COMS 4771 HW2

Due: Sat Mar 03, 2018 at 11:59pm

You are allowed to work in groups of (at max) three students. These group members don't necessarily have to be the same from previous homeworks. Only one submission per group is required by the due date on Gradescope. Name and UNI of all group members must be clearly specified on the homework. You must cite all the references you used to do this homework. You must show your work to receive full credit.

1 **[Multi-class classification]** In class, we have primarily been focusing on binary classifiers. In this problem, we will study how one can use binary classification to do multiclass classification. Suppose we want to construct a k-class classifier f that maps data from some input space $\mathcal{X}$ to the label space $\{1, 2, ..., k\}$. There are two popular methods to construct f from just combining results from multiple binary classifiers.

- one-vs-rest (OvR) technique – for each class $i \in [k]$, one can view the classification problem as computing a function $f_i : \mathbb{R}^d \to \{\text{class } i, \text{not class } i\}$ (i.e., assigning examples from class i the label 1 and all other classes the label 0). One can combine the results for each f_i to construct a multiclass classifier f.
- one-vs-one (OvO) technique – For each pair of classes $i, j \in [k]$ (i, j distinct), one can view the classification problem as computing a function $f_{ij} : \mathbb{R}^d \to \{\text{class } i, \text{class } j\}$ (taking only training points with labels i or j). One can combine the results for each f_{ij} to construct a multiclass classifier f.

(i) Describe how one can design a multiclass classifier f from OvR and OvO techniques. Make sure the precisely (mathematically) define f for each technique. For a new test example, how many calls to binary classification are made in each technique?

(ii) Assuming that your base binary classifiers can only be linear, show a training dataset for each of the following cases. (Your example training dataset for each case must have the following properties - (i) number of classes in the dataset $k > 2$, (ii) dataset contains equal number of datapoints per class, and (iii) each class contains at least two datapoints.)

- OvR gives better accuracy over OvO
- OvO gives better accuracy over OvR
- For any $\epsilon > 0$, both OvO and OvR give accuracy of at most ϵ. (your example training set can depend on ϵ)
- For any $\epsilon > 0$, both OvO and OvR give accuracy of at least $1 - \epsilon$. (your example training set can depend on ϵ)

(iii) From part (i) we observed that OvO/OvR techniques made certain number of calls to binary classification during test time. Suppose our goal is to minimize the number of calls made to binary classification during test time (let's call this quantity c). Propose a technique to construct a k-class classifier f from binary classifiers that minimizes c. For your proposed technique, what is c? (i.e., express it in terms of parameters of the data, such as, number of classes k, number of datapoints n, dimensionality of your dataset d, etc). Prove that your technique is indeed minimizes c, that is, there is no other technique that makes fewer binary classification calls than your technique during test time and still achieve comparable accuracy.

2 **[Constrained optimization]** Show that the distance from the hyperplane $g(x) = w \cdot x + w_0 = 0$ to a point x_a is $|g(x_a)|/\|w\|$ by minimizing the squared distance $\|x - x_a\|^2$ subject to the constraint $g(x) = 0$.

3 **[A better output Perceptron algorithm guarantee]** In class, we saw that when the training sample S is linearly separable with a maximum margin $\gamma > 0$, then the Perceptron algorithm run cyclically over S is guaranteed to converge after $T \leq \big(R/\gamma\big)^2$ updates, where R is the radius of the sphere containing the sample points. This does not guarantee however that the hyperplane solution returned by Perceptron, i.e. w_T achieves a margin close to γ.

(i) Show an example training dataset S in $\mathbb{R}^2$ that has margin γ, and an order of updates made by the Perceptron algorithm where the hyperplane solution returned has arbitrarily bad margin on S.

(ii) Consider the following modification to the perceptron algorithm:

```
Modified Perceptron Algorithm
Input: training dataset $S = (x_i, y_i)_{i=1,\ldots,n}$
Output: learned vector $w$
- Initialize $w_0 := 0, t := 0$
- while there exists an example $(x, y) \in S$, such that $2y(w_t \cdot x) \leq \gamma\|w_t\|$
-    set $w_{t+1} := w_t + yx$
-    set $t := t + 1$
- return $w_t$.
```

(a) If the Modified Perceptron Algorithm (MPA) terminates after T rounds, what margin guarantee is achieved by the hyperplane w_T returned by MPA? Justify your answer.

(b) We will now prove step-by-step the mistake bound for the Modified Perceptron Algorithm (MPA) algorithm.

i. Show that after T rounds $T\gamma \leq \|w_T\|$, and observe that if $\|w_T\| < 4R^2/\gamma$, then $T < 4R^2/\gamma^2$.

In what follows, we will assume that $\|w_T\| \geq 4R^2/\gamma$.

ii. Show that for any iteration t when mistake was made, the following holds:

$$\|w_t\|^2 \leq (\|w_{t-1}\| + \gamma/2)^2 + R^2.$$

iii. Infer from that that for any iteration t, we have

$$\|w_t\| \leq \|w_{t-1}\| + \gamma/2 + \frac{R^2}{\|w_{t-1}\| + \|w_t\| + \gamma/2}.$$

iv. Using the previous question, show that for any iteration t such that either $\|w_{t-1}\| \geq \frac{4R^2}{\gamma}$ or $\|w_t\| \geq \frac{4R^2}{\gamma}$, we have

$$\|w_t\| \leq \|w_{t-1}\| + \frac{3}{4}\gamma.$$

v. Show that $\|w_0\| \leq R \leq 4R^2/\gamma$. Since by assumption we have $\|w_T\| \geq \frac{4R^2}{\gamma}$, conclude that there must exist some largest iteration t_0 such that $\|w_{t_0-1}\| \leq \frac{4R^2}{\gamma}$ and $\|w_{t_0}\| \geq \frac{4R^2}{\gamma}$.

vi. Show that $\|w_T\| \leq \|w_{t_0-1}\| + \frac{3}{4}T\gamma$, and finally deduce the mistake bound.

4 **[Making data linearly separable by feature space mapping]** Consider the infinite dimensional feature space mapping

$$\begin{aligned} \Phi_\sigma &: \mathbb{R} \to \mathbb{R}^\infty \\ x &\mapsto \Big(\mathbf{1}\big[|\alpha - x| < \sigma\big] \cdot \exp\big(-1/(1 - (|\alpha - x|/\sigma)^2)\big)\Big)_{\alpha \in \mathbb{R}}. \end{aligned}$$

(It may be helpful to sketch the function $f(\alpha) := \mathbf{1}\big[|\alpha| < 1\big] \cdot \exp\big(-1/(1-\alpha^2)\big)$ for understanding the mapping and answering the questions below)

(i) Show that for any n distinct points $x_1, \ldots, x_n$, there exists a $\sigma > 0$ such that the mapping Φ_σ can linearly separate *any* binary labeling of the n points.

(ii) Show that one can efficiently compute the dot products in this feature space, by giving an analytical formula for $\Phi_\sigma(x) \cdot \Phi_\sigma(x')$ for arbitrary points x and x'.

(iii) Given an input space X and a feature space mapping ϕ that maps elements from X to a (possibly infinite dimensional) inner product space V. Let $K : X \times X \to \mathbb{R}$ be a kernel function that can efficiently compute the inner products in V, that is, for any $x, x' \in X$, $K(x, x') = \phi(x) \cdot \phi(x')$.

Consider a binary classification algorithm that predicts the label of an unseen instance according to the class with the closest average. Formally, given a training set $S = (x_1, y_1), \ldots, (x_m, y_m)$, for each $y \in \{\pm 1\}$ define

$$c_y := \frac{1}{m_y} \sum_{i : y_i = y} \phi(x_i),$$

where $m_y = |\{i : y_i = y\}|$. Assume that m_+ and m_- are nonzero. Then, the algorithm outputs the following decision rule:

$$h(x) := \begin{cases} 1 & \|\phi(x) - c_+\| \leq \|\phi(x) - c_-\| \\ 0 & \text{otherwise.} \end{cases}$$

(a) Let $w := c_+ - c_-$ and let $b = \frac{1}{2}(\|c_-\|^2 - \|c_+\|^2)$. Show that

$$h(x) = \text{sign}(\langle w, \phi(x) \rangle + b).$$

(b) Show that $h(x)$ can be expressed via $K(\cdot, \cdot)$, without accessing individual entries of $\phi(x)$ or w, thus showing that $h(x)$ is efficiently computable.

5 **[Predicting Restaurant-reviews via Perceptrons]**

Download the review dataset `hw2data_1.zip` from the course website. This data set is comprised of reviews of restaurants in Pittsburgh; the label indicates whether or not the reviewer-assigned rating is at least four (on a five-point scale). The data are in CSV format (where the first line is the header); the first column is the label (`label`; 0 or 1), and the second column is the review text (`text`). The text has been processed to remove non-alphanumeric symbols and capitalization. The data has already been separated into training data `reviews_tr.csv` and test data and `reviews_te.csv`.

The first challenge in dealing with text data is to come up with a reasonable "vectorial" representation of the data so that one can use standard machine learning classifiers with it.

Data-representations

In this problem, you will experiment with the following different data representations.

1. Unigram representation.

   In this representation, there is a feature for every word t, and the feature value associated with a word t in a document d is

   $$\text{tf}(t; d) := \text{number of times word } t \text{ appears in document } d.$$

   (tf is short for *term frequency*.)

2. *Term frequency-inverse document frequency (tf-idf)* weighting.

   This is like the unigram representation, except the feature associated with a word t in a document d from a collection of documents D (e.g., training data) is

   $$\text{tf}(t; d) \times \log_{10}(\text{idf}(t; D)),$$

   where $\text{tf}(t; d)$ is as defined above, and

   $$\text{idf}(t; D) := \frac{|D|}{\text{number of documents in } D \text{ that contain word } t}.$$

   This representation puts more emphasis on rare words and less emphasis on common words. (There are many variants of tf-idf that are unfortunately all referred to by the same name.)

   Note: When you apply this representation to a new document (e.g., a document in the test set), you should still use the idf defined with respect to D. This, however, becomes problematic if a word t appears in a new document but did not appear in any document in D: in this case, $\text{idf}(t; D) = |D|/0 = \infty$. It is not obvious what should be done in these cases. For this homework assignment, simply ignore words t that do not appear in any document in D.

3. Bigram representation.

   In addition to the unigram features, there is a feature for every *pair* of words (t_1, t_2) (called a *bigram*), and the feature value associated with a bigram (t_1, t_2) in a given document d is

   $\mathrm{tf}((t_1, t_2); d) :=$ number of times bigram (t_1, t_2) appears consecutively in document d.

   In the sequence of words "a rose is a rose", the bigrams that appear are: (a, rose), which appears twice; (rose, is); and (is, a).

For all the of these representations, ensure to do data "lifting".

Online Perceptron with online-to-batch conversion

Once can implement the Online Perceptron algorithm with the following online-to-batch conversion process.

- Run Online Perceptron to make *two* passes through the training data. Before each pass, randomly shuffle the order of the training examples. Note that a total of $2n + 1$ linear classifiers $\hat{w}_1, \ldots, \hat{w}_{2n+1}$ are created during the run of the algorithm, where n is the number of training examples.
- Return the linear classifier $\hat{w}_{\text{final}}$ given by the simple average of the final $n + 1$ linear classifiers:

$$\hat{w}_{\text{final}} := \frac{1}{n+1} \sum_{i=n+1}^{2n+1} \hat{w}_i .$$

Recall that as Online Perceptron is making its pass through the training examples, it only updates its weight vector in rounds in which it makes a prediction error. So, for example, if $\hat{w}_1$ does not make a mistake in classifying the first training example, then $\hat{w}_2 = \hat{w}_1$. You can use this fact to keep the memory usage of your code relatively modest.

(i) What are some potential issued with Unigram representation of textual data?

(ii) Implement the online Perceptron algorithm with online-to-batch conversion. You must submit your code via Courseworks to receive full credit.

(iii) Which of the three representations give better predictive performance? As always, you should justify your answer with appropriate performance graphs demonstrating the superiority of one representation over the other. Example things to consider: you should evaluate how the classifier behaves on a holdout 'test' sample for various splits of the data; how does the training sample size affects the classification performance; etc.

(iv) For the classifier based on the unigram representation, determine the 10 words that have the highest (i.e., most positive) weights; also determine the 10 words that have the lowest (i.e., most negative) weights.

6 **[Neural networks as universal function approximators]** Here we will experimentally verify that (feed-forward) neural networks can approximate any smooth function. Recall that a feed-forward neural network is simply a combination of multiple 'neurons' such that the output of one neuron is fed as the input to another neuron. More precisely, a neuron ν_i is a computational unit that takes in an input vector $\vec{x}$ and returns a wighted combination of the inputs (plus the bias associated with neuron ν_i) passed through an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, that is: $\nu_i(\vec{x}; \vec{w}_i, b_i) := \sigma(\vec{w}_i \cdot \vec{x} + b_i)$. With this notation, we can define a layer ℓ of a neural network $\mathcal{N}^\ell$ that takes in an I-dimensional input and returns a O-dimensional output as

$$\begin{aligned} \mathcal{N}^\ell(x) &:= \big(\nu_1^\ell(\vec{x}), \nu_2^\ell(\vec{x}), \cdots, \nu_O^\ell(\vec{x})\big) && x \in \mathbb{R}^I \\ &= \sigma(W_\ell^\mathsf{T} \vec{x} + \vec{b}_\ell) && W_\ell \in \mathbb{R}^{d_I \times d_O} \text{ defined as } [\vec{w}_1^\ell, \ldots, \vec{w}_O^\ell], \\ & && \text{and } \vec{b}_\ell \in \mathbb{R}^{d_O} \text{ defined as } [b_1^\ell, \ldots, b_O^\ell]^\mathsf{T}. \end{aligned}$$

Here $\vec{w}_i^\ell$ and b_i^ℓ refers to the weight and the bias associated with neuron ν_i^ℓ in layer ℓ, and the activation function σ is applied pointwise. An L-layer (feed-forward) neural network $\mathcal{F}_{L\text{-layer}}$ is then defined as a network consisting of network layers $\mathcal{N}^1, \ldots, \mathcal{N}^L$, where the input to layer i is the output of layer $i-1$. By convention, input to the first layer (layer 1) is the actual input data.

(i) Consider a nonlinear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ defined as $x \mapsto 1/(1+e^{-x})$. Show that $\frac{\partial \sigma}{\partial x} = \sigma(x)(1-\sigma(x))$.

(ii) Consider a *single layer* feed forward neural network that takes a d_I-dimensional input and returns a d_O-dimensional output, defined as $\mathcal{F}_{1\text{-layer}}(x) := \sigma(W^T x + b)$ for some $d_I \times d_O$ weight matrix W and a $d_O \times 1$ vector b. Given a training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, we can define the *average error* (with respect to the network parameters) of predicting y_i from input example x_i as:

$$E(W, b) := \frac{1}{2n} \sum_{i=1}^{n} \|\mathcal{F}_{1\text{-layer}}(x_i) - y_i\|^2.$$

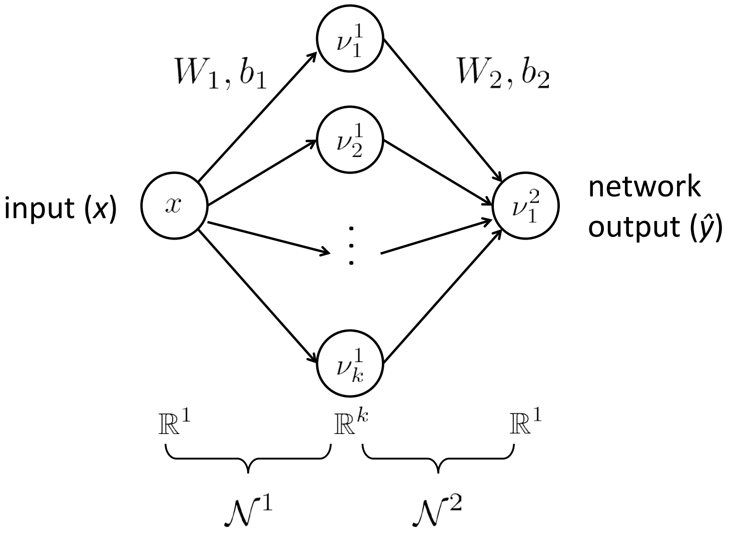
What is $\frac{\partial E}{\partial W}$, and $\frac{\partial E}{\partial b}$?

(note: we can use this gradient in a descent-type procedure to minimize this error and learn a good setting of the weight matrix that can predict y_i from x_i.)

(iii) Although reasonably expressive, single layer neural networks are not flexible enough to approximate arbitrary smooth functions. Here we will focus on approximating one-dimensional functions $f : \mathbb{R} \to [0, 1]$ (the range of the function is taken in the interval $[0, 1]$ only for convenience, one can transform any bounded function to this interval by simple scaling and translating) with a *multi-layer* neural network. Consider a L-layer neural network $\mathcal{F}_{L\text{-layer}}$ as described above. Given a sample of input-output pairs $(x_1, y_1), \ldots, (x_n, y_n)$ generated from an unknown fixed function f, one can approximate f from $\mathcal{F}_{L\text{-layer}}$ by learning the appropriate parameters by minimizing the following error function.

$$\begin{aligned} E(W_1, b_1, \ldots, W_L, b_L) &:= \frac{1}{2n} \sum_{i=1}^{n} \|\mathcal{F}_{L\text{-layer}}(x_i) - y_i\|^2 \\ &= \frac{1}{2n} \sum_{i=1}^{n} \|\mathcal{N}^L \circ \cdots \circ \mathcal{N}^1(x_i) - y_i\|^2. \end{aligned}$$

Assuming that our input data is one dimensional, that is each $x_i \in \mathbb{R}$ and $y_i \in [0,1]$, implement a gradient descent procedure to learn the parameters of a two-layer (ie, $L = 2$) feed forward neural network. Note that since each y_i is 1-dimensional, layer $\mathcal{N}^2$ only contains one neuron. Layer $\mathcal{N}^1$ can contain an arbitrary number, say k, neurons. Graphically the network you are implementing looks as follows.



Here is the pseudocode of your implementation.

Learn 2-layer neural network for 1-d functions

```
input: data $(x_1, y_1), \ldots, (x_n, y_n)$,
.        size of the intermediate layer $k$
- Initialize weight parameters $(W_1, b_1), (W_2, b_2)$ randomly
- Repeat until convergence:
-    for each training example $(x_i, y_i)$
-        compute the network output $\hat{y}_i$ on $x_i$
-    compute gradients $\frac{\partial E}{\partial W_2}, \frac{\partial E}{\partial b_2}, \frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial b_1}$
-        update weight parameters:
-            $W_2^{\text{new}} := W_2 - \eta\frac{\partial E}{\partial W_2}$,    $b_2^{\text{new}} := b_2 - \eta\frac{\partial E}{\partial b_2}$
-            $W_1^{\text{new}} := W_1 - \eta\frac{\partial E}{\partial W_1}$,    $b_1^{\text{new}} := b_1 - \eta\frac{\partial E}{\partial b_1}$
```

You must submit your code on Courseworks to receive full credit.

(iv) Download hw2data_2.mat from the course website. It contains two vector valued variables X and Y. Each row in Y is a noisy realization of applying an unknown smooth 1-dimensional function to the corresponding row in X. You can visualize the relationship between these variables by plotting X on the x-axis and Y on the y-axis.

Use your implementation from part (iii) to learn the unknown mapping function which can yield Y from X. Once the network parameters are learned, plot the network output on the y-axis (in red) along with the given Y values (in blue) for each input value X on the x-axis.