-Problem set 1

1/ We have the definition for maximum likelihood as

h_{ML} = argmax P(D|h) for all h in H

Since the data came from a nondeterministic function, the result of P(D|h) is

$$\mathbf{P(D|h) = \prod(P(x_i, d_i \mid h)) \text{ for } i \text{ from } 1 \text{ to } m}$$

x is independent of h, so we can rewrite the equation as

$$\mathbf{P(D|h) = \prod P(x_i, d_i \mid h) = \prod P(d_i \mid x_i, h)P(x_i)}$$

$$\mathbf{P(d_i| h, x_i) = h(x_i) \text{ if } d_i = 1 \text{ and } 1 - h(x_i) \text{ if } d_i = 0}$$

=> $\mathbf{P(d_i| h, x_i) = h(x_i)^{di}(1 - h(x_i))^{1-di}}$

=> $\mathbf{P(D|h) = \prod h(x_i)^{di}(1 - h(x_i))^{1-di}P(x_i)}$

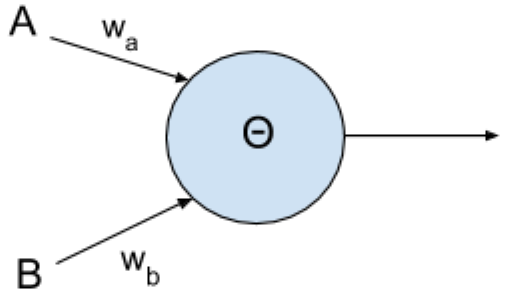
=> $\mathbf{h_{ML} = argmax\prod h(x_i)^{di}(1 - h(x_i))^{1-di}P(x_i)}$

=> $\mathbf{h_{ML} = argmax\prod h(x_i)^{di}(1 - h(x_i))^{1-di}}$

We will work with the log likelihood for ease of computation

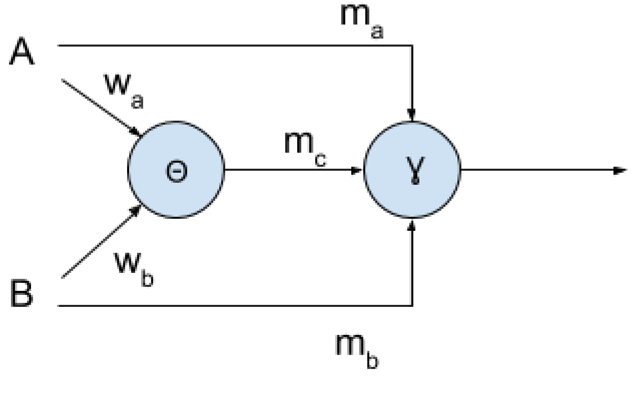
$$\mathbf{h_{ML} = argmax\prod d_i ln(h(x_i)) + (1 - d_i)ln(1 - h(x_i))}$$

2/ Design perceptron for A ∧¬B, where A and B takes on values { 0, 1 }



The corresponding weights are $\mathbf{w_a = 1, w_b = -2, Θ = 0.5}$ which satisfies the equation: $\mathbf{A*w_a + B*w_b >= Θ}$

Design 2-layer perceptrons network for XOR



The first layer will represent an AND logic function. The output of the first layer is fed into the second layer together with A and B (figure above)

The corresponding weights for layer one are **$w_a = 0.5$, $w_b = 0.5$, $\Theta = 1$** which satisfies the equation: **$A*w_a + B*w_b >= \Theta$**

The corresponding weights for layer two are **$m_a = 0.5$, $m_b = 0.5$, $m_c = -1$, $\gamma = 0.5$** which satisfies the equation: **$A*m_a + B*m_b + layer\_1*m_c >= \gamma$**

3/ The current definition for output is:

$o(x) = 1$ if $w_0 + w_1x_1 + w_1x_1^2 + \dots + w_nx_n + w_nx_n^2 > 0$, -1 otherwise

We can define a constant x_0 such that **$x_0^2 + x_0 = 1$** and simplify the output function into the form **$o(x) = 1$ if $\sum w_i(x_i + x_i^2) > 0$ for $i = 0..n$, -1 otherwise**

We can further simplify the formula by defining capturing the sign of the output function: **$sgn(o) = 1$ if $o > 0$, -1 otherwise**

For perception rule, all the weights are modified according to each x_i following the rule: **$w_i = w_i + \Delta w_i$** where **$\Delta w_i = \eta(y - sign(o))x_i$**

For gradient descent, we will start with the error function:

$E(\vec{w}) = ½ \sum (y_d - o_d)^2$ where d is a training sample

The idea for gradient descent is to minimize the error function with respect to vector $w^{->}$

The same rule applies for updating th e weight:

$w_i = w_i + \Delta w_i$ where **$\Delta w_i = -\eta \nabla E(w^{->})$**

Taking the partial derivative of **$\nabla E(w^{->})$** with respect to each w_i we come up with the following formula:

$\nabla E(w^{->}) = \sum(y_d - o_d)(-x_{id})$

$\Rightarrow \Delta w_i = \eta \sum(y_d - o_d)x_{id}$

The advantage of gradient descent over perceptron rule is that it can deal with non-linearly separable dataset. Perceptron rule will fail to converge.

4/ In order to implement decision tree for a regression problem, we will use a new splitting criteria which is standard deviation reduction.

Notation definition:

S(T) = standard deviation for the target

S(X) = standard deviation for multiple values = $\sum P(c)S(c)$ for c in attributes(X)

SDR(T, X) = standard deviation reduction = S(T) - S(X)

Step 1, we will calculate the standard deviation for the current training dataset S(T).

Step 2, for each feature will calculate its standard deviation, and calculate the standard deviation reduction.

Step 3, pick the feature with highest standard deviation reduction as the new node in the tree

Step 4, divide the dataset into subsets according to the values of the feature.

Step 5, repeat the process on the non-leaf branches, until all data is processed. When the number of instances is more than one at a leaf node, we will calculate the average as the final value.

5/ The lazy implementation of ID3 will be similar to the KNN algorithm in that there is no training phase to build the tree classifier. The algorithm will use the query instance to tailor which imaginary sub-tree should be built to predict the label.

The training phase will now be used to determine which attributes are important by looking at the query instance's features.

The classification phase will now build the actual tree that is relevant to the query instance. The algorithm will only look at the attributes derived in the training phase as the possible splits when building the tree. The splitting logic and building the tree is very similar to the original ID3 algorithm.

The advantages of using the lazy algorithm are:

- Significantly reduce the training time.
- The resulting tree for the query instance is smaller than the actual tree
- The total time spent to answer one query instance is much less than the original algorithm

The disadvantages are:

- The resulting tree does not generalize the structure of the training data. Instead, a new tree will have to be built every time a new instance is queried.
- Over the long run, the lazy algorithm will take more time to construct the trees to answer the queries.
- We may have many trees that are very similar.

6/ I would choose decision tree. The reason is that it is possible for KNN to incorrectly classify the data points that are close to the line depending on which k value is used. Given enough data, decision tree can estimate the function of the line.

7/ An origin-centered circle (2D):

We will make the assumption that points inside or on the circle will have "plus" and outside will be "minus".

- The VC dimension is 1. The point can be in or out of the circle with different radius. For 2 points, there is no way to label the closer point minus when the farther point is already labeled plus

  An origin-centered circle (3D):
- The VC dimension is 1. Same argument as above for points in three dimensions.