Computer Vision – 67542

MATLAB Assignment 2 – Basic Object Recognition

Preliminary Instructions

General

- Submission deadline: 8.5.2016.
- Late submission is allowed, but every day past the deadline will reduce 5 points from the assignment grade.
- The assignment is individual (i.e. you are not allowed to work in pairs or groups).
- Submit your solution through the course website. All of your files should be zipped in a folder named *matlab2_[ID].zip* ([ID] standing for your 9-digit ID).
- Your submission should include all .m files used during the assignment. Provide a brief description of each code file in a text file named *readme.txt*.
- Your code should be supplemented by a PDF file named *report.pdf*, which will include documentation of your results.
- Throughout the assignment you will carry out several SVM classifications. For this purpose you may use MATLAB's built-in *svmtrain* and *svmclassify* functions, or any other SVM implementation you find suitable (there are plenty of implementations available on the web). If you choose to use a non-MATLAB implementation, add to your submission all files required to run it.
- When working with images during the assignment, make sure intensity values are in the range $[0, 255]$. In particular, **do not normalize intensities to the range** $[0,1]$. If this requirement is not met, some of the parameter values you are instructed to use become irrelevant.
- The assignment consists of several mandatory parts and a bonus part. The basic 100 points of the grade are distributed evenly between the mandatory parts. The bonus part is worth 10 points.
- **Good luck!**

Dataset

The dataset you will be working with throughout this assignment consists of 40-by-100 grayscale images in PGM format, some presenting the side-view of a car ('positive' images) and some not ('negative' images). Below are samples of positive and negative images:

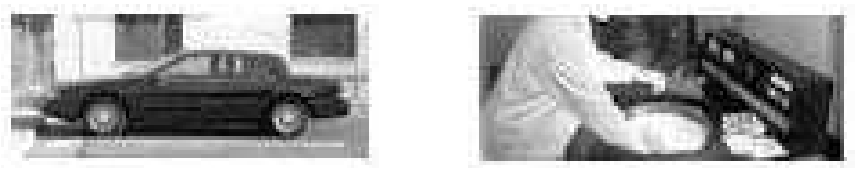

Figure 1 – *left*: positive image (side-view of a car), *right*: negative image (not a side-view of a car)

Your objective in the assignment will be to classify given images as positive or negative, with the highest possible accuracy. The approach which you will undertake for tackling this task is that of machine learning. You will train classifiers on a collection of training images, and then evaluate their performance on a separate collection of validation images. A third collection of test images is not given to you, and will be used to objectively evaluate the performance of the algorithms you submit.

The training and validation images of the assignment reside in the sub-folders *train* and *valid* (respectively) of the folder *dataset*. Positive training images (training images presenting a side-view of a car) are named *TrnP1.pgm*, *TrnP2.pgm* etc., whereas negative training images (training images not presenting a side-view of a car) are named *TrnN1.pgm*, *TrnN2.pgm* etc. Similarly, positive validation images are named *VldP1.pgm*, *VldP2.pgm* etc., while negative validation images are named *VldN1.pgm*, *VldN2.pgm* etc.

Part 1

In this part you will use linear SVM to classify images, where the image features (measurements) are the pixel intensity values. Namely, you will do the following:

- Map all images (training and validation) into vectors in $\mathbb{R}^d$, where the *i*'th entry holds the intensity of the *i*'th pixel.
- Train a linear SVM classifier on the training vectors (vectors corresponding to training images).
- Use the trained SVM classifier to classify the training and validation vectors, and evaluate the training error (# of misclassified training images / # of training images) and validation error (# of misclassified validation images / # of validation images). Document these error rates in your report.
- Add to your documentation some validation images which have been misclassified. Make sure to include at least one example of false-positive misclassification and one example of false-negative misclassification (if such exist).

Part 2

Implement the function:

w = RidgeTrain(X, y, lambda)

Inputs

- **X**: *m*-by-*d* real matrix where each row holds a training vector.
- **y**: *m*-element real column vector where entry *i* holds the label of row *i* in *X*.
- **lambda**: Positive scalar – regularization parameter λ.

Outputs

- **w**: *d*+1-element column vector representing the trained regressor (see description below).

Description

The function implements inhomogeneous Ridge-regression training. Namely, it returns the vector $w \in \mathbb{R}^{d+1}$ which minimizes the following term:

$$\frac{1}{2}\left\|\left[X,\mathbf{1}\right]w - y\right\|_2^2 + \lambda\left\|w\right\|_2^2$$

where $\mathbf{1}$ stands for a column vector of 1's.

Part 3

In this part you will use Ridge-regression to classify images based on pixel intensity values:

- Apply the mapping from Part 1 to all training and validation images.
- Form the matrix *X*, whose rows hold the vectors corresponding to the training images.
- Construct the column vector *y*, where each entry corresponds to a row of *X* and holds the respective label (+1 for positive, -1 for negative).
- For $\lambda = 10^4, 10^{4.1}, 10^{4.2}, \ldots, 10^{9.9}, 10^{10}$:
  - Call *RidgeTrain* (from Part 2) on *X*, *y* and λ to produce the vector *w*.
  - Apply the following rule to classify the training and validation vectors:

    $$\textit{estimated label of vector } x = \textit{sign}\left(\left[x^T, 1\right]w\right)$$

  - Calculate the training and validation errors obtained.
- Add to your documentation a plot of the training and validation errors vs. the regularization parameter λ (the latter being in logarithmic scale). Explain this plot in detail, and in particular how it can be used to select a value for λ. For the selected λ-value, add a few misclassified validation images (if possible, include at least one false-positive and one false-negative examples).

Part 4

Implement the function:

H = HOG(I, cell_x, cell_y, B)

Inputs

- **I**: *m*-by-*n* grayscale image.
- **cell_x**: Positive integer holding the width of a HOG cell.
- **cell_y**: Positive integer holding the height of a HOG cell.
- **B**: Positive integer holding the number of HOG bins.

Outputs

- **H**: (*floor*(*m*/*cell_y*)-1)-by-(*floor*(n/*cell_x*)-1) cell-array with entries corresponding to HOG blocks. Each entry holds a *B*-element row vector representing the oriented gradient histogram of the respective block.

Description

The function applies HOG measurements to a grayscale image. Namely, it does the following:

- Calculate the horizontal and vertical derivatives of the image by correlating it with the filters $h_x = [-1,0,1]$ and $h_y = [1,0,-1]^T$ respectively.
- Calculate the gradient magnitude and angle at each pixel of the image.
- For *i* = 1,…, (*floor*(*m*/*cell_y*)-1) and *j* = 1,…, (*floor*(*n*/*cell_x*)-1):
    - Collect the gradients in block (*i*,*j*) of the image, i.e. in cells (*i*,*j*), (*i*+1,*j*), (*i*,*j*+1) and (*i*+1,*j*+1) of the image, or equivalently in the pixels with horizontal coordinates between (*j*-1)**cell_x*+1 and (*j*+1)**cell_x* and vertical coordinates between (*i*-1)**cell_y*+1 and (*i*+1)**cell_y*.
    - For *b* = 1,…,*B*, sum up the magnitudes of the gradients with angles in the range $\left(-\pi + 2\pi(b-1)/B, -\pi + 2\pi b/B\right]$. Concatenate the results in a *B*-element row vector and place in entry (*i*,*j*) of *H*.
- Normalize all vectors in *H* to unit length (a vector of zeros will remain as such).

Note:

This version of HOG is slightly different from the one you have seen in class. The difference can be succinctly described as follows: in class we generated a block representation with concatenation of cell histograms, whereas here we use their sum instead.

Part 5

In this part you will classify images in the HOG domain using SVM. Namely, you will repeat Part 1 while replacing the pixel intensity vectors with HOG vectors. The HOG vector of an image is attained by calling *HOG* (from Part 4) with *cell_x* = 10, *cell_y* = 10 and *B* = 9, and concatenating the vectors of the resulting *H*.

Part 6

Repeat Part 3 while replacing the pixel intensity mapping with the HOG mapping specified in Part 5. Here, the regularization values to work with are $\lambda = 10^{-3}, 10^{-2.9}, 10^{-2.8}, \ldots, 10^{2.9}, 10^{3}$.

Part 7

In this part we will classify based on a small subset of HOG blocks. This is useful in terms of both run-time (less computations are required to classify an image) and sample complexity (the 'dimension' of the hypothesis class decreases).
For each HOG block, we will train a separate 'weak' classifier. We will then employ Adaboost to choose a subset of 'weak' classifiers.

In the root folder of the assignment you will find an .m file implementing the function *adaboost*. This function can be called in two modes:

- **Train mode**:

  [y_est, h] = adaboost('train', X, y, T)

- **Apply (classify) mode**:

  y_est = adaboost('apply', X, h)

Inputs\Outputs

- **X**: *m*-by-*d* real matrix where each row holds the feature measurements of an instance.
- **y**: *m*-element column vector where each entry holds the label of the corresponding row in *X* (+1 for positive, -1 for negative).
- **T**: Positive integer holding the number of Adaboost rounds to carry out (in training).
- **h**: A struct characterizing the trained Adaboost classifier.
- **y_est**: m-element vector holding the estimated labels of *X* according to the classifier characterized by *h*.

Description

In train mode, the function implements the Adaboost algorithm with the 'weak' classifiers being decision stumps on the features of *X*. The algorithm consists of choosing *T* 'weak' classifiers and corresponding weights, such that the weighted sum of the 'weak' classifiers produces a 'strong' classifier (characterized by *h*). It then applies this 'strong' classifier to the rows of *X* and accordingly assigns *y_est*.
In apply mode, the function simply applies the given 'strong' classifier (characterized by *h*) to the rows of *X*, producing the estimates *y_est*.

Do the following:

- Apply *HOG* (from Part 4) with *cell_x* = 10, *cell_y* = 10 and *B* = 9 to each of the training and validation images. This results in *k*-by-*l* cell-arrays holding *B*-element vectors corresponding to HOG blocks.
- For *i* = 1,2,…,*k* and *j* = 1,2,..,*l*:
    - Train an SVM classifier based on HOG block (*i*,*j*) of the **training** images.
    - Apply this classifier to the respective HOG block in the training and validation images. The results of this classification will serve as a feature fed into Adaboost.
- Construct training and validation vectors using the *k***l* features obtained above.
- For *T* = 1,…,20:
    - Train *adaboost* on the constructed **training** vectors, with *T* rounds.
    - Apply *adaboost* to the constructed training and validation vectors, and calculate the training and validation errors.
- Add to your documentation a plot of the training and validation errors vs. the number of Adaboost rounds *T*. Explain this plot in detail, and in particular how it can be used to select a value for *T*. For the selected *T*-value, add a few misclassified validation images (if possible, include at least one false-positive and one false-negative examples).

Part 8

Repeat Part 7, while using Ridge-regression instead of SVM for 'weak' classification. Particularly, for each HOG block, instead of training an SVM classifier, train a Ridge-regressor using *RidgeTrain* (from Part 2) with $\lambda = 10^{-2}$. Then, apply this Ridge-regressor to the respective HOG block in each of the training and validation images to obtain an input feature for Adaboost. Unlike in Part 3, **we do not discretize the regression here by taking sign**. In other words, applying a Ridge-regressor *w* to a vector *x* gives $\left[x^T, 1\right] w$ - a continuous value.

Bonus Part

Using any classification method of choice, try to reach the lowest validation error you can. Make sure not to use labels of the validation images during the training of your classifier. Include in your documentation:

- Description of the chosen classifier and its training.
- The validation error achieved and the corresponding training error.
- A few misclassified validation images (if possible, include at least one false-positive and one false-negative examples).