CS 4700: Foundations of Artificial Intelligence **(Due: 5/12/20, 11:59pm EDT)**

Homework #5: Machine Learning

76 Points Total

Instructor: Haym Hirsh *Name:* Student name, *Netid:* NetId

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission. **These can be specific to each assignment.**

- Please include your name and NetIDs on the first page. We recommend typesetting your submission in LaTeX, and an Overleaf template is here https://www.overleaf.com/read/dzytbjjnhwkw . See course policy for general typesetting requirements.
- Homeworks must be submitted via Gradescope by the due date and time.
- Late homeworks are accepted until **5/14/20 at 11:59pm EDT** for a 50% penalty per course policy.
- All sources of material outside the course must be cited. The University Academic Code of Conduct will be strictly enforced.

Problem 1: Naive Bayes Classifier for Text Classification (22 points)

An application of the Naive Bayes Classifier is for text classification. In this problem you will be classifying the statements, **"a very close race"** and **"a very close election"**, given the categories *Politics* and *Not Politics*. Here is the training set

Statement	Category
basketball is a great game to play	Not Politics
the election is over	Politics
very clean debate	Politics
a close but forgettable race	Not Politics
the election is a race	Politics

Given this training set, the goal of the Naive Bayes Classifier would be to compute $P(Politics \mid \textbf{a very close race})$ and $P(Not\ Politics \mid \textbf{a very close race})$ and classify the statement as the category with the higher probability. One way that this is frequently done is to create word features and assume that the occurrence of each word is an independent event. More specifically, $P(\textbf{a very close race}) = P(\textbf{a})P(\textbf{very})P(\textbf{close})P(\textbf{race})$.

A (4 points): Now given the two statements we would like to classify and the two categories, what are the following probabilities according to Bayes' Theorem? Please make sure to decompose the joint distribution with the conditional independence assumption.

1. $P(Politics \mid \textbf{a very close race}) =$

   [Your Solution Here]

2. $P(Not\ Politics \mid \textbf{a very close election}) =$

   [Your Solution Here]

B (6 points): A very natural way of engineering features for words is word frequency. Let us define it in the following way

$$tf(w, c) = \text{number of times word, } w\text{, appeared for category, } c$$

A more complicated feature representation is called Term Frequency - Inverse Document frequency (tf-idf). Let us define tf-idf in the following way

$$tfidf(w, c) = tf(w, c) \cdot \ln\left(\frac{N}{n(w)}\right)$$

where N is the number of training examples, and $n(w)$ is the number of training examples containing word w. Using this feature, we can then obtain the likelihood like so

$$P(w|c) = \frac{tfidf(w,c)}{N(c)}$$

where $N(c)$ is the number of words in the training set for category c (i.e. $N(Not\ Politics) = 12$ for our example).

Now given this formulation, do we need to use Laplace Smoothing? Please give a specific reason for why/why not Laplace smoothing would help us classify the statements above given our training set. What would the likelihood be with Laplace smoothing?

[Your Solution Here]

C (12 points): Please classify **"a very close race"** and **"a very close election"** according to the likelihood with Laplace smoothing. Please report the resulting probabilities from running the Naive Bayes algorithm (both P(*Politics*)P(statement|*Politics*) and P(*Not Politics*)P(statement|*Not Politics*)).

[Your Solution Here]

Problem 2: Multi-layer neural network and back-propagation (30 points)

The Figure 1 below shows an example of a multi-layer neural network, with 1 input layer (2 input units x_1, x_2), h_3 and h_4 are hidden units, and 1 output unit h_5. w_{ij} means the weight from unit i to unit j, w_{0j} means the bias, and a_i is the activation function.

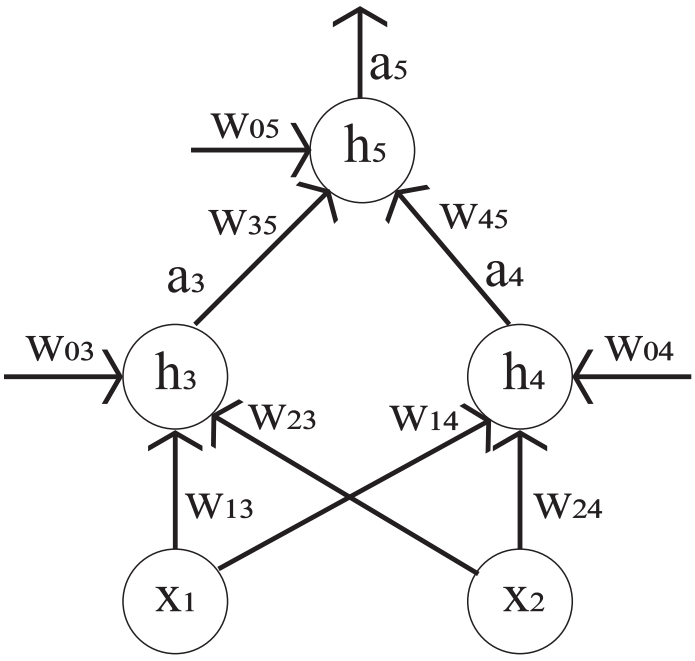

Figure 1: Multi-layer neural network architecture.

When an input is passed into a neural network, the signals are passed along the network by following the connections between the units, starting at the input layer and ending in the output layer. This process is referred to as **forward propagation**. In order for the neural network to learn, it must update its weights across its multiple layers. The common approach for neural network learning is called **back-propagation**, which relies on the gradient descent method.

A (4 points): Imagine you are using backprop to do weight updates for an example for which $x_1 = 1$ and $x_2 = -1$. Use Δw_{ij} to refer to the amount weight w_{ij} changes as the result of the update. If $\Delta w_{03} = d_3$ and $\Delta w_{04} = d_4$, what if anything can we say about Δw_{13}, Δw_{23}, Δw_{14}, and Δw_{24}?

[Your Solution Here]

B (13 points): This question will have you simulate the process of forward propagation and back-propagation on an example for which $x_1 = 1$ and $x_2 = -1$ and whose desired output is $y = 1$. The values for the various weights are as follows:

- Weight and bias for $h3$: $w_{03} = -1, w_{13} = 1, w_{23} = -1$
- Weight and bias for $h4$: $w_{04} = 2, w_{14} = -1, w_{24} = 1$
- Weight and bias for $h5$: $w_{05} = -2, w_{35} = 1, w_{45} = 1$

For the rest of this question you'll be using the algorithm for forward and back-propagation given in class. Assume **the learning rate α is 1**.

What is the output and error after the forward propagation, and what is the gradient and updated value after the back-propagation for each weight with the logistic activation function $g(z) = \frac{1}{1+e^{-z}}$. Recall that $g'(z) = g(z)(1 - g(z))$. Please also give the network's new output and error after the update. Please give your answer to three significant digits after the decimal: 0.05549 should be written 0.055, and 1.9995 should be written 2.000.

[Your Solution Here]

C (13 points): Please do the same for the activation function $g(z) = \tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$. Recall that $g'(z) = 1 - g^2(z)$.

[Your Solution Here]

Problem 3: Multi-class logistic regression (24 points)

In class we covered logistic regression for binary classification, when there are only two labels. This question considers the case where we have more than two classes. The approach that we will take — known as Maximum Likelihood Estimation (MLE) — comes from thinking of the value that the logistic function gives — between 0 and 1 — as a probability. More precisely, given a particular $\bar{x}$ a particular set of weights $\bar{w}$ assigns a probability that the label is 1 as $P(y = 1|\bar{x}) = \frac{1}{1+e^{-\bar{w}\cdot\bar{x}}}$. The probability that it is 0 is 1 minus this, or $P(y = 0|\bar{x}) = 1 - \frac{1}{1+e^{-\bar{w}\cdot\bar{x}}}$. We would then assign as the label either 1 or 0 to $\bar{x}$ based on whichever of these two values is greater, namely we assign the category

$$argmax_{c\in\{0,1\}} P(y = c|\bar{x})$$

where the probabilities are computed using the logistic function as just described. (Strictly speaking we should be writing the probabilities as $P(y = 1|\bar{x}, \bar{w})$ so that it is conditioned on the $\bar{w}$ that we are considering. Since it will be clear from context that everything is defined by a particular $\bar{w}$ we'll omit it from the probability to keep the notation a bit tidier.)

During learning we have a set of N examples, $D = \{(\bar{x}_1, y_1), \ldots, (\bar{x}_N, y_N)\}$, and doing well on the training data according to this probabilistic approach means trying to get these probabilities right for the training data.

If we assume the label for each example is independent of the labels for other examples, the probability of seeing all the labels that we've gotten is

$$L_D(\bar{w}) = \prod_{i=1}^{N} P(Y = y_i | X = \bar{x}_i)$$

namely for each example we have the probability that the label Y is the label in the training data given that the example X is $\bar{x}_i$ — and since each label is independent we can just multiply them together to get the overall probability for the full data set. Notice that $\bar{w}$ doesn't appear on the right-hand side, but it's implicitly there in our use of the logistic function to define the probabilities (it's omitted so that the formulation doesn't become too cluttered). You can think of this as roughly analogous to the error function that we used to derive logistic regression in class. Here we again have a function of the data that is defined by a particular $\bar{w}$, but we have a different function that we want to maximize rather than minimize.

In order to do this maximization we will need to explicitly express the product of the probabilities using the logistic function, thereby exposing the $\bar{w}$ that defines the value. There is a notational trick that lets us do this fairly concisely. Notice that each y_i is either 0 or 1. Consider the following formula:

$$(\frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})^{y_i}(1 - \frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})^{(1-y_i)}$$

This is the product of two terms. If $y_i = 0$ the exponent for the first term is 0 and for the second term it is 1, yielding $1 - \frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}}$, which is precisely the probability formula that we want if $y_i = 0$. If instead $y_i = 1$ something similar happens, leaving you with $\frac{1}{1+e^{-\bar{w}\cdot\bar{x}}_i}$, which is the correct formula for the 1 label. In other words, regardless of whether $y_i = 0$ or $y_i = 1$ this expression gives you the probability calculation for that particular label. You can think of it as a case statement implemented by clever use of exponents.

With this in hand we can write down $L_D(\bar{w})$ using the logistic function to estimate the probabilities in the following fashion:

$$L_D(\bar{w}) = \prod_{i=1}^{N}[(\frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})^{y_i}(1 - \frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})^{(1-y_i)}]$$

For each example we use the notational trick to state that example's probability without having to write each one as "If $y_i = 0$ then use one particular formula and if $y_i = 1$ then use a different formula." We multiply those numbers together for all the data. $L_D(\bar{w})$ is called the "likelihood function". We can now state the maximization task that we face: we want a $\bar{w}$ that maximizes the probabilities, or in other words we want $argmax_{\bar{w}} L_D(\bar{w})$.

Notice that if $a > b$ then $\ln(a) > \ln(b)$. In other words if I'm doing an argmax over a set of choices I get the same result if I do the argmax over the ln of those choices. Let's give it a try on our likelihood function — to create what is known as "log-likelihood":

$$LL_D(\bar{w}) = \ln(L_D(\bar{w})) = \ln \prod_{i=1}^{N}[(\frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})^{y_i}(1 - \frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})^{(1-y_i)}]$$

$$= \sum_{i=1}^{N}[y_i \ln(\frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}}) + (1-y_i)\ln(1 - \frac{1}{1+e^{-\bar{w}\cdot\bar{x}_i}})]$$

$$= \sum_{i=1}^{N}[y_i \ln(\frac{e^{\bar{w}\cdot\bar{x}_i}}{e^{\bar{w}\cdot\bar{x}_i}+1}) + (1-y_i)\ln(\frac{1}{e^{\bar{w}\cdot\bar{x}_i}+1})]$$

$$= \sum_{i=1}^{N}[y_i(\bar{w}\cdot\bar{x}_i) - \ln(e^{\bar{w}\cdot\bar{x}_i}+1)]$$

If we're planning to use gradient ascent on this function, let's take the derivative of $LL_D(\bar{w})$ with respect to a weight w_j:

$$\frac{\partial}{\partial w_j} LL_D(\bar{w}) = \frac{\partial}{\partial w_j} \sum_{i=1}^{N}[y_i(\bar{w}\cdot\bar{x}_i) - \ln(e^{\bar{w}\cdot\bar{x}_i}+1)]$$

$$= \sum_{i=1}^{N} \frac{\partial}{\partial w_j} y_i(\bar{w}\cdot\bar{x}_i) - \sum_{i=1}^{N} \frac{\partial}{\partial w_j} \ln(e^{\bar{w}\cdot\bar{x}_i}+1)$$

$$= \sum_{i=1}^{N} [y_i x_{ij} - x_{ij} \frac{e^{\bar{w} \cdot \bar{x}_i}}{e^{\bar{w} \cdot \bar{x}_i} + 1}] = \sum_{i=1}^{N} x_{ij} [y_i - \frac{1}{1 + e^{-\bar{w} \cdot \bar{x}_i}}]$$

Look at the last term carefully. A single example $\bar{x}_i$ contributes to the derivative of the log-likelihood an amount equal to $\bar{x}_i(y_i - \frac{1}{1+e^{-\bar{w} \cdot \bar{x}_i}})$. The term $\frac{1}{1+e^{-\bar{w} \cdot \bar{x}_i}}$ is the value of the logistic function for example $\bar{x}_i$, and subtracting it from y_i is how far off it is from the desired label. In other words it's exactly what we got when we derived the logistic regression update rule in class using sum-of-squares error!

We can now finally turn to the multi-class case. To do this notice that in the derivation above we rewrote $\frac{1}{1+e^{-z}}$ as $\frac{e^z}{e^z+1}$ and $1 - \frac{1}{1+e^{-z}}$ as $\frac{1}{e^z+1}$. The numerator of one is e^z and the other is 1, and they're both divided by the sum of these two numbers. We're going to do something that will look similar for the multi-class case.

We will use k to refer to the number of classes that our problem has, and each class will be assigned to a distinct integer between 1 and k. Further, rather than having a single set of weights, we're going to assign each class c its own weight vector $\bar{w}_c$, resulting in k weight vectors $\bar{w}_1, \ldots, \bar{w}_k$. Each of these $\bar{w}_c$ will be used to assign a probability that an example $\bar{x}$ has the given label c as:

$$P(y = c|\bar{x}) = \frac{e^{\bar{w}_c \cdot \bar{x}}}{\sum_{i=1}^{k} e^{\bar{w}_i \cdot \bar{x}}}$$

You can think of this as each class c is assigned a score $e^{\bar{w}_c \cdot \bar{x}}$ (the numerator of this term), and then we divide each by the sum of all k scores so that the numbers are valid probabilities that sum to one. Whichever c has the greatest probability is the label we would assign to $\bar{x}$.

Finally, to use the notational trick that used y_i and $(1 - y_i)$ as exponents instead of thinking of the label for some y_i as $y_i = c$ where c is the integer representing the correct label for that example, we're going to create k variables, $y_{i1}, \ldots y_{ik}$. If the correct label is c for a given example $\bar{x}_i$ we would have $y_{ic} = 1$ and all other $y_{ij} = 0$. As a result the label for each $\bar{x}_i$ will now be a vector $\bar{y}_i$ where the y vector is over k dimensions and for any example only one of the dimensions is 1 and the rest are 0.

Using Maximum Likelihood Estimation for this multi-class formulation of logistic regression means we're seeking a set of weights $\bar{w}_1, \ldots, \bar{w}_k$ that maximize the log-likelihood function in fashion akin to what we just derived above. Doing this requires two steps: (1) write the log-likelihood function and (2) find the $\bar{w}_1, \ldots, \bar{w}_k$ that maximize it.

A (12 points): Derive the log-likelihood function $LL_D(\bar{w}_1, \ldots, \bar{w}_k)$ for the multi-class logistic regression model. Your derivation should roughly follow the derivation given above for the two-class case. Use w_{ic} to refer to the i-th weight of the c-th weight vector. (Hint: Instead of there being just two terms in the sum there will be k terms, so you will need a second summation across classes within the outer summation for examples.)

[Your Solution Here]

B (12 points): Generate the derivative of $LL_D(\bar{w}_1, \ldots, \bar{w}_k)$ with respect to a weight w_{cj}. where j is the j-th feature in the input vector.

[Your Solution Here]