Perceptron vs SVM

Tommi Jaakkola, course materials for Machine Learning, Fall 2006.
MIT OpenCourseWare (http://ocw.mit.edu/)

1 Percepton

Implement a Perceptron classifier in MATLAB. Start by implementing the following functions (you should attach a printout of your MATLAB code of these functions with your submission):

- 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. You may assume that the input data provided to your function is linearly separable.
- a function **perceptron_test(theta, X test, y test)** where *theta* 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.

For this problem, we have provided you two custom-created datasets. The dimension d of both the datasets is 2, for ease of plotting and visualization.

(a) Load data using the **load_p1_a** script and train your Perceptron classifier on it. Using the function perceptron test, ensure that your classifier makes no errors on the training data. What is the angle between *theta* and the vector $(1, 0)^T$? What is the number of updates k_a required before the Perceptron algorithm converges?

(b) Repeat the above steps for data loaded from script **load_p1_b**. What is the angle between theta and the vector $(1, 0)^T$ now? What is the number of updates k_b now?

(c) For parts (a) and (b), 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 x_t from the line $\theta^T x = 0$ is $|\frac{\theta^T x_t}{\|\theta\|}|$.

(d) For parts (a) and (b), compute R_a and R_b, respectively. Recall that for any dataset X , $R = \max\{\|x\| \, | x \in X\}$.

(e) Plot the data (as points in the $X - Y$ plane) from part (a), along with decision boundary that your Perceptron classifier computed. Create another plot, this time using data from part (b) and the corresponding decision boundary. Your plots should clearly indicate the class of each point (e.g., by choosing different colors or symbols to mark the points from the two classes). We have a provided a MATLAB function **plot_points_and_classifier** which you may find useful.

2 SVM

Implement an SVM classifier in MATLAB, arranged like the Perceptron in problem 1, with functions (svm_train(X, y)) and (svm_test(theta, X_test, y_test)). Again, include a printout of your code for these functions.

Hint: Use the built-in quadratic program solver $quadprog(H, f, A, b)$ which solves the quadratic program: $\min\{\frac{1}{2}x^T Hx + f^T x\}$ subject to the contraint $Ax \leq b$.

(a) Try the SVM on the two datasets from Problem 1. How different are the values of *theta* from values the Perceptron achieved? To do this comparison, should you compute the difference between two vectors or something else?

(b) For the decision boundaries computed by SVM, compute the corresponding geometric margins (as in Problem 1(c)). How do the margins achieved using the SVM compare with those achieved by using the Perceptron?

3 SVM light

Now that you are familiar with SVMs, download this highly optimized version called SVM_{light}: **http://svmlight.joachims.org/**

We will use this package to experiment with classifying images of handwritten digits. To simplify things, we will only be distinguishing between 0s and 1s.

(a) Train a plain-vanilla SVM (i.e, no regularization) on **train-01-images.svm** (which is already in SVM_{light}'s format). What is the training error, i.e., the fraction of examples in the training data that are misclassified? From the set of misclassified images, pick one. Attach of plot of it with your solution. Why do you think the SVM fails to classify it correctly during training? Now apply the SVM to the test set **test-01-image.svm**; what is the test error, i.e., the fraction of test data that is misclassified?

(b) Experiment with different values of the regularization term C (set using the -c flag). Start by guessing/estimating a range in which you think C should lie. Then choose the values of C (within that range) at which you will evaluate the performance of the SVM. You need not pick more than 10 such values, though you should feel free to pick as many (or as few!) as you want. For these values of C, plot the corresponding error. What value of C gives you the best training error on the dataset from part (a)? How does the test error for this choice of C compare with the test error you computed in part (a)? Could you optimize C so that it leads to the best test error, rather than the training error? Should you?

(c) An adversary (we'll call him W) mislabeled 10% of the images in the original training set to produce **train-01-images-W.svm**. Using the same choices of C as in part (b), find the C that results in the best training error. What is the corresponding test error on **test-01-image.svm**?