Assignment 2: Classification via Loss Minimization and Margin Maximization CS 725, IIT Bombay *

February 14, 2015

Introduction

This assignment will require the use of MATLAB and the CVX package [1]. Students not familiar with MATLAB will find a friend in `http://ocw.mit.edu/courses/mathematics/18-s997-introduction-to-MATLAB-programming-fall-2011/`. Those familiar with MATLAB will find it extremely easy to work with cvx.

Training data

A dataset similar to the r3 dataset from the previous assignment will be used for the entire assignment. A training set with m examples, (X, y), will have:

- X**:** m x 3 matrix, every row is an instance, each column represents a feature. All the features are real numbers.
- y**:** m x 1 vector, $y_i \in \{-1, 1\} \forall i \in 1, 2, ..., m$.

Note that we define the classifier to be $\sum_{k=1}^{3}(W_i * X_i^k) - b$ or more succinctly, $(W^T X_i - b)$ for an input X_i.

Solving the assignment

5 steps have to be completed in order to complete the assignment. There is a MATLAB file corresponding to each of the steps. For the first three steps, the files have some missing functions. Your task is to supply those functions. The last two steps are demonstrations, you have to add your comments for these parts.

*Prepared by Aman Madaan (amanmadaan@cse.iitb.ac.in), T.A., CS 725 (Instructor: Prof. Saketha Nath)

[1] feel free to use any other optimization toolbox of your choice

Getting started

Unzip the folder "assignment2.zip" to a directory of your choice, and set the current working directory to assignment2, using the "cd" command. This folder has all the files that you need to complete the assignment. You will need to add all the files to this directory to complete the assignment correctly.

Step 1: Classification using loss functions

Once you have changed the directory to assignment2, read train_loss.m .

It samples n points for each class, where each of the classes are defined by some gaussian distribution.

These points are then used to train the classifiers.

Now type the following in the command window:

```
>> train_loss(100)
```

You should see the following error (and no other error)

```
??? Undefined function or method 'hinge_loss' for input arguments of type 'double'.

Error in ==> train_loss at 9
    [W b] = hinge_loss(X, y);
```

To remove this error, you will need to supply an implementation of the hinge loss based classifier.

Create a new file, hinge_loss.m and add your implementation of the *hinge_loss* in it. You can write one using cvx (or an optimization toolbox of your choice). The signature of the function should be

```
function [W1, b] = hinge_loss(X, y)
```

That is, it should take the training data as described above and return the separating plane (W, b).

Once you do that, call train_loss to see if your hinge loss classifier works:

```
>> train_loss(100)
```

It will plot the points, and the separating plane that your (hinge loss based) classifier has just returned. Verify that it is doing an okay job at separating the points by calling the train_loss(100) several times.

If the plane does not seem to be how it should be, or there is an error, fix it before proceeding. If everything is fine, note another (new) error:

```
??? Undefined function or method 'hinge_loss_square' for input arguments of type 'double'.

Error in ==> train_loss at 14
    [W_hsq b_hsq] = hinge_loss_square(X, y);
```

To fix this error, follow the same steps: add hinge_loss_square.m and add your implementation of hinge_loss_square with the appropriate signature.

```
>> train_loss(100)
```

The whole process should finish without any error now. You should see 3 planes, the green bayes optimal, yellow hinge loss square and the red hinge loss based along with the data. Save the plot as your_rollnumber_part1.png.

Observe

Call train_loss with argument 10, 20, 50 and 100.

Observe how the planes start overlapping with the (green) bayes optimal plane as you increase the amount of training data.

Step 2: Classification by minimizing loss and maximizing margin

In this step, you will complete the implementation of train_loss_margin.m

Recall from the lectures that since you now have both maximization of margin and minimization of loss in the objective, a regularization (hyper)parameter, C, is required. For this step, let's agree to fix $C = 1$. We'll find the best C in the next step.

The following should looks familiar now:

```
>> train_loss_margin(100, 1)
??? Undefined function or method 'hinge_loss_max_margin' for input arguments

of type 'double'.

Error in ==> train_loss_margin at 9
    [W b] = hinge_loss_max_margin(X, y, C);
```

To finish this step, you will need to supply implementation of hinge_loss_max_margin() and hinge_square_loss_max_margin(). Add them in the files called hinge_loss_max_margin.m and hinge_square_loss_max_margin.m respectively.

Once you are done, you should see 3 planes, along with the data as in step 1. Save the plot as rollnumber_part2.png.

Observe

From the class, recall what was supposed to happen when you set $C = 0$.

```
>> train_loss_margin(100, 0)
```

As discussed, the slacks can now be shot up to a **big** number, and Ws be set to 0 to minimize the objective. Note that this is precisely what happens.

Here is an actual output from one of the runs:

```
Status: Solved
Optimal value (cvx_optval): +6.3946e-14

W1 =

   1.0e-13 *

   -0.0618
    0.3071
   -0.4873


b =

   -0.0049
```

You should observe something similar.

Step 3: Finding the best C

In the above part, we fixed C to 1. As discussed in the class, these parameters play a very important role in the performance of a classifier and should not be pulled out of air.

In this part, you are required to implement 5-fold cross validation on a given dataset and find the best value of C for hinge loss with max margin and hinge square loss with max margin objectives.

Read cross_validation_driver.m before starting.

The function you will have to complete for this part is called cross_validate (to be added in a file called cross_validate.m) and has the following signature:

```
function [best_accuracy, best_C] =
                  cross_validate(X_train,y_train, C_range, folds, classifier)
```

Once you are done implementing the function, test by typing the following in the command window:

```
>> c_range = 2 .^ [1:5]

c_range =

     2     4     8    16    32

>> cross_validation_driver(c_range)
```

The last two lines of the output should read:

```
Hinge loss max margin: Best Accuracy = ? for C = ?
Hinge square loss max margin: Best Accuracy = ? for C = ?
```

Copy and paste these two lines in a file called rollnumber_cvecres.txt.

Observe

It was discussed in the class that testing over a large range of C values does not necessarily mean that we will obtain better results on a C. Can you observe this? See trying_more_c.m for some sample code.

Step 4: The case of separable datasets and the dancing hyperplanes

Perceptron is another linear classification algorithm that is guaranteed to converge in the case of separable datasets. However, peceptron is happy with any separating plane, and not necessarily the plane with maximum margin.

In this step, we will observe how max margin objective leads to *stabler* separating planes for different datasets, all sampled from the same distribution.

In the command console, type

```
>> hyp_dance(30)
```

The demonstration will start. After the first plot, there will be a prompt asking you to fix a viewpoint. Rotate the plot so that both the classes are visible and press any key.

You should observe that the hyperplane obtained by max margin objective is very stable (i.e. stays about the same) and is closer to the bayes optimal. The perceptron *dances* around the bayes optimal.

You don't have to write any code for this part (you will need to implement hinge loss with max margin though (step 2)). Add your comments (3-4 lines) (was this expected, why?) lines on this demo in a file called step4.txt.

Step 5: The dual and the support vectors

If you have implemented your hinge_loss_max_margin function using cvx, chances are that cvx will the solve dual of the problem.

As discussed in the class, dual formulation is good because for "natural" datasets, we expect a large number of points to be away from either of the boundaries, and thus methods like coordinate ascent converge quickly.

It was also mentioned that if all the points in the dataset happen to be the support vectors, the dual won't be of much help and the convergence will be very slow.

To see for yourself, type:

```
>> on_plane(40)
```

Compare the time taken by planar data case and the non planar data case.

You don't have to write any code for this part (you will need to implement hinge loss with max margin though (step 2)). Add your comments (3-4 lines) (was there a lot of difference in time? why/not?) lines on this demo in a file called step5.txt.

Submission

Your assignment2 folder now has everything that you have to submit: the code files, text and the required plots (which you were asked to save in some of the steps).

Here is a list of the files that you must have added to the folder as you progressed:

- hinge_loss.m
- hinge_loss_square.m
- hinge_loss_max_margin.m
- hinge_loss_square_max_margin.m
- cross_validate.m
- rollnumber_part1.png
- rollnumber_part2.png
- rollnumber_cvecres.png
- step4.txt
- step5.txt

Rename your assignment2 folder to yourrollnumber_assignment2, zip and submit.