Assignment 2: Digit classification

February 18, 2012

1 Warming up

In this assignment we will do digit recognition, using the original MNIST dataset [1]. We will try to reproduce many of the results presented in [2]. The state-of-the-art error rate on this dataset is roughly 0.5%; the methods in this assignment will get you to 1.1% (modulo implementation details). In this assignment we will exclusively use linear kernels: the error rate can be reduced further by using non-linear kernels [2].

Before starting, read Appendix B. To avoid computational expense, we will only train on a subset of the data. (Once you have finished this assignment, if you have enough compute power at your disposal, you can try your code on the full dataset.) The file `train_small.mat` contains 7 different subsets, containing 100, 200, 500, 1000, 2000, 5000 and 10000 training points respectively. The full training set containing 60000 images is in `train.mat`. The test set is in `test.mat`.

Q1 As a warm-up, use the raw pixels as features to train a linear svm. Plot the error rate vs the number of training examples.

2 Spatial pyramids

The performance can be improved if we consider more sophisticated features rather than raw pixel values. [2] consider a feature where the image is overlaid with a grid in which each cell is of size $c \times c$ and the values inside each cell are added up to produce one feature per cell. [2] also suggest using multiple overlapping grids that are offset by $\frac{c}{2}$ instead of a single grid. We will construct two sets of grids, one with a cell size of 4 pixels and another with a cell size of 7 pixels. The final feature vector will be the raw pixel values, together with the features from all the cells concatenated together.

Q2 a) Explain briefly why adding these features should help over raw pixel values.

b) Implement these features and use them to train a linear svm. Plot the error rate vs the number of training examples. Do you get a significant boost over Q1?

3 Orientation histograms

To get to state-of-the-art performance however, we will need to go beyond pixel intensity. We will use a variant of PHOG(Pyramidal Histogram of Oriented Gradients) [3]. To construct the PHOG feature vector, we first compute the gradient magnitude and orientation at each pixel in the image. Then we construct "pyramid features" similar to Q2. However, instead of summing the pixel intensities in each cell we will construct a histogram of orientations in each cell. Orientation histograms typically have between 8 to 10 bins.

The final feature vector is the concatenations of all the histograms from all the cells. Typically the histograms are normalized (so that the bin counts sum to 1) before being concatenated.

Q3 a) Explain briefly why gradient orientations should help over pixel intensities : what is the gradient orientation trying to capture?

b) Implement these features and use them to train a linear SVM. For computing the gradient, use the tap filter:$[-1\ 0\ 1]$. Use 9 orientation bins. Use the same cell sizes (4 and 7) as in Q2. Plot the error rate vs the number of training examples. Do you get a significant boost over Q2?

c) Does the performance drop if you don't normalize the histograms before concatenating them? Why or why not? Plot error rate vs the number of training examples.

d) How does performance change if you replace the tap filter with a Gaussian derivative filter?

4 Visualizing errors

Q4 Visualize the images on which you go wrong. Do the errors your classifier makes seem reasonable to you? How many of these images do *you* as a human have a hard time recognizing?

5 Conclusion

The system you have built now should be almost on par with the state of the art as far as features are concerned. You can try training on the full training set to get an idea of how good you do. To get the full boost however, we will have to shift to a non-linear kernel, such as an intersection kernel [3]. We won't, however, be doing this in this assignment.

A Submission instructions

You should submit a zip file that contains the following three things:

1. Your code (We should be able to run it out of the box).
2. A short write-up containing the answers to the above.
3. A short README telling us how to use your code to reproduce your results.

To submit, email the zip file to `cs280.sp12.berkeley@gmail.com`. The subject of your mail should be `Assignment 2: partner_1_name partner_2_name`. To allow us to get back to you in case of any issues, in the text of the mail mention your names and email ids.

Please take care that your code doesn't take up inordinate amounts of time or memory. For reference, our solution using PHOG features trains on all 7 small training sets in under 15 minutes. This of course depends on the specifics of the machine, but if your code takes half a day to train, you know there is something wrong. Plus, we won't be able to grade your submission in that case.

B Running the code, and other such things

Before you begin, you need to make sure everything compiles well. You will need MATLAB, and either a Mac or Linux. If you are in Windows, you will need a version of `make`. Most files are just MATLAB, so no compilation is required. However, you need to compile the linear SVM package. cd into `liblinear-1.5-dense` and type `make`. Then cd into `liblinear-1.5-dense/matlab`. Edit the Makefile to point to your local copy of MATLAB, and then type `make`.

The full training dataset is in `train.mat` and the full test set is in `test.mat`. We have also provided 7 smaller training sets, with various number of training instances in `train_small.mat`.

There are also several utility functions. `train_linear_svm.m` will train a linear SVM out of features and labels while `predict_linear_svm.m` will use the trained model to predict labels on a set of test features. Both these functions take in the features as a matrix each column of which is a feature vector. `benchmark.m` will take in the predicted labels and true labels and return the error rate, and also the indices of the labels that are wrong. `montage_images.m` will produce a montage of a set of images: use this, for instance, to visualize the images on which you make errors.

References

[1] http://yann.lecun.com/exdb/mnist/

[2] S. Maji and J. Malik. Fast and Accurate Digit Classification. Technical Report, EECS Berkeley.

[3] S. Maji, A. Berg and J.Malik. Classification using Intersection Kernel Support Vector Machines is Efficient. In *CVPR*, 2008.