Course: DD2427 - Exercise Set 3

Due date: 06/05/2016
Cut-off date: 07/06/2016

In this exercise we investigate finding a hyper-plane that discriminates between images of pedestrians from non-pedestrian images. You will use stochastic gradient descent applied to cost-function consisting of an L_2 regularization term and the hinge loss:

$$\min_{\mathbf{w},b} \left(\frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\} \right) \tag{1}$$

You will extract a HOG descriptor from each image patch and use this as your image feature descriptor.

First download a subset of the *Daimler Pedestrian Path Prediction Benchmark Dataset* contained in the `.mat` files `pedestrian_train.mat`, `pedestrian_validation.mat`, and `pedestrian_test.mat` at the course website. These files contain images of both pedestrians and non-pedestrians. Each `mat` file contains a cell array containing images and vector with the label for each image.

Exercise 1: *The Pegasos Algorithm - training an SVM with SGD*

Your first task is to use the function `vl_hog` in the software package `vl_feat` to extract the HOG descriptor of each image patch in the training set. The function `vl_hog` takes as input two variables one is the image (where it is assumed the intensity values of the image are stored as `single`'s) and the other is a whole number defining the size of each cell. For this assignment set `cellSize = 4`. To convert the image `im` to type `single` you can use the command

```
    im = im2single(im)
```

You should create a matrix `Xtrain` of size `d`×`n` where each column of `Xtrain` is the HOG descriptor of a training image and it is assumed we have `n` training images.

Training of an SVM is usually more stable if the values in each dimension of the feature vector have the same scale and are centred. Therefore you should normalize `Xtrain` so that each row has mean 0 and standard deviation 1. Keep a record of the means and standard deviations that you compute because these will be needed at test time.

Now you have training data $\mathcal{D}$ stored in `Xtrain`. Next you will implement the algorithm called the *The Pegasos Algorithm*, which is an implementation of the SGD algorithm to minimizing equation (1). The steps of the *Pegasos* Algorithm are as follows:

- **Input**: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, λ, T
- **Initialize**: Set $\mathbf{w} = \mathbf{0}$ and $b = 0$
- for $t = 1, 2, \ldots, T$
  - Choose $i_t \in \{1, \ldots, n\}$ uniformly at random.
  - Set $\eta_t = \frac{1}{\lambda t}$.
  - If $y_{i_t}(\mathbf{w}_t^T \mathbf{x}_{i_t} + b_t) < 1$ then

    $$\begin{aligned} \mathbf{w}_{t+1} &= (1 - \eta_t\, \lambda)\mathbf{w}_t + \eta_t\, y_{i_t}\, \mathbf{x}_{i_t} \\ b_{t+1} &= b_t + \eta_t\, y_{i_t} \end{aligned}$$

    else

    $$\begin{aligned} \mathbf{w}_{t+1} &= (1 - \eta_t\, \lambda)\mathbf{w}_t \\ b_{t+1} &= b_t \end{aligned}$$

  - (Optional: Normalize the parameters that is

    $$a = \min\left\{1, \frac{1}{\|\mathbf{w}_{t+1}\|\sqrt{\lambda}}\right\} \tag{2}$$

    $$\mathbf{w}_{t+1} = a\, \mathbf{w}_{t+1} \tag{3}$$

    $$b_{t+1} = a\, b_{t+1} \tag{4}$$

)
- **Output**: $\mathbf{w}_{T+1}, b_{T+1}$

Write a function `TrainSVM` that takes the training data `Xtrain` and its labels `ys`, and the The prototype of the function will be

```
function [w, b] = TrainSVM(Xtrain, ys, lambda, T)
```

for this exercise you should set `lambda = .0001` and `T` to `40` $\times$ `n` (the number of training examples). Here are some hints for the implementation

- I used the `Matlab` function `inds = randperm(n)` to construct a random ordering of the indices from `1` to `n`. I called this function for each training epoch (one run through all the training data) and then I just ran through the elements of `inds`. Therefore I called the function with the number of epochs as opposed to the total number of update steps.

- Because computing the norm in equation (2) is expensive. I only did the optional normalization after each epoch.

To help you debug after 20 epochs of training I got a training loss of `197.3272` and a accuracy on the training set of `0.99643`. (Remember this is a stochastic algorithm so it is unlikely we will get exactly the same answers, but they should be in the same ballpark. There will be a much bigger variance on the training loss than on the training accuracy.)

Exercise 2: *Test your linear SVM on the test set*

Once you have trained your classifier you should apply it to the labelled data in the file `pedestrian_test.mat`. Remember the accuracy of your classifier is the number of correct predictions you make on the test examples. Also remember after you have extracted the HOG descriptor for each test sample you have to apply the same normalization to the data that you applied to the training data before you compute the classification function. That is you have to take away the means from the training data and divide by standard deviations from the training data. To help you debug the test accuracy of my classifier as $\approx$ `.89`. (Note if you use both the training and validation set for training you can bump up the test accuracy to $\approx$(.945 $\pm$.005).)

Exercise 3: *Optional: Use the validation set to search for a good value of `lambda`*

You should enumerate a possible set of values of `lambda` such as `.01, .001, .0001, .0001`. Train a classifier for each value of `lambda`. Compute the accuracy of each resulting classifier on the validation set. Then set `lambda` to the value which produced the best accuracy on the validation set. Finally re-train your SVM classifier using the training and the validation data and the best value of `lambda`. Then compute the accuracy of this final classifier on the test set.

Upload to course web:

- *your code and*
- *in the comment section state the accuracy of your classifier on the test set.*