greater than or equal to 2, it must be possible for x_2 to be within the circle (+), while x_2 is outside of it (-). However x_2 being within the circle implies $\|x_2\| \leq r$, which in turn implies $\|x_1\| \leq r$. Therefore it impossible to shatter x_1 and x_2. The VC dimension is 1.

2. An origin-centered sphere (3D)

Like in 2-dimensional space, I consider a point $x \in \mathbb{R}^3$ inside the origin-centered sphere if and only if $\|x\| \leq r$. If you accept that definition, then following the same argument as before, the VC dimension is 1.