CIS 520, Machine Learning, Fall 2018: Assignment 5
Due: Sunday, October 28th, 11:59pm

Instructions. Please write up your responses to the following problems clearly and concisely. We encourage you to write up your responses using LaTeX; we have provided a LaTeX template, available on Canvas, to make this easier. **Submit your answers in PDF form to Gradescope. We will not accept paper copies of the homework.**

Collaboration. You are allowed and encouraged to work together. You may discuss the homework to understand the problem and reach a solution in groups up to size **two students.** However, *each student must write down the solution independently, and without referring to written notes from the joint session.* **In addition, each student must write on the problem set the names of the people with whom you collaborated.** You must understand the solution well enough in order to reconstruct it by yourself. (This is for your own benefit: you have to take the exams alone.)

Collaborators:

Type Collaborator Name Here

1 Multiclass Boosting [25 points]

In this problem you will analyze the AdaBoost.M1 algorithm, a multiclass extension of AdaBoost. Given a training sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$, where x_i are instances in some instance space $\mathcal{X}$ and y_i are multiclass labels that take values in $\{1, \ldots, K\}$, the algorithm maintains weights $D_t(i)$ over the examples (x_i, y_i) as in AdaBoost, and on round t, gives the weighted sample (S, D_t) to the weak learner. The weak learner returns a multiclass classifier $h_t : \mathcal{X} \to \{1, \ldots, K\}$ with weighted error less than $\frac{1}{2}$; here the weighted error of h_t is measured as

$$\mathrm{er}_t = \sum_{i=1}^{m} D_t(i) \cdot \mathbf{1}(h_t(x_i) \neq y_i)\,.$$

Note that the assumption on the weak classifiers is stronger here than in the binary case, since we require the weak classifiers to do more than simply improve upon random guessing (there are other multiclass boosting algorithms that allow for weaker classifiers; you will analyze the simplest case here). For convenience, we will encode the weak classifier h_t as $\widetilde{h}_t : \mathcal{X} \to \{\pm 1\}^K$, where

$$\widetilde{h}_{t,k}(x) = \begin{cases} +1 & \text{if } h_t(x) = k \\ -1 & \text{otherwise.} \end{cases}$$

In other words, $\widetilde{h}_t(x)$ is a K-dimensional vector that contains $+1$ in the position of the predicted class for x and -1 in all other $(K-1)$ positions. On each round, AdaBoost.M1 re-weights examples such that examples misclassified by the current weak classifier receive higher weight in the next round. At the end, the algorithm combines the weak classifiers h_t via a weighted majority vote to produce a final multiclass classifier H:

```
Algorithm AdaBoost.M1
Inputs: Training sample $S = ((x_1, y_1), \dots, (x_m, y_m)) \in (\mathcal{X} \times \{1, \dots, K\})^m$
        Number of iterations $T$
Initialize: $D_1(i) = \frac{1}{m} \;\; \forall i \in [m]$
For $t = 1, \dots, T$:
  – Train weak learner on weighted sample $(S, D_t)$; get weak classifier $h_t : \mathcal{X} \to \{1, \dots, K\}$
  – Set $\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1 - \mathrm{er}_t}{\mathrm{er}_t}\right)$
  – Update:
        $D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha_t \widetilde{h}_{t,y_i}(x_i))}{Z_t}$
    where $Z_t = \sum_{j=1}^m D_t(j) \exp(-\alpha_t \widetilde{h}_{t,y_j}(x_j))$
Output final hypothesis:
    $H(x) \in \arg\max_{k \in \{1,\dots,K\}} \underbrace{\sum_{t=1}^T \alpha_t \widetilde{h}_{t,k}(x)}_{F_{T,k}(x)}$
```

You will show, in five parts below, that if all the weak classifiers have error er_t at most $\frac{1}{2} - \gamma$, then after T rounds, the training error of the final classifier H, given by

$$\mathrm{er}_S[H] = \frac{1}{m} \sum_{i=1}^m \mathbf{1}(H(x_i) \neq y_i)\,,$$

is at most $e^{-2T\gamma^2}$ (which means that for large enough T, the final error $\mathrm{er}_S[H]$ can be made as small as desired).

(a) [5 points] Show that

$$D_{T+1}(i) \;=\; \frac{\frac{1}{m}\, e^{-F_{T,y_i}(x_i)}}{\prod_{t=1}^T Z_t}\,.$$

(b) [5 points] Show that

$$\mathbf{1}(H(x_i) \neq y_i) \;\leq\; \mathbf{1}\big(F_{T,y_i}(x_i) < 0\big)\,.$$

(*Hint:* Consider separately the two cases $H(x_i) \neq y_i$ and $H(x_i) = y_i$, and note that $\sum_{k=1}^K F_{T,k}(x_i) = -(K-2)\sum_{t=1}^T \alpha_t$.)

(c) [5 points] Show that

$$\mathrm{er}_S[H] \;\leq\; \frac{1}{m}\sum_{i=1}^m e^{-F_{T,y_i}(x_i)} \;=\; \prod_{t=1}^T Z_t\,.$$

(*Hint:* For the inequality, use the result of part (b) above, and the fact that $\mathbf{1}(u < 0) \leq e^{-u}$; for the equality, use the result of part (a) above.)

(d) [**5 points**] Show that for the given choice of α_t, we have

$$Z_t \;=\; 2\sqrt{\mathrm{er}_t(1-\mathrm{er}_t)}\,.$$

(e) [**5 points**] Suppose $\mathrm{er}_t \le \frac{1}{2} - \gamma$ for all t (where $0 < \gamma \le \frac{1}{2}$). Then show that

$$\mathrm{er}_S[H] \;\le\; e^{-2T\gamma^2}\,.$$

2 Perceptron vs. Winnow [25 points]

For online binary classification problems, you saw the Perceptron algorithm in class. Another algorithm that is used for such problems is the *Winnow* algorithm, which also maintains a linear classification model $\mathbf{w}_t$, but makes *multiplicative updates* to $\mathbf{w}_t$ rather than additive ones (such multiplicative updates now play an important role in many modern optimization algorithms). In this case, the weight vectors $\mathbf{w}_t$ always have positive entries that add up to 1:

```
Algorithm **Winnow**
**Learning rate parameter** $\eta > 0$
**Initial weight vector** $\mathbf{w}_1 = (\frac{1}{d}, \ldots, \frac{1}{d}) \in \mathbb{R}^d$
For $t = 1, \ldots, T$:
  – Receive instance $\mathbf{x}_t \in \mathbb{R}^d$
  – Predict $\widehat{y}_t = \mathrm{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$
  – Receive true label $y_t \in \{\pm 1\}$
  – Update: If $\widehat{y}_t \neq y_t$ then
      For each $i \in \{1, \ldots, d\}$:   $w_{t+1,i} \leftarrow \frac{w_{t,i}\exp(\eta\, y_t x_{t,i})}{Z_t}$
              where $Z_t = \sum_{j=1}^{n} w_{t,j}\exp(\eta\, y_t x_{t,j})$
    else
      $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t$
```

For examples that are linearly separable by a non-negative weight vector, the Winnow algorithm is known to have the following mistake bound:

Theorem (Winnow mistake bound). *Suppose that the examples seen in T trials are linearly separable by a non-negative weight vector, i.e. that there exists a weight vector $\mathbf{u} \in \mathbb{R}^d_+$ and $\gamma > 0$ such that*

$$y_t(\mathbf{u}^\top \mathbf{x}_t) \ge \gamma \;\; \textit{for all } t \in \{1, \ldots, T\}.$$

Also suppose $\|\mathbf{x}_t\|_\infty \le R_\infty$ for all t. If $\|\mathbf{u}\|_1$, γ, and R_∞ are known, then one can select the learning rate parameter η in a way that the number of mistakes in the T trials is at most

$$2\left(\frac{R_\infty^2 \|\mathbf{u}\|_1^2}{\gamma^2}\right)\ln(d)\,.$$

(a) Sparse target vector u, dense feature vectors $\mathbf{x}_t$. **[10 points]**
Suppose you are in a setting with high-dimensional features (large d), and that all features are of roughly constant magnitude; for simplicity, suppose $\mathbf{x}_t \in \{\pm 1\}^d$ for all t. Suppose you are told that the examples in

T trials are linearly separable by a sparse weight vector $\mathbf{u} \in \{0,1\}^d$ which has only $k \ll d$ non-zero entries, and that you are given $\gamma > 0$ such that $y_t(\mathbf{u}^\top \mathbf{x}_t) > \gamma$ for all t. Calculate upper bounds on the numbers of mistakes that would be made by both Perceptron and Winnow. Which algorithm would be a better choice here?

(b) Dense target vector u, sparse feature vectors $\mathbf{x}_t$. [10 points]
Suppose you are in a setting with high-dimensional features (large d), and that the feature vectors are sparse; for simplicity, suppose $\mathbf{x}_t \in \{0,-1,+1\}^d$ for all t and that each $\mathbf{x}_t$ has $k \ll d$ non-zero entries. Suppose you are told the examples in T trials are linearly separable by a dense weight vector $\mathbf{u} \in \mathbb{R}_+^d$ with $\|\mathbf{u}\|_1 = d$ and $\|\mathbf{u}\|_2 \leq 2\sqrt{d}$, and that you are given $\gamma > 0$ such that $y_t(\mathbf{u}^\top \mathbf{x}_t) > \gamma$ for all t. Calculate upper bounds on the numbers of mistakes that would be made by both Perceptron and Winnow. Which algorithm would be a better choice here?

(c) If your problem has non-negative feature vectors $\mathbf{x}_t \in \mathbb{R}_+^d$, is the Winnow algorithm a meaningful choice? Why or why not? [5 points]

3 Principal Component Analysis [35 points]

In this exercise, you are provided part of the MNIST digit data set, containing 12,000 images of handwritten digits (data is provided in `MNIST_train.mat`). Each example is a 28×28 grayscale image, leading to 784 pixels. You will perform PCA in an unsupervised manner; however for some parts of the problem, you will be asked to analyze data points corresponding to particular digits, and for this purpose you are also provided labels indicating the digit for each image (these are given separately in a column vector, with rows corresponding to rows of the image data matrix). The labels are shifted by 1, e.g. the label for the digit "0" is 1, and so on. At the end of the problem, you will also implement an undercomplete autoencoder and compare its reconstruction error to PCA.

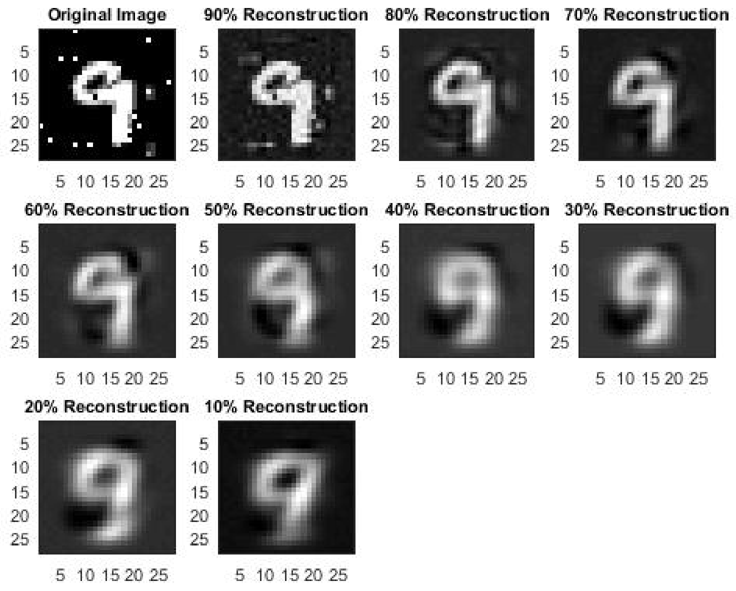
1. To get familiar with what the data looks like, generate the last example (the $12,000^{th}$ row of `X_train`) as an image. Provide your code and the resulting image. (*Hint:* You can reshape the data back into a 28×28 image using `reshape` in MATLAB, and you can use `imagesc` and `colormap gray` to see the picture. Make sure you are able to get the picture to face the right way, which should look like a "9".)

2. Run PCA on the data. You can use the built-in `pca` function in MATLAB. (*Reminder:* Make sure you understand the MATLAB function you used, especially whether the function standardizes the data set before running PCA. Make sure that the data is mean-centered.) Just like in the previous part, generate the first principal component vector as an image. Repeat for the second and third. Do these images make sense? Explain.

3. Create a 2D scatter plot showing all examples for digit "0" (so `Y_train` is 1) and digit "7" (so `Y_train` is 8) in the first 2 PC dimensions, clearly labeling your axes; use different colors for points corresponding to the digits "0" and "7" to see the difference and provide a legend. Make sure the data points are mean-centered before projecting onto the principal components. Now create a similar plot showing the same data points projected in the 100^{th} and 101^{st} PC dimensions. What do you observe? Can you explain why you might expect this?

4. Graph the (in-sample) fractional reconstruction accuracy as a function of the number of principal components that are included. Also give a table listing the number of principal components needed to achieve each of 90%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10% reconstruction accuracy (i.e., to explain X% of the variance).

5. Using the numbers you found in the previous part, reconstruct the 500^{th}, 6000^{th}, and 10000^{th} examples using each of the different numbers of principal components. (*Hint:* Take a look at `subplot` and try to do it in a loop to save time.) For instance, the last example looks like:



6. Now, train several undercomplete autoencoders using our matlab's built-in function: `trainAutoencoder520`.The function has all the same functionalities as the ordinary Matlab `trainAutoencoder` function, except that it cannot take sparsity or regularization parameters, and can use a linear activation function in the encoder. Refer to documentation on trainAutoencoder for key-value arguments. By default, the function learns an autoencoder with 1 hidden layer. When training, set "MaxEpochs" to 250 and LossFunction to "mse". And before training each model, be sure to call rng("default").. Expect each autoencoder to take 5 minutes to train.

   (a) Train one autoencoder with the size of the hidden layer to the number of principal components required to achieve 30%, 60%, 90% reconstruction accuracy threshold in question 4. Be sure to use mean-centered data as your input.

   (b) Repeat the previous step with linear transfer functions. (You should have trained 6 autoencoders total)

   (c) Graph the (in-sample) reconstruction accuracy of your autoencoders as a function of the size of the hidden state. Re-graph your PCA reconstruction accuracies from part 4 on the same plot. (Your plot should have three lines: one for PCA, one for non-linear autoencoders, and one for linear autoencoders. Each autoencoder line should have 3 points, and the PCA line should have 9.)

   (d) Compare the accuracies of the three algorithms. Explain why you do (or do not) observe differences in performance between each of the algorithms.

7. How would you expect increasing the number of hidden layers in the encoder and decoder to affect your reconstruction error? Why? For this question, you do not need to perform a rigorous proof. Describe your intuitions and defend them.

4 Eigenvectors [15 points]

1. **[7 points]** Show that if X has rank p (all its columns are linearly independent) and $n > p$ then using the p-dimensional pseudo-inverse $X^+ = V_k \Lambda_k^{-1} U_k^T$ in $\widehat{w} = X^+ y$ with $k = p$ solves the least squares problem $\widehat{w} = argmin_w (y - Xw)^T (y - Xw)$.

2. **[8 points]** We want to efficiently find the largest eigenvectors of the matrix $X^T X$ where X is $n * p$ with $p \gg n$. Show how to do this using the largest eigenvectors of XX^T