Problem Set 3
Due on Friday, October 21, 2016 at 11:55 pm

How to Submit

Create one archive file (.zip, **not** .rar) of your code and submit it via `ilearn.ucr.edu`. Supply your answers to question 1 as `q1.txt` or `q1.pdf`. Supply your code for question 2 as `learnsvm.m`. Supply your code for question 3 as `q3.m`. You may also include other `.m` files containing functions you wrote that are helper functions to those above.

Do not supply any directories in your zip file. Each file should (in comments) list

- Your name
- Your UCR student ID number
- The date
- The course (CS 229)
- The assignment number (PS 3)

Q1: Gradient Descent for Linear SVM [4 points]
Recall that the total loss for a linear SVM can be written as

$$C\sum_i [1 - y_i f(x_i)]_+ + \frac{1}{2} w^\top w$$

Use this SVM-loss function with regularization to derive the stochastic gradient descent and "regular" gradient descent learning rules for a linear SVM.

How does this compare to the perceptron learning rule? Why?

Supply your answer (the derivation, the update rule, and the comparison asked above) in a file titled `q1.txt` or `q1.pdf`.

Q2: Linear SVM Code [4 points]
Use the equation you derived above to implement a function that finds the weights (w) and intercept (b) solution for an SVM problem. Your function (defined in the file `learnsvm.m`) should have signature

```
function [w,b] = learnsvm(X,Y,C)
```

Use gradient descent (not stochastic gradient descent). Start w and b at 0. Selecting the step size can be tricky. To be consistent and (relatively) simple use the following method.

- Start the step size at $\frac{1}{Cm}$ (m is the number of data points).
- Take 100 steps and compare the objective before the 100 steps to that after 100 steps.
- If the 100 steps improved the objective, increase the step size by 5%.
- If the 100 steps made the objective worse, decrease the step size by 50%.

Stop when the step size is less than $10^{-6}/Cm$.

Use the Matlab plotting capabilities to debug your algorithm. You may find the provided `drawline.m` function helpful. The supplied "simple" files give two simpel 2D cases to test your algorithm.

Q3: SVM application [3 points]

For this problem, use the supplied `qplearnsvm.m` function instead of the one you derived above. It is much faster to use a dedicated method for SVM and not gradient descent.

The files `spamtrainX.data` and `spamtrainY.data` have examples and their labels (+1 is spam, -1 is not spam) for a spam-classification dataset (already randomly ordered). The features are the relative frequency of different words, characters, and particular sequences in the e-mail. `http://archive.ics.uci.edu/ml/datasets/Spambase` has more information about the data.

The files `spamtestX.data` and `spamtestY.data` have the same, but for the testing set.

Use 3-fold cross-validation to pick C. Plot, as a function of C (on a semi-log plot), the error *rate* for both the cross-validation and for the testing data. Note that for the testing data, this would be the error from training on the entire training set and then evaluating the classifier on the testing set. For the cross-validation, this would be the total error *rate* across all three splits of training on two-thirds and testing on the remaining third. It is up to you to select a reasonable range of C values to plot.

Title your plot with the C value selected by this cross-validation method and the resulting *testing* accuracy. Create one *function* that runs this part of the assignment and call it `q3` (it should take no parameters, but be a *function*).