Machine Learning *CS 4364/6364, Fall 2017*

HW4

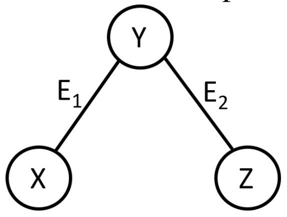
Instructions

Collaboration: Collaboration is allowed, to discuss the problems with other students. However you must write up your own solutions for the math questions, and implement your own solutions for the programming problems. Please list all collaborators and sources consulted, at the top of your homework.

Submission: Please submit homework pdf's via blackboard. Electronic submissions preferred. (Math can be formatted in Latex (some easy editors for Latex are Lyx, TexShop, Texmaker), Word, or other editors). Written homework can be scanned or submitted in class. All code should be submitted electronically. You may code in your language of choice. All homeworks are due at 11:59 pm on Monday 12/11. 20% penalty for late homeworks. No homeworks accepted after one week after the due date.

Questions

1. **Conditional Independence in Markov Random Fields (MRFs)**
In this question, we will verify the conditional independence property of a simple MRF:



Recall from lecture that two nodes (variables) X and Z are conditionally independent given Y if Y acts as a separating set between X and Z. In our simple MRF above, Y is clearly a separating set between X and Z, so we will use the functional form of the joint probability to verify the conditional independence $X \perp\!\!\!\perp Z|Y$.

Assume that the potential factor corresponding to edge E_1 is $\phi_1(X, Y)$ and the potential factor corresponding to edge E_2 is $\phi_2(Y, Z)$.

Provide an expression for the joint probability function in terms of these ϕ functions, and show that when we condition on a value of Y, we can establish the (conditional) independence of X and Z.

Hint: Consider what happens to the functional form of a potential factor when we condition on one of its arguments. If we have a potential function $\phi(A, B)$ and we condition on a value *of B, the resulting potential factor will now only be a function of A: $\phi'(A)$*

2. **Gradients in Feed-Forward Neural Networks**
   In this question, we will calculate the gradient of the error function at an output neuron in a feed-forward network. While of the networks we saw in lecture used the Sigmoid function as the activation function, we will use a basic cubic function as the activation function. Since we are only considering a single output neuron in this problem, we will use the following simplified notation:

   - Error function:
     $$E = \frac{1}{2}(t-o)^2$$
     Where t is the provided target output (label).
   - Output from the neuron:
     $$o = S^3$$
     Where S is is the weighted sum (defined below). Note that this is different than the Sigmoid function we used in lecture.
   - Weighted sum for the neuron:
     $$S = \sum_{i=1}^{n} w_i x_i$$
     Where w_i is the weight for x_i, and x_i is one of the inputs from the previous layer in the network.
     Now calculate
     $$\frac{\partial E}{\partial w_i}$$
     which is the i-th component of the gradient.

3. **Classification with "Naive Bayes" generative model**
   You are given the files `spectTrainData.csv`, `spectTrainLabels.csv`, `spectTestData.csv`, and `spectTestLabels.csv`. These were created from the medical data of on cardiac Single Proton Emission Tomography (SPECT) images of patients and each patient is classified into two categories: normal or abnormal. The database of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. The pattern was further processed to obtain a training set of 187 patterns and a test set of 80 patterns, each with 22 binary feature. The goal is to build a generative model of each group (normal and abnormal), and to use these models to classify future patients.

   (a) For each of the groups, use the training data to build a simple probabilistic model, assuming that the different features are independent. The model for a group should have 22 parameters $p_i \in [0,1]$; the probability of a particular data point $x \in \{0,1\}^{22}$ is then
   $$\prod_{i=1}^{22} p_i^{x_i}(1-p_i)^{1-x_i}$$

A natural choice is to set p_i to the proportion of training documents (from that particular group) for which $x_i = 1$. In practice, this can be dangerous - when these are lots of features, and any given feature is 1 only a tiny fraction of the time, there often isn't enough data to reliably estimate all the p_i in this way. Therefore, it is common to smooth the estimates somewhat, by setting:

$$p_i = \frac{(\text{number of points with } x_i = 1) + n\tilde{p}}{(\text{number of points}) + n} \tag{1}$$

where n is a small integer and $\tilde{p}$ is a prior estimate of the value of p_i. To keep this simple, use $n = 2$ and $\tilde{p} = 0.5$.

(b) Now implement the `Naive Bayes` model to classify the test documents. Recall that `Naive Bayes` assumes that the conditional probability of the features are independent given the label variable.

(c) Find the error rate of your Naive Bayes algorithm on test set.

4. **[Graduate Students Only] Boosting**

In this problem, we slightly modify the AdaBoost algorithm to better explore some properties of the algorithm. Specifically, we no longer normalize the weights on the training examples after each iteration. The modified algorithm, which is set to run for T iterations, is shown in Algorithm 1.

Note that in the modified version, the weights associated with the training examples are no longer guaranteed to sum to one after each iteration (and therefore can not be viewed as a "distribution"), but the algorithm is still valid. Let us denote the sum of weights at the start of iteration t by $Z_t = \sum_{i=1}^{n} w(t)$. At the start of the first iteration of boosting, $Z_1 = n$. Let us now investigate the behavior of Z_t, as a function of t.

(a) At the t^{th} iteration, we found a weak classifier that achieves a weighted training error ϵ_t. Show that the choice, $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$ is optimal in the sense that it minimizes Z_{t+1}.
Hint : Look at Z_{t+1} as a function of α and find the value of α for which the function achieves its minimum. You may also find the following notational shorthand useful:

$$W_l = \sum_{i=1}^{n} w_i^{(t)}(1 - \delta(y_i, h_t(x_i)))$$

$$W_c = \sum_{i=1}^{n} w_i^{(t)}(\delta(y_i, h_t(x_i)))$$

where, W_c is the total weight of the points classified correctly by h_t and W_l is the total weight of the misclassified points. $\delta(y, h_t(x)) = 1$ whenever the label predicted by h_t is correct and zero otherwise. The weights here are those available at the start of iteration t.

(b) Show that the sum of weights Z_t is monotonically decreasing as a function of t.

[*An interesting side note :* Why is Z_t so interesting? It is useful in bounding how successive boosting iterations reduce the training error. Specifically, it can be shown that the average number of misclassified training examples of the combined classifier after m iterations of boosting is bounded from above by Z_{m+1}/n. You are not required to prove this here.]

Algorithm 1 AdaBoost: slightly modified version (the weights are not normalized to sum to one)

```
**Input:** Set of $n$ labeled examples $(x_1, y_1), ..., (x_n, y_n), y_i = \pm 1$
Set of associate weights $w_1^{(1)}, ...., w_n^{(1)}$, initially all $w_i^{(1)} = 1$
Required number of iterations $T$.
**Procedure:**
**for** $t = 1..T$ **do**
    Find a weak classifier $h_t$, which outputs binary predictions $h_t(x) = \pm 1$, such that its weighted training error
        $$\epsilon_t = \frac{1}{Z_t} \sum_{i=1}^{n} w_i^{(t)} (1 - \delta(h_t(x_i), y_i))$$
    satisfies, $\epsilon_t < 1/2$
    Set the vote $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
    Update the weights :
    **for** $i = 1...n$ **do**
        $w_i^{(t+1)} = w_i^{(t)} \exp(-y_i \alpha_t h_t(x_i))$
    **end for**
**end for**
**Output:** The combined classifier defined by
        $$\hat{h}_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$
```