Topics in Machine Learning (Fall 2018)	December 11, 2018
Assignment 3	
Lecturer: Ohad Shamir	*Due: January 1st*

General Instructions

- **Read the instructions carefully, and follow the required filename and function syntax.**
- You're allowed to submit the assignment either individually or in pairs.
- If you're unfamiliar with MATLAB, a separate PDF file contains a quick MATLAB tutorial which will give you the knowledge you need to complete this assignment. Many other MATLAB tutorials are available online.
- You don't need to check for illegal input (e.g. that the binary labels are all $+1$ or -1, that positive input parameters are indeed positive etc.). To help us evaluate your work, please document your code.
- Send all submitted files (should be 11 overall) as a single zip file, with the file name being your full name/s, to `safran.itay@gmail.com`, with '[mlassignment3]' in the email title.

Problem 1 Linear Predictors

1. Implement in MATLAB a function which given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ and a parameter B, solves

$$\min_{\mathbf{w}:\|\mathbf{w}\|\leq B} \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{x}_i, \mathbf{w}\rangle\}$$

   Specifically, the function should be in a file named `learn_linear.m`, with the following syntax: `function [w]=learn_linear(X,Y,B)`. `X,Y` is the training set (each row `X(i,:)` corresponds to example i, with label `Y(i)`), and `w` is the predictor returned by the algorithm.

   - The function should implement the online gradient descent algorithm learned in class (on December 18th), where $f_t(\mathbf{w}) = \max\{0, 1 - y_{i_t} \langle \mathbf{x}_{i_t}, \mathbf{w}\rangle\}$ and i_t is chosen uniformly at random from $\{1, \ldots, m\}$, and (for simplicity) $\eta_t = 1/\sqrt{t}$. The algorithm should run for $T = 100m$ iterations and return the average $\frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$.

- To implement the algorithm, you need to compute projections of the form $\arg\min_{\mathbf{w}:\|\mathbf{w}\|\leq B} \|\mathbf{w}-\mathbf{a}\|^2$. This equals $\mathbf{a}$ if $\|\mathbf{a}\| \leq B$, and $\frac{B}{\|\mathbf{a}\|}\mathbf{a}$ otherwise (you're encouraged to prove this yourself).

2. Load `linear_data.mat`, which contains labeled examples in $\mathbb{R}^2$. Run your algorithm (with $B = 10$) to get `w`, and then apply the function `show_linear(X,Y,w)` (provided by us) to display the resulting predictor. Save the resulting plot as a JPEG file `linear.jpg`.

3. Load `mnist.mat`. This file contains two datasets: `Xtrain,Ytrain` for training, and `Xtest,Ytest` for validation. Each row of `Xtrain` and `Xtest` encodes an image of a handwritten image (either '4' or '7'), as a vector of pixel values. Our goal is to classify the digits correctly.

   - To see a particular image (say row i in `Xtrain`), type `imagesc(reshape(Xtrain(i,:),28,28)'); colormap gray;`.

4. Run your algorithm on `Xtrain,Ytrain`, for 3 different values of the parameter B ($10^{-5}, 10^0, 10^5$), and in each case, test the resulting predictor $\mathbf{w}$ by calculating the average $0-1$ loss on the test set `Xtest,Ytest` (recall this is defined as $\ell(\mathbf{w},(\mathbf{x};y)) = \mathbf{1}(\text{sign}(\langle \mathbf{w},\mathbf{x}\rangle) \neq y)$). In a text file called `linear.txt`, write the average loss you got for each value of B; indicate which value of B seems best; and briefly explain why other values of B lead to worse results.

   - The average loss may vary due to the algorithm randomness. You may repeat the experiment a few times to make sure which value of B tends to be better, and report the average loss in each experiment.

5. Rerun the algorithm with the parameter B chosen above. After finding `w`, type `imagesc(reshape(w,28,28)'); colormap gray;`. This shows you how the predictor `w` looks like graphically: bright pixels are those on which `w` has a large positive weight, and dark pixels are those on which `w` has a large negative weight. Based on this image, explain in `linear.txt` why the predictor `w` you found works well. It may help you to look at a few digit images from `Xtest`, both those which were correctly predicted by `w`, and those which were incorrectly predicted by `w`.

6. Submit `learn_linear.m`,`linear.jpg`, and `linear.txt`.

Problem 2 Kernel Predictors

1. Implement in MATLAB a kernel prediction algorithm which given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ and a parameter B, solves

$$\min_{\mathbf{w}:\|\mathbf{w}\|\leq B} \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i \langle \Psi(\mathbf{x}_i), \mathbf{w}\rangle\},$$

where Ψ is the feature mapping corresponding to a Gaussian kernel with parameter σ^2: $K(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / \sigma^2)$.

Specifically, write a matlab function in a file named `learn_gaussian.m`, with the following syntax: `function [alphas] = gaussian(X,Y,B,sigma2)`. `X,Y` is the training set, (in the same format as the previous question), B is the norm bound, and `sigma2` is the kernel parameter. `alphas` is an $m \times 1$ vector, with the weights of the predictor $\sum_{i=1}^{m} \alpha_i \Psi(\mathbf{x}_i)$ returned by the algorithm.

- To solve the optimization problem, implement the online gradient descent algorithm as in the linear case (with $\eta_t = 1/\sqrt{t}$ and $T = 100m$), replacing each use of $\langle \Psi(\mathbf{x}), \Psi(\mathbf{x}') \rangle$ with $K(\mathbf{x}, \mathbf{x}')$, and maintaining a vector of weights α_t over the training set to implicitly represent $\mathbf{w}_t = \sum_{i=1}^{m} \alpha_{t,i} \Psi(\mathbf{x}_i)$.
- Do not keep recalculating $K(\mathbf{x}_i, \mathbf{x}_j)$. Instead, compute a matrix `G` of size $m \times m$, where `G(i,j)`$=K(\mathbf{x}_i, \mathbf{x}_j)$, and use it throughout the run of the algorithm. You can do this efficiently in MATLAB by executing
  `G = exp(-((repmat(diag(Z)',m,1)-2*Z+repmat(diag(Z),1,m)))/sigma2);`,
  where `Z=X*X';`.
- Other than the for loop appearing explicitly in the pseudocode, you don't need any other loop in the MATLAB code. This is important for your algorithm to run reasonably fast, because MATLAB is slow in executing loops.

2. Load `gaussian_data.mat`, which contains labeled examples in $\mathbb{R}^2$. Run your algorithm with `sigma2`=2 to get `alphas`, and then apply the function `show_gaussian(X,Y,alphas,sigma2)` (provided by us) to display the resulting predictor. Do this once for `B` = 1 and once for `B` = 100. Save the resulting plots as JPEG files `gaussian1.jpg`, `gaussian100.jpg` respectively.

3. In a text file `gaussian.txt`, describe what is the difference between the two figures. Explain briefly why this illustrates that picking a large hypothesis class (corresponding to a larger B) may make the learning algorithm more likely to overfit.

4. Submit `learn_gaussian.m`, `gaussian1.jpg`, `gaussian100.jpg`, and `gaussian.txt`.

Problem 3 Viola-Jones Face Detector

1. Implement in MATLAB the AdaBoost algorithm for the Viola-Jones face detection scheme. Open the `vjAdaBoost.m` file, and complete the matlab function with the same name.

   - You may use the function `WL` which returns a weak learner for a given sample and distribution over the sample, and the function `predict` which returns the predictions of a weak learner over a sample.

- The algorithm should run for a few minutes for each iteration of AdaBoost, so make sure to debug your code on a small training set.

2. Run your algorithm on a training set of size 1000, for 20 iterations, and save its output.

3. Plot a graph of the weak learner's (weighted) training error against the strong learner's training error, as a function of the iteration number. Save the resulting figure as a JPEG file `vj1.jpg`.

4. Pick a sample instance of your choice and use the function `showFeatures` to plot the first five features AdaBoost returned. Save the resulting figure as a JPEG file `vj2.jpg`.

5. Submit `vjAdaBoost.m`, `vj1.jpg`, `vj2.jpg` and `vjRes.mat`.