Machine Learning for Computer Vision, MVA 2013-2014

Instructor: Iasonas Kokkinos, iasonas.kokkinosecp.fr

Programming Assignment 2 (4 Points/20) Deliverable: 17/11/14

1 Support Vector Machine classifiers (1.5/4)

Use the LibSVM package to train a SVM classifier: a simple example of its use is provided in demo_libsvm.m. The first 3 bullet items outline the steps followed in `lab3.m`.

1 Use a Gaussian kernel, and use 10-fold cross-validation to choose the cost and precision parameters (referred to as `-c: cost / -g: gamma` parameters in LibSVM documentation).

Visualize the cross-validation error (a 2D function) using imagesc.

Once you have picked the best combination of cost and precision evaluate the classifier's generalization error on the test-set - report the number of misclassified points.

.5 Plot the decision boundaries, by appropriately modifying the code from the previous exercises. Superimpose on your plot also the support vectors, using the code outlined in `week_2_a.m`.

If you work is working right, you should be getting results of the following form for the cross-validation error and the decision boundaries:

Figure 1: Left: Cross-validation error, as a function of σ and γ, middle: value of the svm-based classifier, right: level-sets at $\{-1, 0, 1\}$ and support vectors.

2 Adaboost (1.5/4)

In `week_2_b.m` your task is to train an Adaboost classifier with synthetic data. For reference, you are provided with the posterior $P(y = 1|x), x \in \{[0, 1] \times [0, 1]\}$ regularly sampled on the $[0, 1] \times [0, 1]$ domain, so

that you will see how the output of the Adaboost classifier better approximates the posterior at each round.

- (.1/1.5) Implement the Adaboost algorithm described in class. This involves iterating over the following steps:
  1 Find the best decision stump at each round.
  2 Evaluate the weak learner's weighted error, and estimate α_t.
  3 Update the distribution over the training samples.
- (.2/1.5) Plot $E(-y(x)f(x)) = \sum_{i=1}^{N} \exp(-y^i f(x^i))$ as a function of the boosting round. Make sure that E is monotonically decreasing with time (otherwise your code is wrong). Verify that $E(-y(x)f(x))$ provides an upper bound for the number of errors.
- (.3/1.5) Show the response of your strong learner side-by-side with the posterior. Make sure that they look increasingly similar at larger iterations.

If you code is working right, you should be getting results of the following form for the cross-validation error and the decision boundaries:

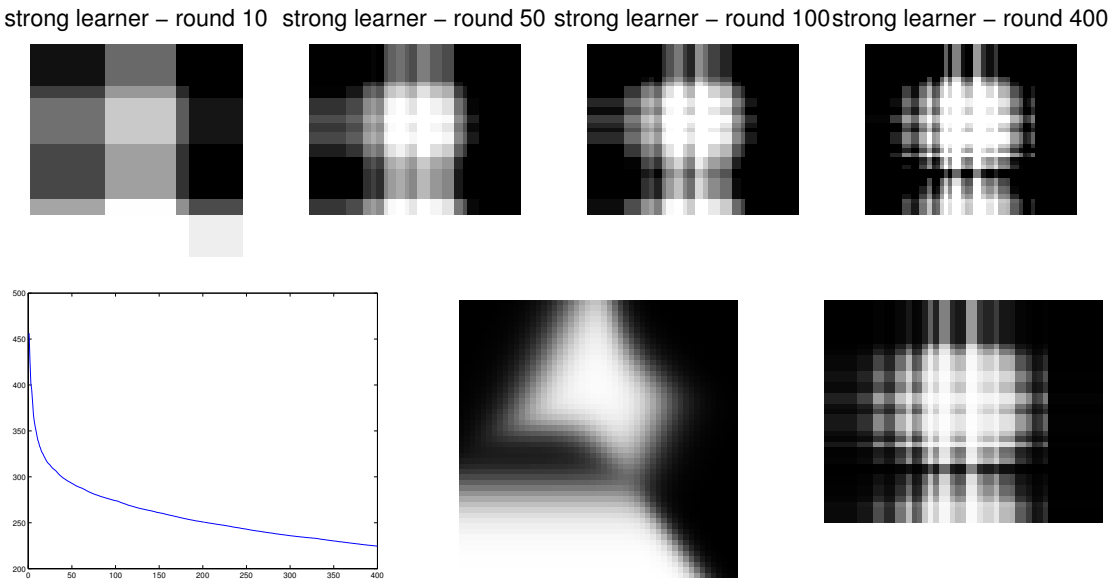

Figure 2: Top row: strong learner scores, displayed within $[-1, 1]$ for different numbers of boosting rounds. Bottom row: left: loss function of Adaboost for increasing training rounds; middle: class posterior (ground-truth); right: adaboost-based approximatiot to posterior, at round 400.

Some clarifications

- The form of a decision stump is $f(x) = s(2[x_d > \theta] - 1)$, where d is the the dimension along which the decision is taken, $[\cdot]$ is 1 if $\cdot$ is true and 0 otherwise, θ is the threshold applied along dimension d and $s \in \{-1, 1\}$ is the polarity of the decision stump.
- When searching for the best decision stump you need in principle to consider all possible combinations of θ, d, s. As discussed in class, for a given dimension d the only values of θ that you actually need to work with are the values taken on by the training set features along that dimension.

2.1 SVM vs. Adaboost (.2/4.0)

Compare the performance of the classifier you obtained from the previous part of the exercise with that of Adaboost. Use the same training and test sets for both classifiers. As above, make sure you have reproducable results, by appropriately setting the `seed` variable in `rand`,randn+.

Report their average misclassification errors on the training set and on the test set. What do you observe?

2.2 Fast weak learner selection (.8/4.0)

When picking the best weak learner at each round a substantial speedup can be gained by a clever implementation of the threshold selection. Below some hints are provided, you can try to implement it quickly based on these hints.

Consider that we are working with feature k, and want to choose the best threshold θ for a decision stump of the form

$$h_\theta[x] = [x \geq \theta] \tag{1}$$

The naive implementation is to consider all possible thresholds $\theta = \{f_1^k, \ldots, f_N^k\}$, where N is the training sample size, and for each of those evaluate the associated weighted error:

$$C(\theta_j) = \sum_{i=1}^{N} [y_i = 1] w_i [f_i^k < \theta_j] + \sum_{i=1}^{N} [y_i = -1] w_i [f_i^k \geq \theta_j] \tag{2}$$

The complexity of this computation is $O(N^2)$, since for every of the N threshold values we need to perform N summations and multiplications.

Instead of this, a faster algorithm will first sort the N training samples according to their value of f^k; namely consider that we have a permutation π of the training set such that $f^k[\pi[i]] > f^k[\pi[j]]$ if $i > j$. For simplicity, we will denote the permuted features, weights, and labels by f', w', y' respectively. We then have the following recursion:

$$C(\theta'_1) = \sum_{i=1}^{N} [y_i = 1] w_i \tag{3}$$

$$C(\theta'_j) = \sum_{i=1}^{j-1} [y_i = 1] w_i + \sum_{i=j}^{N} [y_i = 1] w_i \tag{4}$$

$$= C(\theta'_{j-1}) - w_j [y = 1] + w_j [y = -1] \tag{5}$$

The first and second equations follow from the definition in Eq. **??** and the fact that f' have been sorted. The third equation can be verified by substitution. Sorting has $O(N \log N)$ complexity, while the recursion can be implemented in $O(N)$ time, so the overall complexity goes down from $O(N^2)$ to $O(N \log N)$.

You can use this idea to accelerate your Adaboost training code. In practice it means you will have to sort the k-th feature dimension, and use the sorting indices to permute accordingly the weights and labels. Subsequently you have to either implement the recursion above, or use the cumulative sum of an appropriately formed sequence.

Look into `sort.m, cumsum.m` for more details.