FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Machnine Learning (PDEEC0049 : 15-782PP)
Homework 3

António Damião das Neves Rodrigues (200400437 : 700098386)

November 20, 2013

1 PROBLEM 1

We are given a window or kernel function to proceed with a non-parametric estimation of $P(\mathrm{x}|C_k)$ using the Parzen Window approach. With this, and after calculating the priors from the training dataset, we would be able to classify the 3 test points via a Bayesian classifier.

$$P(\mathbf{x}|C_k) = \frac{1}{N_k}\sum_{i=1}^{N_k}\frac{1}{(2\pi h^2)^{3/2}}\exp\left\{\frac{-(\mathbf{x}-\mathbf{x}_{i,k})^t(\mathbf{x}-\mathbf{x}_{i,k})}{2h^2}\right\} \tag{1.1}$$

where $\mathbf{x}_{i,k}$ is the i-th training point labeled with class C_k and N is the total number of training points and N_k is the number of training points labeled with class C_k.The prior probabilities are given as

$$P(C_k) = \frac{N_k}{N}$$

Then, using Bayes' Theorem, the expression for the posterior probabilities becomes:

$$\begin{aligned} P(C_k|\mathbf{x}) &= \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})} \\ &= \frac{1}{NP(\mathbf{x})}\sum_{i=1}^{N_k}\frac{1}{(2\pi h^2)^{3/2}}\exp\left\{\frac{-(\mathbf{x}-\mathbf{x}_{i,k})^t(\mathbf{x}-\mathbf{x}_{i,k})}{2h^2}\right\} \end{aligned} \tag{1.2}$$

Taking into account that $N_k = 10$ for each class C_k, the code on Listing 1 is given for implementing the classifier.

Listing 1: MATLAB function for a classifier using Parzen window density estimation (a spherical Gaussian window function is used as kernel).

```
function prediction = parzenWindowClassifier (trainData, testData, hh)
% number of elements in the training set
N = size(trainData,1);
% dimension of the data
d = size(trainData,2);
% number of elements in the training set, such that C_k = y
n = 10; % it is 10 for all C_k

prediction = zeros(3,3);

for j = 1:3
    % P(C1|X)
    pXC1 = 0;
    for i = 1:10
        % P(C1|X)*P(X) = P(X|C1)*P(C1)
        pXC1 = pXC1 + (1/n)*(1/((2*pi*hh^2)^(d/2)))*exp(-((testData(j,:) ...
            - trainData(i,1:3))*(testData(j,:) - ...
            trainData(i,1:3))')/((2*hh)^2));
    end
    pXC2 = 0;
    for i = 11:20
        pXC2 = pXC2 + (1/n)*(1/((2*pi*hh^2)^(d/2)))*exp(-((testData(j,:) ...
            - trainData(i,1:3))*(testData(j,:) - ...
            trainData(i,1:3))')/((2*hh)^2));
    end
    pXC3 = 0;
    for i = 21:30
        pXC3 = pXC3 + ...
            (1/n)*(n/N)*(1/((2*pi*hh^2)^(d/2)))*exp(-((testData(j,:) - ...
            trainData(i,1:3))*(testData(j,:) - ...
            trainData(i,1:3))')/((2*hh)^2));
    end
    PX = (pXC1 + pXC2 + pXC3);

    % each line of the prediction array states the values of the
    % posteriors P(C1|x), P(C2|x), P(C3|x). Each column for each of the
    % 3 points to be classified.
    prediction(:,j) = [pXC1*(n/N)/PX; pXC2*(n/N)/PX; pXC3*(n/N)/PX];
end

return;
```

Setting $h = 1$, the classification results for the given points is shown in Table 1.1. The values at bold show the classification decisions.

	C_k	$P(C_1\|\mathbf{x})$	$P(C_2\|\mathbf{x})$	$P(C_3\|\mathbf{x})$
$(0.50, 1.00, 0.00)^t$	2	0.069	**0.202**	0.063
$(0.31, 1.51, -0.50)^t$	2	0.077	**0.209**	0.047
$(-0.30, 0.44, -0.10)^t$	2	0.072	**0.214**	0.047

Table 1.1: Classification of given points, with $h = 1$.

2 PROBLEM 2

Use the MATLAB code given as attachment (`problem2` folder) for implementation details.

2.1

$$y = \sigma(\mathbf{a}_1^t\mathbf{x}) + (\mathbf{a}_2^t\mathbf{x})^2 + 0.30z \tag{2.1}$$

where $\sigma(.)$ is the sigmoid function, z is a standard normal random variable (i.e. mean 0.0, variance 1.0), $\mathbf{x} = [x_1 x_2]^t$, with each x_j being independent standard normal, and $\mathbf{a}_1 = [33]^t$ and $\mathbf{a}_2 = [3-3]^t$.

Check the MATLAB code in files `problem2.m` and `generateTrainingData.m` for further details.

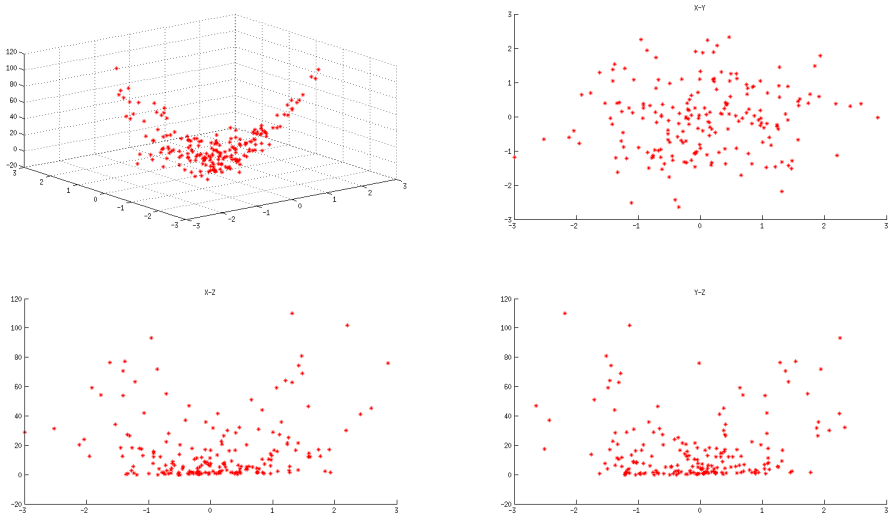

Figure 2.1: Graphical representation of 100 observations generated according to the model given in 2.1. 2D projections on the X-Y, X-Z and Y-Z planes are given for clarity.

2.2

Since this is a regression problem with an 1-dimensional output y, a single output neuron was chosen, with a linear activation function. This way the output value y is free to swing along the ranges shown in Figure 2.1 and is not restricted to a particular interval.

The sigmoid function (logistic function, defined as $\frac{1}{1+e^x}$) was chosen as the activation function for the hidden units. It seemed natural to include it in this particular case, since it is part of the model given in 2.1.

To include the weight decay, one followed expression 5.112 on [1]. This resulted in the addition of a parcel $\lambda\mathbf{w}$ when determining the derivatives of the error with respect to the first-layer and second layer weights. A learning factor of 0.001 has been fixed. The initial set of weights are randomly generated within the interval [-0.5, 0.5].

Check the MATLAB code (and comments) in the file `trainAndTestNeuralNet.m` for further details.

2.3

Figure 2.2 shows the error curves as a function of the number of training epochs. A sum-of-squares error function was used of the form

$$E = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2$$

where N is the number of training/test samples, y_n is the output value obtained by the model after a given epoch and t_n is the desired output.

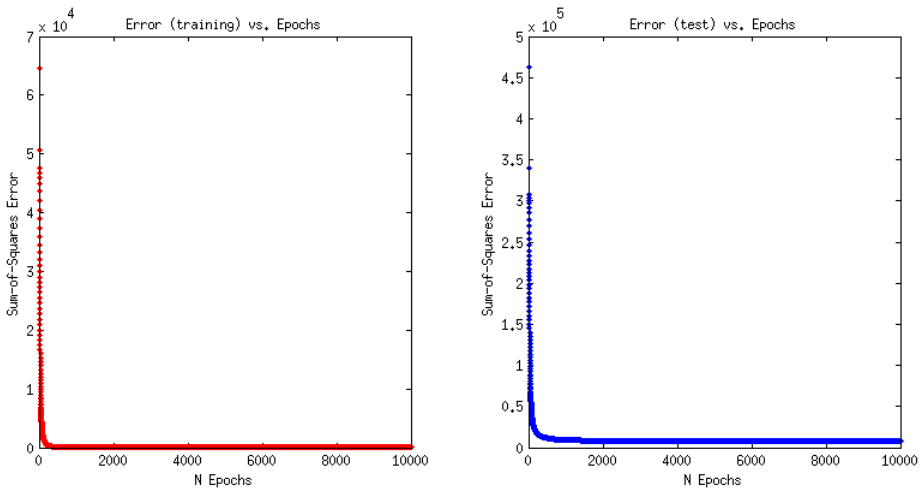

Figure 2.2: Training and test error curves as a function of the number of training epochs, for 10000 epochs and $\lambda = 0$.

As an over-fitting behavior I was expecting to find something similar to that of Figure 5.12 on [1], in which the test error increases after a given number of epochs is reached. Figure 2.2 shows the result after 10000 epochs, with a regularization factor $\lambda = 0$. By purposely not using a regularization factor, I have tried to force over-fitting. Such is not verified after 10000 epochs. In any case, the sum-of-squares error after 10000 epochs is still large for the test error case, which indicates there's a problem with the implementation of the neural network.

3 Problem 3

K	$d(P,\hat{P}(1))$	$(P,\hat{P}(3))$
10	∞	-0.326
20	∞	-0.333
30	∞	-0.336
40	∞	-0.337

Table 3.1: Results for the Kullback-Liebler divergence values for $K = 10, 20, 3040$ and posterior calculation methods given by equations (1) and (3) given in the exercise.

The results of the Kullback-Liebler divergence for equation (1) show that a pure KNN prediction is considered to be rather inaccurate, with a infinite divergence (this is caused by the values of 0 obtained for some $\hat{P}(C_k|x_n)$).

Check the MATLAB code (and comments) in the files `problem3.m`, `knnclassifier.m` and `getknn.m` for further details.

References

[1] C.M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.