Problem set 1

Rohan D. Kekatpure

1. Problem 1

1.1. Derivation of error function

Assume our training set is,

$$\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\} \tag{1}$$

where $y_i \in \{0, 1\}$.

We want to learn a hypothesis h that outputs the probability that $y_i = 1$ given the data point x_i. Explicitly,

$$P(y_i = 1|h, x_i) = h(x_i) \tag{2}$$
$$P(y_i = 0|h, x_i) = 1 - h(x_i) \tag{3}$$

Using a trick (Mitchell p. 169), these two equations can be combined as

$$P(Y = y_i|h, x_i) = h(x_i)^{y_i}(1 - h(x_i))^{1-y_i} \tag{4}$$

When $y_i = 1$ the second term drops out and and we get (2). When $y_i = 0$, the first term drops and we recover (3).

To derive the likelihood function, we have to assume that (1) all data points are independent and (2) the error terms are identically distributed (i.e. the probability distribution of the error terms does not depend on the input x_i). Under these assumptions, the likelihood function is:

$$P(D|h) = \prod_{i=1}^{n} h(x_i)^{y_i}(1 - h(x_i))^{1-y_i} \tag{5}$$

The maximum likelihood (ML) hypothesis is the one that maximizes (5):

$$h_{ML} = \arg\max_h \prod_{i=1}^{n} h(x_i)^{y_i}(1 - h(x_i))^{1-y_i} \tag{6}$$

Equivalently, we can simplify (6) by taking natural logs of both sides. Also, since we're asked to derive the *error* function, we take the negative logarithm:

$$\boxed{J_n = -\sum_{i=1}^{n} y_i \ln h(x_i) + (1 - y_i)\ln(1 - h(x_i))} \tag{7}$$

(7) is the error or cost function to minimize in case of non-deterministic functions.

1.2. Comparison to error function for deterministic function

Below we compare the error terms obtained for deterministic and non-deterministic functions:

1. Both error functions are convex (sum of convex functions like 'log' is convex), so both are equally amenable to be solved by standard optimization techniques.

2. For non-deterministic function, we had to assume that the individual error terms followed a Bernoulli distribution (i.e. a flip of a coin) instead of a Gaussian.
3. Because of unbounded functions like natural logs, the logistic error terms seems more susceptible to numerical instabilities due to noise in data (where the arguments of the log function get close to zero).

1.3. Neural network

Non-deterministic functions cannot guaranteed to be linearly separable. So a neural network trained using the perceptron would not be able to find a decision boundary with certainty. A neural network trained using gradient descent would need to be used.

However, for gradient descent training, a neural network trained with least squares error function seeks maximum likelihood hypothesis under the assumption that training data is modeled by a Normally distributed noise added to the target function value. In contrast a neural network trained using cross entropy minimization will seek maximum likelihood hypothesis under the assumption that the observed boolean value is a probabilistic function of the input (Mitchell P. 171)

1.4. Y continuous instead of discrete variable

When y is a probability (i.e., a continuous variable), then its a *regression* problem. The error function above (7) does not make sense for a regression problem. In this case, it is preferable to perform an ordinary least-squares (OLS) regression.

2. Problem 2

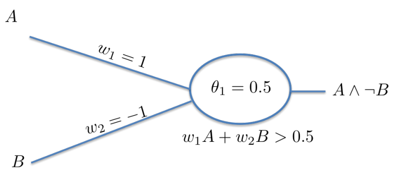

Fig. 1. Two input perceptron for $A \wedge \neg B$

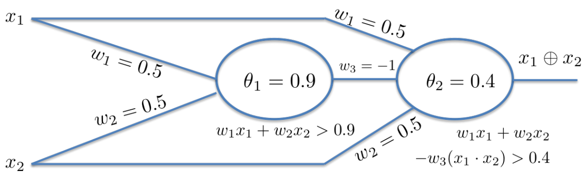

Fig. 2. Two layer perceptron for $A \oplus B$

3. Problem 3

3.1. Perceptron training

The output of the perceptron now contains a quadratic term

$$o = w_0 + w_1(x_1 + x_1^2) + w_2(x_2 + x_2^2) + \cdots + w_n(x_n + x_n^2) \tag{8}$$

Perceptron training rule follows from Mitchell §4.4.2 by replacing $x_i \to x_i + x_i^2$:

$$\boxed{\Delta w_i = \eta(t-o)(x_i + x_i^2)} \tag{9}$$

Note that $\Delta w_i > 0$ (i.e. weight should be increased) if $t > o$ and $x_i + x_i^2 > 0$ and vice versa.

3.2. Gradient descent training

For gradient descent, we use the component form of the training rule:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{10}$$

where E is the sum-of-squared error function defined as

$$E(w) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{11}$$

Thus,

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{1}{2} \frac{\partial}{\partial w_i} \sum_{d \in D} (t_i - o_i)^2 \\
&= \sum_{d \in D} (t_i - o_i) \frac{\partial}{\partial w_i} (t_i - o_i) \\
&= -\sum_{d \in D} (t_i - o_i)(x_{id} + x_{id}^2),
\end{aligned} \tag{12}$$

where x_{id} refers to i'th component (feature) of the d'th training example.

The GD training rule is thus:

$$\boxed{\Delta w_i = \eta \sum_{d \in D} (t_i - o_i)(x_{id} + x_{id}^2)} \tag{13}$$

4. Problem 4

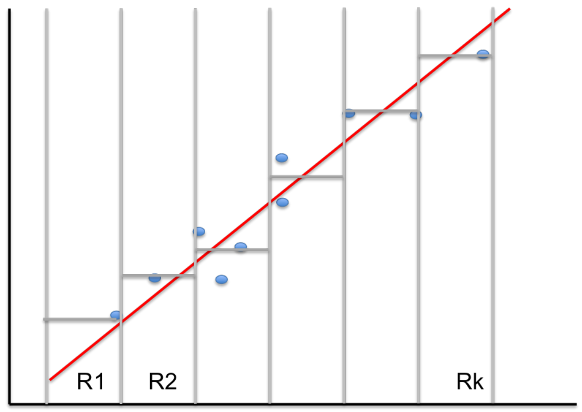

Fig. 3. Regression trees

Using decision trees for regression: Decision trees can be used to perform regression by partitioning the feature space into suitable number of regions and assigning a target y value to each region. A one-dimensional example is shown in figure 3 where the X-axis is partitioned into segments and each segment is assigned a y value shown by the horizontal line.

The next question is the splitting criterion and choice of y-value. One possible choice, as mentioned in the problem statement, is a least squared error function. Under this choice, one can perform a recursive, greedy binary splits at every stage such that the sum of squares in each half plane is minimized. The procedure is then repeated recursively in every half plane until the stopping criterion (depth, number of leaves etc) is met.

The y-value for each region R_i is the *mean* of all y values falling in that region. This statement can be proved in the following way. The total sum of squares is written in the usual way:

$$E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{14}$$

where $\hat{y}_i$ is the predicted output. According to our algorithm, $\hat{y}_i$ is a constant y_r for region $R = r$. Thus the sum of squares can be written over each region r as:

$$E = \sum_{r \in R} \sum_{i=1}^{n_r} (y_i - y_r)^2 \tag{15}$$

where n_r is the number of training examples with output falling in region $R = r$. We can find out all the y_r values by minimizing (15) with respect to y_r.

$$\frac{\partial E}{\partial y_r} = 2 \sum_{i=1}^{n_r} (y_i - y_r)$$

$$\frac{\partial E}{\partial y_r} = 0 \Rightarrow \boxed{y_r = \frac{1}{n_r} \sum_{i=1}^{n_r} y_i} \tag{16}$$

In words, total sum of squares is minimized when the ‘constant’ y-value in each leaf node is the average of the all y values falling in that region.

4.1. Regression on Boston housing dataset:

Not implemented.

5. Problem 6

Short answer: If the line defining the target function is known to be axis-parallel (i.e., parallel to one of the axes) then decision trees or KNN would perform equally well. If the line defining the target function is not axis parallel (i.e. at a non-zero angle to both axes), then I’d choose KNN.

Rationale: Decision trees splits divide the plane into axis-parallel rectangles. As such, when the data is not separable by axis parallel rectangles, decision trees tend to underperform. Particularly, decision trees face issues when classifying data points which are linearly separable but the separating line is *not* axis parallel. This leads to the ‘staircase’ decision boundary. As such KNN seems to be a better choice since it does not suffer from the bias of choosing axis-parallel splits.

However, KNN could potentially suffer in the case when there is not enough training data.