Machine Learning *CS 4364/6364, Fall 2016*

HW1

Instructions

Collaboration: You are 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 at this address: machine.learning.gwu 'at' gmail.com. Electronic submissions are preferred. Math can be formatted in Latex (some easy editors for Latex are Lyx, TexShop), Word, or other editors. Math solutions can be optionally submitted on paper at the Department of Computer Science (in the dropbox to the right of the entrance to SEH 4000), but all code should be submitted electronically. The assignment is due Friday, September 30, at 5 pm.

Questions

1. **Linear Algebra - 10 pts**

   Given matrix $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$

   (1) Are the vectors $\boldsymbol{x} = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix}$, $\boldsymbol{y} = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}$, $\boldsymbol{z} = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}$ linearly independent? Justify your answer.

   (2) Find the eigenvalues and the corresponding eigenvectors of A

   (3) Let M be any matrix with real entries. M is *positive semidefinite* if, for any vector $\boldsymbol{x}$ with real components, the dot product of Mx and x is nonnegative, $\langle M\boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$

   Let $B = \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}$. Show that B is positive semidefinite.(i.e., show that $\langle B\boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$)

   (4) A *symmetric* matrix H is positive semidefinite if and only if the eigenvalues of H are all non-negative . Is matrix A positive semidefinite? Why?

2. **Perceptrons - Matlab Programming - 10 pts**
Implement a Perceptron classifier in MATLAB. Start by implementing the following functions:

   - a function `perceptron_train(X,y)` where X and y are $n \times d$ and $n \times 1$ matrices respectively. This function trains a Perceptron classifier on a training set of n examples, each of which is a d dimensional vector. The labels for the examples are in y and are -1 or 1. The function should return $[\theta, k]$, the final classification vector and the number of updates performed, respectively. This is the simple perceptron classifier seen in class, where the linear separator passes through the origin.
   - a function `perceptron_test(` $\theta, X_{test}, y_{test}$ `)` , where θ is the classification vector to be used. X_{test} and y_{test} are $m \times d$ and $m \times 1$ matrices respectively, corresponding to m test examples and their true labels. The function should return $test_{err}$, the fraction of test examples which were misclassified.

   1. Load the data using the `load_p1_a` script and train your Perceptron classifier on it. This data set is linearly separable, through the origin. Using the function `perceptron_test`, ensure that your classifier makes no errors on the training data. What is the angle between θ and the vector $(1, 0)^T$? What is the number of updates k_a required before the Perceptron algorithm converges?
   2. Now, consider the IRIS dataset discussed in lecture. We have created a binary classification problem to determine whether a given flower is a setosa or not. To create this, we pre-processed the labels to create a label vector where setosa's label is unchanged (i.e. its label is 1), but both versicolor and virginica are now labeled as -1. The data contains two out of the four attributes, petal width and petal length. This training data and the modified labels can be loaded using the script `load_p1_b`. From class we know that this data is linearly separable but the separator does not pass through the origin. Therefore you will need to follow the technique used in lecture to further pre-process the data to be linearly separable through the origin: for each data point, add an additional feature, set to a constant value (please use the value 1 for consistency). Then obtain the classification vector using your function `perceptron_train`. What is the angle between θ and the vector $(1, 0)^T$? What is the number of updates k_b now?
   3. For the previous two questions, compute the geometric margins, γ^a_{geom} and γ^b_{geom} of your classifiers with respect to their corresponding training datasets. Recall that the distance of a point from x_t from the line $\theta^T x = 0$ is $\frac{\theta^T x_t}{||x||}$.
   4. Plot the data (as points in the X-Y plane) from the first part, along with the decision boundary that your Perceptron classifier computed. Create another plot, this time using data from the second part and the corresponding decision boundary. Your plots should clearly indicate the class of each point. We have a MATLAB function `plot_points_and_classifier` which you may find useful.

3. **k-Nearest Neighbor - Matlab programming - 10 pts**
For this problem, you will need MNIST optical character recognition (OCR) data included with the problem set in the file `MNISTdata.mat`. There are 2000 training samples and 1000

test samples, stored in four files called `trainx`, `trainLabel`, `testx`, and `testLabel`. To load the data into the Matlab workspace, use the Matlab command:

```
load('MNISTdata.mat')
```

called within the folder where you saved the files. The training data consists of two arrays: a 2000×784 array `trainx` containing the data points, one point per row, and a one-dimensional array `trainLabel` (of length 2000) containing the labels $(0 - 9)$. The test data is similar, but contains only 1000 points in `testx` and `testLabel`.

In order to view one of the data points, you can use Matlab's `image` command, although you first need to `reshape` the format from a 784-dimensional vector into a 28×28 matrix. For instance, to see point number 75, you would use:

```
image(reshape(trainx(75,:), [28 28]));
```

If you find the output somewhat garish, try prefacing the last command with:

```
colormap(1.0-gray);
```

In this exercise, we try to understand the effect of training data on the performance of a nearest neighbor classifier.

1. For each class, extract all the elements in the training set that belong to the class. Using this, write a Matlab function for computing the mean of every class. Call this set of means, M.
2. Write a Matlab function for computing the k nearest neighbors of a query point z, along with their majority vote:

   ```
   function [class, ids] = knn(X,C,z,k)
   ```

   Here X is the training data (some $n \times d$ array of n points in $\mathbf{R}^d$) and C is a vector of training labels (of length n). The values returned are `ids`, a vector of length k holding the indices (into X) of the k nearest neighbors; and `class`, the majority vote over those neighbors.
3. One of the things the above function does is to compute the Euclidean distances from z to each row of X. Perhaps the most obvious way to do this is in a loop:

   ```
   dist = zeros(1,n);
   for i = 1:n
     dist(i) = norm(z, X(i,:));
   end;
   ```

   Here is a faster alternative. Start by precomputing an array `sqlength`:

   ```
   sqlength = sum(X .* X, 2);
   ```

   Now, when given a query point z (say in row vector format), use the following in place of the distance computation shown above:

   ```
   dist = sqlength - 2 * X * z';
   ```

This does not actually compute Euclidean distance, but it does return the right answer when used in a nearest-neighbor procedure. Explain why. [*Hint:* use the expansion $||x-y||^2 = ||x||^2 + ||y||^2 - 2x \cdot y$.]

4. Determine the error rate of the k nearest-neighbor classifier (on the test set) as a function of k, for $k \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}$. Plot two separate error curves for the following cases:
   - Using the entire training data as the input to the function `knn`
   - Using only M as the input to the function `knn`.

   The Matlab `plot` function is useful for this.

4. **Prototype Selection (required for grad students only) - 10 pts**

1. *Prototype selection* is the problem of choosing a few instances that most accurately and completely represent observed data for some particular task. In the context of k-nearest neighbors, the prototype selection problem can be stated as follows:
   - You are given a labeled dataset (X, y) where X is a matrix with one data point per row, $x_i \in \mathbb{R}^d$, and y is a column vector with $y_i \in \mathcal{Y}$, the space of possible labels.
   - You are also given a number p.
   - Rather than using the entire dataset for k-NN classification, you want to pick p prototypes from $\mathbb{R}^d \times \mathcal{Y}$ which will give good performance. (And of course, performance is always evaluated on a separate test set drawn from the same underlying distribution.) In particular, the prototypes do not need to be chosen from X. Design, implement, and evaluate such a k-NN prototype selection algorithm. Clearly describe your algorithm in prose and/or pseudocode, and explain what it is trying to optimize.
2. Write a Matlab function that takes a labeled dataset (X, y), a value k, and a total number of prototypes p; and returns the p prototype vectors and their labels:

   ```
   function [Xp,yp] = prototypes(X,y,k,p)
   ```

3. Apply your method to the OCR training data you used in question 4 with $p = 40, 80, 160, 320$ and $k = p/20$. You *cannot* use the test data in selecting your prototypes; but if you like, you can set aside some of the (labeled) training data as a holdout validation set. Evaluate the resulting prototypes on the OCR test data using the k-nearest neighbors classifier.