MACHINE LEARNING AND PATTERN RECOGNITION
Assignment 1

Matriculation number - s1569105
Examination number - B076165

November 2015

1 The Next Pixel Prediction Task

1.1 Data preprocessing and visualization

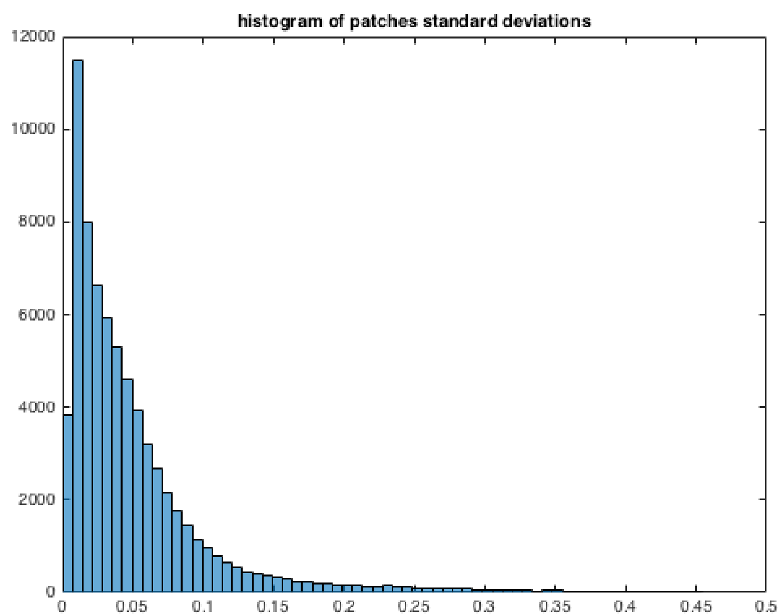


Figure 1: histogram of standard deviations in the xtr dataset after normalisation

(a) The maximum possible value of standard deviation is $\frac{max.-min.}{2}$, so in our case after normalisation it is $\frac{1-0}{2} = 0.5$. Our threshold to distinguish discrete values of pixels is $\frac{1}{64} = 0.5/32 \approx 0.0156$. If we use 32 bins on a range of possible values of standard deviation (between 0 and 0.5) then the width of one bin will be 0.0156 and standard deviations with values 0.0151 or 0.0021 will go to the same bin. But we usually would associate (after rounding using threshold) standard deviation 0.0151 with the discrete (original) pixel value of 1 and 0.0021 with 0 because

$round(0.0151/0.0156) = round(0.968) = 1$ and $round(0.0021/0.0156) = round(0.135) = 0$. Therefore, we must choose minimum 64 bins in order to distinguish such cases because we will have bins width $\frac{0.5}{64} \approx \frac{0.0156}{2} = 0.0078$ and each bin will correspond to the specific discrete (original) pixel value.
From the 1 we can see that after the peak on the second bin the number of patches declines exponentially as standard deviation increases . We can conclude that most of patches have standard deviation within 0 and 0.05 range, and 0.05 is quite small standard deviation, therefore, most of the patches are flat ones.

(b) I would choose mean of the all the pixels (1032) above and to the left of target pixel as a simplest predictor of the target pixel value for flat patches. Given definition of flat patches the value of the pixel in the flat patch will be something like this $f(x) = const_{flat\ patch} + o(0, \sigma_{flat\ patch})$ where o is random and small in comparison to $const_{flat\ patch}$, and it has 0 mean and as $\sigma_{flat\ patch}$ its standard deviation which follows $\sigma_{flat\, patch} \leq \sigma_{flat\, pach\, max}$. In general, I would prefer median because it is more robust to outliers if our dataset is noisy but in our case pixels can take only discrete values and I will show that mean suits us.
Consider extreme case where after normalisation (all pixel values between 0 and 1) in our flat patch most pixels are zeroes and small portion of pixels are ones (correspond to 63 intensity of original pixel). Let $N - m$ be number of zeros and let m be number of ones and I denote μ as mean.

$$m < N - m$$

$$\mu = \frac{(N-m)*0 + m*1}{N} = \frac{m}{N}$$

$$\sigma^2 = (N-m)(0 - \frac{m}{N})^2 + m(1 - \frac{m}{N})^2$$

$$= \frac{(N-m)m^2}{N^3} + \frac{m(N-m)^2}{N^3}$$

$$N^3\sigma^2 = Nm^2 - m^3 + mN^2 - 2m^2N + m^3$$

$$= mN^2 - m^2N$$

$$m^2 - mN + N^2\sigma^2 = 0$$

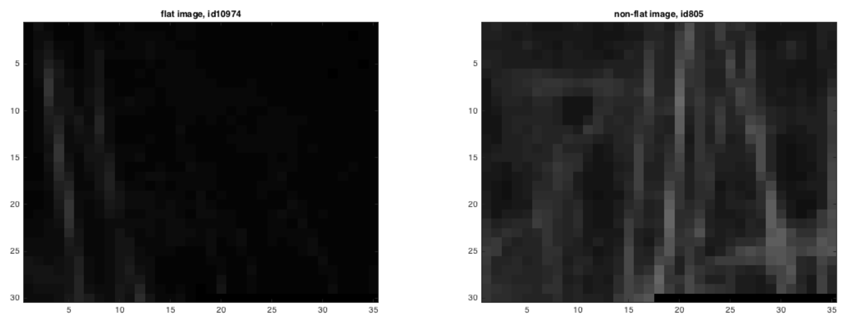
$$m = \frac{N}{2}(1 - \sqrt{1 - 4\sigma^2}) \quad \text{(minus because our case is } m < N - m \text{)}$$

putting $\sigma_{flat\, pach\, max} = \frac{4}{63} \approx 0.0635$ instead of σ and using $N = 1032$ we get

$$m = \frac{1032}{2}(1 - \sqrt{1 - 4 * 0.0635^2}) \approx 4.178$$

rounding m to the closest integer we receive $m = 4$.
Thus, in most extreme case of flat patch we can have 4 ones (correspond

Figure 2: patch images



(a) flat image, id 10974

(b) non-flat image, id 805

to original 63 pixel intensity) and 1028 zeros, so it is natural that we want to predict zero as discrete value of our target pixel. The mean gives us $\mu = \frac{1028*0+1*4}{1032} \approx 0.0038$. Dividing range between 0 and 1 by 64 we get 0.0156 as our threshold to distinguish discrete pixel values. $round(0.0038/0.0156) = round(0.244) = 0$ so our mean value will correspond to 0 as the discrete value of our target pixel and that is what we wanted.

(c) The code snippet for showing patch images on figure 2:

```
%load imgregdata.mat % I do it via terminal
%=============== part a ===============
xx = xtr ./ 63;
xx_std = std(xx,0,2);

%plot histogram
figure;
h = histogram(xx_std,64);
title('histogram of patches standard deviations');

%=============== part c ===============
%get indixes for flat and non flat patches respectively
xx_f_ids  = bsxfun(@le, xx_std, ones(size(xx_std)) .* 4/63);
xx_nf_ids = bsxfun(@gt, xx_std, ones(size(xx_std)) .* 4/63);

%slicing
xx_f  = xx(xx_f_ids, :);
xx_nf = xx(xx_nf_ids,:);

%pick one random example of flat patch and non-flat patch
get_rnd_row = @(X) randi(size(X, 1), 1);

%flat
rnd_flat_id = get_rnd_row(xx_f);
display(rnd_flat_id, 'random index of flat patch');
```

```
flat_patch = xx_f(rnd_flat_id, :);

%non-flat
rnd_non_flat_id = get_rnd_row(xx_nf);
display(rnd_non_flat_id, 'random index of non-flat patch');
non_flat_patch = xx_nf(rnd_non_flat_id, :);

%expanding patches to the full size
flat_patch(1050) = 0;
non_flat_patch(1050) = 0;

%creating images
flat_image = reshape(flat_patch, [35, 30]);
non_flat_image = reshape(non_flat_patch, [35, 30]);

%show images
%inverting them to ensure right position
%last index of the patch vector patch_vector(1050) == patch_image(30, 35)

%flat
figure;
imagesc(flat_image', [0, 1]);
title(strcat('flat image, id ', num2str(rnd_flat_id)));
colormap gray;

%non-flat
figure;
imagesc(non_flat_image', [0, 1]);
title(strcat('non-flat image, id ', num2str(rnd_non_flat_id)));
colormap gray;
```

1.2 Linear regression with adjacent pixels

(a) I used 5000 training points from xtr_nf and ytr_nf to plot figure 3. From it we can see that x(j, end), x(j, end - 34), y(j) are strongly positively correlated. However, there is some relatively small number of deviations from this trend. It seems that these deviations are normally distributed so linear regression should be reasonable model to describe such data.

(b) Derivation of this solution can be taken from MLPR lecture 7 slides 8-11 here. The solution for weights from there is:

$$\hat{\boldsymbol{w}} = (\Phi^T \Phi)^{-1} \Phi^T \boldsymbol{y}$$

In our notation matrix Φ will become:

$$\Phi = X = \begin{pmatrix} 1, x(1, end), x(1, end - 34) \\ 1, x(2, end), x(2, end - 34) \\ \dots \\ 1, x(N, end), x(N, end - 34) \end{pmatrix}$$

$$\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y}$$

where N is a number of training data points and x is our dataset (it will be xtr_nf in the next task)

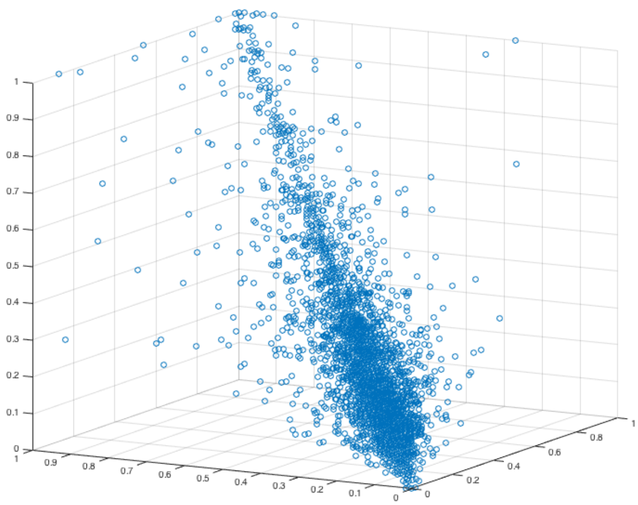


Figure 3: scatter plot of neighbour's pixels: x(j, end), x(j, end - 34), y(j)

(c) code snippet which I will be using to get linear regression predictor:

```
function [w, predictor] = lr_predictor(x_tr, y_tr)
    %assume y is column vector

    %adding bias term
    function Phi = calc_Phi(x)
        N = size(x, 1);
        Phi = [ones(N, 1), x];
    end

    %computing weights
    w = pinv(calc_Phi(x_tr)) * y_tr;

    function y_predicted = predict(x)
        Phi = calc_Phi(x);
        y_predicted = Phi * w;
    end

    predictor = @predict;
end
```

code snippet to compute root mean square error (RMSE):

```
function error = rmse(t, y)
    %assume y and are column vectors
    N = size(t, 1);
    diff = t - y;
```

```
    sqr_errors = diff' * diff;
    avg_sqr_errors = sqr_errors / N;
    error = avg_sqr_errors ^ 0.5;
end
```

code snippet for this particular task:

```
%load imgregdata.mat % I do it via terminal

%left and above neighbours
get_neighbours = @(x) [x(:, 1032), x(:, 1032 - 34)];

%prepare training set
X_train = get_neighbours(xtr_nf);
%train (it will add bias term automatically)
[w, predictor] = lr_predictor(X_train, ytr_nf);
display(w, 'weights for neighbours pixels features');

%predicted y on the training set
Yp_train = predictor(X_train);
%display training error
error_train = rmse(ytr_nf, Yp_train);
display(error_train, 'rmse on the training set');

%prepare testing set
X_test = get_neighbours(xte_nf);
%predicted y on the test set
Yp_test = predictor(X_test);
%display test error
error_test = rmse(yte_nf, Yp_test);
display(error_test, 'rmse on the test set');

%show surface
figure,
[dim1, dim2] = meshgrid(0:0.01:1, 0:0.01:1);
%swapped ones from original snippet, because w(1) corresponds to bias
%in my case
ysurf = [ones(numel(dim1),1), [dim1(:), dim2(:)]] * w;
surf(dim1, dim2, reshape(ysurf, size(dim1)));
hold on;
scatter3(xte_nf(:, 1032), xte_nf(:, 1032 - 34), yte_nf, 'red');
```

after training the weights are:

bias	left pixel	above pixel
0.0026	0.4606	0.5241

the RMSE for test and training set:s

	Training set	Test set
RMSE	0.0506	0.0503

This is not a typo, surprisingly my performance is a bit better on the test set. That is why we can conclude that linear regression is not overfitting

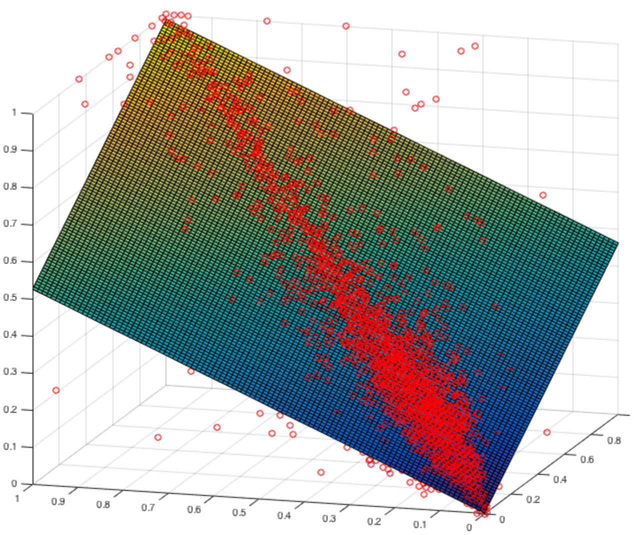


Figure 4: plot of the linear regression function after training along with test data points

the data in this problem. It can be seen from figure 4 that indeed there is strong positive correlation between closest neighbours of the target value pixel.